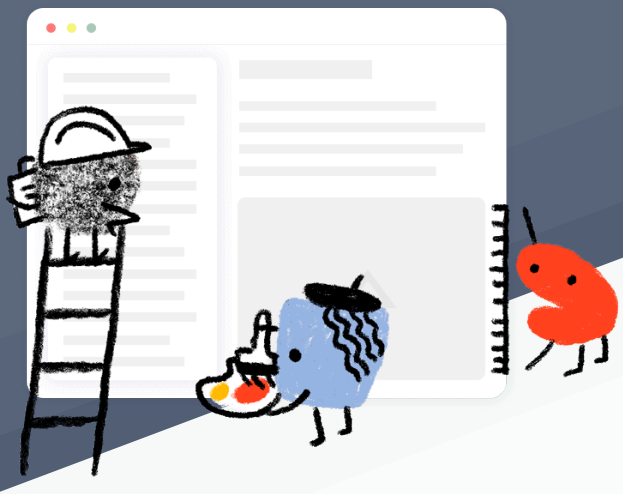
# 2022 Web Almanac

HTTP Archiveの年次報告書
ウェブの状態レポート



# 目次

## 部 I. ページコンテンツ



## 部 II. ユーザー体験



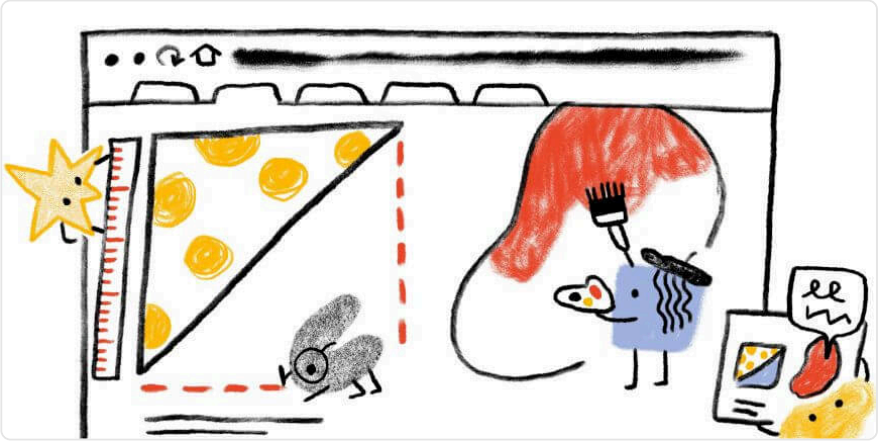
## 部 III. コンテンツ公開



## 部 IV. コンテンツ配信

# 付属資料

# 部 I 章 1
# CSS



***Rachel Andrew** によって書かれた。*
*Chris Lilley と Jens Oliver Meiert によってレビュ。*
*Rick Viscomi による分析 と 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

CSSは、Webページなどのレイアウトや書式設定に使われる言語です。構造を表すHTML、動作を表すJavaScriptに続く、Webの3大言語のひとつです。

ここ数年、CSSの新機能が続々と登場しています。その多くは、開発者がすでにJavaScriptやプリプロセッサで行っていたことにヒントを得たものですが、一方で、数年前には不可能だったことを可能にする方法を提供するものもあります。しかし、実際に開発者はその機能をWebページやアプリケーションで使っているのでしょうか？この疑問に対して、私たちはデータで答えようとしています。

この章ではTwitterで話題になったり、カンファレンスで紹介されたり、巧妙なデモで見つかったりした機能ではなく、開発者が実際に制作現場で使っているものをデータで調べています。どの新機能が採用され、どの古い技術が使われなくなり、どのレガシー技術がスタイルシートに、頑固に残っているかを見ることができます。

# 使用方法

毎年、CSSの規模が大きくなっていることがわかりますが、2022年も例外ではありませんでした。

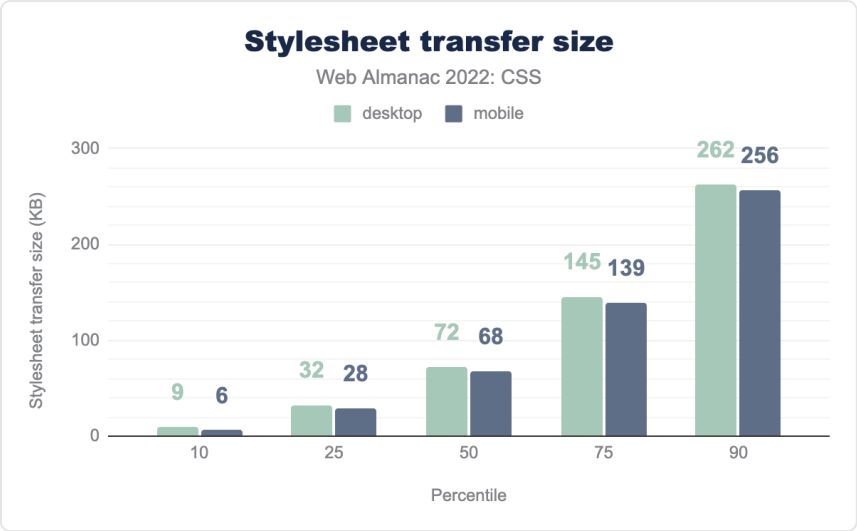



*図1.1. スタイルシートの転送サイズのページ別分布。*

25パーセンタイルが1ポイント下がった他は、各パーセンタイルでわずかながら増加した。90パーセンタイルでは、7%近く増加し、2020[1]と2021[2]の間で見られた増加と同様の増加でした。モバイル用のスタイルシートは、デスクトップ用に提供されるスタイルシートよりもわずかに小さいままです。

CSSのウェイトがもっとも大きいデスクトップページは62,631KBで、昨年よりわずかに小さくなっています。もっとも大きなモバイル用スタイルシートは、17,823KBから78,543KBに増加していますが、これはありがたいことに例外的なケースです。

1. *https://almanac.httparchive.org/ja/2020/css*
2. *https://almanac.httparchive.org/ja/2021/css*

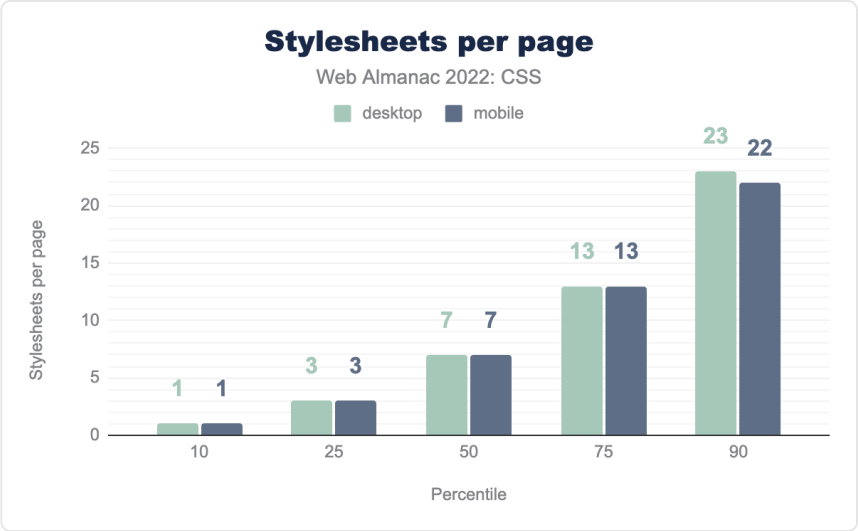



*図1.2. ページごとのスタイルシート数の分布。*

ページあたりのスタイルシート数は2021年とほぼ同じで、50パーセンタイル台でモバイルが1つ増えています。

昨年は、1つのページが読み込むスタイルシートの数が2,368となり、記録が更新されました。今年は、モバイルで1,387個のスタイルシートを読み込んでいるサイトが見つかりましたが、それでもかなりの量です。

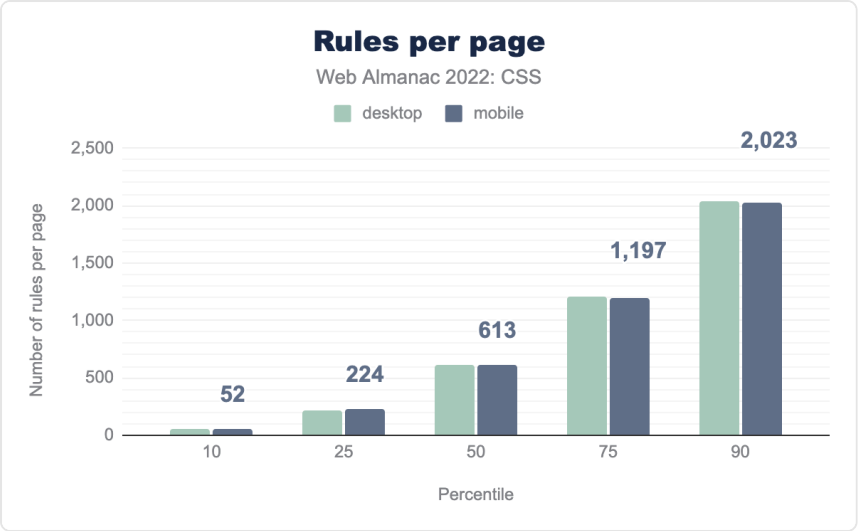



*図1.3. ページごとのスタイルルールの総数の分布。*

ページ内のスタイルルールの数を見ると、すべてのパーセンタイルで増加しており、低いパーセンタイルほどモバイル向け、高いパーセンタイルほどデスクトップ向けのルールが多くなっています。この増加幅は相当なものです。50パーセンタイルではデスクトップのルールが130個、90パーセンタイルでは202個増えています。

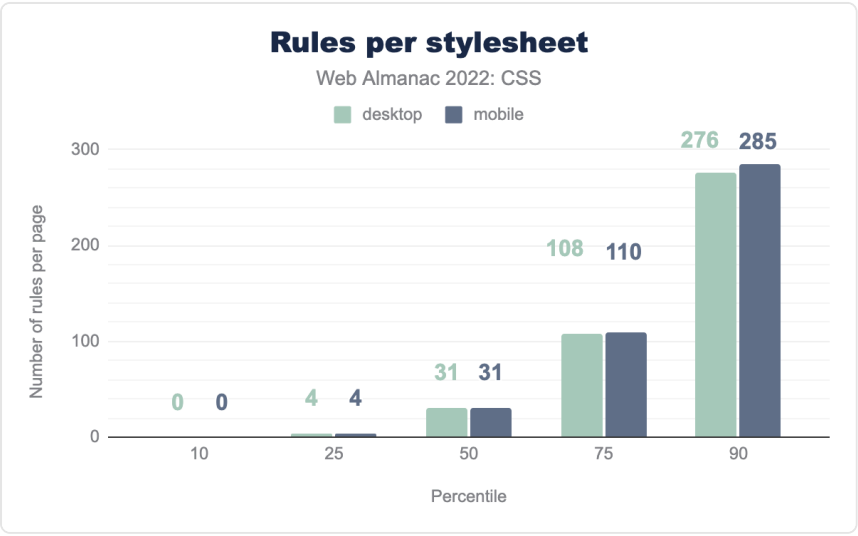



*図1.4. スタイルシートごとのルール数の分布。*

読み込まれたスタイルシートの総数から、通常、人々はCSSを複数のスタイルシートに分割していることがわかります。50パーセンタイルでは、スタイルシートごとに31ルールが読み込まれ、90パーセンタイルでは、デスクトップで276ルール、モバイルで285ルールにまで増加します。

## セレクターとカスケード

2022年には、`@layer`がすべてのエンジンに搭載され、カスケードに関しても変化がありました。この新しいアットルールは、セレクターをレイヤーにグループ化し、そのレイヤーの優先順位を管理できます。

この新しいカスケード管理方法が広く使われるようになるにはまだ少し早いですが、セレクターの使い方がどのように進化してきたかを見てみましょう。

## クラス名

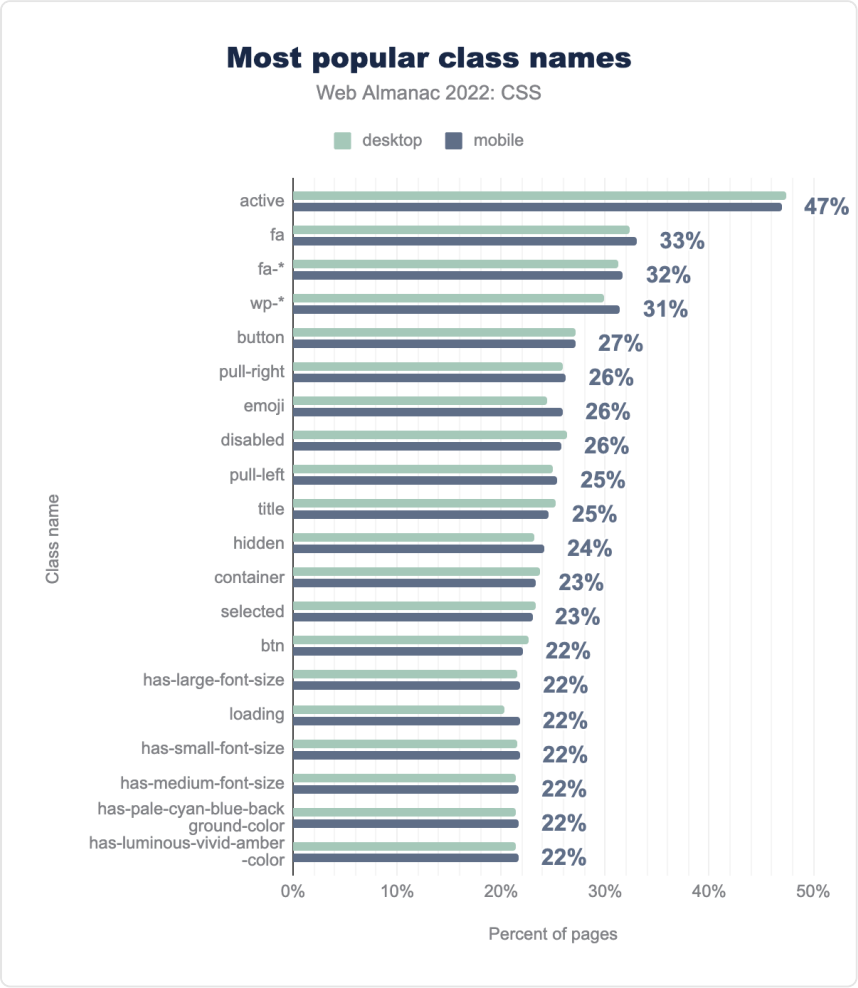



*図1.5. ページ数の割合でもっとも普及しているクラス名です。*

2020年、2021年と同様、ウェブ上でもっとも普及しているクラス名は`active`です。Font Awesomeのプレフィックスである`fa`、`fa-*`は依然として2位と3位を占めています。しかし、`wp-*`クラス名は順位を上げ、4位となりました。2021年には20%だったのが、現在では31%のページで表示されています。また、`has-large-font-size`といったクラス名も登場しており、これらは新しいWordPressブロックエディターで使用されています。

トップ20から`clearfix`が消え、今では10%のページで見られるだけです。これは、フロー

トベースのレイアウトがウェブから消えつつあることを明確に示しています。

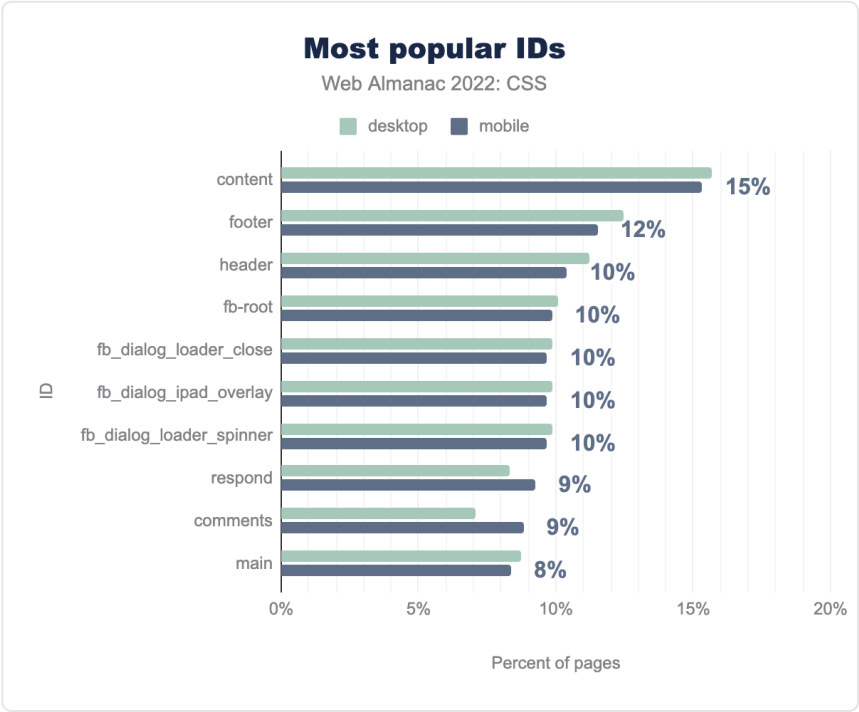



*図1.6. ページ数の割合でもっとも普及しているID名です。*

ID名は`content`がもっとも普及しており、`footer`、`header`がそれに続く。`fb_`で始まるIDは、Facebookウィジェットの使用を示しています。2021年には、GoogleのreCAPTCHAシステムの使用を示す`rc-`で始まるIDが7%のページで見られ、FacebookのID名に押されてトップ10から外れたものの、現在も同じ頻度で見られています。

## `!important`

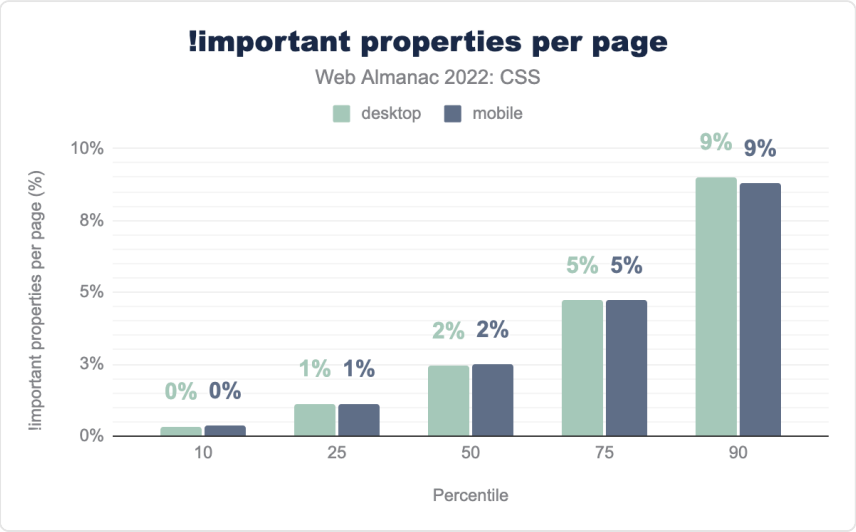



*図1.7. ページごとの `!important` プロパティの数の分布。*

今年は、上位2つのパーセンタイルで `!important` の使用率がわずかに増加しています。`@layer` の使用が定着するにつれて、通常は特異性の問題へ対処するために使用されるこのプロパティの使用にどのような影響を与えるか興味深いところです。

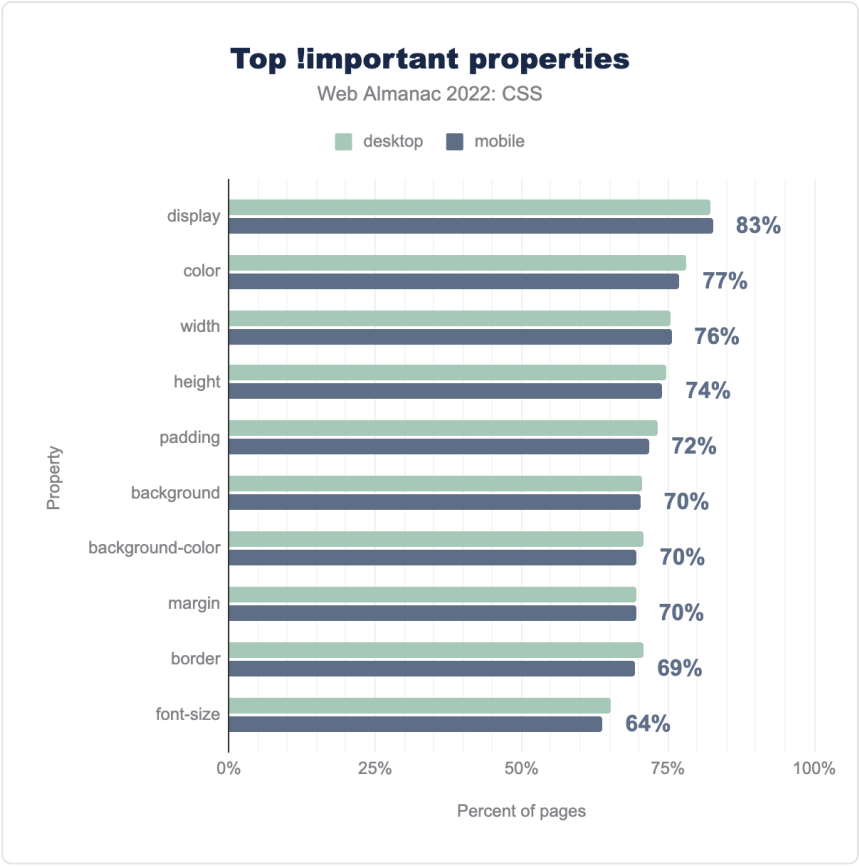



*図1.8. ページ数の割合で、`!important`が適用される上位のプロパティを示します。*

`!important`が適用されるものについては、上位のプロパティは変更されていません。しかし、`position`は上位10位から外れ、`font-size`に置き換わりました。

## Selector specificity

| パーセンタイル | デスクトップ | モバイル |
|---|---|---|
| 10 | 0,1,0 | 0,1,0 |
| 25 | 0,1,2 | 0,1,3 |
| 50 | 0,2,0 | 0,2,0 |
| 75 | 0,2,0 | 0,2,0 |
| 90 | 0,3,0 | 0,3,0 |

*図1.9. ページごとの特異性の中央値の分布。*

25パーセンタイル台のデスクトップを除き、特異度の中央値は昨年とまったく同じで、過去2年間一定です。これらの値は、BEM[3]などの方法論によって作られた平坦化された特異性を示しています。

3. https://en.bem.info/methodology/quick-start/

## 擬似クラスと擬似要素

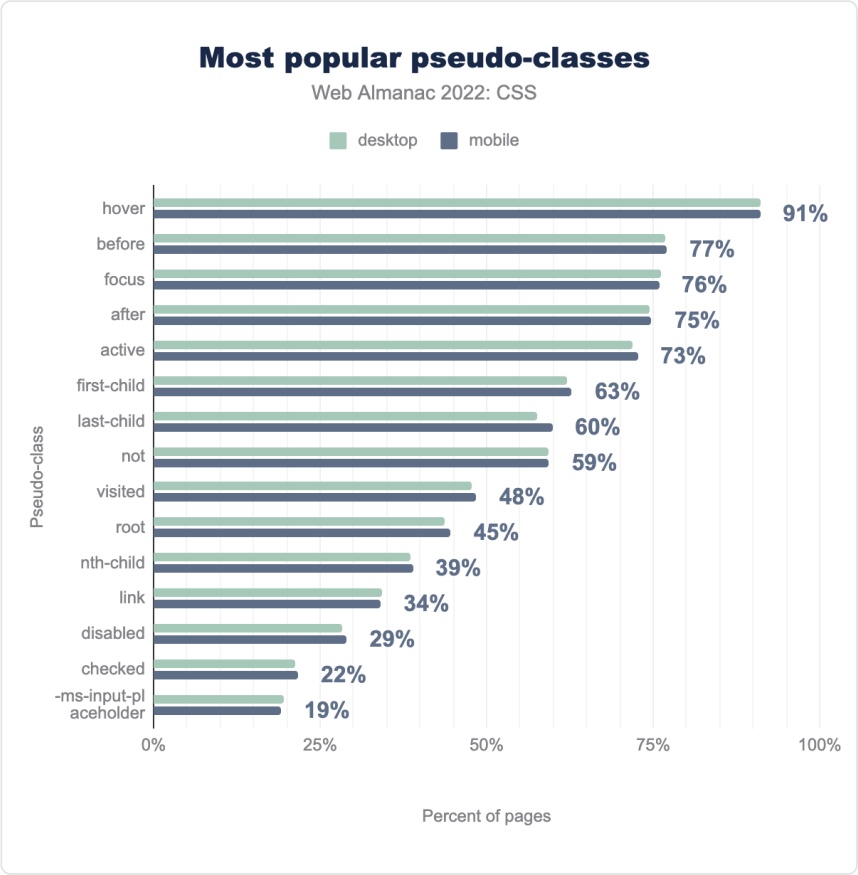



*図1.10. もっとも普及している擬似クラスのページ数割合。*

今回もユーザーアクション擬似クラス `:hover`、`:focus`、`:active` が上位3位を占めています。否定擬似クラス `:not()` も、カスタムプロパティを作成するために使用される `:root` と共に、人気が上昇し続けているようです。

昨年、ユーザーの期待によりマッチする方法でフォーカスのある要素をスタイルする方法である `:focus-visible` が、ページの1%未満にしか出現していないことが指摘されました。このプロパティは、2022年3月以降、3つの主要なエンジンすべてで利用可能となり、現在ではデスクトップページの10%、モバイルページの9%で見られるようになっています。

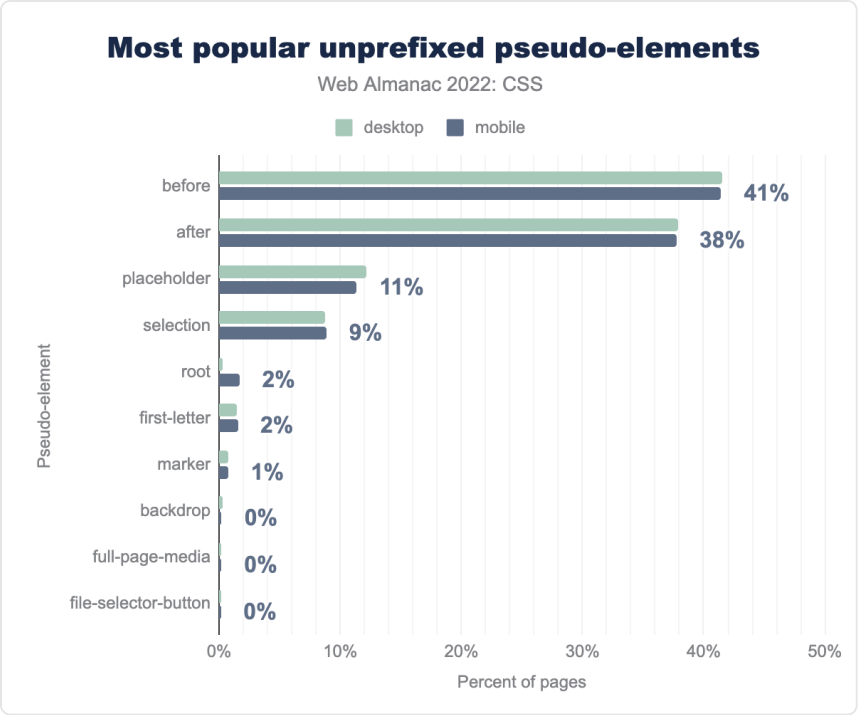



*図1.11. もっとも普及している擬似要素のページ数比率。*

接頭辞のついた、つまりブラウザ固有の擬似要素を除外しています。これらは通常、インターフェイスコンポーネントやブラウザクロームの一部を選択するために使用され、開発者が実際に使用している擬似要素に興味があります。

昨年から `::before` と `::after` の使用頻度が増えています。これらは、生成されたコンテンツをドキュメントへ挿入するために使用されます。`content` プロパティの使用状況を確認すると、空文字列を挿入するために使用されていることが多く、スタイリングのために使用されていることがわかります。生成されたコンテンツは、要素を追加することなくグリッド領域にスタイルを設定する方法の1つです。おそらく、これがこれらのプロパティの使用率の上昇に寄与しているのでしょう。

擬似要素である `::marker` の使用率が1%になり、リストマーカーを選択し、スタイルを設定する機能が徐々に利用され始めていることがわかります。

## 属性セレクター

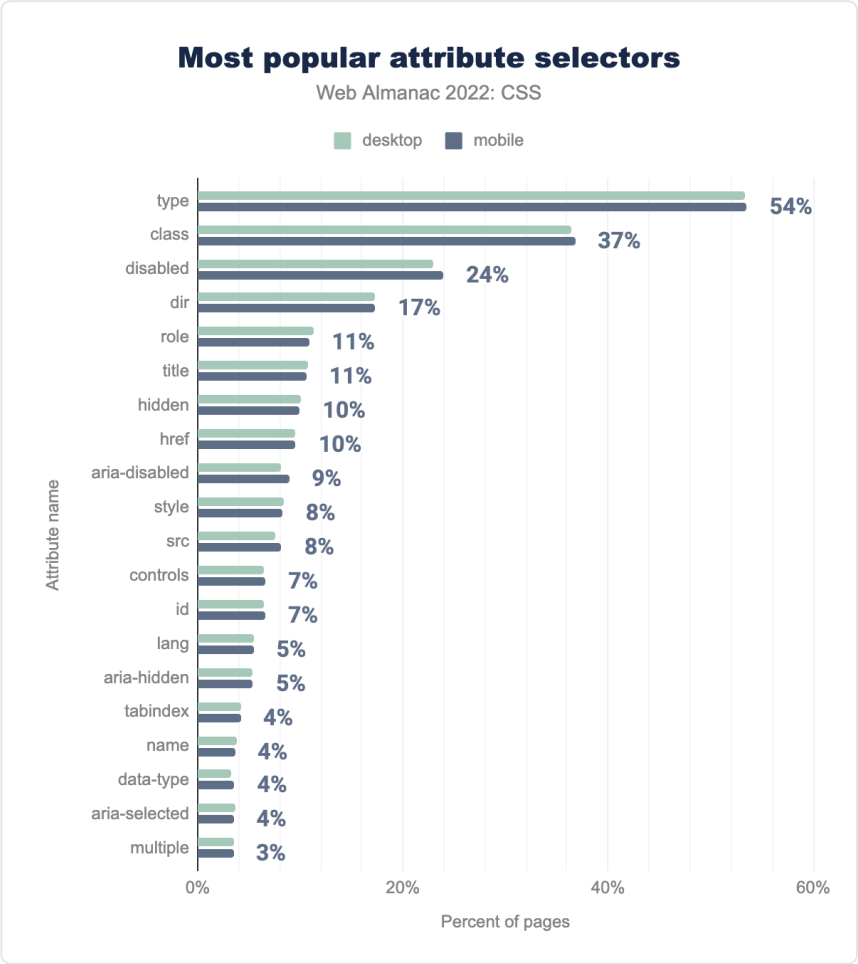



*図1.12. もっとも普及している属性セレクター（ページ数の割合）です。*

もっとも普及している属性セレクターは `type` で、54%のページで利用されています。次にもっとも普及している属性セレクターは `class` で37%、 `disabled` で25%、そして `dir` で17%のページで利用されています。

# 値と単位

CSSでは値や単位を指定する方法として、設定された長さや、グローバルキーワードに基づく計算など、複数の方法が用意されています。

## 長さ

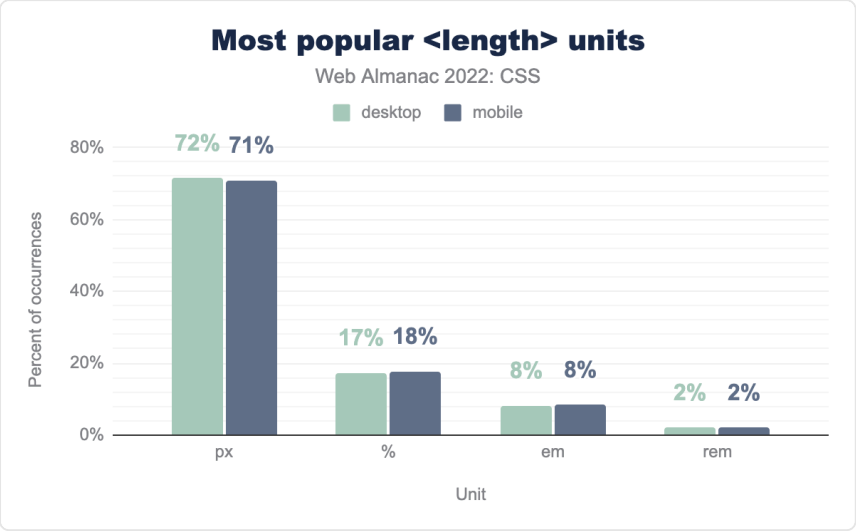



*図1.13. もっとも普及している `<length>` 単位（ページ数の割合）。*

ピクセルは、2021年と同じ71%で、もっとも普及している。用途の広がりもほぼ同じです。

| プロパティ | px | <number> | em | % | rem | pt |
|---|---|---|---|---|---|---|
| font-size | (▲2%) 71% | 2% | (▼1%) 15% | 5% | (▲1%) 6% | (▼1%) 2% |
| border-radius | (▼1%) 64% | (▼1%) 20% | 3.13% | (▲1%) 11% | (▲2%) 2% | 0% |
| line-height | (▼5%) 49% | (▲4%) 35% | 12.94% | (▼1%) 2% | (▲1%) 1% | 0% |
| border | (▼1%) 70% | 28% | 2% | 0% | 0% | 0% |
| text-indent | (▼5%) 26% | (▲13%) 65% | (▼4%) 5% | (▼3%) 5% | 0% | 0% |
| vertical-align | (▼26%) 3% | (▼9%) 3% | (▲39%) 94% | 0% | 0% | 0% |
| gap | (▲4%) 25% | (▼6%) 10% | (▲32%) 33% | 0% | (▼31%) 32% | 0% |
| margin-inline-start | (▼31%) 7% | (▲3%) 49% | (▲30%) 44% | 0% | 0% | 0% |
| grid-gap | (▲5%) 68% | (▼1%) 10% | (▼2%) 7% | 0% | (▼1%) 15% | 0% |
| margin-block-end | (▼1%) 3% | (▲54%) 85% | (▼53%) 12% | 0% | 0% | 0% |
| padding-inline-start | (▼4%) 29% | (▲11%) 16% | (▼10%) 53% | 0% | (▲3%) 3% | 0% |
| mask-position | (▲1%) 1% | (▲3%) 3% | (▼14%) 36% | (▲10%) 60% | 0% | 0% |

*図1.14. プロパティごとの分布。*

このグラフの上下の矢印は、2021年の実績[4]からの変化を示しています。昨年と同様に、ほとんどの場合、ピクセルの使用から他の長さ単位への移行が見られます。今回も `vertical-align` プロパティはピクセルと `<number>` の使用が大幅に減少し、 `em` の使用が大幅に増加しました。

4. https://almanac.httparchive.org/ja/2021/css#fig-15

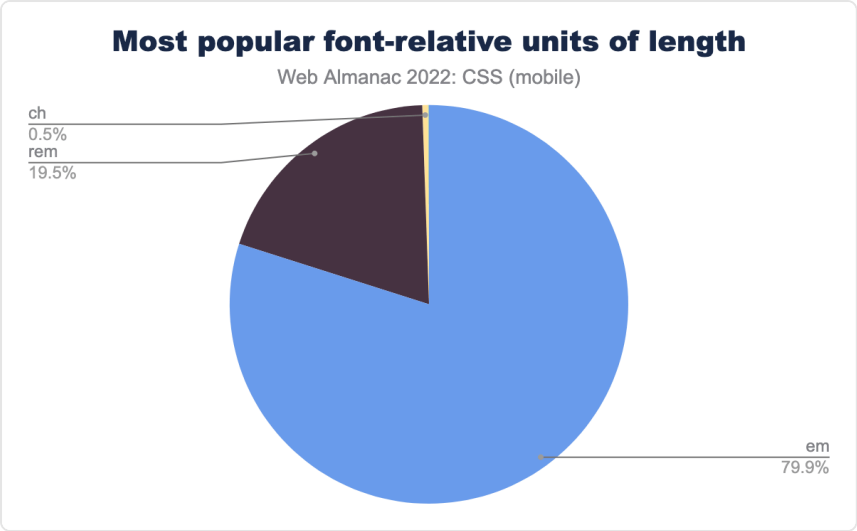



*図1.15. もっとも普及している相対的フォントの単位。*

フォントのサイズ調整方法としては、`em`がもっとも普及しているが、`rem`への移行が続いており、昨年より若干（2ポイント弱）増加している。

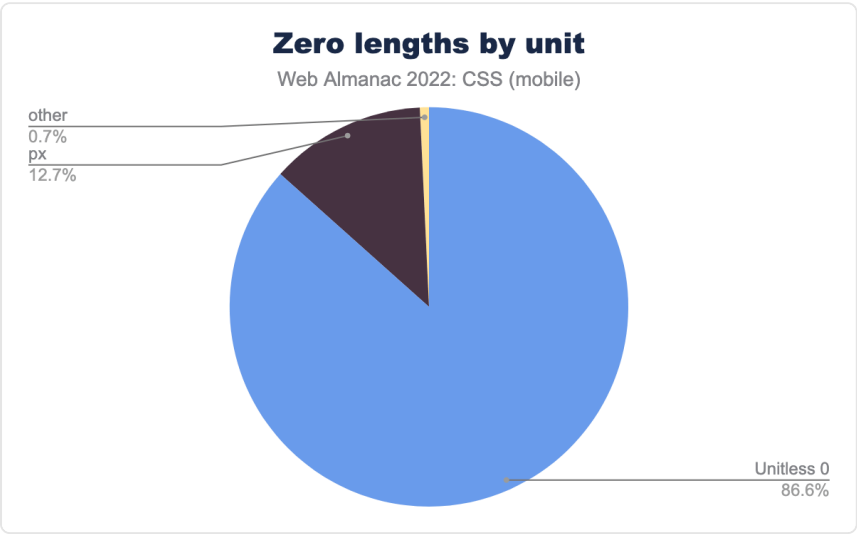



*図1.16. 長さ0の値で使用される単位（またはその欠如）。*

素の`<number>`単位が許されるプロパティはいくつかありますが(たとえば`line-height`)

、`<length>`値は、長さがゼロでも単位が不要という特殊なケースです。長さ0の値をすべて調べたところ、ほぼ87%が単位を省略しており、これは昨年より少し減少しています。単位を含む長さ0の値のほぼすべてがピクセル（0px）を使用していました。

## 計算方法

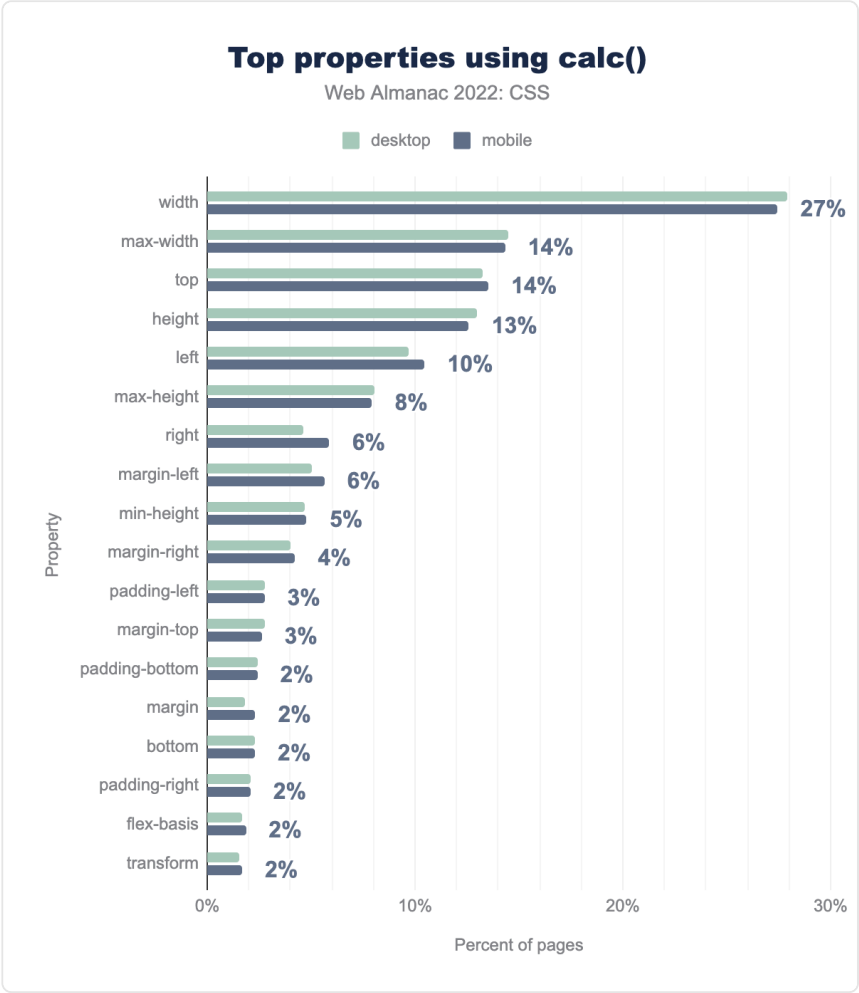



*図1.17. `calc()`関数を使ったもっとも普及しているプロパティです。*

例年通り、`calc()`の、もっとも普及している用途は、widthの値です。この使用は12%ポイント減少しましたが、`max-width`の使用は9ポイント増加しました。

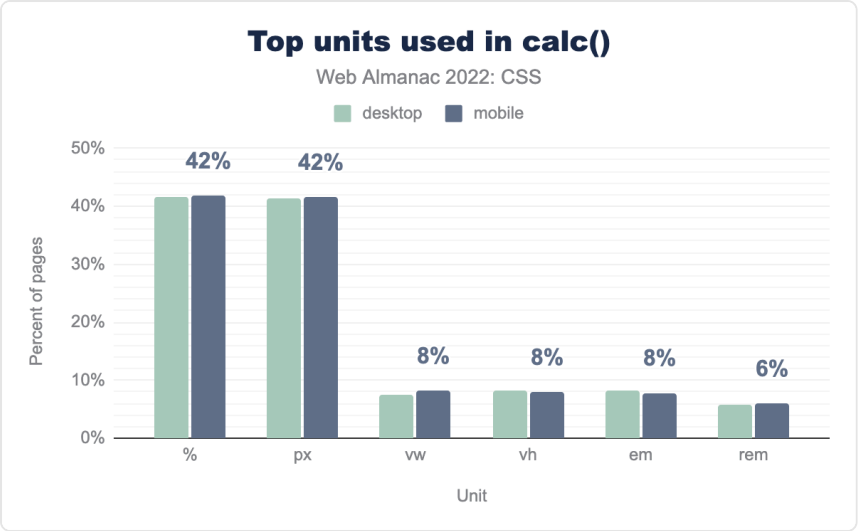



*図1.18. `calc()` 関数で使用されるもっとも普及している長さの単位です。*

計算にピクセルを使用しているサイトの割合は9ポイント減少し、現在では `%` の使用率と同じ42%になっています。ビューポートの単位である `vw` と `vh` は今年2%から8%に、 `em` は同じだけ増加し、 `rem` の使用率は3%から6%に倍増しています。

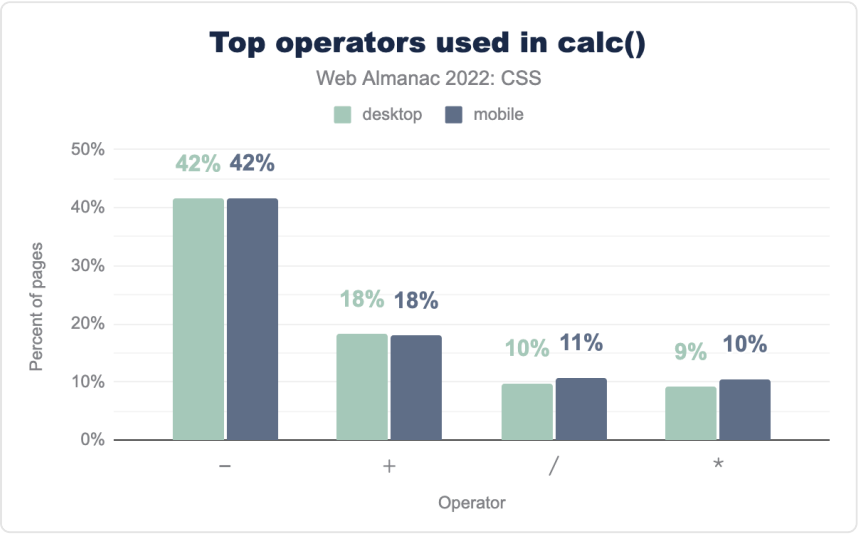



*図1.19. `calc()` 関数で使用されるもっとも普及している演算子です。*

計算演算子では「引き算」が依然としてダントツの人気ですが、「足し算」が変わらない以外は、上位4つの値とも2021年以降に低下しています。

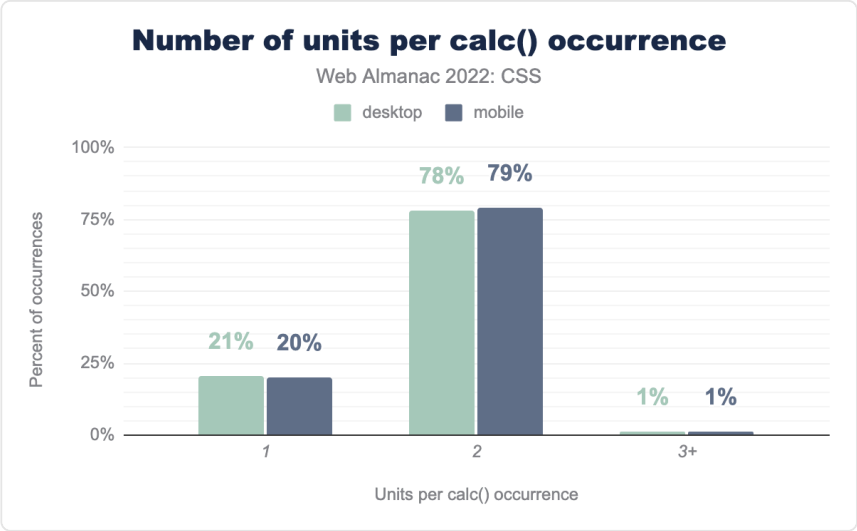



*図1.20. `calc()` の値で使用されるユニークな単位の数。*

昨年と同様、`calc()` の値はかなり単純なものになる傾向があります。たとえば、ピクセルなどの固定長をパーセンテージから引くという一般的な使用例など、大多数が2つの値を使用しています。1単位の値が少し増加し、2単位の値が少し減少した。

# グローバルキーワード

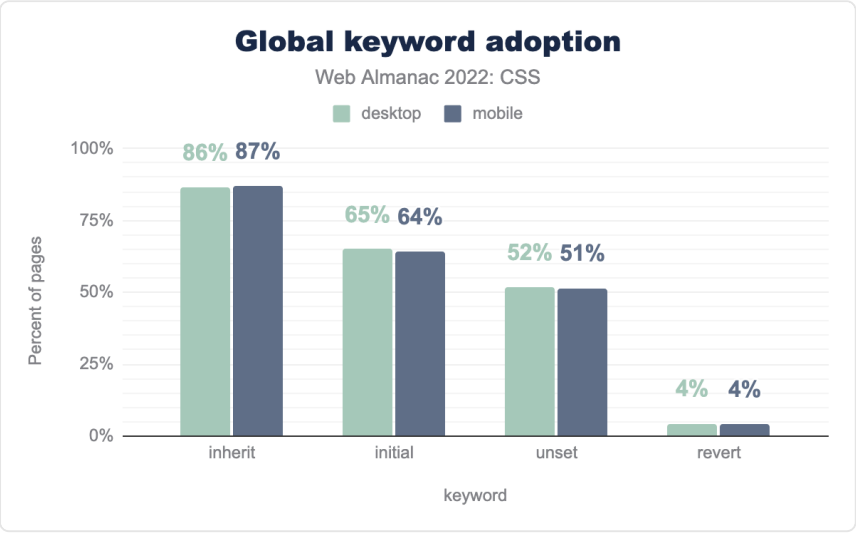



*図1.21. グローバルキーワードの値の使用状況。*

昨年はグローバルキーワードの使用率が大幅に上昇しましたが、2022年は`inherit`が同じ割合で使用されています。しかし、他の3つの数値は使用率が上昇しています。新しい値では、`revert`が1%から4%に増加しました。

# カスタムプロパティ

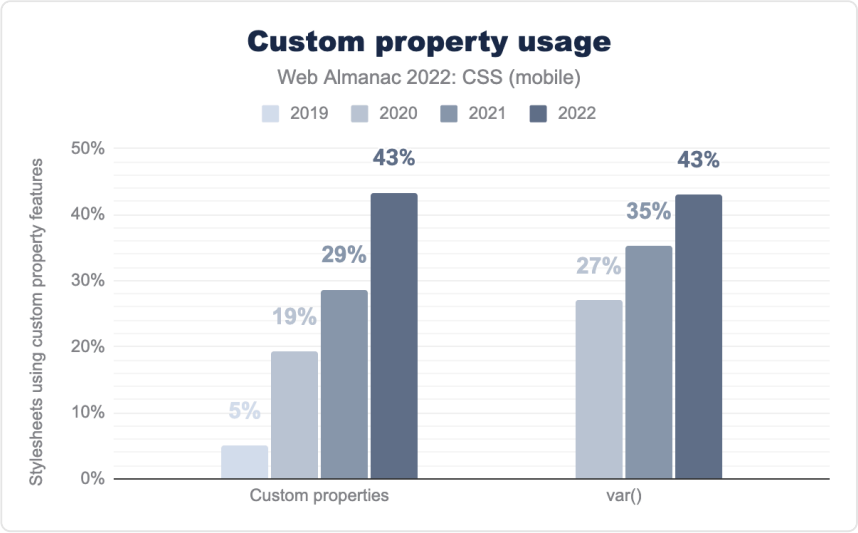



*図1.22. 過去4年間のカスタムプロパティの使用状況。*

カスタムプロパティ（CSS変数として知られることもある）の使用は大きく急増し、2021年から2022年にかけての成長も例外ではありません。デスクトップとモバイルの両方で、43%のページがカスタムプロパティを使用しており、少なくとも1つの `var()` 関数を持っています。

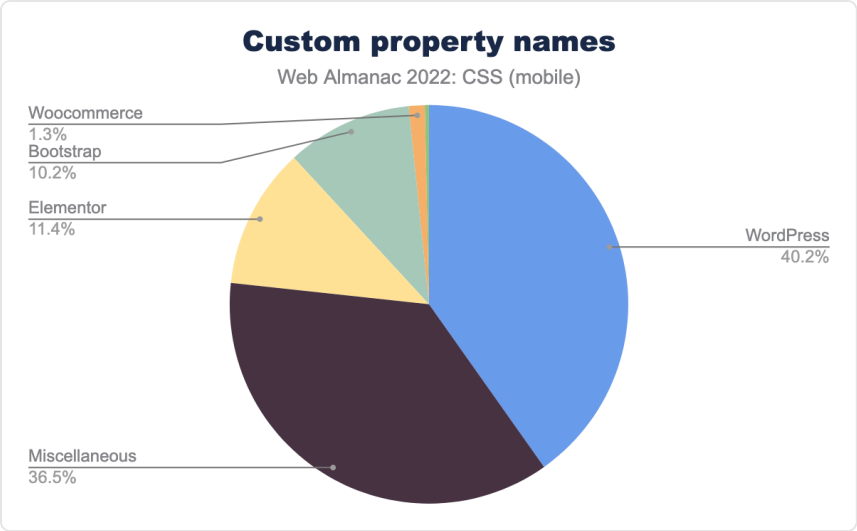



*図1.23. 一般的なカスタムプロパティ名のソース。*

昨年と同様に、WordPressは、もっとも一般的なカスタムプロパティ名のドライバーであり、これらは `-wp-*` というプレフィックスによって簡単に識別できます。これに続いて、`—white``—blue` などの色名が多く、特定の色合いを指定するために使用されています。

## 種類

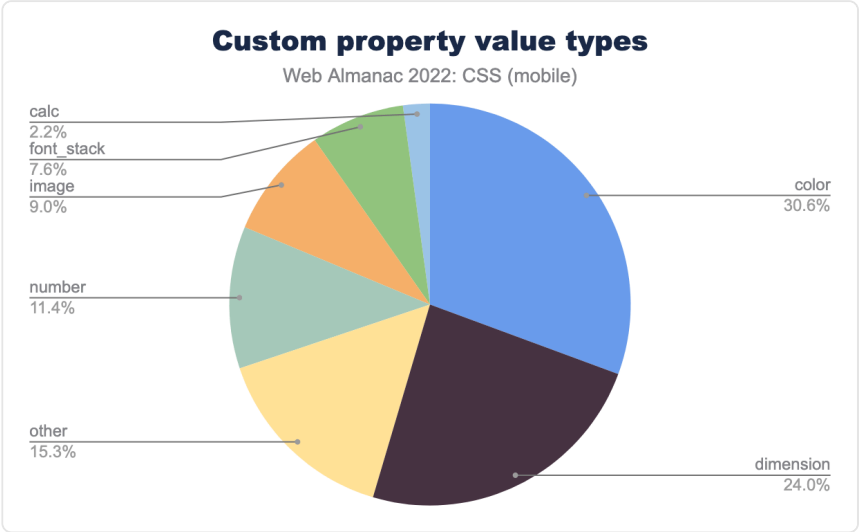



*図1.24. カスタムプロパティの値の種類を配布。*

カスタムプロパティの値には、タイプが含まれる。たとえば、`--red: #EF2143`は`--red`に色の値を割り当てていますが、`--multiplier: 2.5`は数値の値を割り当てています。タイプは昨年から少し変化しています。カスタムプロパティの用途としては、色を設定することがもっとも多く、カラータイプが見られるページも増えていることが分かっています。しかし、使用割合でいうと、40%から30%に低下しています。この分布に入るのが`calc()`であり、値型としてのimagesです。

## プロパティ

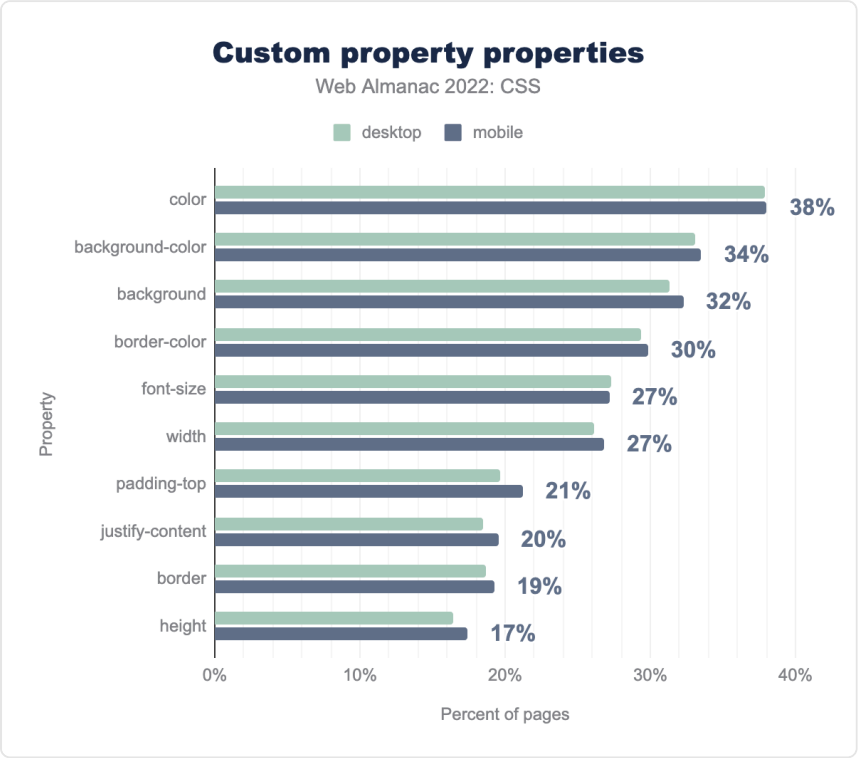



*図1.25. もっとも普及しているカスタムプロパティのプロパティをページ数の割合で表示します。*

これらのプロパティを含むページが増える一方で、カスタムプロパティを値として持つプロパティは昨年とほぼ同じ順位で推移しています。カスタムプロパティは `color` に使用されることが多いようです。当然のことながら、カラースキームの作成はこの機能の明らかな使用法です。しかし `var()` 関数を使って `font-size` を設定することは、リストの10位から5位になり、 `justify-content` のalignment値を設定することは、トップ10に移動しました。2021年にはモバイルページの5%、デスクトップページの4%がこのアライメント値を設定するためにカスタムプロパティを使用していましたが、これが20%に跳ね上がりました。データから、この増加の一部はWordPressの使用によるものであるように見えます。たとえば、5%のページが `-navigation-layout-justify` カスタムプロパティを使用しています。

## 機能

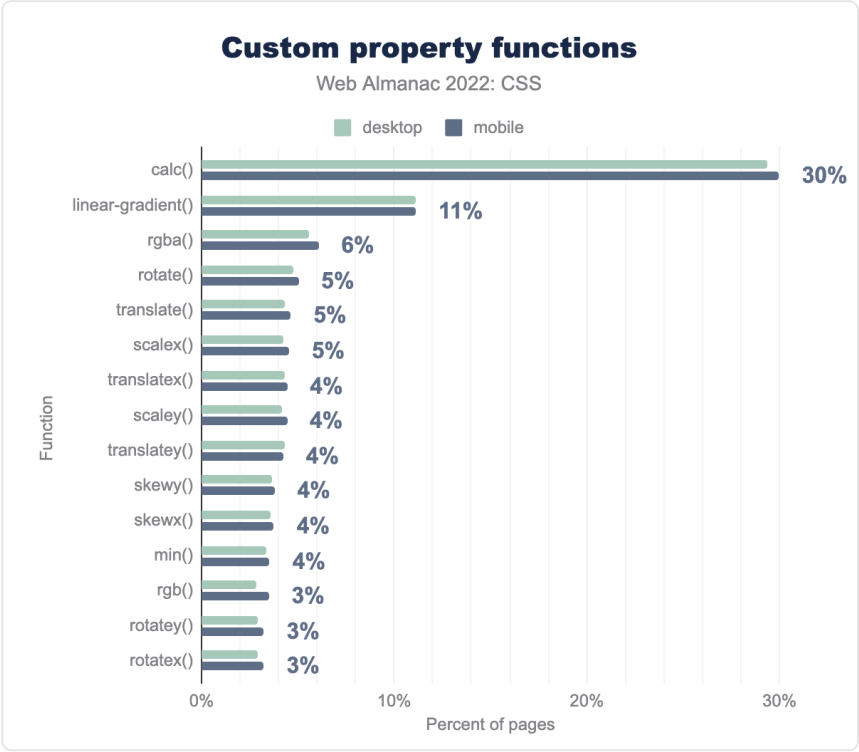



*図1.26. もっとも普及しているカスタムプロパティ機能をページ数の割合で示したもの。*

カスタムプロパティの値型として `calc()` が注目され始めていることを見てきましたが、このように使われる関数は圧倒的に多いようです。続いて、`linear-gradient()`、アルファチャンネル付きのRGBカラー値を設定するための `rgba()` 関数と続きます。 続いて、トランジションやアニメーションに使われるさまざまな関数があり、この分野でのカスタムプロパティの利用が進んでいることがわかります。

## 複雑さ

カスタムプロパティを他のカスタムプロパティの値に含めることは可能です。2020年版Web Almanacのこの例[5]を考えてみましょう。

5. *https://almanac.httparchive.org/ja/2020/css#複雑さ*

```
:root {
  --base-hue: 335; /* depth = 0 */
  --base-color: hsl(var(--base-hue) 90% 50%); /* depth = 1 */
  --background: linear-gradient(var(--base-color), black); /*
depth = 2 */
}
```

前の例のコメントで示したように、これらのサブリファレンスが連鎖すればするほど、カスタムプロパティの深さは大きくなります。

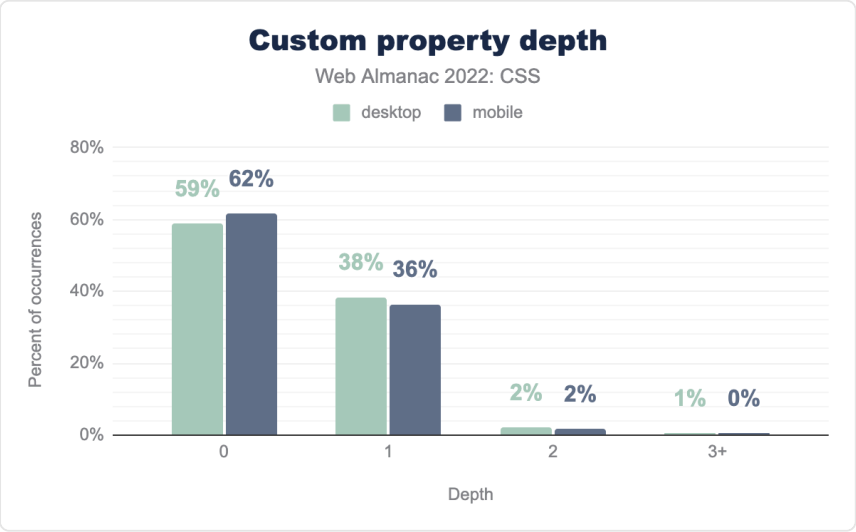



*図1.27. カスタムプロパティの深さの分布。*

2021年に見られるように、大半のカスタムプロパティは深さが0であり、他のカスタムプロパティの値をその値に含めないことを意味する。深度1のプロパティの数は少し増え、深度2の数は少し減りました。しかし、このデータからは、過去1年間にカスタムプロパティの使い方がそれほど複雑になったとは思えません。

# 色

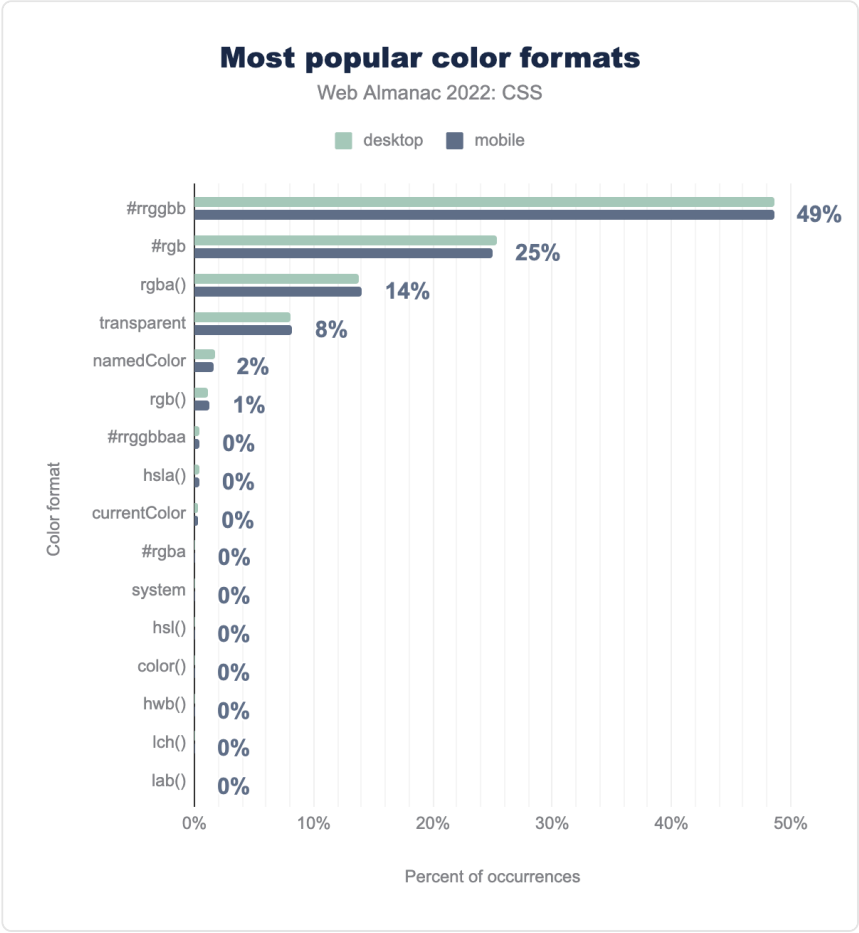



*図1.28. もっとも普及しているカラーフォーマットの出現率。*

伝統的な6桁の `#RRGGBB` 構文の使用は2021年以来変わっておらず、色の宣言の半分で使用されています。8桁の `#RRGGBBAA` 16進法が広く使われているにもかかわらず、`rgba()` 形式がアルファ成分を追加する方法としてもっとも広く使われているのは、おそらくこれがずっと以前にブラウザへ実装されたからでしょう。

他の値の使い方も同様で、ウェブコミュニティはまだ他のカラーフォーマットを利用し始めておらず、`hsl()` のような広くサポートされているものでさえも利用していません。

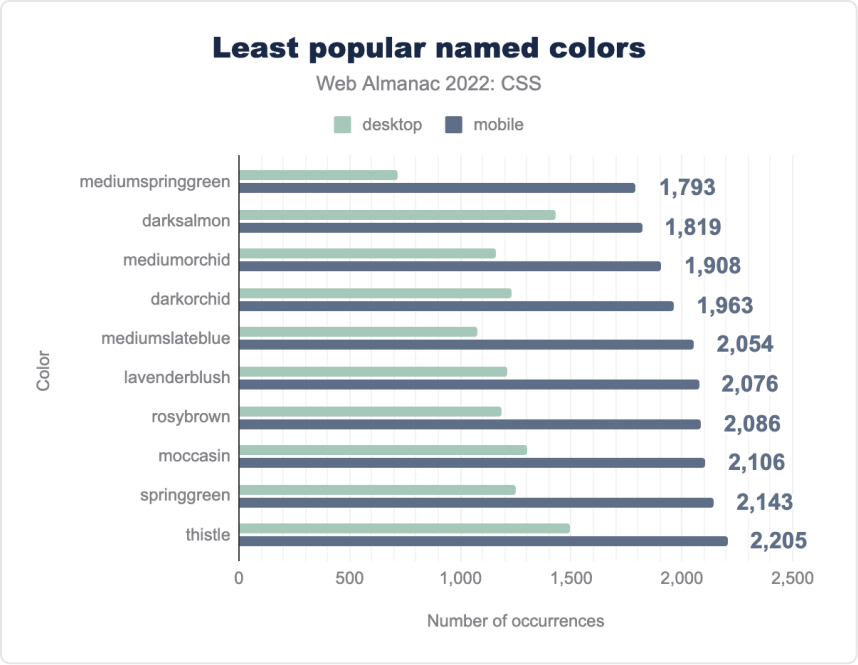



*図1.29. 登場回数がもっとも少ないネーミングカラー。*

8%のページがキーワード `transparent` を使用しており、もっとも普及している色の名前です。2%のページが他の色を使用しており、`white` がもっとも普及しており、`black` がそれに続いています。一方、`mediumspringgreen` はもっとも人気のない色として低迷しています。

## アルファのサポートと活用

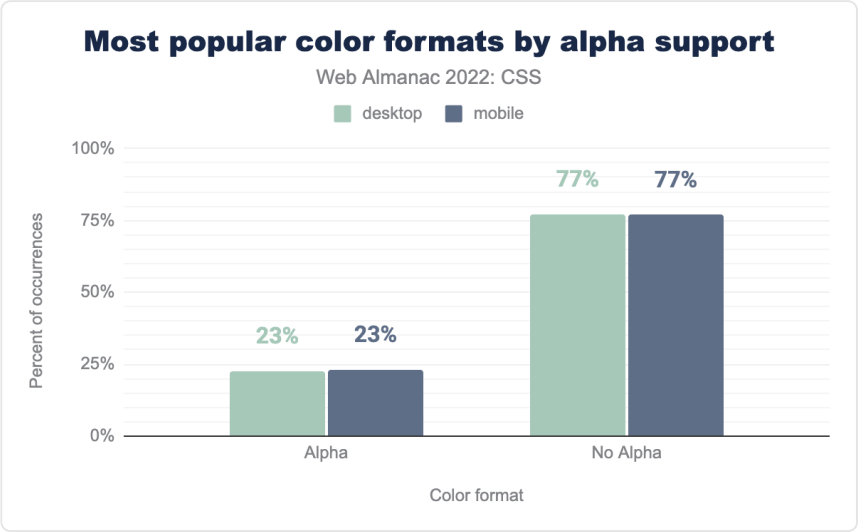



*図1.30. アルファ対応でもっとも普及しているカラーフォーマットです。*

`rgba()` 関数は3番目に普及しているカラーフォーマットで、`rgb()` 形式よりもかなり多く使われていますが、これはおそらくアルファチャネルのサポートを利用するためと思われます。アルファチャンネルをサポートしている値とサポートしていない値の出現頻度を調べたところ、使用されているカラーフォーマットの77%はアルファチャンネルをサポートしていないことがわかりました。

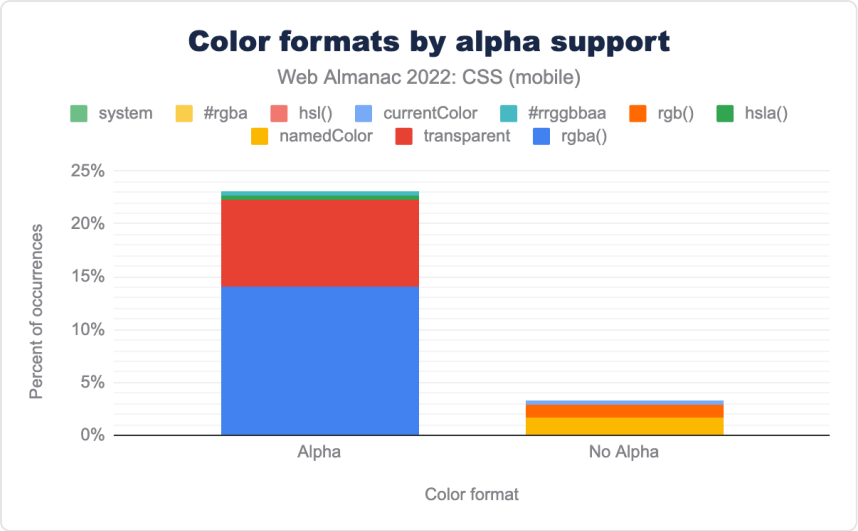



*図1.31. アルファ対応によるカラーフォーマットの分布。*

他のデータから予想されるように、`rgba()`はもっとも普及しているアルファ対応フォーマットで、`transparent`キーワードがそれに続いています。`hsla()`のような他のフォーマットはほとんど使われていません。

## 新しいカラープロパティと値

色の世界ではおもしろいことが起きています。新しい色空間に加え、色に関連するプロパティや値も多数登場しています。私たちは、これらのどれかがデータに影響を及ぼしているのではと考えました。

`accent-color`プロパティを使うと、チェックボックス、ラジオボタン、レンジスライダーなど、スタイルが決まりにくいフォーム要素にアクセントカラーとしてブランドカラーを追加できます。今年3月から全エンジンに搭載されたばかりのためか、使用率はまだ0.3%未満です。

今年、すべてのエンジンで利用可能になったもうひとつのプロパティは`color-scheme`で、コンポーネントをどの配色（明暗）でレンダリングするかを指定できます。このプロパティは、意外なことに、今のところ0.2%のページでしか見つかっていない。

# グラデーションと画像

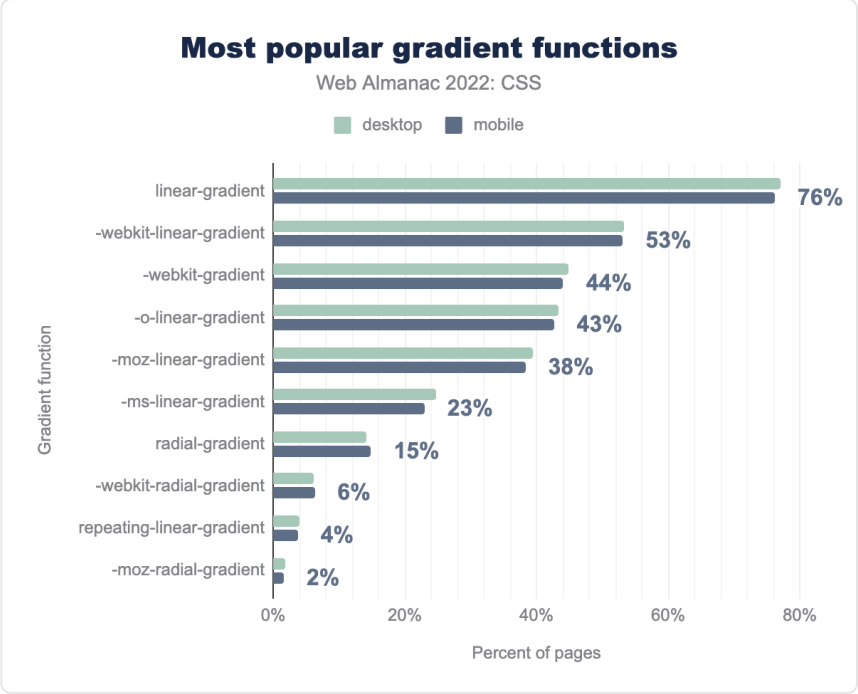



*図1.32. もっとも普及しているグラデーション機能をページ数の割合で紹介。*

リニアグラデーションは引き続き主要な選択肢であり、2021年よりもわずかに高い割合で表示されていますが、グラデーションの使用は過去2年間ほぼ同じです。`linear-gradient`プロパティに関しては、9年以上前からすべてのエンジンで接頭辞なしでサポートされているにもかかわらず、接頭辞の使用頻度が非常に高いです。

## 画像フォーマット

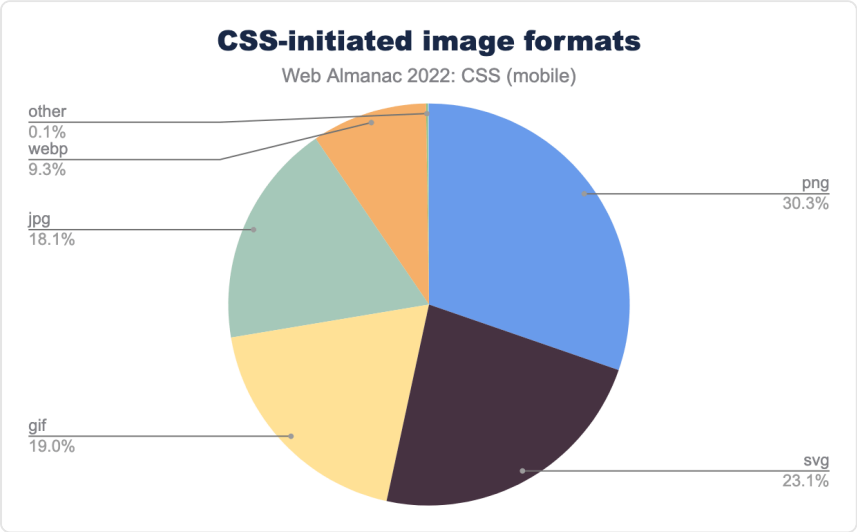



*図1.33. CSSから読み込まれる画像フォーマット。*

この表は、CSSから読み込まれる画像の画像形式を分類したものです。HTMLから読み込まれる画像は含まれず、スタイルルールに表示される画像のみです。PNGが44%から30%に減少し、SVGとWebPがそれぞれ6%ポイント増加しています。

## CSSに含まれる画像の数

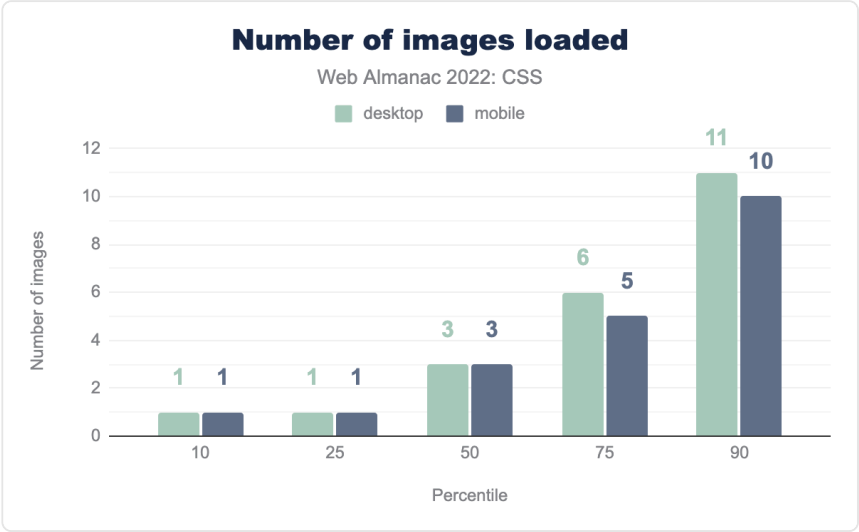



*図1.34. CSSから読み込んだ画像の枚数分布。*

CSSから読み込まれる画像の数は、2021年と同じままです。CSSは多くの画像を読み込む原因にはなりません。下位2パーセンタイルはそれぞれ1枚、90パーセンタイルでも、すべての画像タイプで10枚前後を推移しています。

## CSSにおける画像の重み付け

CSSでは画像の読み込みはあまり発生しませんが、その画像の重さは重要です。データでは、画像の数は変わらないにもかかわらず、画像の重さが2021年から増えていることがわかりました。

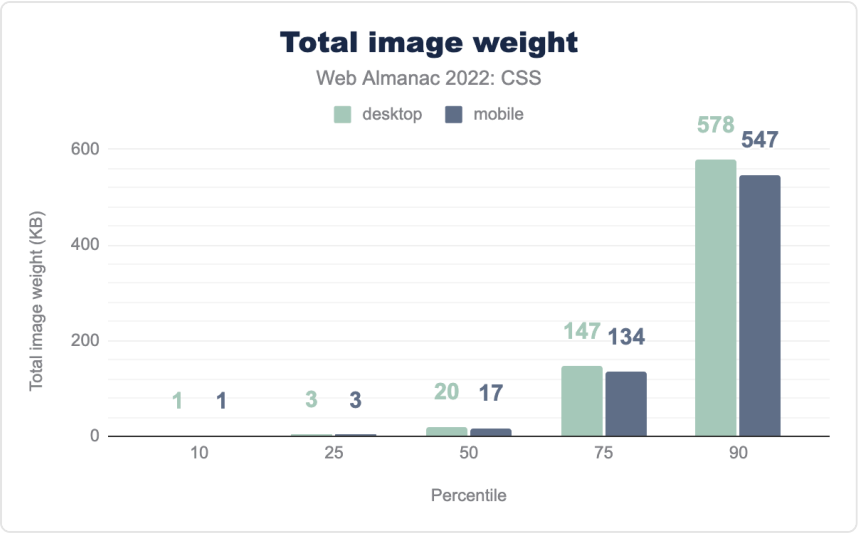



*図1.35. CSSから読み込まれる画像の総重量の分布。*

中央のページでは、モバイルで1KB増加し、17KBとなっています。しかし、グラフの上端、90パーセンタイルでは、モバイルで67KB、デスクトップで42KB増加していることがわかります。2021年と同様、モバイルでは一貫してウェイトが低くなっており、開発者がより小さな画像をモバイルのコンテキストに提供しようとしていることがうかがえます。

### CSSにおける画像のピクセルサイズ

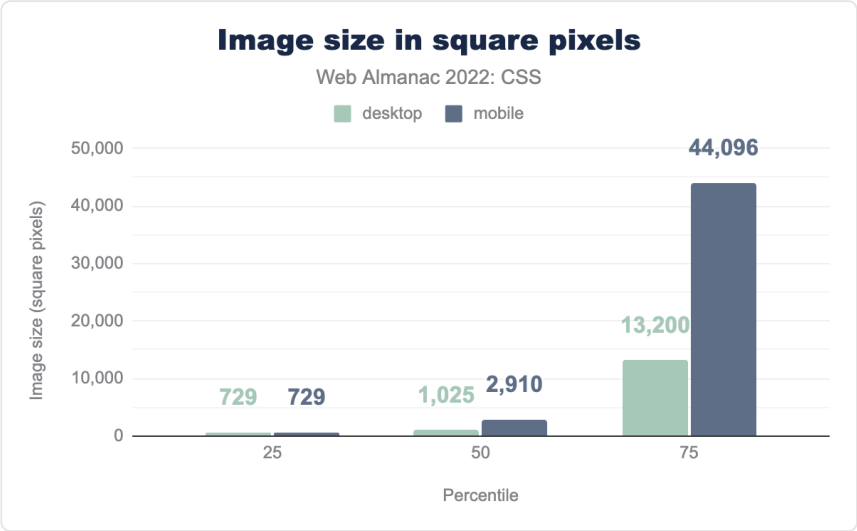



*図1.36. CSSから読み込んだ画像のサイズ分布。*

このグラフは興味深いもので、グラフの下限では、デスクトップとモバイルでほぼ同じサイズの画像を提供していること、50および75パーセンタイルでは、デスクトップよりもはるかに大きな画像をモバイルユーザーに提供していることを表しています。このデータからわかることは、モバイルユーザーに対して、より大きな画像を配信しているということです。おそらく、タブレット端末のランドスケープモードを考慮しているのでしょう。

## レイアウト

ウェブでレイアウトを行う際には、多くの選択肢から選ぶことができ、ほとんどのサイトでは、これらのさまざまな方法が使用されていることでしょう。使用されているレイアウト方法を検出するために、プロパティと値の組み合わせでデータを簡単に検索すると、次の表が得られます。

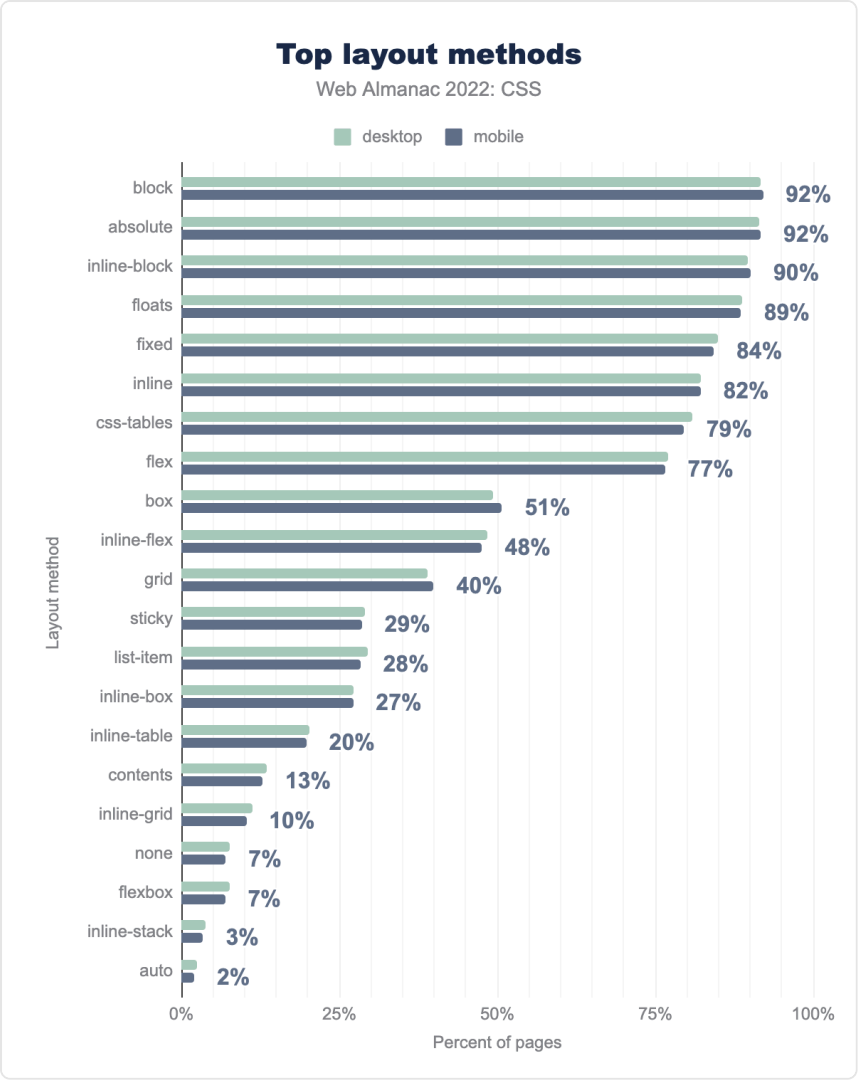



*図1.37. ページ数の割合によるレイアウト方法。*

この表は、そのページで使われている主なレイアウト方法を教えてくれるものではありません。これらのページのCSSに表示されるプロパティまたは値を示しています。たとえば、51% のページが2009年の古いバージョンのflexboxを使用しており、`display: box` と表示されています。これは後方互換性のために、おそらくAutoprefixerなどのツールで追加されたものと思われます。

## フレックスボックスとグリッドの採用

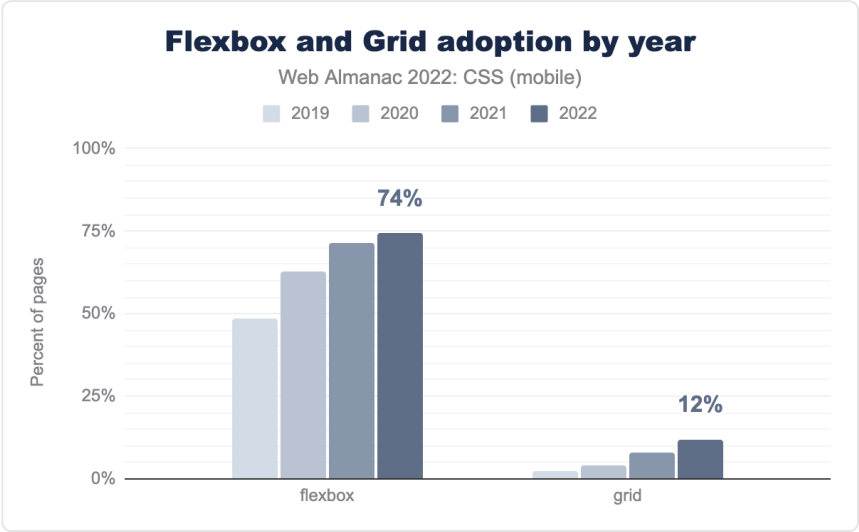



*図1.38. 過去4年間のFlexboxとグリッドの採用状況。*

フレックスボックスとグリッドの利用が増え続けています。2021年、フレックスボックスの採用率は71%でしたが、現在は74%に達しています。グリッドは8%から12%に急増しています。前のセクションとは対照的に、ここで測定されているのは、レイアウトに実際にフレックスボックスやグリッドを使用しているページの割合であり、スタイルシートに単に何らかのフレックスボックスやグリッドのプロパティを持つページとは異なることに注意してください。

グリッドの採用はかなり遅れています。これは、レイアウトに使用されるフレームワークが普及しており、その多くがフレックスボックスをベースとしたレイアウトを採用しているためだと思われます。

また、この新機能の採用がどのように進展しているかを見るために、私たちにとって新しい `flex` と `grid` のプロパティの値もいくつか見てみました。

`flex-basis` プロパティのcontentの値は、ブラウザがアイテムに設定された幅ではなく、アイテムに内在するコンテンツのサイズを参照するよう明示的に指示するものです。これは新しい値で、執筆時点ではSafariのリリースバージョンでは使用できません。現在、この値を使用しているのは、モバイルサイトの0.5%、デスクトップサイトの0.6%にすぎません。

`grid-template-rows` と `grid-template-columns` の `subgrid` 値は、クエリを実行した時点では、Firefoxでのみサポートされています。当然といえば当然ですが、データセット全体のうち、モバイル向けは211ページ、デスクトップ向けは212ページにしか登場しませ

ん。Interop 2022プロジェクトの一環としての価値なので、これが相互運用可能になった後、どのように支持が広がっていくのかに注目したいですね。

### Box sizing


*図1.39.* `box-sizing: border-box` *を設定したページの割合。*

ウェブは圧倒的に、オリジナルのW3Cボックス モデルを拒否して `box-sizing: border-box` を支持することに票を投じました。このプロパティと値の組み合わせを使っているページの数は、また少し増えて90%以上になっています。


*図1.40. セレクター* `*` *で* `box-sizing: border-box` *を宣言しているページの割合です。*

分析したページのほぼ半数が、ユニバーサルセレクター（`*`）を介して、ページ上のすべての要素に `border-box` のサイズ設定を適用しています。

約22%のページがチェックボックスとラジオボタンに `border-box` を使用しています。また、`.wp-` クラスが多く、分析したページの20%でWordPressが使用されていることがわかります。

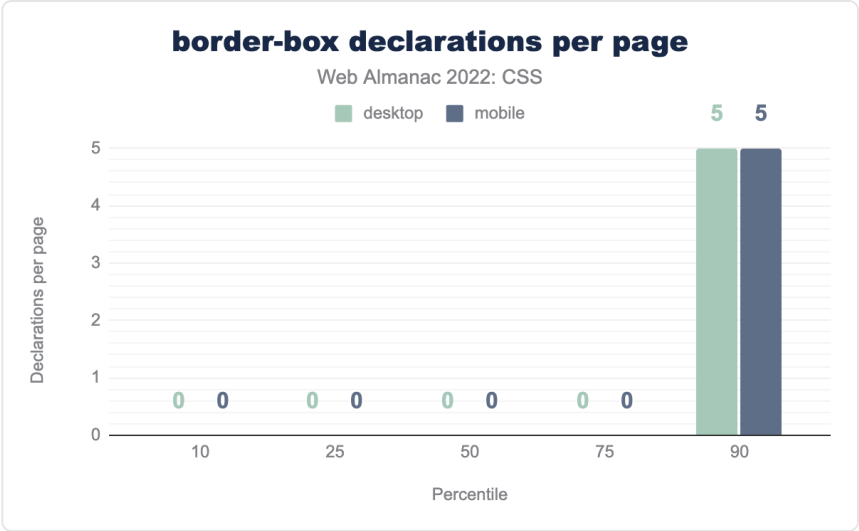



*図1.41. ページごとの `border-box` 宣言の数の分布。*

モバイルページの中央値では、`border-box`は22回宣言されています。90パーセンタイルでは、101回と圧倒的な回数です。なお、前年のクエリには、この指標に影響を与えるバグがありました。それを修正すれば、2021年の結果は同等になります。

### マルチカラム

**23%**

*図1.42. マルチカラムレイアウトを使用しているページの割合。*

CSS 段組みレイアウト[6]レイアウトの利用が再び増加し、現在23%のページで見られ、2021年から3ポイント上昇しました。

6. *https://developer.mozilla.org/docs/Web/CSS/CSS_Columns*

### `aspect-ratio` プロパティ

**2%**

*図1.43. `aspect-ratio` プロパティを使用しているページの割合です。*

新しい `aspect-ratio` プロパティは、2%のページで使用されています。これは2021年末から相互運用可能になったので、このプロパティの使用率が時間とともに伸びていくのが興味深いところです。

## トランジションとアニメーション

`animation` プロパティは、モバイルページの77%（昨年と同じ）に表示され、デスクトップでは76.8%とわずかに増加しました。`transition` プロパティはさらに人気があり、モバイルページの85%、デスクトップページの85.6%に見受けられます。デスクトップの頻度は、2021年から4ポイントほどわずかに減少しています。

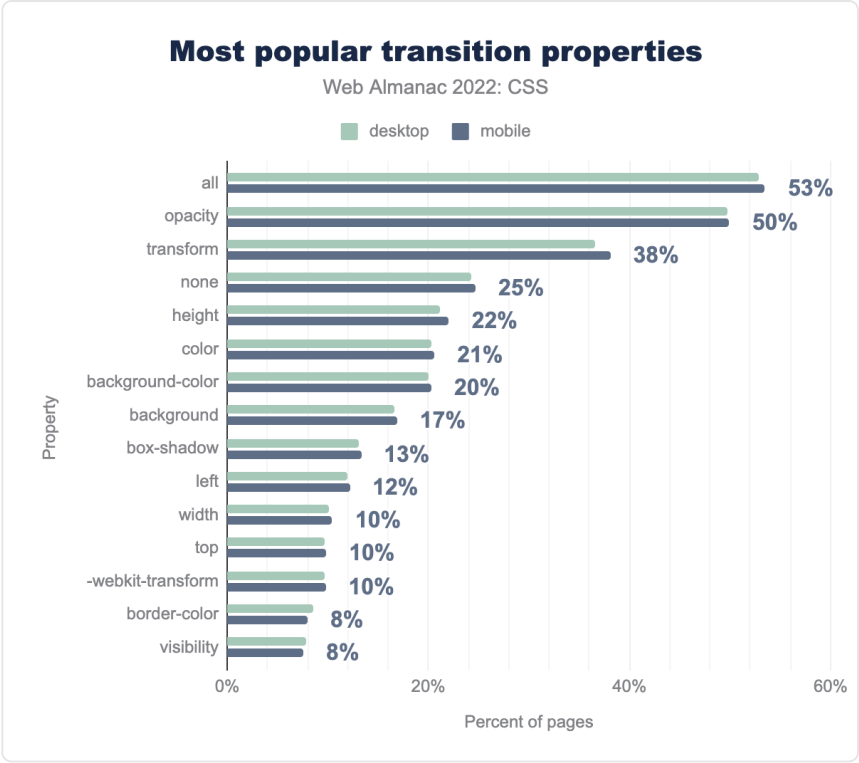



*図1.44. もっとも普及している `transition` プロパティを、ページ数の割合で表示します。*

昨年と同様、もっとも一般的な使用法は、`all` キーワードですべてのアニメート可能なプロパティにトランジションを適用することです。この使用法は7ポイント増の53%に達し、次いで50%のページで `opacity` が使用されています。

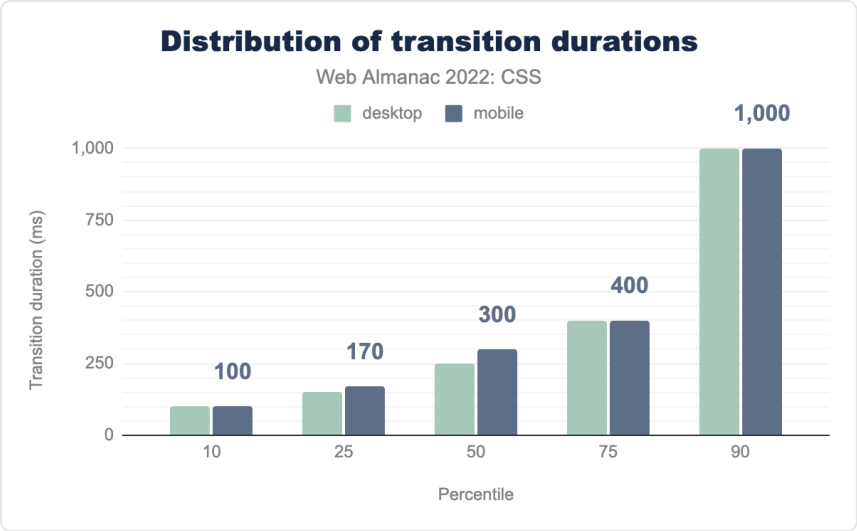



*図1.45. 遷移期間の分布。*

遷移の持続時間を見ると、昨年からの変化が見られます。 2021年、90パーセンタイルでは、遷移時間の中央値は0.5秒でしたが、現在では1秒に跳ね上がっています。上位4つのパーセンタイルすべてで増加していることがわかります。

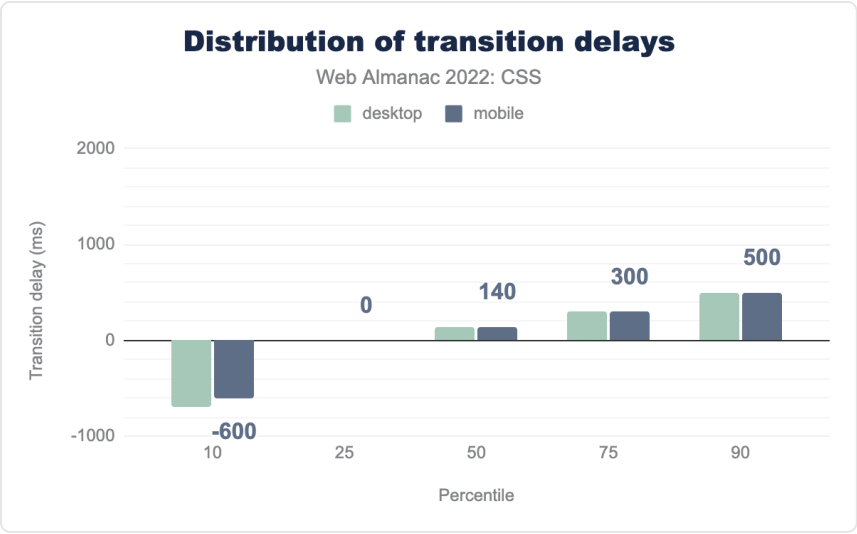



*図1.46. 遷移遅延の分布。*

また、遷移遅延の分布も変化しています。90パーセンタイルの遅延は1.7秒から0.5秒に減少している。しかし、10パーセンタイルの遅延の中央値は0.5秒以上になっています。これは、トランジションがアニメーションの途中から始まる場合に見られます。

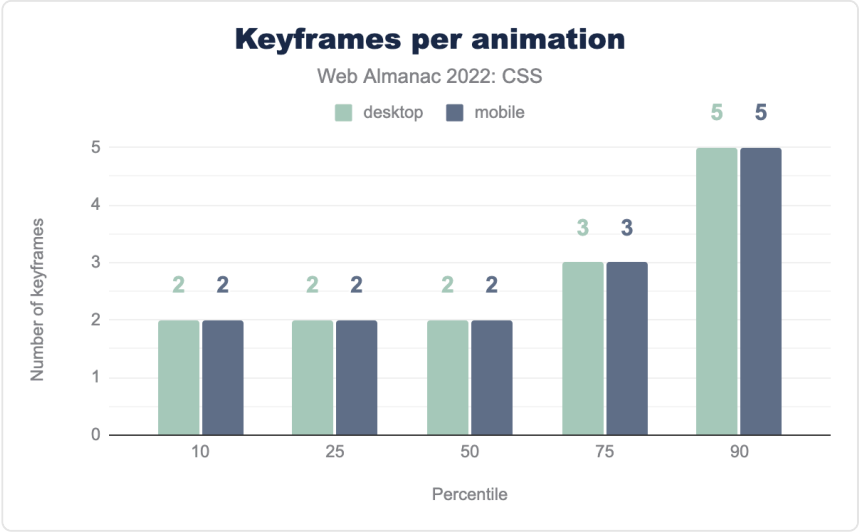



*図1.47. アニメーションごとのキーフレームの分布。*

また、1つのアニメーションに使用されるキーフレームの平均数を調べたところ、6,995個という驚異的な数のキーフレームを使用しているサイトが見つかりました。しかし、これは異常なことで90パーセンタイルでも、1アニメーションあたりのキーフレーム数は、デスクトップとモバイルの両方で5個です。

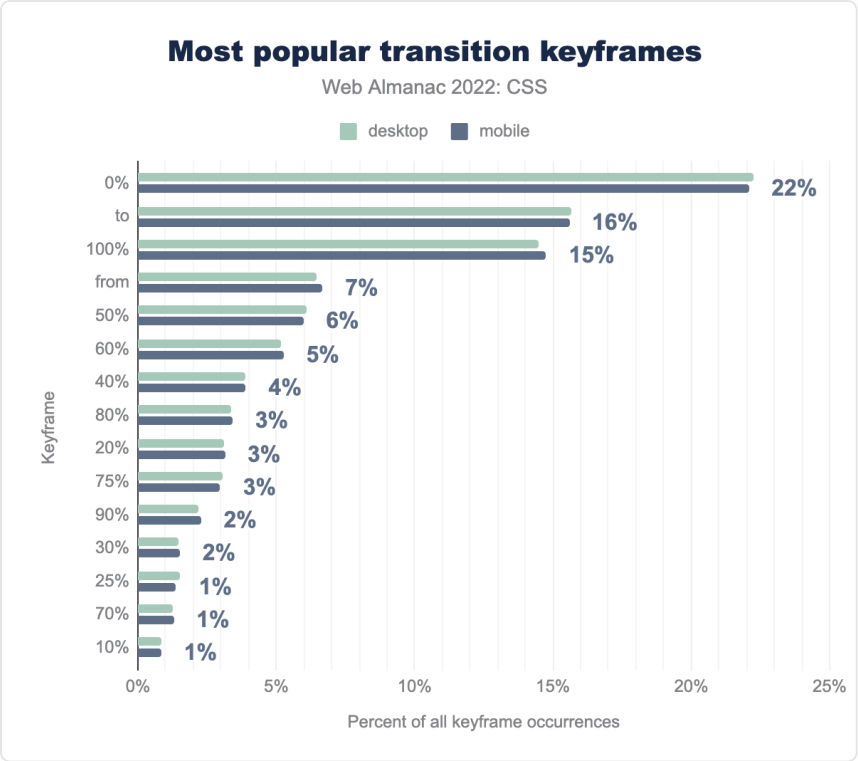



*図1.48. もっとも普及しているトランジションキーフレームを出現率で表示。*

ご想像の通り、もっとも普及しているのは0%から100%までで、次いで50%となっています。開発者は一般的にこれらのストップを10%間隔で設定し、たとえば33%を使っているページはわずか1%です。

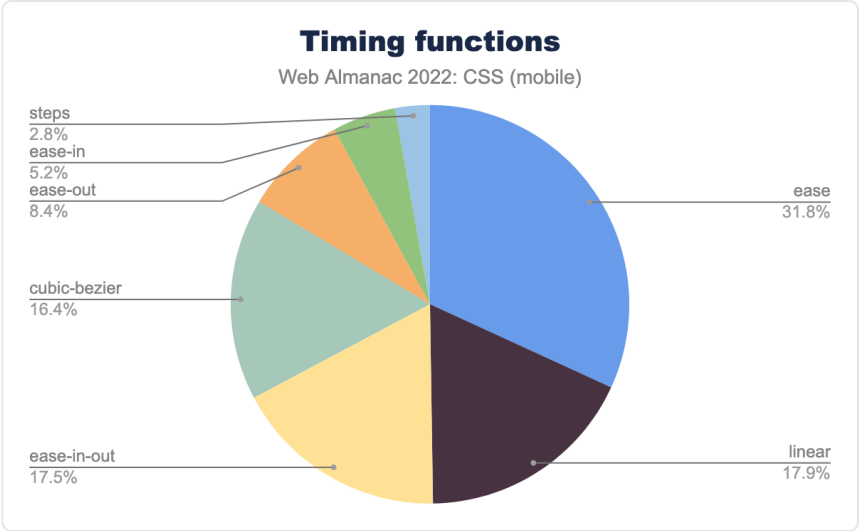



*図1.49. タイミング機能の配布。*

2021年と比較すると、遷移時に使用されるタイミング機能の分布はほとんど変化していない。当時と同様に、明確なトップは`ease`です。

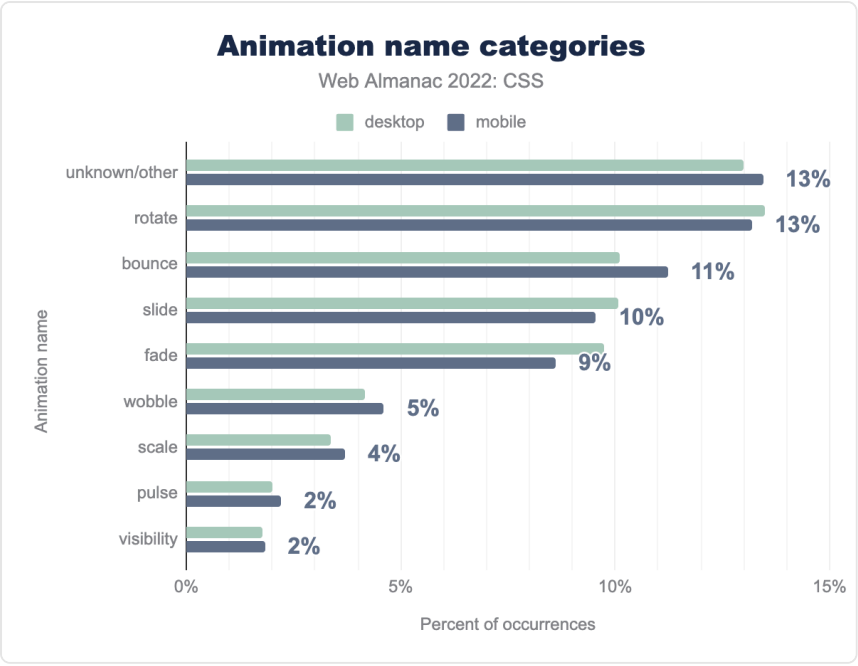



*図1.50. アニメーション名で識別されるアニメーションの種類。*

開発者がアニメーションを何に使っているかを理解するために、アニメーションクラスに使われている名前を見てみました。たとえば、クラス名に `spin` が含まれるものは、rotateと判断されます。Rotateアニメーションは、2021年と同じくもっとも普及しています。しかし、その割合は18%から13%に低下し、bounceアニメーションは5位から3位にランクアップしています。

昨年同様、unknown/otherが多いのは、クラス名 `a` が多いためで、特定のアニメーションタイプに対応させることができないためです。

## 視覚効果

**18%**

*図1.51. ブレンドモードを使用しているページの割合です。*

CSSで使用されているいくつかの視覚効果について調べました。たとえば、デスクトップペ

ージの18%は `background-blend-mode` または `mix-blend-mode` プロパティでスタイルを定義しています。

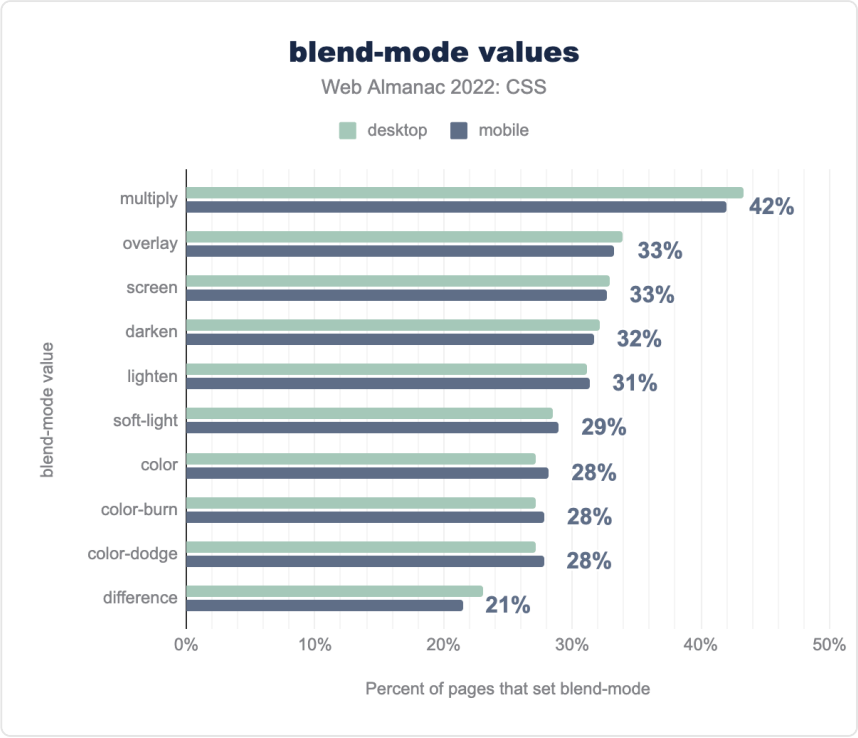



*図1.52. ブレンドモードを設定するページでもっとも普及しているブレンドモード。*

ブレンドモードでもっともよく見られる値は `multiply` で、42%のページで見られました。しかし、他の値についても、かなりの割合で分布しています。

約18%のページが、カスタムプロパティ `var(--overlay-mix-blend-mode)` を使っていました。これは、何らかのライブラリやツールに由来する、特定の名前です。

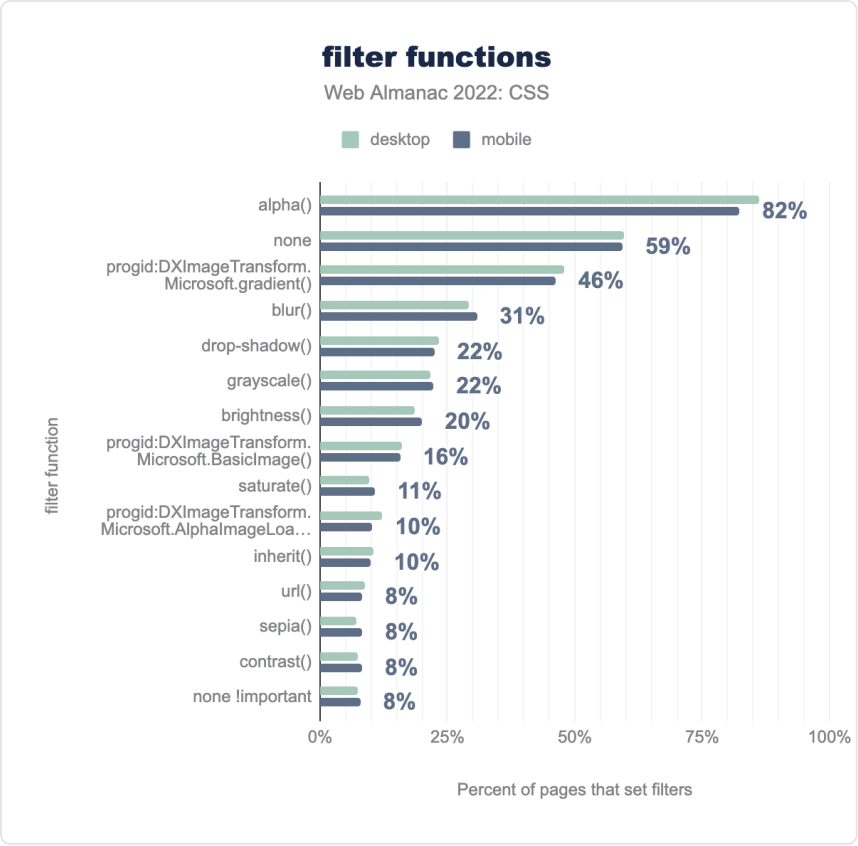



*図1.53. フィルターを設定するページでもっとも普及しているフィルター機能。*

グラフィカルな効果を適用するフィルターを設定しているページの割合のうち、82%がInternet Explorer 8以下で使用される非標準の値である `alpha()` を使用しています。また、`Microsoft.gradient()` フィルターの使用率も高いことがわかります。

標準値[7]のうち、31%のページで `blur()` が使用されており、`none` に次いでもっとも普及している値となっています。

7. *https://developer.mozilla.org/docs/Web/CSS/filter*

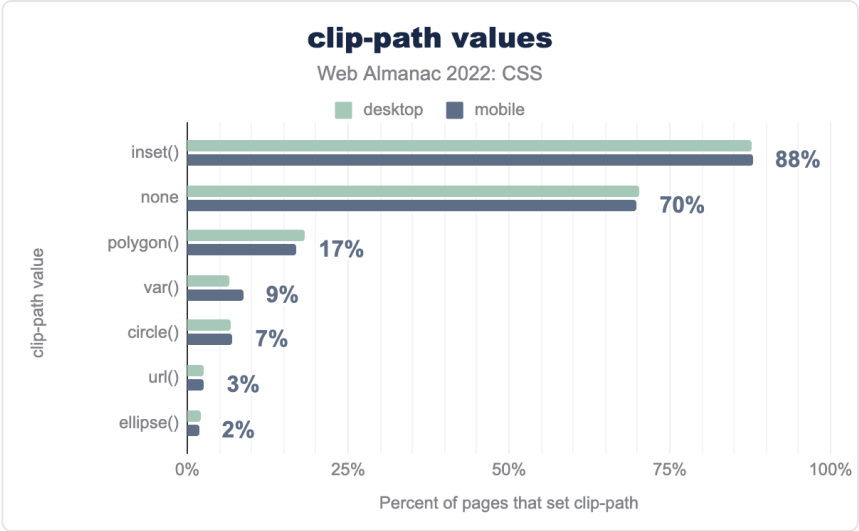



*図1.54. `clip-path()` を設定したページで、人気のある `clip-path` 値です。*

`clip-path` を使って要素をクリップしているページでは、大多数が `inset()` という、単純に要素のボックスを挿入する値を使っており、`clip-path` を使っているページの88% がこの関数を使っているとのことです。

その後、値 `none` もありますが、ほとんどの開発者は、独自のパスをもっとも柔軟に定義できる値である `polygon()` を使うことにしています。

## レスポンシブデザイン

多くの開発者がコンテナクエリ[8]を熱望しており、フレックスボックスやグリッドなどの新しいレイアウト手法によって、複数の画面サイズでうまく動作するデザインが可能になることも多いですが、レスポンシブデザインを行うページの大半でメディアクエリ[9]が使用されています。

開発者がメディアクエリを書くとき、もっとも多いのはビューポートの幅をテストすることです。`max-width` と `min-width` は、2020年、2021年と同じように、もっとも普及しているクエリでした。3位と4位の結果も順位変動はありませんでした。

8. *https://developer.mozilla.org/docs/Web/CSS/CSS_Container_Queries*
9. *https://developer.mozilla.org/docs/Web/CSS/Media_Queries/Using_media_queries*

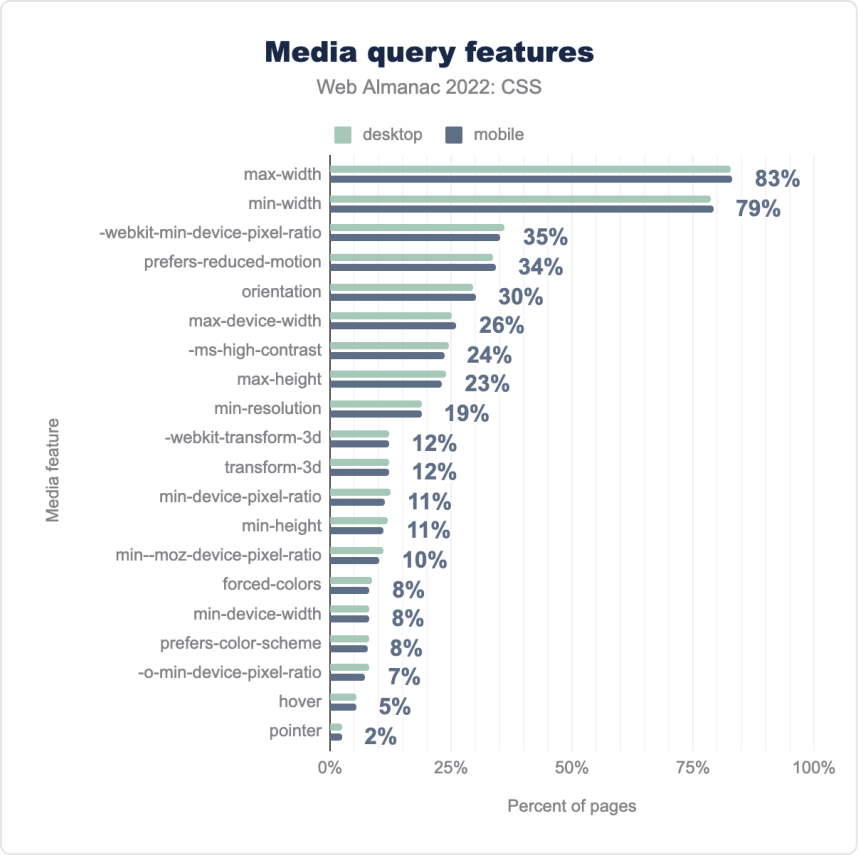



*図1.55. 人気のメディアクエリ機能。*

しかし、2021年にランキング上昇を指摘された`prefers-reduced-motion`メディアクエリは、今回`orientation`を抑えて4位を獲得しました。これは、`prefers-reduced-motion`が2%上昇した一方で、`orientation`が4%下落したためです。

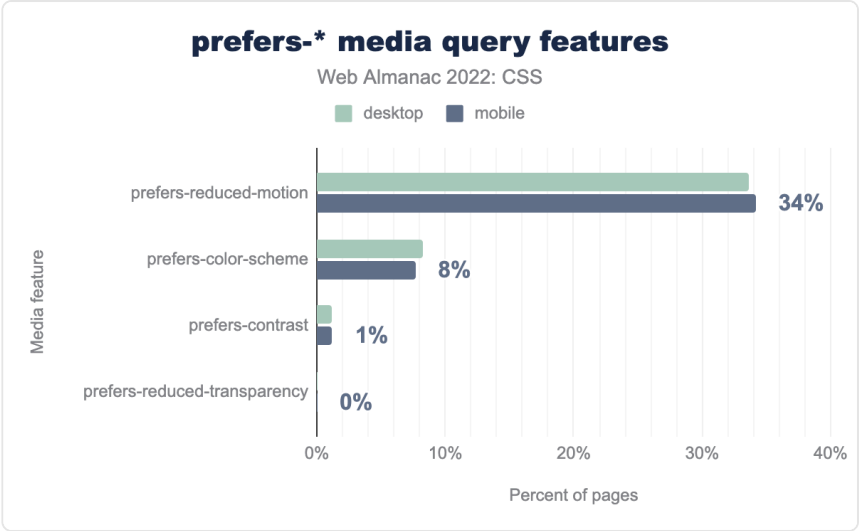



*図1.56. ユーザー・プレファレンス機能の使用率（ページ数ベース）。*

`prefers-*` ユーザープリファレンス機能だけを見ると、`prefers-reduced-motion` がもっとも普及していることがわかります。これは、ブラウザのサポートが良く、ウェブ上でアニメーションや遷移が普及していることが原因です。`prefers-color-scheme` 機能は、ユーザーが明るい色か暗い色のどちらを好むかをチェックする機能で、ウェブサイトやアプリケーションでダークモードを使用することが一般的になったため、使用率がわずかに増加しています。

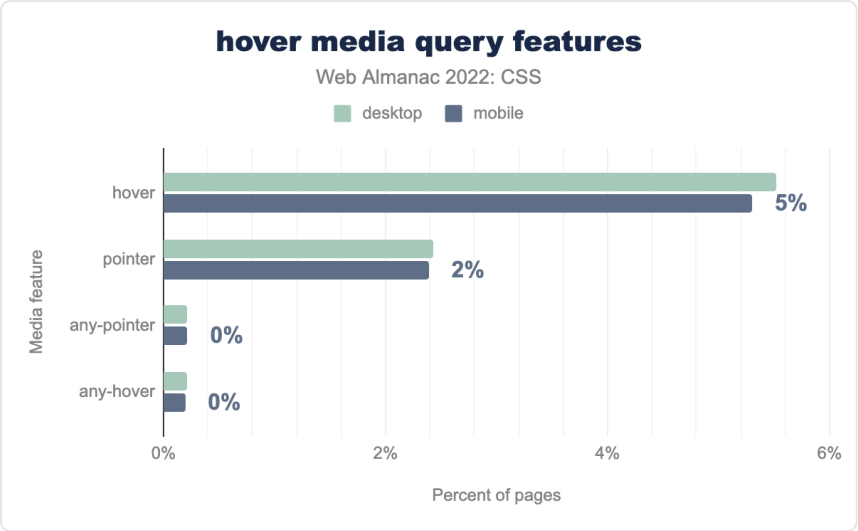



*図1.57. ホバーやポインターのメディア機能を利用する。*

メディア機能の `hover` と `pointer` は、開発者がデバイスの機能をテストし、ユーザーがどのようにデバイスとインタラクトしているかを確認するのに役立ちます。たとえば、大型のタブレットやタッチスクリーンのラップトップが数多く使用されていることから、ユーザーがタッチスクリーンを使用しているかどうかを発見するには、スクリーンサイズだけよりもこれらの機能の方が適しています。

`hover` と `pointer` の両方がもっとも多い機能のトップ10に入っています。あまり役に立たない `any-pointer` と `any-hover` はほとんど使われていないようです。`any-pointer` を使用すると、 `pointer` が現在タッチスクリーンを使用していることを示していても、ユーザーがマウスやトラックパッドなどの細かいポインターにアクセスしているかどうかを判断できます。これらの機能を組み合わせることで、ユーザーがどのような環境で作業しているかを把握できます。

### 共通ブレークポイント

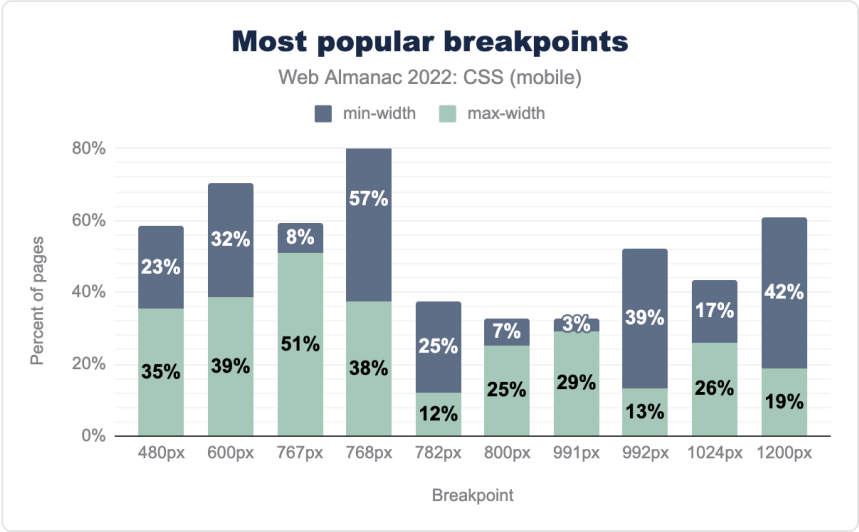



*図1.58. もっとも普及しているブレイクポイントの分布。*

過去2年間と同様、共通のブレイクポイントはほとんど変化していません。グラフは同じ形をしており、もっとも一般的なブレークポイントは、`max-width`が767px、`min-width`が768pxとなっています。2021年に述べたように、これはポートレートモードのiPadに対応します。

繰り返しますが、ブレイクポイントは圧倒的にピクセル値で設定されているので、他の値をピクセルに変換してグラフにすることはしていません。最初の`em`値は、やはり`48em`で、位置78で見つかりました。

## クエリで変更されたプロパティ

メディアクエリブロック内に表示されるプロパティを調べ、ブレークポイントに応じてどのプロパティを変更しているかを確認しました。

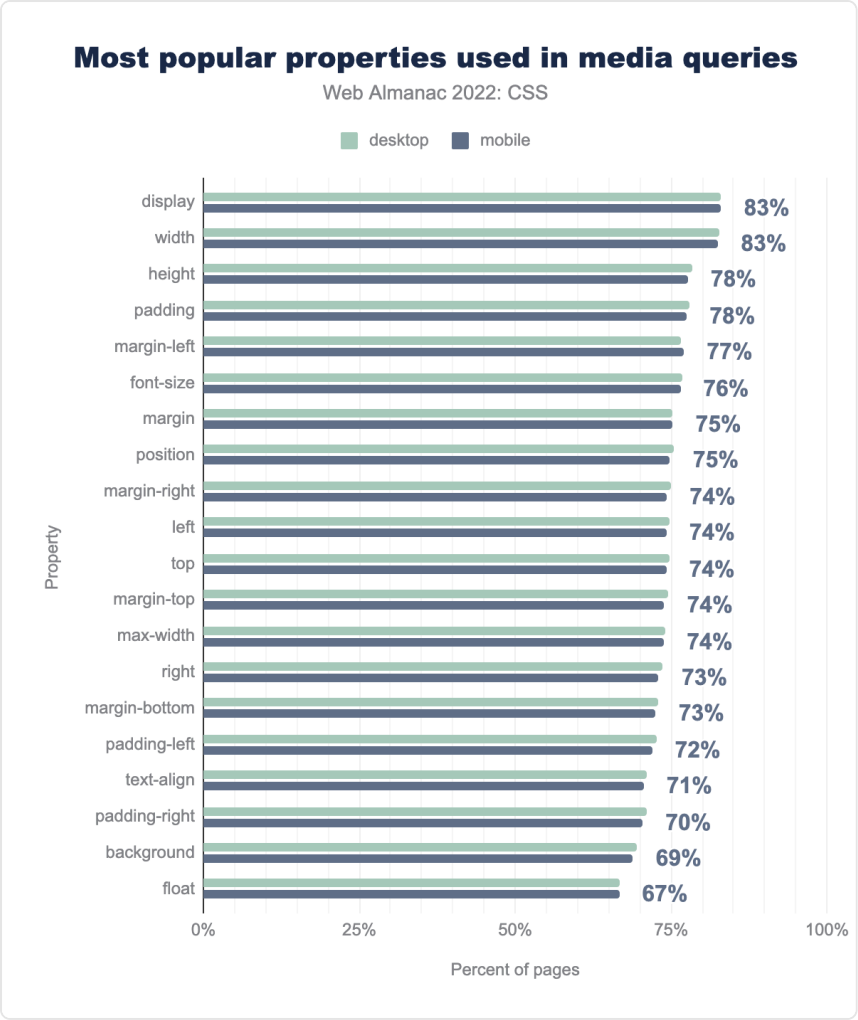



*図1.59. メディアクエリーブロックでもっとも普及しているプロパティです。*

メディアクエリで変更されたプロパティでは、`display`プロパティがもっとも多いのですが、順位に若干の入れ替えがありました。これらは、見た目ほど劇的な変化ではありません。`color`プロパティはチャートから消えましたが、これは74%から67%への変化を表しているに過ぎません。しかし、`background-color`の使用率は65%から63%に減少しています。これは、何らかのフレームワーク、あるいはWordPressがスタイルシートでこれを使用しなくなったのではないかと思わされます。

もう1つ興味深いのは、2020年には`font-size`がメディアブロックの73%に出現し、5位で

あったことです。2021年には60%のブロックに表示され、12位に表示されました。今年は76%、6位と順位を上げている。

## 機能クエリ

CSS機能の対応テストに使用されるフィーチャー・クエリーは、モバイルページの40%、デスクトップページの38%で確認されました。これは、2021年の48%という数字から減少しています。これは、テストされた一般的な機能のサポートが、使用前に機能のテストを気にする必要がないほど大きくなっていることを示しているのかもしれません。

1ページあたりの機能クエリブロック数は、75%で4、90%でデスクトップが7、モバイルが8となっています。ただし、1,722個の特徴クエリブロックを持つサイトがありました。

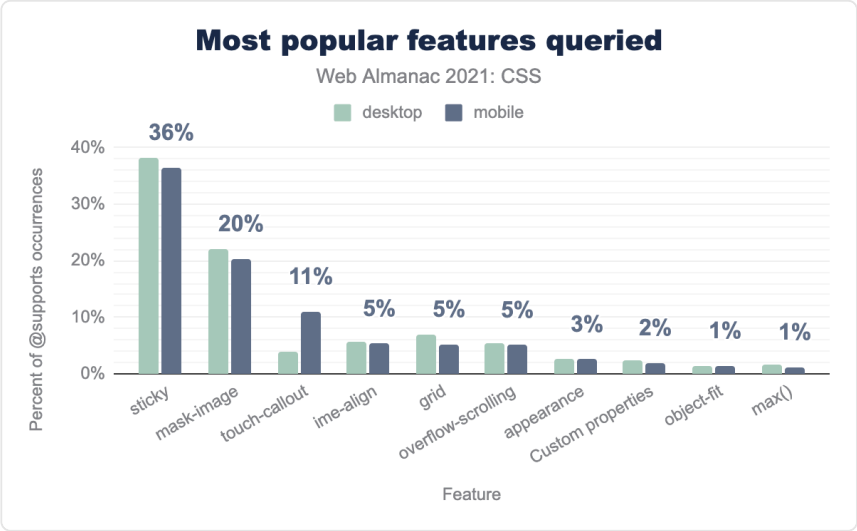



*図1.60. フィーチャークエリでテストされたもっとも普及している機能。*

昨年と同様、機能検索でもっとも普及している機能は`position: sticky`ですが、この機能のブラウザサポートが向上したためか、出現率は53%から36%に減少しています。

このテストでは、非標準の機能が強く現れており、`touch-callout`(`-webkit-touch-callout`)と`ime-align`(`-ms-ime-align`)があります。前者は使用率が5%から11%に増加し、`ime-align`は7%から5%に減少しています。

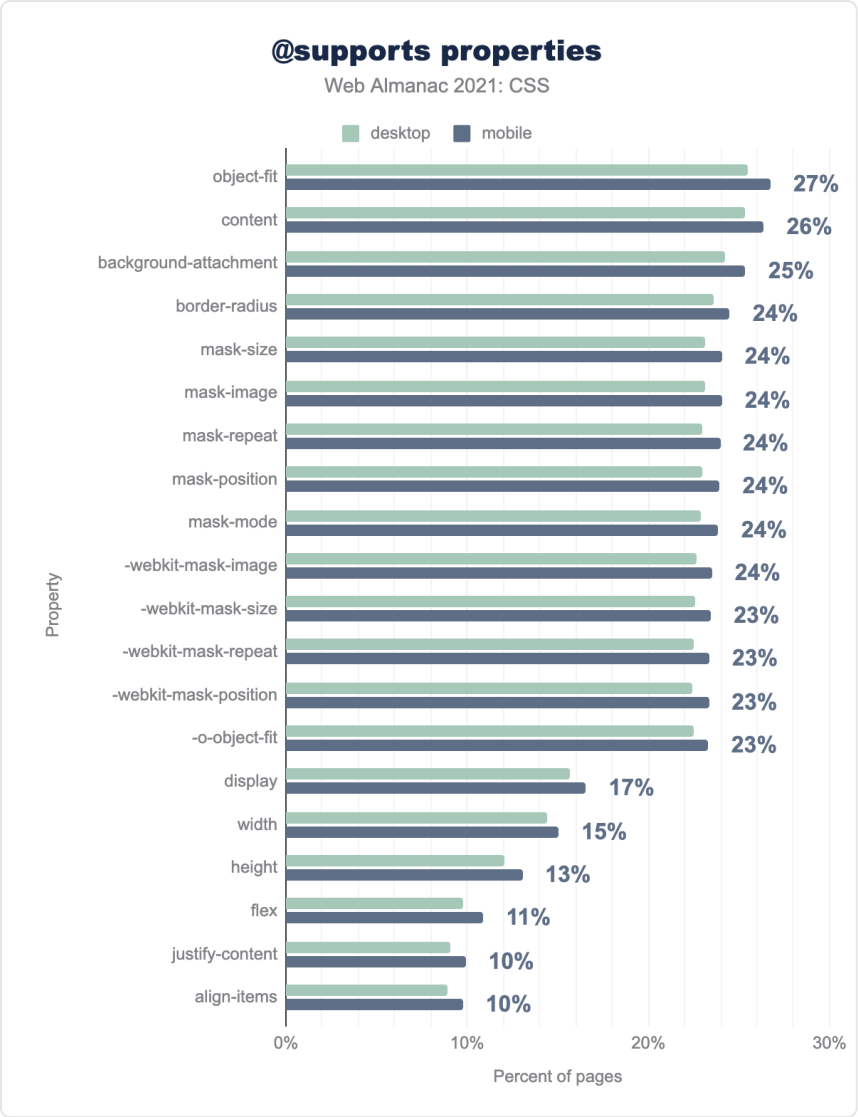



*図1.61. フィーチャークエリーブロック内で使用されているプロパティ（ページ数の割合）。*

サポートテストを行った後、これらの機能クエリブロックの内部で、どのプロパティを使用するのでしょうか？`object-fit` プロパティがもっとも多く、`mask-*` プロパティは `-webkit-mask-*` プロパティと同様に良い結果を出しています。これは、最近までマスクの相互運用性がなく、Chromeではプロパティがまだ `-webkit` プレフィックスを必要と、し

たためと思われます。

`display` プロパティは上位20位以内にありますが、グリッドプロパティを見つけるには、リストのかなり下の方に行かなければなりません。`grid-template-columns` プロパティは、機能クエリブロックの2%で見つかりました。

## 国際化

英語は、文章がページの上から順に水平に書かれていることから、水平トップ・トゥ・ボトム言語と言われている。文字の方向は左から右へ（LTR）です。アラビア語、ヘブライ語、ウルドゥー語も上から下への水平方向の言語だが、スクリプトの方向は右から左（RTL）です。また、中国語、日本語、モンゴル語など、上から下へ垂直に書かれる言語もある。CSSは、このような異なる書き方や文字の向きに対応できるように進化してきた。

### 方向性

`direction` プロパティを使ってCSSを `<body>` か `<html>` 要素に設定しているページ数は2021年から変わらず、 `<html>` に設定しているページが11%、 `<body>` に設定しているページが3%となっています。 `direction` を設定するにはCSSではなくHTMLを使うことが推奨されているので、ここではそのベストプラクティスへ合わせた低い数値にしています。

## 論理的・物理的プロパティ

`border-block-start` や `text-align` の `start` のような論理的またはフローに関係するプロパティは、画面の物理的な寸法に縛られるのではなく、テキストのフローに従うので、国際化には有用です。これらのプロパティに対するブラウザのサポートは現在とても優れているので、私たちはもっと採用が進むのではないかと考えていました。

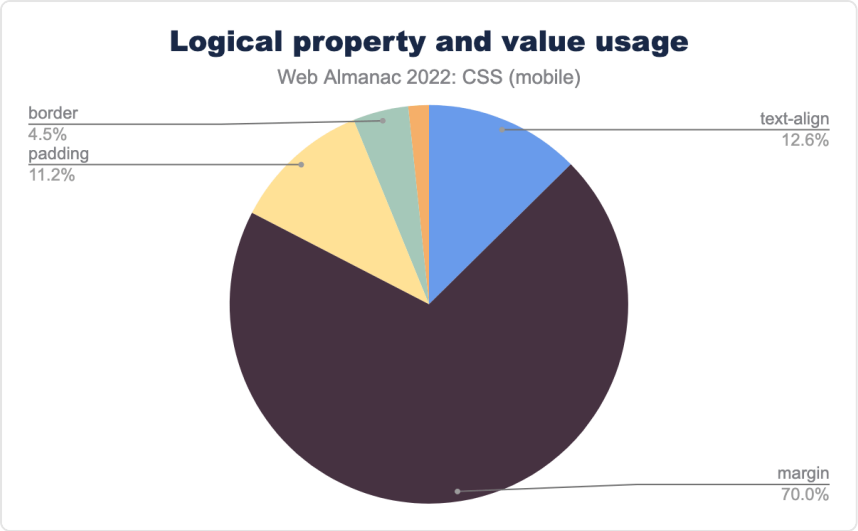



*図1.62. 使用される論理プロパティの分布。*

論理プロパティの使用率は、2021年からわずかに増加し、4%から5%になった。しかし、2022年のグラフは、2021年のグラフと大きく異なっています。圧倒的に多くの人が論理プロパティを使用して `margin` プロパティを設定しており、26%から70%に増加しています。もっとも普及している `margin` プロパティは `margin-inline-start` と `margin-inline-end` で、全体の9%のページで見受けられます。これらは、たとえばラベルとそれに続くフィールドの間隔が、LTRとRTLのスクリプトで同じように動作することを確認するためにとくに有用です。

## Ruby

今回も、CSS Ruby[10]の使用を確認しました。これは、インターリニアアノテーション（ベーステキストに沿った短いテキスト）に使用するプロパティの集合体です。


*図1.63. モバイルページでCSS Rubyを使用している割合です。*

その使用率はまだ微々たるものですが、2021年から増加しました。2021年にはデスクトップ8,157ページ、モバイル9,119ページと、分析した全ページの0.1%未満しか使用されていな

10. https://developer.mozilla.org/docs/Web/CSS/CSS_Ruby

いことが判明しました。今年は、デスクトップ16,698ページ、モバイル21,266ページ、つまり分析した全ページの0.2%が利用していた。

## CSS in JS

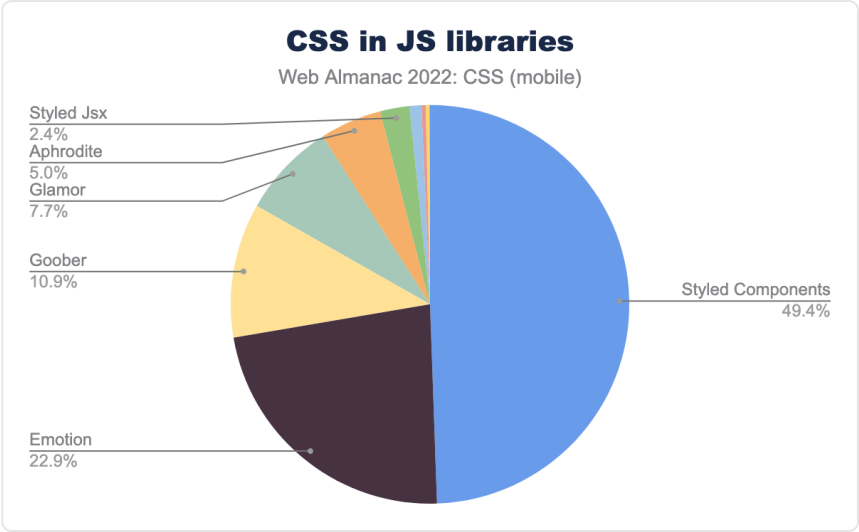



*図1.64. CSS in JSの利用。*

CSS-in-JSの利用は昨年から増えておらず、3%にとどまっています。この利用は、ほとんどがライブラリによるもので、もっとも普及しているのはStyled Componentsです。このライブラリのシェアは57%から49%に低下し、新しいライブラリがほぼ11%でミックスに加わっています。Goober[11] は、自らを「1KB未満のcss-in-jsソリューション」と表現しており、この種のものが好きな人々の間に確実に浸透している。

## Houdini

Houdini[12]は、オープンなWeb上ではまだほとんど使われていないんです。アニメーションのカスタムプロパティを使用しているページ数を見ると、2021年以降、わずかな増加にとどまっています。また、Houdini Paint APIの利用状況も調べてみました。私たちは、Web上でこれが使用されている事例を発見しました。使用されている小作品名を見ると、その多くが

11. *https://goober.js.org/*
12. *https://developer.mozilla.org/docs/Web/CSS/CSS_Houdini*

smooth corners[13] 小作品で、通常の `border-radius` にうまくフォールバックできることから、人々はこれを段階的強化として使用していることがわかります。

## Sass

Sassのようなプリプロセッサーは、開発者がCSSでできるようになりたいと思っていてもできないことの良い指標と見ることができます。そしてCSSのパワーアップに伴い、開発者からのよくある質問は、Sassを使う必要がまったくないのか、というものです。カスタムプロパティの利用が増加していることから、プリプロセッサでよく使われる変数や定数の機能が、CSSに組み込まれ同等の機能を持つようになったことがわかります。

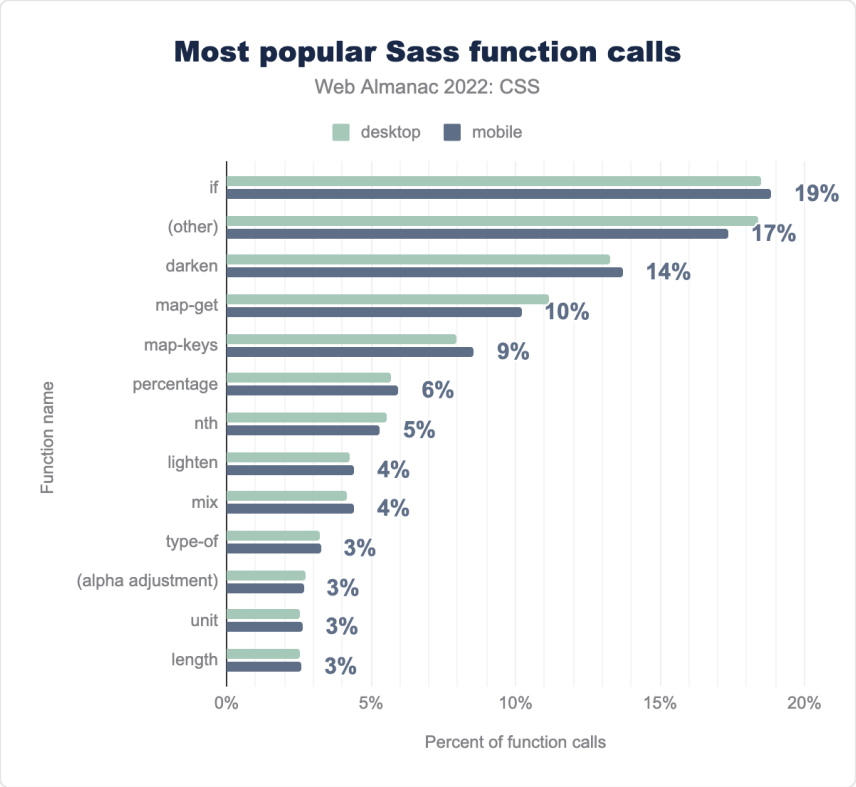



*図1.65. もっとも普及しているSass関数呼び出しの割合。*

関数の呼び出しを見ると、色関数はまだSassの非常に一般的な使い方であり、近いうちに

13. *https://css-houdini.rocks/smooth-corners/*

`color-mix()` などの新しいCSS色関数[14]に置き換えられるかもしれないことがわかります。昨年からの変化もある。`darken` 機能が2ポイント下がって14%で3位となった。しかし、`lighten` 機能は1ポイント増えている。

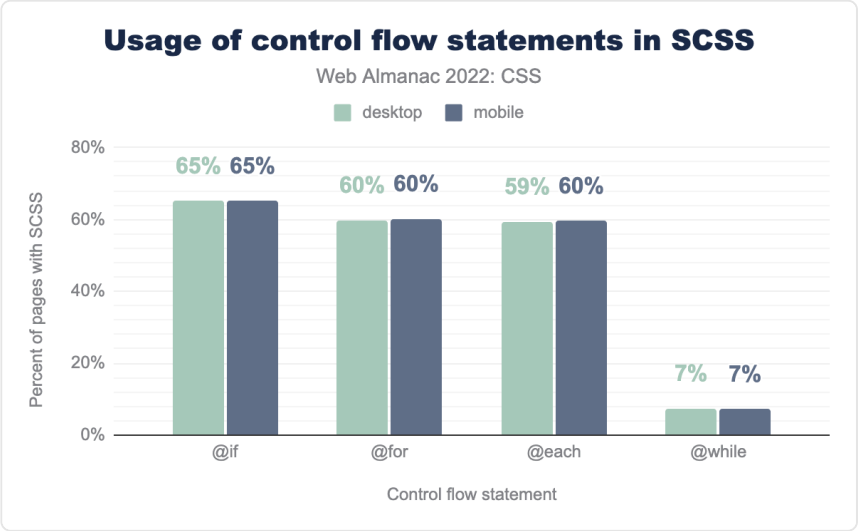



*図1.66. SCSS上の制御フロー文の配布。*

コントロールフローを見ると、`@for` と `@each` は少し増えていますが、`@while` は2%から7%に増えています。

14. https://www.w3.org/TR/css-color-5/

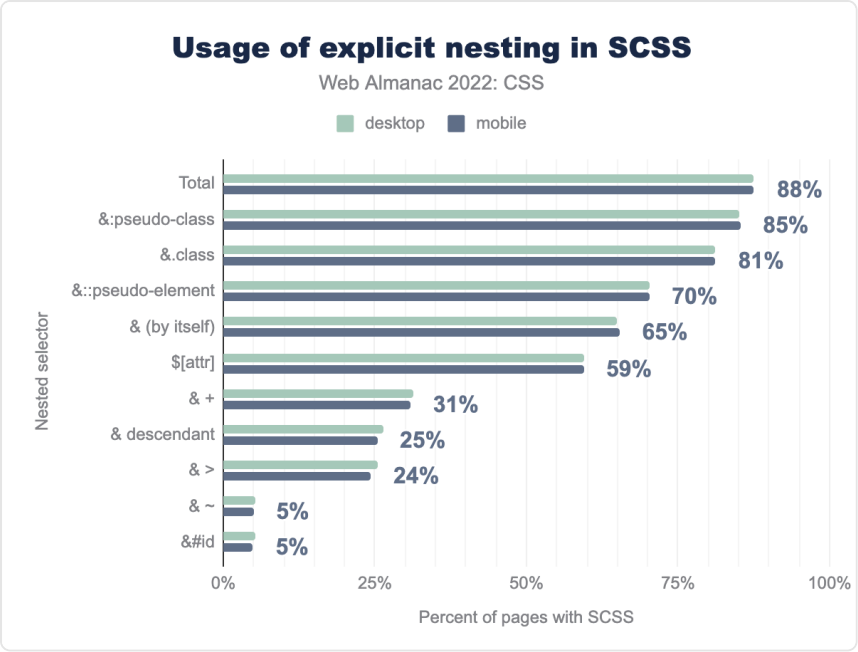



*図1.67. SCSSにおける明示的なネストの使用率（SCSSを使用しているページの割合）。*

ネスティングはまた、CSSネスティングの将来の仕様が現在CSSワーキンググループで開発・議論されていることを考えると、興味深いものです。SCSSシートのネスティングは非常に一般的で、`&`という文字を探すことで識別できます。昨年と同様に、`:hover`のような擬似クラスや、`.active`のようなクラスがもっともよくネストされるケースを構成しています。すべての用途で微増しているが、`&子孫`は18%から25%へと7ポイント増加している。暗黙の入れ子は特殊文字を使用しないため、本調査では測定していない。

## 印刷用CSS

**5%**

*図1.68. 印刷に特化したスタイルを持つデスクトップページの割合。*

よりよい印刷体験を提供するために、開発者が印刷用スタイルシートを作成しているかどうかを調べたところ、デスクトップサイトの5%、モバイルサイトの4%しか作成していませんでした。

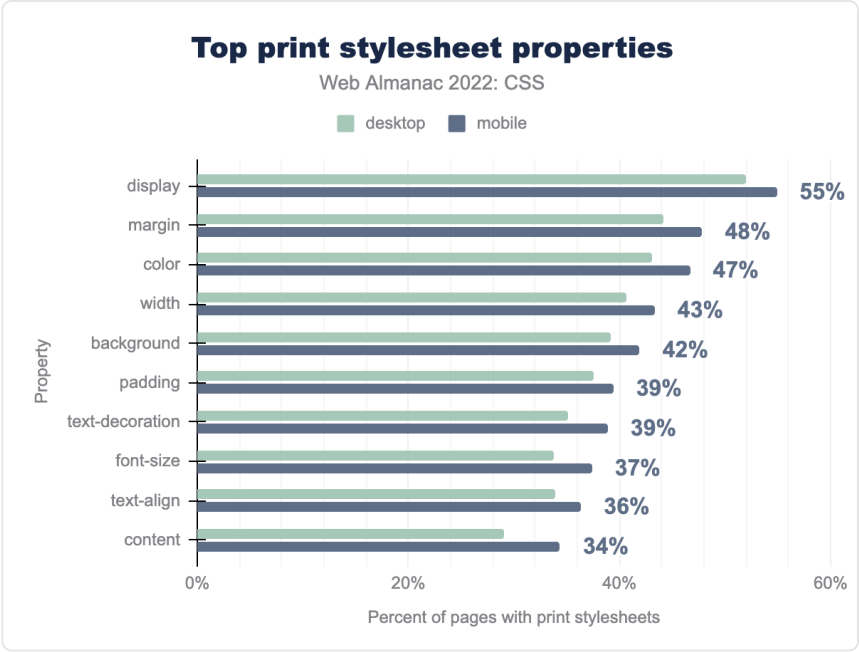



*図1.69. 印刷スタイルシートを持つページの印刷スタイルでもっとも多いプロパティ。*

印刷用スタイルを使用しているページのうち、半数以上が`display`の値を変更しています。おそらく、印刷用にグリッドやフレックスのレイアウトを簡素化するためでしょう。また、色を変えたり、マージンやパディングを調整したり、`font-size`を設定している人も見受けられます。34%は`content`プロパティで、これは生成されたコンテンツを挿入するために使用されます。

印刷物は断片的なメディアです。コンテンツはページごとに分割され、その分割を制御するために一連の断片化プロパティが用意されています。たとえば、開発者は通常、見出しがページの最後に表示されることや、キャプションが関連する図から切り離されることを避けたいと考えています。

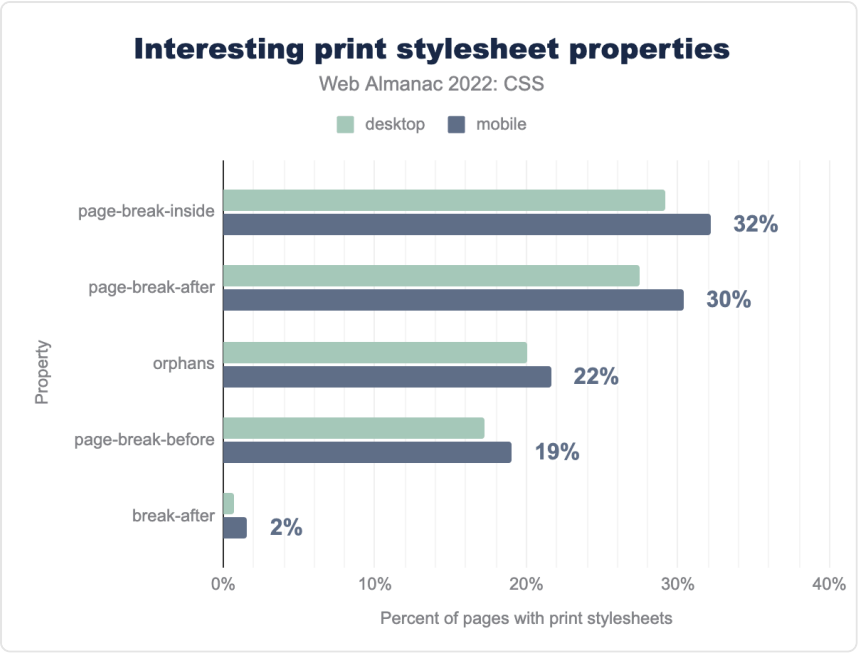



*図1.70. 印刷スタイルシートで使用されるフラグメンテーションプロパティ。*

このグラフから、多くの開発者が、使用率の低い `break-before` のような新しいプロパティではなく、 `page-break-inside`, `page-break-after`, `page-break-before` という古い断片化プロパティを用いていることがわかります。

`orphans` プロパティはFirefoxでサポートされていないにもかかわらず、印刷用スタイルシートの22%で表示されています。このプロパティは、フラグメンテーションブレークの前にページの下に残すべき行数を定義します。 `widows` プロパティ（fragmentation breakの後にそれ自体で残される行数を設定する）もほぼ同じ頻度で見受けられます。おそらく、人々は両方に同じ値を設定しているのでしょう。

## ページングされたメディア

Paged Mediaを扱うための全体の仕様があり、印刷用のCSSもある。しかし、これはブラウザにうまく実装されていません。これらの機能の優れた実装を見つけるには、印刷に特化したユーザーエージェントを使用する必要があります。

ブラウザは、 `@page` ルールとその擬似クラスをいくらかサポートしており、開発者はこれらを使って、最初のページと見開きの左右のページに対して異なるページプロパティを設定しているのを見かけました。

| 擬似クラス | デスクトップ | モバイル |
|---|---|---|
| `:first` | 5,950 | 7,352 |
| `:right` | 1,548 | 2,115 |
| `:left` | 1,554 | 2,101 |

*図1.71. `@page` の拡散擬似クラスを使って見つかったページの数。*

これらの擬似クラスを使っている人のうち、もっとも多くの人が使っていたのは、ページの余白や大きさを設定することでした。

## メタ

このセクションでは、CSSの使用に関する一般的な情報、たとえば宣言が繰り返される頻度や、CSSの一般的な間違いについてまとめています。

### 宣言の繰り返し

2020年と2021年には、「宣言の繰り返し」の多さを調べる分析が行われました。これは、同じプロパティと値を使用する宣言の数を調べることで、スタイルシートがどれだけ効率的かを特定することを目的としています。

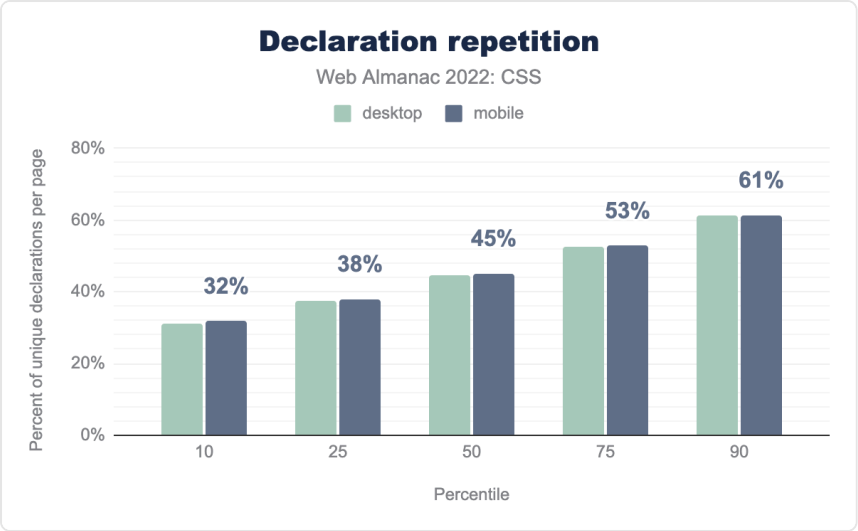



*図1.72. 繰り返しの分布。*

2021年には、繰り返しがわずかに減少したと報告されましたが、今年はわずかに増加しています。したがって、この指標は前年比ではかなり安定していると思われます。

## ショートハンドとロングハンド

CSSでは、ショートハンドプロパティとは、1つの宣言で複数のロングハンドプロパティを設定できるものを指します。たとえば、`background`というショートハンドプロパティは、以下のすべてのロングハンドプロパティを設定するために使うことができる。

- `background-attachment`
- `background-clip`
- `background-color`
- `background-image`
- `background-origin`
- `background-position`
- `background-repeat`
- `background-size`

開発者がスタイルシートの中で `background` のようなショートハンドのプロパティと `background-size` のようなロングハンドのプロパティを混在させる場合、ロングハンドをショートハンドの後に置くのが常に最善とされています。私たちは、どのロングハンドがもっとも一般的であるかを確認するために、この事例を調べました。

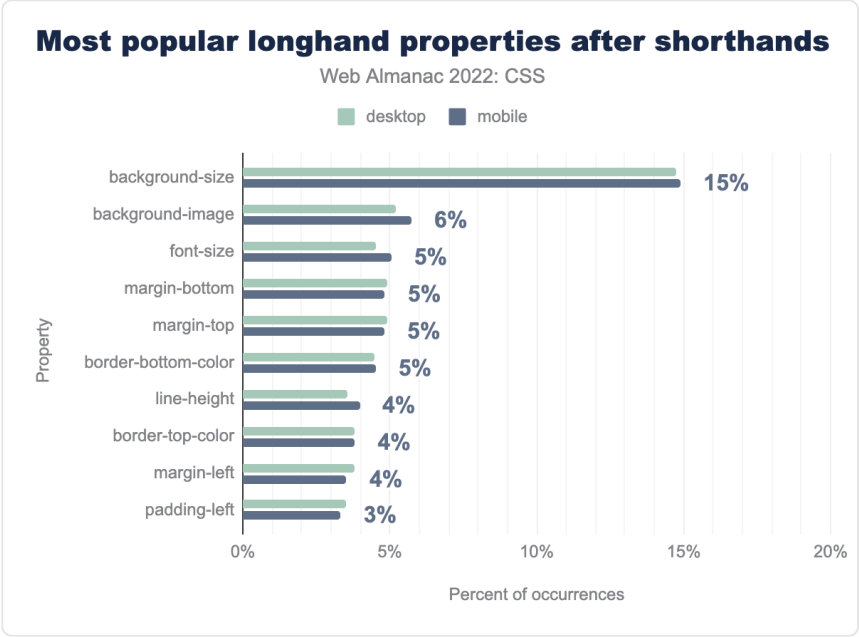



*図1.73. ショートハンドの次に来るもっとも普及しているロングハンドプロパティです。*

2020年、2021年と同様、 `background-size` がもっとも多く、2021年との差はほとんど見られない。

## 回復不能な構文エラー

例年通り、CSSの解析にはRework[15]エンジンを使用しています。回復不可能なエラーとは、エラーの程度がひどく、スタイルシート全体をReworkで解析することができないものです。昨年は、デスクトップページの0.94%、モバイルページの0.55%に回復不可能なエラーが含まれていました。今年は、デスクトップページの13%、モバイルページの12%にそのようなエラーが見られました。これは大きなジャンプのように見えますが、いくつかの方法の変更（サイズの閾値の追加）により、すべての事例が回復不可能なエラーであるとは限らないようです。

15. *https://github.com/reworkcss/css*

# 存在しないプロパティ

例年通り、有効な構文でありながら、実際には存在しないプロパティを参照している宣言をチェックしました。これには、スペルミスやベンダープレフィックスの間違い、開発者が勝手に作ったものなどが含まれます。

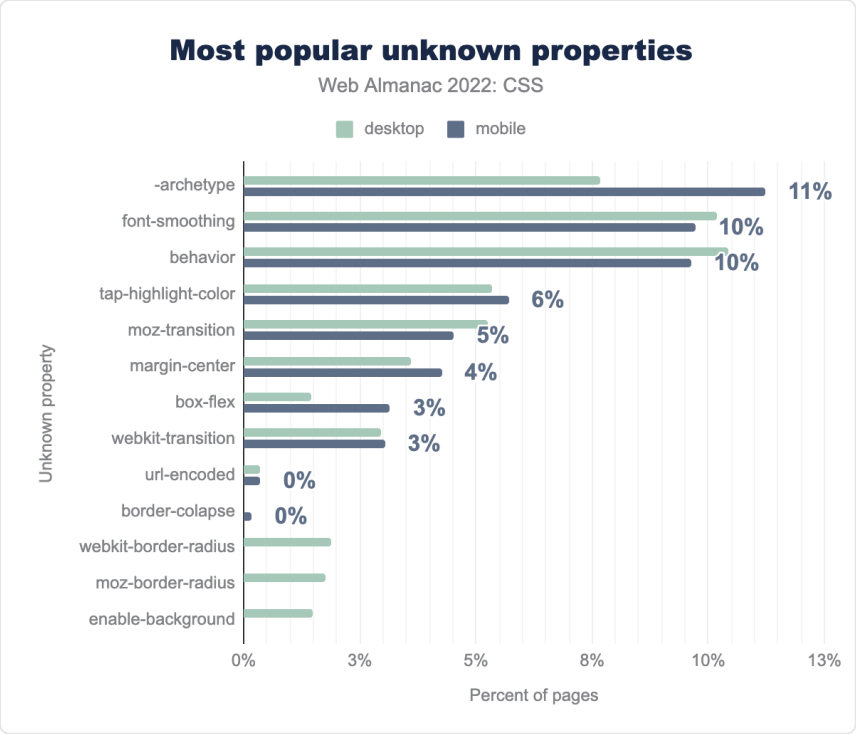



*図1.74. もっともよく見かける未知のプロパティ。*

謎のプロパティのトップは `-archetype` で、存在しないプロパティを持つスタイルシートのケースの11%で出現するようになりました。このプロパティは、昨年の4%から11%に急上昇し、トップの座を獲得しています。2番目のプロパティは `font-smoothing` で、昨年より4%ポイント低下している。これは実際には存在しない `-webkit-font-smoothing` の接頭辞なしバージョンであるように見えます。不正な `webkit-transition`（本来は `-webkit-transition` であるべき）の使用は、14%から3% に減少しました。このことから、おそらくフレームワークなどのサードパーティを介して大量のスタイルシートに入り込んでいたものと思われますが、その後アップデートで問題が修正されました。

# 結論

CSSは急速な進化を続けていますが、主要なエンジンに数年前から搭載されている機能であっても、新しい機能の採用はかなり遅いことがデータから読み取れます。この記事を書いている時点では、コンテナクエリなど、要望の高い機能がいくつかブラウザに搭載されています。これらの機能の採用が需要に見合うかどうか、興味深いところです。

このデータで明らかになったことは、人気のあるプラットフォーム、とくにWordPressが利用統計にどれだけ影響を与えるかということです。WordPressのクラス名やカスタムプロパティ名はデータではっきりと確認できますが、WordPressサイトの大部分に追加されたクラスで使用されているプロパティや値は見えにくくなっています。WordPressが新しい機能を採用した場合、これらの標準クラスの一部として、使用率が急激に上昇することが予想されます。

昨年の結論で述べたように、このデータは、新しい機能（グリッドレイアウトなど）やベストプラクティス（物理プロパティではなく論理プロパティの使用など）が徐々に、そして着実に採用されていることを物語っています。今後数年間、このような変化がどのように展開されるかを楽しみにしています。

### 著者

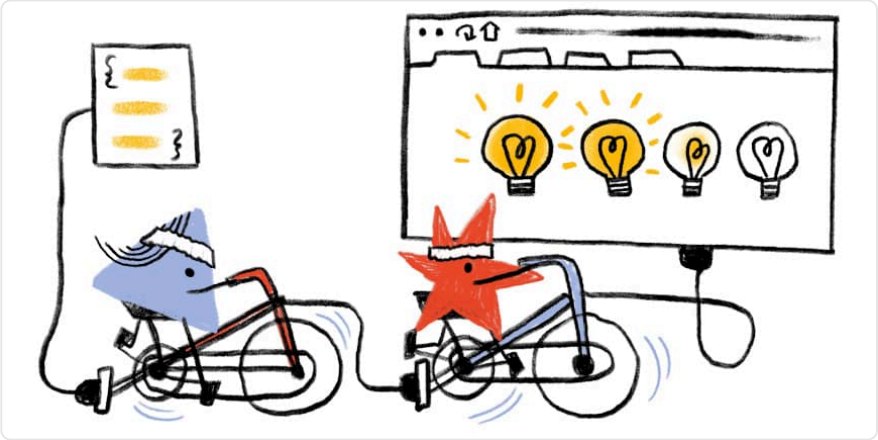
## Rachel Andrew

@rachelandrew rachelandrew https://rachelandrew.co.uk

Rachel Andrew はテクニカルライターとして Google に勤務し、web.dev[16] や Chrome Developers site[17] に携わっています。フロントエンドおよびバックエンドのウェブ開発者、著者、講演者であり、The New CSS Layout[18]など22冊の本の著者または共著者、オンとオフラインの両方で数多くの出版物の常連寄稿者でもあります。CSSワーキンググループのメンバーであるレイチェルは、Twitterで@rachelandrewとして愛猫の写真を投稿しているのを見ることができます。

16. *https://web.dev*
17. *https://developer.chrome.com/*
18. *https://abookapart.com/products/the-new-css-layout*

# 部 I 章 2
# JavaScript



***Jeremy Wagner によって書かれた。***
*Minko Gechev、 Pankaj Parkar、 Nishu Goel、 Houssein Djirdeh、 Kevin Farrugia と Barry Pollard によってレビュ。*
*Nishu Goel と Kevin Farrugia による分析。*
*Abel Mathew と Rick Viscomi 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

JavaScriptは、ウェブ上のインタラクティブ機能の大部分を提供する強力な力です。単純なものから複雑なものまで、さまざまな動作を実現し、Web上でかつてないほど多くのことを可能にしています。

しかし、リッチなユーザー体験を提供するためにJavaScriptを使用する機会が増えたことで、代償を払うことになりました。JavaScriptがダウンロードされ、解析され、コンパイルされた瞬間から実行されるコードのすべての行まで、ブラウザはすべてを可能にする為あらゆる種類の作業をオーケストレーションする必要があります。JavaScriptの使用量が少なすぎると、ユーザー体験やビジネス上の目標を達成できない可能性があります。一方JavaScriptを使いすぎると、読み込みが遅く、反応が鈍く、ユーザーをイライラさせるようなユーザー体験を作ることになります。

今年も、WebにおけるJavaScriptの役割について、2022年の調査結果を発表し、快適なユーザー体験を実現するためのアドバイスを提供します。

## JavaScriptの負荷はどのくらい？

まず始めに、ウェブ開発者がウェブ上で提供しているJavaScriptの量を評価します。結局のところ、改善を行う前に、現在の状況の評価を行う必要があるのです。

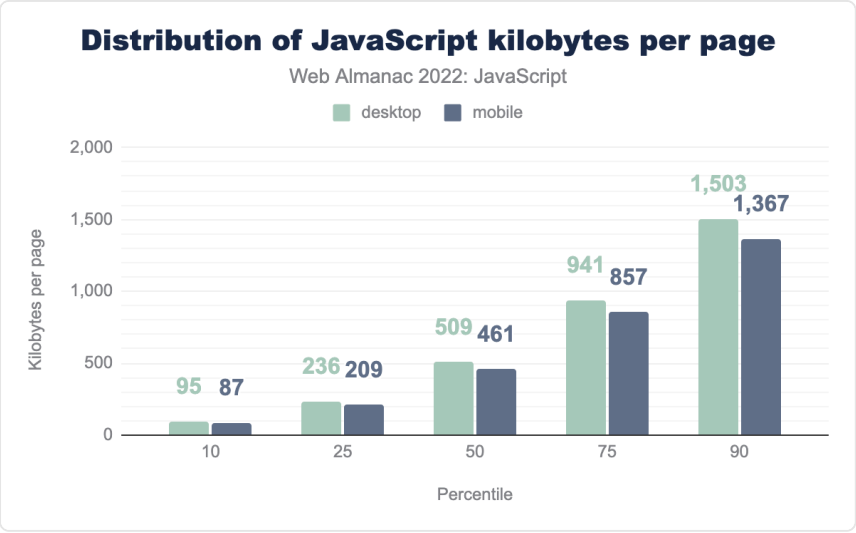



*図2.1. 1ページあたりに読み込まれるJavaScriptの量の分布。*

昨年に引き続き、今年もブラウザに搭載されるJavaScriptの量が増えています。2021年[19]から2022年にかけて、モバイル端末では8%の増加、一方、デスクトップ端末では10%の増加が確認されています。この増加幅は例年より少ないものの、それでも懸念される傾向が続いています。デバイスの性能は向上し続けていますが、すべての人が最新のデバイスを使用しているわけではありません。JavaScriptが増えれば増えるほど、端末のリソースに負担がかかるのは事実です。

19. *https://almanac.httparchive.org/ja/2021/javascript#JavaScriptをどれだけ読み込むか？*

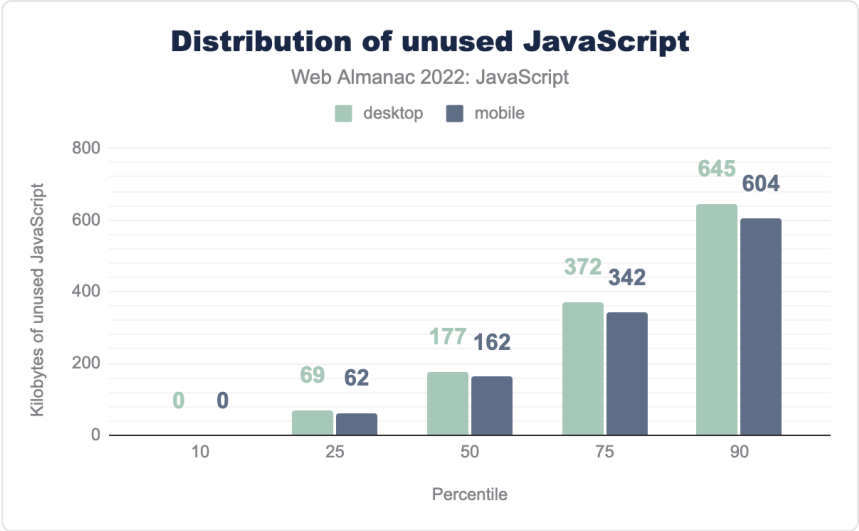



*図2.2. JavaScriptの未使用バイト数の分布。*

Lighthouse[20] によると、中央のモバイル ページでは、162KBの未使用のJavaScriptがロードされているそうです。90パーセンタイルでは604KBのJavaScriptが、未使用であることがわかります。これは、未使用のJavaScriptの中央値と90パーセンタイルがそれぞれ155KBと598KBであった昨年からわずかに上昇したものです。とくにこの分析ではJavaScriptリソースの転送サイズを追跡しているため圧縮されている場合は、使用されたJavaScriptの解凍部分が、このグラフが示すよりもはるかに大きい可能性があることを考えると、この量は未使用のJavaScriptに相当します。

モバイルページで読み込まれた総バイト数の中央値と比較すると、未使用のJavaScriptは、読み込まれたスクリプト全体の35%を占めています。これは、昨年の36%からわずかに減少していますが、それでも未使用で読み込まれるバイトの大部分であることに変わりはありません。このことは多くのページが、現在のページでは使われないかもしれない、あるいはページのライフサイクルの後半でのインタラクションによって引き起こされるスクリプトを読み込んでおり、起動コストを減らすためにダイナミックな`import()`が有効かもしれないことを示唆しています。

## 1ページあたりのJavaScriptリクエスト数

ページ上のすべてのリソースは、少なくとも1つのリクエストを開始し、リソースがより多くのリソースに対して追加のリクエストを行う場合は、より多くのリクエストを開始する可

20. https://web.dev/unused-javascript/

能性があります。

スクリプトのリクエストは多ければ多いほど、より多くのJavaScriptを読み込むだけでなくスクリプトリソース間の競合が増え、メインスレッドを停滞させ起動が、遅くなる可能性が高くなるのです。

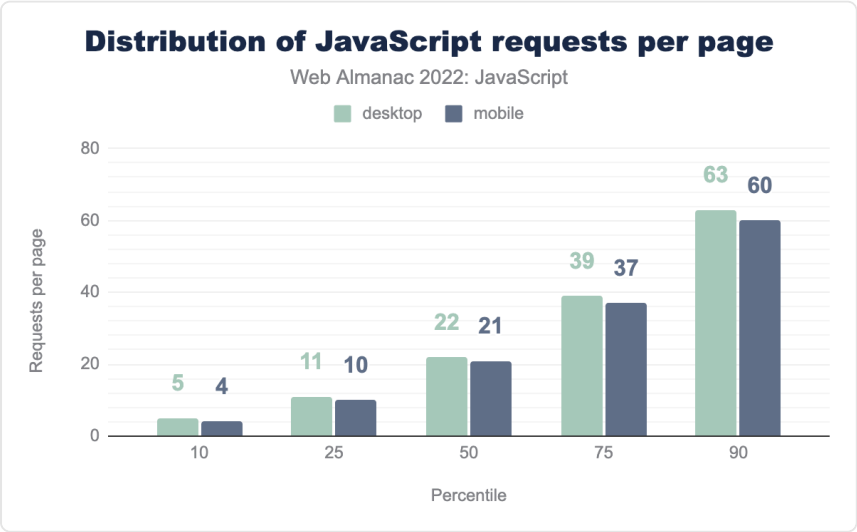



*図2.3. 1ページあたりのJavaScriptリクエスト数の分布。*

2022年、モバイルページの中央値は21のJavaScriptリクエストに応答したのに対し、90パーセンタイルでは60のリクエストに応答しています。昨年と比較すると、中央値で1リクエスト、90パーセンタイルで4リクエストの増加です。

2022年のデスクトップデバイスでは、JavaScriptのリクエスト数が中央値で22、90パーセンタイルで63になっています。昨年と比較すると、中央値で1件、90パーセンタイルで4件増加しており、モバイルデバイスと同じ増加率になっています。

リクエスト数の増加幅は大きくないものの、2019年のWeb Almanac開始以来、前年比でリクエスト増加の傾向が続いています。

## JavaScriptはどのように処理されるのですか？

Node.jsのようなJavaScriptランタイムの出現以来、JavaScriptをバンドルし変換するためビルドツールへ依存することがますます一般的になってきました。これらのツールは紛れもなく便利ですが、JavaScriptの提供量に影響を与える可能性があります。今年のWeb Almanac

では、新たにバンドルとトランスパイラの使用状況に関するデータを提示します。

## バンドル

JavaScriptバンドルは、プロジェクトのJavaScriptソースコードを処理し、変換や最適化を適用するビルドタイムツールです。出力はプロダクションレディのJavaScriptです。たとえば、次のようなコードです。

```
function sum (a, b) {
  return a + b;
}
```

バンドルはこのコードをより小さく、しかしより最適化された同等品に変換し、ブラウザがダウンロードする時間をより短くします。

```
function n(n,r){return n+r}
```

バンドルは、ソースコードを最適化し、実稼働環境でのパフォーマンスを向上させるための重要な役割を担っています。

JavaScriptのバンドルに関しては豊富な選択肢がありますが、よく頭に浮かぶのはwebpack[21]でしょう。幸いなことに、webpackの生成するJavaScriptにはいくつかのシグネチャ（たとえば `webpackJsonp`）が含まれており、ウェブサイトのプロダクションJavaScriptがwebpackを使ってバンドルされているかどうかを検出することが可能です。

21. *https://webpack.js.org/*

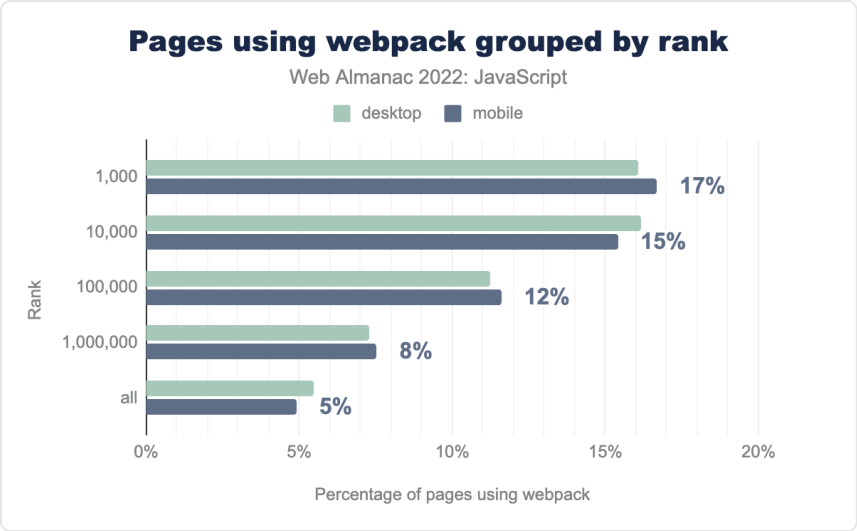



*図2.4. webpackでバンドルされたJavaScriptを使用しているページの順位。*

もっとも普及している1,000のWebサイトのうち、17%がバンドルとしてwebpackを使用しています。HTTP Archiveがクロールするもっとも多いページの多くは、ソースコードのバンドルと最適化にwebpackを使用している知名度の高いeコマースサイトである可能性が高いため、これは理にかなった結果です。それでも、HTTP Archiveのデータセットに含まれる全ページの5%がwebpackを使用しているという事実は、重要な統計データです。しかし、使用されているバンドルはwebpackだけではありません。

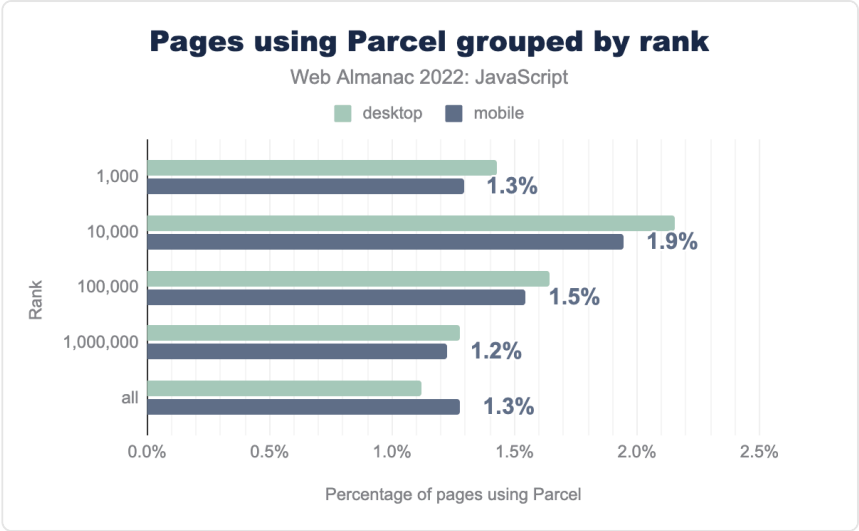



*図2.5. Parcelでバンドルされた JavaScript を使用しているページの順位。*

Parcel[22]は、webpackの代替品として注目され、その採用実績は大きいです。Parcelの採用率はすべてのランクで一貫しており、ランク間で1.2%から1.9%の幅を占めている。

HTTP Archiveはエコシステム内のすべてのバンドラーの使用状況を追跡することはできませんが、バンドラーの使用状況はJavaScriptの全体像において開発者の体験にとって重要であるだけでなく、依存性管理コードという形で貢献できるオーバーヘッドがJavaScriptの提供量の要因になりうる、という点で重要です。ユーザーが使用するブラウザに対してもっとも効率的に出力できるよう、プロジェクト全体の設定がどのように行われているかを確認することは価値があります。

## トランスパイラ

トランスパイラとは、新しいJavaScriptの機能を古いブラウザで実行できる構文に変換するため、ビルド時にツールチェーンでよく使われるものです。JavaScriptは長年にわたって急速に進化してきたため、これらのツールは今でも使用されています。今年のWeb Almanacでは、広く互換性のあるプロダクション対応のJavaScriptを提供する際の Babel[23] の使用に関する分析が新たに掲載されました。とくにBabelに注目したのは、開発者コミュニティで代替品よりも広く使われているためです。

22. *https://parceljs.org/*
23. *https://babeljs.io/*

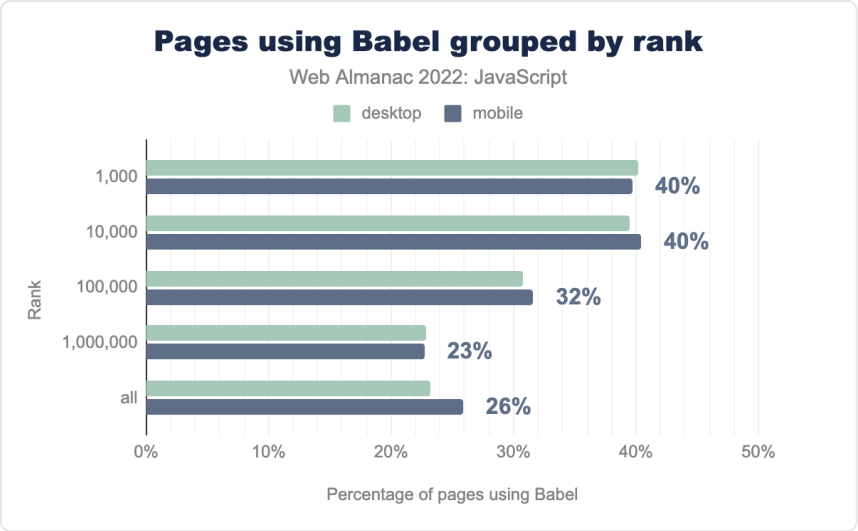



*図2.6. Babelを使用しているページをランク別に紹介。*

この結果は、JavaScriptが長年にわたって進化してきたことを考えれば、驚くような展開ではありません。特定のブラウザのセットに対して幅広い互換性を維持するために、Babelはトランスフォーム[24]を使用して互換性のあるJavaScriptコードを出力しています。

トランスフォームは、トランスフォームしていないものよりも大きくなることがよくあります。トランスフォームが大規模であったり、コードベース全体で重複していたりすると、潜在的に不必要な、あるいは使用されていないJavaScriptがユーザーに出荷される可能性があります。これはパフォーマンスに悪影響を及ぼす可能性があります。

上位100万位にランクインしたページの26%でさえ、Babelを使ってJavaScriptのソースコードをトランスフォームしていることを考えると、これらの体験の中には必要のないトランスフォームを提供している可能性もないわけではありません。プロジェクトでBabelを使用する場合、Babelの利用可能な 設定オプション[25] とプラグインを慎重に検討し、その出力を最適化する機会を探してください。

Babelは特定の機能をレガシーな構文に変換する必要があるかどうかを判断するため Browserslist[26]に依存しているため、あなたのコードがあなたのユーザーが実際に使用しているブラウザで動作するように変換されるよう、Browserslist設定も必ずレビューしてください。

24. *https://babeljs.io/docs/en/babel-plugin-transform-runtime#why*
25. *https://babeljs.io/docs/en/options*
26. *https://github.com/browserslist/browserslist*

# JavaScriptはどのようにリクエストされるのですか？

JavaScriptを要求する方法は、パフォーマンスにも影響を与える可能性があります。JavaScriptをリクエストする方法には最適なものがありますが、場合によっては最適とは言い難い方法もあります。ここでは、ウェブが全体としてどのようにJavaScriptを提供しているか、そしてそれがパフォーマンスへの期待にどのように合致するかを見ていきます。

## `async`、`defer`、`module`と`nomodule`

HTMLの `<script>` 要素にある `async` と `defer` 属性は、スクリプトの読み込みに関する挙動を制御できます。`async` 属性はスクリプトが解析の妨げになることを防ぎますが、ダウンロードされるとすぐに実行されるため、レンダリングをブロックする可能性があります。`defer` 属性はDOMの準備が整うまでスクリプトの実行を遅らせますので、パースとレンダリングの両方をブロックすることはありません。

`type="module"` と `nomodule` 属性は、ブラウザに提供されるES6モジュールの有無にとくに関係します。`type="module"` が使用された場合、ブラウザはそれらのスクリプトのコンテンツがES6モジュールを含むことを予期し、デフォルトでDOMが構築されるまでそれらのスクリプトの実行を延期します。反対の `nomodule` 属性は、現在のスクリプトがES6モジュールを使用していないことをブラウザに示します。

| *機能* | *デスクトップ* | *モバイル* |
|---|---|---|
| *async* | *76%* | *76%* |
| *defer* | *42%* | *42%* |
| *asyncとdefer* | *28%* | *29%* |
| *module* | *4%* | *4%* |
| *nomodule* | *0%* | *0%* |

*図2.7. `<script>` 要素で `async`、`defer`、`type="module"`、`nomodule` 属性を使用しているページの割合です。*

76% のモバイル ページが `async` を使用してスクリプトを読み込んでいることは、開発者がレンダー ブロッキングの影響を認識していることを示すものであり、励みになります。しかし、`defer` の使用率がこれほど低いということは、レンダリングパフォーマンスを向上させる機会が残されていることを示唆しています。

昨年[27]で述べたように、`async`と`defer`の両方を使うことはアンチパターンで、`defer`部分は無視されて`async`が優先されるので避けるべきものです。

一般的に`type="module"`と`nomodule`がないのは、JavaScriptモジュールを提供しているページがほとんどないためです。時間が経つにつれて、開発者がブラウザに変換されていないJavaScriptモジュールを提供するため、とくに`type="module"`の使用は増加する可能性があります。

全サイトのスクリプト全体の割合で見ると、少し違った見方ができる。

| 機能 | デスクトップ | モバイル |
|---|---|---|
| *async* | *49.3%* | *47.2%* |
| *defer* | *8.8%* | *9.1%* |
| *asyncとdefer* | *3.0%* | *3.1%* |
| *module* | *0.4%* | *0.4%* |
| *nomodule* | *0%* | *0%* |

*図2.8. スクリプトの`<script>`要素で`async`、`defer`、`type="module"`、`nomodule`属性を使用している割合です。*

ここでは、`async`と`defer`の両方があまり使用されていないことがわかります。これらのスクリプトの一部は、最初のレンダリングの後に動的に挿入されるかもしれません。しかし、かなりの割合のページが、最初のHTMLに含まれる多くのスクリプトにこれらの属性を設定せず、レンダリングを遅らせている可能性もあります。

## `preload`、`prefetch`と`modulepreload`

`preload`、`prefetch`、`modulepreload`などのリソースヒントは、どのリソースを早期に取得すべきかをブラウザへ示唆するのに有用です。それぞれのヒントは異なる目的を持っており、`preload`は現在のナビゲーションに必要なリソースを取得するために使用され、`modulepreload`はJavaScriptモジュール[28]を含むスクリプトを`preload`するために相当し、`prefetch`は次のナビゲーションで必要なリソースに使用されます。

27. *https://almanac.httparchive.org/ja/2021/javascript#asyncとdefer*
28. *https://developer.mozilla.org/docs/Web/JavaScript/Guide/Modules*

| リソースヒント | デスクトップ | モバイル |
|---|---|---|
| preload | 16.4% | 15.4% |
| prefetch | 1.0% | 0.8% |
| modulepreload | 0.1% | 0.1% |

*図2.9. 各種リソースヒントを使用しているページの割合。*

リソースヒントの導入動向を分析するのは難しい。すべてのページがリソースヒントの恩恵を受けるわけではありませんし、リソースヒントを広く使うことを一律に推奨するのは、使い過ぎるとそれなりの結果を招くので賢明ではありません。とくに `preload` が関係している。しかし、モバイルページの15%に `preload` のヒントが比較的多く見られることから、多くの開発者がこのパフォーマンスの最適化を認識し、それを利用しようとしていることがうかがえます。

`prefetch` は、長い複数ページのセッションには有益ですが、使用するにはトリッキーです。それでも、 `prefetch` は完全に推測的で、ある種の条件下ではブラウザが無視することもあるほどです。これは、あるページが未使用のリソースをリクエストすることによって、データを浪費する可能性があることを意味します。それは本当に「場合による」のです。

`modulepreload` が使われていないのは、 `<script>` 要素に `type="module"` 属性が採用されていないのと同じで、理にかなっています。それでも、トランスフォームなしでJavaScriptモジュールを配布しているアプリは、このリソースヒントの恩恵を受けることができます。なぜなら、指定されたリソースだけを取得するのではなく、モジュールツリー全体を取得することができるからです。これは特定の状況で役に立つかもしれません。

では、各リソースヒントの種類はいくつ使用されているのか、分析を掘り下げてみましょう。

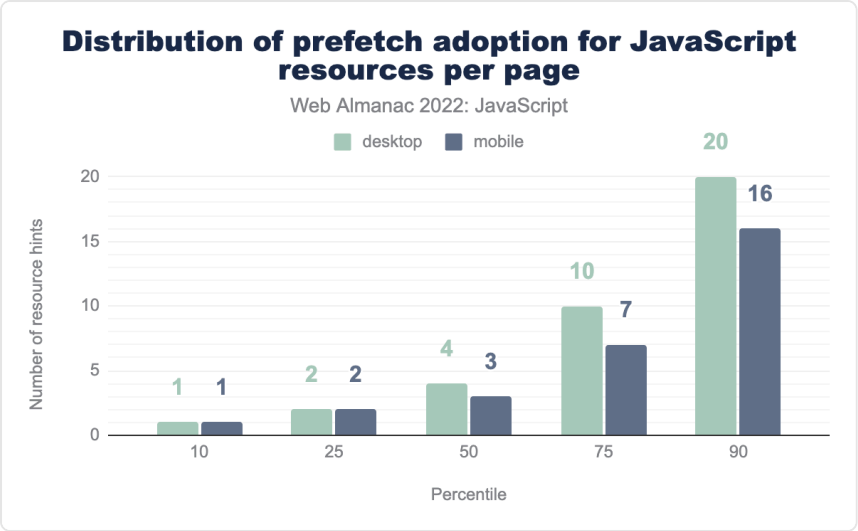



*図2.10. ページごとのJavaScriptリソースに対する `prefetch` 採用の分布。*

ここでの `prefetch` の採用はやや意外で、1ページあたりJavaScriptリソースに対して3つの `prefetch` ヒントを出しています。しかし、75パーセンタイルと90パーセンタイルにあるこれらのヒントの数は、発生しないページ移動のための未使用リソースという形で、かなりの量のムダがあることを示唆しています。

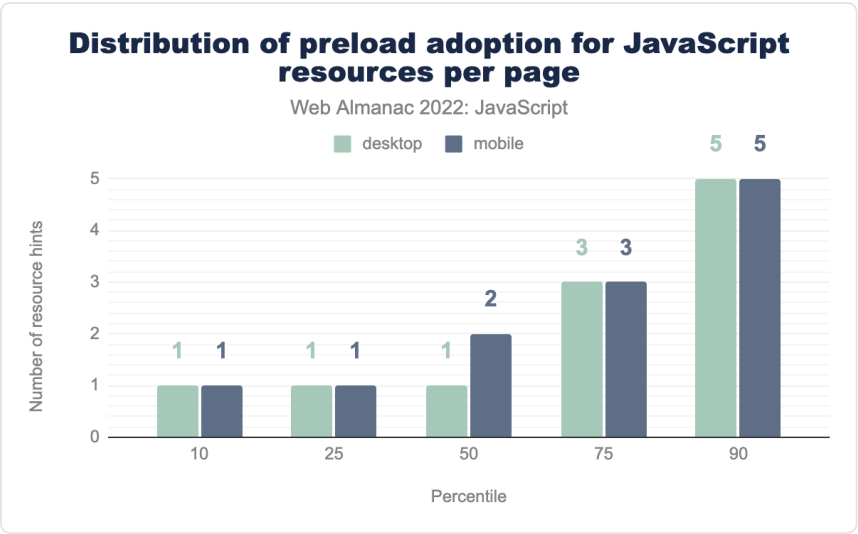



*図2.11. ページごとのJavaScriptリソースに対する `preload` 採用の分布。*

この分析では、1つ以上の `preload` ヒントを使用しているページで、JavaScriptリソースにいくつのリソースヒントが使用されたかを追跡しています。中央のページはJavaScriptに対して2つの `preload` ヒントを配信しており、表面上は悪くありませんがスクリプトのサイズ、スクリプトが起動する処理量、あるいは `preload` で取得したスクリプトが最初のページロードに必要かどうかによって、しばしば異なります。

残念ながら、90パーセンタイルのJavaScriptリソースには5つの `preload` ヒントがあり、これは多すぎるかもしれません。これは、90%台のページがとくにJavaScriptに依存しており、その結果生じるパフォーマンスの問題を克服しようとして `preload` を使用していることを示唆しています。

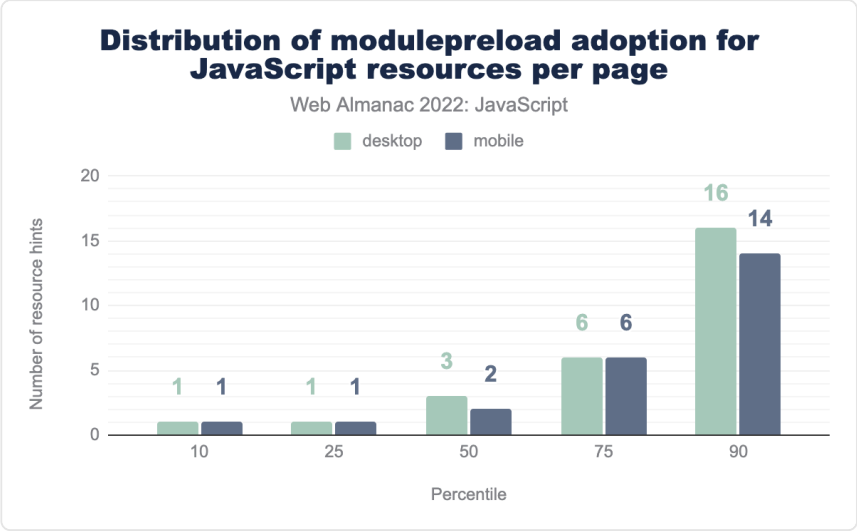



*図2.12. ページごとのJavaScriptリソースに対する `modulepreload` 採用の分布。*

`modulepreload` では、75パーセンタイルで6個、90パーセンタイルで14個という驚異的な数のヒントが表示されました。このことから、上位のパーセンタイルで1つ以上の `modulepreload` ヒントを使用しているページでは、変換されていないES6モジュールを直接ブラウザに送信している一方で、非常に多くのリソースヒントが必要なため、上位の範囲ではJavaScriptに過度に依存していることが示唆されます。

リソースヒントはブラウザにリソースを読み込む方法を最適化するための素晴らしいツールですが、使用する際のアドバイスがひとつあるとすれば、控えめに、そして最初は発見できないようなリソースに使用することです。たとえば、最初にDOMへ読み込んだJavaScriptファイルが、別のファイルを要求する場合などです。スクリプトを大量にプリロードするよりも、出荷するJavaScriptの量を減らすようにしましょう。そうすれば、より良いユーザー体験につながります。

## JavaScript を <head> 内に記述する

古くからよく言われているパフォーマンスのためのベストプラクティスは、JavaScriptをドキュメントのフッターにロードしてスクリプトのレンダーブロックを回避し、スクリプトが実行される前にDOMが構築されるようにすることでした。しかし最近では、ある種のアーキテクチャでは `<script>` 要素をドキュメントの `<head>` に配置することがより一般的になってきています。

これはウェブアプリケーションでJavaScriptの読み込みを優先させる良い方法ですが、DOMのレンダーブロックを避けるために、可能な限り `async` と `defer` 属性を使用すべきです。レンダーブロッキングとは、ページが依存しているリソースを処理するために、ブラウザがページのすべてのレンダリングを停止しなければならないことを指します。これは、ふらちなコンテンツ[29]のような不快な効果や、DOMの準備に依存するスクリプトのためにDOMの準備ができていないときに発生しうるJavaScript実行時エラーを避けるために行われます。


*図2.13. ドキュメント <head> 内にレンダーブロッキングスクリプトがあるモバイルページの割合です。*

モバイルページの77%が `<head>` 内にレンダリングをブロックするスクリプトを少なくとも1つ含んでいるのに対し、デスクトップページの79%はこのようなスクリプトを含んでいることがわかりました。スクリプトがレンダリングをブロックすると、ページコンテンツが迅速に描画されなくなるため、これは懸念すべき傾向です。

29. *https://en.wikipedia.org/wiki/Flash_of_unstyled_content*

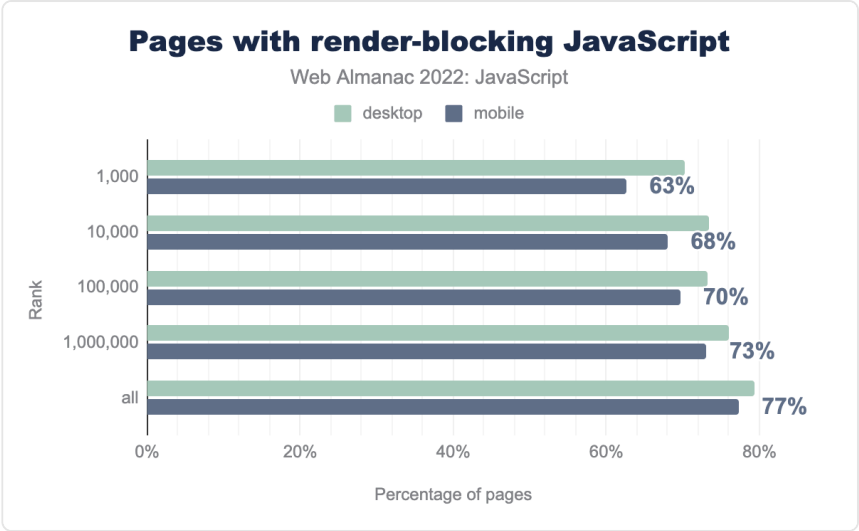



*図2.14. ドキュメント `<head>` にレンダリングをブロックするスクリプトがあるランク別ページ。*

ランキングされたページでこの問題を見ると、同じように厄介なパターンが見られます。とくに、モバイル端末からアクセスされた上位1,000サイトのうち63%は、`<head>` 内に少なくとも1つのレンダーブロッキングスクリプトを含んでおり、その割合は順位が上がるにつれて増加します。

これには解決策があります。`defer` を使用するのは比較的安全で、DOMのレンダリングをブロックしないようにします。`async` を使用するのも良い方法ですが、他の `<script>` 要素に依存したスクリプトを作成すると、エラーを発生させる可能性があります。

可能であれば、レンダリングに不可欠なJavaScriptをフッターに配置し、ブラウザがそれらのリソースを要求する際に先手を打てるようにプリロードできます。いずれにせよ、レンダーブロッキングを引き起こすJavaScriptの状態と、私たちが提供するJavaScriptの量は、決して良いものではありません。

## スクリプトインジェクション

スクリプトインジェクションとは、`HTMLScriptElement` を `document.createElement` を使ってJavaScriptで作成し、DOM挿入メソッドでDOMに注入するパターンのことです。また、文字列中の `<script>` 要素のマークアップは、`innerHTML` メソッドによってDOMに注入できます。

スクリプトインジェクションは多くのシナリオで使用されるかなり一般的な手法ですが、こ

の方法の問題は、最初のHTMLペイロードが解析されるときにスクリプトを発見できないようにして、ブラウザのプリロード スキャンナー[30]を破ってしまうということです。これは、注入されたスクリプト リソースが最終的にマークアップのレンダリングを担当する場合、最大のコンテントフルペイント (LCP)[31]などのメトリックに影響を与える可能性があり、マークアップの大きな塊をその場で解析する長いタスクを開始させる可能性があります。

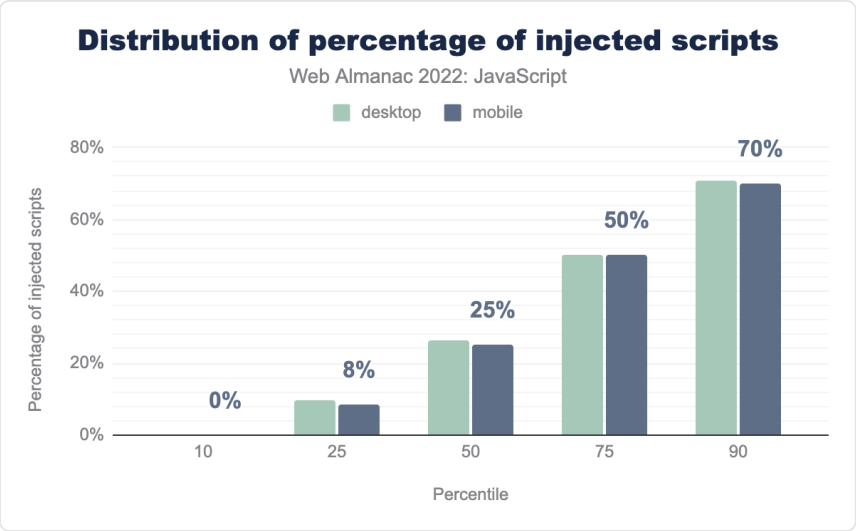



*図2.15. 注入されたスクリプトの割合の各パーセンタイルでの分布。*

中央値では、ページのスクリプトの25%が注入され、最初のHTML応答で発見できるようにするのとは対照的であることがわかります。さらに問題なのは、ページの75パーセンタイルと90パーセンタイルでは、それぞれ50%と70%のスクリプトが注入されていることです。

スクリプト・インジェクションは、ユーザーが消費するページのコンテンツを表示するために使用すると、パフォーマンスに悪影響を与える可能性があります[32]ので、このような場合は必要なときに避けるべきです。スクリプト・インジェクションが今日のウェブでこれほど普及していることは、懸念すべき傾向です。最新のフレームワークやツールはこのパターンに依存している可能性があり、つまり、いくつかのアウトオブボックス体験は、Webサイトの機能を提供するためこの潜在的なアンチパターンに依存している可能性があります。

## ファーストパーティーのJavaScriptとサードパーティーのJavaScript

Webサイトがよく提供するJavaScriptには、2つのカテゴリがあります。

30. https://web.dev/preload-scanner/#injected-async-scripts
31. https://web.dev/i18n/ja/lcp/
32. https://www.igvita.com/2014/05/20/script-injected-async-scripts-considered-harmful/

- ウェブサイトの重要な機能を支え、インタラクティブ性を提供するファーストパーティのスクリプト。
- UX調査、分析、広告収入の提供、動画やソーシャルメディア機能などの埋め込みなど、さまざまな要件を満たす外部ベンダーから提供されるサードパーティスクリプトです。

ファーストパーティのJavaScriptは最適化しやすいかもしれませんが、サードパーティのJavaScriptはそれ自体がパフォーマンス問題の重大な原因となり得ます。サードパーティのベンダーは、顧客のために新たなビジネス機能を提供するための新機能を追加するよりも、そのJavaScriptリソースの最適化を優先させないかもしれないからです。さらに、UX研究者、マーケティング担当者、およびその他の非技術者は、これらのスクリプトが提供する機能や収益源を放棄することをためらうかもしれません。

このセクションでは、ファーストパーティとサードパーティのコードの内訳を分析し、現在のWebサイトがJavaScriptをどこからどのように読み込むかを分割している現状についてコメントします。

## リクエスト

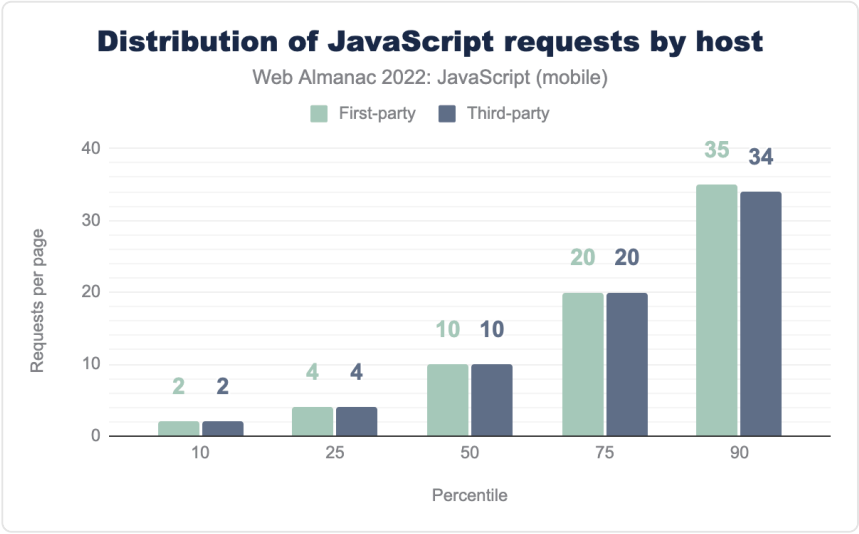



*図2.16. ホスト別のファーストパーティとサードパーティのJavaScriptリクエストの分布。*

ここでは、悲観的な図が示されています。パーセンタイルに関係なく、観測されたすべてのホストは、ファースト・スクリプトとサード・パーティ・スクリプトを同量ずつ配信してい

るようです。中央値のホストは各タイプを10個、75パーセンタイルのホストは各タイプを20個、90パーセンタイルのホストはサードパーティ・スクリプトを34個提供しています。

これは問題であり、心配な傾向です。サードパーティのスクリプトは、パフォーマンスに関してあらゆる種類の損害を与える責任があります。サードパーティのスクリプトは多くのタスクをオーケストレーションする高価なタイマーを実行したり、独自のイベントリスナーを追加してインタラクティブ性を遅延させたり、ビデオやソーシャルメディアのサードパーティの中には提供するサービスを強化するために法外な量のスクリプトを出荷したりするなど、さまざまなことを行うことがあります。

サードパーティ製スクリプトを軽減するための手順は、技術的なものよりも文化的なものであることが多い。サードパーティ製スクリプトを過剰に出荷している場合は、各スクリプトとその実行内容を監査し、アクティビティのプロファイルを作成して、パフォーマンスに関する問題を見つけます。

もしあなたがかなりのUX研究をしているのなら、（オリジンが適切な`Timing-Allow-Origin`ヘッダーを送信する場合）独自のフィールドデータを収集し、サードパーティ製スクリプトが引き起こすパフォーマンスの問題を回避するための情報に基づいた意思決定を検討することをオススメします。サードパーティーのスクリプトを追加するたびに、ロードコストだけでなく、ユーザーの入力に対する応答性が重要なランタイム中のコストも発生します。

### バイト

そこで、ホストがサードパーティ製スクリプトを大量に提供していることが分かったのですが、ファーストとサードパーティ製スクリプトのバイトコストはどうなっているのでしょうか。

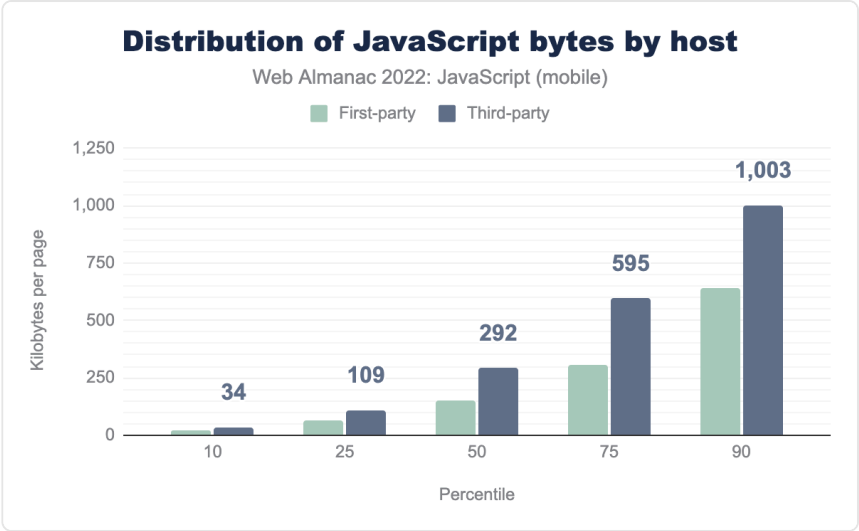



*図2.17. ホスト別のファーストパーティとサードパーティのJavaScriptのバイト数の分布。*

ほぼすべてのパーセンタイルで、サードパーティ製スクリプトの送信バイト量が、ファーストパーティ製スクリプトの送信バイト量を上回っています。75パーセンタイルでは、サードパーティのスクリプトのペイロードは、ファーストパーティのスクリプトの2倍であるように見えます。90パーセンタイルでは、送信されるサードパーティ製スクリプトの量は、ほぼ1メガバイトになるようです。

もし、あなたのウェブサイトのファーストスクリプトとサードパーティスクリプトのペイロードが上記のグラフと同様であることがわかったら、エンジニアリング組織と協力して、この数字を減らす努力をすることが重要です。そうすることで、ユーザーの役に立つことができるのです。

## ダイナミックな`import()`

ダイナミックな`import()`は静的な`import`構文の変形で、スクリプト内のどこでも実行できます。一方静的な`import`式はJavaScriptファイルの先頭で実行しなければならず、他の場所では実行することができません。

ダイナミックな`import()`により、開発者はメインのバンドルからJavaScriptコードのチャンクを効果的に「分割」してオンデマンドでロードすることができ、前もってロードするJavaScriptを少なくして起動時のパフォーマンスを改善できます。


*図2.18. 動的な `import()` を使用しているモバイルページの割合。*

現在観測されているモバイルページの0.34%という驚異的な低さで、デスクトップページの0.41%で動的な `import()` が使用されています。とはいえ、バンドルによっては動的な `import()` 構文をES5互換の代替構文にトランスフォームすることが一般的です。この機能が広く使われている可能性は高いのですが、プロダクションのJavaScriptファイルではあまり使われていないようです。

これは難しいことですが、バランスを取ることができ、それにはユーザーの意図を測ることが必要です。インタラクションを遅らせることなくJavaScriptのロードを延期する1つの方法は、ユーザーがインタラクションを行う意図を示したときにJavaScriptをpreload[33]することです。たとえば、フォームの検証のためにJavaScriptのロードを延期し、ユーザーがそのフォームのフィールドにフォーカスした時点でそのJavaScriptをプリロードできます。そうすれば、JavaScriptが要求されたとき、すでにブラウザーのキャッシュにあるはずです。

また、サービスワーカーのインストール時に、インタラクションに必要なJavaScriptをプリキャストしておくという方法も考えられます。インストールは、ページの `load` イベントでページが完全にロードされた時点で行う必要があります。これにより、必要な機能が要求されたときに、サービスワーカーのキャッシュから起動コストをかけずに取得することができる。

動的な `import()` は使い方が難しいのですが、より広く採用することで、JavaScriptを起動時に読み込むパフォーマンスコストをページのライフサイクルの後半、おそらくネットワークリソースの競合が少ない時点にシフトさせることができます。起動時に読み込まれるJavaScriptの量は増える一方なので、動的な `import()` の採用が増えることを期待しています。

### Web ワーカー

Webワーカー[34]は、DOMへ直接アクセスせずに専用のJavaScriptファイルを独自スレッドで回転させることで、メインスレッドの作業を軽減するウェブプラットフォームの機能です。この技術は、メインスレッドに負担をかけるような作業を別スレッドで、行うことで負荷を軽減するために利用されます。

33. *https://developer.mozilla.org/docs/Web/HTML/Link_types/preload*
34. *https://developer.mozilla.org/docs/Web/API/Web_Workers_API/Using_web_workers*

**12%**

*図2.19. Web Workerを利用したモバイルページ数。*

現在、モバイルおよびデスクトップページの12% が、ユーザー体験を悪化させる可能性のあるメインスレッドの作業を軽減するために、1つ以上のウェブワーカーを使用していることは喜ばしいことですが、改善の余地は多くあります。

DOMに直接アクセスしなくてもできる重要な作業がある場合は、Web Workerを使用するのがよいアイデアです。Web Workerとのデータ転送には専用の通信パイプライン[35]を使う必要がありますが、この技術を使うことで、ユーザーの入力に対するWebページの応答性を格段に向上させることは十分可能です。

しかし、その通信パイプラインは、設定や使い方にコツがいるものですが、このプロセスを簡略化できるオープンソースのソリューションがあります。comlink[36]はこれを支援するライブラリの1つで、Webワーカーに関する開発者の体験をより楽しいものにできます。

ウェブワーカーを自前で管理するかライブラリで管理するかは別として、ポイントは高価な作業を行う場合、それがメインスレッドで行われる必要があるかどうかを判断し、そうでなければウェブサイトのユーザー体験を可能な限り良くするためにウェブワーカーの使用を強く検討する、ということです。

## ワークレット

ワークレットは、ペイントやオーディオ処理などのタスクのためのレンダリングパイプラインに低レベルでアクセスできる特殊なワーカーの一種です。4種類のワークレットがありますが、現在利用可能なブラウザに実装されているのは、ペイントワークレット[37]とオーディオワークレット[38]の2種類のみです。ワークレットは独自のスレッドで実行されるため、メインスレッドで高価な描画やオーディオ処理から解放されるという明確なパフォーマンス上の利点があります。

**0.0013%**

*図2.20. ペイントワークレットが1つ以上登録されているモバイルページの割合です。*

35. *https://developer.mozilla.org/docs/Web/API/Web_Workers_API/Using_web_workers#%E3%83%AF%E3%83%BC%E3%82%AB%E3%83%BC%E3%81%A8%E3%81%AE%E3%83%87%E3%83%BC%E3%82%BF%E8%BB%A2%E9%80%81%E3%81%A*
36. *https://www.npmjs.com/package/comlink*
37. *https://caniuse.com/mdn-api_css_paintworklet*
38. *https://caniuse.com/mdn-api_audioworklet*

ワークレットはニッチな技術なので、あまり使われていないのも無理はありません。ペイントワークレットは、ジェネレーティブアートワークの高価な処理を別のスレッドにオフロードする優れた方法であり、ユーザー体験にちょっとしたセンスを加えるのに最適な技術であることは言うまでもありません。100万件のWebサイトのうち、ペイントワークレットを使用しているのはわずか13件です。




*図2.21. 音声ワークレットが1つ以上登録されているモバイルページの割合です。*

音声ワークレットの採用率はさらに低く、100万件に4件の割合で採用されているに過ぎません。これらの技術の採用が時間とともにどのように推移するかは興味深いところです。

## JavaScriptはどのように配信されるのですか？

JavaScriptのパフォーマンスで同様に重要なのは、ブラウザにスクリプトを配信する方法です。これには、JavaScriptを圧縮する方法から始まる、一般的でありながら時に見過ごされる最適化の機会がいくつか含まれています。

### 圧縮

圧縮はHTML、CSS、SVG画像、そしてJavaScriptなど、主にテキストベースの資産に適用される、よく使用される技術です。ウェブで広く使われているさまざまな圧縮技術があり、ブラウザへのスクリプトの配信を高速化し、リソースの読み込み段階を効果的に短縮できます。

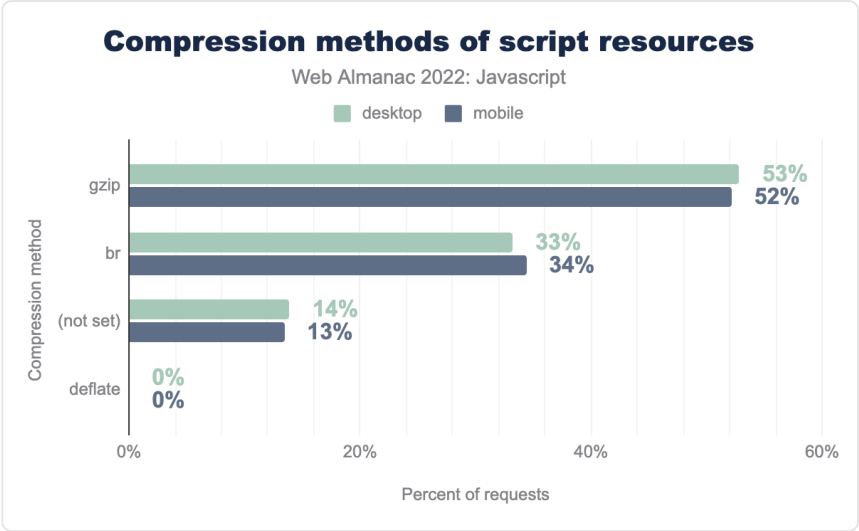



*図2.22. JavaScriptのメソッド別圧縮。*

スクリプトの転送サイズを減らすために使える圧縮技術はいくつかありますが、Brotli[39] (br) 方式がもっとも有効[40]だと言われています。Brotliの モダンブラウザでの優れたサポート[41]にもかかわらず、gzip[42]がもっとも好ましい圧縮方法であることは明らかでしょう。これは、多くのウェブサーバーがデフォルトとして使用しているためと思われます。

何かがデフォルトである場合、そのデフォルトは、より良いパフォーマンスのためにチューニングされるのではなく、そのままの状態で維持されることがあります。Brotliでスクリプトを圧縮しているページが34%しかないことを考えると、スクリプトリソースの読み込み性能を向上させる機会があることは明らかですが、昨年の30%に比べて改善されていることも特筆すべきことでしょう。

39. https://github.com/google/brotli
40. https://www.smashingmagazine.com/2016/10/next-generation-server-compression-with-brotli/
41. https://caniuse.com/brotli
42. https://www.gzip.org/

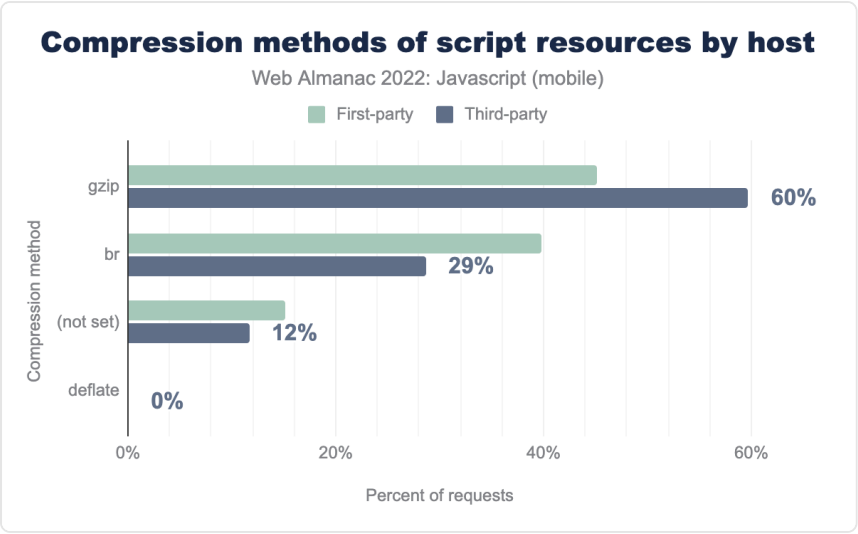



*図2.23. ホスト別のスクリプトリソースの圧縮方法。*

この問題は、サードパーティのスクリプトプロバイダーによってさらに悪化し、それぞれ60%対29%と、依然としてBrotliよりも広くgzip圧縮を導入しています。サードパーティーのJavaScriptが今日のWebにおける深刻なパフォーマンスの問題であることを考えると、これらのリソースのロード時間は、代わりにBrotliを使用してサードパーティーのリソースを展開することで短縮できる可能性があります。

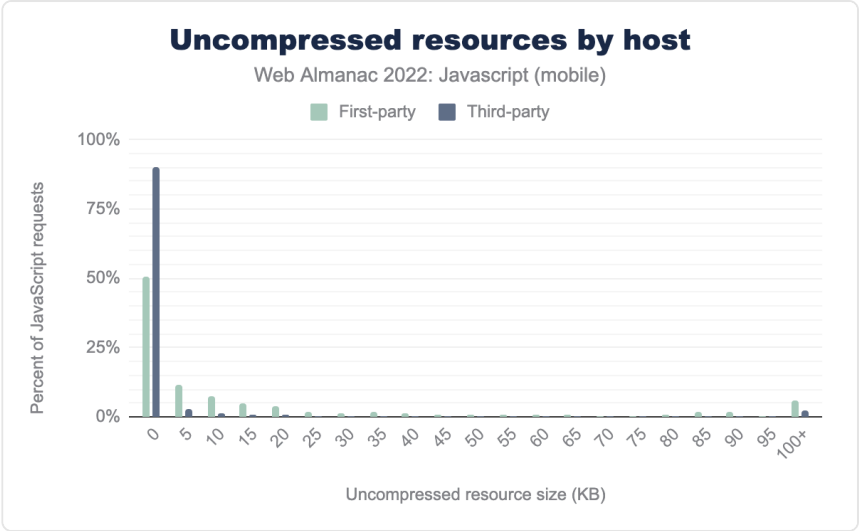



*図2.24. 非圧縮のリソースをサイズ別に紹介。*

ありがたいことに、圧縮せずに配信されるのはもっとも小さなリソース、とくにペイロードが5KB未満のサードパーティ製スクリプトだけであることが分かっています。これは小さいリソースに圧縮を適用した場合、圧縮の収穫が少なくなるためで、実際、動的圧縮のオーバーヘッドが追加されると、リソースの配信を遅延させる可能性があります。残念ながら、ペイロードが100KBを超えるファーストパーティのスクリプトなど、より大きなリソースを圧縮する機会が、あらゆる分野で存在します。

圧縮設定を常に確認し、ネットワーク上で配信されるスクリプトのペイロードを可能な限り小さくするようにしてください。これらのスクリプトは、ブラウザに配信された後、解凍され、その処理時間は圧縮によって変化しません。圧縮は、起動時のインタラクティビティを悪化させる可能性のある巨大なスクリプトペイロードを配信するための良い言い訳ではありません。

## 最小化

テキスト資産の最小化は、ファイルサイズを小さくするために古くから行われている方法です。ソースコードに含まれる不要なスペースやコメントをすべて削除し、転送サイズを小さくするものです。さらに、JavaScriptでは、変数名、クラス名、関数名などを読みにくい記号に変換する「UGL化」という手法を採用している。LighthouseのMinify JavaScript[43]監査は、最小化されていないJavaScriptをチェックします。

43. https://web.dev/unminified-javascript/

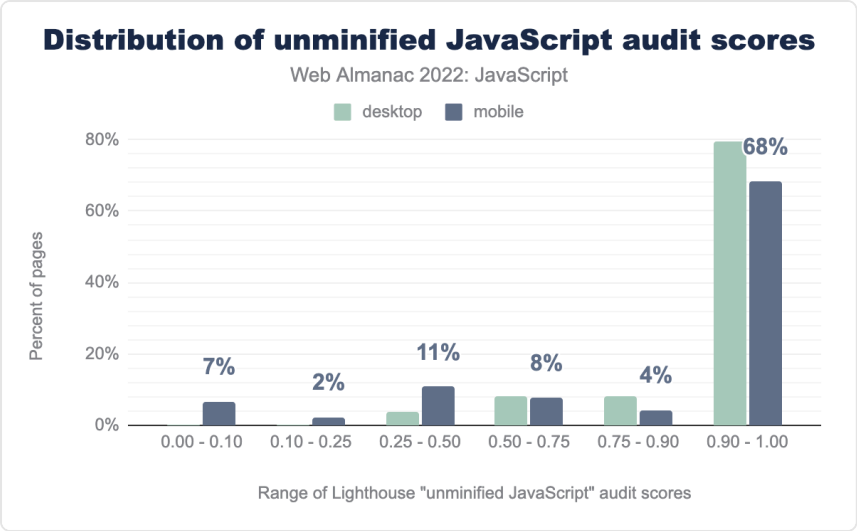



*図2.25. JavaScript監査スコアの分布。*

ここで、0.00は最悪のスコアを表し、1.00は最高のスコアを表します。Lighthouseのminified JavaScript監査では、モバイルページの68%が0.9から1.0のスコアであるのに対し、デスクトップページでは79%となっています。つまり、モバイルでは32%のページがminified JavaScriptを使用する機会があるのに対し、デスクトップでは21%ということになります。

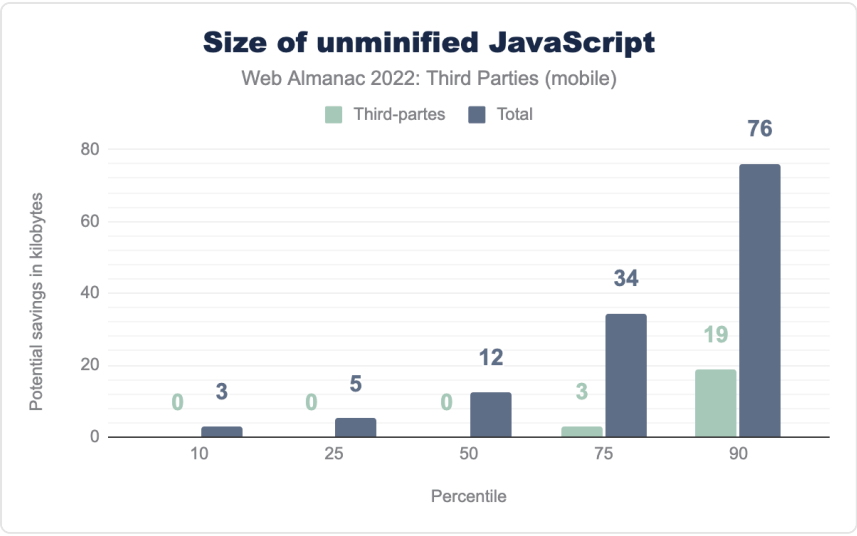



*図2.26. JavaScriptのMinifyによる潜在的な節約効果の分布。*

中央値では、最小化可能なJavaScriptは約12KBです。しかし、75パーセンタイルと90パーセンタイルになると、この数字は34KBから約76KBへと大きく跳ね上がります。サードパーティは、90パーセンタイルまでは非常に良好で、最小化されていないJavaScriptが約19キロバイト提供されています。

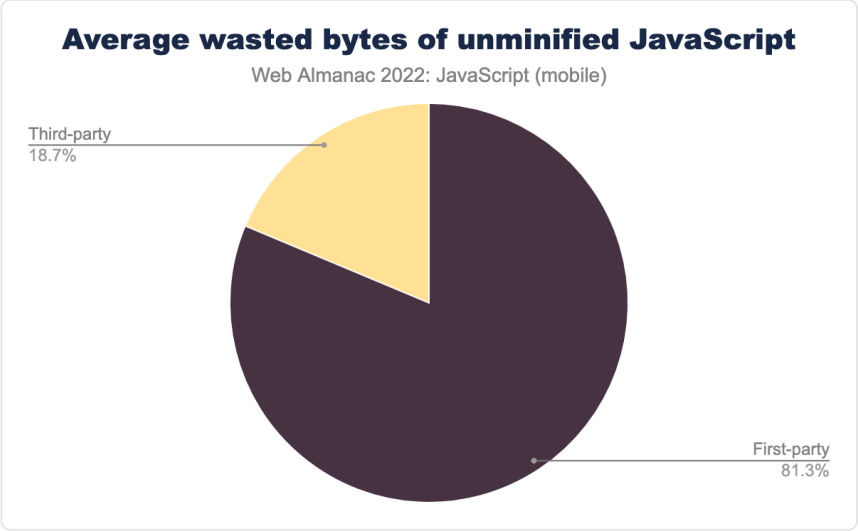



*図2.27. 最小化されていないJavaScriptの平均的なムダなバイト数。*

先ほど紹介したデータから、未定義のJavaScriptのムダなバイト数は、平均を見ればそれほど驚くことではありません。ファーストパーティーは、80%強の未修飾JavaScriptを提供する最大の原因となっています。残りの20% 弱は、送信するバイト数を減らすためにもう少し努力できます。

最小化は、ウェブパフォーマンスの最初の原則の1つである「より少ないバイト数で提供する」ことに対応しています。Lighthouseの監査でJavaScriptが最小化されていないことが判明した場合、バンドラーの設定を確認し、ファーストパーティのコードが実運用で可能な限り効率化されていることを確認します。サードパーティのスクリプトが未定義であることに気づいたら、そのベンダーとチャットして、修正できることを確認する必要があるかもしれません。ウェブ上のサードパーティのあり方については、サードパーティの章を参照してください。

## ソースマップ

ソースマップ[44]は、Web開発者が、最小化および醜形化したプロダクション コードを元のソ

44. *https://firefox-source-docs.mozilla.org/devtools-user/debugger/how_to/use_a_source_map/index.html*

ースへマッピングするために使用するツールです。ソースマップはプロダクションのJavaScriptファイルで使用され、デバッグのための便利なツールです。ソースマップはリソースの最後にあるソースマップファイルを指すコメントで指定するか、`SourceMap` HTTPレスポンスヘッダーとして指定することが可能です。




*図2.28. 一般に公開されているソースマップに対して、ソースマップコメントを指定しているモバイルページの割合。*

モバイルデバイスからアクセスしたJavaScriptリソースの14% が、一般にアクセス可能なソースマップへのソースマップコメントを配信しているのに対し、デスクトップデバイスからアクセスしたJavaScriptリソースでは15% が配信しています。しかし、ソースマップHTTPヘッダーを使用しているページでは、話はまったく異なります。




*図2.29. ソースマップヘッダーを指定しているモバイルページの数。*

モバイルデバイスのJavaScriptリソースに対するリクエストのうち、ソースマップHTTPヘッダーを使用したものは0.12%に過ぎませんが、デスクトップデバイスの場合は0.07%となっています。

パフォーマンスの観点からは、これはあまり意味がありません。ソースマップは開発者の体験を向上させるものです。しかし、避けるべきはインラインソースマップの使用です。インラインソースマップは、オリジナルのソースのbase64表現を制作可能なJavaScriptアセットに挿入するものです。ソースマップをインライン化すると、JavaScriptリソースをユーザーに送信するだけでなく、ソースマップも送信することになり、ダウンロードと処理に時間がかかる巨大なJavaScript資産になる可能性があります。

## レスポンシブ

JavaScriptが影響を与えるのは、スタートアップのパフォーマンスだけではありません。インタラクティブ性を提供するためJavaScriptに依存している場合、それらのインタラクションは、実行に時間がかかるイベントハンドラーによって駆動されます。インタラクションの複雑さと、それを駆動するためのスクリプトの量によっては、ユーザーは入力の応答性の悪さを体験する可能性があります。

## 指標

実験室と現場の両方で応答性を評価するために多くのメトリクスが使用されており、Lighthouse、Chrome UXレポート (CrUX)、HTTP Archiveなどのツールはこれらのメトリクスを追跡して、今日のウェブサイトの応答性の現状をデータに基づいて表示します。とくに断りのない限り、以下のグラフはすべて、原点レベルでその指標の75パーセンタイルを表しており合格と判定される閾値[45]です。

その最初のものは、最初の入力までの遅延 (FID)[46]で、ページとの最初のインタラクションの入力遅延を記録するものです。入力遅延とは、ユーザーがページとインタラクションを行った後、そのインタラクションのイベントハンドラーが実行され始めるまでの時間です。Webサイトとのインタラクションでユーザーが受ける第一印象に焦点を当てた、負荷応答性の指標と考えられています。

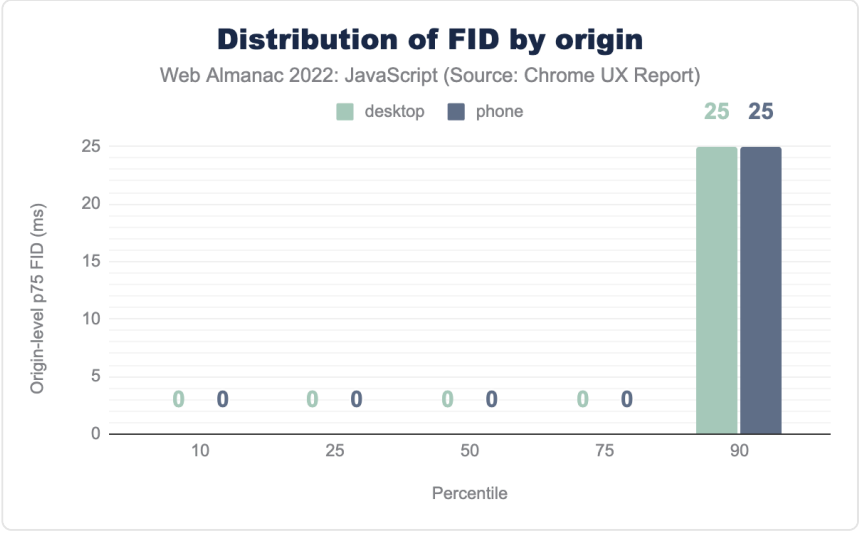



*図2.30. Webサイトの75パーセンタイルFID値の分布。*

このグラフは、全ウェブサイトの75パーセンタイルFID値の分布を表しています。中央のウェブサイトは、デスクトップとモバイルの両方のユーザー体験の少なくとも75%において、FID値が0msです。この「完璧なFID」体験は、75パーセンタイルのWebサイトにまで及んでいます。90パーセンタイルになると、不完全なFID値が見られるようになりますが、これは25ミリ秒にすぎません。

良いFIDの閾値が100 ms[47]であることを考えると、少なくとも90%のWebサイトがこの水準を

45. https://web.dev/i18n/ja/vitals/#core-web-vitals
46. https://web.dev/i18n/ja/fid/
47. https://web.dev/i18n/ja/fid/#fid-

満たしていると言えるでしょう。実際、パフォーマンスの章で行われた分析から、実際に100%のウェブサイトがデスクトップデバイスで、92%がモバイルデバイスで良いFID体験をしていることが分かっているのです。FIDは、異常に寛容な指標です。

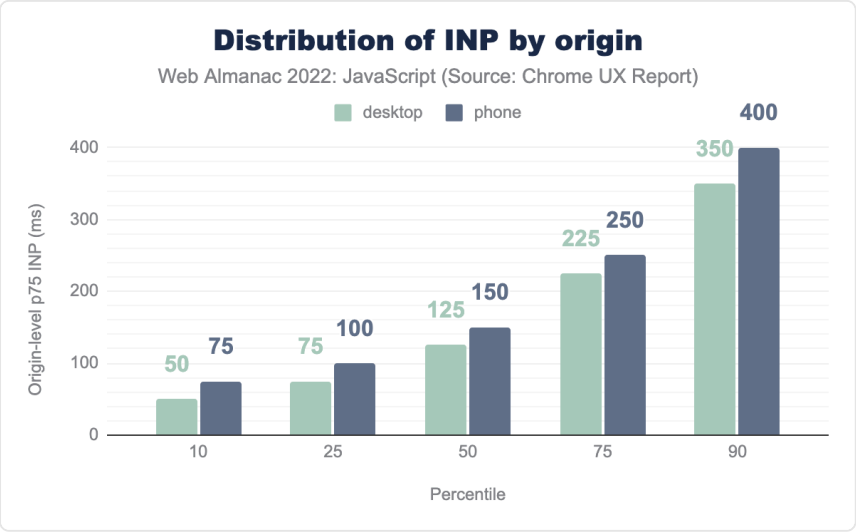



*図2.31. Webサイトの75パーセンタイルINP値の分布。*

しかし、ページのライフサイクル全体にわたるページの応答性を包括的に見るには、次のペイントまでのインタラクション (INP)[48]を見る必要があります。これは、ページに対して行われたすべてのキーボード、マウス、タッチ操作を評価し、総合的にページの応答性を表すことを意図して、操作待ち時間の高いパーセンタイルを選択するものです。

良いINPスコアは200ミリ秒[49]以下だと考えてください。中央値では、モバイルとデスクトップの両方がこの閾値を下回っていますが、75パーセンタイルでは別の話で、モバイルとデスクトップの両方のセグメントが「要改善」の範囲に大きく入っています。このデータは、FIDとはまったく異なり、INPスコアがあまりよくない主な原因であるページ上の 長いタスク[50]を減らすために、ウェブサイトができることをすべて行う機会がたくさんあることを示唆しています。

%E3%81%AB%E3%81%8A%E3%81%91%E3%82%8B%E8%89%AF%E3%81%84%E3%82%B9%E3%82%B3%E3%82%A2%E3%81%A8%E3%81%AF%EF%BC%9F
48. *https://web.dev/articles/inp*
49. *https://web.dev/articles/inp#what's-a-%22good%22-inp-value*
50. *https://web.dev/i18n/ja/long-tasks-devtools/*

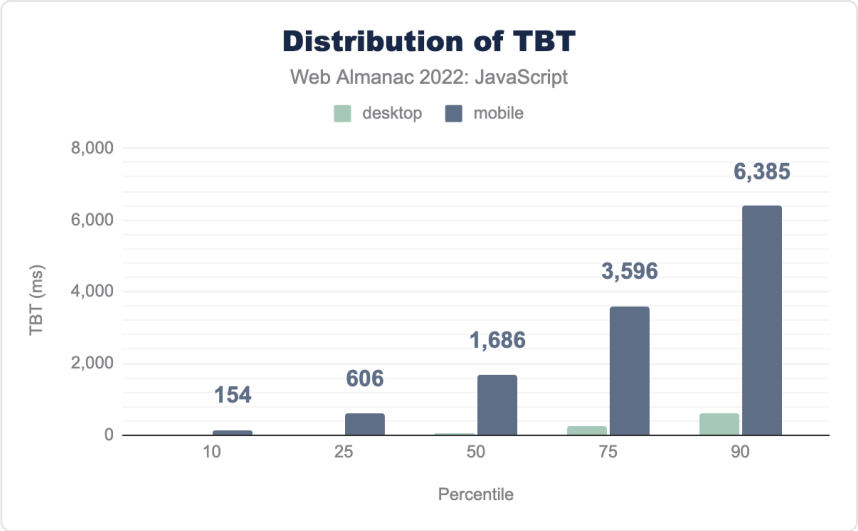



*図2.32. ページのラボベースTBT値の分布。*

長いタスクに関連するものとして、総ブロッキング時間（TBT）[51]メトリクスがあり、これは起動中の長いタスクの合計ブロック時間を計算します。

なお、TBTとTTI（下記）は、先のFIDとINPの統計と異なり、実ユーザーのデータを元にしたものではありません。その代わりに、デバイスに適したCPUとネットワークのスロットルを有効にしたデスクトップとモバイル環境のシミュレーションで、合成パフォーマンスを測定しています。この手法の結果、Webサイト全体に実ユーザーの値が分布するのではなく、各ページで厳密に1つのTBTとTTIの値が得られます。

INPはTBTと非常によく相関している[52]ことを考えると、TBTスコアが高いとINPスコアが低くなると考えるのは妥当なことです。合成アプローチを使用すると、デスクトップとモバイルのセグメントの間に大きな隔たりがあります。これは、処理能力とメモリに優れたデスクトップ端末が、性能の劣るモバイル端末を大きく上回っていることを示しています。75パーセンタイルでは、ページのブロック時間が3.6秒近くあり、体験が悪いと認定されます。

51. *https://web.dev/i18n/ja/tbt/*
52. *https://github.com/GoogleChromeLabs/chrome-http-archive-analysis/blob/main/notebooks/HTTP_Archive_TBT_and_INP.ipynb*

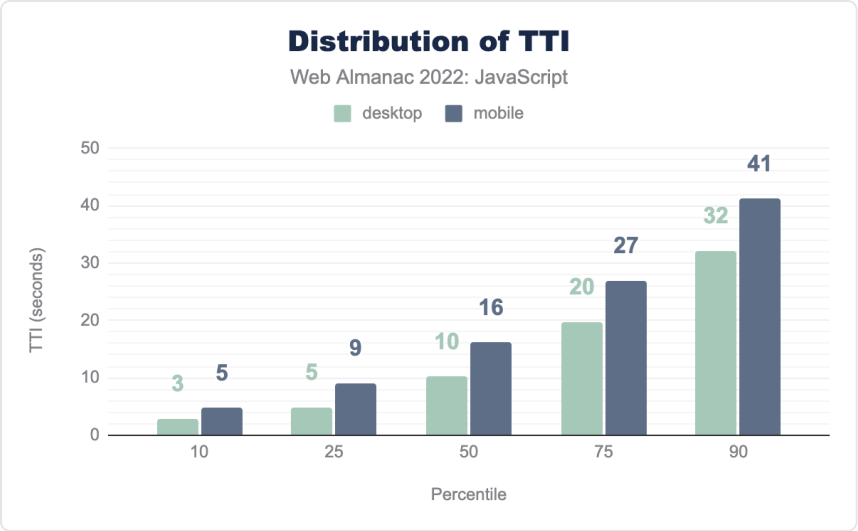



*図2.33. TTIスコアのオリジン、パーセンタイル別の分布。*

最後は、操作可能になるまでの時間（TTI）[53]です。この指標は、5秒未満であれば「良い」と見なされます。10パーセンタイルだけがかろうじて5秒を切っていることから、シミュレーション環境のほとんどのウェブサイトがJavaScriptに依存しており、ページが適切な時間枠内でインタラクティブになることができません。とくに90パーセンタイルでは、インタラクティブになるまでに41.2秒という途方もない時間がかかっています。

## 長いタスク/ブロック時間

前のセクションでおわかりのように、インタラクションの応答性が悪くなる主な原因は、長いタスクです。長いタスクとは、メインスレッドで50ミリ秒以上実行されるタスクのことです。50ミリ秒を超えるタスクの長さは、そのタスクのブロッキング時間であり、タスクの合計時間から50ミリ秒を引くことで計算できます。

長いタスクは、そのタスクが終了するまでメインスレッドが他の作業を行えないようにブロックするため、問題となります。ページに長いタスクがたくさんあると、ブラウザがユーザーの入力に反応するのを遅く感じられることがあります。極端な例では、ブラウザがまったく反応しないように感じることさえあります。

53. https://web.dev/i18n/ja/tti/

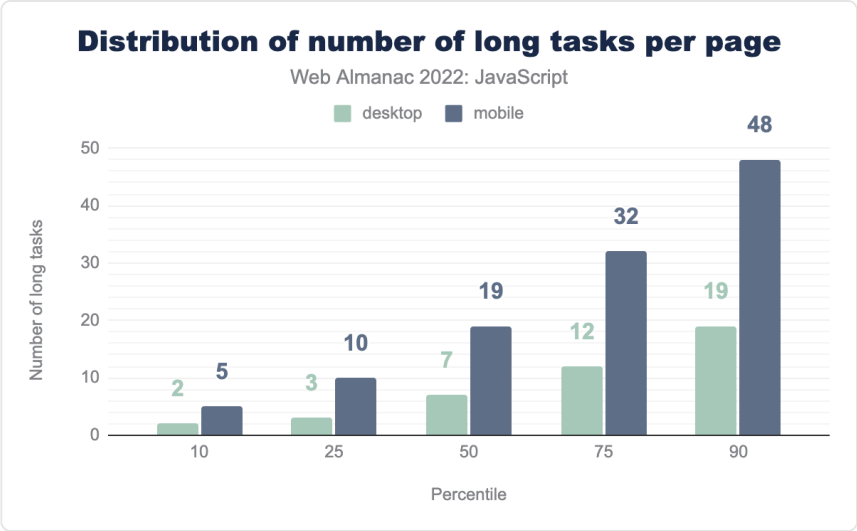



*図2.34. ページごとの長いタスクの数の分布。*

中央値のページは、モバイルデバイスで19の長いタスクに遭遇し、デスクトップデバイスでは7の長いタスクに遭遇しています。もっとも、デスクトップ端末はモバイル端末よりも処理能力やメモリリソースが高く、活発に冷却されていることを考えれば、これは理にかなった結果です。

しかし、パーセンタイルが高くなると、状況はかなり悪化します。ページあたり75パーセンタイルでの長いタスクは、モバイルとデスクトップでそれぞれ32と12です。

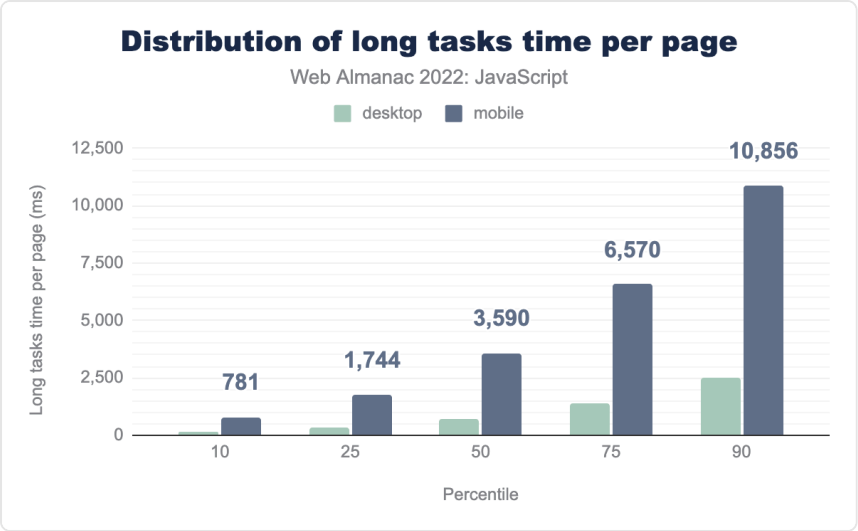



*図2.35. ページごとの長時間タスクの時間分布。*

ページごとの長時間のタスクの数を知るだけでは不十分で、それらのタスクがページ上でかかっている総時間を理解する必要があります。モバイルページの中央値では、長時間のタスクに費やされる時間は3.59秒ですが、デスクトップページでは0.74秒とはるかに少なくなっています。

75パーセンタイルでは、モバイル端末の処理時間が大幅に低下しており、1ページあたり約6.6秒の処理時間が長いタスクの処理に費やされています。これは、ブラウザが最適化できる、あるいは別のスレッドのウェブワーカーに移行できる可能性のある激しい作業に費やしている多くの時間です。いずれにせよ、この結果は、モバイルウェブと応答性に問題があることを示唆しています。

## スケジューラAPI

JavaScriptのタスクのスケジューリングは、歴史的にブラウザに委ねられてきました。新しいメソッドとして、`requestIdleCallback`や`queueMicrotask`がありますが、これらのAPIは粗い方法でタスクをスケジュールします。とくに`queueMicrotask`の場合、使い方を誤るとパフォーマンスの問題を引き起こす可能性があります。

Scheduler APIは最近リリースされ、優先順位に基づくタスクのスケジューリングをより細かく制御できるようになりましたが、Chromiumベースのブラウザ[54]にしか今のところ限定されていません。

54. *https://caniuse.com/mdn-api_scheduler_posttask*

# 0.002%

*図2.36. Scheduler APIを使用しているモバイルページの割合です。*

現在、Scheduler APIを使用するJavaScriptを搭載しているモバイルページは2千分の1 (0.002%) に過ぎませんが、デスクトップページでは3千分の1 (0.003%) が搭載しています。この非常に新しい機能に関するドキュメントの欠如と、その限定的なサポートを考慮すると、驚くべきことではありません。しかし、この機能に関する文書が利用可能になり、とくにフレームワークで使用されるようになれば、この数は増加すると思われます。この重要な新機能を採用することで、最終的にはより良いユーザー体験の結果が得られると信じています。

## 同期型XHR

AJAX、つまりナビゲーション リクエストなしでページ上のデータを非同期に取得し情報を更新する `XMLHttpRequest` (XHR) メソッドの使用は、動的なユーザー体験を生み出す方法として非常に一般的でした。これは主に非同期の `fetch` メソッドに取って代わられましたが、XHRはすべての主要なブラウザ[55]でまだサポートされています。

XHRには同期的なリクエストを行うためのフラグがあります。同期XHR[56]は、イベントループとメインスレッドがリクエストを終了するまでブロックされ、結果としてデータが利用可能になるまでページが停止するため、パフォーマンス上有害です。 `fetch` は、よりシンプルなAPIで、より効果的かつ効率的な代替手段であり、データの同期取得はサポートされていません。

# 2.5%

*図2.37. モバイルページで同期XHRを使用している割合です。*

同期XHRはモバイルページの2.5%、デスクトップページの2.8%で使用されているだけですが、どんなに小規模であっても、その使用が続いていることは一部のレガシーアプリケーションがこの時代遅れの手法に依存している可能性があり、ユーザー体験を損なうシグナルであることに変わりはありません。

同期XHR、および一般的なXHRの使用は避けてください。 `fetch` はより人間工学的な代替

55. https://caniuse.com/mdn-api_xmlhttprequest
56. https://developer.mozilla.org/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests#synchronous_request

手段ですが、設計上同期的な機能はありません。同期XHRを使わない方がページのパフォーマンスは良くなりますし、いつの日かこの数がゼロになることを望んでいます。

### `document.write`

DOMの挿入メソッド（`appendChild`など）が導入される前は、`document.write`を使用して、ドキュメント内の`document.write`が行われた位置にコンテンツを挿入しました。

`document.write`は非常に問題があります。ひとつにはHTMLパーサをブロックしてしまうこと、そして他の多くの理由から問題があり、HTMLの仕様書では、その使用について警告しています[57]。低速な接続では、この方法でノードを追加するためにドキュメントの解析をブロックすると、完全に回避可能なパフォーマンスの問題が発生します。


*図2.38. `document.write`を使用したモバイルページの数です。*

観測されたページの18%は、適切な挿入方法の代わりに`document.write`を使ってDOMにコンテンツを追加しており、一方、デスクトップページの17%はまだそうしていることがわかりました。これは、ドキュメントに新しいノードを挿入するために望ましいDOMメソッドを使用するように書き換えられていないレガシーアプリケーションや、サードパーティのスクリプトがまだこれを使用していることが原因であると思われます。

この傾向が弱まることを期待します。すべての主要なブラウザは、このメソッドを使用しないよう明確に警告しています。まだ非推奨ではありませんが、今後数年間のブラウザでの存在も保証されていません。もし`document.write`の呼び出しがあなたのウェブサイトにあるのなら、できるだけ早く取り除くことを優先させるべきでしょう。

### レガシーJavaScript

JavaScriptはここ数年でかなり進化しています。新しい言語機能の導入により、JavaScriptはより有能でエレガントな言語となり、開発者はより簡潔なJavaScriptを書くことができるようになり、結果としてJavaScriptの読み込み量が減りました。ただし、それらの機能がBabelのようなtranspilerを使って不必要に古い構文に変換されていないことが条件です。

Lighthouseは現在、`async`と`await`やJavaScript クラス[58]などの新しい、しかし広くサポートされている言語機能のトランスフォームなど、現代のウェブでは不要かもしれない

57. *https://html.spec.whatwg.org/multipage/dynamic-markup-insertion.html#document.write()*
58. *https://developer.mozilla.org/docs/Web/JavaScript/Reference/Classes*

Babelトランスフォームをチェックします。


*図2.39. レガシーJavaScriptを搭載しているモバイルページの割合です。*

モバイルページの3分の2強は、トランスフォームされているJavaScriptリソースを出荷しているか、あるいは不要なレガシーJavaScriptを含んでいます。

トランスフォームは互換性のためにプロダクションのJavaScriptに多くのバイトを追加しますが、古いブラウザをサポートする必要がない限り、これらのトランスフォームの多くは不要であり、起動時のパフォーマンスに悪影響を与える可能性があります。モバイルでは非常に多くのページが、デスクトップでは68% のページが、このようなトランスフォームを実装していることが気になります。

Babelはコンパイラの仮定機能[59]などを通じて、この問題を解決するために多くのことを行っていますが、Babelはまだユーザー定義の設定によって動いており、古い設定ファイルの存在下では多くのことを行うことができるだけです。

前述のように開発者には、Babel[60]とBrowserslist[61]の設定を慎重に見直し、必要なブラウザで動作するように、最小限のトランスフォームがコードに適用されているかどうか確認することを強く推奨します。そうすることで、エンドユーザーへの出荷バイト数を大幅に削減できます。このあたりは開発者の腕の見せ所ですが、言語の進化が比較的安定してきた今、この数字が時間とともに減少することを期待しています。

## JavaScriptはどのように使われているのですか？

Webページを構築する方法は1つだけではありません。ウェブプラットフォームを直接使うことを選ぶ人もいるかもしれませんが、ウェブ開発者業界のトレンドは、私たちの仕事をやりやすく、推論しやすくする抽象化されたものに手を伸ばすことであることは否定できません。例年通り、ライブラリやフレームワークの役割と、それらのライブラリやフレームワークがユーザーにとってウェブをより危険な場所にする可能性のあるセキュリティ上の脆弱性をどの程度頻繁に提示するかを探っていきます。

59. *https://babeljs.io/docs/en/assumptions*
60. *https://babeljs.io/docs/en/configuration*
61. *https://github.com/browserslist/browserslist*

## ライブラリとフレームワーク

ライブラリとフレームワークは、開発者の体験の大きな部分を占めますが、フレームワークのオーバーヘッドによってパフォーマンスが損なわれる可能性もあります。開発者はこのトレードオフを概ね受け入れていますが、ウェブで一般的に使用されているライブラリとフレームワークを理解することは、ウェブがどのように構築されているかを理解する上で非常に重要です。このセクションでは、2022年のウェブにおけるライブラリとフレームワークの状態について見ていきます。

### ライブラリの利用状況

ライブラリやフレームワークの利用状況を把握するために、HTTP ArchiveではWappalyzerを使って、ページで使われている技術を検出しています。

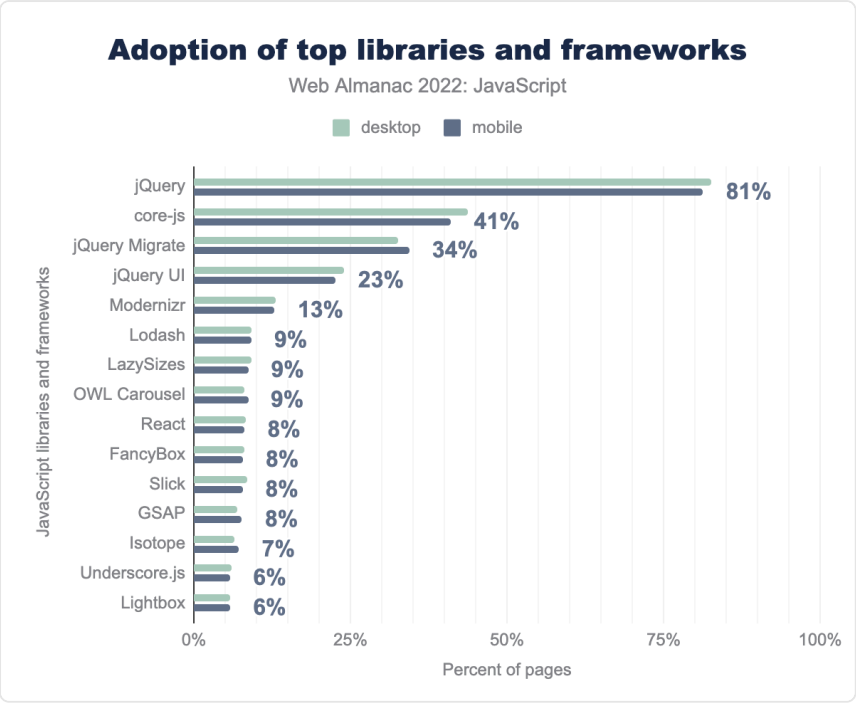



*図2.40. もっとも多いライブラリやフレームワークの採用。*

jQueryが今日のWebでもっとも使用されているライブラリであることは、いまだ驚くに値しない。WordPressが35%のサイトで使用されていることも理由の1つですが、それでもjQueryの使用の大半はWordPressプラットフォーム以外での使用です。

jQueryは比較的小さく、動作もそれなりに速いのですが、それでもアプリケーションの中ではある程度のオーバーヘッドを占めます。jQueryが提供するもののほとんどは、現在ではネイティブDOM APIで可能[62]であり、今日のWebアプリケーションでは不要かもしれません。

多くのウェブアプリケーションがBabelでコードをトランスフォームし、ブラウザ間のAPIの欠落を埋めるためにcore-jsを使用することが多いため、core-jsの使用は驚くべきことでありません。ブラウザが成熟するにつれて、この数字は下がるはずです。これは実に良いことで、最近のブラウザはこれまで以上に高性能となっており、core-jsのコードを提供してもムダなバイトに終わる可能性があるからです。

Reactの利用状況は昨年と変わらず8%で、JavaScriptのエコシステムの選択肢が増えたため、ライブラリの利用が頭打ちになったことを示唆しているのかもしれない。

### 併用するライブラリ

同じページで複数のフレームワークやライブラリが使われているのは、決して珍しいシナリオではありません。昨年に引き続き、この現象を検証することで、2022年にどれだけのライブラリとフレームワークが併用されたかを洞察します。

| ライブラリ | デスクトップ | モバイル |
|---|---|---|
| jQuery | 10.19% | 10.33% |
| jQuery、jQueryマイグレート | 4.30% | 4.94% |
| core-js、jQuery、jQueryマイグレート | 2.48% | 2.80% |
| core-js、jQuery | 2.78% | 2.74% |
| jQuery、jQuery UI | 2.40% | 2.07% |
| core-js、jQuery、jQueryマイグレート、jQuery UI | 1.18% | 1.36% |
| jQuery、jQueryマイグレート、jQuery UI | 0.88% | 0.99% |
| GSAP、Lodash、Polyfill、React | 0.48% | 0.93% |
| Modernizr、jQuery | 0.87% | 0.86% |
| core-js | 0.92% | 0.85% |

*図2.41. 観測されたページで併用されているライブラリやフレームワークの解析。*

jQuery、UIフレームワーク、マイグレーションプラグインの組み合わせが上位7位を占め、

62. https://youmightnotneedjquery.com/

core-jsもライブラリの利用状況において重要な役割を果たしていることから、jQueryが大きな力を秘めていることは明らかです。

### セキュリティの脆弱性

今日のWebではJavaScriptが広く普及しており、インストール可能なJavaScriptパッケージの出現により、JavaScriptのエコシステムにセキュリティ上の脆弱性が存在することは驚くことでありません。


*図2.42. 脆弱性のあるJavaScriptのライブラリやフレームワークをダウンロードしたモバイルページの割合です。*

モバイルページの57%が脆弱なJavaScriptライブラリまたはフレームワークを提供していることは重要ですが、この数字は昨年の64%から減少しています。これは心強いことですが、この数字を下げるにはかなりの努力が必要です。より多くのセキュリティ脆弱性にパッチが適用されることで、開発者が依存関係を更新し、ユーザーを危険にさらすことを回避する動機付けになることを期待しています。

| ライブラリまたはフレームワーク | デスクトップ | モバイル |
|---|---|---|
| jQuery | 49.12% | 48.80% |
| jQuery UI | 16.01% | 14.88% |
| Bootstrap | 11.53% | 11.19% |
| Moment.js | 4.54% | 3.91% |
| Underscore | 3.41% | 3.11% |
| Lo-Dash | 2.52% | 2.44% |
| GreenSock JS | 1.65% | 1.62% |
| Handlebars | 1.27% | 1.12% |
| AngularJS[63] | 0.99% | 0.79% |
| Mustache | 0.44% | 0.57% |

*図2.43. もっとも多く利用されているライブラリやフレームワークの上位10位までのうち、JavaScriptの既知の脆弱性を持つページの割合です。*

jQueryは現在Web上でもっとも普及しているライブラリであり、jQueryとそれに関連するUIフレームワークが、現在Web上でユーザがさらされているセキュリティ脆弱性のかなりの部分を代表していることは、驚くことではありません。これは、一部の開発者が、既知の脆弱性に対する修正を利用していない旧バージョンのスクリプトをまだ使用していることが、原因である可能性があります。

Bootstrapは、開発者がCSSを直接使用せずに、素早くプロトタイプや新しいレイアウトを構築できるようにするUIフレームワークで、注目すべきエントリーであると言えます。GridやFlexboxなどの新しいCSSレイアウトモードがリリースされれば、Bootstrapの使用は減少し、代わりに開発者がBootstrapの依存関係を更新して、より安全でセキュアなウェブサイトを提供できるようになるかもしれません。

どのようなライブラリやフレームワークを使用しているかにかかわらず、ユーザを危険にさらすことを避けるために、可能な限り依存関係を定期的に更新するようにしてください。パッケージの更新により、リファクタリングやコードの修正を発生させることがありますが、その労力は、責任の軽減とユーザーの安全性の向上に見合うものです。

63. https://angularjs.org

## ウェブコンポーネントとシャドウDOM

これまでWeb開発では、多くのフレームワークで採用されているコンポーネント化モデルが推進されてきました。Webプラットフォームも同様に、Webコンポーネントとshadow DOMを通じてロジックとスタイリングのカプセル化を提供するように進化してきました。今年の分析の手始めとして、カスタム要素[64]から始めます。

**2.0%**

*図2.44. カスタムエレメントを使用したデスクトップページの割合です。*

この数字は、昨年のデスクトップページでのカスタム要素使用率の分析結果である3%から少し減少しています。カスタム要素が提供する利点と、モダンブラウザでのそれなりに幅広いサポートにより、ウェブコンポーネントモデルが開発者にウェブプラットフォームの組み込み要素を活用し、より高速なユーザー体験を実現することを促してくれることを期待しています。

**0.39%**

*図2.45. モバイルページでシャドウDOMが使用されている割合。*

シャドウDOM[65]は、サブ要素やスタイリングのための独自のスコープを含む専用のノードをドキュメント内に作成し、メインDOMツリーからコンポーネントを分離することができるようにします。シャドウDOMを使用しているページは全体の0.37%という昨年の数値と比較すると、モバイルページで0.39%、デスクトップページでは0.47%と、その採用率はほぼ横ばいとなっていることがわかります。

**0.05%**

*図2.46. モバイルページでテンプレートが使用されている割合。*

`template` 要素は、開発者がマークアップのパターンを再利用するのに役立ちます。テンプレートは、JavaScriptから参照されたときのみ、その内容を表示します。テンプレートはWebコンポーネントとうまく機能します。JavaScriptからまだ参照されていないコンテンツ

64. https://developers.google.com/web/fundamentals/web-components/customelements
65. https://developers.google.com/web/fundamentals/web-components/shadowdom

は、シャドウDOMを使ってシャドウルートに追加されるからです。

現在、デスクトップとモバイルの両方で、およそ0.05%のウェブページが `template` 要素を使用しています。テンプレートはブラウザで十分にサポートされていますが、現在その採用はほとんどありません。


*図2.47. `is` 属性を使用したモバイルページの割合です。*

HTMLの `is` 属性は、カスタム要素をページに挿入するための代替手段です。カスタム要素の名前をHTMLタグとして使用するのではなく、その名前は任意の標準的なHTML要素に渡され、ウェブコンポーネントのロジックが実装されます。`is` 属性は、ウェブコンポーネントがページに登録できなかった場合でも、標準的なHTML要素の動作にフォールバックすることができるウェブコンポーネントを使用する方法です。

この属性の使用状況を追跡するのは今年がはじめてですが、当然のことながら、その採用率はカスタム要素そのものよりも低くなっています。Safariがサポートされていないため、iOSのブラウザとmacOSのSafariはこの属性を利用できず、この属性の利用が限定的である一因になっている可能性があります。

## 結論

JavaScriptの状況は、昨年のトレンドが示唆するように、ほぼ継続しています。確かに提供数は増えていますが、minification、リソースヒント、圧縮、さらには使用するライブラリに至るまで、さまざまなテクニックを利用することで過剰なJavaScriptの悪影響を軽減しようとしています。

JavaScriptの状況は常に進化しています。私たちがこれまで以上にJavaScriptへ依存していることは明らかですが、これはウェブの総合的なユーザー体験にとって問題をもたらすものです。私たちは、プロダクションウェブサイトに搭載されるJavaScriptの量を減らすために、できる限りのことをする必要があり、またそれ以上のことをする必要があります。

ウェブプラットフォームが成熟するにつれ、そのさまざまなAPIや機能を直接採用することが意味のあることであれば、より多く採用されるようになることを期待しています。より良い開発者体験のためにフレームワークを必要とする体験については、さらなる最適化と、フレームワークの作者がより良い開発者体験とユーザー体験の両方を開発するために新しいAPIを採用する機会があることを期待しています。

来年は、その流れが変わることを期待したい。その間、できる限りWebを速くするために[66]、lab[67]とfield[68]両データに目を向けつつできる限りのことを続けていこうではありませんか。

## 著者

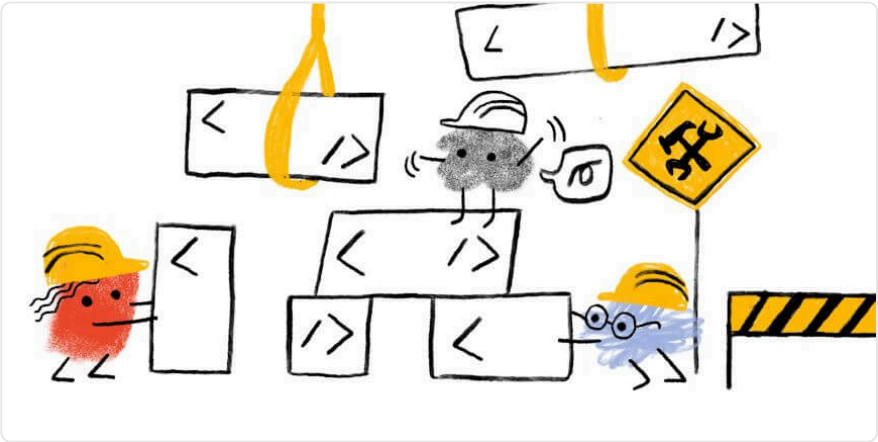

### Jeremy Wagner

@malchata malchata https://jlwagner.net/

Jeremy Wagnerは、GoogleのパフォーマンスとCore Web Vitalsに関するテクニカルライターです。また、A List Apart、CSS-Tricks、Smashing Magazineでも執筆しています。いつか、砂がまだ考えることを教えられていない人里離れた荒野に移住する予定です。それまでは、妻と義理の娘たちとともにミネソタ州のツインシティに住み、ストリップモールの存在を嘆き続けています。


66. *https://web.dev/i18n/ja/fast/*
67. *https://web.dev/lab-and-field-data-differences/#lab-data*
68. *https://web.dev/lab-and-field-data-differences/#field-data*

# 部 I 章 3
# マークアップ



*Jens Oliver Meiert によって書かれた。*
*Brian Kardell と Simon Pieters によってレビュ。*
*Rick Viscomi による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

2020年版の章[69]にあるように、HTMLがなければ、WebページもWebサイトもWebアプリもありません。HTMLがなければ、Webは存在しないと言ってもよいでしょう。そのため、HTMLはもっとも重要なWeb標準の1つ、とは言えないまでも、もっとも重要なWeb標準の1つであると言えます。

そこで、例年通り、数百万ページのデータセットを使って、モバイルセットで790万ページ、デスクトップセットで540万ページ（重複あり）を対象に、HTMLの調査も実施しました。この章では、HTMLに関する「すべて」をカバーしているわけではありません。したがって、私たちが集めたデータを分析し[70]、あなた自身の結論を共有することを明確に推奨します。その際には、#htmlalmanac[71]のタグを付けてください。

69. *https://almanac.httparchive.org/ja/2020/markup#序章*
70. *https://docs.google.com/spreadsheets/d/1grkd2_1xSV3jvNK6ucRQ0OL1HmGTsScHuwA8GZuRLHU/edit*
71. *https://twitter.com/hashtag/htmlalmanac*

# ドキュメントデータ

HTMLをどのように書くかについては、好奇心をそそられることがたくさんあります。たくさんの質問をできますが、一般的なHTMLに関しては、マークアップ自体の内容に触れる前、私たちのHTMLがどのようにブラウザに送信されるかを見てみましょう。

## Doctypes

| *Doctype* | デスクトップ | モバイル |
|---|---|---|
| `html` | 88.1% | 90.0% |
| `html -//w3c//dtd xhtml 1.0 transitional//en http://www.w3.org/tr/xhtml1/dtd/xhtml1-transitional.dtd` | 4.7% | 3.9% |
| Doctypeなし | 3.0% | 2.7% |
| `html -//w3c//dtd xhtml 1.0 strict//en http://www.w3.org/tr/xhtml1/dtd/xhtml1-strict.dtd` | 1.2% | 1.1% |
| `html -//w3c//dtd html 4.01 transitional//en http://www.w3.org/tr/html4/loose.dtd` | 0.9% | 0.6% |
| `html -//w3c//dtd html 4.01 transitional//en` | 0.4% | 0.4% |

*図3.1. Doctypeの使用法。*

まずはdoctypeから。もっとも人気のあるdoctypeはどれでしょう？でも、この答えはわかっているはずです。それは、短くてシンプルで退屈な標準的なHTMLのdoctype、つまり、`<!DOCTYPE html>` です。


*図3.2. 標準的なHTMLのdoctypeを使用したモバイル。*

すべてのモバイルページの90%がこれを使用しています。モバイルデータセットが最大であるため、この章では通常そのデータを使用します。次に多いのはXHTML 1.0

Transitional（3.9%、2021年の4.6%から減少[72]）です。その次は2.7%でまったくdoctypeが設定されていない、昨年の2.5%から上昇しています。

## 圧縮

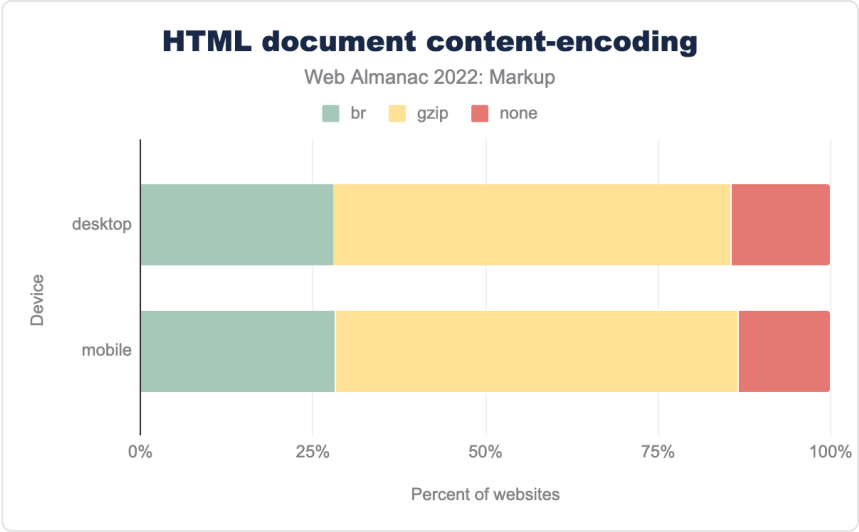



*図3.3. HTML文書のコンテンツのエンコーディング。*

HTML文書は圧縮されていますか？いくつ？どのように？全体の58%（昨年比5.8%減）がgzip圧縮、28%（同6.1%増）がBrotliで圧縮されています。全体として、圧縮される文書がわずかに増え、より効果的に圧縮されています。

72. *https://almanac.httparchive.org/ja/2021/markup#doctypes*

## 言語

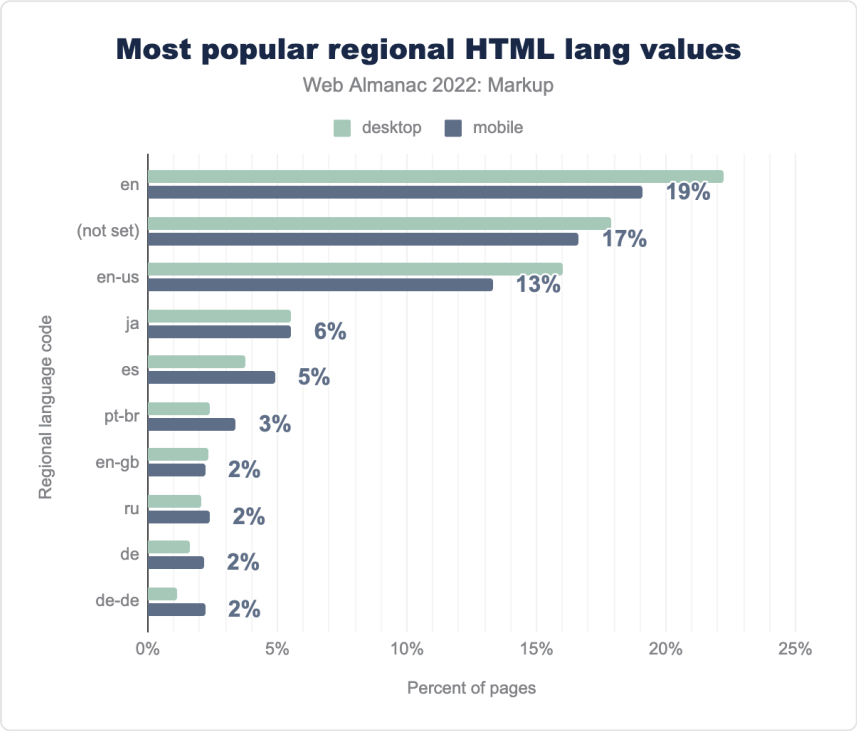



*図3.4. もっとも一般的な地域別HTMLのlangの値です。*

言語についてはどうでしょうか？私たちのデータセットでは、35%のページで `lang` 属性が英語にマッピングされていました。17%は言語が設定されていません。サンプルは偏っている可能性が高く、また全世界を反映するほど大きくはありませんし、`lang` 属性が使われていないことは言語が設定されていないことと同じではないので、このデータは役に立ちません。

## 適合性

ドキュメントがHTMLの仕様に準拠しているか。有効ですか？簡単に見分けるには、W3Cマークアップ検証サービス[73]のようなツールを使うとよいでしょう。

まだしていませんし、確認もできません。では、なぜこのセクションを入れたのか？

73. *https://validator.w3.org/*

少なくとも適合性について言及する理由は適合性をチェックしない場合、検証しない場合、かなりの確率で実際には、事実上100%の確率で[74]、少なくともいくつかの架空の、空想の（したがって間違った）HTMLを書いてしまうことになるからです。しかしHTMLはフィクション、ファンタジーでもなく、何が有効、無効かという明確なルールがあるハードな技術標準なのです。

プロフェッショナルにとって、これらのルールを知っておくことは良いことです。また、動作するコードを作成し、余分なものを含まないことも良い仕事です。そして学習することと、動作しないものや余計なものを提供しないことの両方が、適合性が重要な理由であり、バリデーションが重要な理由なのです。

Web Almanacで共有できる適合性データはまだありませんが、だからといって、このポイントの重要性が低いというわけではありません。もしあなたがまだ適合性に注目していないのなら、HTML出力の検証を始めてください。もしかしたら、Web Almanacの次のエディションで、あなたのおかげで共有できるポジティブなニュースがあるかもしれません。

## ドキュメントサイズ

HTMLペイロードとドキュメントサイズは、この連載の定番です。私たちは2019年からこの情報を見てきました。しかし、その傾向は明らかで、他の章でも確認できる共通のテーマに沿っていますが、決して素晴らしいものではありません。

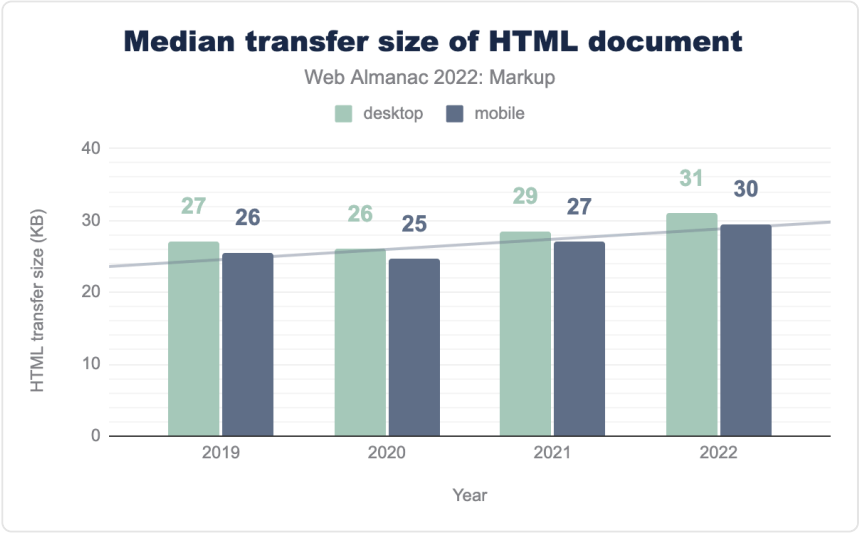



*図3.5. HTML文書の転送サイズの中央値。*

74. https://meiert.com/en/blog/valid-html-2022/

2020年に一時的に緩和されたものの、2021年、2022年と文書サイズは成長を続け、モバイルデータセットにおける転送サイズの中央値は30kBとなりました。

この傾向に対抗するひとつの方法は、HTML、（XHTMLの方法ではなく）[75]の方法で書くことで、すでにHTMLの転送サイズを小さくできます。*公開: この著者は、HTMLの書き方の分類を考えるのが好きで、最小限のHTMLを推進するのを楽しんでいます。*

## 要素

`svg` と `math` 要素を含めない場合（これらはHTMLの外部で指定されているから）、現在のHTML仕様は111個の要素で構成されています。

*タグではなく要素です。なぜなら、`<li>` や `</ins>` のような単なる開始タグや終了タグを指しているのではないからです。また、HTML 要素の数え方を変える人もいますが、もっとも重要なのは、どのように数えているかをはっきりさせることです。*[76]

何を観察できるのか？

### 要素の多様性

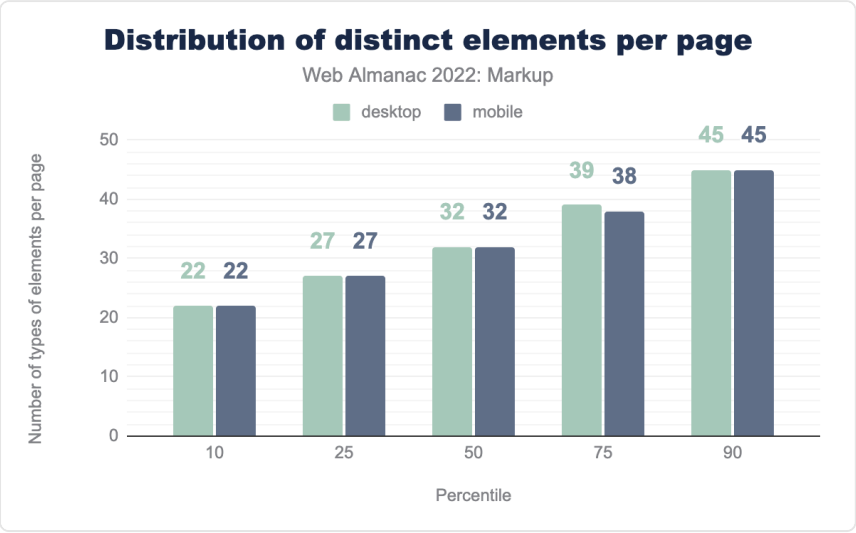



*図3.6. ページごとの明確な要素の分布。*

75. https://css-tricks.com/write-html-the-html-way-not-the-xhtml-way/
76. https://meiert.com/en/blog/the-number-of-html-elements/

まず注目すべきは、開発者が1ページあたり使用する要素の種類が若干増えていることで、1文書あたりの要素の中央値は32種類となっています。

中央値は2021年31要素[77]、2020年30要素[78]から上昇した。これは全体的な傾向として、開発者がHTML要素をより有効に活用し、その目的に応じた使い方をするようになったことの表れかもしれません。

しかし、原稿のサイズが大きくなると、1ページあたりの要素数が増えるという傾向もあります。

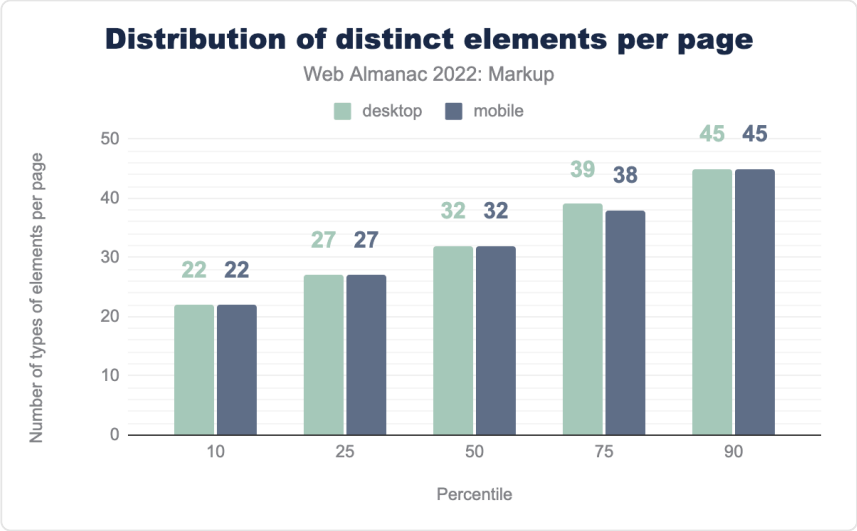



*図3.7. 1ページあたりの要素数分布。*

中央値は現在、1ページあたり653要素で、2021年の616要素、2020年の587要素から増加しています。すべて各モバイルデータによるものです。私たちはより多くのコンテンツを発行し、それらを保持するためにより多くの要素を必要とするのでしょうか（たとえば、テキストごとに段落を増やし、より多くの `p` 要素を必要とするようなもの）？それともこれは単に `div` が抑制されずに蔓延していることの表れなのでしょうか？私たちのデータはこれに答えてはいませんが、おそらくその両方、あるいはそれ以上の理由によるものでしょう。

## トップ要素

もっともよく使われるのは、以下の要素です。

77. *https://almanac.httparchive.org/ja/2021/markup#要素の多様性*
78. *https://almanac.httparchive.org/ja/2020/markup#要素の多様性*

| 2019 | 2020 | 2021 | 2022 |
|---|---|---|---|
| div | div | div | div |
| a | a | a | a |
| span | span | span | span |
| li | li | li | li |
| img | img | img | img |
| script | script | script | script |
| p | p | p | p |
| option | link | link | link |
| | i | meta | i |
| | option | i | meta |

*図3.8. もっとも使われている要素。*

`div` 要素はもっともよく使われている要素です。モバイル データセットでは2,123,819,193回、デスクトップ データセットでは1,522,017,185回、出現しました。


*図3.9. `div` 要素の割合。*

ディバイス[79]は実在します。

もしあなたが奇妙な存在である `i` 要素について疑問に思うならば、それはやはり Font Awesome[80]とその議論の余地のないこの要素の誤用によるところが大きいのだろうと推論できます。この要素は、XHTML時代には誰もが `em` を代わりに使うように勧めていたため、悪い評判もあります。しかし、そのアドバイスは適切でなく、 `i` 要素にはその使用例があるのです。

もっとも多くの文書で使用されている要素については、少し違った印象を受けます。

79. https://en.wiktionary.org/wiki/divitis
80. https://fontawesome.com/

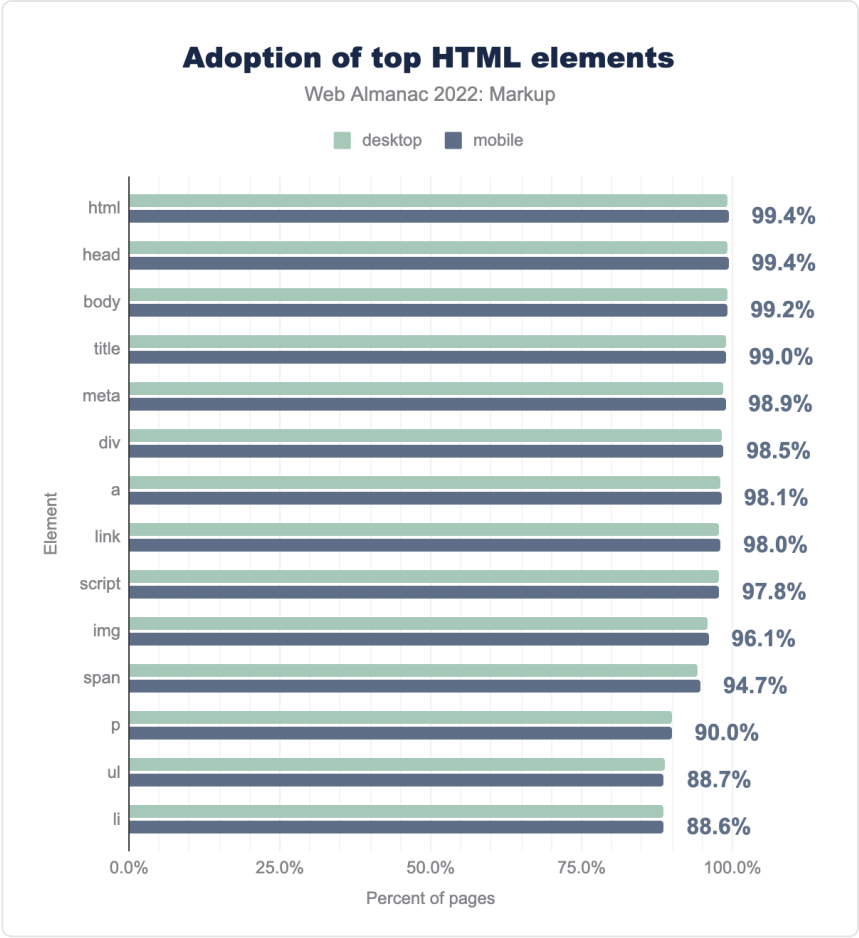



*図3.10. トップHTML要素の採用。*

ほぼすべての文書で `html`、`head`、`body` タグが使われているのは当然のことで、これらは自動的にDOMに挿入され、ここでカウントされています。100%より少し少ないのは、私たちが使用しているJavaScript APIをオーバーライドすることで検出を中断している少数のページによるものです。たとえば、MooTools[81] は `JSON.stringify()` APIをオーバーライドしています。

全サンプル文書の1%で `title` を見逃す方がよっぽど驚きです。この要素はオプションではなく、DOMに挿入されておらず、その省略は適合性チェックの欠如の指標となる。

81. *https://mootools.net/*

とくに、`a`、`img`、`meta`は、2005年のIan HicksonによるHTMLの研究[82]以来、人気のある要素となっています。

現在の標準に含まれるHTML要素の中で、もっとも使用頻度の低いものは何かと聞かれたら？これは、`samp`で、モバイルセットではわずか2,002件しかありません。

## カスタム要素

Custom要素[83]内側の名前にハイフンが使われていることで大まかに識別できる要素も再びサンプルに、含まれるようになりました。しかし、今年はトップ10をSlider Revolution[84]が完全に独占しているのです。

| *カスタム要素* | *デスクトップ* | *モバイル* |
|---|---|---|
| *rs-module-wrap* | *2.1%* | *2.3%* |
| *rs-module* | *2.1%* | *2.3%* |
| *rs-slides* | *2.1%* | *2.3%* |
| *rs-slide* | *2.1%* | *2.3%* |
| *rs-sbg-wrap* | *2.0%* | *2.2%* |
| *rs-sbg-px* | *2.0%* | *2.2%* |
| *rs-sbg* | *2.0%* | *2.2%* |
| *rs-progress* | *2.0%* | *2.2%* |
| *rs-layer* | *1.8%* | *2.0%* |
| *rs-mask-wrap* | *1.8%* | *2.0%* |

*図3.11. もっとも使用されているカスタムエレメント。*

しかし、Slider Revolutionは全サンプルページの約2%で使用されているというだけで、ほとんど参考になりません。

Slider Revolutionに含まれない、次に人気のあるカスタム要素とは？

82. *https://web.archive.org/web/20060203035414/http://code.google.com/webstats/index.html*
83. *https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements-core-concepts*
84. *https://www.sliderrevolution.com/*

| カスタム要素 | デスクトップ | モバイル |
|---|---|---|
| pages-css | 1.1% | 2.0% |
| wix-image | 1.1% | 2.0% |
| router-outlet | 0.7% | 0.5% |
| wix-iframe | 0.4% | 0.7% |
| ss3-loader | 0.5% | 0.5% |

*図3.12. `rs-` で始まらない、もっともよく使われるカスタム要素。*

これはより多様です。`pages-css`、`wix-image`、`wix-iframe`は、Wixウェブサイトビルダーからきています。`router-outlet`はAngularに由来するものです。そして、`ss3-loader`はSmart Sliderと関係があるようです。

## 廃止された要素

廃止された要素はまだあるのですか？検証しないことがまだあることを考えると、そうです。

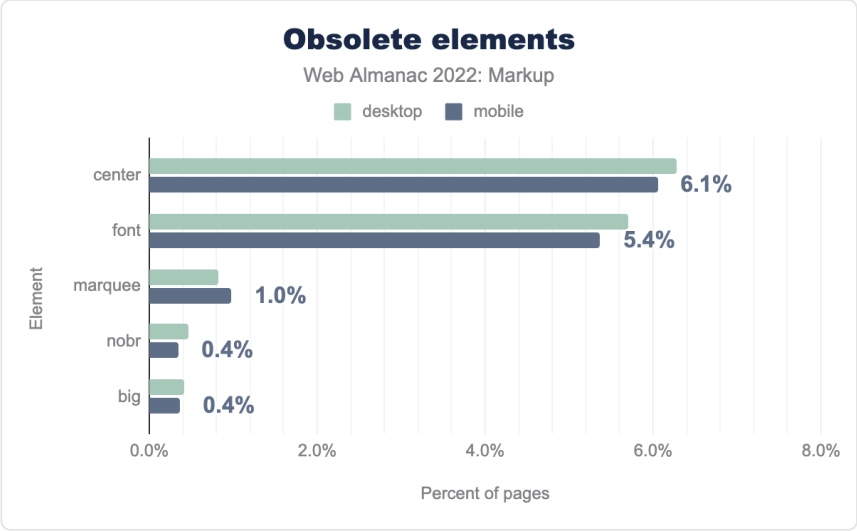



*図3.13. 廃止された要素。*

6.1%のページでまだ `center` 要素 (hi Googleホームページ[85]) が見つかり、5.4%のページで `font` 要素が見つかっています。幸いなことに、この2つの要素の使用率は下がりました（どちらも0.5%減）。一方、`marquee`、`nobr`、`big` には大きな変化は見られませんでした。

私たちの分析では、`center` と `font` が廃止された要素の大部分（81.2%）を占めています。

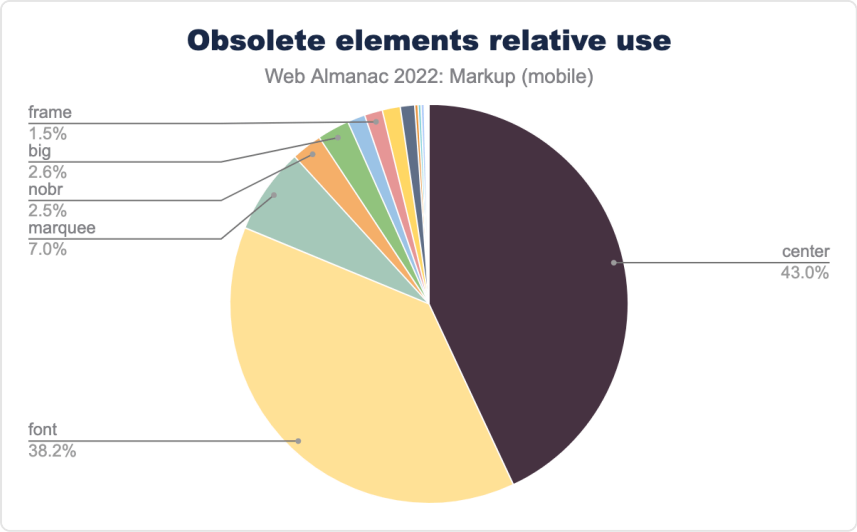



*図3.14. 相対的な使用を廃止した要素。*

## 属性

要素がHTMLのパンだとしたら、属性はバターです。ここで私たちは何を学ぶことができるでしょうか？

### トップ属性

もっとも人気のある属性は、昔も今も `class` です。

85. https://www.google.com/

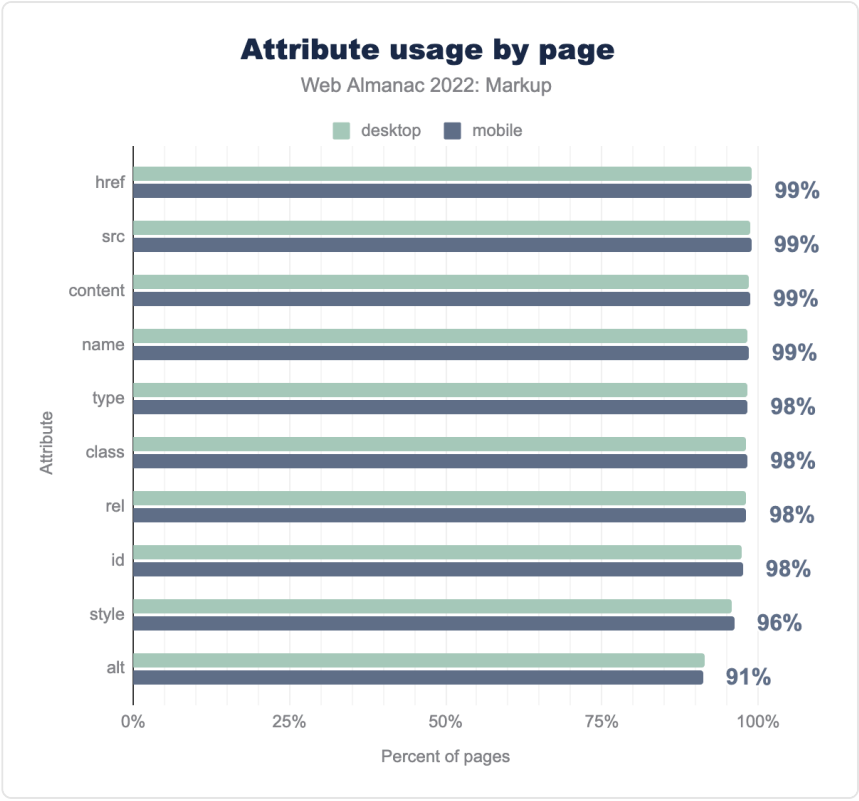



*図3.15. 属性の使い方。*

この順番は昨年までと変わらないのですが、いくつか変更点があります。

- `class` (▼0.3%)、`href` (▼0.9%)、`style` (▼0.6%)、`id` (▼0. 2%)、`type` (▼0.1%)、`title` (▼0.3%)、and `value` (▼0.5%) は、いずれも使用頻度が以前より少し減っています。
- `src` (▲0.3%) と `alt` (▲0.1%) は以前より多く使われており、アクセシビリティにとって比較的良いニュースです!
- `rel` の使い方は大きく変わっていません。

（ほぼ）すべての文書に見られる属性があるのでしょうか？はい、あります。

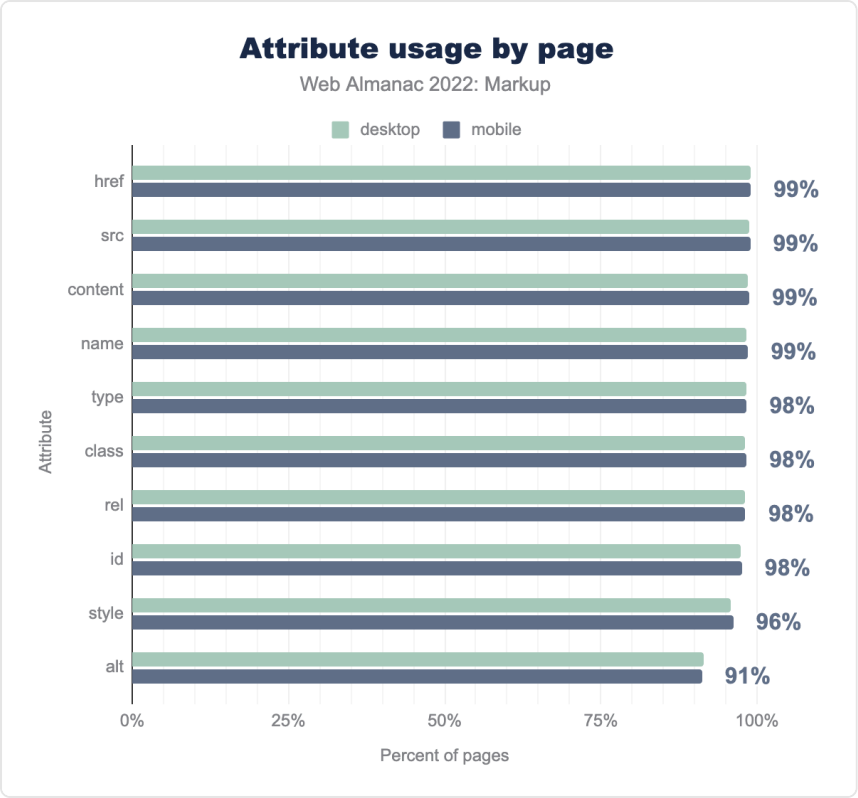



*図3.16. ページごとの属性の使用状況。*

`href`、`src`、`content`（メタデータ）、`name`（メタデータ、フォーム識別子）は、サンプルのほぼすべての文書に存在します。

## `data-*`属性

`data-*`属性（作者が独自のメタデータを埋め込むことができる属性）についても、新しい情報が引き出されました。

これは、昨年の`data-*`属性の統計 と比べると、ほとんど変化がありません。以下は、その変更点です。

- `data-id`は2021年と比較して0.7% 増加しており、もっとも人気のある`data-*`属性であることに変わりはありません。

- `data-element_type` は、順位は変わらないが、同じく0.7%上昇した。
- `data-testid` は以前6位でしたが、0.3%上昇し、4位に躍進しました。
- `data-widget_type` は8位で0.4%の人気を獲得し、さらに2つ順位を上げて2022年には6位を獲得しました。

`data-element_type` と `data-widget_type` は Elementor[86] に関連しており、一方data-testidは Testing Library[87] から来るものです。

それでは、`data-*` という属性がページ上でどの程度見られるか見てみましょう。

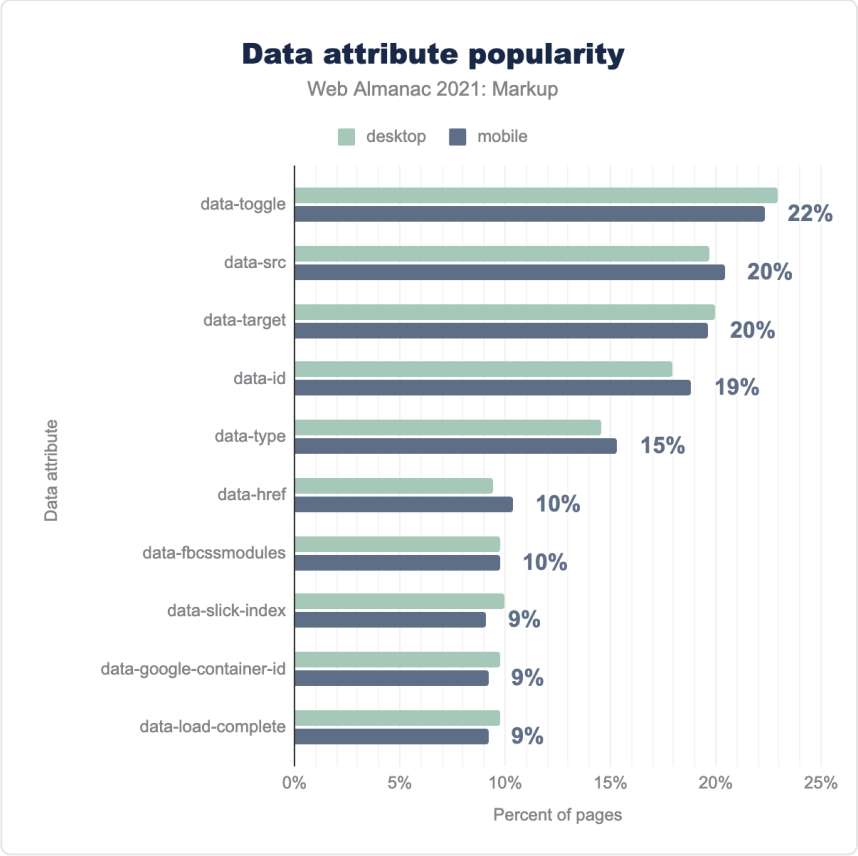



*図3.17. データ属性の人気度。*

86. https://developers.elementor.com/
87. https://testing-library.com/

その人気は高いです。上のグラフでは、4つめのドキュメントに `data-*` 属性が使用されています。しかし、全体のデータでは、88%のドキュメントが少なくとも1つの `data-*` 属性を使用しています。 これはかなりの普及率です。

## ソーシャルマークアップ

昨年の版では、ソーシャルマークアップのセクション[88]という、ソーシャルプラットフォームがそれぞれのメタデータを識別して表示することを容易にする特別なマークアップが紹介されました。ここでは、2022年のアップデート内容を紹介します。

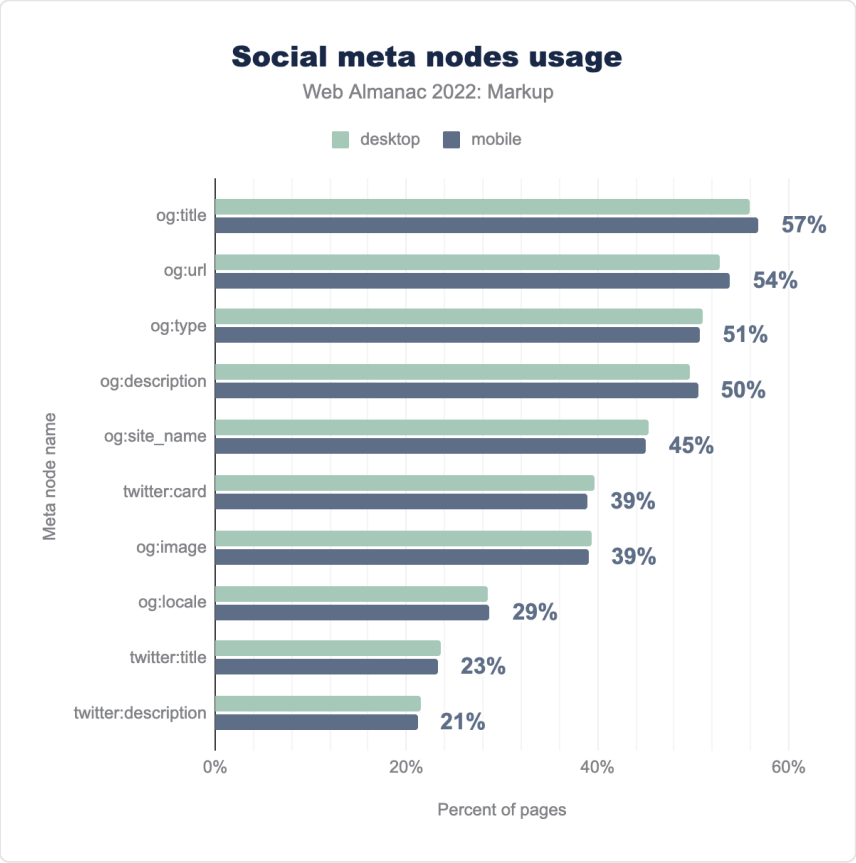



*図3.18. ソーシャルメタノードの利用状況。*

このメタデータはすべて必要なのでしょうか？それは、あなたの要件によります。しかし、

88. *https://almanac.httparchive.org/ja/2021/markup#ソーシャルマークアップ*

これらの要件がタイトル、説明、画像の表示に関するものであれば、ほぼ必要ないように思われます。`twitter:card`、`og:title`、`og:description` (標準の `description` メタデータにフックされる）、そして `og:image` でできるかもしれません。筆者や他の多くの人が、最小限のソーシャル マークアップ[89]のためのオプションを説明しています。

## 結論

これは、2022年のHTMLを一目見ただけです。

結論は簡潔です。年ごとに見ると、どのような重要なトレンドが始まったのか、あるいは逆転したのかはわかりません。少なくとも、2020年、2021年、2022年の3年間は、ドキュメントのサイズが大きくなり続けているようです。1ページの要素数も年々増えています。しかし、それは相対的なものであり、より多くの画像が適切な `alt` 属性を持つかどうか、またそのテキストが本当に有意義[90]であるかどうかは分かりません。

でも、そんなとき、Web Almanacが役に立ちます。来年も、再来年も、その次の年も、またHTMLについて見ていくつもりです。そしてまた、より詳細に、より多くの年を振り返っていくつもりです。

また、適合性にも目を向けることができるようになるかもしれません。私たちの分野では、現時点では誰もがこのことを気にしているわけではないかもしれません。しかし、私たちは皆プロフェッショナルであり、全体として基礎となる標準[91]に対応する作品を作っているかどうかを知ることは少なくとも関連性があると思われます。結局のところ、この章は空想のHTMLについての章ではなく、実際に機能するHTMLについての章であるべきなのです。もっとも重要なWeb標準の1つです。

### 著者

Jens Oliver Meiert

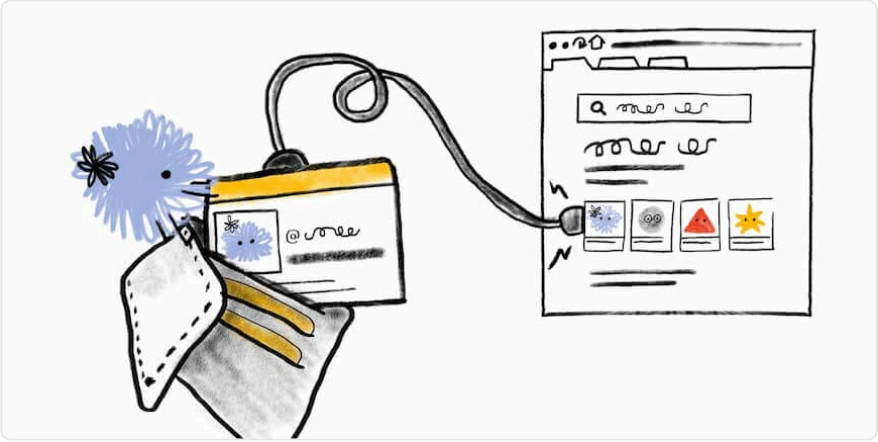
@j9t j9t https://meiert.com/en/

Jens Oliver Meiertは、エンジニアリングリードであり、著者(*The Web Development Glossary*[92]、*Upgrade Your HTML*[93])で、LivePerson[94]でエンジニアリング・マネージャーとして活躍する。専門はHTMLとCSSの最小化・最適化。Jensは、自身のウェブサイト meiert.com[95]で定期的にウェブ開発の技術について執筆しています。

89. *https://meiert.com/en/blog/minimal-social-markup/*
90. *https://html.spec.whatwg.org/multipage/images.html#alt*
91. *https://html.spec.whatwg.org/multipage/*

92. *https://leanpub.com/web-development-glossary*
93. *https://www.amazon.com/dp/B094W54R2N/*
94. *https://www.liveperson.com/*
95. *https://meiert.com/en/*

# 部 I 章 4
# 構造化データ



***Andrea Volpini と Allen ONeill によって書かれた。***
*Rob Teitelman と Jono Alderson によってレビュ。*
*Rick Viscomi による分析。*
*Jasmine Drudge-Willson 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

Web Almanacが構造化データに関する章を設けるのは今年で2年目です。昨年のコンテンツでは構造化データの概念について、その存在理由、もっとも頻繁に使用されるタイプ、そして構造化データが組織にもたらすメリットについて概説しその基礎を固めました[96]。今年は、2022年に収集したデータを昨年の前回データと比較することで、その間に発生した傾向を把握することができました。

機械学習、とくに「自然言語処理」の分野では多くの進歩が見られるが、データを機械的に読みやすい形式で表示する必要があることに変わりはない。構造化データは、ウェブ検索やデータ連携、アーカイブなどの情報発見を支援するものです。Webサイトに構造化データを導入することで、エンジニアやWebコンテンツ制作者は容易に構造化データを利用できるようになる。

96. *https://almanac.httparchive.org/ja/2021/structured-data#主要コンセプト*

- ウェブサイトデータをより広く公開し、自動的な発見とリンクを可能にする。
- 公共研究用データのオープン化
- 組織のデータの品質が、データが発信元を離れた後も維持されるようにすること

あらゆる規模や種類の組織が、自社のコンテンツをウェブ上で発見してもらいたいと考えています。GoogleやBingなどの検索エンジンは、構造化データの利用を促進することで、データの発見力を重視しています。SEOの観点からは、データを見つけやすく、解析しやすい形で提示することが有利です。これらの利点の一部は、本章の使用例と主要コンセプトのセクションで説明することにする。

昨年の紹介[97]では、「機械が構造化データを確実に、大規模な抽出ができるようになれば、新しくスマートなタイプのソフトウェア、システム、サービス、ビジネスが可能になる」と指摘されています。今年の章では、構造化データに関する最近発表された研究、オープンソースのフレームワーク、高品質な構造化データの生成を支援するツールなどを紹介します。

今年は、さまざまな構造化データ型の存在やそれらの構造化データ型の成長などの測定基準の最初の前年比を提供し、構造化データを使用することの進化した利点を検証しています。2021年からのデータのベースラインを持つことで、構造化データの利用がその間にどのように変化したかについての洞察を得ることができ、たとえばこの間のTikTokの成長など、興味深い傾向を観察できます。

## データの注意点

構造化データはさまざまな形で表示され、特定のドメインとそれに対応するWebサイトでは、他のドメインよりも目に付きやすい場合があります。たとえば、ニュースサイトとeコマースサイトを比較してみましょう。一般に、ニュースサイトでは、もっとも重要な最新ニュースがホームページに表示されます。したがって、ニュース記事に関連する構造化データは、個々の記事の見出しにデータスニペットとして付加されたメインサイトのランディングページに存在する可能性があります。これに対してeコマースにおける構造化データは、個々の製品に関連するため、ウェブサイトの製品カタログ自体に存在することがほとんどであり、ウェブサイトのメインナビゲーションやプロモーション部分のハイレベルな検索からは、多くの点で「見えない」状態になっています。これが、構造化データの章とレポートとの関連で意識しなければならない重要な注意点です。

ウェブサイトからデータを収集するために使用される技術は、サイトの表面（ホームページなど）をかすめるだけで、サイトの完全なクロールで深部まで行かないため、必然的にサイトの深部にデータが含まれるサイトでの構造化データの使用範囲の全体像を把握することが

97. *https://almanac.httparchive.org/ja/2021/structured-data#序章*

できません。将来的にはこの問題を解決し、構造化データのドメイン固有の使用に関するさらなる洞察を得るために、さまざまなドメインのサイトのサンプルを採取し、深く掘り下げていきたいと考えています。

昨年の章にあったハイレベルな注意点はまだ残っている。

- **自動生成された構造化データ**: これは、コンテンツ作成システムなどの技術が、テンプレートに基づいて構造化データスニペットを自動生成する場合です。この場合、テンプレートに基づくエラーは、表示されるすべてのデータで必然的に発生することになります。

- **データフォーマットの重複**: 構造化データは、JSON-LD、RDFなど、さまざまな方法で表示できます。たとえばFacebookのメタタグとRDFaセクションで、異なる方法で表示された同じタグの間で重複が、見られる可能性があることを意味します。2021年のベースラインに対して作成されたクエリに基づいて分析が行われるため、クリーニング/正規化およびデータの平坦化の影響は、同様の分析に引き継がれるものと思われます。

# 主要コンセプト

構造化データは豊かで複雑な分野であるため、分析に取りかかる前、このトピックの重要な概念をいくつか説明することが重要です。

## リンクされたデータ

ウェブページに構造化データを追加し、ページが含む/参照するエンティティへのURIリンクを提供することで、*リンクされたデータ* が作成されるのです。この構造化データは相互リンクされ、セマンティッククエリを通じてより有用なものとなる。

ウェブページの内容を記述するためにリンクされたデータを追加することで、機械がウェブページをデータベースとして扱うことができるようになります。大規模なものでは、セマンティック・ウェブ[98] に寄与しています。セマンティックウェブは、*リソース記述フレームワーク（RDF）* によってデータを結びつけています。これは、ウェブ上の情報を表現するためのフレームワークで、URIを使ってエンティティやその間の関係を定義します。

98. *https://ja.wikipedia.org/wiki/%E3%82%BB%E3%83%9E%E3%83%B3%E3%83%86%E3%82%A3%E3%83%83%E3%82%AF%E3%83%BB%E3%82%A6%E3%82%A7%E3%83%96*

RDFデータモデルにおけるエンティティ間の関係は、*セマンティックトリプル*として知られています。セマンティックトリプル[99]（または単に *トリプル*）で、データに関する文をコード化できます。これらの表現は、主語ー述語ー目的語の形式にしたがっている（たとえば、"アレンがジョンを知っている"）。

RDFデータを取得・操作するためには、標準的なRDF問合せ言語であるSPARQL[100]のようなRDF問合せ言語が使用できます。

後述するように、このセマンティックウェブは、ビジネスや技術に多くのチャンスをもたらす。

## オープンデータ

リンクされたデータは、*オープンデータ*として、*リンクされたオープンデータ* と表現されることもあります。オープンデータとは、その名の通り、誰でも、どんな目的でも、オープンにアクセスできるデータのことです。このデータは、オープンライセンスのもとでライセンスされています。

オープンデータは、Tim Berners-Leeが提案した展開スキームである5 stars of open data[101]の最初のものです。オープンデータ・ハンドブック[102]によると、最大5つ星を獲得するためには、(1)オープンライセンスの下ウェブ上で利用可能、(2)構造化データの形式、(3)非専有ファイル形式、(4)識別子としてURIを使用、(5)他のデータ源へのリンク（データ連携参照）であること、としています。

構造化データは5つ星オープンデータ計画の2つ目の星ですが、リンクデータはオープンデータの5つ星すべての要件を満たす必要があります。

## セマンティック検索エンジンとリッチリザルトとその先へ

セマンティック検索エンジンとは、セマンティック検索[103]を行うエンジンのことです。これは、検索エンジンが単語や文字列の完全一致や近傍一致を探す語彙検索とは異なるものです。セマンティック検索は、ユーザーの意図や検索語の文脈を理解し、検索の精度を向上させることを目的としています。たとえば、「ローカル ビジネス：美容院」の構造化データ・エンティティと「TG Locks n Lashes」の構造化データ・エンティティがあります。後者はビジネスネームであり、ヘアサロンというクリエイティブな名前をキーワードとして伝えていますが、検索エンジンがそのビジネスを理解するためにはほとんど役に立ちません。構造化データを使用することで、ウェブサイトは検索エンジンがその情報の文脈をより理解しやすくなり、その結果、検索ユーザーが尋ねたクエリの文脈に沿ったより良い検索結果を提供す

99. *https://en.wikipedia.org/wiki/Semantic_triple*
100. *https://www.w3.org/TR/sparql11-query/*
101. *https://5stardata.info/ja/*
102. *https://opendatahandbook.org/*
103. *https://en.wikipedia.org/wiki/Semantic_search*

ることができるようになります。GoogleとBingは、セマンティック検索エンジンの優れた例です。

Googleはセマンティック検索技術を使って、検索結果をインフォボックスで提供するために使われる知識ベースであるGoogleナレッジグラフ[104]から関連情報を提供しています。このインフォボックスは、ナレッジパネル[105]として知られており、多くの結果で見ることができます。このナレッジボックスは、構造化データによって有効化または強化できます。

構造化データとリンクデータを組み合わせることで実現するもう1つの検索結果が、リッチリザルト[106]です。これらの結果は、イベント、FAQ、ハウツー、求人情報、より一層[107]などの形で、検索結果にリッチな機能を表示します。構造化データを実装してWebページをリッチリザルトの対象にすることで、クリック率を高めることができる[108]。下の画像は、ヘアースタジオのビジネスの詳細を構造化データにすることで、検索エンジンがそのビジネスの情報を簡単に抽出して表示し、ハイライトしてSEOを最適化できることを示しています。

104. *https://blog.google/products/search/introducing-knowledge-graph-things-not/*
105. *https://support.google.com/knowledgepanel/answer/9163198*
106. *https://developers.google.com/search/docs/appearance/structured-data/search-gallery?hl=ja*
107. *https://developers.google.com/search/docs/appearance/structured-data/search-gallery?hl=ja*
108. *https://www.searchenginejournal.com/how-important-is-structured-data/257775/*

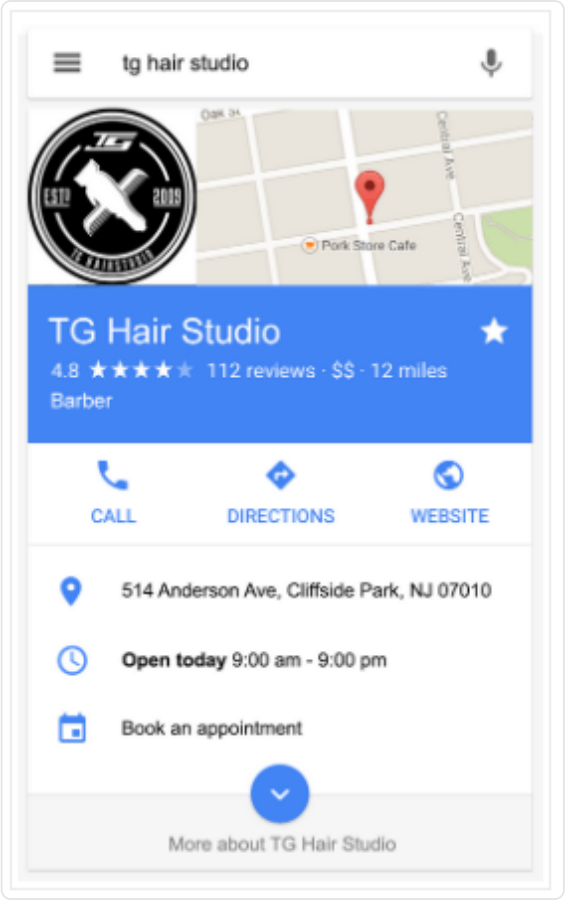



*図4.1. ウェブ検索で浮上した構造化データ。*

ナレッジ パネルやウェブページのリッチな結果を超えて、構造化データは、検索におけるファクト クエリ[109]への回答を可能にすることも可能です。事実に基づくクエリ検索は、異なる構造化データソースから複数のシグナルを取得し、より正確な結果をサポートできます。ここで構造化データの実装とそれを可能にするテクノロジーは、ユーザー体験を向上させるために、より速く、より信頼性の高い情報へのアクセスを提供します。

SEOの重要性、クリック率の向上、ユーザー体験の改善、解析のための機械読み取り可能なデータという組み合わせは、構造化データの実装による大きな利点を示しています。これらの重要な概念を理解することは、コンテンツプロバイダーとサイトを構築する技術者の両方にとって、より良いナビゲーションの実装方法と、ウェブページからの自動データ取得の機

109. *https://gofishdigital.com/blog/answering-questions-structured-data/*

能を理解するのに役立ちます。

## 構造化データ研究

今回は構造化データの分野でどのような学術研究が行われているのか、また、構造化データが最先端の技術やサービスの開発に役立っているという記録があるのか、調査してみました。

発表された研究を探すために、Google Scholar[110]、ConnectedPapers[111]および大学ベースの引用データベースなどの学術検索ツールを使用しました。最近の出版物だけでなく、引用され続けている古い研究も探しました。

検索の結果、構造化ウェブデータの生成、管理、構築に関して実施された、引用度の高い最近の研究はあまりないことがわかりました。しかし、ナレッジグラフ、レコメンデーションエンジン、情報検索、チャットボット、説明可能なAIなどの構造化ウェブデータ（“セマンティック・ウェブ”[112]）の応用に関する研究は、過去12か月間に行われ、今もなお増え続けています。

ウェブ構造化データは、機械可読ラベル[113]の生成に使用できる適切な統一参照指標（URI）語彙を持つ一貫したデータを提供することによって、機械学習の分野と相乗的な関係を共有しています。私たちの検索や背景を読むと、構造化データは、機械学習アルゴリズムを学習するための高品質なウェブデータを生成するための作業と入力時間をかなり削減していることがわかります。

実用的なレベルでは、構造化データが改善された3つの分野を取り上げます。

- ナレッジグラフ
- 知識グラフを用いた質問応答
- 説明可能なAI

### ナレッジグラフ

構造化されたウェブデータは、エンティティやオブジェクト間の固定語彙をドメイン固有の言語として提供し、一般にRDF形式で保存される。RDFを用いた知識グラフは、エンティティ間の関係を照会するための優れたツールであることが証明されている。たとえば、

110. *https://scholar.google.com/*
111. *https://www.connectedpapers.com/*
112. *https://ja.wikipedia.org/wiki/%E3%82%BB%E3%83%9E%E3%83%B3%E3%83%86%E3%82%A3%E3%83%83%E3%82%AF%E3%83%BB%E3%82%A6%E3%82%A7%E3%83%96*
113. *https://developers.google.com/machine-learning/crash-course/framing/ml-terminology?hl=ja*

Wikidated 1.0は、Wikipediaの改訂履歴を保存するためにWeb構造化データを使用した進化する知識グラフです。その対応する論文[114]は、RDFタプルの追加と削除のセットとしてページの改訂を集約するプロセスについて話しています。著者らは、wikipediaのダンプを知識グラフに変換する方法をオープンソースで公開しています。doordashエンジニアリングによって行われた応用研究では、知識グラフを使うことで検索性能が劇的に向上する[115]ことが実証されています。

## 知識グラフを用いた質問応答

質問応答システムは、エンドユーザーが自分の質問に対する答えを見つけることを可能にする。知識グラフの上に構築された質問応答システムは、知識グラフに格納された豊富で構造化されたデータへアクセスすることを可能にします。知識グラフにRDFトリプルとして格納された情報を問い合わせるために、SPARQL[116]などの問い合わせ言語がしばしば使用される。

しかし、SPARQLのクエリを書くことは、エンドユーザにとって退屈で困難な場合がある。そこで、自然言語質問（NLQ）は、知識グラフへの問い合わせにおける数々の複雑さを克服することができる魅力的なソリューションです。本研究では、2つの主要なフェーズからなるKGベースの質問応答システム（KGQAS）を提案する。1) オフラインフェーズと、2) 意味解析フェーズです。

オフラインフェーズでは、自然言語の質問を半自動で正式なクエリーパターンに変換することを目的とし、セマンティックパージングフェーズでは、機械学習を活用して予測モデルを構築する。このモデルは、最初のフェーズの出力に対して学習されます。これにより、与えられた質問に対してもっとも適切なクエリパターンを予測することができる。評価のために、SalzburgerLand KGを実際のユースケースとして使用する。これはスキーママークアップボキャブラリーを用いて構築された実世界の知識グラフであり、その主な焦点はオーストリアのザルツブルク地域の観光エンティティを記述した構造化データの自動化です。

## 説明可能なAI

Explainable AIは、AIモデルの意思決定を説明することに重点を置いています。もっとも、AIモデルは一般に公開されていないため、その判断の根拠を示すことはできません。セマンティックウェブの上に構築された知識グラフのおかげで、エンティティ間の見つけにくい関係を見つけることができます。これらは、モデルの結果を遡るための「グランドトゥルース」として使用されます。もっとも一般的なアプローチは、ネットワーク入力やニューロンをオントロジーのクラスやウェブ構造化データのエンティティにマッピングすることです。

114. *https://arxiv.org/abs/2112.05003*
115. *https://doordash.engineering/2020/12/15/understanding-search-intent-with-better-recall/*
116. *https://ja.wikipedia.org/wiki/SPARQL*

*参考文献:*

- ナレッジグラフ: Wikidated 1.0: ウィキデータの改訂履歴の進化する知識グラフ・データセット[117]。
- 知識グラフを用いた質問応答: 知識グラフを用いた質問応答。観光産業におけるケーススタディ[118]。
- 構造化データを用いた説明可能なAI: 説明可能な機械学習モデルのためのセマンティックウェブ技術: 文献レビュー[119]

## 構造化データのオープンソース利用

構造化データの利用に大きく依存している注目のプロジェクトは、以下の3つです。

- **オープンソース・メタデータ・フレームワーク(OMF)** - OMFは、オープンソースドキュメントに関するデータ/メタデータを収集することを目的としており、通常、ドキュメントの説明に使用される構造化されたデータ形式で保存されます。OMFは、存在する多くのオープンソースドキュメントプロジェクトのための洗練されたカードカタログのようなシステムとして機能することが期待されています。
- **DBpedia** は、構造化されたウェブデータに関連するデータセット、ツール、サービスの集合体です。現在までに2億2,800万以上の自由利用可能な実体が含まれています。メインのDBpediaナレッジグラフは、ウィキペディアのクリーンなデータを包含しています。DBPediaは、サポートされているすべてのWikipediaの言語で利用可能で、年間平均600kファイル以上のダウンロードがあります。DBpediaの上に構築されたいくつかのオープンソースツールは、データアクセス、バージョン管理、品質管理、オントロジー可視化、リンクのインフラストラクチャを提供します。
- **Wikidata** は、WikipediaなどのWikimediaプロジェクトの構造化データを格納しています。ドキュメント指向のデータベースで、構造化されたウェブデータを格納することに重点を置いています。

117. *https://arxiv.org/abs/2112.05003*
118. *https://ieeexplore.ieee.org/abstract/document/9810255*
119. *https://www.researchgate.net/profile/Matthias-Pfaff/publication/336578867_Semantic_Web_Technologies_for_Explainable_Machine_Learning_Models_A_Literature_Review/links/5daafb99a6fdccc99d91d120/Semantic-Web-Technologies-for-Explainable-Machine-Learning-Models-A-Literature-Review.pdf*

# 使用例

構造化データの実装は、多くの分野で広く有益であるが、そのうちのいくつかにこのセクションで焦点を当てることにする。これらの分野の多くは、リンクデータおよび構造化データの性質上、重複していることに注意することが重要です。

## データ連携

データを構造化しリンクさせることで、場所、イベント、人、コンセプトなどを識別子を用いて指定しながら、他のデータソースからデータを引用することができ、その結果、メタデータの記述にアクセスしやすくなる。そして、このデータはより共有しやすく、再利用しやすくなる。

データリンクにより、異なるソースから情報を収集し、より豊かで有用なデータを作成することができる。これは構造化データのおかげであり、そのグローバルでユニークな識別子によって、機械は異なるタイプのデータの関係を読み取り、理解することができる。これにより、よりつながりのあるウェブ上の関係を作り出すことができるのです。

## 検索エンジン最適化＆ディスカバリー性

検索エンジン最適化（SEO[120]）は、検索エンジンからより良い結果を得られるように、ウェブページのコンテンツを構築することに焦点を当てた分野です。SEO対策がうまくいけば、検索エンジンの検索結果ページ（SERP[121]）で上位に表示される可能性があるので、当然、これは発見性にとって非常に重要なことです。SERPは、検索クエリからタイトル、URL、メタディスクリプションが表示される場所です。

ウェブページに構造化データを追加することで、ウェブページを検索エンジンに最適化するとともに、SERPから見えるように余分なコンテンツを追加できます。この追加コンテンツはさまざまな形で提供され、そのうちのいくつかは以前ナレッジパネル、リッチスニペット、関連質問で議論されています。

検索エンジンからWebページへのトラフィックを増やすには、構造化データによる発見力の向上が欠かせません。そのため、企業やeコマースページは、これらの技術に大きな価値を見出すことができるだろう。

## Eコマース＆ビジネス

Eコマースのウェブページに構造化データを実装することは、ビジネスに関わる人々にとっ

120. *https://www.webopedia.com/definitions/seo/#How_does_SEO_work*
121. *https://www.webopedia.com/definitions/serp/*

て非常に有益なことです。これらのビジネスでSEO対策に広く利用されている構造化データ型は多数あります。

ローカル ビジネス[122]は構造化データ型で、関連する検索クエリ（例：「ダブリンの人気レストラン」）の際に構造化データ型へ入力した詳細をGoogleナレッジパネルに返す可能性があります。このタイプはまた、営業時間、ビジネス内の異なる部署、ビジネスに対するレビューを持つ可能性があり、これらはすべてマップアプリの検索クエリから同様に返される可能性があります。

商品[123]は構造化データ型で、検索クエリで豊富な結果を返すことができるという点で、LocalBusinessと同様に機能する。これらの結果には、価格、在庫状況、レビュー、評価、そして画像さえも検索結果に含めることができます。これらの追加要素により、製品が検索から注目される可能性がはるかに高くなります。製品属性は、製品同士を結びつけ、検索クエリに適切に対応することで、発見力を高めることができます。

これらは、eコマースにおける構造化データの使用例のほんの一例です。eコマースページが実装することで利益を得られる構造化データ タイプは、他にもたくさんあります[124]。

## 1年を振り返って

構造化データは、出版社が事前に定義された方法でデータを適合させ、提示することができるメタレベルのスキーマを記述するフォーマットと標準によって支えられています。本章の分析では、RDFa、OpenGraph、JSON-LD、その他の確立されたフォーマットが使用されています。

122. *https://developers.google.com/search/docs/appearance/structured-data/local-business?hl=ja*
123. *https://developers.google.com/search/docs/appearance/structured-data/product?hl=ja*
124. *https://developers.google.com/search/docs/specialty/ecommerce/include-structured-data-relevant-to-ecommerce?hl=ja*

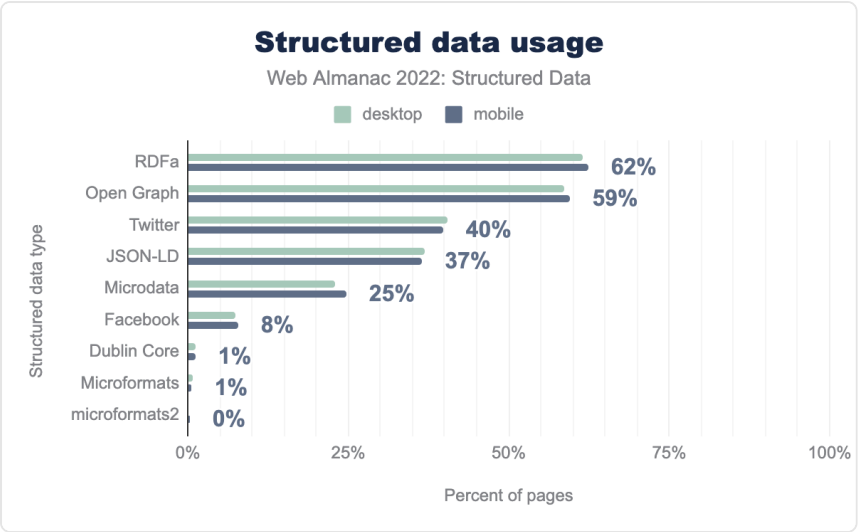



*図4.2. 構造化データ型*

RDFaとOpen Graphは、それぞれモバイルページの62%と57%を占めており、依然として大多数を占めています。構造化データ型はモバイルとデスクトップのページで一貫して見られますが、Microformatsとmicroformats2はこの章で調べた他の構造化データ型ともっとも異なっています。Microformatsはモバイルページで86%著しく、一方microformats2はモバイルページで171%著しくなっています。これらの2つの構造化されたデータ・タイプは、私たちのセットで見つかったものの小さな割合を構成しています。

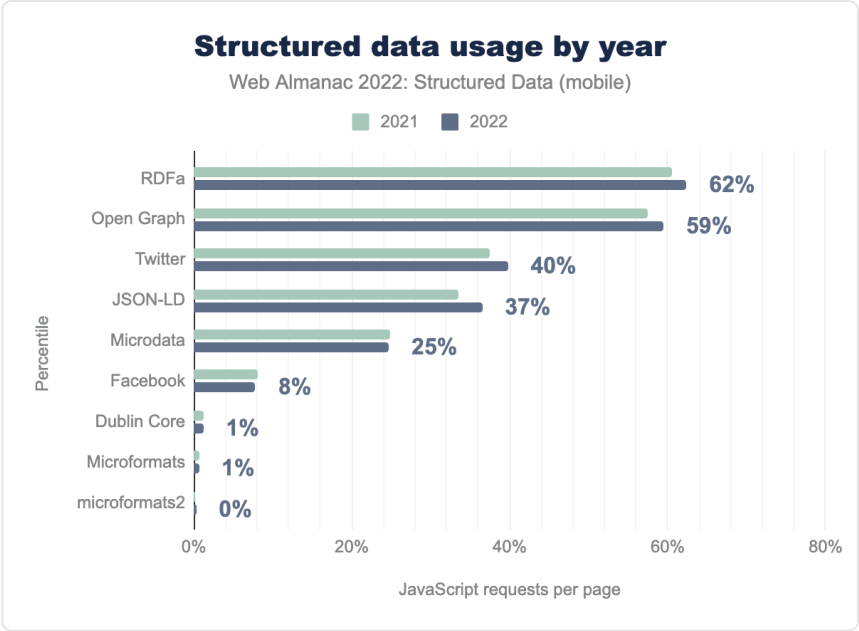



*図4.3. モバイルにおける年別の構造化データ利用状況*

Twitterメタタグ（37%から40%に増加）やJSON-LD（2021年の全体34%から2022年の全体37%にカバー率が増加）など、これらの広く使われている構造化データ型の全般的な増加が見て取れる。Microdata、Facebook meta tags、Dublin Core、Microformatsなど、あまり普及していない構造化データ型については、利用率がやや減少しています。デスクトップの動きも非常に似ています。

以下の表は、過去1年間に構造化データ形式が変更された主なものを示しています。変更のあったタイプのみ記載しています。

| データ型 | 変更 |
| --- | --- |
| RDFa | RDFa の基本フォーマットに変更はありませんが、データカタログ用語集 (DCAT) のバージョン 3 には重要な更新が含まれていました。DCAT は、「ウェブ上で公開されたデータカタログ間の相互運用性を促進するために設計された RDF ボキャブラリー」です。これは、ウェブ上のオープン データセットの可用性が高まっているため、重要な意味を持ちます。データセットの内容全体を記述できるようになると、公共データセットの発見性、ひいては有用性が大幅に高まり、連携した検索や配布がより可能になるのです。<br>参考文献:<br>• DCAT: https://www.w3.org/TR/2022/WD-vocab-dcat-3-20220510<br>• Google データセット検索エンジン[125]<br>• Google データセット 構造化データ形式ガイド"[126] |
| JSON-LD | 過去1年間の更新・追加内容は軽微なものであった。もっとも、「Organizationのサブタイプとして OnlineBusiness を追加し、OnlineBusinessのサブタイプとして OnlineStore を追加した」というように、メンテナンスとコンテキストの小さな拡張に関連するものであった。<br>参考文献:<br>• https://schema.org/docs/releases.html |

*図4.4. データ型フォーマットの2021年と2022年の間の変更点。*

全体としては、表にあるように主要なデータタイプの定義にほとんど変化はないが、特定のドメインでいくつかのフォーマットが進化している。

それぞれのタイプについて、もう少し掘り下げてみましょう。

125. https://datasetsearch.research.google.com/
126. https://developers.google.com/search/docs/advanced/structured-data/dataset

## RDFa

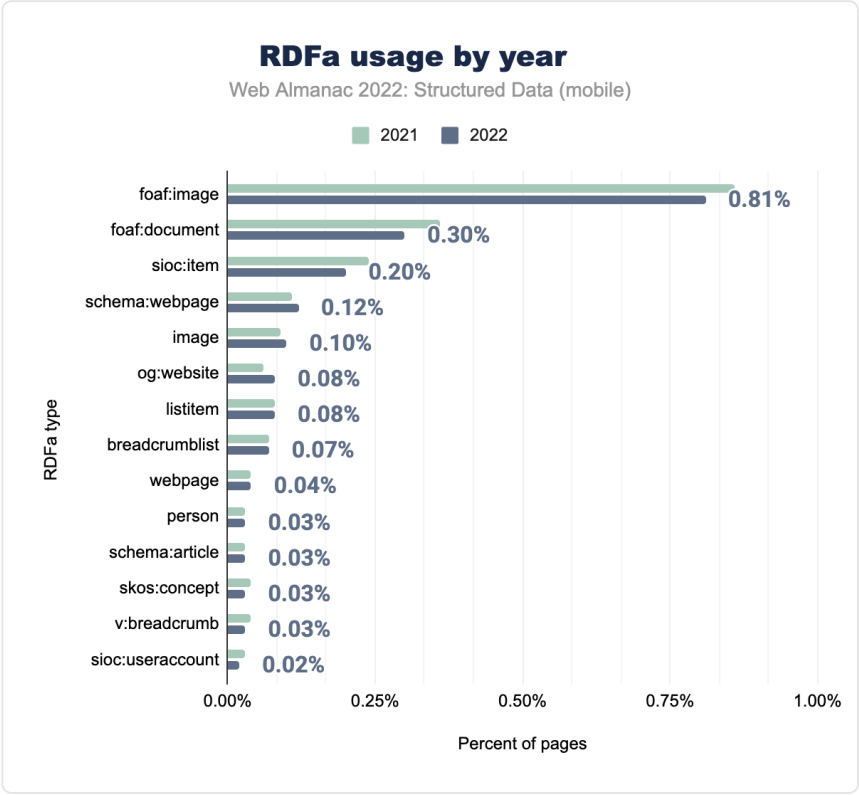



*図4.5. モバイルでのRDFaの年別利用状況*

RDFaのタイプを評価すると、`foaf:image`は他のどのタイプよりも多くのページに存在していますが、2021年以降、セットに含まれるページの割合が減少していることがわかります。これは次の2つのタイプ、`foaf:document`と`sioc:item`にも当てはまり、使用率がわずかに減少しています。他の多くのタイプは、RDFaが全体として見ているように、使用率がわずかに増加していることがわかります。

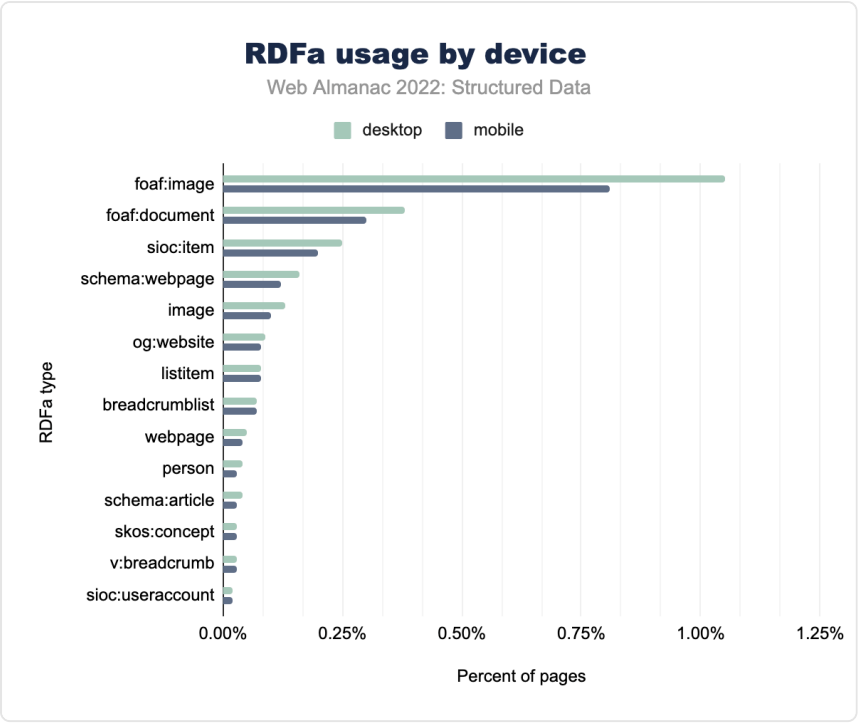



*図4.6. デバイス別のRDFa利用状況*

RDFaはデスクトップでより顕著で、`foaf:image`はモバイルページの0.81%に対し、デスクトップページの1%に出現しています。その他のRDFaタイプは、モバイルよりもデスクトップページでの出現率がわずかに上昇しましたが、`og:website`は例外で、モバイルページでは0.08%、デスクトップページは0.07%で先に到達しています。

## ダブリンコア

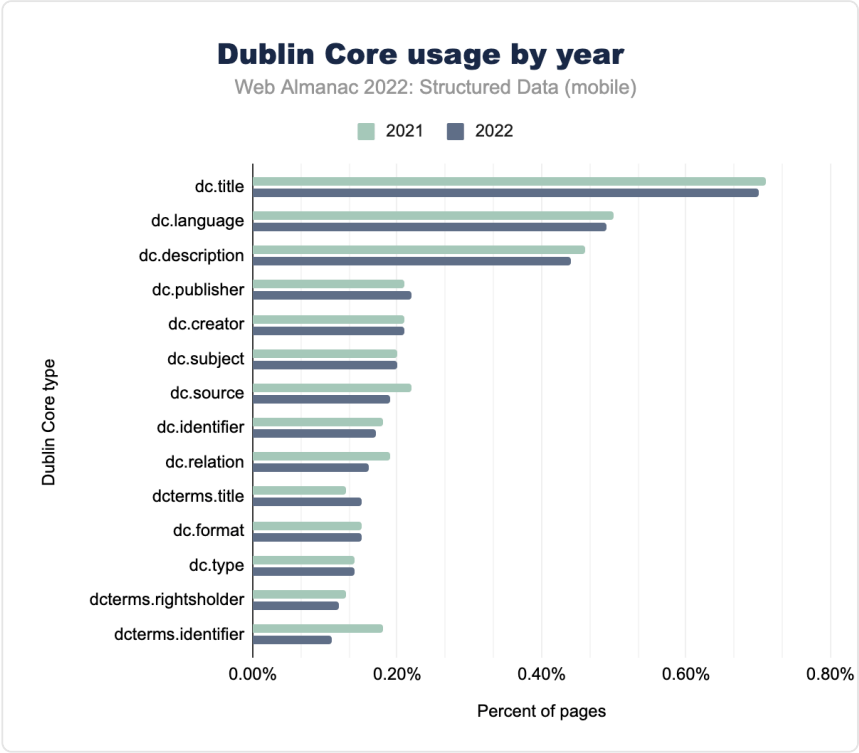



*図4.7. 年別のダブリンコア利用状況（モバイル）*

ダブリンコアの属性タイプの使用率は、もっとも顕著な属性タイプで非常によく似たままです。注目すべき例外は`dcterms.identifier`で、2021年の0.11%から2022年にはモバイルページで0.18%に増加しています。割合としては小さいですが、これは私たちのセットで合計すると約15,000の使用回数になります。この増加は、デスクトップ・ページでも見られましたが、それほど大きくはなく、2021年の0.14%から2022年の0.18%になりました。

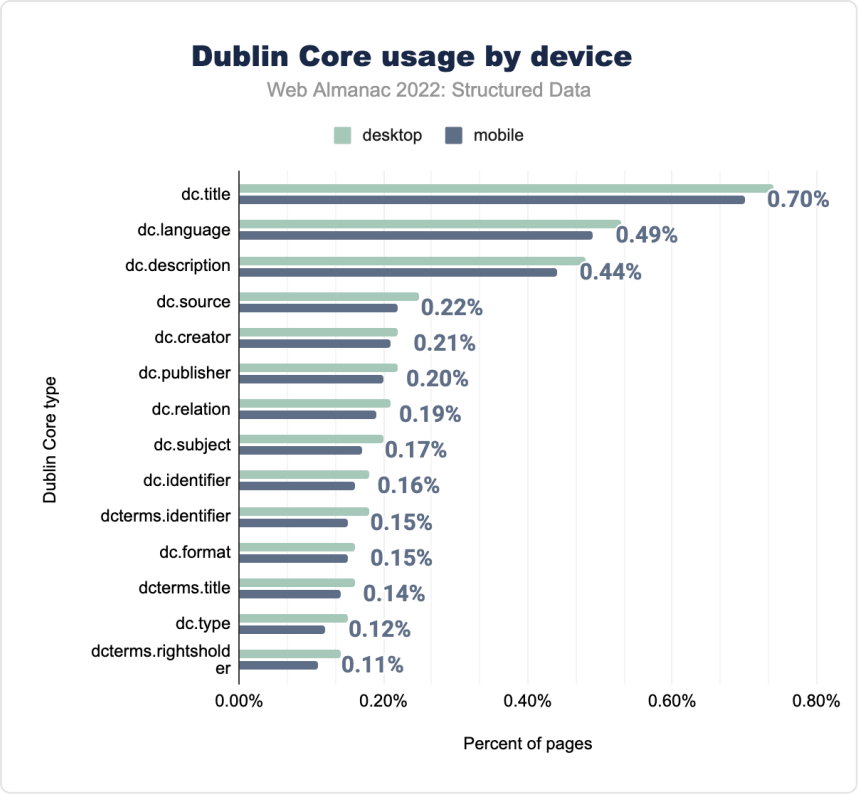



*図4.8. デバイス別ダブリンコア利用状況*

その他、ダブリンコアのタイプはモバイルページとデスクトップページでほぼ同じで、前年と比較して出現率が若干増加しているのは共通しています。

## オープングラフ

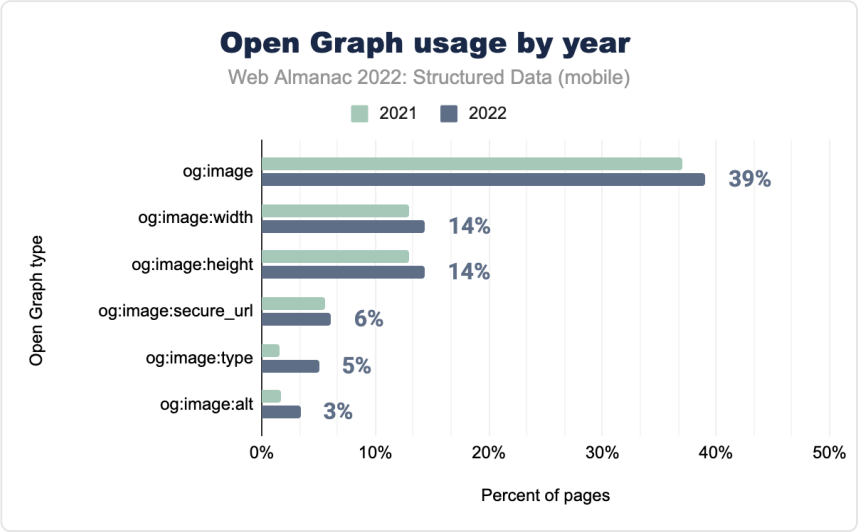



*図4.9. オープングラフの年度別利用状況（モバイル）*

Open Graphタグは、広く使用されるようになりました。これらのタグのうちもっとも一般的なのは、`og:title`であり、`og:url`、`og:type`とともに、セット内の全ページの半数以上に出現しています。もっとも多くの増加幅は小さく、例外的に`og:image:type`は2021年以降、モバイルページで3倍以上になっています。これはデスクトップでも同様で、1年の間に1.6%から5.4%になりました。

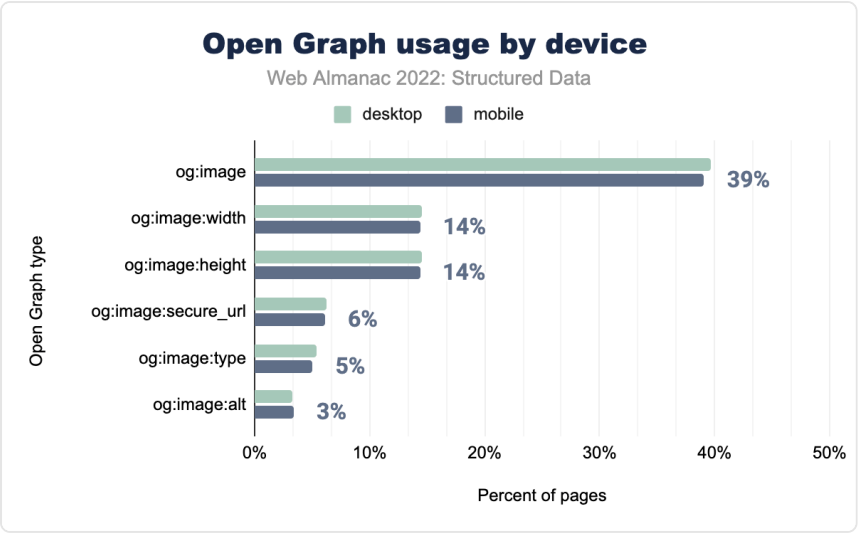



*図4.10. デバイス別のOpen Graph利用状況*

モバイル、デスクトップともに上位10位までの各Open Graphタイプの利用が増加し、2021年以降のOpen Graphの相対的な成長率は1.5%という結果になりました。

## Twitter

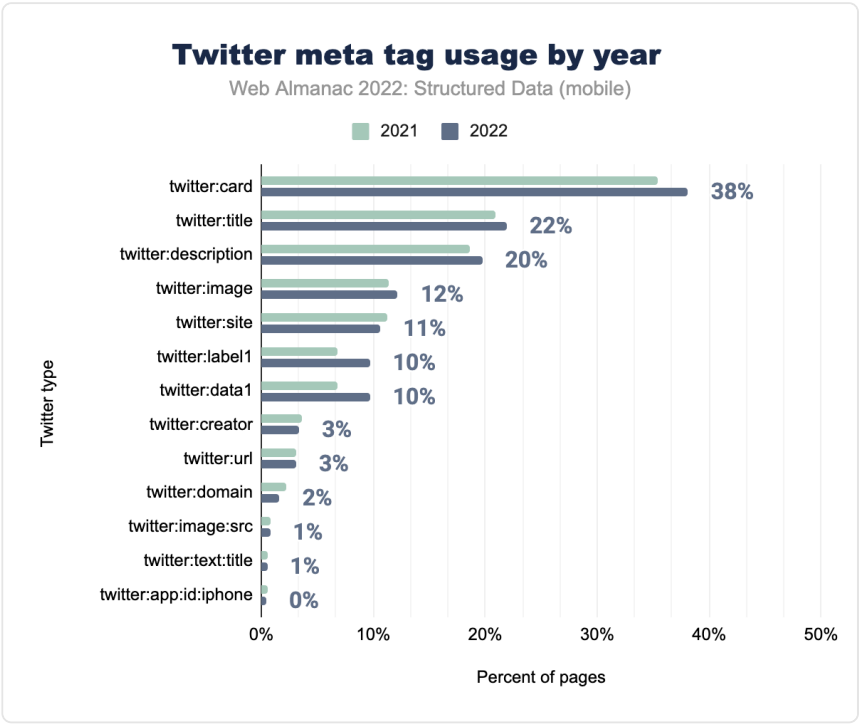



*図4.11. Twitterメタタグの利用状況（年別）（モバイル）*

Twitterのメタタグは再び、一般的な使用率の増加パターンにしたがっており、とくに `twitter:card`、`twitter:title`、`twitter:description`、`twitter:image` の共通タグで増加しています。注目すべきは、`twitter:label1` と `twitter:data1` で、ともに2021年の7%から2022年にはモバイルページで10%に増加していることが確認できます。

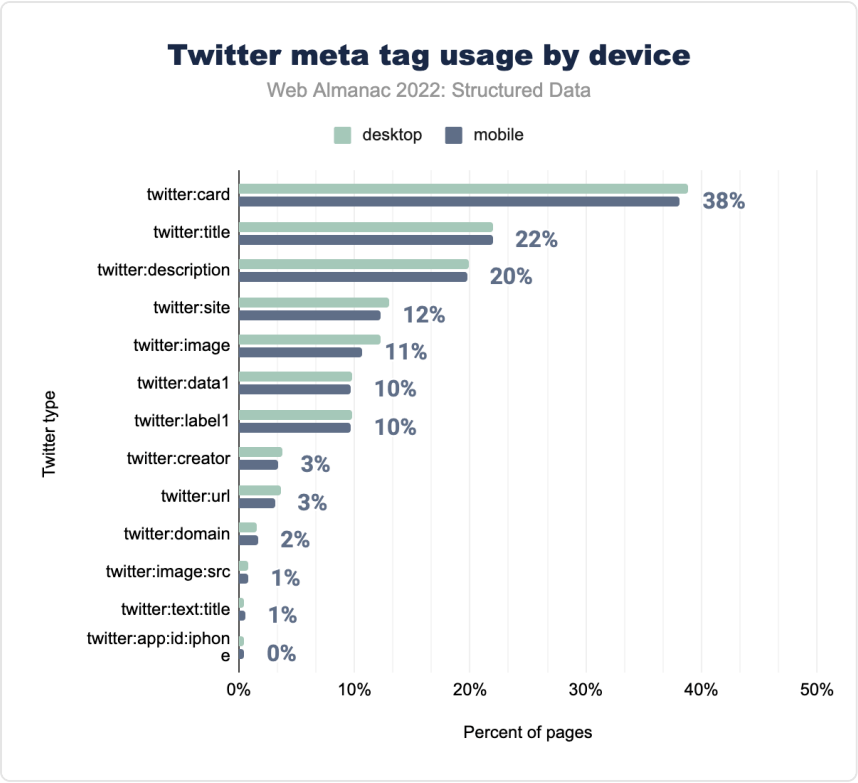



*図4.12. Twitterのメタタグのデバイス別利用状況*

`twitter:site` や `twitter:image` などのTwitterメタタグは、デスクトップサイトでより大きな存在感を示していますが、これらのメタタグの大半はモバイルとデスクトップの間、また前年比で同じ普及率を示しています。あまり一般的でないタグの中には、今年、使用率がわずかに減少したものもありますが、Twitterのメタタグの使用率は昨年から全体的に増加を維持しています。

## Facebook

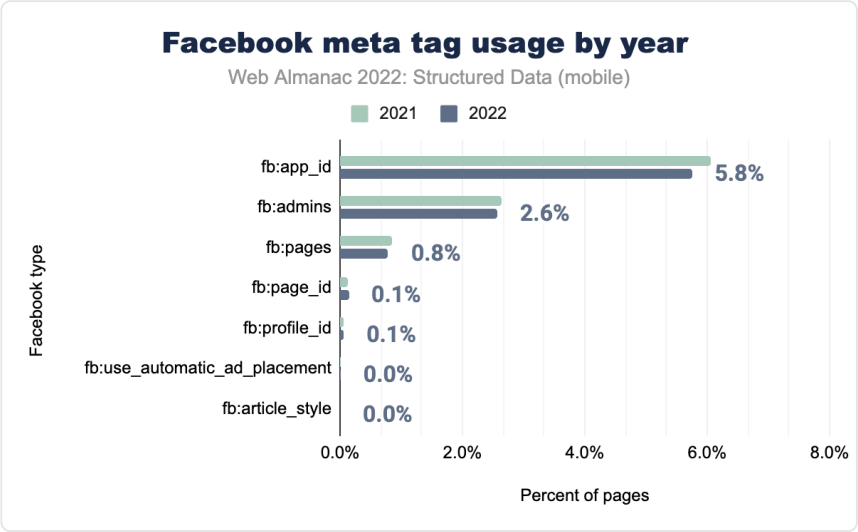



*図4.13. Facebookメタタグのモバイル利用状況（年別）*

ここにあるすべてのFacebookタグのうち、出現数が多いのは3つだけです。これらは2021年の上位3つと同じで、`fb:app_id`が5.8%、`fb:admins`は2.6%、モバイルでは`fb:pages`が0.8%と、いずれも昨年より微減しています。

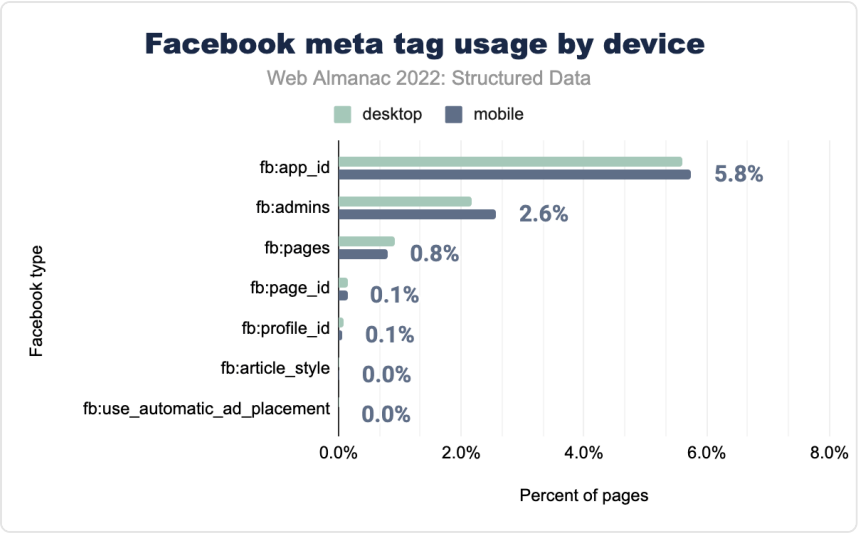



*図4.14. Facebookメタタグのデバイス別利用状況*

これはデスクトップページでも同様で、例外として `fb:pages` は2021年の0.90%から2022年には0.92%に微増することが分かっています。メタタグ `fb:pages_id` はモバイルページ、デスクトップページともに微増ですが、facebookメタタグの使用率は昨年からモバイルページ、デスクトップページともに全体として減少しています。

## マイクロフォーマットとマイクロフォーマット2

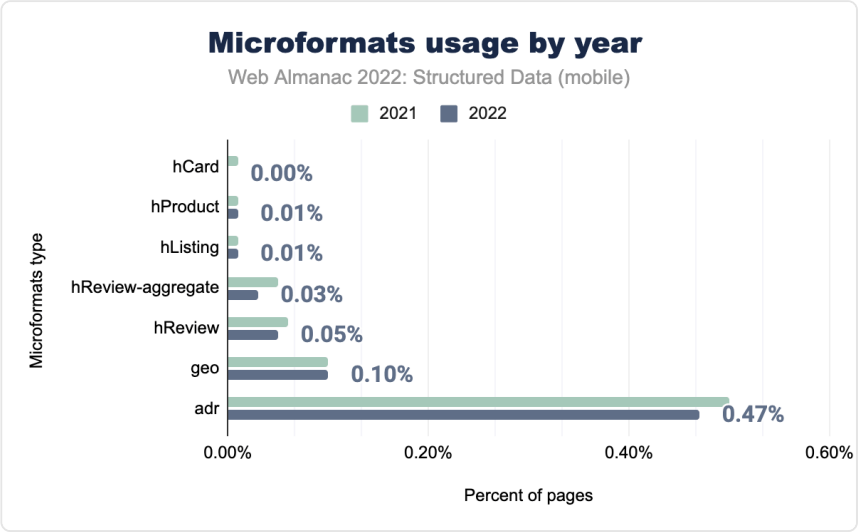



*図4.15. マイクロフォーマットのモバイル利用状況（年別）*

マイクロフォーマットは、2021年以降、使用法の数はもっとも一般的なもので、`adr`（私たちのセットの0.47%のページに出現）は、依然としてリストの中でもっとも一般的なものです。

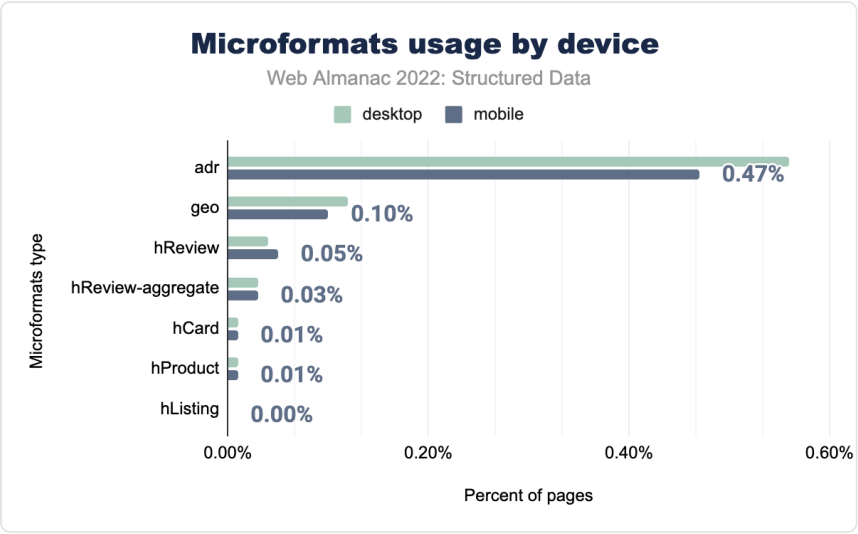



*図4.16. デバイス別のマイクロフォーマットの使用状況*

モバイルとデスクトップともに、マイクロフォーマットの種類によって利用が増加したり減少したりしますが、どちらも平均すると昨年の数値より少なくなっています。減少の要因となったのは、`hReview`（モバイルページで0.06%から0.05%、デスクトップページで0.06%から0.04%）と`hReview-aggregate`（モバイルとデスクトップの両方で0.06%から0.04%）のタイプでした。

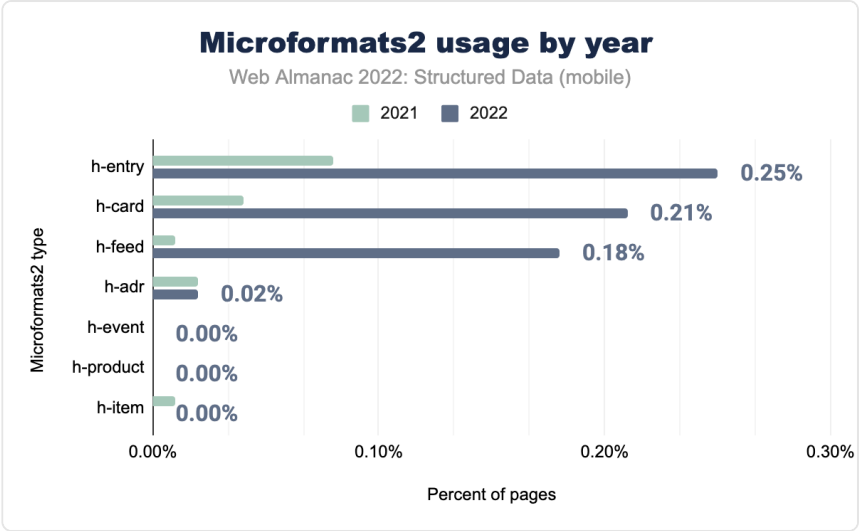



*図4.17. マイクロフォーマット2の年別使用状況*

一方、2021年以降、マイクロフォーマット2の属性は急増しています。`h-entry`、`h-card`、`h-feed`のプロパティは、我々のページのセットで大きな増加を示しており、これはマイクロフォーマット2の属性が前年からほぼ3倍になったことを説明するものです。

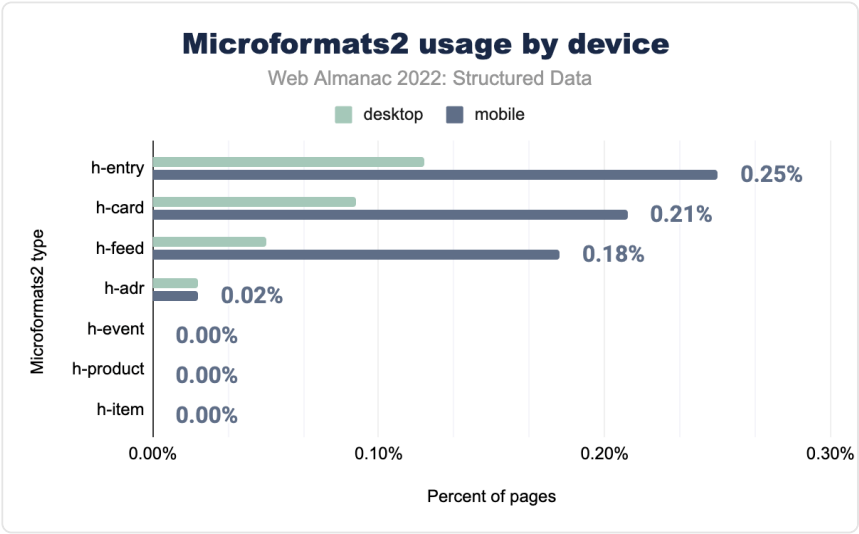



*図4.18. マイクロフォーマット2のデバイス別利用状況*

この増加は、モバイルページでより顕著に見られますが、デスクトップページも同じパターンを辿っています。それ以外では、`h-adr`は両年、両プラットフォームとも0.02%でまったく変わりません。

## マイクロデータ

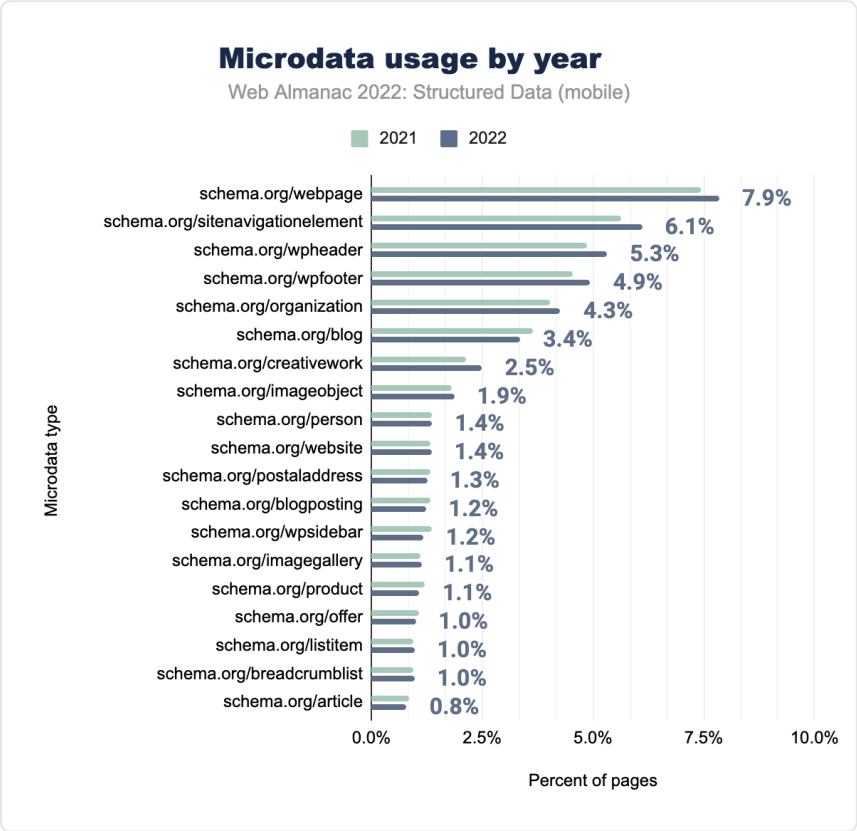



*図4.19. マイクロデータのモバイル利用状況（年別）*

マイクロデータのプロパティのほとんどはあまり変化していませんが、`webpage`、`sitenavigationelement`、`wpheader`といったより一般的なプロパティは、それぞれモバイルページの7.9%、6.1%、5.3%に表示されており、若干の増加が見られます。

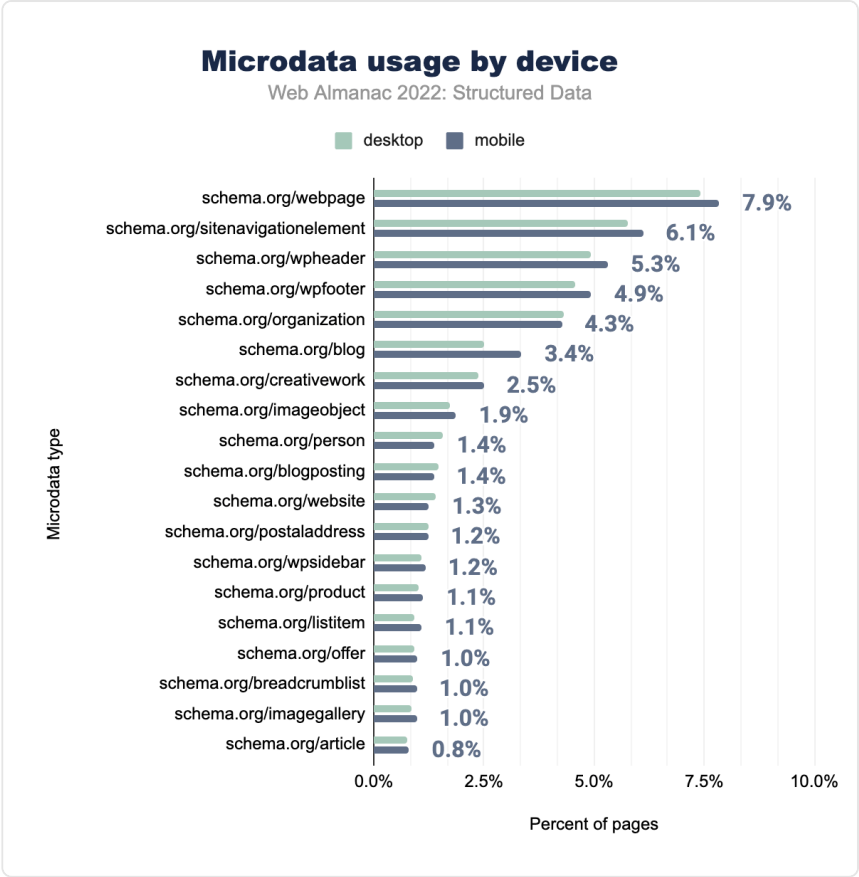



*図4.20. マイクロデータのデバイス別利用状況*

これらの増加はデスクトップでも同様で、`wpsidebar`（モバイルページでは1.4%から1.2%、デスクトップページでは1.3%から1.1%）など他の場所ではわずかに減少し、モバイル、デスクトップページともに全体として過去1年間の変化は最小となっています。

## JSON-LD

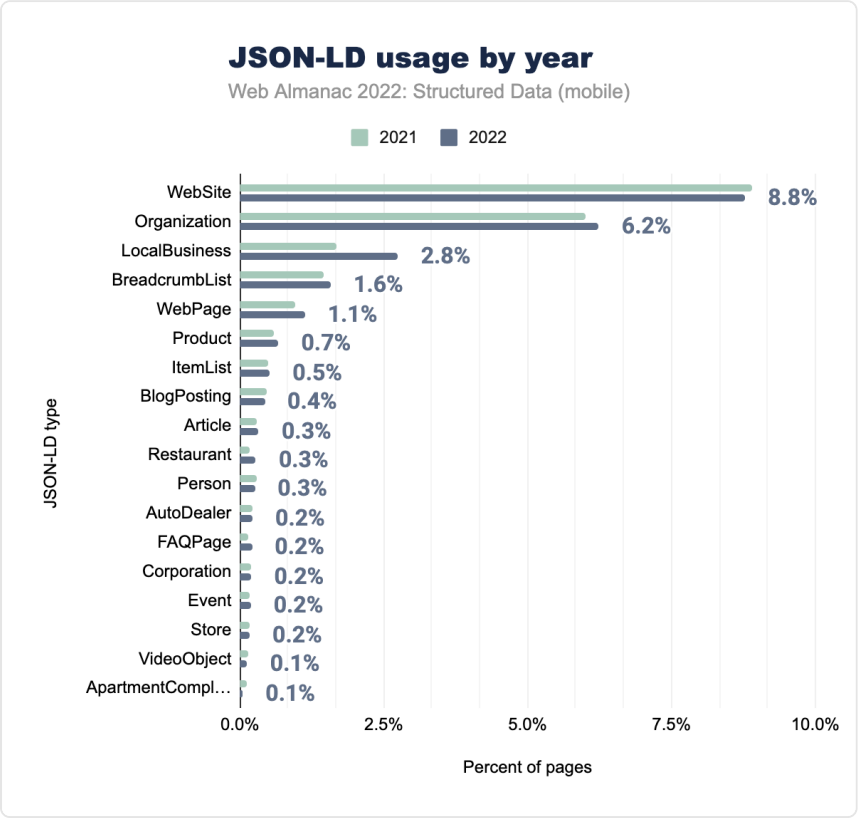



*図4.21. JSON-LDのモバイル利用状況（年度別）*

JSON-LDのタイプは、前年と比較していくつかの顕著な増加があるものの、ほぼ同様の傾向が続いています。とくに、`LocalBusiness`（セット内のページの2.8%に増加）と`Restaurant`（セット内のページの0.3%に増加）が挙げられます。

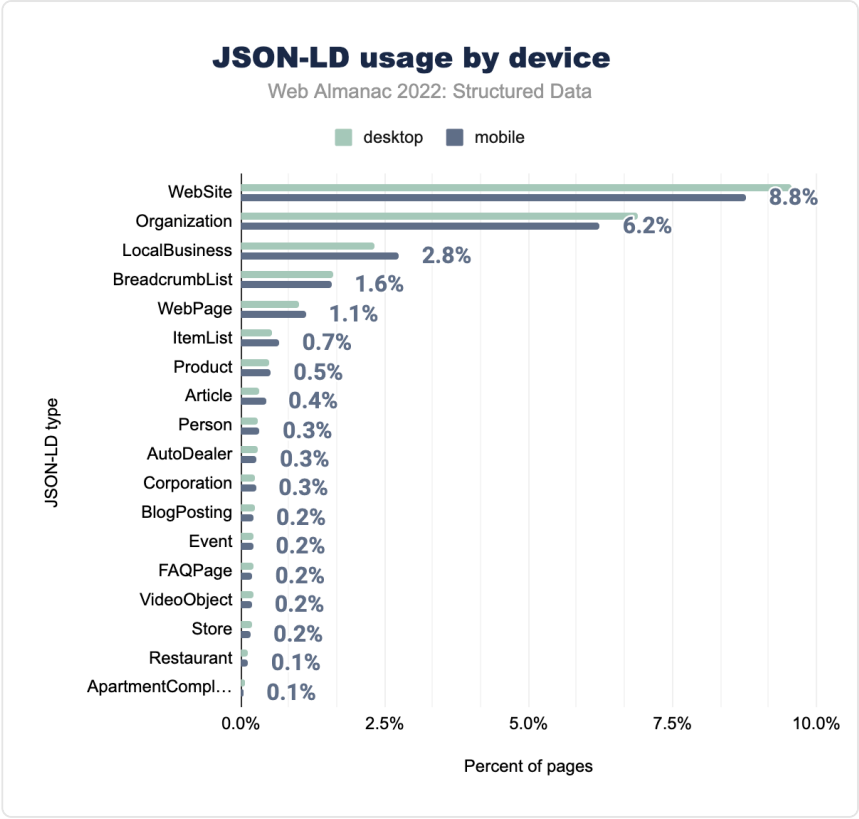



*図4.22. デバイス別JSON-LD使用状況*

これらの増加により、JSON-LDタイプは2021年以降で2番目に大きなポジティブな変化を遂げ、セット内のモバイルページでは33.5%から36.5%に、デスクトップページでは34.1%から36.9%に表示されています。

### JSON-LDの関係

JSON-LDを評価する場合、異なるクラス間の関係でもっとも繰り返されるパターンに注目することができる。他の構文に比べて、JSON-LDは構造化されたデータのグラフの価値を表現している。たとえば、`Article`は、リンクされた`image`とその *著者* を表す`Person`というエンティティタイプによって頻繁に特徴付けられます。同様に、`BlogPosting`も画像とリンクしていますが、`Publisher`として機能する`Organization`との関係が頻繁に見られます。

いくつかのタイプは純粋に構文的なもので、たとえば`BreadcrumbList`はサイトナビゲー

ションシステムの異なるアイテム（`itemListElement`）を接続するためにのみ使用され、`Question` は通常その答え（`acceptedAnswer`）と連動しています。他の要素は意味を扱います。`LocalBusiness` は通常、`address` と営業時間（`openingHoursSpecification`）にリンクされています。

この分析では、エンティティ間のもっとも一般的なタイプの関係と、たとえば `Article` と `BlogPosting` の微妙な違いについて、鳥瞰図的な概観を共有したいと思います。

ここでは、すべての構造体/リレーションシップの値において、どれだけ頻繁に出現するかに基づいて、異なるタイプ間の共通のリンクを見ることができます。これらの構造のいくつかは、通常、より大きなリレーションシップチェーンの一部となっています。

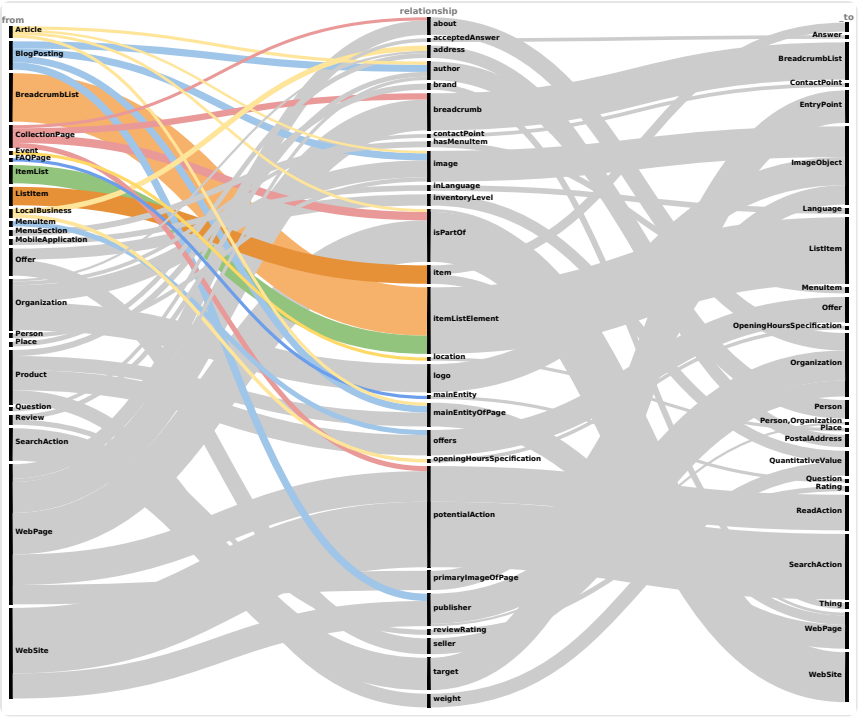



*図4.23. JSON-LDのエンティティ関係をSankeyダイアグラムで表示したもの。*

また、ニュースやメディアからEコマース、ローカルビジネスからイベントなど、これらの構成の背景にあるバーティカル（垂直）な要素も分析されています。

以下では、同じデータを、左がソース属性、右がターゲットクラスでインタラクティブに見ることができます。

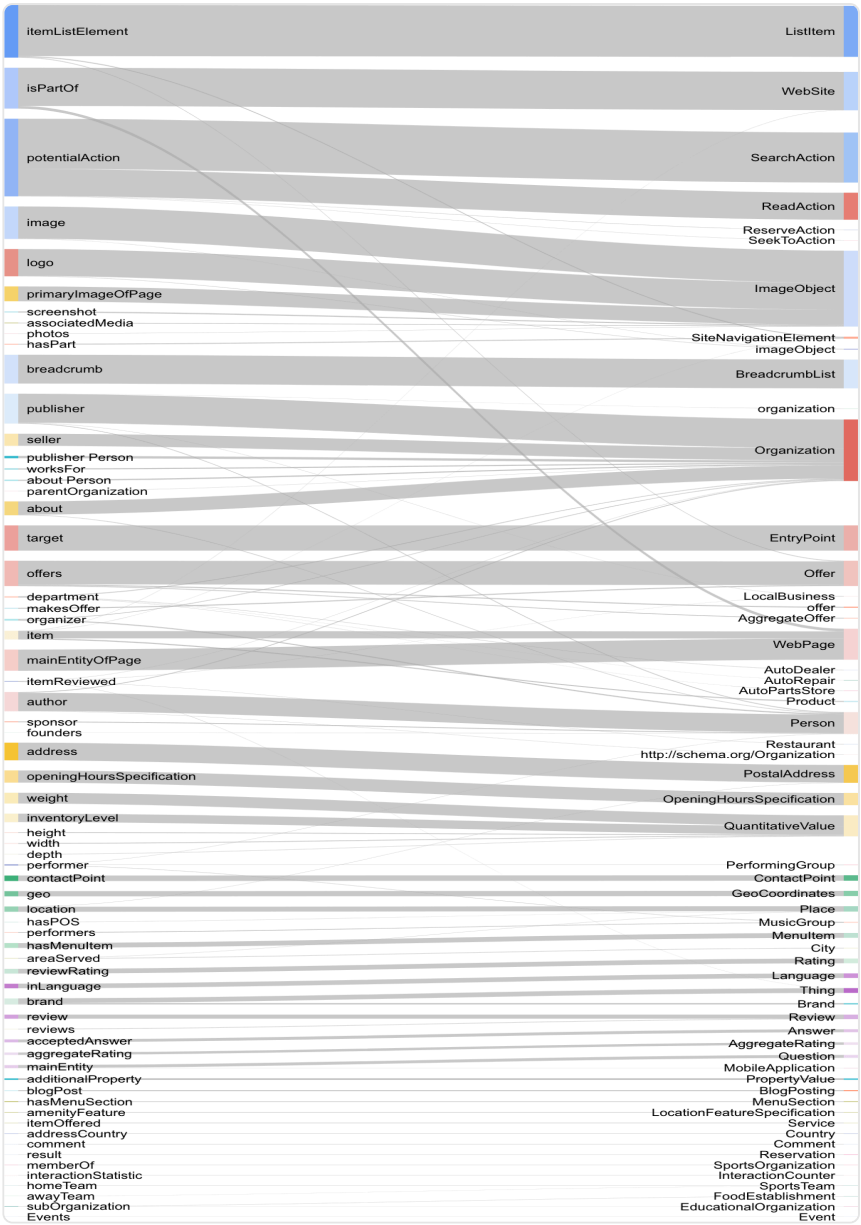



*図4.24. Sankey図。*

この分析の主な限界は、関係性の連鎖をホームページレベルで評価できることに表されている。

## SameAs

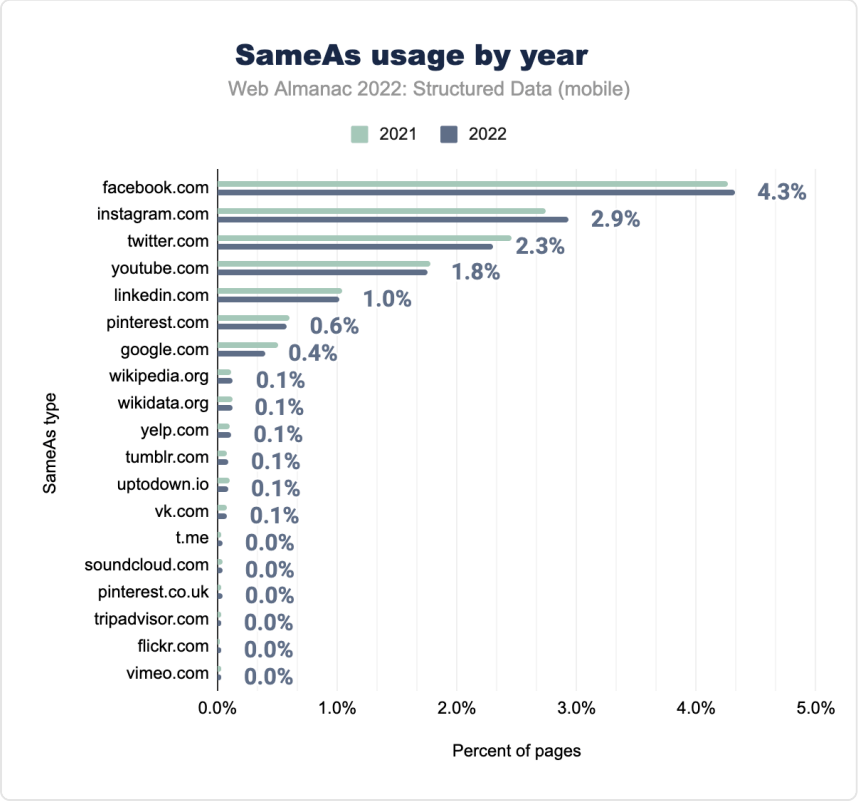



*図4.25. 年別の `SameAs` モバイル利用状況*

2021年と同様、`sameAs`プロパティのもっとも一般的な値は、ソーシャルメディア・プラットフォームです。facebook.com（モバイル：4.32%、デスクトップ：4.94%）、instagram.com（モバイル：2.93%、デスクトップ：3.34%）、twitter.com（モバイル：2.30%、デスクトップ：2.86%）などであった。前2者は2021年からモバイルでやや増加し、3者ともデスクトップで増加している。

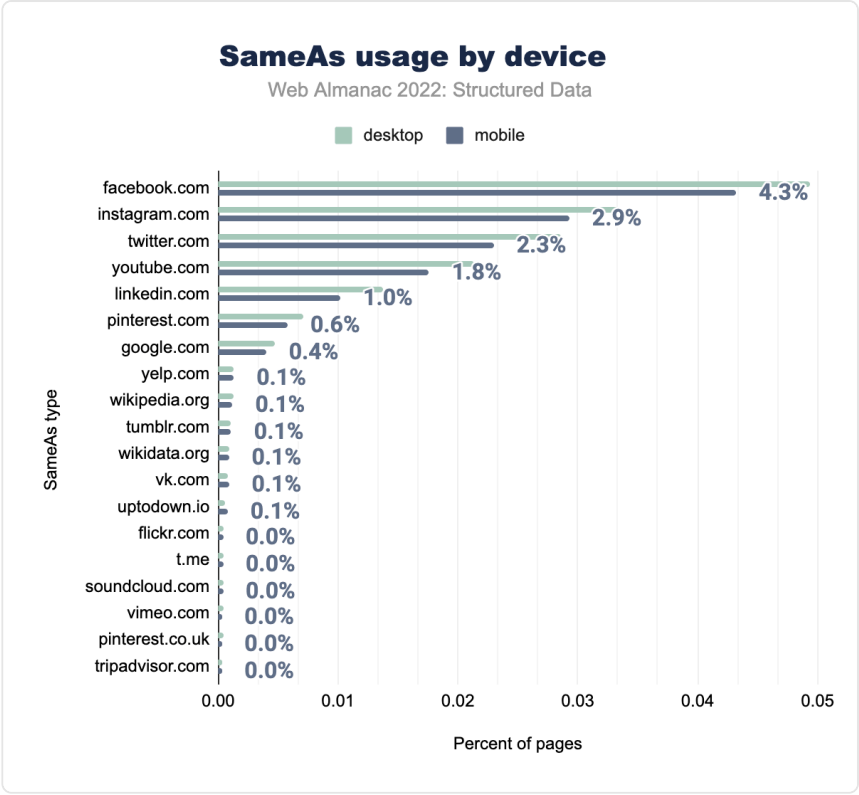



*図4.26. デバイス別の `SameAs` の使用状況*

残りのリストには、wikipedia.org（モバイル、デスクトップともに0.13%）、yelp.com（モバイル0.11%、デスクトップ0.13%）などの情報源が含まれており、いずれも前年より増加しています。

### JSON-LDのさらなる洞察 – 相対的な変化

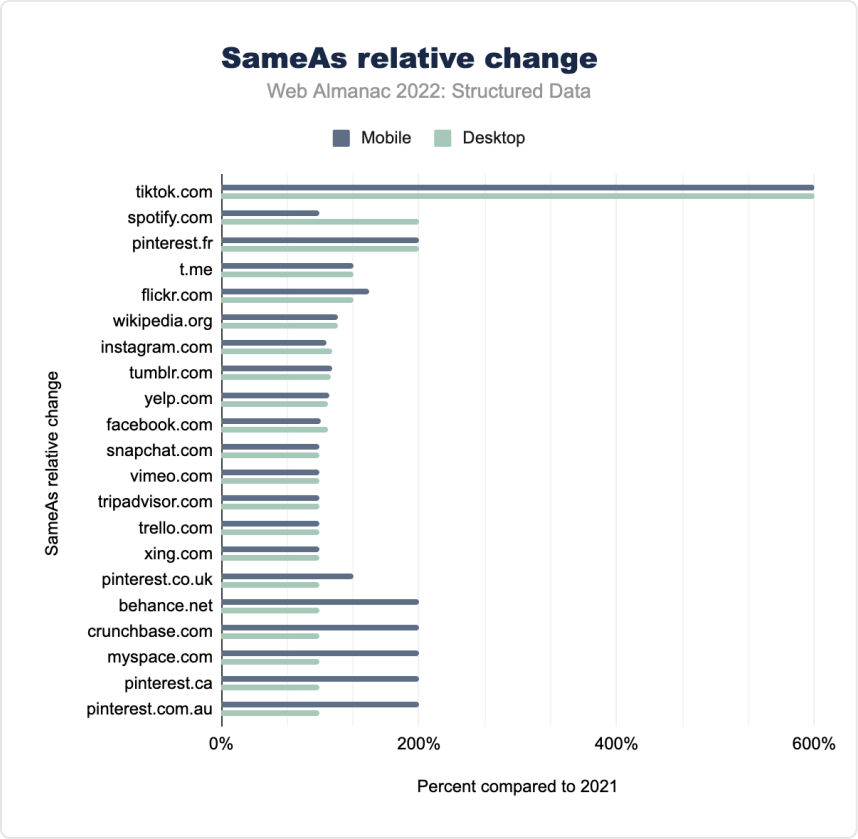



*図4.27. `SameAs` 相対的変化量*

また、`SameAs` の項目とその経時的な変化も興味深い。TikTokは、2021年と比較して、2022年は6倍ものページで表示され、大幅な増加を示しています。この変化は、デスクトップとモバイルの両方のページで同じです。Pinterestとそのドメイン名は、2022年にモバイルページでもっとも多く増加した上位5つのうち3つを占めています。モバイル全体では、デスクトップよりも `SameAs` エントリーの増加が大きいですが、Spotifyは例外で、デスクトップページの出現数が2021年と比較して2倍になっています。

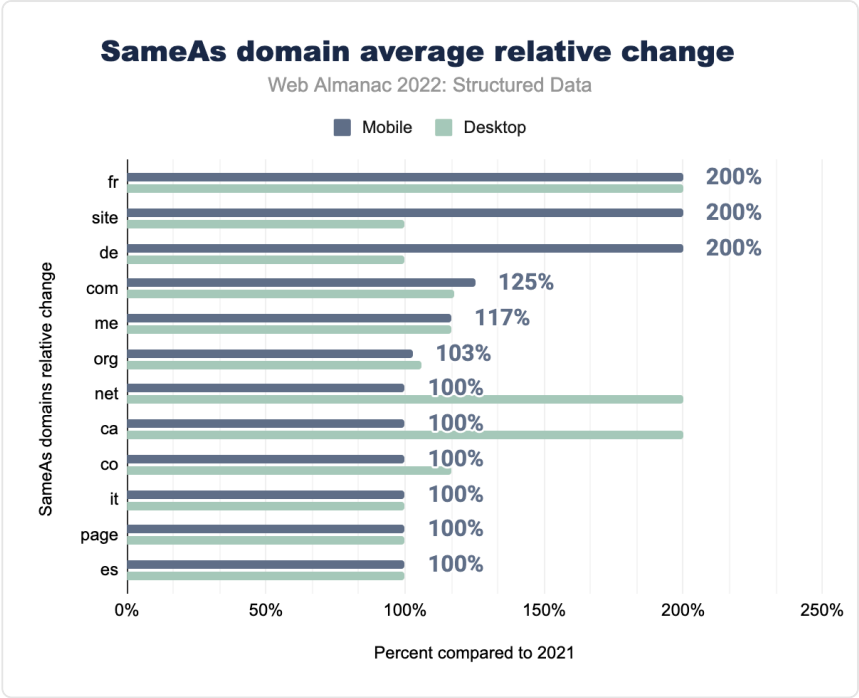



*図4.28. `SameAs` 平均相対変化量*

また、`SameAs`エントリーのドメイン名と、そのドメイン名が時間とともにどのように変化するかを見ることで、貴重な洞察を得ることができるかもしれません。デスクトップページの増加数がもっとも多いのは、`.ca`、`.net`、`.fr`で、後者はモバイルページの増加数でも上位にランクインしています。これは平均値であるため、エントリーの量は考慮されていません。どちらの年も、.comは他のすべてのエントリよりも数が圧倒的に多いのですが、平均の変化はモバイルページで125%、デスクトップページで117%となっています。カナダとフランス語のドメインの平均は、前述のように昨年から広く増加しているPinterestによって大きく押し上げられています。もっとも多い`SameAs`ドメイン上位10個のうち7個がPinterestを利用しており、時にはそれが唯一の利用例となることもあります。

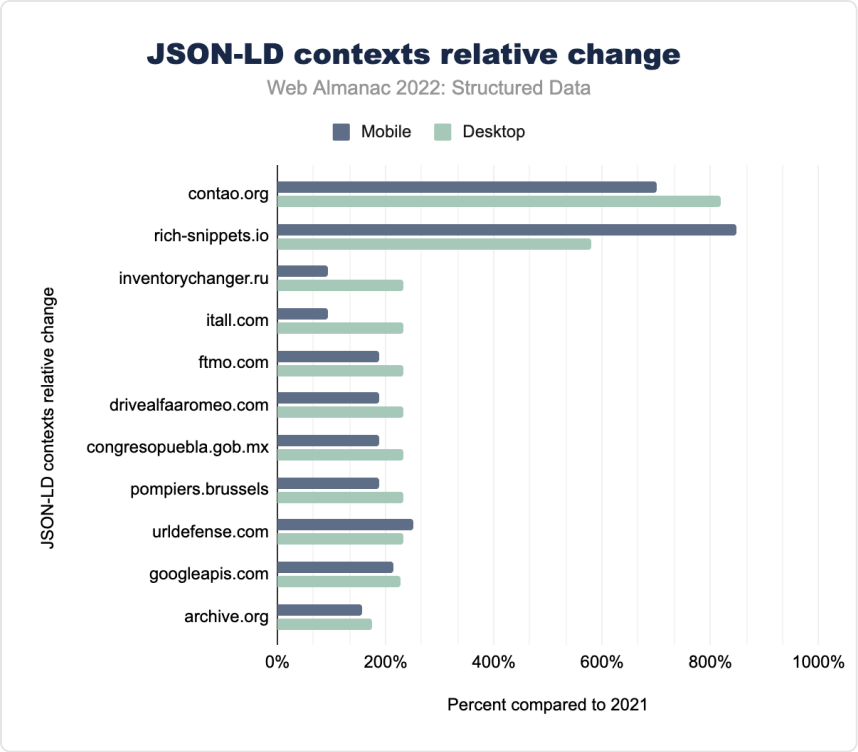



*図4.29. JSON-LDコンテキストの相対的な変化*

JSON-LDコンテキストについては、schema.orgが圧倒的に貢献度が高く、デスクトップページでは6,000倍以上、モバイルページでは2位のgoogleapis.comの3,500倍以上の登場回数となっています。とはいえ、schema.orgが昨年の108%という控えめな増加率だったのに比べ、googleapis.comの出現数はデスクトップとモバイルページの両方で2倍以上になっています。トップグロワーを見ると、contao.orgとrich-snippets.ioがトップで、contao.orgのデスクトップページの増加率は819%、 rich-snippets.ioのモバイルページの増加率は849% です。Contao.orgは総エントリー数で4位、rich-snippets.ioは8位にランクインしています。

## 結論

構造化データがウェブに、ひいては私たちの体験にどのような影響を与えるかについて、多くの概説がなされてきました。今年は、構造化データの採用が1年間でどのように変化したかを比較することにも焦点を当てました。たとえば、LocalBusiness（とくにレストラン）やFAQのようなクラスが全般的に増加し、 `foaf` やマイクロデータ構文に由来するあまり関連性のない構造化データ型の利用がわずかに減少していることがわかります。

包括的なリストとは言いがたいが、構造化（リンク）されたデータは、いくつかの利点をもたらすと考えられる。

- Eコマースページ
- ビジネスページ
- 研究者紹介
- ソーシャルメディアサイト
- ユーザー

SEOを通じて、発見力の向上、一般的なデータのリンク、リッチな結果が採用を後押しする。ウェブページにセマンティックマークアップを実装することで、よりつながりやすく、アクセスしやすいウェブが実現します。

これまで以上に多くの情報と洞察が得られるようになった現在、ウェブページのトラフィックを増やそうとする場合、特定のテクニックや選択肢の能力と限界を理解することが不可欠です。SEOを向上させる目的で偽のレビューや誤解を招くデータを追加すると検出され、その結果、上記のようなメリットが少なくなり、検索エンジンからの検出性とパフォーマンスが低下します。

前年度にすでに述べたように、SEOが構造化データ導入の重要な推進力であることに変わりはありませんが、検索エンジン以外の使用事例が増えつつあります。ウェブサイトの所有者はシステムの相互運用性を高め、コンテンツ推薦システムを訓練し、ウェブページから新しい洞察を得るために、さまざまなシナリオでデータを追加しています。

Web Almanacのこの章はまだ2年目ですが、これらのトレンドがどのように継続し変化していくのか、またWebにおける構造化データの状態も含めて、私たちは楽しみにしています。構造化データがもたらすあらゆる利点から、これらの技術の導入が進むことを期待しています。

### 著者

## Andrea Volpini

@cyberandy cyberandy https://wordlift.io/blog/en/entity/andrea-volpini

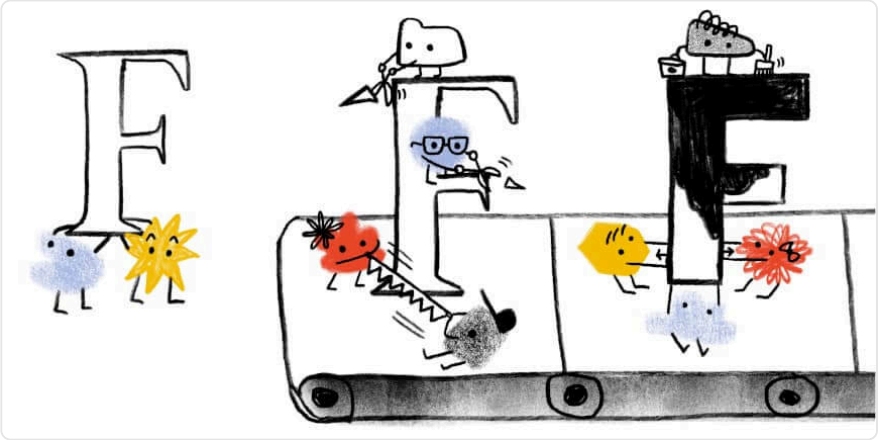
アンドレア・ボルピーニはWordLiftのCEOで、現在はセマンティックウェブ、SEO、人工知能に注力しています。


## Allen ONeill

@DataBytesAI DataBytzAI allenoneill https://webdataworks.io/


アレンは「The DataWorks」の創設者兼CTOで、世界中のもっとも多い組織にAI駆動型のウェブデータソリューションを提供しています。革新的な技術ソリューションを大規模に設計することに重点を置いており、主なバックグラウンドはエンタープライズシステムです。

# 部 I 章 5
# フォント



***Bram Stein によって書かれた。***
*Alex N. Jose、 José Solé、 Roel Nieskens と Chris Lilley によってレビュ。*
*Bram Stein と Kanmi Obasa による分析。*
*Shaina Hantsis 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

ウェブフォントの黎明期から、私たちは長い道のりを歩んできました。ほんの一握りのウェブセーフフォントから、何十万ものウェブフォントというタイポグラフィの爆発的な普及につながったのです。その技術や使いやすさは、ほとんど認識できないほどです。いくつかのフォントフォーマットを使った手の込んだ「bullet-proof」フォント読み込み戦略から、単にWOFF2ファイルを含むだけというものまで。

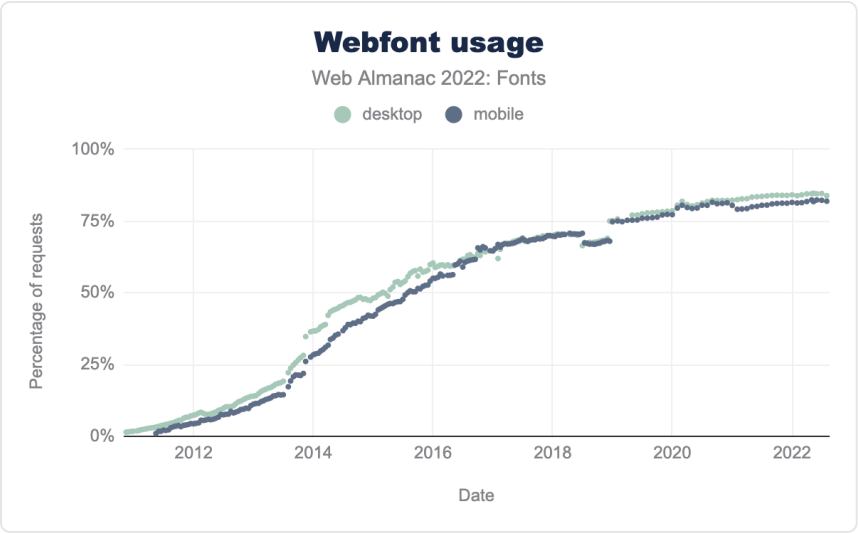



*図5.1. ウェブフォント使用。*

Webフォントのこの進歩が示すもの。ウェブフォントの利用率は伸び続けています。2020年のフォント[127]の章では、デスクトップサイト全体の82%がWebフォントを使用していました。それから2年で、使用率は約84%に増えています。モバイルでは若干数値が下がりますが、同様の伸びを表しています。

私たちは大きな進歩を遂げましたが、まだそこまでには至っていません。世界の人口の大部分はWebフォントを使用できません。なぜなら、彼らの文字システムは、（小さな）Webフォントとして提供するには大きすぎるか、複雑すぎるかのどちらかだからです。幸い、W3Cフォント・ワーキンググループ[128]は、インクリメンタルフォントトランスファー[129]ウェブ標準に熱心に取り組んでおり、うまくいけばこれを解決できるかもしれません。

2021年はFontsの章がありませんでしたが、今年はその分を取り戻せたらと思っています。今年は、フォントファイルの中身や、CSSでフォントがどのように使われているかを詳しく調べることで、少し違った角度から見てみました。もちろん、サービス、`font-display`、リソースヒントの使い方といった「基本」にも立ち返りました。最後に、*可変フォント* と *カラーフォント* に焦点を当てた2つの特別なセクションで、この章を締めくくります。

127. *https://almanac.httparchive.org/ja/2020/fonts*
128. *https://www.w3.org/Fonts/WG/*
129. *https://www.w3.org/TR/IFT/*

## パフォーマンス

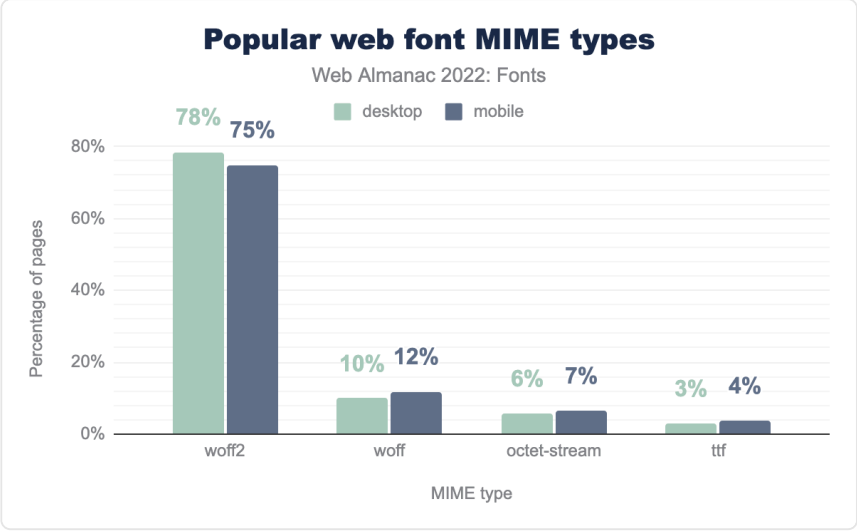



*図5.2. 人気のウェブフォントのMIMEタイプ。*

意外なことに、提供されるフォントの種類にはあまり変化がありません。全フォントファイルの約75%はWOFF2[130]として、約12%はWOFFとして、残りはオクテットストリームまたはとして提供されています。 TrueType Fontと、それからランダムなMIMEタイプの一群です。これは、2020年のフォントの章の結果[131]とかなり似ています。幸いなことに、SVGとEOTのフォント使用はほとんど完全になくなりました。

2019年、2020年に記載した通りです。WOFF2は最高の圧縮を提供し、好ましいフォーマットであるべきです。実際、宣言する時期でもあると思われます。

> *WOFF2だけを使い、他のことは忘れてください。*

これにより、CSSとワークフローが大幅に簡素化され、またフォントのダウンロードが誤って二重になったり、不正確になったりするのを防ぐことができます。WOFF2は現在、あらゆる場所でサポートされています[132]。ですから、よほど古いブラウザをサポートする必要がない限り、WOFF2を使ってください。そうできない場合は、古いブラウザにウェブフォントを一切提供しないことを検討してください。これは、しっかりとしたフォールバック戦略をとっていれば問題ありません。古いブラウザの訪問者は、単にあなたのフォールバックフォントを見ることができます。

130. https://www.w3.org/TR/WOFF2/
131. https://almanac.httparchive.org/ja/2020/fonts#フォーマットとMIMEタイプ
132. https://caniuse.com/woff2

## ホスティング

人々はどこでフォントを入手しているのでしょうか？セルフホストか、ウェブフォントサービスを使うか？両方ですか？数字を見てみましょう。

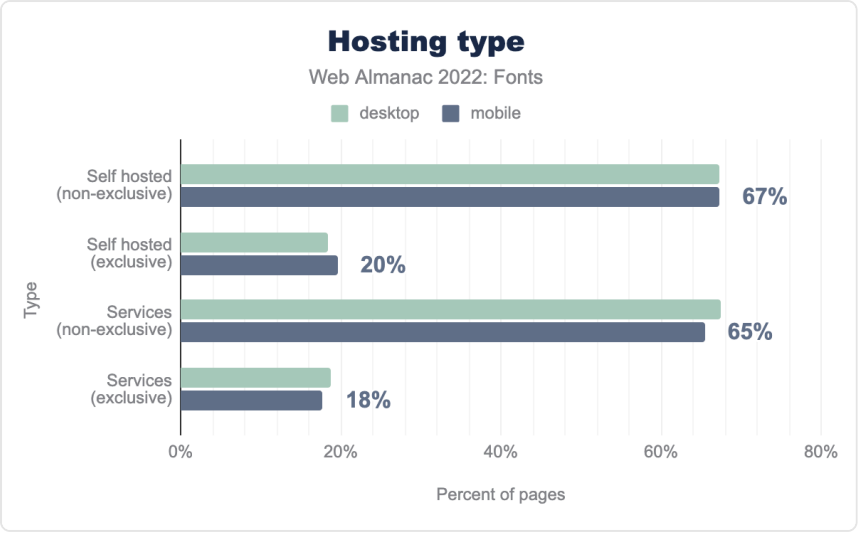



*図5.3. ホスティングタイプ。*

一般的には、67%がセルフホスティングとサービスを利用しています。セルフホスティングのみを利用しているのは19%に過ぎません。キャッシュ パーティショニング[133]の導入後、ホスティング サービスを使用することによるパフォーマンスの利点はもはやないこと、および欧州の裁判所は、欧州に拠点を置く企業がGoogleフォント[134] を使うことに徐々に強い疑念を持つようになっていることの理由からこの数字は今後数年間で上昇することが見込まれています。

このデータをさらにサービス別に分けることができます。驚くことではありませんが、Googleフォント[135]はもっとも多くのウェブフォントサービスで、すべてのウェブページの65%近くが使用しています。無料に勝るものはありませんね。

133. https://developer.chrome.com/ja/blog/http-cache-partitioning/
134. https://www.theregister.com/2022/01/31/website_fine_google_fonts_gdpr/
135. https://fonts.google.com/

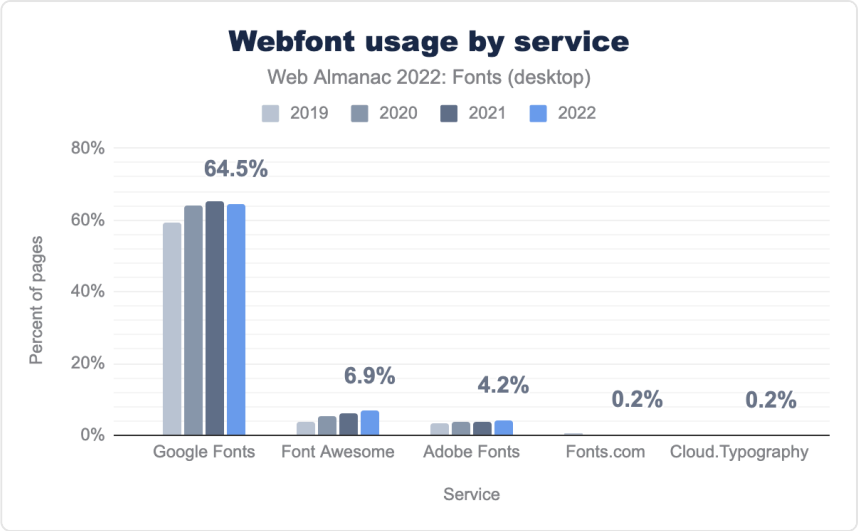



*図5.4. サービス別のウェブフォント利用状況。*

次点は、フォントAwesome[136]というウェブサービスで、7%近くのサイトで使われているそうです。たった1つのフォントファミリーで、この結果はすごいことです。3位はAdobeフォント[137]で、4.2%のサイトで使用されています。また、Fonts.com[138] と Cloud.Typography[139] は、どちらも0.2%のサイトで使用されているサービスです。

例年を振り返ってみると、今年ははじめてGoogleフォントの使用率が減少していることがわかります。これが前述のキャッシュパーティションによるものなのか、GDPRによるものなのか、それともまったく別のものなのか、何とも言えません。減少幅はわずかなので、この傾向が来年も続くかどうかが注目されます。

一方、フォントAwesomeとAdobeフォントの両サービスは、ここ数年で大きく成長しました。フォントAwesomeサービスの利用率は2019年から2022年にかけて86%成長し、Adobeフォントの利用率は同期間に24%成長した。

なお、サービスデータは、2020年および2019年のフォントの章と比較して、測定方法が異なります。それらの章では、サービスへのリクエスト数を調べたのに対し、2022年のデータでは、サービスを利用しているページを調べています。したがって、2022年のデータは、サイトに読み込まれたフォントの数に影響されないため、より正確なものとなっています。たとえば、2020年の章で指摘したGoogle Fontsの使用率の低下は、Google Fontsが *可変フォント* に切り替えたことにより、サービスへのリクエスト数が大幅に減少したことがもっ

136. *https://fontawesome.com/*
137. *https://fonts.adobe.com/*
138. *https://www.fonts.com/*
139. *https://www.typography.com/webfonts*

とも大きな原因でした。

## ファイルサイズ

圧縮は、ダウンロードする必要のあるデータ量を減らすのに最適な方法ですが、それには限界があります。フォントのファイルサイズに影響を与えるものをよりよく理解するために、全フォントのフォントサイズの中央値を見てみましょう。

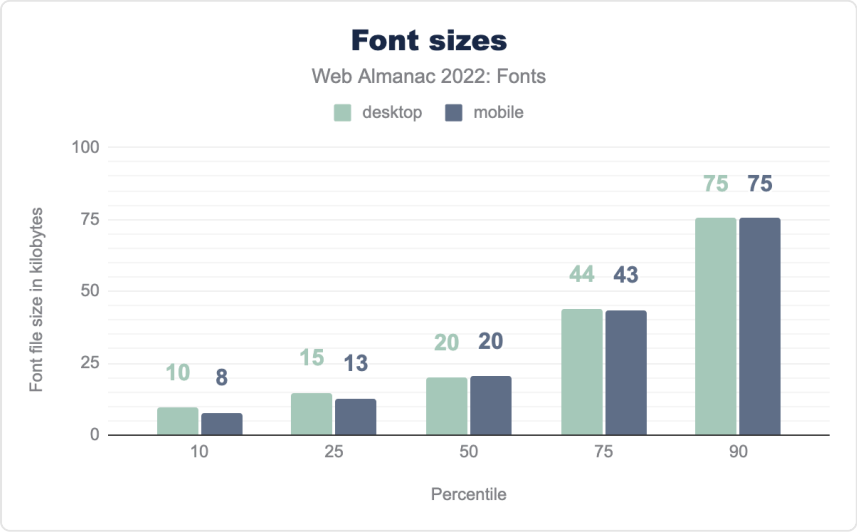



*図5.5. フォントサイズ*

フォントサイズの中央値は約20キロバイトです。これは結構なことです。しかし、先に見たように、フォントサービスは全フォントリクエストの70%近くを占めています。GoogleフォントやAdobeフォントのようなサービスには、フォントをできるだけ最適化するためのチームがあり、フォントサイズの中央値は、これらのサービスによって大きく歪められている可能性があります。

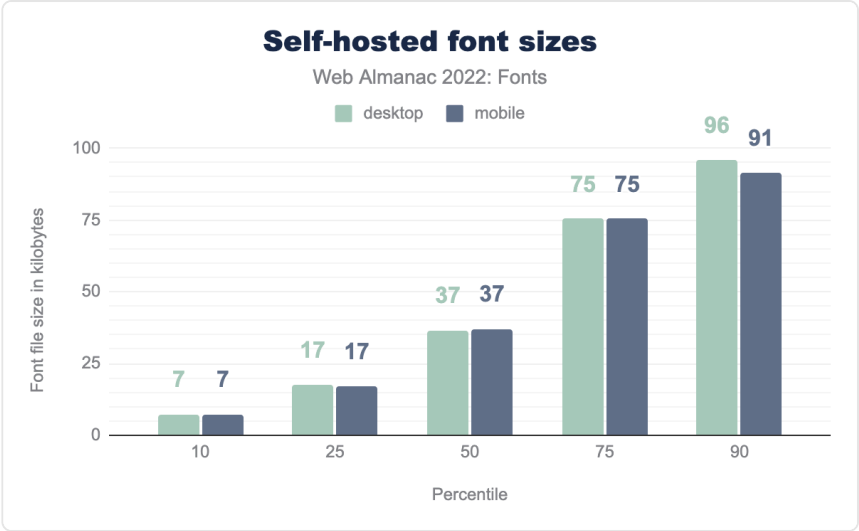



*図5.6. セルフホスティングのフォントサイズ。*

セルフホスティングのフォントサイズを見ると、まったく違うことがわかります。フォントサイズの中央値は約2倍の約40キロバイトに達しています。これはいったいどういうことなのでしょうか。人気のあるウェブフォントのMIMEタイプのグラフをもう一度見て、ウェブフォントサービスからのリクエストをすべて削除してみると、何が起こっているのかがわかるかもしれませんね。

セルフホストフォントを使用している多くのサイトがWOFF2ではなく、いまだにWOFFを使用しています。これらのサイトのフォントが、WOFF2が導入されてから一度も更新されていないのか、あるいはWOFF2について知っている開発者が少ないのか、はっきりしません。いずれにせよ、簡単にできる最適化であり、多くのサイトが恩恵を受ける可能性があります。

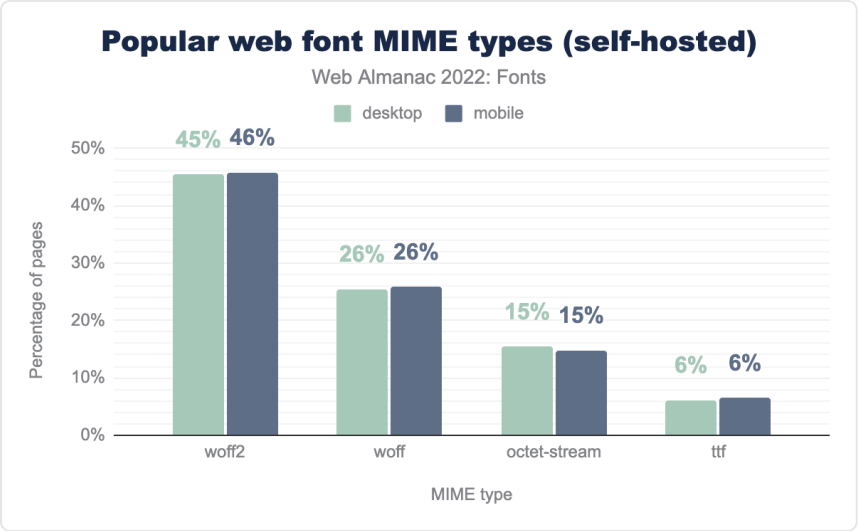



*図5.7. 人気のウェブフォントのMIMEタイプ（セルフホスティング）。*

しかし、サービスとセルフホスティングのフォントサイズの中央値の差は、WOFF2の使用率の低さで説明するには大きすぎる。WOFF2は優れた圧縮機能を備えていますが、圧縮だけではフォントサイズの中央値の大きな差は説明できません。ウェブフォントサービスは、セルフホストフォントでは十分に行われていない、他の種類の最適化を実行しているはずです。その答えを見つけるには、フォントの内部を見る必要があります。

## OpenTypeのテーブルサイズ

典型的なフォントは、基本的に 小さなリレーショナルデータベース[140] であり、それぞれのテーブルにはグリフの形状、グリフの関係、メタデータなどのデータが格納されています。たとえば、フォントの文字であるグリフを構成するベクターベジエ曲線を格納するテーブルがあります。また、グリフ同士を関連付けるテーブルもあり、カーニングや合字の関係（有名な リガチャ 合字のように、この2つのグリフを一緒に使うときはこのグリフと入れ替える）などが格納されています。

あるテーブルがファイルサイズ全体に与える影響を測定する合理的な方法は、そのテーブルの中央値を、そのテーブルを含むフォントの数に乗じることです。

140. *https://simoncozens.github.io/fonts-and-layout/opentype.html*

| OpenTypeテーブル | インパクト |
|---|---|
| glyf (ベクター形状) | 79.6% |
| CFF (ベクター形状) | 4.9% |
| GPOS (位置づけ関係) | 4.7% |
| hmtx (水平配置量) | 2.5% |
| post (メタデータ) | 2.2% |
| name (名前メタデータ) | 1.4% |
| cmap (文字コードとグリフを対応させる) | 1.3% |
| fpgm (フォントプログラム) | 0.9% |
| kern (カーニングデータ) | 0.6% |

*図5.8. 影響度（ファイルサイズの中央値×全体に占めるリクエスト数の割合）。*

もっとも高い影響力を持つテーブルのトップ10は、`glyf`、`CFF`、`GPOS`、`hmtx` のテーブルから始まります。これらには、すべてのグリフのアウトラインを構成するベジエ曲線（`glyf` と `CFF`）、OpenTypeポジショニング機能（`GPOS`）、水平メトリクス（`hmtx`）のデータが含まれています。これらの表はフォントのグリフ数に直接関係しているので、これは素晴らしいことです。不要なグリフを削除してフォントのグリフ数を減らせば、そのファイルサイズを劇的に減らすことができます。

何が必要、不要なのか[141]を見極めるのは難しいところです。誤ってグリフを削除してしまったり、テキストを正しくレンダリングするために必要なOpenType機能を壊してしまったりする可能性があります。たとえば フォントツール[142] を使って手動でサブセットする代わりに、subfont[143] や glyphhanger[144] などのツールを使って、サイトのコンテンツに基づいて「完璧な」サブセットを自動的に作成できます。ただし、フォントのライセンスがそのような変更を許可しているかどうかには注意してください。

興味深いのは、`name` テーブルと `post` テーブルがもっとも多い10位に入っていることです。この2つのテーブルは主にデスクトップフォントには重要だが、ウェブフォントには必要ないメタデータを含んでいます。これは多くのウェブフォントが、名前テーブルのエントリ、`post` テーブルのグリフ名、非Unicodeの `cmap` エントリなど、影響を与えずに削除できるメタデータを含んでいることを示しています。私たちは、ファウンドリやウェブ開発者

141. *https://bramstein.com/writing/web-font-anti-patterns-subsetting.html*
142. *https://fonttools.readthedocs.io/en/latest/subset/index.html*
143. *https://github.com/Munter/subfont*
144. *https://github.com/zachleat/glyphhanger*

がウェブ フォントの不要なバイトをすべて削除するために使用できる、普遍的な推奨事項（または pngcrush[145] に似たツール）をぜひ見たいと思っています。

## アウトラインフォーマット

OpenTypeのサイズテーブルには、ベクターグリフのアウトラインデータとして2つの項目があることにお気づきでしょうか。`glyf` と `CFF` です。OpenTypeには、ベクターのアウトラインを保存するための4つの競合する方法があります。TrueType（`glyf`）、コンパクトフォントフォーマット（`CFF`）、コンパクトフォントフォーマット2（`CFF2`）、スケーラブルベクターグラフィックス（`SVG`; 古いSVGフォントフォーマットと混同しないように）。また、画像ベースのフォーマットも3つあります。そのうちの2つについては、カラーフォントのセクションで説明します。

OpenTypeの仕様は、慈善的に「妥協の産物」と呼べるようなものです。コンセンサスが得られなかったため、もっとも同じことをする競合するいくつかのアプローチが仕様に加えられました。この経緯に興味がある方は、David Lemon氏のフォント戦争[146]が素晴らしい読み物です。競合するアプローチのこのパターンは、可変フォントとカラーフォントのセクションで何度も繰り返されます（ただし、アクターは異なります）。結局のところ、ベクター アウトラインを保存する複数の方法があることはもっともですが、タイプ デザイナーと実装に大きな追加負担をかけることになり、言うまでもなく、悪用のための攻撃対象領域が増えます。

タイプデザイナーは好みのアウトライン形式を選べる。アウトラインフォーマットの分布を見ると、タイプデザイナーが何を選択しているかがよくわかる。圧倒的多数（91%）のフォントが `glyf` のアウトライン形式を使用し、9%が `CFF` のアウトライン形式を使用しています。`SVG` カラーフォントの使用もありますが、1%未満です（写真には写っていません）。

145. *https://pmt.sourceforge.io/pngcrush/*
146. *https://www.pastemagazine.com/design/adobe/the-font-wars/*

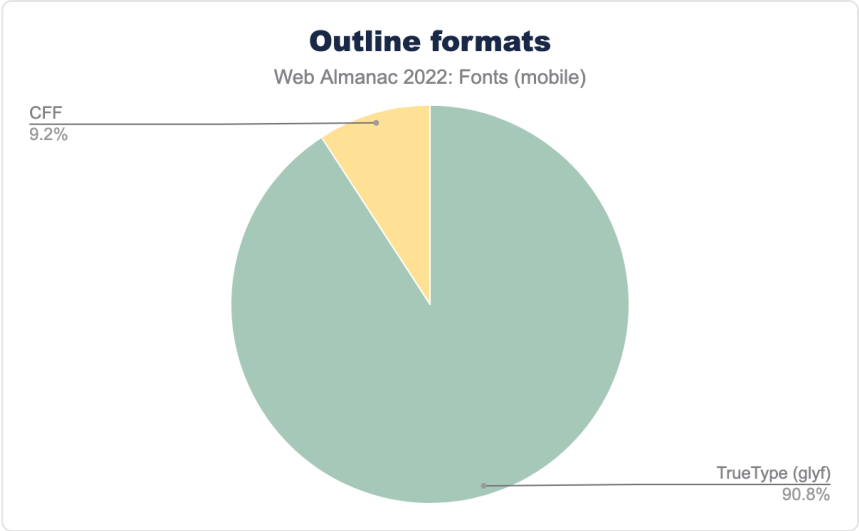



*図5.9. アウトラインのフォーマット。*

OpenTypeの仕様では、`glyf`と`CFF`の違いを挙げています。

- `glyf`フォーマットは2次ベジエ曲線を使用し、`CFF`（および`CFF2`）は3次ベジエ曲線を使用します。このことは、一部のタイプデザイナーにとっては重要ですが、フォントのユーザーにとっては重要ではありません。
- `glyf`フォーマットはヒンティングをよりコントロールし、小さな調整を行うことができます。一方、`CFF`はテキストラスターライザーにヒント機能をもっとも多く組み込んでいます。
- `CFF`フォーマットは、より効率的なフォーマットであり、フォントサイズを小さくすることができると主張しています。

最後の主張がおもしろいですね。`CFF`は小さいのか？

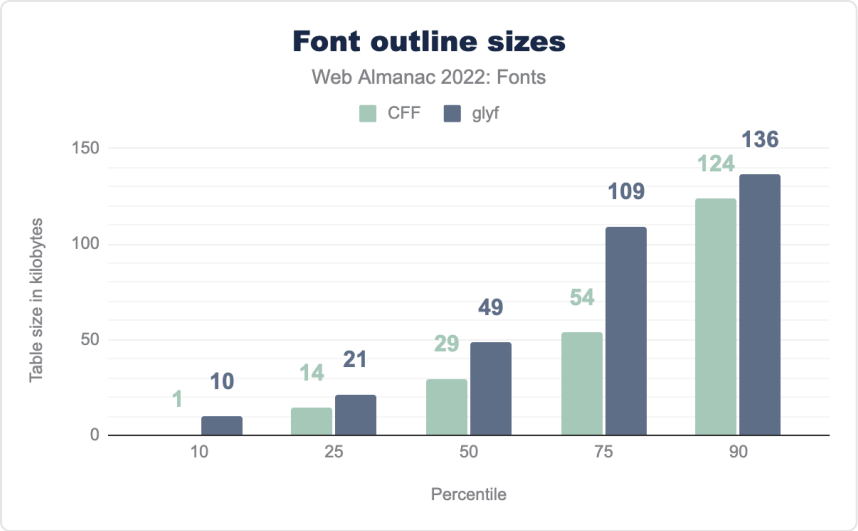



*図5.10. フォントのアウトラインサイズ。*

平均すると、確かに`CFF`の方がテーブルサイズは小さくなります。しかし、現実はもっと微妙で、圧縮を考慮していないため、テーブルサイズはフォントが解凍された後に記録されます。

WOFF2評価レポート[147]に見られるように、`glyf`圧縮の中央値は66%、`CFF`圧縮の中央値は50%です（WOFF2による非圧縮から圧縮までの場合）。

147. *https://www.w3.org/TR/2016/NOTE-WOFF20ER-20160315/#brotli-adobe-cff*

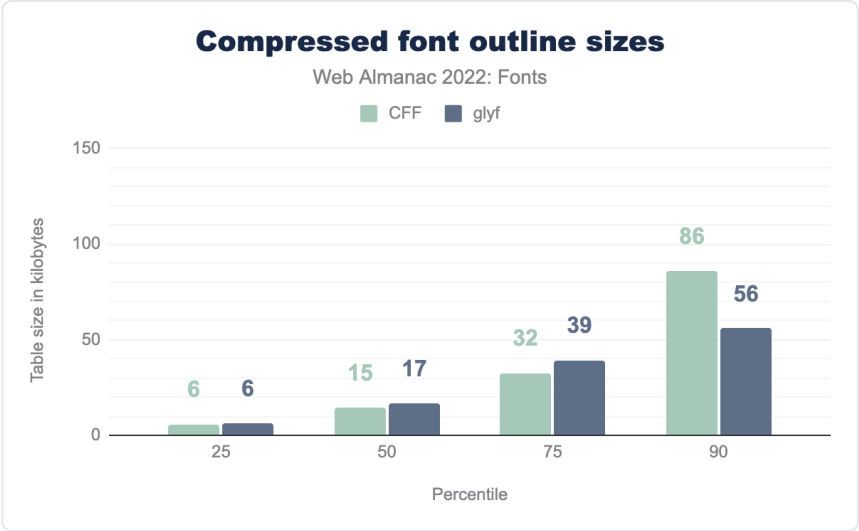



*図5.11. フォントのアウトラインサイズを圧縮しました。*

圧縮を適用すると、まったく異なる画像になります。ファイルサイズの中央値の違いはごくわずかで、`glyf`アウトラインを使用すると、大きなフォントがより小さくなります。

つまり、最初は`CFF`の方が小さくても、`glyf`よりも圧縮量が少ないので、最終的にはすべて同じになるのです。実際、大きなファイルでは、`glyf`形式を使用した方がサイズは小さくなるようです。

## リソースのヒント

リソースヒントとは、ブラウザが通常より早くリソースをロードまたはレンダリングするための特別な指示です。ブラウザは通常、ページでフォントが使用されていることが分かっている場合にのみ、ウェブフォントを読み込みます。それを知るためには、HTMLとCSSの両方を解析する必要があります。しかし、ウェブ開発者としてフォントを使用されることが分かっている場合は、リソースヒントを使用して、ブラウザにフォントをもっと早く読み込むように指示できます。

ウェブフォントに関連するリソースヒントには、`dns-prefetch`、`preconnect`、`preload`といういくつかの種類があり、影響の小さいものから高いものへと順番に並んでいます。理想的にはもっとも重要なフォントをプリロードしたいところですが、それらがホストされている場所によっては、必ずしもそれが可能とは限りません。

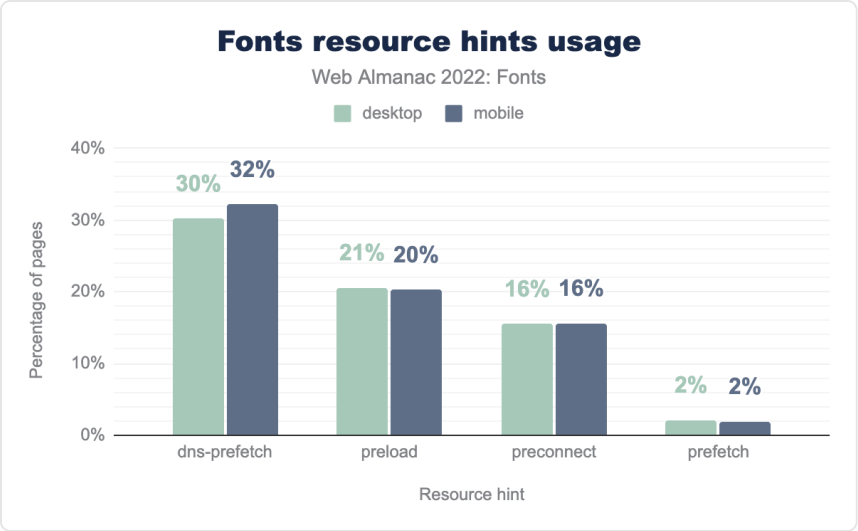



*図5.12. フォントのリソースヒントの使い方。*

2020年以降、`dns-prefetch`ヒントの使用には大きな変化はありませんが、`preload`（2020年の17%から2022年の20%）と`preconnect`（2020年の8%から2022年の15%）の使用はかなり大きく増えています。これはすごい！と思いました。

2020年の章で述べた[148]ように、`preload`と`preconnect`リソースヒントはフォントの読み込み性能にもっとも大きな影響を与えるものです。ほとんどの場合、それは頭にリンク要素を追加するのと同じくらい簡単です。たとえば、Googleフォントを使っている場合。

```
<link rel="preconnect" href="https://fonts.gstatic.com">
```

フォントをセルフホストしている場合は、さらに進んで、もっとも重要なフォント（たとえば主要なテキストフォント）を事前に読み込むためのヒントをブラウザに提供できます。そうすれば、ブラウザは早期にフォントを読み込むことができ、テキストのレンダリング開始時に利用できる可能性が高くなります。

## `font-display`

長年にわたり、もっとも多くのブラウザは、ウェブフォントが読み込まれるまでテキストを描画しませんでした。低速の接続では、テキストコンテンツがすでに読み込まれているにも

148. *https://almanac.httparchive.org/ja/2020/fonts#リソースのヒント*

かかわらず、数秒間、見えないテキストを表示することがよくありました。この動作は、テキストのレンダリングをブロックするため、*block* と呼ばれています。他のブラウザでは、フォールバックフォントをすぐに表示し、ウェブフォントが読み込まれたとき入れ替えるようになっていました。フォールバックフォントをウェブフォントに置き換えることを、*swap* と呼びます。

ウェブ開発者がフォントの読み込みをよりコントロールできるようにするため、ウェブフォントの読み込み中にブラウザがどのように振る舞うべきかを伝える `font-display` ディスクリプターが導入されました。これは、フォントの読み込み中に何をすべきかについて、4 つの異なる値を定義しています。これらの値は、*block* と *swap* の異なる組み合わせで実装されています。

- `swap`: のブロックは非常に短い期間で、常に入れ替わります。
- `block`: を短期間でブロックし、常に入れ替える。
- `fallback`: ブロックはごく短時間で、スワップは短時間で行う。
- `optional`: をブロックし、スワップ期間はありません。

また、`auto` もあり、これはブラウザに判断を委ねるもので、最近のブラウザはすべて `block` をデフォルト値としています。

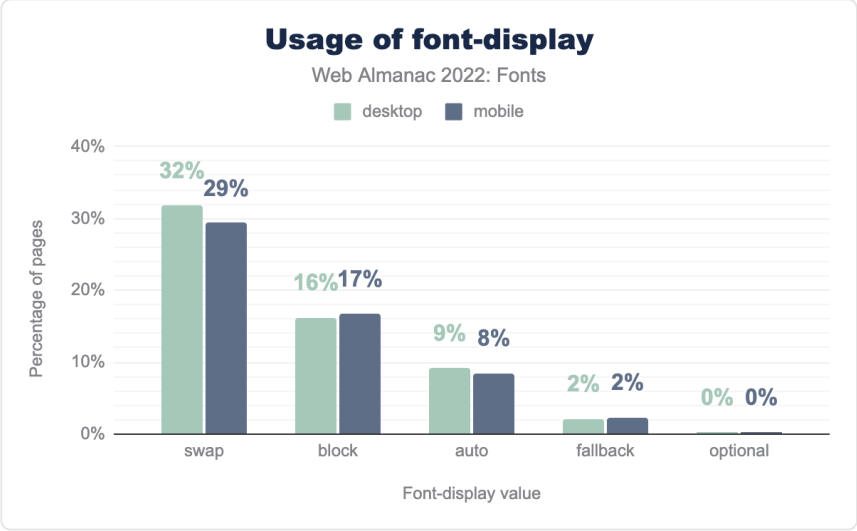



*図5.13. `font-display` の使用法。*

`font-display: swap` の使用は、印象的な30% にまで成長しました（2020 年の 11%[149] から）。この増加のかなりの部分は、Googleフォントが2019年にswapを推奨値にしたこと[150]を起因している可能性がもっとも高いです。また、`block` 値が `auto` を抜いて、2番目に多く使われている値になっているのも興味深い。開発者がなぜ意図的にサイトのパフォーマンスを低下させるのかはわかりませんが、少し心配ではあるものの、興味深い展開です。

私たちの推測では、開発者やその顧客は、フォールバックフォントのフラッシュを見るのを本当に嫌がると思います。`font-display: block` を使うことは、その問題を簡単に「解決」できます。しかし、もっと良い解決策があるのです。近い将来、CSSフォントメトリックオーバーライドを使って、フォールバックフォントを微調整し、ウェブフォントのメトリックに近づけることができるようになります。これにより、フォールバックフォントをウェブフォントと入れ替えたときに起こる、テキストの乱反射を抑えることができます。

CSSの `ascent-override`, `descent-override`, `line-gap-override`, `size-adjust` 記述子は `@font-face` ルールに入り、任意のフォントのメトリクスをオーバーライドするために使用できます。これらの記述子を `local()` と一緒に使うことで、Webフォントとほぼ一致するカスタマイズされたフォールバック[151]フォントを作成できます。`local()` の良い使い方がやっとできました。

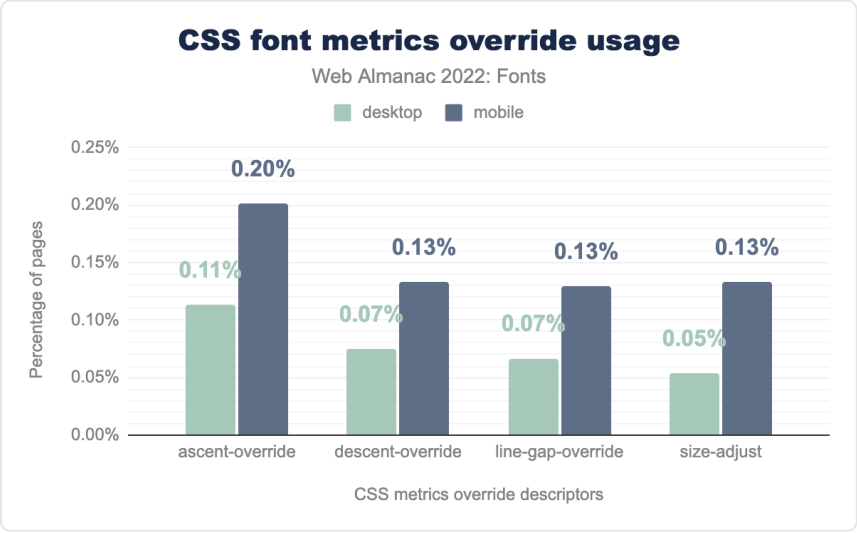



*図5.14. CSSフォントメトリクスのオーバーライド使用。*

この `@font-face` 記述子は非常に新しいものですが、すでにいくつかの使用例がありま

149. https://almanac.httparchive.org/ja/2020/fonts#ファーストペイントへの挑戦
150. https://www.youtube.com/watch?v=YJGCZCaIZkQ&t=1880s
151. https://developer.mozilla.org/ja/docs/Web/CSS/@font-face/ascent-override#%E4%BB%A3%E6%9B%BF%E3%83%95%E3%82%A9%E3%83%B3%E3%83%88%E3%81%AE%E5%AF%B8%E6%B3%95%E3%82%92%E4%B8%8A%E6%9B%B8%E3%81%8D

す。これらをさらに便利にするため、開発者は2つのことを必要としています。

1. すべてのブラウザとプラットフォームで利用可能な、一貫したフォールバックフォントのセットです。可変フォントにすることも可能です。可能性を想像してください。
2. フォントのサイズやメトリクスをいじって、自動的にマッチングさせるツール。これを手作業で行うのは非常に手間がかかるので、ツールは必須です。これはウェブフォントの代わりとしてではなく、単にウェブフォントが読み込まれている間の一時的なフォールバックとして（あるいはフォントが読み込まれなかったり、ブラウザが非常に古かったりした場合の代用品として）意図されています。

Font Style Matcher[152]や Perfect-ish Font Fallback[153]などのツールで徐々に実現しつつありますが、残念ながら、代替フォントはまだ非常にプラットフォームに依存します。

## フォントの使用方法

性能も重要ですが、ウェブ上でフォントがどのように使われているかということも興味深いです。たとえば、もっとも普及しているフォントやファウンドリは何か？人々はOpenTypeの機能を使っているのでしょうか？データを見てみましょう。

152. *https://meowni.ca/font-style-matcher/*
153. *https://www.industrialempathy.com/perfect-ish-font-fallback/*

## ファミリー＆ファウンドリー

| ファミリー | デスクトップ | モバイル |
|---|---|---|
| *Roboto* | *14.5%* | *1.5%* |
| *Font Awesome* | *10.5%* | *12.8%* |
| *Noto Sans* | *10.1%* | *8.0%* |
| *Open Sans* | *5.9%* | *7.7%* |
| *Lato* | *3.6%* | *3.9%* |
| *Poppins* | *3.0%* | *4.0%* |
| *Montserrat* | *2.5%* | *3.1%* |
| *Source Sans Pro* | *1.6%* | *1.9%* |
| *icomoon* | *1.3%* | *1.5%* |
| *Proxima Nova* | *1.0%* | *1.0%* |
| *Raleway* | *1.0%* | *1.3%* |
| *Noto Serif* | *0.8%* | *1.0%* |
| *Ubuntu* | *0.7%* | *0.9%* |
| *NanumGothic* | *0.7%* | *0.3%* |
| *Oswald* | *0.6%* | *0.8%* |
| *PT Sans* | *0.6%* | *0.8%* |
| *GLYPHICONS Halflings* | *0.5%* | *0.6%* |
| *Rubik* | *0.4%* | *0.4%* |
| *eicons* | *0.4%* | *0.6%* |
| *revicons* | *0.4%* | *0.5%* |

*図5.15. もっとも使われているフォント。*

Googleフォントがウェブフォントサービスに多大な影響を与えている今、ウェブでもっとも使われているフォントがRobotoであることは驚くにはあたらない。Robotoは、GoogleがAndroidオペレーティングシステムのシステムフォントとして作成したものです。このことは、Robotoのモバイルでの使用がデスクトップサイトと比較して大きく異なっていることの

説明にもなります。Androidでは、Google Fontsはウェブフォントとしてダウンロードするのではなく、システムにインストールされたバージョンを使用します。

2位はFont Awesomeで、実質的に1つのフォントファミリーであるにもかかわらず、素晴らしい成果をあげています。フォントAwesomeとIcomoon、Glyphicons、eicons、reviconsを合わせると、ウェブサイトで使用されているウェブフォント全体の約18%を占めています。アイコンフォントは、アクセシビリティの観点[154]から見て問題があるので、これがこれほど普及しているのを見ると心配になります。


*図5.16. アイコンウェブフォントを使用しているサイト*

また、使用率1%のProxima Novaも特筆すべき点です。これは、トップ20の中で唯一の商用、非アイコン、フォントです。これはMark Simonson Studio[155]の驚くべき成果です。

もう1つの興味深い事実は、上位のファミリーの大部分がオープンソースであるということです。これは、Google Fontsがこれらのフォントを委託しているか、既存のオープンソースフォントをライブラリに含めているためと考えられます。

154. *https://fontawesome.com/docs/web/dig-deeper/accessibility*
155. *https://www.marksimonson.com/*

| ベンダー | デスクトップ | モバイル |
|---|---|---|
| Google | 30.5% | 17.7% |
| フォントAwesome | 12.3% | 15.6% |
| Łukasz Dziedzic | 3.6% | 4.3% |
| Indian Type Foundry | 3.0% | 4.1% |
| Julieta Ulanovsky | 2.5% | 3.1% |
| Adobe | 1.6% | 1.9% |
| Ascender Corporation | 1.6% | 2.0% |
| Icomoon | 1.3% | 1.5% |
| Mark Simonson Studio | 1.3% | 1.3% |
| ParaType Inc. | 1.0% | 1.4% |

*図5.17. もっとも普及しているフォントファウンドリー。*

もっとも普及している活字鋳造所（場合によっては活字デザイナー）のリストも、同様に魅力的だ。Google、Adobe、Ascender（Monotype）のような大企業に加え、いくつかの小さな会社、さらには数人の個人がこれほど大きな影響を及ぼしているのを見るのは良いことです。

## OpenTypeの特徴

OpenTypeの機能は、しばしばフォントのスーパーパワーと呼ばれます。そしてもちろん、スーパーパワーを持つフォントは、認識されていないことが多い。OpenTypeの機能は、発見するのも使うのも難しいのです。幸いWakamai Fondue[156]のようなウェブツールがあり、どの機能があり、何をし、どう使うかを明確に示してくれます。

**44%**

*図5.18. OpenTypeの機能を含むフォント。*

OpenTypeの機能には、文体を整えるためだけのものもあれば、テキストを正しくレンダリ

156. https://wakamaifondue.com/

ングするために必要なものもあります。この2つは、裁量的機能と必須機能として言及されているのをよく見かけるかもしれません。全フォントの44%近くが、裁量的または必須的なOpenType機能を備えています。つまり、あなたが使っているフォントもスーパーパワーを持っている可能性が高いのです。

裁量的機能は、たとえば、隣接する2つのグリフを合字に置き換えて読みやすさを向上させるために使用できます。また、OpenTypeの機能では、たとえばスワッシュを追加してグリフの代替バージョンを提供することが一般的です。

かなりの数のフォントが、OpenTypeの裁量的機能[157]を備えています。もっとも一般的な裁量的機能は、驚くなかれ、合字です。これに続くのは、分数、比例数詞、表形式数詞、裏数字、序数詞など、数字を修正するあらゆる機能です。また、上付き文字もある程度一般的です。

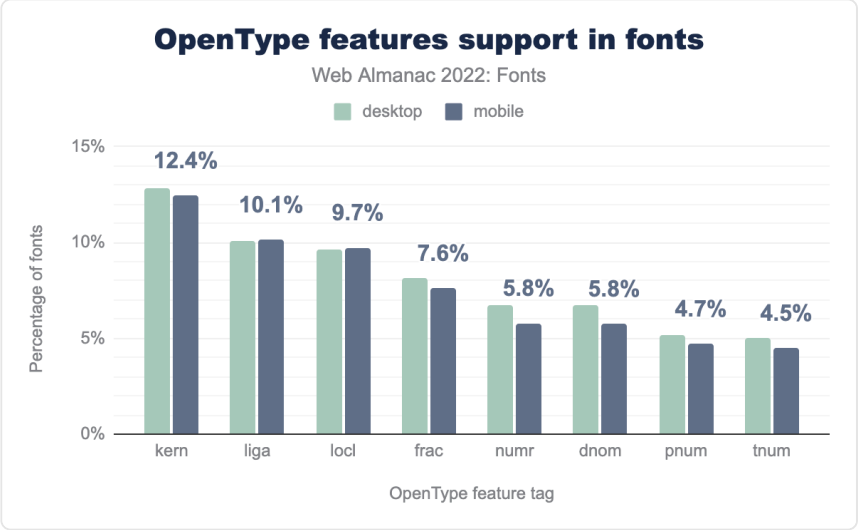



*図5.19. フォントでOpenType機能をサポート。*

OpenTypeの必須機能を見てみると、他のものよりも際立っているものが2つあります。 *カーニング*と *ローカライズフォーム* です。ローカライズフォームは、いくつかの言語で必要とされたり好まれたりする代替グリフを指定するために使用されます。これは、かなりの数のフォントが多言語をサポートしていることを意味し、国際化の進展の大きな証となります。

カーニングとは、2つのグリフの任意の組み合わせの間のスペースをわずかに増やしたり減らしたりして、それらの間のスペースをより均等に見えるようにすることです。カーニング

157. *https://fonts.google.com/knowledge/introducing_type/introducing_alternate_glyphs*

はすべてのブラウザでデフォルト有効になっているので、フォントがカーニングをサポートしている限りは有効になっています。

**34%**

*図5.20. カーニングデータを含むフォント。*

OpenTypeの機能として、または旧来の`kern`テーブルを使用してカーニングデータを保存しているウェブフォントは、全体の34%に過ぎません。ほとんどすべてのフォントは、正しく見えるようにカーニングを必要とするので、この数字は実際よりもずっと高くなると予想されます。ウェブフォントが普及し始めた頃、ブラウザのカーニングのサポートがあまり良くなかったため、初期のウェブフォントの多くは、スペースを節約するためにカーニングデータを含めなかったという説明があります。現在では、すべてのブラウザがカーニングをサポートしているため、フォントにカーニングデータが含まれていない理由はありません。

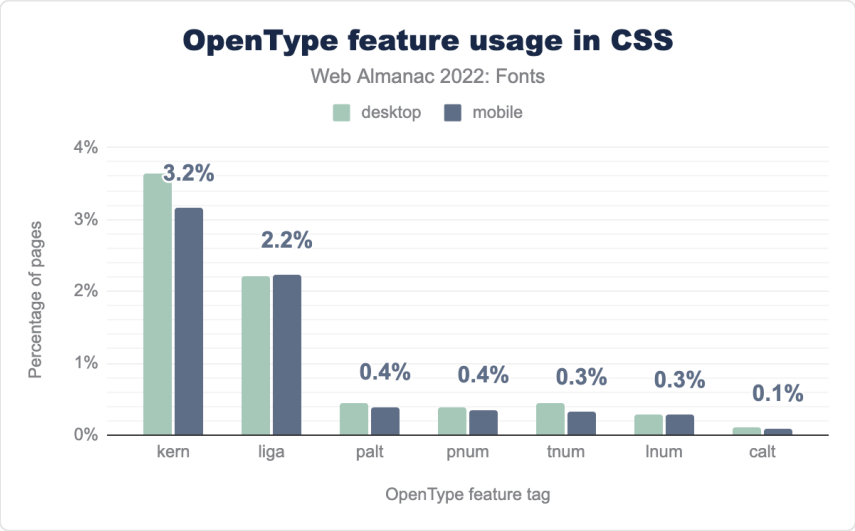



*図5.21. CSSにおけるOpenTypeの機能利用。*

さらに興味深いのは、カーニングは`font-feature-settings`プロパティでもっともよく使われる機能タグでもあるということです。3.2%近くのサイトがカーニングを手動で有効にしています（もっとひどい場合は無効にしています）。その必要はないのです。それはデフォルトで有効になっています。実際、`font-feature-settings`や、より高いレベルの`font-kerning`プロパティでカーニング設定を変更する必要はありません。カーニングを無効にすると、サイトが速くなるわけではありませんが、タイプセットが貧弱になるのは事実

です。

その他のもっともよく使われる機能は、実際にタイプデザイナーが使用するものとほぼ一致している：合字とさまざまな数字です。このリストに `palt`（プロポーショナルオルタネート）機能があるのは興味深いことで、これは主に日中韓フォントに使用されるものです（これ自体は、WOFF2圧縮でもウェブフォントとして使用するには通常大きすぎるため一般的ではありません）。カーニングと同様に、`calt` 機能（文脈上の代替文字）はデフォルトで有効になっており、明示的に有効または無効にする必要はありません。スタイルセット、文字バリエーション、スモールキャップ、スワッシュなど、使用頻度は低いものの、タイポグラフィを本当に向上させる可能性を秘めた便利なOpenType機能が他にもたくさんあります。私たちのオススメは、フォントを Wakamai Fondue[158]にドロップして、すべての隠れたスーパーパワーを探索することです。

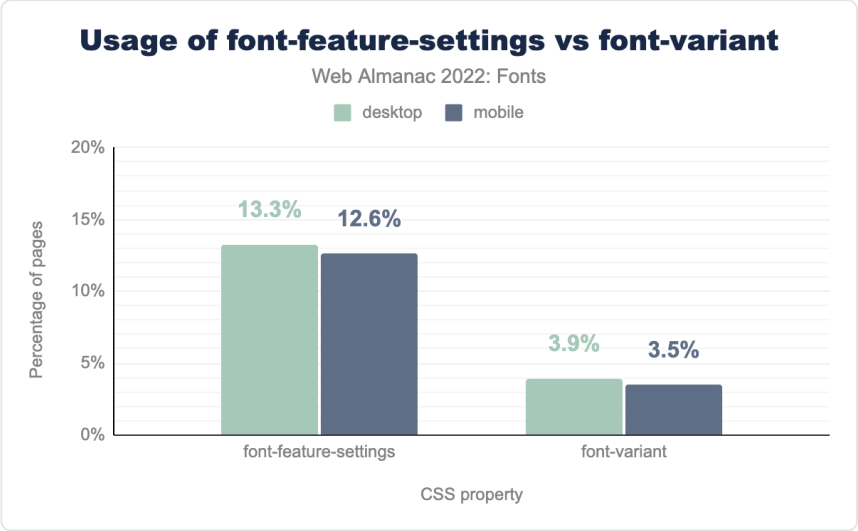



*図5.22. font-feature-settingsとfont-variantの使い分け。*

全体的に、OpenType機能の使用率はウェブ上でかなり低いです。私たちは、もっとも多くの人がOpenTypeの機能を有効にするため高レベルの `font-variant` プロパティを使用していることを期待していましたが、その使用率は `font-feature-settings` よりもさらに低くなっています。`font-feature-settings` プロパティは12.6%のサイトで使用されており、`font-variant` プロパティは3.5%のサイトでしか使用されていません。

これは残念なことです。人々はフォントに存在するOpenTypeの機能を使用していないだけでなく、高レベルの `font-variant` プロパティの代わりに、エラーが起こりやすい `font-feature-settings` プロパティを主に使用しているのです。このプロパティは、明示的に

158. *https://wakamaifondue.com/*

リストアップしなかったOpenType機能をデフォルト値に戻す[159]ので、`font-feature-settings`プロパティにはとくに注意する必要があります。

もっともよく使われる`font-feature-settings`の値はすべて`font-variant`と同等で、より読みやすく、副作用として他のOpenType機能を設定解除しないようになっています。これらのハイレベルな機能のサポートは過去にはそれほど良くなかったのですが、最近ではよくサポートされています[160]。最近導入された`font-variant-alternates`プロパティを除いては。

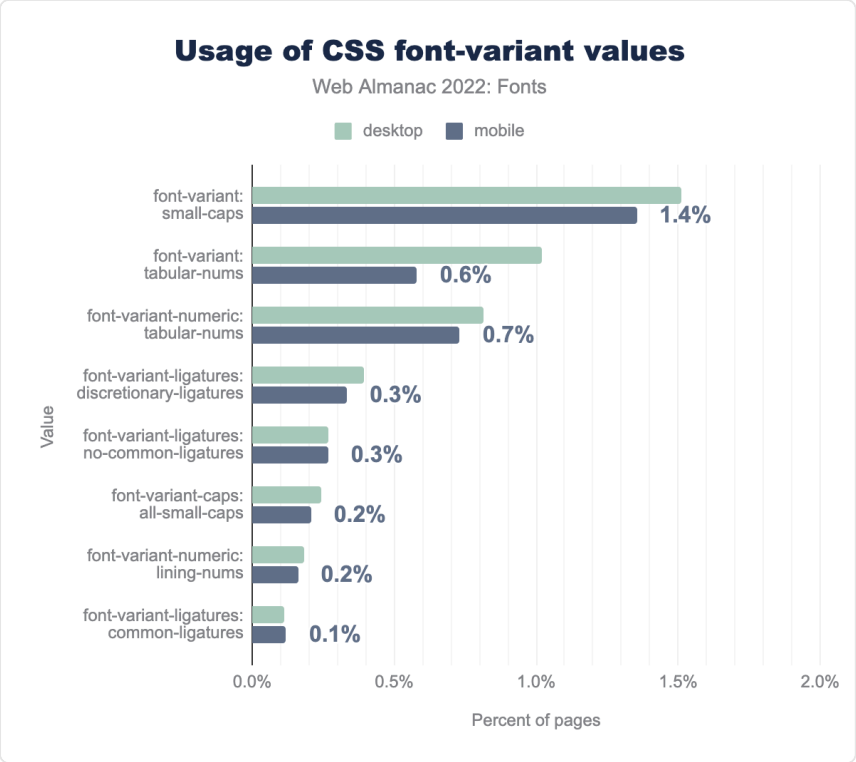



*図5.23. CSS font-variant値の使用。*

もっとも多くのページで使われている`font-variant`の値は`small-caps`で、1.4%でした。これは驚くべきことで、スモールキャップは0.7%のフォントでしかサポートされていないからです。つまり、`font-variant: small-caps`や`font-variant-caps`を使っているもっとも多くの人は、タイプデザイナーが作ったスモールキャップの代わりに、ブラウザが生成した偽のスモールキャップを受け取ってしまうということです。将来的には、font-

159. *https://pixelambacht.nl/2022/boiled-down-fixing-variable-font-inheritance/*
160. *https://caniuse.com/?search=font-variant*

synthesis-small-capsプロパティ[161]を使用することによって、偽のスモールキャップを回避できます。

その他の値は、フォントそのものが提供するものとほぼ一致しています。`font-variant`プロパティの使用率は低いですが、時間が経つにつれてこの数値が上がり、`font-feature-settings`の使用はカスタムまたは独自のOpenType機能での使用に追いやられることを期待、というより希望します。

## ライティングシステムと言語

どのようなフォントが作られ、使われているかを理解するために、フォントがどのような言語をサポートしているかを見てみるのもおもしろいのではないかと考えました。たとえば、ドイツ語やベトナム語、ウルドゥー語のフォントはたくさん作られているのでしょうか。残念ながら、この質問に答えるのは難しいのですが、多くの言語が同じ文字体系を共有しているからです。つまり、フォントは同じ文字セットを共有しているかもしれませんが、ある特定の言語向けにのみ明示的に設計されているかもしれません。そこで、言語ではなく、文字体系について見ていきましょう。

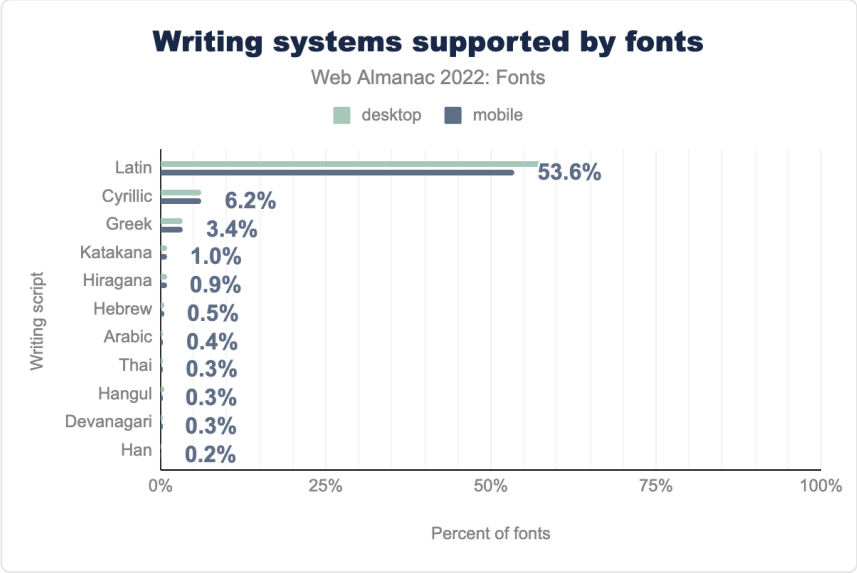



*図5.24. フォントがサポートするライティングシステム。*

驚くなかれ、ラテン系がリードしており、なんと全フォントの53.6%がラテン文字系[162]の

161. *https://drafts.csswg.org/css-fonts-4/#font-synthesis-small-caps*
162. *https://ja.wikipedia.org/wiki/%E6%96%87%E5%AD%97%E4%BD%93%E7%B3%BB%E5%88%A5%E3%81%AE%E8%A8%80%E8%AA%9E%E3%81%AE%E4%B8%80%E8%A6%A7#%E3%83%A9%E3%83%*

（重要な）サブセットをサポートしています。この中には、英語、フランス語、ドイツ語など、欧米で使われている多くの言語が含まれています。しかし、ベトナム語やタガログ語など、アジア圏の言語も含まれています。2位と3位は、キリル文字とギリシャ語です。これらは一般的に使われており、追加しなければならないグリフの数が限られているため、フォントに追加する作業もそれなりに少なくて済むからです。カタカナとひらがな（日本語）は、それぞれ1%と0.9%で、合計1.9%です。なお、約10%のフォントが、文字システムの最低基準を満たせませんでした（写真には写っていません）。たとえば、少数の文字にしか対応していないフォントや、サブセット化が進んでいるフォントが含まれます。

悲しいことに、他の文字システムはもっと普及していません。たとえば、漢文（中国語）は世界で2番目に普及している文字システム[163]です（ラテン語の次に）。しかし、ウェブフォントの0.2%しかサポートしていません。アラビア語は3番目に多く使われている文字システムですが、やはりウェブフォントの0.4%しかサポートされていません。これらのライティングシステムがウェブフォントとして使われていない理由[164]は、サポートしなければならないグリフの数が非常に多く、それらを正しくサブセットするのが難しいため、非常に大きくなってしまうからです。

GoogleフォントやAdobeフォントなどのサービスはこれらの文字システムをサポートしていますが、これらは独自の技術を使用しており、単にセルフホスティングで利用することはできません。しかし、W3Cフォントワーキンググループは、インクリメンタルフォントの転送[165]という新しい標準に取り組んでおり、Web開発者が大きなフォントをセルフホストできるようにします。この技術が広く利用されるようになれば、他の文字体系がラテン語に追いつくことを期待しています。

## 一般的なフォントファミリー

フォールバックフォントについては、`font-display`について説明したときにすでに触れました。たとえばUI要素やフォームの入力など、ウェブフォントがまったく必要ない場合もあります。ウェブフォントの使用を避けるための私たちのお気に入りの方法の1つは、オペレーティングシステムによって使用されるフォントファミリーにマッピングされる新しい一般的な`system-ui`ファミリー名を使用することです。他にもいくつかの一般的なファミリーがあることをご存知でしょうか？オペレーティングシステムのフォントを使いたいが、もう少し特殊なニーズがある場合は、`ui-monospace`, `ui-sans-serif`, `ui-serif`, `ui-rounded`もあります。

163. *https://www.worldatlas.com/articles/the-world-s-most-popular-writing-scripts.html*
164. *https://www.w3.org/TR/PFE-evaluation/#fail-large*
165. *https://www.w3.org/TR/IFT/*

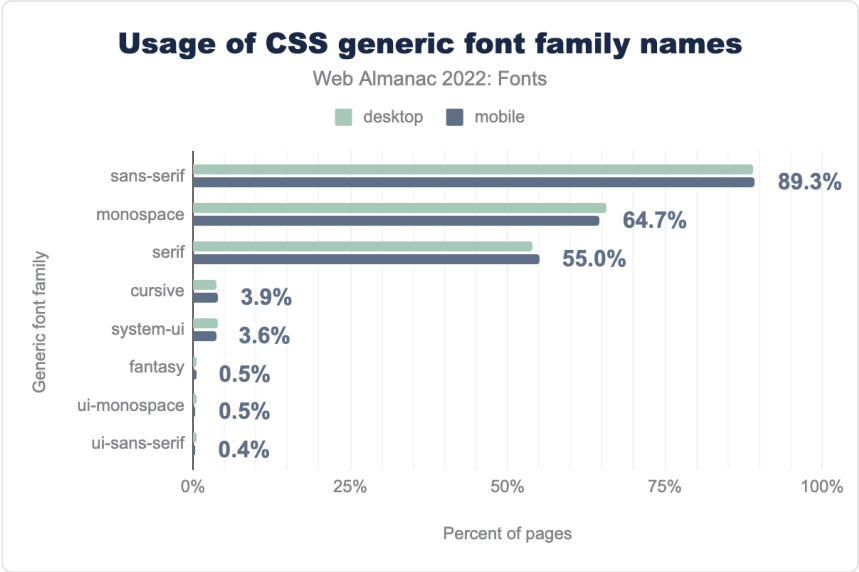



*図5.25. CSS汎用フォントファミリー名の使用。*

これらはかなり新しいものですが、すでに重要な用途に使用されています。とくに、`sans-serif`、`monospace`、`serif`は、CSS仕様の最初のバージョンから存在しているため、明らかにリードしていることがわかります。

もっとも普及しており、よく知られているのは`system-ui`で3.6%、次いで`ui-monospace`で0.5%、`ui-sans-serif`で0.4%です。0.5%の`fantasy`のリクエストは何を望んでいたのかは不明です。このジェネリックは仕様が不十分で、事実上役に立たないからです。

来年は、このような一般的なファミリーネームをより多く使用することを期待します。UI要素やフォームなど、ネイティブ感を出したい場合に最適です。さらに、ローカルにインストールされたフォントを使用することが保証されているため、パフォーマンス面でも優れています。

### フォントスムージング

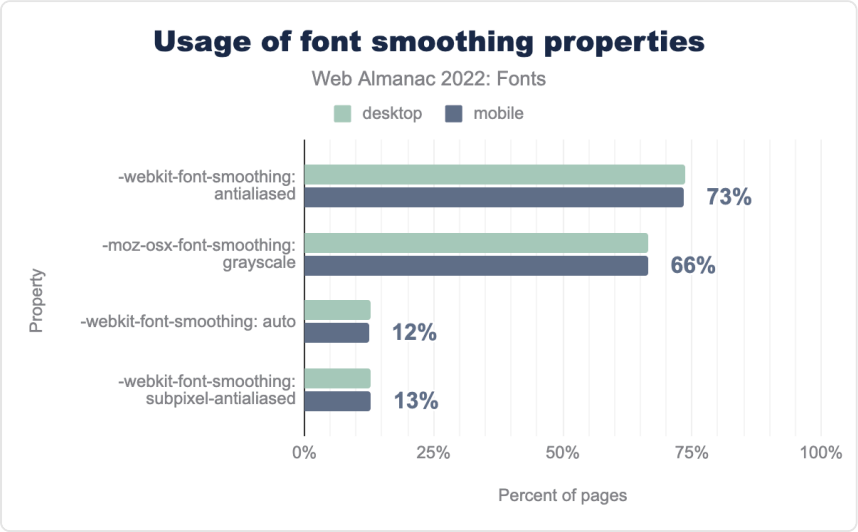



*図5.26. フォントスムージングプロパティの使用方法。*

そして、私たちにとっても驚きだったのが、macOS専用のフォントスムージング設定[166]を指定するのが。好きな人が多いことです。たとえば、`-webkit-font-smoothing: antialiased`という値は、全サイトの73.4%で使用されています。これは驚くべきことで、この値とその兄弟である`-mox-osx-font-smoothing`や`font-smoothing`は、公式のCSSプロパティですらないからです。これは、もっとも使用されている非公式CSSプロパティと言えるかもしれません。

これは、CSSフレームワークがこれらのプロパティを含んでいることと、macOSでフォントが少し太く表示されることを嫌って、可変フォントグレードを使用するようになったことが原因だと思われます。2023年の「フォント」の章で、これらのプロパティに再び触れるのもおもしろいかもしれません。また、これらのプロパティを標準化する時期が来ているのかもしれません。需要があるのは明らかです。

## 可変フォント

*可変フォント*により、タイプデザイナーは、1つのファミリーの複数のスタイルを1つのフォントファイルにまとめることができます。これは、ウェイト（細字、普通字、太字）や幅（コンデンス、ノーマル、エキスパンド）など、1つまたは複数のデザイン軸を定義するこ

166. *https://developer.mozilla.org/ja/docs/Web/CSS/font-smooth*

とによって実現されます。バリアブルフォントは、各スタイルを個別のフォントファイルとして保存する代わりに、一握りの「マスター」インスタンスからそれらを補間します。

たとえばタイプデザイナーが明示的にセミボールドのスタイルを作成しなかったとしても、ウェイト軸を持つバリアブルフォントを使用すれば、テキストレンダリングエンジンは単純にセミボールドのスタイルを補間します（バリアブルフォントのウェイト軸がその範囲をサポートしていれば、他のウェイトも必要な場合があります）。バリアブルフォントは、タイポグラフィの表現力を高めるだけでなく、ウェブ開発者にとって、複数のフォントバリエーションを使用した場合のファイルサイズの節約という大きなメリットもあります。ただし、バリアブルフォントは、1つのバリエーションよりもサイズが大きくなります。

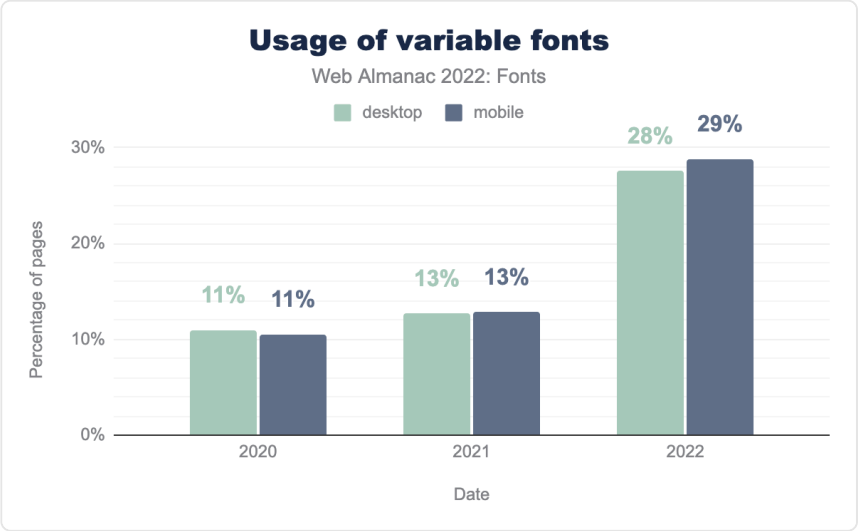



*図5.27. 可変フォントの使用。*

アルマナック2020年フォント章の前回の測定から、可変フォントの使用率が3倍近くに増加! 約29%のウェブサイトが可変フォントを使用しています。この成長のもっとも大きなものは、ここ1年で起こったようで、125%という驚くべき伸びを示しています。

**97%**

*図5.28. Googleフォントで提供されている可変フォントを使用しています。*

この驚異的な使用量の増加は、リクエストデータをホスト別に分割することで説明できます。Googleフォントは、提供される可変フォントの97%を占め、それ以外は3%に過ぎませ

ん。Googleがウェブフォントサービスに多大な影響を与えたとしても、この成長は、まったく新しい可変フォントを導入するのではなく、人気のある既存の静的スタイルを可変バージョンに置き換えることでしか説明できないのです。

その結果、Googleフォントとそのユーザーは、おそらくパフォーマンスにおいて大きなメリットを得ていることでしょう。可変フォントは、通常、複数の静的インスタンスを使用するよりも小さいです。たとえば、ウェブサイトが同じファミリーのスタイルを2つか3つ以上使用する場合、可変フォントはファイルサイズが小さく、1回のリクエストですみます。通常、レギュラー、ボールド、ライトウェイトの使用は、可変フォントを使用する十分な理由となります。さらに、可変フォントでは、ニーズに合わせて軸を調整することもできます、セミデミボールドは？

一人の俳優がこの成長に関与しているかどうかは別として、これは驚くべき成果であり、サイトのパフォーマンスを最適化するための可変フォントの有用性を示す良い指標となります。

可変フォントはまた、2つの競合するフォーマットを持っています: `glyf` フォーマットの可変拡張とコンパクトフォントフォーマット2（`CFF2`）フォーマットです。`glyf` フォーマットと `CFF2` の主な違いは、`CFF` の前身と同じで異なるタイプのベジエ曲線、より自動化されたヒンティング、そしてファイルサイズの縮小を謳っていることです。




*図5.29. アウトラインフォーマット `glyf` を使用した可変フォント。*

では、どのフォーマットを使えばいいのでしょうか？幸いなことに、開発者、タイプデザイナー、ブラウザベンダーにとって、状況は非常にシンプルです。提供されているすべての可変フォントのうち、99.99%が可変の `glyf` 形式を使用しています。Google Fontsやその他のフォントサービスを計算から除外しても、その数はなんと99.98%になります。誰も `CFF2` を使っていないのです。

`CFF2` ベースの可変フォントは、（少なくとも今のところは）避けることをオススメします。ブラウザやオペレーティングシステムが `CFF2` をサポートするようになったのはごく最近のことで、一部のブラウザではまだサポートしていません。`CFF2` を使用することで `glyf` ベースの可変フォントと比較して、唯一目に見える利点はファイルサイズの節約とされていますが、パフォーマンスのセクションで見てきたように、この主張はもっとも高いレベルで疑わしいものなのです。

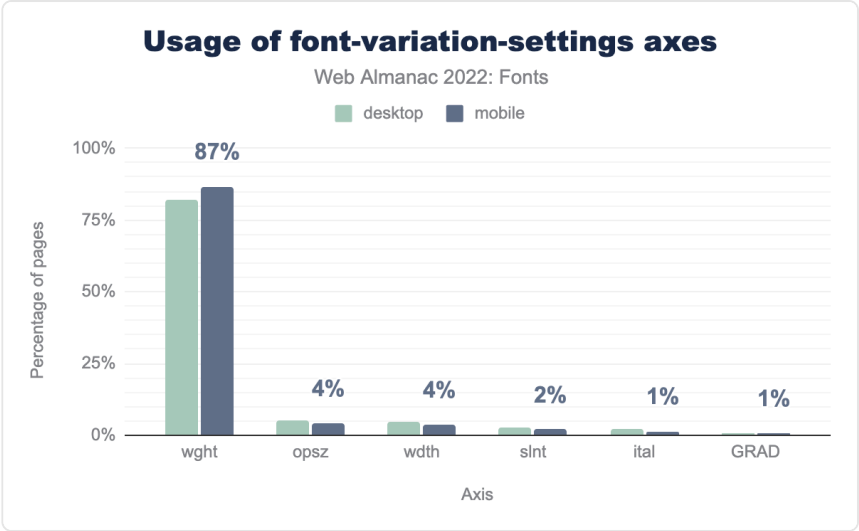



*図5.30. font-variation-settingsの軸の使用法。*

では、人々はどのように可変フォントを使用しているのでしょうか？当然のことながら、`font-variation-settings`プロパティで使用される値としては、ウェイト軸がもっとも普及しており、次いでオプティカルサイズ、幅、スラント、イタリック、グレードと続きます。

なぜなら、カスタムウェイト軸の値を設定するために、低レベルの`font-variation-settings`プロパティを使用する必要がないからです。たとえば、500と600の間の重さを設定するには、`font-weight: 550`のように、`font-weight`プロパティにカスタム値を指定するだけでよいのです。

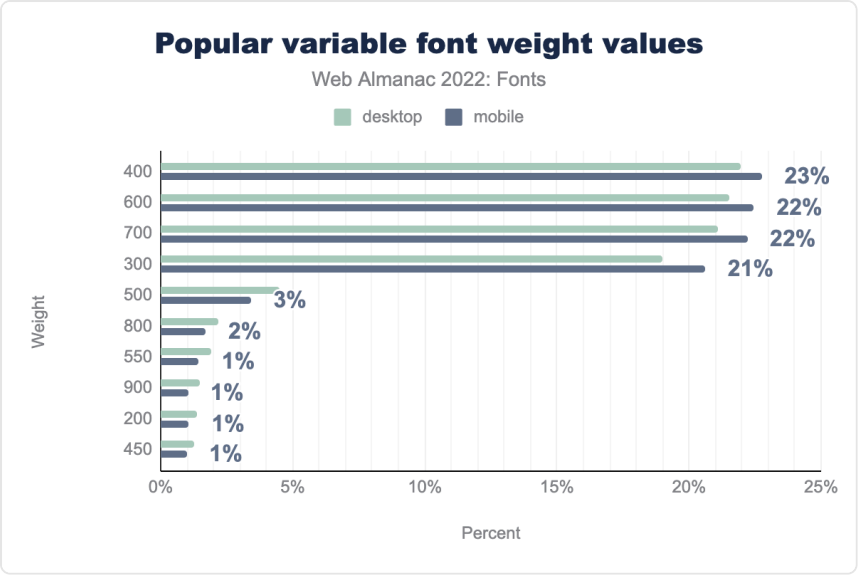



*図5.31. 人気の可変フォントウェイト値。*

さらに不可解なのは、もっとも普及しているウェイト軸の値が、CSSの初期からある「デフォルト」のウェイト値であることです！ウェイト軸の値は、CSSの初期からある「デフォルト」のウェイト値です。ウェイト値のうち、ウェイト範囲に沿ったカスタム値は16%しかないのです。

もっとも普及している「カスタム」ウェイト値は `550` で、使用率はわずか1.5%、次いで `450` で使用率は1%です。同様の結果は、オプティカルサイズ、ワイド、イタリック、スラント軸にも見られ、これらは高レベルの `font-optical-sizing`, `font-stretch`, `font-style` プロパティを使用して設定することができる。高レベルのプロパティを使用すると、CSSがより読みやすくなり、低レベルのプロパティでよくあるエラーの原因である軸を誤って無効にしてしまうことを防ぐことができます。

バリアブルフォントの利点の1つとして、1軸または複数軸のアニメーション化が強く推奨されています。可変フォントの使用率が高いにもかかわらず、CSSの遷移やキーフレームを使って実際にアニメーションさせている人はごくわずかです。HTTP Archiveの全データセットのうち、可変フォントを使ったアニメーションを行っているウェブサイトは、わずか数百件です。

私たちの目には、バリアブルフォントは主にパフォーマンス上の利点で使用され、タイポグラフィの調整能力ではあまり使用されていないように見えます。それはそれで素晴らしいことですが、今後、より多くの人がバリアブルフォントを使い、そのタイポグラフィの可能性を最大限に発揮してくれることを期待しています。

# カラーフォント

カラーフォントは、皆さんが期待する通り、色が内蔵されたフォントです。元々は絵文字のために作られた技術ですが、今では絵文字よりも文字色フォントの方が多くなっています。

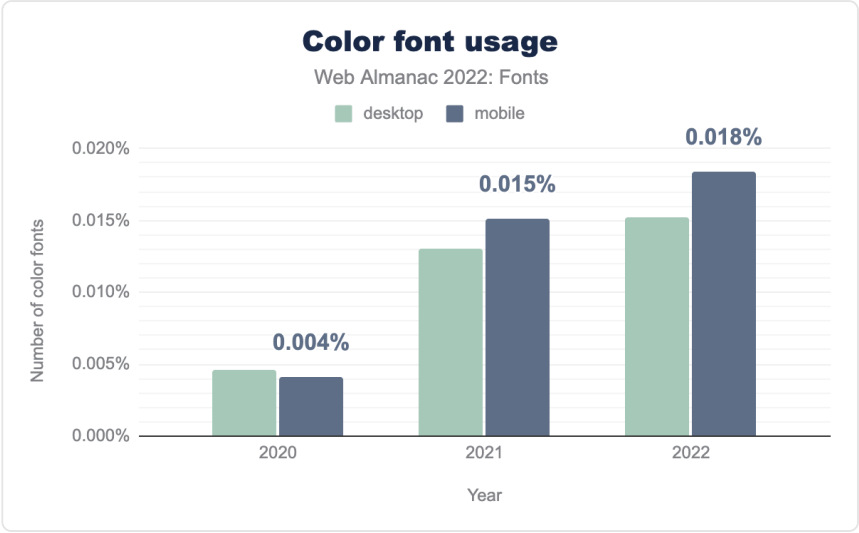



*図5.32. カラーフォントの使用方法。*

カラーフォントの使用率は、前回のフォント章（2020年）からかなり伸びています。2020年にカラーフォントを使用しているページの0.004%だった使用率が、2022年には約0.018%になりました。この数字はまだ非常に小さいですが、その使用率は明らかに伸びています。

しかし、可変フォントの使用率の伸びと比較すると、カラーフォントの取り込みが限定的であることは、やや残念なことです。カラーフォントはOpenType仕様に追加された比較的新しいもの（2015年）ですが、可変フォントはさらに最近追加されたもの（2016年）です。

カラーフォントの普及を大きく妨げてきた（そしてこれからも妨げるかもしれない）主な要因は、*唯一の真のカラーフォント形式*を求める標準化の「戦い」が続いていることと、カラーフォントパレットを選択・編集するためのCSSが、最近までブラウザでサポートされていないことです。

現在、4つのカラーフォントフォーマットが競合しています。2つはベクターアウトラインに基づくもの（`SVG`と`COLR`）、2つはイメージに基づくもの（`CBDT`と`sbix`）です。`COLR`フォーマットは、既存のグリフのアウトラインを再利用し、それにソリッドカラーとレイヤリングを追加したものです。もっとも新しいバージョンは`COLRv1`と呼ばれ、グラデーション、合成、ブレンドモードも導入された。既存のグリフアウトラインを再利用するため、

COLR フォーマットは可変フォントにも対応しており、アニメーションのカラーフォント[167]を作成できます。SVG フォーマットは異なるアプローチをとり、基本的にフォントの各グリフに対してSVG画像を埋め込みます。残念ながら、SVG フォーマットは可変フォントに対応していませんし、将来的にも対応する可能性は低いでしょう。CBDT と sbix'はどちらも各グリフに画像を埋め込みますが、対応する画像形式が異なるだけです。

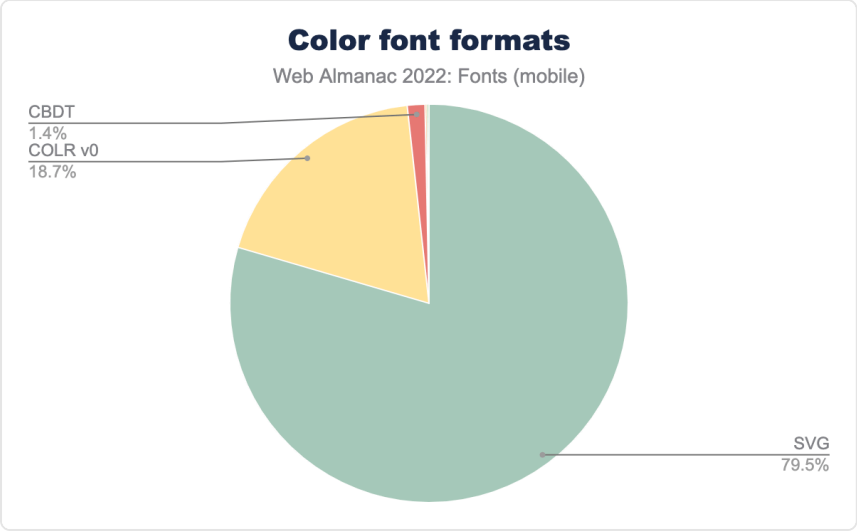



*図5.33. カラーフォントのフォーマット。*

カラーフォントの使用率は79%が SVG、19%が COLRv0、2%が CBDT です。

画像ベースのフォーマットは人気がないと結論づけられますが、それには理由があります：埋め込まれた画像はうまく拡大縮小できず、そのファイルサイズはウェブでの使用には適していません。

しかし、ベクターカラーフォントフォーマット間の分裂は、より微妙なものです。現時点では SVG が優勢に見えますが、COLR も依然として大きな使用率を誇っています。COLR フォーマットは、すべてのブラウザでサポートされていること、可変フォントで使用できること、そして実装が簡単であることなど、多くの利点がある。このような理由だけで、もっとも普及しているフォーマットになると予想されます。もっと皮肉な見方をすれば、GoogleがChromeとAndroidで SVG のサポートを実装することを拒否しているため、このフォーマットがもっとも普及しているだろうということです。興味深いことに、Appleは COLRv1 の実装を拒否しています。なぜなら COLRv1 の機能の多くは、すでに SVG フォーマットでサポートされているからです。残念ながら、ウェブ開発者はこの「カラーフォント戦争」の渦中に巻き込まれています。私たちは、この状況がすぐに解決され、私たち全員がカラーフォント

167. *https://www.typearture.com/variable-fonts/*

を使い始めることができるようになることを願っています。

CSSの仕様が更新され、カラーフォントに対応して、パレットの選択とカスタマイズ[168]が可能になりました。パレットとは、タイプデザイナーがフォントに保存したカスタムカラースキームのことです。CSSの `font-palette` プロパティでは、フォントからパレットを選択でき、`@font-palette-values` ルールでは、新しいパレットを作成したり既存のパレットを上書きできます。この技術のより明白な使用例の1つは、カラーフォントの中にライトモードとダークモードのパレットを組み込むことです。そこには、未開拓の可能性がたくさんあります。


*図5.34. David Jonathan Ross[169] によるCOLR v1とマルチパレットを使用したBradley Initials[170]。*

残念ながら、これらのCSSプロパティの使用は、現在では存在しません。これは、これらのプロパティのサポートが最近になってブラウザに追加されたことと、カラーフォントの数が限られていることが原因だと思われます。

カラーフォントの技術を発展させた大きな原動力のひとつが絵文字でした。しかし、カラー絵文字を持つWebフォントは数十種類しかありません。もっともカラーフォントはテキストを書くためのもので、絵文字には対応していない。これには、いくつかの説明が考えられます。

- どのOSもすでに独自のカラー絵文字フォントを搭載しているので、ユーザーはそれ以外のものを使う必要性を感じない。

168. https://css-tricks.com/colrv1-and-css-font-palette-web-typography/
169. https://djr.com/
170. https://tools.djr.com/misc/bradley-initials/

- 絵文字は数が多いので、そのフォントを作るのは大変な労力とお金がかかります。
- 絵文字フォントは一般的にかなり大きく、ウェブフォントほど適していません。

しかし、絵文字のフォントにもう少し多様性があってもいいのではないでしょうか。COLR v1フォーマットの導入により、今後、絵文字フォントを目にすることがより多くなりそうです。

これらはすべて非常に低い使用率に基づくものですが、いくつかの発展的な傾向があるように思われます。2023年をカラーフォントの年と宣言するのはまだ早いですが、今後数年間、カラーフォントが大きく成長することは間違いないようです。とくに、業界が推奨するカラーフォント形式を1つに定め、ブラウザのカラーフォントへの対応が進むにつれて、カラーフォントは大きく成長するでしょう。Googleフォントも、カラーフォントの第一弾[171]をライブラリに追加したばかりで、きっとインパクトがあるはずです。

## 結論

この章とその前の年を振り返ってみると、ウェブフォントサービスがどれほど大きな影響力を持っているか、そしておそらくこれからも持ち続けるであろうことがよくわかります。たとえばGoogleフォントだけで、ウェブフォントのもっとも多く、もっとも普及しているウェブフォントのほとんど、そして可変フォントのほとんどに対応しています。これはすごいことです。

私たちは、ウェブフォントの将来はセルフホスティングであると強く信じていますが、ウェブフォントサービスを利用することが多くの開発者にとって理にかなっていることも否定はできません。ウェブフォントサービスは使いやすく、もっとも高いパフォーマンス（ベストではありませんが）を提供し、もっともフォントのライセンスについて心配する必要がありません。これは良いトレードオフです。

一方セルフホスティングは今までになく簡単で、もっとも高いパフォーマンス、より多くのコントロール、そしてプライバシーの問題に悩まされることはないでしょう。セルフホスティングを計画する場合、WOFF2、リソースヒント、`font-display`を必ず使用してください。これらを組み合わせることで、あなたのサイトのフォント読み込みパフォーマンスにもっとも大きな影響を与えることができます。

ここ2、3年、Googleのおかげで、可変フォントが華々しく登場しました。もっともパフォーマンスが高いからという理由で使っている人が多いようですが、私たちは、このようなケースは、採用が革新をもたらすと信じています。今後、どのような楽しいタイポグラフィや

171. *https://material.io/blog/color-fonts-are-here*

クレイジーなタイポグラフィが登場するか、楽しみです。

カラーフォントについても、慎重に楽観視しています。利用がようやく増えてきたのです。技術は以前からあったのですが、カラーフォントのフォーマットに関する意見の相違や、CSSのサポートが限られていることが、採用の妨げになっています。これらがすぐに解決され、本当の意味での成長が見られるようになることを期待しています。

Webフォントの利用は、今後も増え続け、時代とともに進化していくことは間違いないでしょう。私たちは、未来がどうなるかを知りたいと思っています。Incremental Font Transfer[172]のようなテクノロジーは、より多くの文字システムにウェブ フォントを提供し、何十億もの人々がはじめてウェブ フォントを使い始めることを可能にします。それは楽しみですね。

### 著者

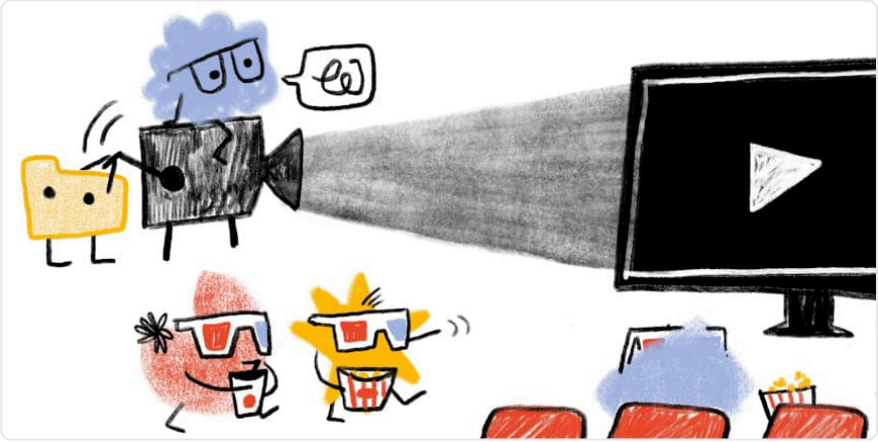
## Bram Stein

@bram_stein bramstein http://www.bramstein.com/

Bram Steinは開発者であり、プロダクトマネージャーです。タイポグラフィをこよなく愛し、デザインとテクノロジーの交差点で働くことに喜びを感じている。A Book ApartによるWebフォントハンドブック[173]とFontFace Observer[174]ライブラリの著者である。また、世界中のカンファレンスでタイポグラフィやウェブパフォーマンスについて講演している。

172. *https://www.w3.org/TR/IFT/*
173. *https://abookapart.com/products/webfont-handbook*
174. *https://fontfaceobserver.com*

# 部 I 章 6
# メディア



***Eric Portis と Akshay Ranganath によって書かれた。***
*Nicolas Hoizey と Yoav Weiss によってレビュ。*
*Eric Portis と Akshay Ranganath による分析。*
*Michael Lewittes 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

ハイパー テキスト であるにもかかわらず、ウェブは極めて視覚的です。実際、画像や動画はウェブユーザーの体験に欠かせない要素となっています。AVIF、ワイドカラー、アダプティブ・ビットレート・ストリーミング、レイジー・ローディングなどの新技術を採用し、ビジュアルウェブはどこまで進化したのか、そしてまだどこまで進化しなければならないのか、Web Almanacは私たちに調査するユニークな機会を与えてくれます。私は、アニメーションGIFを見ているのです。

さっそく潜入してみましょう。

## 画像

一般的なウェブサイトのページウェイトのうち、画像は大きな割合を占めています。ページ

ウェイトの章から、2021年6月の中央値ウェブサイトの総ウェイトは2,019キロバイト（モバイル時）で、そのうちの881キロバイトは画像だったことがわかります。これは、HTML（30KB）、CSS（72KB）、JavaScript（461KB）、フォント（97KB）の合計よりも多い。

**99.9%**

*図6.1. 画像リソースへのリクエストが少なくとも1回発生したページ。*

背景やファビコンであっても、ほとんどすべてのページで何らかの画像が提供されています。

**70%**

*図6.2. LCPの責任要素に画像があるモバイルページ。*

モバイルでは70%、デスクトップでは80%と、もっとも多くのページで、もっともインパクトのあるリソースは画像です。最大のコンテントフルペイント[175] (LCP)は、折り目の上にある最大の要素を特定するウェブパフォーマンス指標です。もっとも多くの場合、その要素は画像を持っています。

ウェブにおける画像の重要性について言い過ぎたことはない。では、ウェブの画像について、どのようなことが言えるのでしょうか。

## 画像リソース

まず、リソースそのものについて説明します。ビットマップイメージはピクセルでできています。一般的にWebの画像は何ピクセルあるのでしょうか？

175. *https://web.dev/articles/lcp*

### 1画素画像に関する注意点

| クライアント | 1x1画像 |
|---|---|
| モバイル | 7.3% |
| デスクトップ | 7.0% |

*図6.3. 1ピクセルだけを含む `<img>` 要素によって読み込まれるリソース。*

不審なほど多くのものが1×1です。これらの `<img>` は、画像コンテンツをまったく含んでいません。レイアウト用（スペーサーGIF[176]）とトラッキングビーコン[177]の2つの目的で使用されているのです。

新しく作成されたウェブサイトは、レイアウトにCSSを使い、トラッキングにビーコンAPI[178]を使うべきです。既存の多くのコンテンツは、トラッキングピクセルとスペーサーGIFを永遠に使い続けるだろうが、ここのデスクトップ数は昨年[179]から変わっておらず、モバイル数はほんの少ししか縮小していないことに落胆させられる。古い習慣[180]こごえる[181]！

可能な限り、これらの実際には画像ではない `<img>` を分析から除外しました。

176. *https://en.wikipedia.org/wiki/Spacer_GIF*
177. *https://ja.wikipedia.org/wiki/%E3%82%A6%E3%82%A7%E3%83%96%E3%83%93%E3%83%BC%E3%82%B3%E3%83%B3*
178. *https://developer.mozilla.org/en-US/docs/Web/API/Beacon_API*
179. *https://almanac.httparchive.org/ja/2021/media#fig-5*
180. *https://developers.facebook.com/docs/meta-pixel/implementation/marketing-api#intialize-img*
181. *https://spacergif.org/stats/*

### 画像寸法

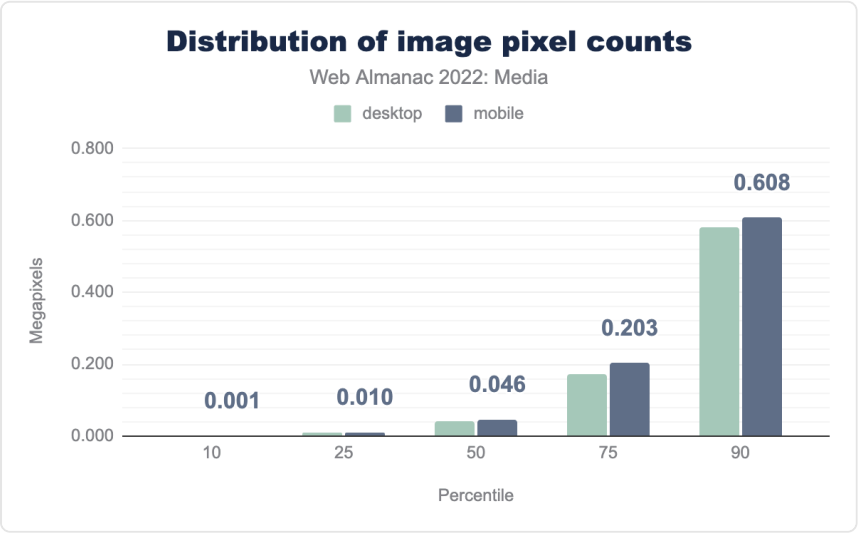



*図6.4. 画像の画素数の分布。*

1ピクセル以上を含む画像に移るもっとも多くのページには、少なくとも1つの大きな画像が含まれているのですが、その画像はかなり小さいです。

「メガピクセル」は、画像サイズを表すもっとも直感的な指標ではありません。たとえば、アスペクト比4:3の場合、中央の画素数0.046MPは、248×186の画像になります。

これは小さいと思われるかもしれませんが、中央のページには少なくとも1つの `<img>` が含まれており、そのピクセル数は中央の `<img>` 要素よりも10倍近く多くなっています。

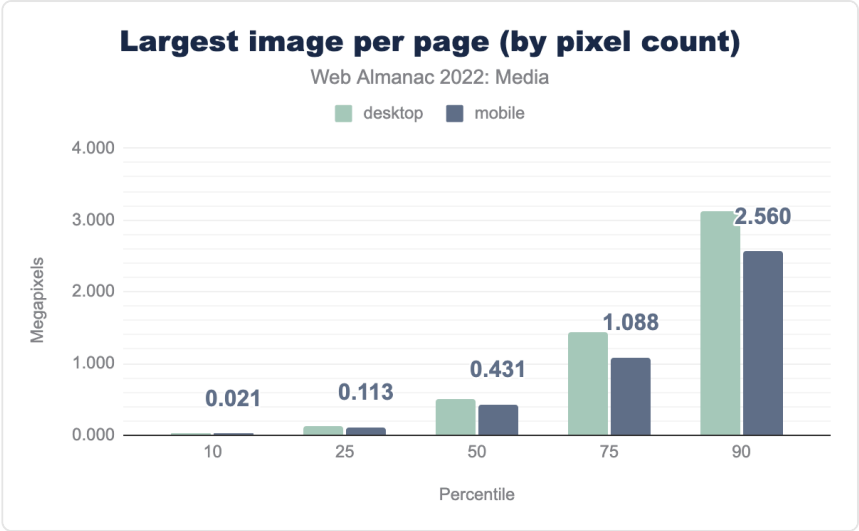



*図6.5. 1ページあたりもっとも大きな画像（画素数で）。*

アスペクト比4:3の場合、0.431MPは758×569に相当する。モバイルクローラーのビューポートが（一般的な）360px幅であることを考えると、これらの大きな画像の多くは、ビューポートのほぼ全体に、高い密度で描かれることになると思われます。

要するに、ほとんどの画像は小さいが、もっとも多くのページに少なくとも1枚の大きな画像が含まれている。

### 画像のアスペクト比

Webではどのようなアスペクト比が一般的なのでしょうか？

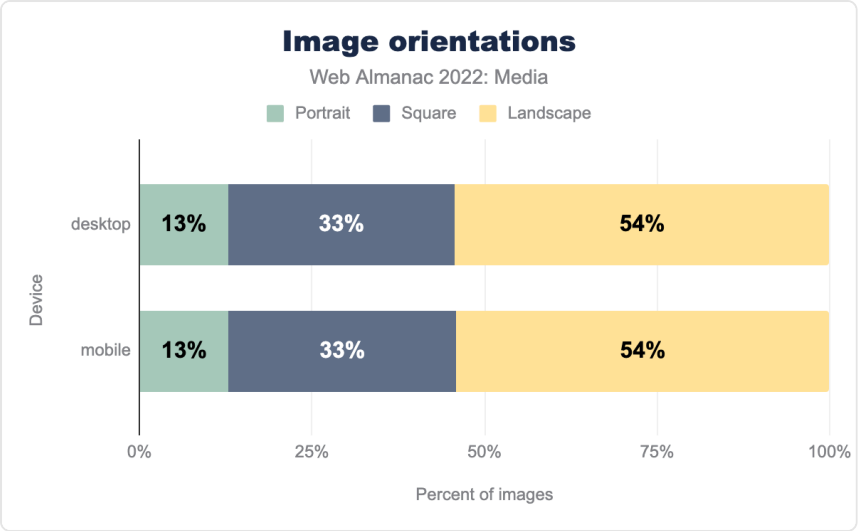



*図6.6. 画像の向き。*

昨年ともっとも似ている[182]、ほとんどの画像が横長で、モバイルとデスクトップの数値にほぼ差が、ないことがわかります。2021年と同様、これは大きな機会損失のように感じられます。多くのインスタグラマーが知っているように縦長の画像は、モバイル画面では、正方形や横長の画像よりも全幅で大きく表示され[183]、高いエンゲージメント[184]を促進する。ソースが横長であっても、アート ディレクション[185]を使用して、モバイル画面用に画像を調整することは可能ですし、そうすべきです。

182. *https://almanac.httparchive.org/ja/2021/media#アスペクト比*
183. *https://uxdesign.cc/the-powerful-interaction-design-of-instagram-stories-47cdeb30e5b6*
184. *https://www.dashhudson.com/blog/best-picture-format-instagram-dimensions*
185. *https://web.dev/codelab-art-direction/*

| アスペクト比 | 画像の割合% |
|---|---|
| 1:1 | 32.92% |
| 4:3 | 3.99% |
| 3:2 | 2.74% |
| 2:1 | 1.66% |
| 16:9 | 1.62% |
| 3:4 | 1.02% |
| 2:3 | 0.72% |
| 5:3 | 0.54% |
| 6:5 | 0.48% |
| 8:5 | 0.47% |

*図6.7. 画像のアスペクト比がもっとも多いランキング表（モバイル）。*

画像のアスペクト比は、4：3、3：2、とくに1：1（正方形）といった「標準的」な値に集中していた。実際、全画像の40%がこの3つのアスペクト比のいずれかを持っており、上位10個のアスペクト比は全`<img>`の約半分を占めていました。

### 画像の色空間

画像は画素でできており、各画素は色を持っています。ある画像内で可能な色の範囲は、その画像の色空間[186]によって決定されます。

Webのデフォルトの色空間はsRGB[187]です。CSSの色はデフォルトでsRGBを指定され、とくに指定がない限りブラウザは画像の色もsRGBであると仮定します[188]。

これは、ほぼすべてのディスプレイやキャプチャーハードウェアがsRGBまたはそれに近いものを扱っていた世界では理にかなっていました。しかし、時代は変わりつつあるのです。2022年には、もっとも携帯電話のカメラがsRGBよりも広い色域で撮影するようになります。また、sRGB以外の豊かな色彩を表現できるディスプレイハードウェアもかなり普及しています。

広色域ディスプレイに対応した最新のブラウザは、sRGBより広い色域で画像をエンコード

186. *https://ja.wikipedia.org/wiki/%E8%89%B2%E7%A9%BA%E9%96%93*
187. *https://en.wikipedia.org/wiki/SRGB*
188. *https://imageoptim.com/color-profiles.html*

すれば、sRGB外の鮮やかな色を喜んで描画します。しかし、そうでしょうか？

要するに、ダメなんです。

ある画像がRGB以外の色空間を使用していることをブラウザに伝えるには、一般的に画像の色空間を記述したICCプロファイル[189]を添付する必要があります。そのICCプロファイルには名前がついています。ウェブ上で使用されているユニークなICCプロファイルの名前を25,000以上見つけました。ここでは、もっとも多い20個を紹介します：

*189. https://ja.wikipedia.org/wiki/ICC%E3%83%97%E3%83%AD%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB*

| ICC プロファイルの説明 | sRGB っぽい | 広色域 | 画像の割合% |
|---|---|---|---|
| タグなし | ✓ | | 90.17% |
| sRGB IEC61966-2.1 | ✓ | | 3.23% |
| c2ci | ✓ | | 2.40% |
| sRGB | ✓ | | 0.88% |
| Adobe RGB (1998) | | ✓ | 0.76% |
| uRGB | ✓ | | 0.54% |
| ディスプレイ P3 | | ✓ | 0.35% |
| c2 | ✓ | | 0.33% |
| ディスプレイ | | | 0.30% |
| sRGB内臓 | ✓ | | 0.24% |
| GIMP内蔵sRGB | ✓ | | 0.22% |
| sRGB IEC61966-2-1黒スケール | ✓ | | 0.19% |
| 一般的なRGB プロファイル | | | 0.06% |
| U.S. Web Coated (SWOP) v2 | | | 0.04% |
| sRGB MozJPEG | ✓ | | 0.02% |
| Artifex ソフトウェア sRGB ICC プロファイル | ✓ | | 0.02% |
| Dot Gain 20% | | | 0.02% |
| Coated FOGRA39 (ISO 12647-2:2004) | | | 0.01% |
| Apple Wide カラーシェアリングプロファイル | | ✓ | 0.01% |
| sRGB v1.31 (Canon) | ✓ | | 0.01% |
| HD 709-A | ✓ | | 0.01% |

*図6.8. ICC色空間記述のもっとも多い20個のランキングリスト（モバイル）。*

ウェブ上の画像10枚のうち9枚はタグ付けされておらず、RGBデータを含んでいればsRGBとして解釈されます。残りの10%のほとんどは、sRGBまたはそれに近いタグが明示的に付けられています。「c2ci」、「uRGB」、「c2」はすべてsRGBの亜種で、最小かつ軽量である

ように設計されています[190]。ウェブ上の全画像のうち、sRGBより広い色域でタグ付けされているのは1%強に過ぎません。もっと簡潔に言うと、広色域画像は現在、グレースケール画像と同じくらいウェブで人気があります。ウェブ画像の1.16%を占めています[191]。

AVIFとPNGは、ICCプロファイルを使用せずに、フォーマット固有のショートハンドを使用して広色域の色空間を持つ画像にタグ付けできます。ICCプロファイルを使用しないワイドガモットAVIFとPNGの検出を試みましたが、エンコードのさまざまな方法と、ツールによる報告の方法を考慮すると、今年取り組むには少し複雑すぎることがわかりました。来年に期待しましょう！

## エンコード

さて、Webの画像コンテンツについて少し理解できましたが、そのコンテンツがどのようにエンコードされて配信されるのかについて、どのようなことが言えるのでしょうか。

### フォーマット採用

GIF、JPEG、PNGは、何十年もの間、ウェブ上の標準的なビットマップ画像ファイルフォーマットとして使われてきました。それが変わり始めたのは、2014年にChromeがWebPのサポートを出荷してからです。この2、3年で、その変化は加速しています。SafariとFirefoxはWebPのサポートを出荷し、3つの主要なブラウザはすべて、少なくともAVIFの実験的サポートを出荷しました。

フォーマット別に、クローラーが見たすべての画像リソースを紹介します。

190. *https://github.com/saucecontrol/Compact-ICC-Profiles*
191. *https://docs.google.com/spreadsheets/d/1T5oVAVmcH3sM6R-WwH4ksr2jFtPhuLXs3-iXXoABb3E/edit?pli=1#gid=560546690&range=P5*

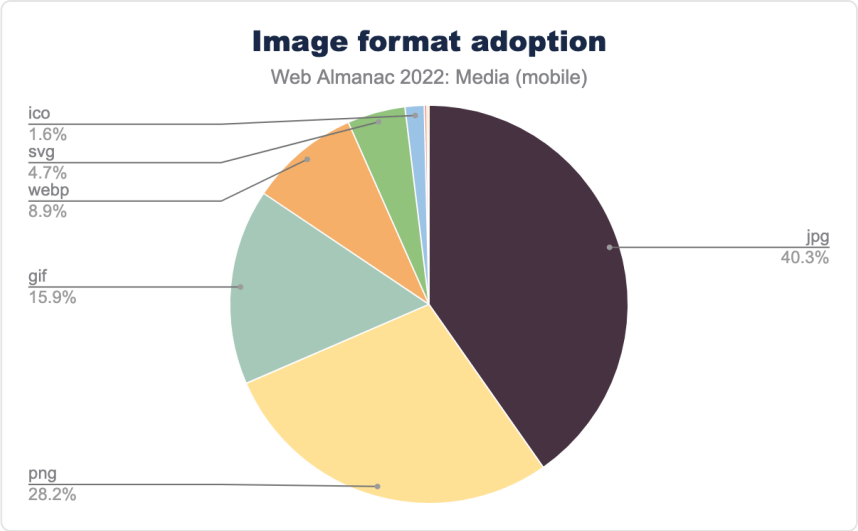



*図6.9. 画像フォーマットの採用。*

0.22%というAVIFのパイは、グラフに表示されていないほど小さなものです。0.22%という数字は、昨年と比較すると、あまり大きくないように思われるかもしれませんが、かなりの進歩があったことを意味します。

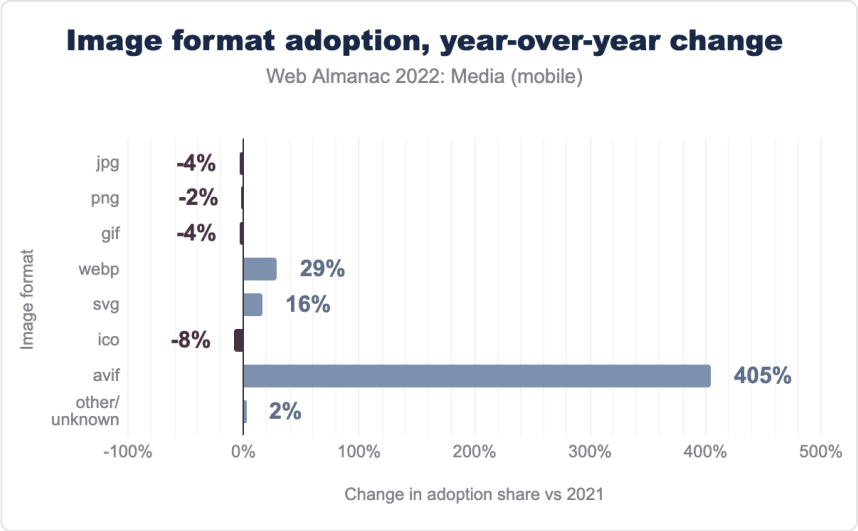



*図6.10. 画像フォーマットの採用状況、前年比。*

少しずつ、古いフォーマットから新しいフォーマットへの移行が進んでいます。そうあるべきでしょう！新フォーマットは、旧フォーマットをかなりの差で凌駕しています。それは、まもなく実感できることでしょう。

### バイトサイズ

Web上の一般的な画像はどのくらい重いのでしょうか？

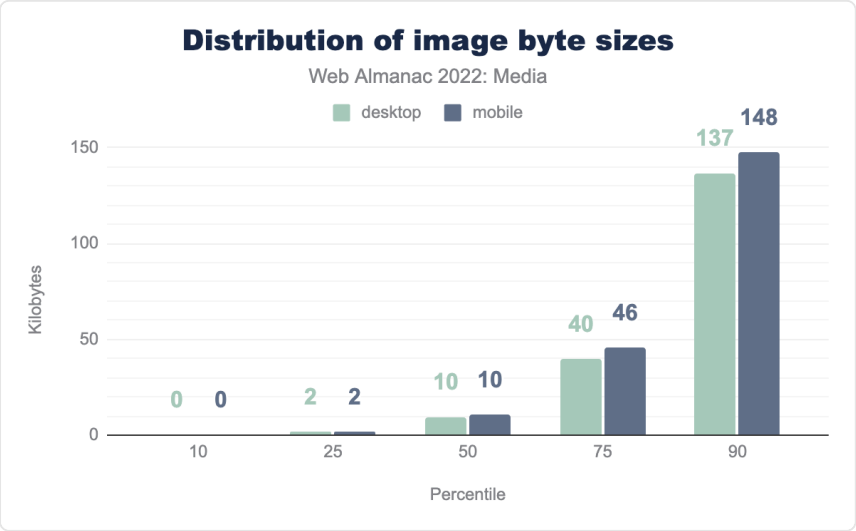



*図6.11. 画像のバイトサイズの分布。*

中央値が10KBだと、「え、そんなに重くないよ！」と思われるかもしれません。しかし、画素数を見たときと同じように、小さな画像が多い一方で、もっとも多くのページには少なくとも1つの大きな画像があります。

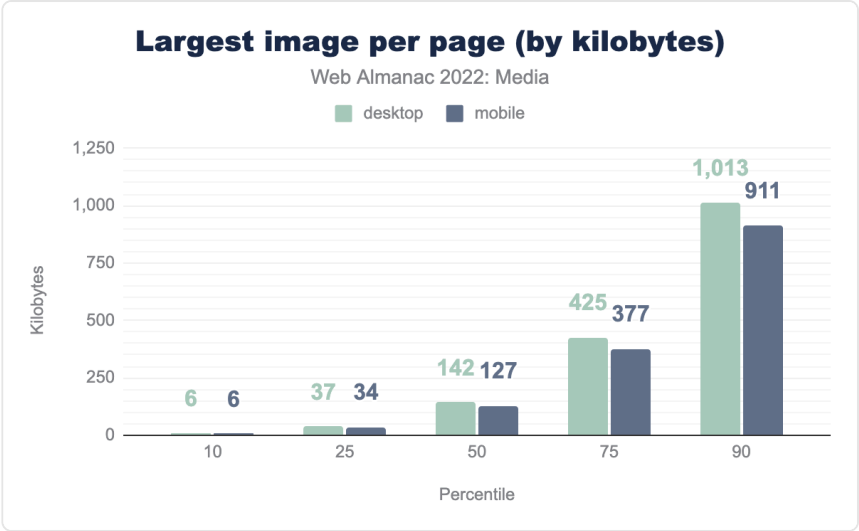



*図6.12. 1ページあたりもっとも大きな画像（キロバイト単位）。*

もっとも多くのページで100KBを超える画像が少なくとも1枚あり、上位10%のページではほぼ1MB以上の重さの画像が少なくとも1枚ある。

### 画素あたりのビット数

バイト数やピクセル数だけでもおもしろいですが、ウェブ上の画像データの圧縮率を知るには、バイト数とピクセル数を合わせて1ピクセルあたりのビット数を算出する必要があります。これにより、解像度が異なる画像であっても、その情報密度を比較することができるようになります。

一般に、Web上のビットマップは、1チャンネル、1ピクセルあたり8ビットの情報にデコードされます。つまり、透明度のないRGB画像の場合、デコードされた非圧縮画像は1ピクセルあたり24ビット[192])になると予想されます。可逆圧縮の経験則は、ファイルサイズを2:1の比率で削減することです（8ビットRGB画像の場合、1ピクセルあたり12ビットになります）。1990年代の非可逆圧縮方式であるJPEGやMP3では、10:1（1ピクセルあたり2.4ビット）の比率が目安でした。画像のコンテンツやエンコーディングの設定によって、これらの比率は大きく変化し、MozJPEG[193] などの最新のJPEGエンコーダーは、デフォルト設定でこの10:1目標を上回ることが一般的であることに注意すべきです。要約すると、次のようになります。

192. *https://ja.wikipedia.org/wiki/%E8%89%B2%E6%B7%B1%E5%BA%A6#%E3%83%88%E3%82%A5%E3%83%AB%E3%83%BC%E3%82%AB%E3%83%A9%E3%83%BC%EF%BC%8824/32%E3%83%93%E3%83%83%E3%83%88*
193. *https://github.com/mozilla/mozjpeg*

| ビットマップデータの種類 | 期待される圧縮率 | 画素あたりのビット数 |
|---|---|---|
| 非圧縮RGB | 1:1 | 24ビット/画素 |
| ロスレス圧縮されたRGB | ~2:1 | 12ビット/画素 |
| 1990年代のロッシーRGB | ~10:1 | 2.4ビット/画素 |

図6.13. ビットマップRGBデータの代表的な圧縮率と圧縮結果のビット/ピクセル数。

では、このような背景を踏まえて、Webの画像をどのように評価するか、ご紹介しましょう。

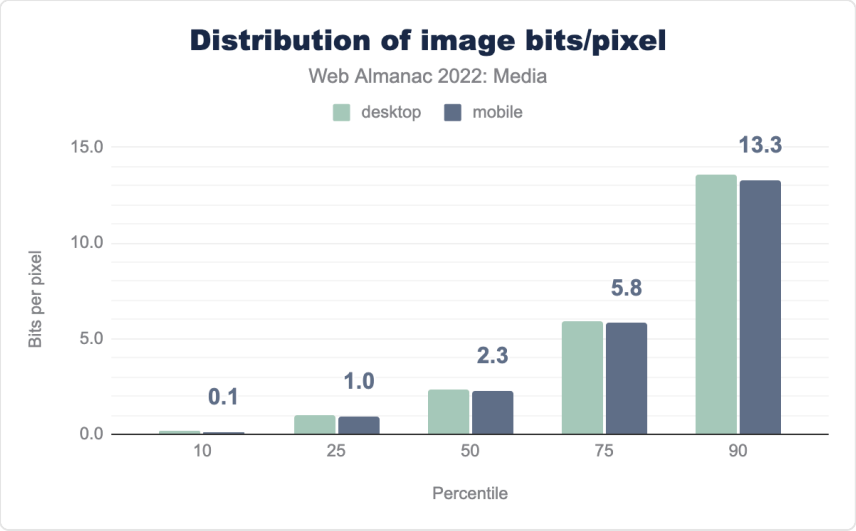



図6.14. 画像ビット/ピクセルの分布。

モバイルの `<img>` の中央値は、1ピクセルあたり2.3ビットで、圧縮率10:1の目標をほぼ達成できます。しかし、その中央値の周辺には、非常に大きな広がりがあります。もう少し詳しく知るために、フォーマット別に分類してみましょう。

### 画素あたりのフォーマット別ビット数

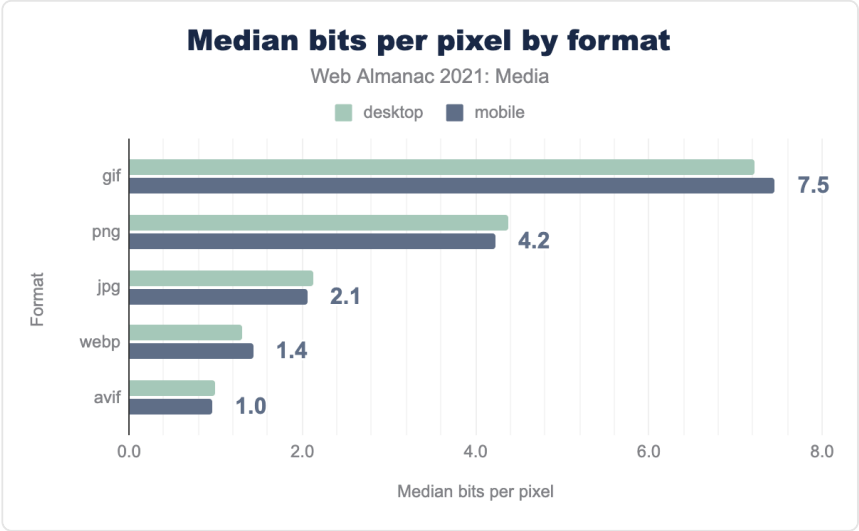



*図6.15. フォーマット別の1画素あたりの中央値ビット数。*

これらの数字のほとんどは、昨年[194]と基本的に変わっていません。

PNGは技術的には「可逆圧縮」技術を採用しています（アルファチャンネルを扱うかどうかにもよりますが、1ピクセルあたり12〜16ビットを想定しています）、そのエンコーダーは一般に非可逆的であることがわかります。圧縮率を高めるために、画像を「可逆圧縮」する前に、カラーパレットを減らし、ディザリングパターンを導入するのです。

そして、典型的なWebPは、典型的なJPEGよりも1ピクセルあたり3分の1軽くなっていることがわかります。これは、私たちが期待するところとほぼ同じです。 正式な研究[195]では、品質比較[196]を用いて、WebPがJPEGよりもほぼ同じマージンで優れていると推定しています。

昨年と比較して、唯一大きく動いたのはAVIFです。このフォーマットは、昨年の1ピクセルあたり1.5ビットから1.0ビットへと、WebPよりも圧縮率を下げています。これは非常に大きな減少ですが、まったく予想外だったわけではありません。AVIFは非常に若いフォーマットで、エンコーダーは素早く反復し、その採用は著しく広がっています。来年には、AVIFの中央値がさらに圧縮されていることでしょう。

非可逆圧縮と品質のトレードオフの品質面を見なければ、この結果だけでは、AVIFがWeb対応フォーマットの中で「もっとも良い」圧縮を提供していると結論づけることはできない。

194. *https://almanac.httparchive.org/ja/2021/media#フォーマット別、画素あたりのビット数*
195. *https://developers.google.com/speed/webp/docs/webp_study*
196. *https://kornel.ski/en/faircomparison*

しかし、今年は、実世界での使用において、AVIFはもっとも圧縮率が高いという結論を出すことができます。この結論と、in-the-lab結果[197]のように、品質を保つのに良い仕事をすることを示唆する結果を組み合わせると、画像はかなり良く見え始めます（冗談です）。

AVIFのブラウザ サポート[198]も今年、大きな飛躍を遂げました。つまり、ウェブ上でビットマップ画像を送信するのであれば、99.9%のページがそうであるように、少なくともAVIFの送信を検討すべきなのです。

### GIF、アニメーション、そうでないもの

圧縮チャートのもう一方の端には、私たちの古い友人であるGIFがあります。これはとくに悪いように見えますが、このフォーマットのせいばかりではありません。この35年前のフォーマットがいまだによく使われている理由のひとつは、アニメーションができることで、画素数を計算する際にフレーム数を考慮していないのです。このことは、いくつかの興味深い問題を提起しています。まず、アニメーションを行うGIFはどれくらいあるのでしょうか？


*図6.16. モバイルでアニメーションされたGIFの割合。*

私はこれが意外に低いことに気づきました。PNGが2006年に普遍的なサポートを獲得して以来[199]、非アニメーションGIFを出荷する正当な理由は ありませんでした[200]。「GIF」という言葉は、その唯一の正当な使用例と同義語になっています。 短い、無音、自動再生、ループするアニメーションのためのポータブルでユニバーサルなフォーマットであることです。これらの非アニメーションGIFはすべてレガシーコンテンツなのか、それとも相当数の新しい非アニメーションGIFが作成されウェブに公開されているのか、疑問が残るが、そうでなければいい！

アニメーションGIFと非アニメーションGIFを分けたところで、非アニメーションGIFとアニメーションGIFの圧縮特性はどうなっているのか？

197. *https://netflixtechblog.com/avif-for-next-generation-image-coding-b1d75675fe4*
198. *https://caniuse.com/avif*
199. *https://caniuse.com/?search=png*
200. *https://en.wikipedia.org/wiki/Portable_Network_Graphics#Compared_to_GIF*

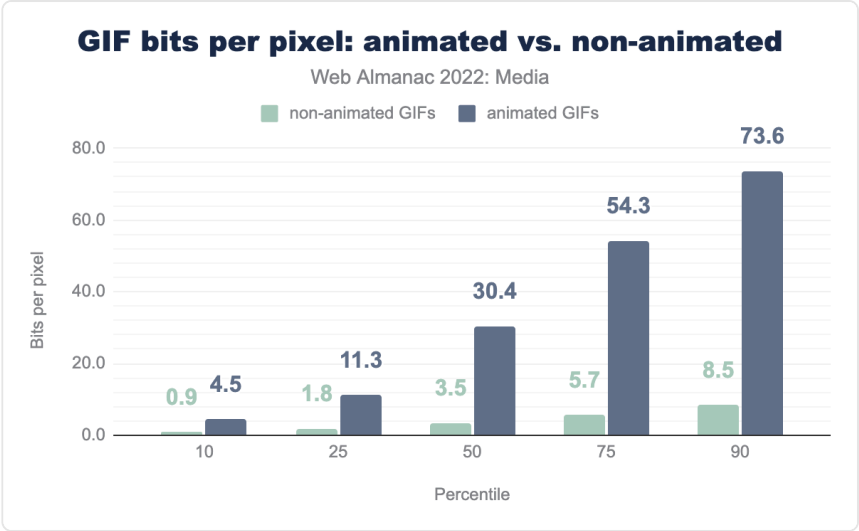



*図6.17. GIFの1ピクセルあたりのビット数：アニメーションと非アニメーションの比較。*

アニメーションGIFを除外してみると、このフォーマットはもっと良くなります。GIFは1ピクセルあたり3.5ビットの中央値で、PNGよりもピクセル単位で小さくなっています。これは、各フォーマットが圧縮を要求されるコンテンツの種類を反映していると思われます。GIFは、設計上、256色と2値の透明度しか含むことができません。GIFは設計上、256色と2値の透明度しか持つことができませんが、PNGは1670万色と完全なアルファチャンネルを持つことができます。

GIFの話をする前に、GIFについてもう1つ質問があります。GIFアニメは通常何フレームあるのでしょうか？

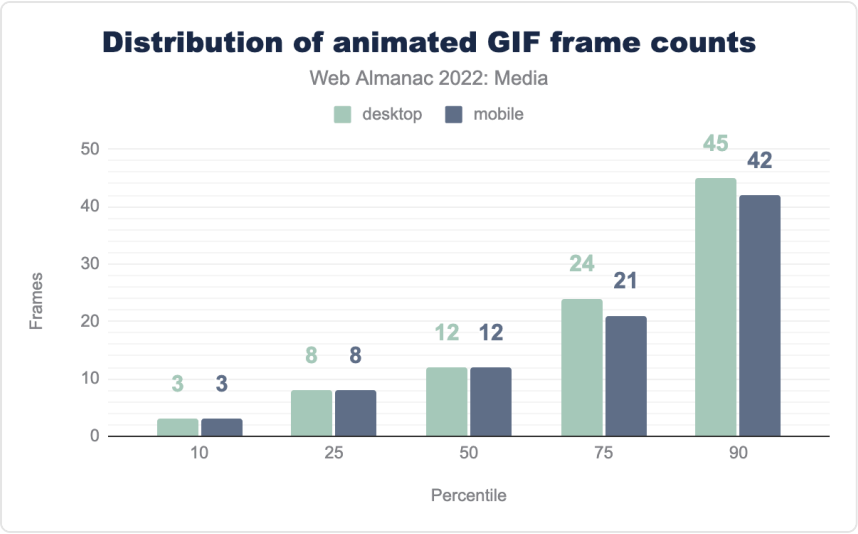



*図6.18. アニメーションGIFのフレーム数の分布。*

GIFアニメの大半は、十数フレーム以下で登場します。ちなみに、私たちが見つけたGIFのもっとも多いフレーム数は15,341フレームでした。30FPSで計算すると、8分半のGIFになる。心が揺れ動く。

## エンベデッド

さて、ウェブの画像リソースがどのようにエンコードされているのかがわかったところで、それらがどのようにウェブページに埋め込まれるのかについて、どのようなことが言えるのでしょうか。

## レイジーローディング

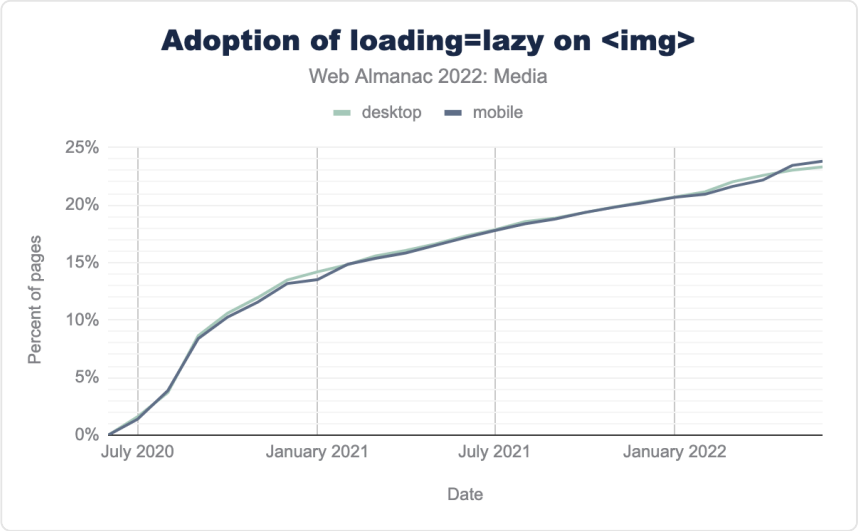



*図6.19. `<img>` に `loading=lazy` を採用した。*

昨年の最大の話題は、レイジーローディングの急速な普及でした。採用のペースが落ちたとはいえ、驚くべき速度で進んでいます。昨年6月、レイジーローディングを採用していたページは17%でした。それが今年は1.4倍に増えました。現在では24%のページがレイジーローディングを利用しています。

ウェブ上の膨大なレガシーコンテンツを考えると、2年間でゼロからクロールされるページの4分の1程度になったというのは驚くべきことで、ネイティブの遅延ローディングに対する需要がいかに大きかったかを示しています。

**9.8%**

*図6.20. モバイルでネイティブレイジーローディングを使用しているLCP `<img>` の割合。*

そして実際に、昨年と同様に、ページが遅延ローディングを少し *多用* しているようです。

LCP要素のレイジーローディングは、LCPのスコアをより悪くする。ページを遅くするアンチパターンです。10個に1個のLCP `<img>` がレイジーローディングされていることを見ると、がっかりします。このアンチパターンが昨年から少しずつ増えているのを見ると、なおさらです。

## `alt` テキスト

`<img>` 要素で埋め込まれた画像は、コンテントフルであることが前提です。つまり、「単なる飾りではなく、意味のあるものが含まれていなければならない」ということです。WCAGの要件[201] と HTML仕様[202]の両方によると、すべての内容のある画像には代替テキストが必要で、その代替テキストは通常 `alt` 属性が提供すべきです。

**54%**

*図6.21. 空白でない `alt` 属性を持つイメージの割合。*

この結果は、全 `<img>` のほぼ半分が明らかにアクセシブルでないことを意味します。今年のアクセシビリティの章の詳細な分析が示唆するところでは、空白でない `alt` 属性を持つ `<img>` の大部分も、それほどアクセシブルではないのです。

私たちはもっとうまくやれるし、やらなければならない。

## `srcset`

Lazy-loadingの前に、ウェブ上の `<img>` に起こった最大の出来事は、「レスポンシブ・イメージ」のための一連の機能で、レスポンシブ・デザインに適合するように画像を調整することを可能にしました。2014年にはじめて出荷された `srcset` 属性、 `sizes` 属性、そして `<picture>` 要素は、10年近く前から作者が適応性の高いリソースをマークアップすることを可能にしています。私たちは、これらの機能をどの程度、どのように使っているのでしょうか？

まず、 `srcset` 属性から説明します。この属性は、文脈に応じてブラウザにリソースの選択メニューを与えることができます。

**34%**

*図6.22. `srcset` 属性を使用しているページの割合。*

3分の1のページが `srcset` を使っているが、3分の2は使っていない。2022年のレスポンシブデザインにおけるフルードグリッドの普及を考えると、 `srcset` を使用していないページ

201. https://www.w3.org/WAI/WCAG22/Understanding/non-text-content
202. https://html.spec.whatwg.org/multipage/images.html#alt

の中には、使用すべきものがたくさんあるのではないでしょうか？

`srcset` 属性は、2つの記述子のうちの1つを使用してリソースを記述することを可能にします。`x` ディスクリプターはリソースがどの画面密度に適しているかを指定し、`w` ディスクリプターは代わりにリソースの幅をピクセルでブラウザに与えます。`sizes` 属性と組み合わせて使用することで、`w` 記述子はブラウザが流動的なレイアウト幅と可変的な画面密度の両方に適したリソースを選択することを可能にします。

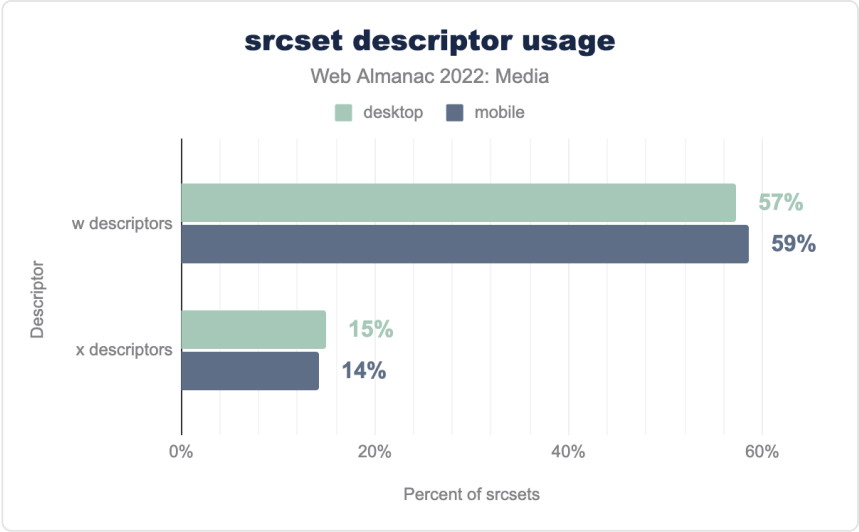



*図6.23. `srcset` ディスクリプターの使用法。*

`x` ディスクリプターは最初に登場したものであり、よりシンプルに考えることができます。何年もの間、より強力な `w` ディスクリプターよりも人気を博していました。10年近く経って、世界が `w` ディスクリプターを受け入れるようになったことは、私の心を温かくしてくれます。

## `sizes`

先ほど、`w` ディスクリプターは `sizes` 属性と組み合わせて使うべきと書きました。私たちは `sizes` をどのように使っているのでしょうか？二つ返事でOKです。あまりうまくはありません。

`sizes` 属性は、ブラウザに画像の最終的なレイアウトサイズを示すヒントとなるもので、通常はビューポートの幅に相対するものです。画像のレイアウト幅に影響を与える変数がたくさんあります。`sizes` 属性は明示的にヒントであることを想定しているので、多少の不

正確さは問題ありませんし、むしろ期待されています。しかし`sizes`属性が少しばかり不正確であれば、リソース選択に影響を与え、実際の画像のレイアウト幅が大きく異なるのに、`sizes`の幅に合うようにブラウザが画像を読み込んでしまうことがあります。

So how accurate are our `sizes`?

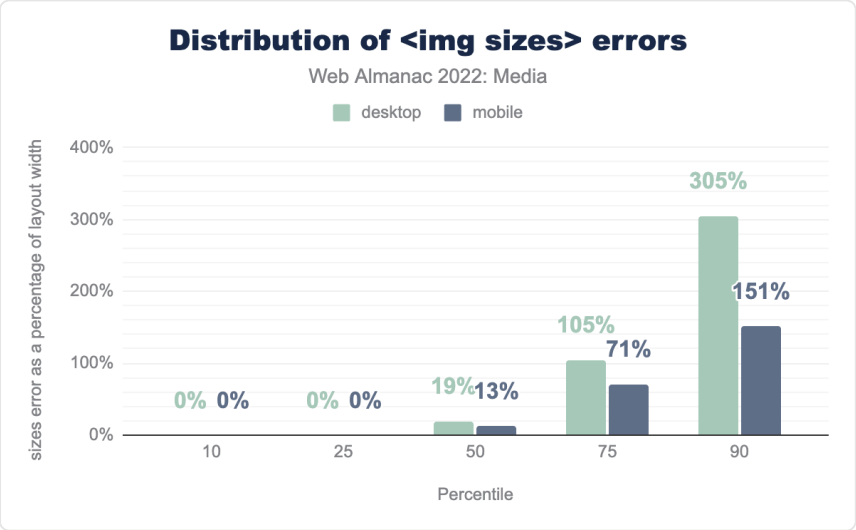



*図6.24. `<imgサイズ>`のエラーの分布。*

多くの`sizes`属性は完全に正確ですが、`sizes`属性の中央値は、モバイルでは13%、デスクトップで19%大きすぎることがわかります。しかし、ご覧のように、p75とp90の数値は美しくなく、悪い結果につながります。

**19%**

*図6.25. デスクトップでは`srcset`の選択に影響を与えるほど不正確な`sizes`属性がありました。モバイルでは、14%です。*

デスクトップでは、デフォルトの`sizes`値(`100vw`)と実際の画像のレイアウト幅の差がモバイルより大きい可能性が高いため、5つに1つの`sizes`属性が不正確で、ブラウザが`srcset`から最適でないリソースを選択する原因となっています。このようなエラーは積み重なります。

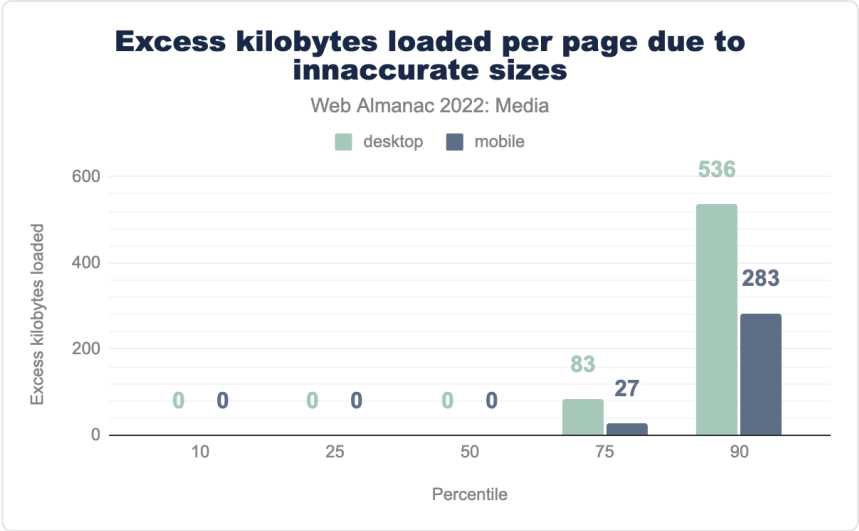



*図6.26. 不正確なサイズにより、1ページあたり過剰なキロバイトが読み込まれます。*

私たちは、デスクトップページの4分の1が、純粋に悪い `sizes` 属性に基づいて、83KB以上の余分な画像データを読み込んでいると推定しています。つまり、次のようなことです。`srcset` にもっと小さくて良いリソースがあるのに、`sizes` 属性が誤っているため、ブラウザがそれを選択しないのです。さらに、sizesを使用するデスクトップページの10%は、間違ったsizes属性のために、5メガバイト以上の過剰な画像データを読み込んでいます！

*注：当社のクローラーは実際には正しいリソースを読み込まなかったため、ここに記載した数字は、クローラーが実際に読み込んだ誤ったリソースのバイトサイズに基づく推定値です。*

短期的に個々の開発者はRespImageLint[203] を使用して、ひどく壊れた `sizes` 属性を監査して修正し、この種のムダを防止できますし、そうすべきです。

中期的には、可能であれば、ウェブプラットフォームはより良いツールを提供する必要があります。多くの開発者にとって、正確な `sizes` 属性を作成し、維持することは、あまりにも困難であることが証明されました。遅延ロードされた画像の自動サイズ[204]を可能にする提案は、テーブル上にあります。2023年に進展することを期待しましょう。

lazysizes.js library[205]は、この種のソリューションに対する意欲をすでに証明しています。10%の `sizes` 属性は現在、JavaScriptの実行前に値 “auto” を持ち、画像を遅延ロードする前にlazysizes.jsによって後で完全に正しい値に書き直されます。遅延ロードに依存しているた

203. https://ausi.github.io/respimagelint/
204. https://github.com/whatwg/html/pull/8008
205. https://github.com/aFarkas/lazysizes

め、このパターンはLCP画像や、折りたたみの上にある `<img>` 要素には適していないことに注意してください。これらの画像の場合、レスポンシブ・ローディングのパフォーマンスを向上させる唯一の方法は、よくオーサリングされた `sizes` 属性を指定することです。

### `<picture>`

2014年に上陸した最後のレスポンシブ画像機能は、 `<picture>` 要素でした。 `srcset` がブラウザにリソースのメニューを渡すのに対し、 `<picture>` 要素は作者が主導権を握り、ブラウザにどの子 `<source>` 要素からリソースを読み込むかについて明示的に指示できます。

`<picture>` 要素は、 `srcset` に比べ、使用頻度が圧倒的に少ない。


*図6.27. モバイルページのうち、 `<picture>` 要素を使用している割合。*

これは昨年より2ティックほど増えていますが `<picture>` を使うページ1つに対して `srcset` を使うページが5つ近くあるということは、 `<picture>` のユースケースがよりニッチであるか、デプロイがより困難であるか、その両方であることを示しています。

人々は何のために `<picture>` を使っているのでしょうか？

`<picture>` 要素は、リソースを切り替えるための2つの方法を提供します。Type-switchingは、最先端の画像フォーマットをサポートするブラウザに提供し、それ以外のブラウザにはフォールバックフォーマットを提供できます。メディアスイッチでは、アートディレクション[206]を促進し、メディアの状況に応じてさまざまな `<source>` を切り替えることができるようになります。

206. https://www.w3.org/TR/respimg-usecases/#art-direction

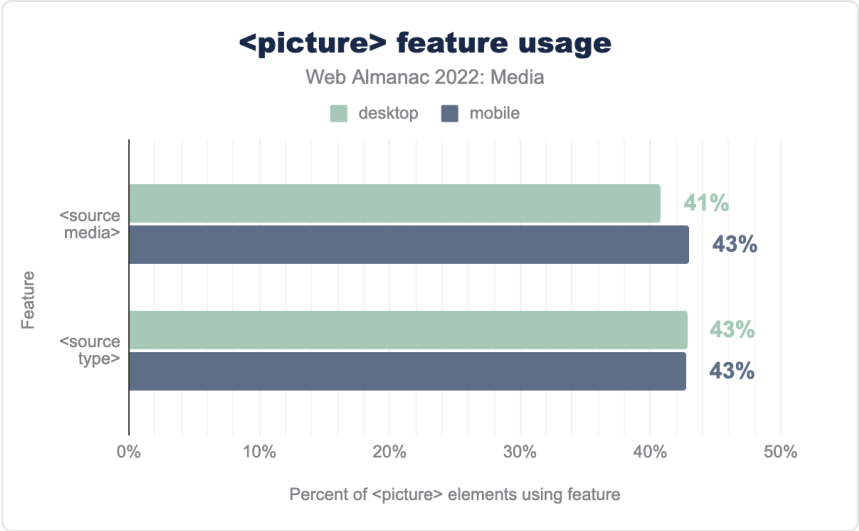



*図6.28. `<picture>` 機能の使い方です。*

使い方はそれなりに均等に分かれている。興味深いのは、昨年[207]からタイプスイッチの差が縮まっていることだ。これは、AVIFやWebPといった次世代画像フォーマットの普及が進んでいることと関係していると思われます。

## レイアウト

レスポンシブ画像の難しさは、HTMLを書くときに `<img>` がどのように配置されるかを考えなければならないことです。そこで、基本的な疑問が生まれます。 `<img>` はどのように配置されるのか？

ウェブの画像リソースがどのようにサイズアップするかはすでに見たとおりです。しかし、ユーザーに表示される前、埋め込み画像はレイアウト内に配置されなければならず、レイアウトに合わせて縮小されたり引き伸ばされたりする可能性があります。

この分析を通じて、クローラーのビューポートに留意することが有用です。デスクトップクローラーは幅1376px、DPRは1倍、モバイルクローラーは幅360px、DPRは3倍でした。

### レイアウト幅

ここで一番シンプルな疑問は、こうかもしれません。 Webの画像は、ページに描かれたと

207. https://almanac.httparchive.org/ja/2021/media#fig-23

き、どれくらいの幅になるのでしょうか。

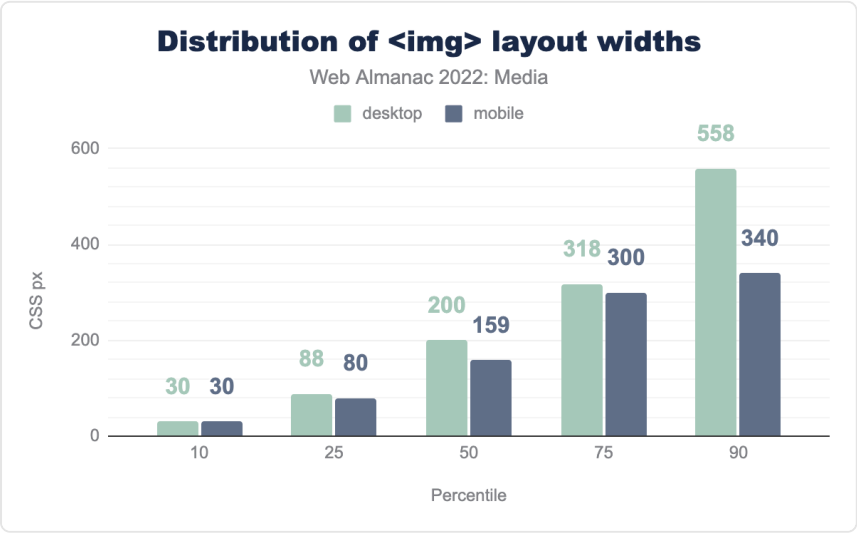



*図6.29. `<img>` のレイアウト幅の分布。*

埋め込むリソースと同じように、ウェブ上の画像の多くは、レイアウトの中ではかなり小さなものになります。同様に、ほとんどのページには、かなり大きな画像が少なくとも1枚はあります。

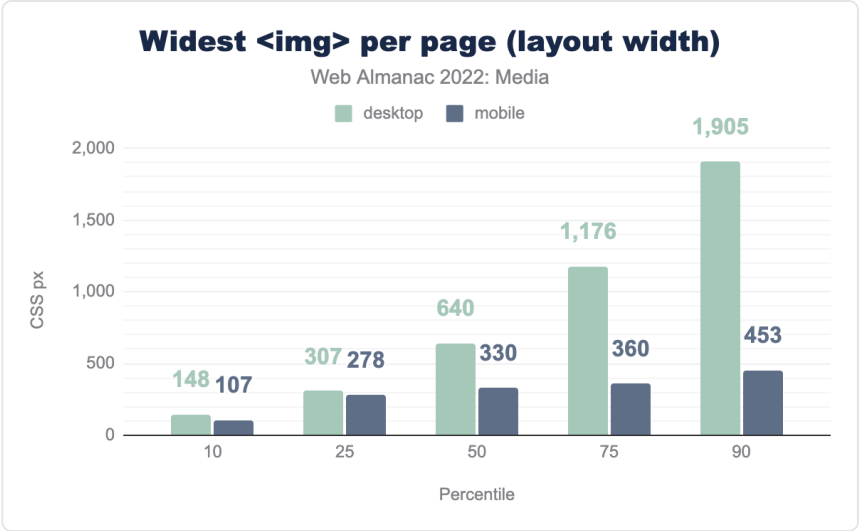



*図6.30. 1ページでもっとも幅の広い `<img>` を表示します（レイアウト幅）。*

モバイルページの75%以上が、75vw以上のビューポートを占める画像を少なくとも1つ持っています。その後、徐々に増加し、最終的にはかなりの数（10〜25%）のページで、ビューポートより広い画像が表示されるようになります。これは、作者がビューポートメタタグ[208]を記述しておらず、デスクトップサイズのページがモバイル画面内に収まるように縮小されているためと思われます。

デスクトップのレイアウト幅と対比してみるとおもしろいのですが、デスクトップのレイアウト幅は頂点に達することはありません。幅はどんどん大きくなっています。デスクトップのページの10%以上が、クローラーのビューポート1360pxを超える幅の画像を含んでおり、おそらく水平スクロールバーを引き起こしていると思われるのは驚きです。

### 本質的なサイジングと外在的なサイジング

なぜWebの画像はこのようなレイアウトサイズになってしまうのでしょうか？CSSで画像を拡大縮小する方法はたくさんあります。しかし、CSSをまったく使わずに画像を拡大縮小しているものはどれくらいあるのでしょうか？

画像は、すべての"置換された要素"[209]と同様に、固有のサイズ[210]を持っています。デフォルトでは、密度を制御する `srcset` やレイアウト幅を制御するCSSルールがない場合、ウェブ上の画像は1xの密度で表示されます。640×480の画像を `<img src>` に配置すると、デフォル

208. https://developer.mozilla.org/ja/docs/Web/HTML/Viewport_meta_tag
209. https://developer.mozilla.org/docs/Web/CSS/Replaced_element
210. https://developer.mozilla.org/docs/Glossary/Intrinsic_Size

トでその `<img>` は640CSSピクセル幅でレイアウトされます。

作者は、イメージの高さ、幅、またはその両方に、外付けのサイズ設定を適用できます。もし画像が一方の次元で外在的な大きさ（たとえば、`width: 100%;` のルール）を与えられ、他方の次元では本来の大きさ（`height: auto;` またはまったくルールなし）に任されていた場合、その画像は本来の縦横比を使って比例的に拡大されます。

さらに複雑なことに、いくつかのCSSルールでは、何らかの制約に違反しない限り、`<img>` はその本来の大きさで表示されます。たとえば、`max-width: 100%;` ルールを持つ `<img>` 要素は本質的な大きさになりますが、その本質的な大きさが `<img>` 要素のコンテナーのサイズより大きい場合は、外在的に縮小されてフィットします。

さて、ここまでの説明でWebの `<img>` という要素は、どのようにレイアウトされるのか、その大きさを説明します。

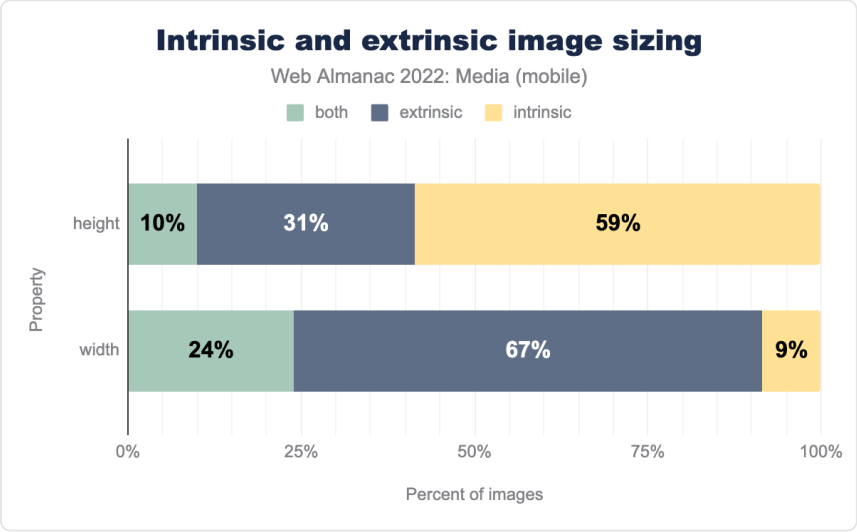



*図6.31. 内在的・外在的なイメージサイジング。*

大半の画像は外付けの幅を持ち、大半の画像は内付けの高さを持つ。幅の「両方」カテゴリは、`max-width` または `min-width` のいずれかのサイズ制約がある画像を表しますが、これもかなり人気があります。画像を本来の幅にすることは、はるかに人気がなく、昨年[211]よりもわずかに人気がないようです。

211. *https://almanac.httparchive.org/ja/2021/media#内在サイジングと外在サイジング*

### `height`、`width`、累積レイアウトシフト

レイアウトサイズが固有の幅に依存する `<img>` は、累積レイアウトシフト[212]を引き起こす危険があります。要するに、このような画像は2回レイアウトされる危険性があります。1回目はページのDOMとCSSが処理されたとき、2回目は読み込みが終了して本来のサイズが判明したときです。

先ほど見てきたように、高さ（とアスペクト比）はそのままにして、ある幅に合うように画像を外挿的にスケーリングすることは非常によくあることです。結果として生じるレイアウトのずれに対抗するため、数年前、ブラウザは `<img>` の `height` と `width` 属性の動作方法を変更することにしました。最近では、リソースの縦横比を反映するように `height` と `width` 属性を一貫して設定することが、普遍的に推奨されるベストプラクティスとなっており、これにより作者は画像リソースの読み込み前、ブラウザにその固有の寸法を伝えることができます。

**28%**

*図6.32. モバイルの `<img>` 要素に `height` と `width` の両方の属性が設定されている割合です。*

残念ながら、ユニバーサル・アダプションに至るまでには、長い道のりがあります。

## デリバリー

最後に、ネットワーク上で画像がどのように配信されるかを見てみましょう。

### クロスドメインイメージホスト

埋め込み先のドキュメントと異なるドメインから配信されている画像はどれくらいある？昨年より3.6ポイント増[213]など、過半数を占める。

212. https://web.dev/i18n/ja/cls/
213. https://almanac.httparchive.org/ja/2021/media#クロスオリジン画像ホスト

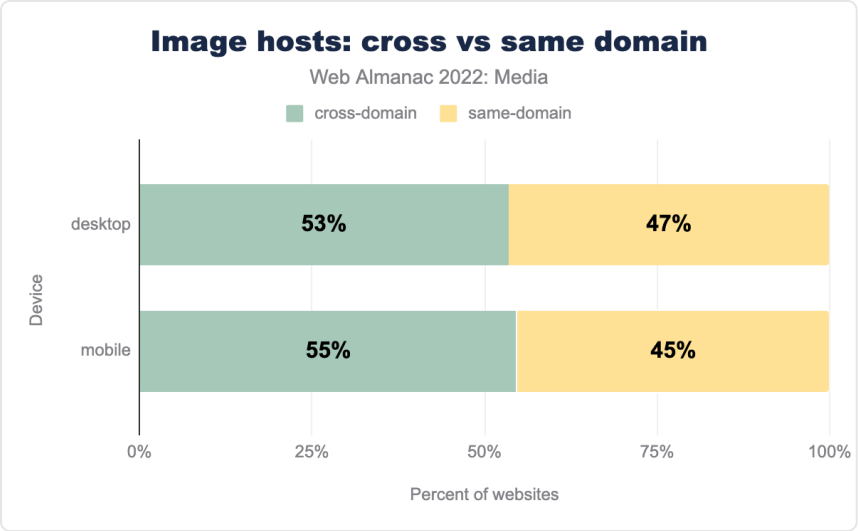



*図6.33. イメージホスト：クロスドメインと同一ドメインの比較。*

画像の大半がドメイン間で配信されるようになっているという事実は、画像を正しく扱うことがいかに難しいか[214]を明確にし、イメージCDN[215]にメディアの処理を依頼するメリットを示しています。

そして、次は `<img>` の若くてダイナミックな兄弟である `<video>` に注目しましょう。

## ビデオ

2010年に登場した `<video>` 要素は、FlashやSilverlightなどのプラグインが廃止されて以来、Webサイトにビデオコンテンツを埋め込むための最良の、そして唯一の方法となっています。

ここ数年、ウェブコンテンツが変化していることを実感しています。かつては静止画（Flickr、Instagram）が主流でしたが、最近は動画（TikTok）が主流になってきています。この感覚は、Web Almanacのデータセットでも実証されているのでしょうか。私たちはウェブ上で `<video>` をどのように使っているのでしょうか？

214. https://css-tricks.com/images-are-hard/
215. https://web.dev/i18n/ja/image-cdns/

## ビデオ採用

`<video>` 要素の利用が増え続けている。

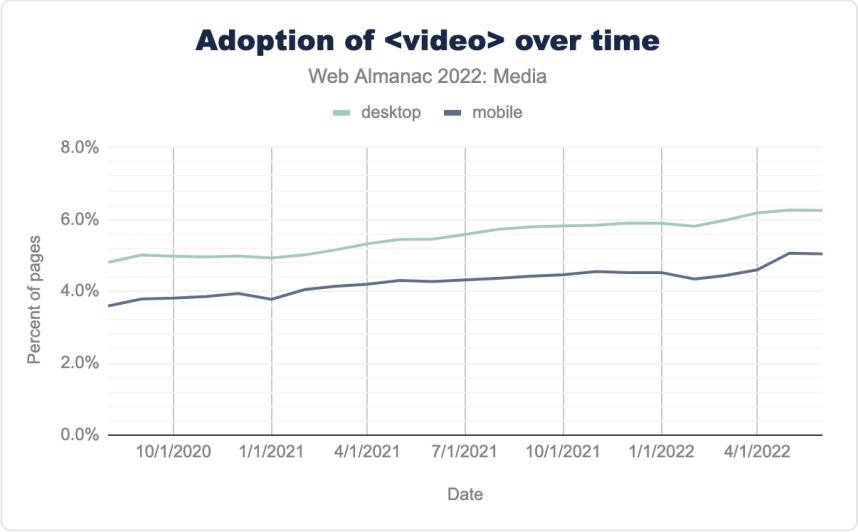



*図6.34. `<video>` の経時的な採用状況。*

モバイルでは、 `<video>` の使用率が2021年6月のページの4.3%から2022年6月のページの5%に上昇しました。20ページに1ページが `<video>` 要素を含むようになり、前年比18%増となりました。ウェブに `<img>` と同じ数の `<video>` が登場することはないでしょうが、`<video>` の数は年々増加しています！

## ビデオの持続時間

その動画はどれくらいの長さなんですか？あまりありません！

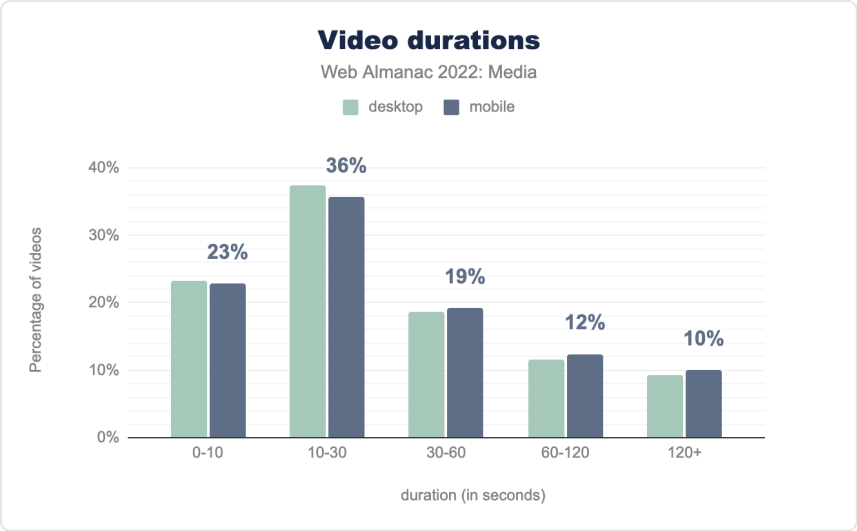



*図6.35. ビデオの持続時間です。*

10本中9本の動画が2分以内。そして半数以上が30秒以下です。ほぼ4分の1の動画は10秒以下です。もしかしたら、これらは`<video>`の服を着たGIFなのかもしれません。

## フォーマット採用

2022年、サイトが配信するフォーマットは？ユニバーサル・サポート・ストーリー[216]を持つMP4は、王者です。

216. *https://caniuse.com/mpeg4*

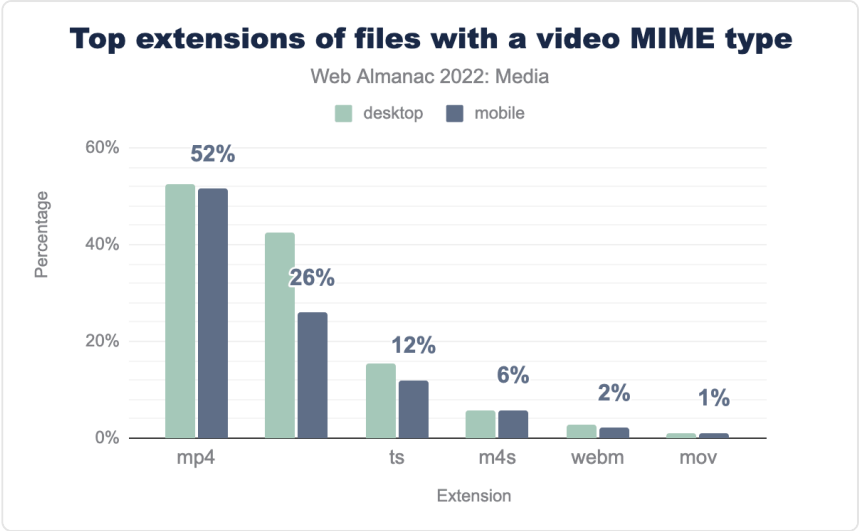



*図6.36. ビデオのMIMEタイプを持つファイルの上位拡張子。*

しかし、MP4の数字は昨年[217]から数ティック下がっており、空白の拡張子を持つファイル、`.ts`ファイル、`.m4s`ファイルの増加が続いている。これらのファイルは、`<video>`がHLS[218]またはMPEG-DASH[219]を使用してアダプティブ・ビットレート・ストリーミングを採用している場合に配信されます。

アダプティブストリーミングを使用したレスポンシブな動画配信が増加傾向にあるのは心強いことです。同時に、Webプラットフォームが、JavaScriptに依存しない、アダプティブビデオのためのシンプルで宣言的な解決法[220]を提供してくれることを期待しています。

## エンベデッド

`<video>`要素には、ビデオをどのように読み込んでページ上に表示するかを制御するための属性がいくつか用意されています。ここでは、これらの属性を使用頻度の高い順に紹介します。

217. *https://almanac.httparchive.org/ja/2021/media#fig-29*
218. *https://en.wikipedia.org/wiki/HTTP_Live_Streaming*
219. *https://en.wikipedia.org/wiki/Dynamic_Adaptive_Streaming_over_HTTP*
220. *https://github.com/whatwg/html/issues/6363*

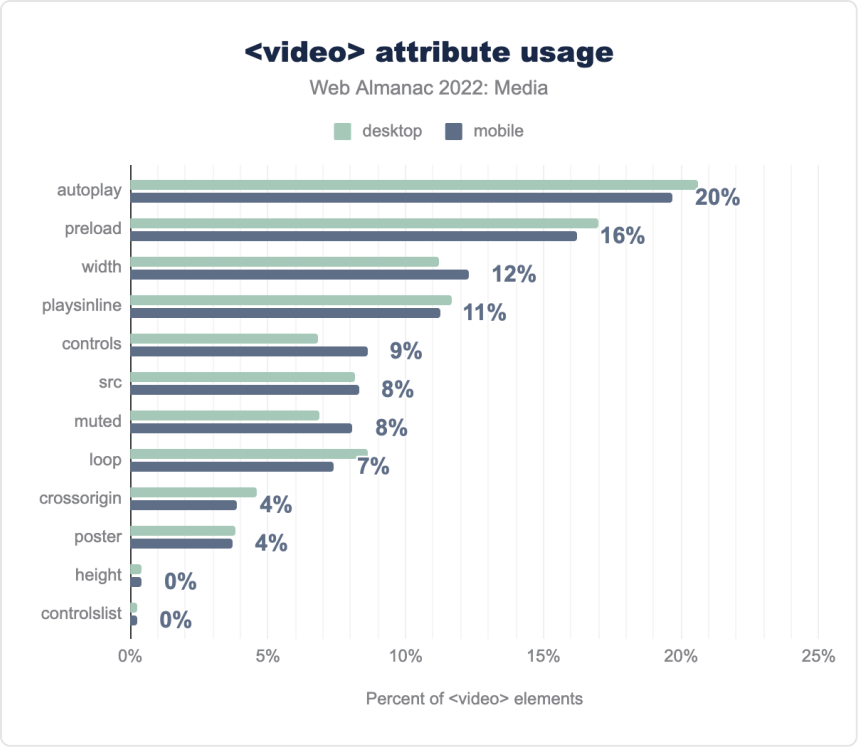



*図6.37. `<video>` 属性の使い方。*

ここで解き明かすべきことはいくつもあります。

まず、`autoplay` が `preload` を抜いて今年もっとも人気のある属性となりました。また、`playsinline`、`muted`、`loop` の人気も高まっているようです。おそらく、アニメーションGIFを置き換えるために `<video>` 要素を使う人が増えているのでしょう？もしそうなら、いいことだ！

つまり、ほとんどの `<video>` 要素は、これらの属性を持たない `<img>` 要素で見られたのと同じ種類の CLS[221] 問題の影響を受けやすいということです。ブラウザの助けを借りて、これらの属性を追加してください！

また、`<video>` 要素に `controls` 属性を持つものが10個に1個以下であることから、かなりの数の人が動画と対話するための独自のユーザーインターフェイスを提供するプレイヤーを使用していることがわかります。

221. *https://web.dev/articles/cls*

`preload` の使い方は、もう少し検討する必要がある。

### `preload`

`preload` 属性は、ここ数年、使用頻度が低下しています。

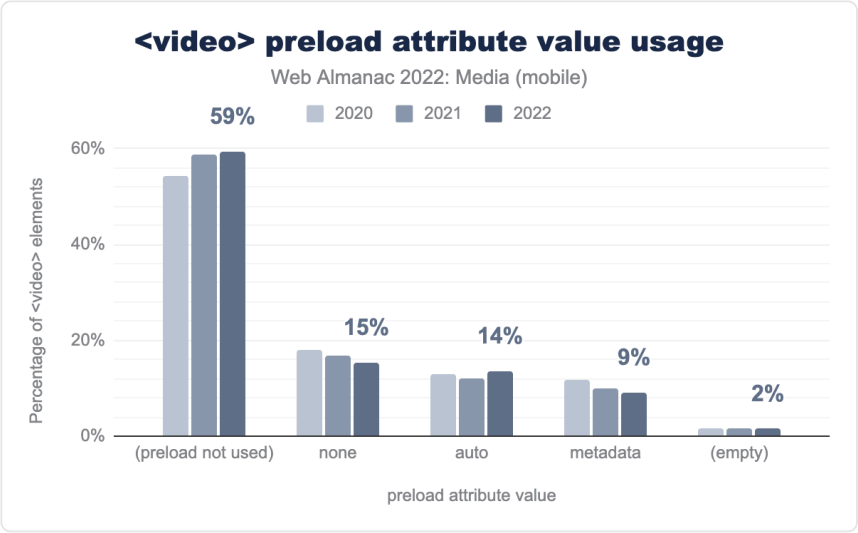



*図6.38. `<video preload>` 属性値の使い方です。*

なぜか？私は、著者がブラウザの邪魔をしないようにするためだと思いたい。

動画データを読み込むタイミングは、ブラウザによって異なる。`preload` 属性は、作者がそのプロセスをよりコントロールできるようにするための方法です。たとえば、`none` でブラウザにプリロードしないよう明示的に指示したり、`metadata` だけをプリロードするように指示したり、`auto` または空の値を使用してブラウザにプリロードするように指示したりすることができる。過去3年間、作者が動画の読み込みをあまりコントロールできなかったことは興味深く、おそらく心強いことでしょう。ブラウザは、ユーザーのコンテキストをもっともよく知っています。プリロード属性をまったく含めないことで、彼らが最善と考えることを行うことができます。

### `src` と `<source>`

`src` 属性は `<video>` 要素の8-9%にしか存在しません。残りの多くは、複数の `<source>` 子要素を使用し、作者が複数の代替フォーマットのビデオリソースを提供できるようにしています。

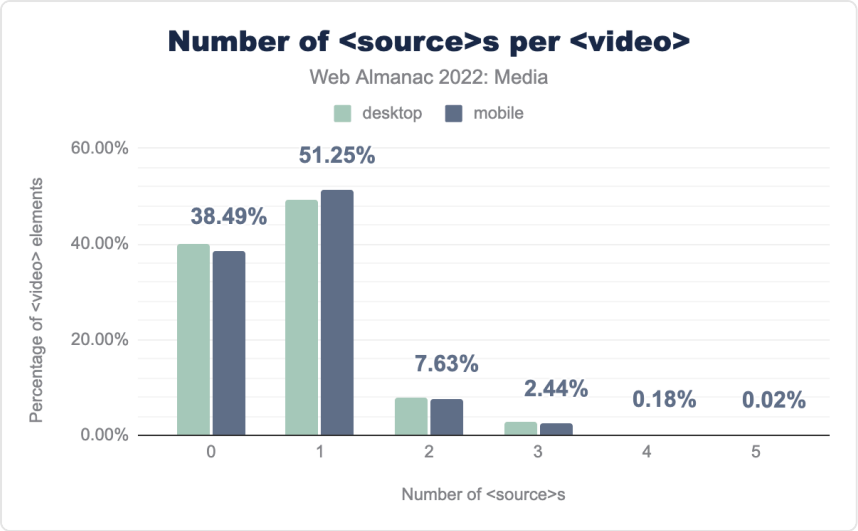



*図6.39.* `<video>` *毎の* `<source>` *の数。*

`<video>` 要素には、いくつの子要素がありますか？ほとんどは1つだけで、複数使うものはほとんどありません。

## 結論

2022年におけるウェブ上のメディア状況のスナップショットをご覧ください。画像や動画がいかにウェブに浸透しているか、そしてウェブの画像や動画がどのようにエンコードされ、埋め込まれているか、その一端を知ることができたと思います。今年のもっともエキサイティングな動きは、AVIFの採用が加速していることと、レイジーローディングとアダプティブ・ビットレート・ストリーミングの採用が増え続けていることです。

しかし、広色域の色空間がほとんどないこと、GIFという永遠のゾンビフォーマット（アニメーションと非アニメーションの両方）、 `sizes` 属性とレイジーローディングというパフォーマンスのために設計された2つの機能が（不適切な使用によって）かなりの数のページでパフォーマンスに影響を与えていることなど、不満な点もいくつかありました。

2023年、ウェブ上でより効果的なビジュアルコミュニケーションを実現するために！

## 著者


### Eric Portis

@etportis eeeps https://ericportis.com

Eric PortisはCloudinary[222]のウェブプラットフォーム提唱者です。

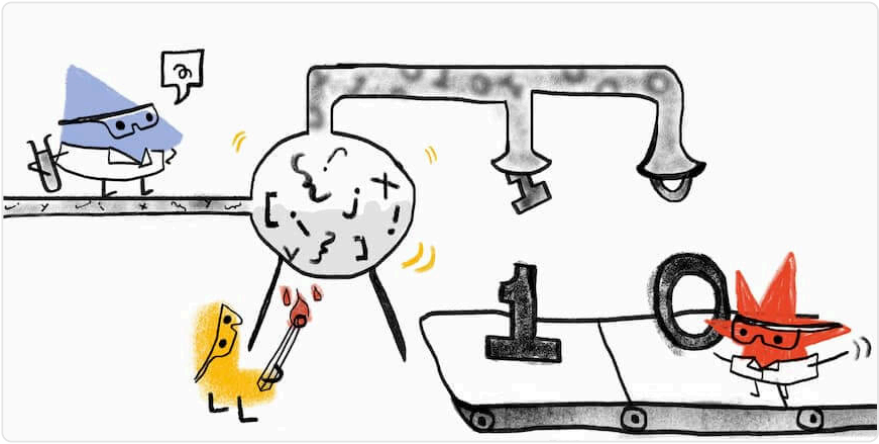
### Akshay Ranganath

@rakshay akshay-ranganath akshayranganath https://akshayranganath.github.io/

Akshay RanganathはCloudinary[222]のSr.Solution Architectで、CDN/WebPerfの課題に取り組むことが好きです。


222. *https://cloudinary.com/*
223. *https://cloudinary.com/*

# 部 I 章 7
# WebAssembly



*Colin Eberhardt によって書かれた。*
*Ben Smith と Ingvar Stepanyan によってレビュ。*
*Jamie Macdonald による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

WebAssemblyまたはWasmは、2019年12月に公式に認められたW3C標準となり、Web技術のファミリー（JavaScript、HTML、CSS）の中では比較的新しい存在です。

WebAssemblyはブラウザに新しいランタイムを導入し、JavaScriptランタイムと一緒に、そして密接に連携して動作する。WebAssemblyは、小さな命令セットと厳格な分離モデル（WebAssemblyはデフォルトでI/Oを持たない）を持ち、比較的に軽量です。WebAssemblyを開発する主な動機の1つは、幅広いプログラミング言語（C++、Rust、Goなど）用のコンパイルターゲットを提供し、開発者がより広いツールセットで新しいWebアプリケーションを書いたり、既存のアプリケーションを移植したりできるようにすることでした。

WebAssemblyの有名な例としては、Google Earth[224] の中で使用されており、C++ デスクトッ

224. *https://blog.chromium.org/2019/06/webassembly-brings-google-earth-to-more.html*

プアプリケーションがブラウザ内で利用できるようになっています。Figma[225] はこの技術を使って大幅にパフォーマンスを改善したブラウザベースのデザイン ツールで、最近では Photoshop[226] が同様の理由でWebAssemblyを使っていました。

## 方法論

WebAssemblyはコンパイル対象であり、バイナリモジュールとして配布されています。そのため、Web上での利用状況を分析する際には、さまざまな課題に直面します。WebAssemblyを含む最初の版である2021年版Web Almanacでは、使用した方法論の詳細なセクション[227]、および関連する注意事項が含まれています。2022年版のここでの調査結果は、同じ方法論に従ったものです。追加された唯一の強化点は、WebAssemblyモジュールのオーサリングに使用された言語を判定するメカニズムです。その解析の精度については、それぞれの項目で詳しく取り上げています。

## **WebAssembly**はどの程度使われているのですか？

デスクトップで3,204件、モバイルで2,777件のWebAssemblyリクエストが確認されました。これらのモジュールは、デスクトップでは2,524ドメイン、モバイルでは2,216ドメインで使用されており、デスクトップとモバイルの全ドメインの0.06%と0.04%に相当します。

これは、クロールで発見したモジュールの数が控えめに減少したことを表しており、デスクトップで16%、モバイルで12%の減少となっています。これは必ずしもWebAssemblyが衰退していることを意味するものではなく、この変化を解釈する場合、以下の点に注意する必要があります。

- WebAssemblyを使用してあらゆる種類のWebベースのコンテンツを作成できますが、その主な利点は、大規模なコードベースを持つより複雑な業務用アプリケーションで、何年も前のものであることが多いです（例：Google Earth、Photoshop、AutoCADなど）。これらのWeb「アプリ」は、Webサイトほど多くはなく、WebAssemblyがあまり普及していないホームページを主な対象とするAlmanacのクロールでは常に利用できるわけではありません。
- 後のセクションで見るように、私たちが見るWebAssemblyの使用の多くは、比較的少数のサードパーティライブラリから来るものです。その結果、これらのライブラリのどれか1つでも小さな変更があれば、見つかるモジュールの数に大きな影響を与えることになります。

225. *https://www.figma.com/blog/webassembly-cut-figmas-load-time-by-3x/*
226. *https://web.dev/ps-on-the-web/*
227. *https://almanac.httparchive.org/ja/2021/webassembly#方法論*

モバイルブラウザに提供されたWebAssemblyモジュールがわずかに少ない（-13%）ことがわかりました。これは、一般的に優れたサポートを持つモバイルブラウザのWebAssembly能力を反映しているわけではありません。むしろ、プログレッシブ・エンハンスメント[228]の標準的な実践によるものと思われます。この場合、WebAssemblyを必要とするより高度な機能は、モバイルユーザーにはサポートされていません。

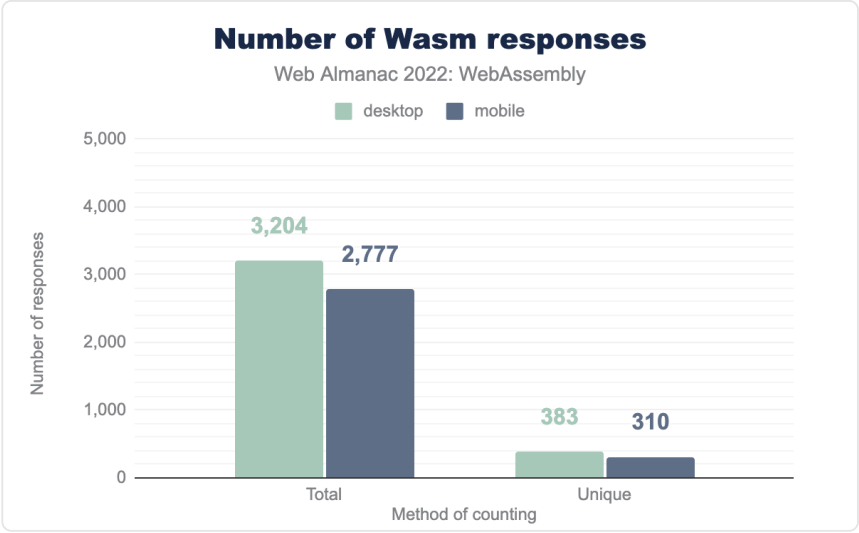



*図7.1. Wasmの対応数。*

WebAssemblyのモジュールをハッシュ化することで、デスクトップ用の3,204個のモジュールのうち、いくつがユニークなモジュールなのかを特定できます。モジュールの重複を排除することで、総数はおよそ10分の1になり、ユニークなモジュールはデスクトップブラウザに383個、モバイルに310個提供されています。これは、異なるウェブサイトが同じWebAssemblyコードを利用し、おそらく共有モジュールによって再利用されていることを意味します。

wasmリクエストのかなりの割合がクロスオリジンであり、再利用されているという考えをさらに強めています。とくに、これは昨年から大幅に増加しています（67.2%対55.2%）。

228. https://developer.mozilla.org/ja/docs/Glossary/Progressive_Enhancement

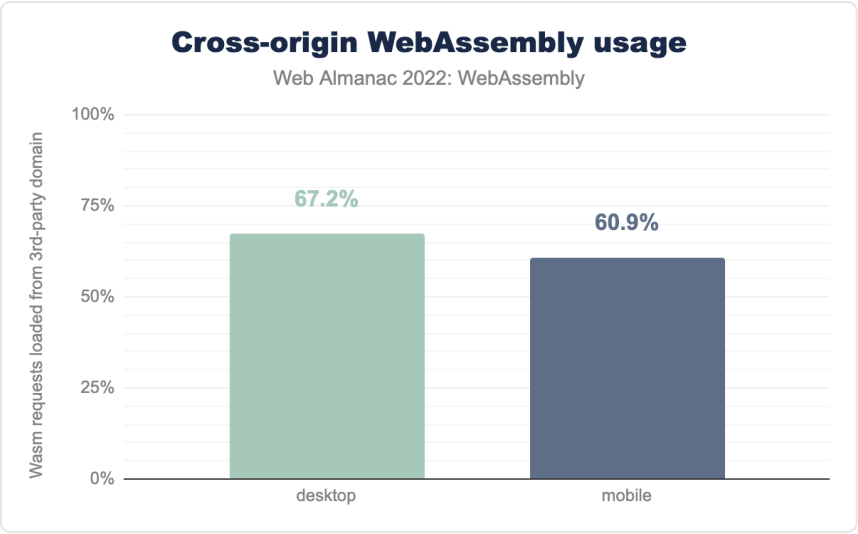



*図7.2. クロスオリジンのWebAssembly使用。*

これらのWebAssemblyモジュールは、サイズがかなり異なっており、小さいものでは数キロバイト、大きいものでは7.3メガバイトになります。さらに詳しく、解凍後のサイズを見ると、中央値（50パーセンタイル）は835KBytesであることがわかります。

もっとも小さなWebAssemblyモジュールは、たとえばブラウザの機能を満たすポリフィリングや、簡単な暗号化タスクなど、非常に特定の機能のために使用されていると思われます。より大きなモジュールは、WebAssemblyにコンパイルされたアプリケーション全体であると考えられます。

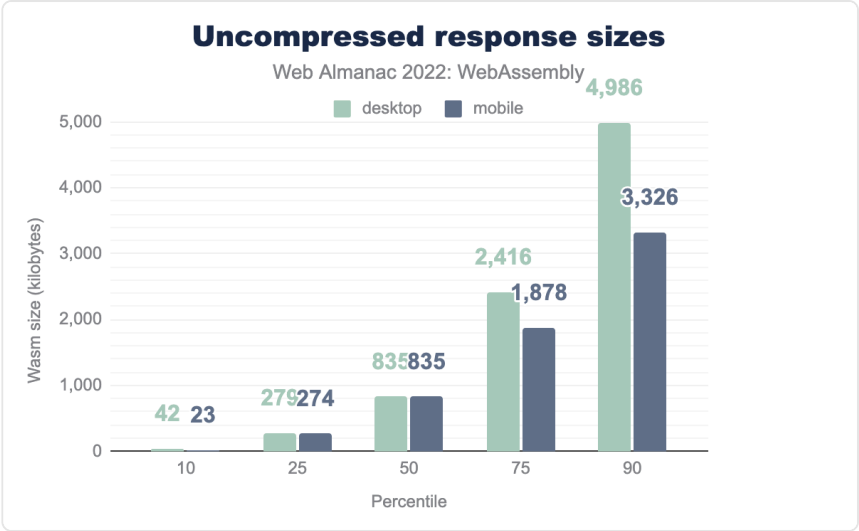



*図7.3. 非圧縮のレスポンスサイズ。*

WebAssemblyが広く使われていないのは明らかで、使用量が増えるどころか、緩やかに縮小しているのがわかります。

# WebAssemblyは何に使われているのですか？

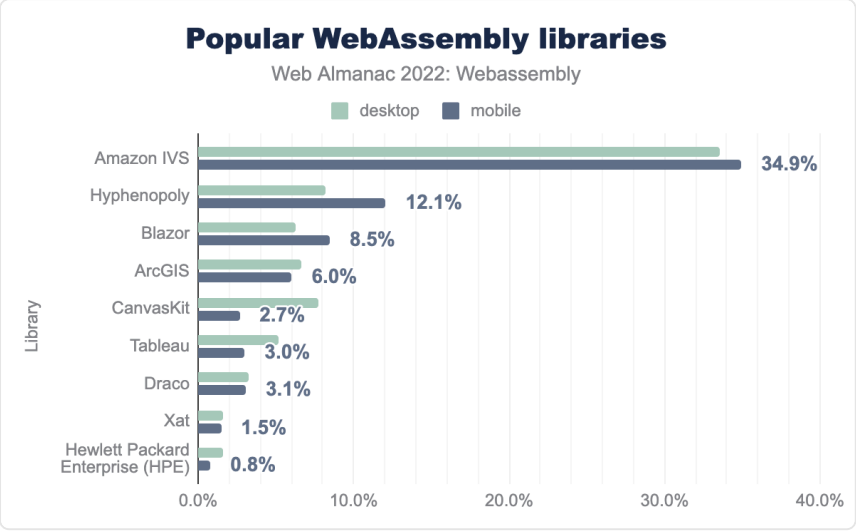



*図7.4. WebAssemblyの人気ライブラリ。*

- Amazon IVS（Amazonインタラクティブビデオサービス）[229] - ここでは、WebAssemblyがビデオコーデックとして使用されていると思われ、ユーザーのブラウザのコーデックサポートに依存しない一貫したビデオデコードを可能にしています。
- Hyphenopoly[230] - CSSハイフネーション用のポリフィルを提供するnpmモジュールです。コアとなるアルゴリズムはWebAssemblyモジュールとして出荷され、小さなフットプリントと一貫したパフォーマンスを提供します。
- Blazor[231] - Microsoft Blazorは、.NETプラットフォームとC#を使用したWebアプリケーションの開発を支援するプラットフォーム・ランタイム・UIライブラリです。
- ArcGIS[232] - インタラクティブなマッピングアプリケーションを構築するための包括的なツール群です。ArcGISチームにとってパフォーマンスは最大の関心事であり、これを実現するためにWebGLなどのさまざまな技術を採用しています。とくに、WebAssemblyは、高速なクライアントサイドプロジェクションを可能にするために使用されています。

229. https://aws.amazon.com/ivs/
230. https://mnater.github.io/Hyphenopoly/
231. https://dotnet.microsoft.com/en-us/apps/aspnet/web-apps/blazor
232. https://developers.arcgis.com/javascript/latest/

- CanvasKit[233] - このライブラリは、標準のCanvas2D APIよりも高度な機能を提供します。C++で書かれたグラフィックライブラリSkiaで実装されており、WebAssemblyにコンパイルされているため、ブラウザ上で実行できます。
- Tableau[234] - インタラクティブなビジュアライゼーションを構築するための一般的なツールです。WebAssemblyが同社の主力製品の一部として使用されているのか、それともクロールの一部として発見された特定のダッシュボードに使用されているだけなのかは明らかではありません。
- Draco[235] - 3次元幾何学メッシュや点群を圧縮・伸張するためのライブラリです。C++で書かれており、WebAssembyの構築により、ブラウザ内で使用できます。
- Xat[236] - 某ソーシャルメディアサイト。WebAssemblyを何に使っているのかは不明です。
- Hewlett Packard Enterprise[237] - 何のためにWebAssemblyを使っているのかが不明です。

人気のあるWebAssemblyライブラリを見ると、その用途はかなり限定的で特定の数値計算タスクに使用されたり、大規模で成熟したC++コードベースを活用して、JavaScriptに移植する必要なくその機能をWebに持ち込むことが多いことがわかります。

## 人々はどのような言語を使っているのでしょうか？

WebAssemblyはバイナリ形式であるため、プログラミング言語、アプリケーション構造、変数名など、ソースの情報の多くが難読化されるか、コンパイル過程で完全に失われます。

しかし、モジュールはしばしばエクスポートとインポートを持ち、それはホスト環境（ブラウザ内のJavaScriptランタイム）内の関数を名付け、モジュールのインターフェイスを記述します。ほとんどのWebAssemblyツールチェーンは、JavaScriptアプリケーションにモジュールを統合することを容易にする「バインド」の目的のために、少量のJavaScriptコードを作成します。これらのバインディングは、モジュールのエクスポートまたはインポートに存在する認識可能な関数名を持つことが多く、モジュールを作成するために使用された言語を識別するための信頼できるメカニズムを提供します。

WebAssembly特有の解析をクローラーに提供する wasm-stats[238] プロジェクトを強化し、エクスポート/インポートを検査して、特定のモジュールを作成するために使用される言語を示す共通のパターンを識別するコードを追加しました。例として、あるモジュールが

233. *https://skia.org/docs/user/modules/canvaskit/*
234. *https://www.tableau.com/*
235. *https://google.github.io/draco/*
236. *https://xat.com/*
237. *https://www.hpe.com/us/en/home.html*
238. *https://github.com/HTTPArchive/wasm-stats*

`wbindgen` という名前のモジュールをインポートした場合、これは wasm-bindgen[239] によって生成されたコードへの参照であり、そのモジュールがRustで書かれたことを明確に示す指標となります。

また、export/import名が縮小されており、ソース言語の特定が困難な場合もある。しかし、Emscripten（C++ツールチェイン）は、短縮名に対して特徴的な規約を持っており、このパターンを示すモジュールはEmscriptenを使用して生成されたと比較的確信できます。

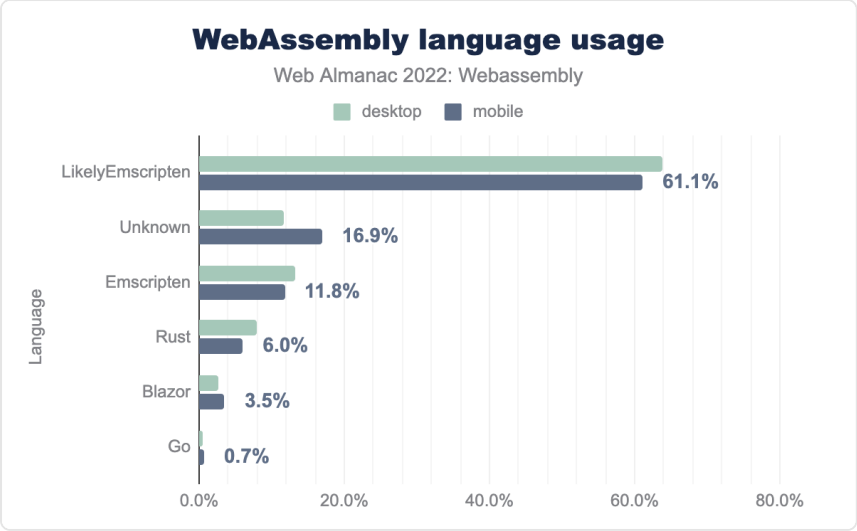



*図7.5. WebAssembly言語の使用法。*

その結果、デスクトップでは、72.8%のモジュールがEmscriptenで作成されている可能性が高く、その結果、C++で書かれている可能性が高いことが判明しました。次に多いのはRustで6.0%、Blazor（C#）で3.5%です。また、Goで書かれたモジュールも少数ながら見受けられました。

特筆すべきは、16.9%のモジュールが識別可能な言語を持っていなかったことです。AssemblyScript[240] は、人気のあるWebAssembly固有の言語ですが、生成されるモジュールには明らかな手がかりがありません。全モジュールの8.2%を占めるHypehnopolyがAssemblyScriptを使用していることが分かっており、これらの「未確認」モジュールのほぼ半分を占めています。

これらの結果を、WebAssemblyの現状2022年調査[241]で、Rustがもっとも頻繁に使用されていた言語と対比すると興味深いです。しかし、その調査では、かなりの数の回答者が、ブラウ

239. https://crates.io/crates/wasm-bindgen
240. https://www.assemblyscript.org/
241. https://blog.scottlogic.com/2022/06/20/state-of-wasm-2022.html

ザベースのアプリケーション以外にWebAssemblyを使用していました。

## どのような機能が使われているのでしょうか？

WebAssemblyの最初のリリースは、MVPとみなされました。他のWeb標準と同様に、World Wide Web Consortium（W3C）のガバナンスのもと、継続的に進化しています。今年は、WebAssembly v2ドラフト[242]の発表があり、多くの新機能が追加されました。

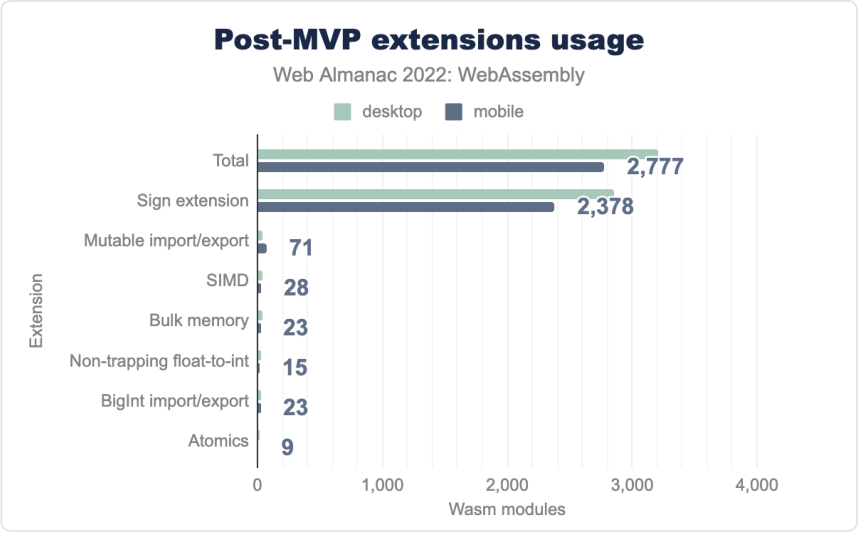



*図7.6. MVP後の拡張子の使用状況。*

使用されているPost-MVPの機能を調べたところ、*符号拡張（整数値をより深くビット拡張するための演算子を追加する比較的簡単な機能拡張）*が圧倒的に多く使用されていることがわかりました。これは、昨年の分析結果と大きな違いはありません[243]。

とくに、Web開発者がHTML/JavaScript/CSSのどの機能を使うかの選択に迫られるのに対し、WebAssemblyではツールチェーンの開発者が決定することが多いのです。その結果、あるツールチェーンがPost-MVP機能を実行可能なオプションと判断した場合、Post-MVP機能の採用が急増することが予想されます。

242. *https://www.w3.org/TR/wasm-core-2/*
243. *https://almanac.httparchive.org/ja/2021/webassembly#MVP後の拡張機能の使い方は？*

## 結論

WebAssemblyは、Webに関しては紛れもなくニッチな技術であり、今後もそうである可能性が非常に高いです。WebAssemblyは、C++、Rust、Go、AssemblyScript、C#など、さまざまな言語をWebに導入してきましたが、これらはまだJavaScriptと同じように使うことはできません。Webサイトの大部分は、コンテンツが比較的静的で（HTMLでCSSでレンダリングされ）、（JavaScriptで）適度なインタラクティブ性があるため、現時点ではWebAssemblyを使用する説得力のある理由はない。

将来的にこれを変える可能性のある重要な提案がいくつかあります。当初はWebIDLで、これはインターフェイスの種類に取って代わられましたが、再びコンポーネントモデル仕様に取って代わられることになりました。これらにより、将来的にはJavaScriptを他のプログラミング言語と簡単に交換できるようになるかもしれませんが、今のところはそうではありません。

ニッチな技術であるにもかかわらず、WebAssemblyはすでにWebに価値を与えています。この技術から大きな利益を得ているWebアプリケーションは数多くあります。しかし、Webアプリケーションは、この研究の基礎となる「クロール」からは見えないことが多い。

最後に、WebAssemblyランタイムのコア機能である多言語、軽量、セキュアは、より幅広い非ブラウザ アプリケーションのための一般的な選択肢となっています。WebAssemblyの現状2022年調査[244]では、サーバーレス、コンテナー化、プラグインアプリケーションにこの技術を使用する人の数が大幅に増加したことが示されています。WebAssemblyの将来は、ニッチなWeb技術としてではなく、他の幅広いプラットフォームで完全に主流となるランタイムとしてかもしれません。

### 著者

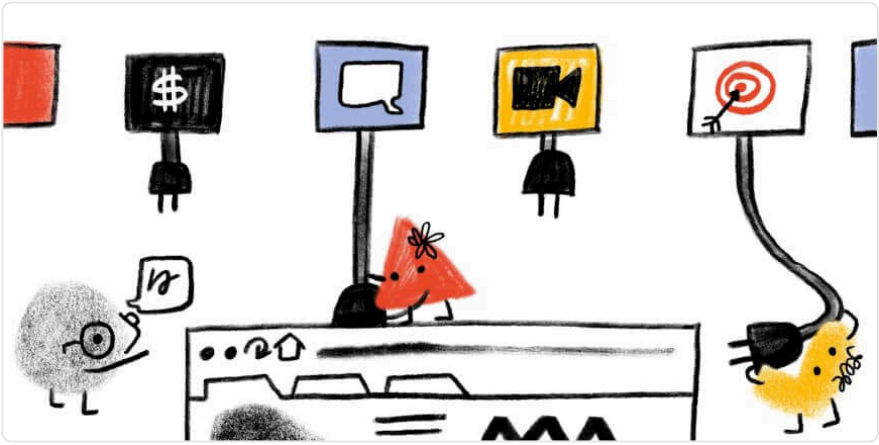
#### Colin Eberhardt

@ColinEberhardt　ColinEberhardt　https://blog.scottlogic.com/ceberhardt/

Scott Logic[245]のCTOであり、様々な技術に関する著者、ブロガー、講演者として活躍しています。また、FINOS[246]のボードメンバーであり、金融分野におけるオープンソースのコラボレーションを奨励しています。また、GitHubで非常に活発に活動しており、さまざまなプロジェクトに貢献しています。

244. *https://blog.scottlogic.com/2022/06/20/state-of-wasm-2022.html*
245. *https://www.scottlogic.com/*
246. *https://www.finos.org/*

# 部 I 章 8
# サードパーティ



***Eugenia Zigisova によって書かれた。***
*Barry Pollard、 Kevin Farrugia と Alex N. Jose によってレビュ。*
*Kevin Farrugia による分析。*
*Shaina Hantsis 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

サードパーティは、もっともWebサイトにとって不可欠な存在です。この章では、ほぼすべてのウェブサイトが少なくとも1つのサードパーティを使用していること、そしてリクエストの約半分がサードパーティのリクエストであることを示しています。

ウェブサイトの所有者は、分析、広告、ライブチャット、同意管理など、いくつかの複雑な機能を委任するためにサードパーティを使用しています。ウェブサイト開発者は、サードパーティのコードを直接制御することはできませんが、サードパーティがウェブサイトに与える影響に影響を与えることは可能です。サードパーティがどれほど広く使われているかを考慮すると、サードパーティはウェブパフォーマンスに決定的な影響を及ぼします。サードパーティがとくにモバイル端末でのページレンダリングをブロックすることはよくあることです。たとえば、もっとも普及しているサードパーティーの上位10社のブロック時間の中央値

の平均は1.4秒です。このため、サードパーティは、Core Web Vitals[247]や、視覚コンテンツの初期表示時間[248]などの重要なパフォーマンス指標に直接影響を与えることがあります。

多くの推奨事項が、マイナスの影響を排除するのに役立ちます。それはリソースを最小化するような単純なテクニックかもしれませんし、たとえばレガシーJavaScriptを提供しないサードパーティ製スクリプトを評価して選択したり、Web Workerを使ってサードパーティ製スクリプトをロードして実行したりするような、より複雑なものであるかもしれません。

この章では、ウェブサイトの所有者とサードパーティの開発者が、サードパーティのネガティブな影響をどのように軽減できるか、またベストプラクティスが守られているかどうかというトピックに焦点を当てます。まず、サードパーティの普及状況や、レンダーブロック時間やメインスレッドへの影響といったウェブパフォーマンスに関連する指標について説明します。後半は、リソースの最小化と圧縮、サードパーティーのファサード、 `async` と `defer` のスクリプト属性、レガシーJavaScript、およびその他の最適化の機会に関するベストプラクティスを分析します。

## 定義

サードパーティとは、主要なサイトとユーザーの関係の外にある組織です。サイト所有者の直接の管理下にない、しかしその承認を得たサイトの側面に関与します。サードパーティのリソースは

- 共有・公開オリジンにてホスティング
- さまざまなサイトで幅広く利用
- 個々のサイトオーナーに影響されない

サードパーティの例としては、Googleフォント、パブリックオリジンで提供されるjQueryライブラリ、YouTubeの埋め込み動画などがあります。

定義に合わせるため、HTTP Archiveデータセット内の少なくとも50のユニークなページでリソースを見つけることができるドメインから発信されたサードパーティーのみを対象としました。

サードパーティのコンテンツがファーストパーティのドメインから提供されている場合、ファーストパーティのコンテンツとしてカウントされます。たとえば、Googleフォントや bootstrap.cssのセルフホスティングは、 ファーストパーティのコンテンツとしてカウントされます。同様に、サードパーティのドメインから提供されるファーストパーティのコンテン

247. https://web.dev/i18n/ja/vitals/
248. https://web.dev/i18n/ja/fcp

ツは、サードパーティのコンテンツとしてカウントされます（「50ページ以上の基準」をクリアしていることが前提です）。

## サードパーティーのカテゴリー

私たちは、サードパーティを特定し分類するために、Patrick Hulce[249] の third-party-web[250] レポジトリに依存しています。このリポジトリでは、サードパーティを以下のカテゴリーに分類しています。

- **Ad** - これらのスクリプトは、広告ネットワークの一部であり、配信または測定を行っています。
- **Analytics** - ユーザーとその行動を測定または追跡するスクリプトです。何を追跡するかによって、ここでの影響は大きく異なります。
- **CDN** - These are a mixture of publicly hosted open source libraries (e.g. jQuery) served over different public CDNs and private CDN usage.
- **Content** - これらのスクリプトは、コンテンツプロバイダーや出版社特有のアフィリエイトトラッキングのものです。
- **Customer Success** - これらのスクリプトは、チャットやコンタクトのソリューションを提供するカスタマーサポート/マーケティングプロバイダーによるものです。これらのスクリプトは、一般的に重量が重くなっています。
- **Hosting*** - これらのスクリプトは、ウェブホスティングプラットフォーム（WordPress、Wix、Squarespaceなど）のものです。
- **Marketing** - これらのスクリプトは、ポップアップやニュースレターなどを追加するマーケティングツールのものです。
- **Social** - ソーシャル機能を実現するスクリプトです。
- **Tag Manager** - これらのスクリプトは、他の多くのスクリプトをロードし、多くのタスクを開始する傾向があります。
- **Utility** - これらのスクリプトは、開発者向けのユーティリティ（APIクライアント、サイト監視、不正検知など）です。
- **Video** - ビデオプレーヤーやストリーミング機能を実現するスクリプトです。
- **Consent provider** - これらのスクリプトにより、サイトはユーザーの同意を管理

249. *https://twitter.com/patrickhulce*
250. *https://github.com/patrickhulce/third-party-web/#third-parties-by-category*

できます（例：EU一般データ保護規則[251]準拠のため）。これらは「Cookie Consent」ポップアップとしても知られており、通常、クリティカルパスでロードされます。

- **Other** - これらは、正確なカテゴリーや属性がなく、共有のオリジンを介して配信される雑多なスクリプトです。

注：ここでいうCDNカテゴリには、公開CDNドメイン（bootstrapcdn.com、cdnjs.cloudflare.comなど）でリソースを提供するプロバイダーが含まれ、単にCDN経由で提供されるリソースは含まれません。たとえば、ページの前にCloudflareを配置しても、当社の基準によるファーストパーティの指定に影響を与えることはありません。

- 前年度と同様に、ホスティングカテゴリは分析対象から除外しています。たとえばブログにWordPress.com、eコマースプラットフォームにShopifyをたまたま使っている場合、そのサイトによるこれらのドメインに対する他のリクエストは、多くの点でこれらのプラットフォームのホスティングの一部であり、真の「サードパーティ」ではないとして無視することにしています。

## 注意点

- ここで紹介するすべてのデータは、非インタラクティブなコールドロードに基づくものです。これらの値は、ユーザーとの対話の後、まったく異なるものになる可能性があります。
- ページは、クッキーが設定されていない米国のサーバーからテストされているため、オプトイン後に要求された第三者は含まれません。これはとくに、EU一般データ保護規則[252]、またはその他の類似の法律の適用範囲にある国にホストされ、主に提供されるページに影響を与えるでしょう。
- トップページのみテストしています。その他のページでは、サードパーティーの要件が異なる場合があります。
- あまり使われていないサードパーティドメインの一部は、 *unknown* カテゴリーに分類されています。
- 私たちは、さまざまなLighthouse監査[253]を活用しています。その中には、カバー範囲が限られているものもあります。すなわち、サードパーティ ファサード監査[254]で、既存のファサード実装をすべてカバーすることは現実的でありません。

251. *https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87*
252. *https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87*
253. *https://github.com/GoogleChrome/lighthouse/tree/master/core/audits*
254. *https://github.com/GoogleChrome/lighthouse/blob/master/core/audits/third-party-facades.js*

当社の方法論について詳しく説明します。

## 普及率

**94%**

*図8.1. 少なくとも1つのサードパーティを使用しているモバイルページの割合*

サードパーティの普及率は、前年と同じ高い水準で推移しました： 94%のウェブサイトが少なくとも1つのサードパーティを使用しています。

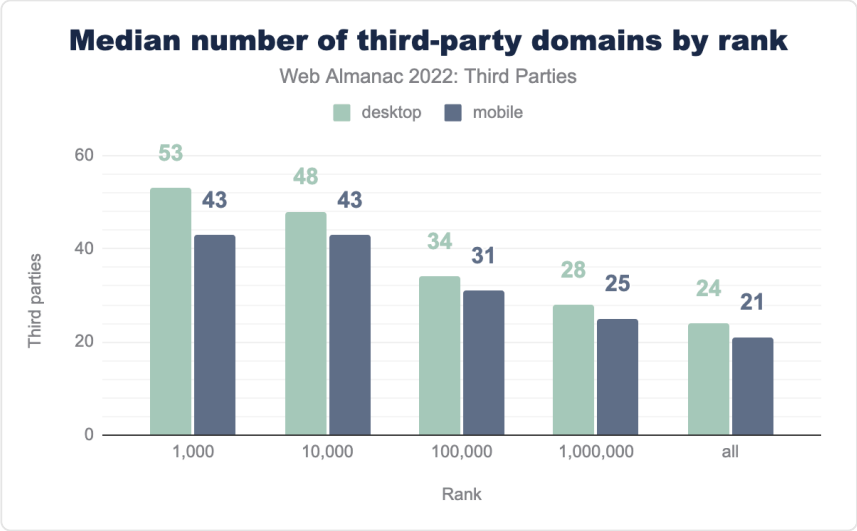



*図8.2. ランク別1ページあたりのサードパーティドメイン数の中央値。*

上の図は、もっとも利用されているウェブサイトのサードパーティドメインの数を示しています。たとえば、人気上位1,000サイトの平均では、モバイルデバイスで43、デスクトップデバイスで53のサードパーティドメインが使用されています。人気の高いウェブサイトほど、サードパーティの数が多いようです。つまり、上位1,000サイトでは、全ウェブサイトのサードパーティの数の中央値よりも2倍多くサードパーティが使われています。たとえば、Yahooは `mempf.yahoo.co.jp`、`yjtag.yahoo.co.jp` などのドメインからコンテンツを配信しています。 サードパーティプロバイダーの正確な数についてはまだ研究が必要ですが、サードパーティドメインに関する現在のデータは、それらがネットワークリクエスト

に費やす時間にどの程度影響するかを概観できます。新しいドメインへのリクエストには、DNSルックアップと初期接続の確立に時間がかかるため、サードパーティドメインの使用量が多ければ多いほど、ページの読み込み速度に影響を与える可能性があります。

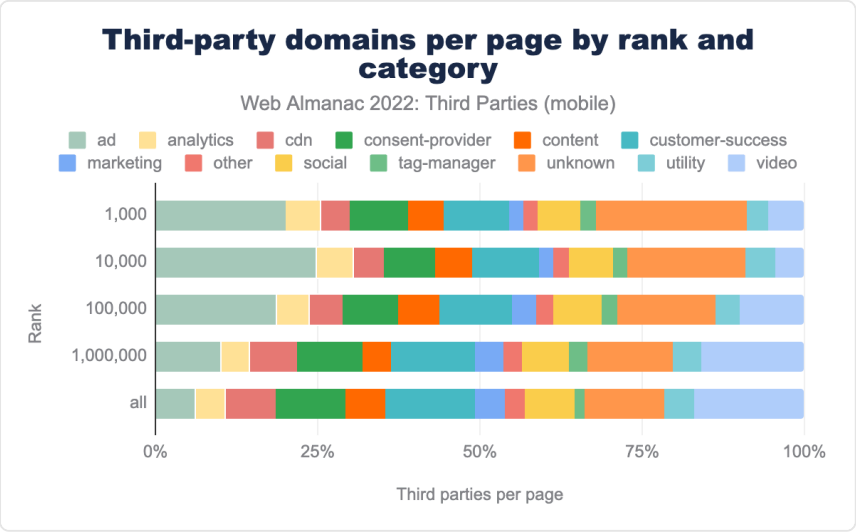



*図8.3. カテゴリ別、ランク別に見た1ページあたりのサードパーティドメイン数の中央値。*

サードパーティのカテゴリ別、ランク別の分布を見ると、もっとも普及しているウェブサイトでのサードパーティの増加は、広告と未知の（分類されていない）サードパーティのカテゴリで占められていることが明らかになりました。つまり、サードパーティ（とくに広告）は、ユーザー数の多いウェブサイトで多く利用されているため、ユーザーに決定的な影響を及ぼしていることがわかります。

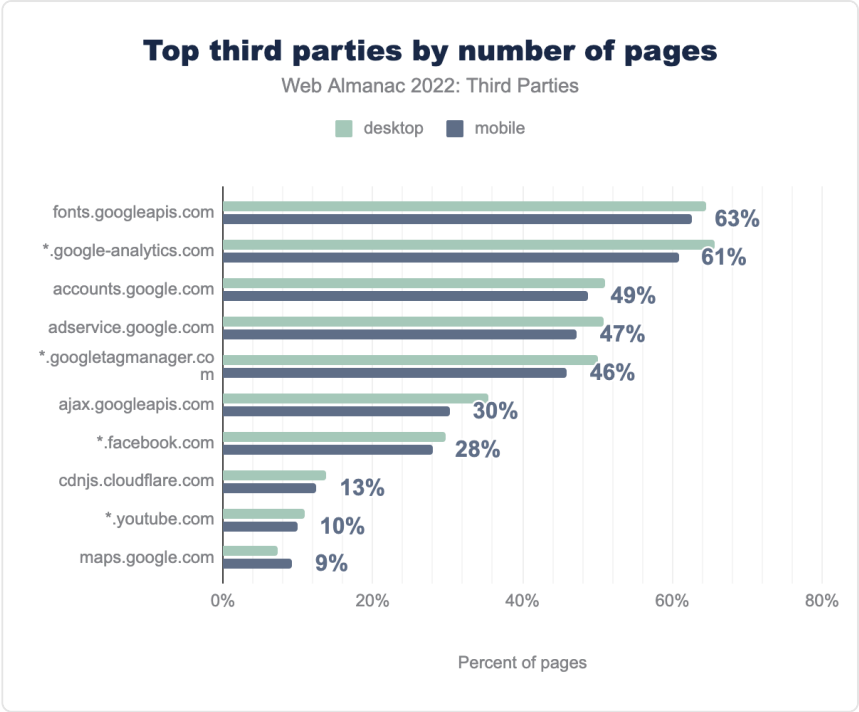



*図8.4. 使用されているページ数でもっとも多いサードパーティ上位10社*

Googleのサービス：フォント、分析、アカウント管理、広告、タグ管理は、ウェブ全体でもっとも普及しているサードパーティです。全ページの63%がGoogleフォントを使用しており、これはデータセットに含まれる790万件のモバイルページのうち、490万件以上に相当します！

**34%**

*図8.5. サードパーティからのリクエストのうち、スクリプトの占める割合*

サードパーティのリクエストは、Webサイトが行うリクエストの45%を占め、そのサードパーティのリクエストのうち、約3分の1がスクリプトです。このことから、ウェブパフォーマンスのベストプラクティスでスクリプトをより詳細に分析する価値があることがわかります。

## パフォーマンスへの影響

サードパーティーの中には、どうしてもページのレンダリングをブロックしてしまい、Webページの読み込み体験に悪影響を及ぼすものがあるかもしれレンダーブロッキングリソース監査[255]があり、レンダーブロッキング時間に関するデータを提供します。

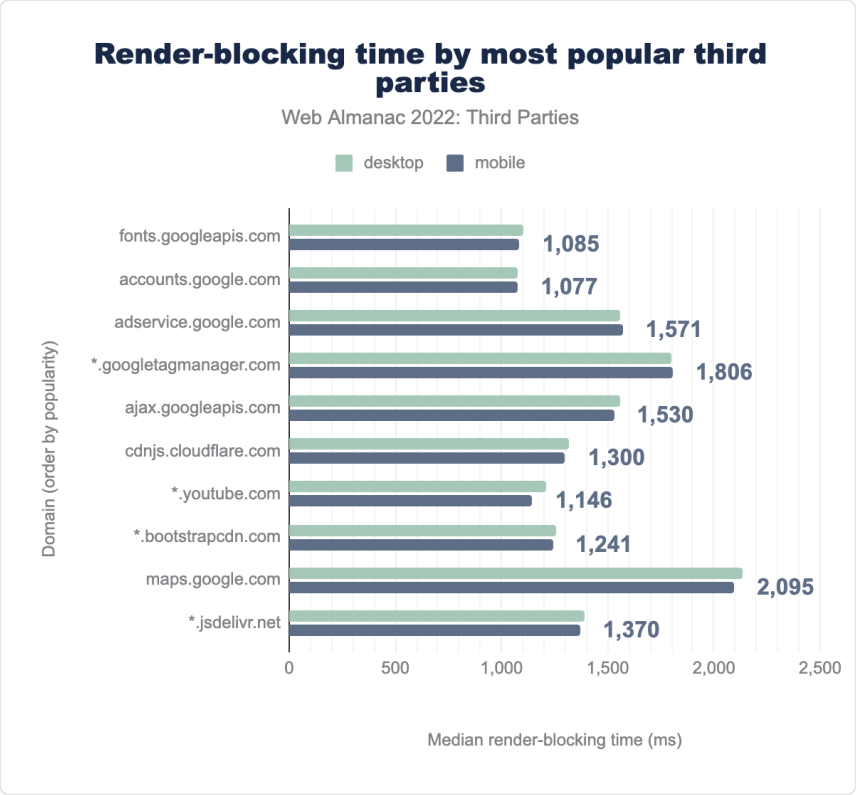



*図8.6. もっとも普及しているサードパーティーの上位10社のレンダリングブロック時間の中央値。*

上の図は、もっとも普及しているサードパーティーの上位10社のレンダリングブロック時間の中央値を示しています。レンダリング時間にもっとも大きな影響を与えるのは、Googleマップです。中央値のウェブサイトでは、2秒以上を占めています。これは、視覚コンテンツの初期表示時間[256]の推奨時間（ページ読み込み速度指標）が1.8秒であることを考慮すると、大きな影響と言えます。

255. https://github.com/GoogleChrome/lighthouse/blob/master/core/audits/byte-efficiency/render-blocking-resources.js
256. https://web.dev/i18n/ja/fcp/

Webサイトは、レンダー ブロックを引き起こすリソースを排除することで、読み込み時間を大幅に短縮できます。ブロッキングしない方法でサードパーティを埋め込む[257]方法はたくさんあります。たとえば、Googleマップの場合、Maps Static API[258]を使って、サードパーティのファサードを実装できます。また、サードパーティのiframeを遅延ロードさせることもできます。

さらに、サードパーティのスクリプトは、ウェブサイトのファーストパーティのコードとメインスレッドのリソースを奪い合います。サードパーティのJavaScriptタスクがメインスレッド上で50ms以上、長時間実行されている場合、「メインスレッドをブロックしている」とみなされます。メインスレッドは、ページのレンダリングだけでなく、ユーザーイベントの処理も担っているため、ページを操作する際のユーザー体験に大きな影響を与える可能性があります。メインスレッドがブロックされると、ユーザーはページの応答性が低下します。

私たちは、第三者要約監査[259]を検査して、第三者によって引き起こされるメイン スレッドのブロック時間をエミュレートしています。

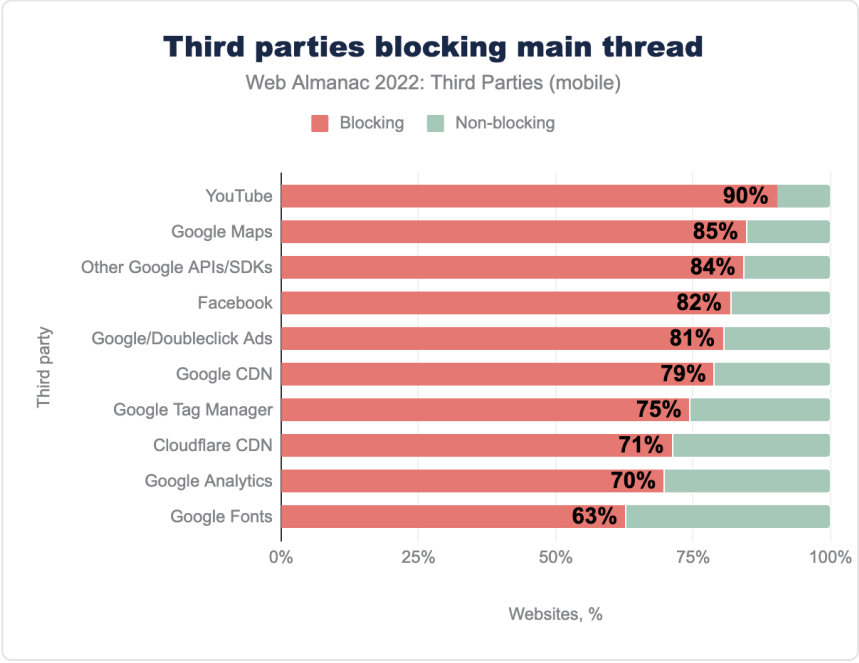



*図8.7. 本スレを阻むサードパーティーのもっとも多い10社*

257. https://web.dev/embed-best-practices/
258. https://developers.google.com/maps/documentation/maps-static/overview?hl=ja
259. https://github.com/GoogleChrome/lighthouse/blob/master/core/audits/third-party-summary.js

上図は、モバイル端末でもっとも多く利用されているサードパーティーの上位10位と、それらがメインスレッドに与える影響を示したものです。なお、デスクトップ端末では、その影響はかなり低くなっています。たとえば、YouTubeは、モバイルウェブサイトの90%でメインスレッドをブロックしますが、デスクトップウェブサイトでは26%しかブロックしていません。これは、デスクトップ端末の方がはるかに高性能であることを考慮すると、理にかなっています。しかし現在では、モバイルユーザー体験は非常に重要であり、モバイルウェブの章によると、ウェブサイト訪問者の58%がモバイルデバイスから訪れているそうです。

メインスレッドのブロッキングによってウェブサイトユーザーがどのような影響を受けるかをより詳細に調べるために、メインスレッドのブロッキング時間の中央値を確認します。

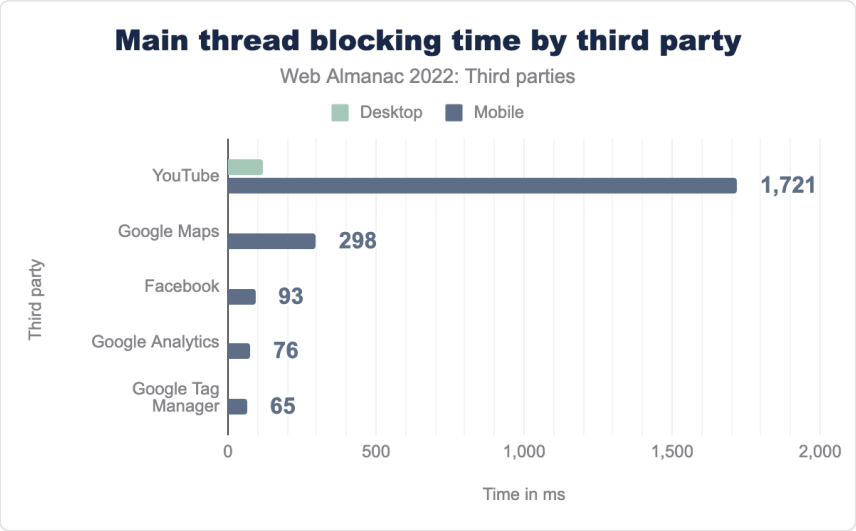



*図8.8. もっとも普及しているサードパーティーの上位5社のメインスレッドブロック時間の中央値（単位：ms）。*

YouTubeは、もっとも多く利用されているサードパーティーの上位5社の中で、もっともブロックが多いサードパーティーです。YouTubeの動画を読み込むウェブサイトの中央値で、1.7秒以上メインスレッドをブロックしています（クロールの一環としてエミュレートしたモバイルデバイスに基づく）。デスクトップのウェブサイトはほとんど影響を受けていないため、デスクトップとモバイルのウェブサイトには顕著な差があります。

Lighthouseは、サードパーティーのサイズが非常に小さく、レンダリングブロッキング時間に明確な影響を与えない場合でも、レンダリングブロッキングの可能性があるとしてマークする場合があります。これは、GoogleフォントやGoogle/Doubleclick Adsのように、レンダリングブロッキング時間の中央値が0ミリ秒であるサードパーティが該当します。

メインスレッドがブロックされると、最初の入力までの遅延(FID)[260] と 次のペイントへのインタラクション(INP)[261] の性能指標に大きな影響を及ぼします。Webページの応答性を高めるためには、FIDは100ms以下、INPは200ms以下であることが望ましいとされています。モバイル機器における総ブロッキング時間と次のペイントへのインタラクション[262]の相関関係については、Annie Sullivan[263]による研究があります。それによると、メインスレッドのブロック時間が小さいほど、サイトが良好なINPとFIDの閾値を満たす可能性が高いことがわかります。このことから、YouTubeの事例のように、サードパーティがメイン スレッドを長時間ブロックしている場合、良好なコアWebパフォーマンス指標を達成することが難しくなるという結論に達します。さらに、他のサードパーティやファーストパーティの資産も、レンダーブロッキング効果の一因となる可能性があります。しかし、サードパーティによるレンダリングブロックを最小限に抑える方法は数多くあります。これについては、次のセクションでさらに検討します。

## ウェブパフォーマンスのベストプラクティス

前節では、サードパーティが潜在的に、Webサイト体験全体に影響を及ぼすような大きなパフォーマンスインパクトを引き起こしていることを確認しました。しかしウェブサイトとサードパーティの開発者は、リソースの最小化からサードパーティのファサードの使用まで、サードパーティのパフォーマンスへの影響を最小限に抑えるために、もっとも高いパフォーマンスを実践できます。私たちは、ベストプラクティスがどのように守られているかを理解するために、さまざまなサードパーティの使用傾向を調べました。

### リソースの最小化

JavaScriptとCSSファイルの最小化は、ウェブパフォーマンスを語る上で、最初に推奨されるものの1つです。サードパーティのリソースがどのように最小化されているかを確認するために、私たちは以下のLighthouseオーディットを利用しています。未定義のJavaScript[264]と未定義のCSS[265]です。

スクリプトはもっとも普及しているサードパーティーのコンテンツタイプであるため、最小化することで大きな効果が得られるはずです。さらに、CSSのような他のコンテンツタイプに比べ、スクリプトはコメントや大きな変数名など、ファイルサイズに影響するような冗長なものになる傾向があります。

260. *https://web.dev/i18n/ja/fid/*
261. *https://web.dev/articles/inp/*
262. *https://github.com/GoogleChromeLabs/chrome-http-archive-analysis/blob/main/notebooks/HTTP_Archive_TBT_and_INP.ipynb*
263. *https://twitter.com/anniesullie*
264. *https://web.dev/unminified-javascript/*
265. *https://web.dev/unminified-css/*

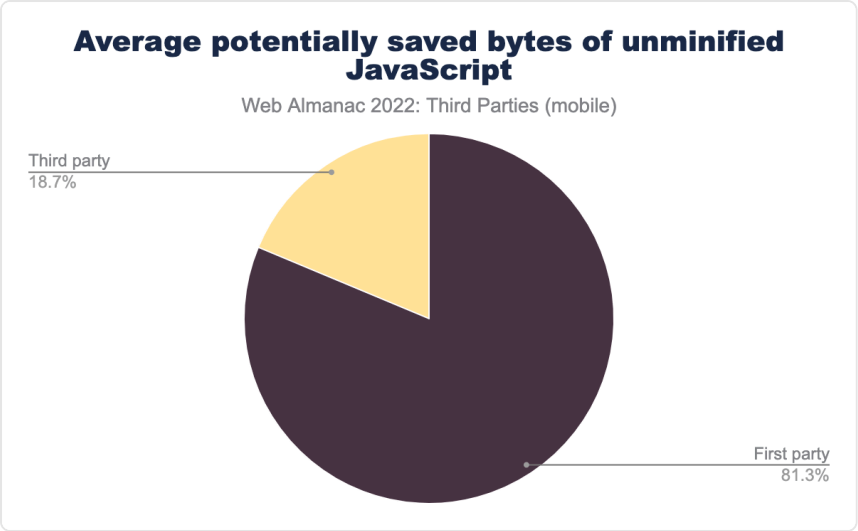



*図8.9. 最小化されていないJavaScriptの平均的な潜在的保存バイトのファーストパーティとサードパーティによる割合。*

サードパーティのリソースを最小化することで、一部のJavaScriptバイトを節約できますが、ファーストパーティのスクリプトはウェブサイト上の未削減JavaScriptのもっとも大きな量、すなわち潜在的に節約できる平均総バイト数の81%を占めています。最小化されていないJavaScriptの分布を見ると、75%のWebサイトで34.5KBの削減が可能であるのに対し、サードパーティによる削減は2.8KBにとどまっています。

次の図は、サードパーティによる潜在的な節約額の大きさを表示したものです。ここで重要なのは使用した手法で、外部ドメインから来たサードパーティのみを含み、ファーストパーティがホストするサードパーティのコンテンツ、たとえばノードモジュールとしてインポートされたライブラリはカウントしないことです。

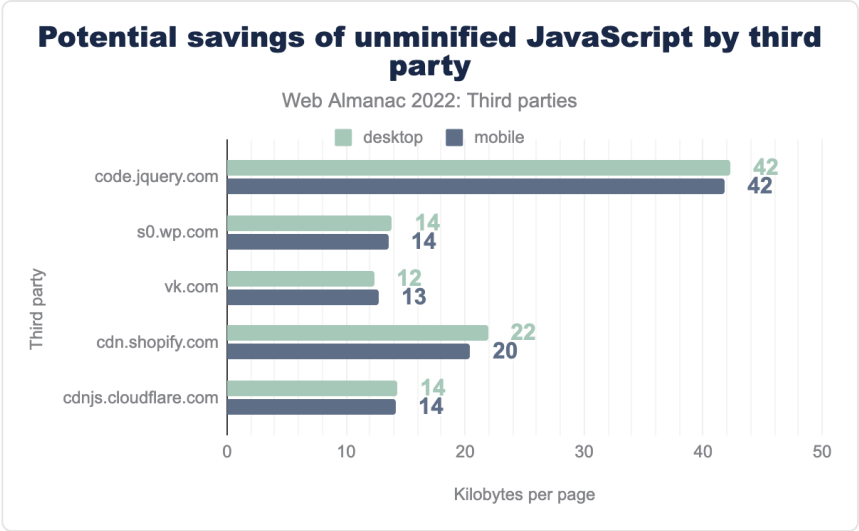



*図8.10. サードパーティ製スクリプトプロバイダー上位5社の、潜在的に保存されている未消化のJavaScriptの平均キロバイト数。*

jQueryのCDNライブラリである`code.jquery.com`は、デスクトップ上の全ウェブサイトの6%で使用されているもっとも普及しているJavaScriptサードパーティ製ライブラリです（なお、jQueryははるかに多くのウェブサイトで使用されていますが、すべての使用がこのCDNから提供されているとは限りません）。このCDNで利用可能な最小化されたバージョンのリソースを使用することで、最小化されていないjQueryを持つページあたり平均43KBのデータを節約できます。

**17%**

*図8.11. jQueryサードパーティを使用している全ページのうち、jQueryを削除していないデスクトップページの割合。*

サードパーティのドメインでホストされているjQueryを使用しているウェブサイトの17%が、縮小されていないJavaScriptのLighthouse監査[266]に失敗しました。ライブラリのインポート方法を詳しく調べると、多くのウェブサイトが、開発目的にのみ使用されるべきjQueryの未定義バージョンを使用していることがわかります。同様の傾向は、他のあまり人気のないサードパーティ製スクリプトの使用にも見られます。

このことは、Web開発者にとって、Webサイトに取り込まれたサードパーティ製スクリプト

266. https://web.dev/unminified-javascript/

が本番環境に最適化されているかどうかを確認するための注意喚起となるはずです。

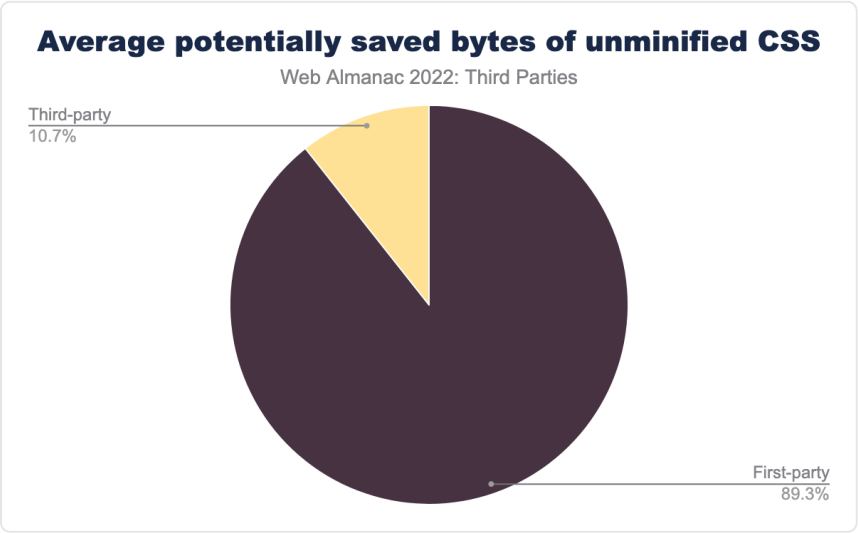



*図8.12. ファーストパーティとサードパーティによる、潜在的に保存されたCSSの平均バイト数の割合*

最小化されていないCSSの監査では、サードパーティの資産の影響がはるかに小さいことがわかります。とくに、ファーストパーティのCSSコードの潜在的な保存バイト数の平均が全体の89%であることと比較すると、サードパーティの資産の影響が大きいことがわかります。このデータは、サードパーティのCSSコンテンツは十分にミニファイされており、ファーストパーティのCSSよりもはるかに影響が小さいことを実証しています。

Googleフォントは、モバイル端末でもっとも普及しているサードパーティであり、全ウェブサイトの62.6%で使用されています。彼らが提供するCSSは、最小化されていません。データによると、Googleフォントを使用している平均的なページでは、最小化することで13.3KBを節約できるそうです。これは、改善のチャンスと思われます。しかしGoogle FontsのCSSには、非常に類似した `font-face` 定義が多数含まれており多くのフォントの場合、ほとんど同じように繰り返されるためHTTPレベルの圧縮がここで非常にうまく機能し、最小化しなくてもファイルサイズを大幅に削減できます。このため、最小化のメリットは、ローカルに複製する可能性のあるCSSを見たい人にとって、コードの読みやすさに比べて非常に低くなります。より複雑で大きなCSSを持つ他のサードパーティは、最小化によってより多くの利益を得られるかもしれません。

## リソースの圧縮

ファイルの圧縮は、サードパーティの開発者がウェブパフォーマンスに好影響を与えるため

にできる、もう1つの手っ取り早い方法です。スクリプトやCSSのような重いコンテンツタイプのほとんどは、圧縮率が高いです。サードパーティからのリクエストのうち、圧縮されていないのはスクリプトの12%とCSSファイルの4.5%にすぎません。

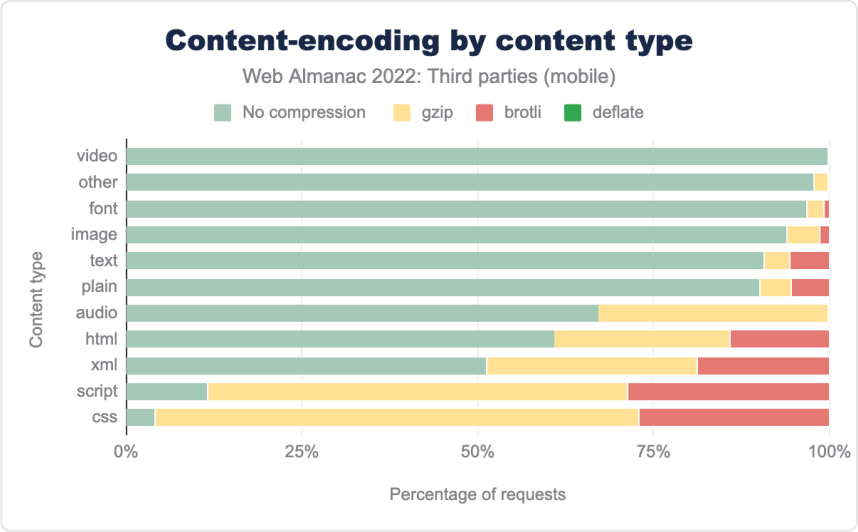



*図8.13. モバイルページにおけるサードパーティーのコンテンツタイプ別のContent-encoding。*

上図に表示されたウェブサイトのコンテンツエンコードデータから、画像圧縮に関する興味深い事実が判明しました。JPEG、PNG、WebP、AVIFなどの画像フォーマットは、そのフォーマットの一部として圧縮機能を備えているにもかかわらず、画像コンテンツの5.2%がGzipまたはBrotli圧縮を使用して再度圧縮されています。標準的な画像圧縮フォーマットの上にさらに圧縮のレイヤーを追加することは通常不要で、ファイルサイズの増加につながり、画像を解凍する際にCPUに余分な負荷を与える可能性があります。

Gzipは依然としてもっとも普及している圧縮タイプで、つまり59%のスクリプトと68%のCSSがGzipで圧縮されています。しかし、Brotli圧縮はより効果的です。ファーストパーティとサードパーティの動向を見ると、この3年間でBrotli圧縮の使用率が上昇し、圧縮なしとGzipの使用率が低下していることがわかります。

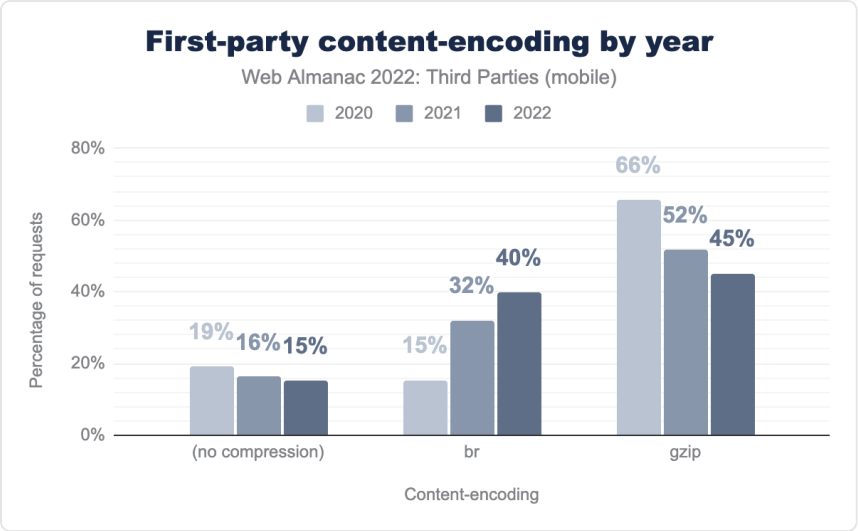



*図8.14. モバイルサイトにおける、コンテンツエンコードタイプ別および年別のファーストパーティースクリプトリクエストの割合。*

brotli経由で圧縮されたファーストパーティーのスクリプトの割合は、15%から40%へと約3倍に増加しました。

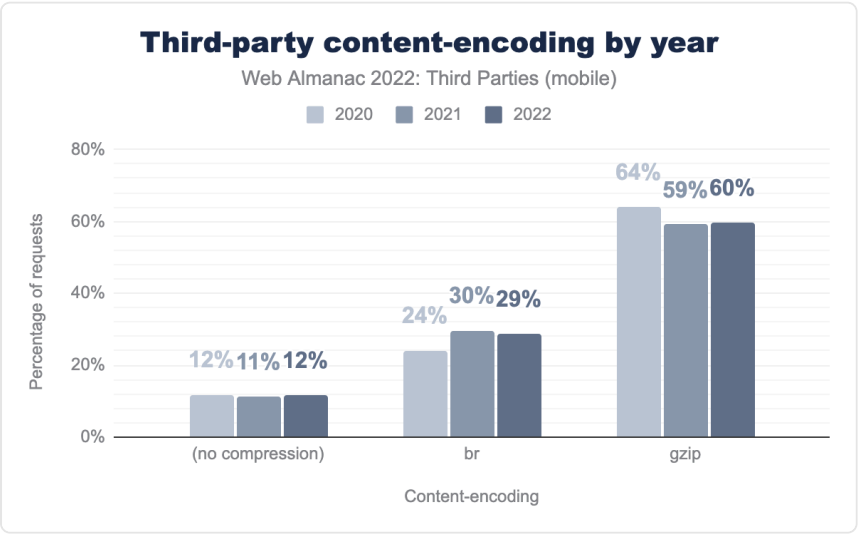



*図8.15. コンテンツエンコードタイプ別、年別のサードパーティ製スクリプトリクエストの割合*

しかし、サードパーティーのBrotli採用率は24%から29%に変化し、ほぼ同じ水準にとどまっています。わずかなプラス傾向とはいえ、サードパーティーのBrotli採用率にはまだ改善の余地があります。

### サードパーティ製ファサードの使用について

レンダーブロッキング リソースを排除するための複数のテクニックがあります。そのうちの1つが サードパーティーファサード[267] で、YouTube動画などの視覚コンテンツやライブ チャットなどの対話型ウィジェットに役立ちます。このファサードは、サードパーティを重要なロード シーケンスから除外し、遅延ロードするのに役立ちます。Lighthouseは、監査 サードパーティーのリソースをファサードで遅延ロード[268] を導入しました。たとえばlite-youtube-embed[269]、lite-youtube[270]、またはいくつかのカスタムアプローチなど複数の第三者ファケードソリューションがありますが、監査中にチェックした 第三者のリスト[271]にはその一部しか含まれていませんでした。この制限により、現時点では、ウェブ全体におけるサード パーティ製ファサードの使用状況を評価することが複雑になっています。

### `async` と `defer` の使用法

`async` と `defer` の使用は、レンダーブロックを引き起こすサードパーティ製スクリプトの読み込みを最適化するために、ウェブサイト開発者が使用できるもう1つのテクニックです。

267. *https://developer.chrome.com/ja/docs/lighthouse/performance/third-party-facades/*
268. *https://github.com/GoogleChrome/lighthouse/blob/master/core/audits/third-party-facades.js*
269. *https://github.com/paulirish/lite-youtube-embed*
270. *https://github.com/justinribeiro/lite-youtube*
271. *https://github.com/patrickhulce/third-party-web/blob/master/data/entities.js*

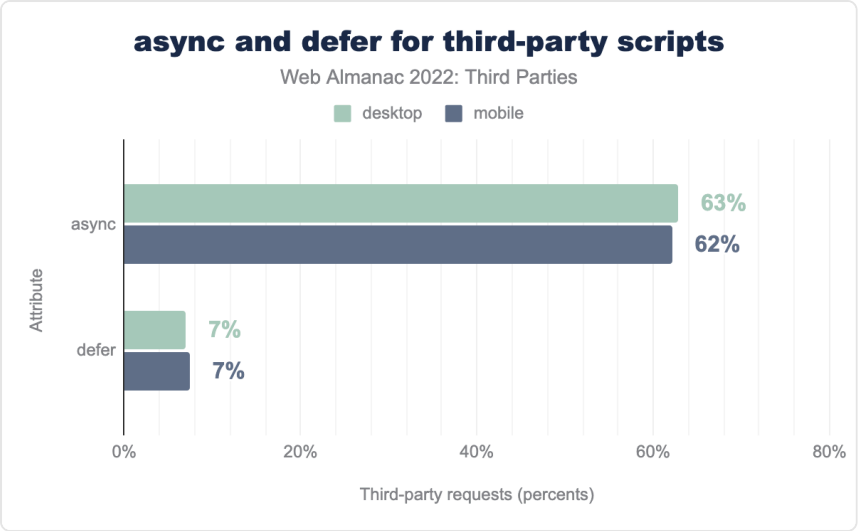



*図8.16. サードパーティーの全スクリプトリクエストのうち、`async`と`defer`の属性による割合です。*

`async`属性は、`defer`よりもかなり人気があります。モバイルデバイスのサードパーティ製スクリプト全体の62%に使用されています。`async`属性を使用しても、HTMLの解析中にスクリプトの実行が開始されるため、レンダーブロッキングが発生することがあります。`async`属性は、ページの読み込み中に重要なリソースがあり、レンダリングが中断される可能性がある場合に有効です。

`async`がより多く使われているということは、サードパーティーのスクリプトがもっとも重要なリソースとして扱われていることを示しています。これは一部のスクリプトには当てはまりますが、多くのサードパーティ、たとえばビデオプレーヤーはそれほど重要ではありません。遅延スクリプトは、潜在的にページのレンダリング時間に良い影響を与え、それは最大のコンテンツフルペイント[272]のようなコアウェブパフォーマンスメトリクスに反映されています。ウェブサイト開発者は、重要なレンダリングパスにとって重要でないサードパーティの資産に`defer`を使用することを検討すべきです。

どのリソースが重要で、どのリソースが延期できるかは、とくに他のサードパーティを使用可能にする *コンセントマネジメント* サードパーティを考慮する場合、厄介な問題かもしれません。たとえば、分析スクリプトは通常サイトオーナーにとって重要なものとされていますが、GDPR[273]や同様の法律がある国ではユーザーの同意なしに使用することはできず、ユーザーの同意がサードパーティにとって重要となっています。クリティカルパスでサードパーティーのリソースをロードすると、累積レイアウトシフトやファーストインプットディレ

272. https://web.dev/i18n/ja/lcp/
273. https://ja.wikipedia.org/wiki/EU%E4%B8%80%E8%88%AC%E3%83%87%E3%83%BC%E3%82%BF%E4%BF%9D%E8%AD%B7%E8%A6%8F%E5%89%87

イの原因となり、ユーザー体験が悪くなる場合があります。したがって、開発者は、サードパーティーのロード方法とユーザー体験のバランスを取るように努力する必要があります。

## レガシーJavaScript

JavaScriptの急速な普及にもかかわらず、レガシーコードの普及はまだ顕著です。私たちはLighthouseの監査の1つを利用して、レガシー JavaScriptをモダンブラウザに提供しているサードパーティがどれくらいあるかをチェックしています[274]。

**59%**

*図8.17. レガシーJavaScript Lighthouseの監査失敗のうち、サードパーティに起因するものの割合*

一般的に、Lighthouseのレガシー JavaScript監査失敗の59%はサードパーティが占めています。監査結果を詳しく見ると、レガシーJavaScriptを含むサードパーティーのスクリプトプロバイダーがもっとも多い5社であることがわかります。

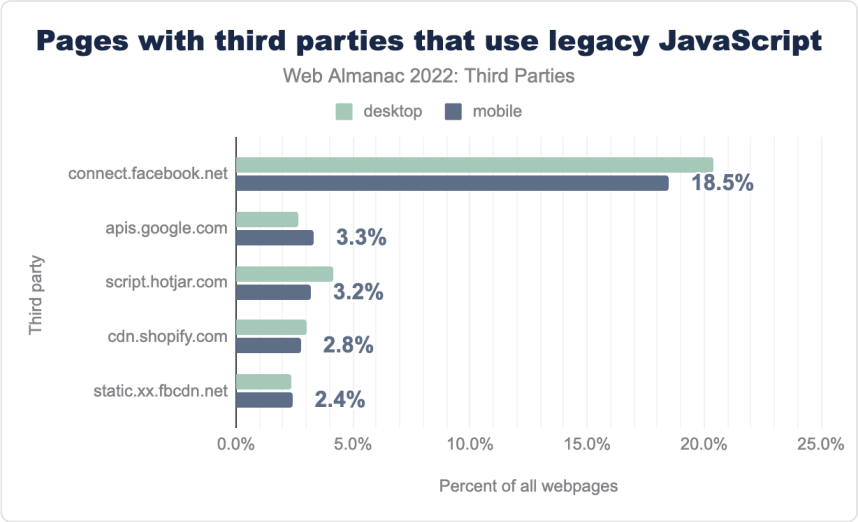



*図8.18. レガシーなJavaScriptを搭載したサードパーティを利用しているウェブサイトの割合*

274. https://github.com/GoogleChrome/lighthouse/blob/master/core/audits/byte-efficiency/legacy-javascript.js

Facebookは、もっとも多くのページに影響を与えるレガシーJavaScriptを持つサードパーティです。上のグラフから`facebook.net`と`fbcdn.net`の両方を見た場合、モバイルデバイスとデスクトップデバイスのウェブページ総数のうち、それぞれ約20%にレガシーコードが導入されていることがわかります。Internet Explorerのような古いブラウザに対応する必要がなくなった現在では、レガシーJavaScriptを残す必要性は低くなっています。しかし、過去3年間のレガシーJavaScriptを搭載したFacebookリソースの利用動向を見ると、`facebook.net`だけで2020年の約14%から2022年には18%と、むしろ増えていることがわかります。これは、このサードパーティを埋め込むWebサイトが増加しているためです。

レガシーなJavaScriptをモダンブラウザに提供すると、冗長で低速なコードが大量に発生します。未使用のJavaScriptのサイズを分析することで、これをより詳しく調べることができます。

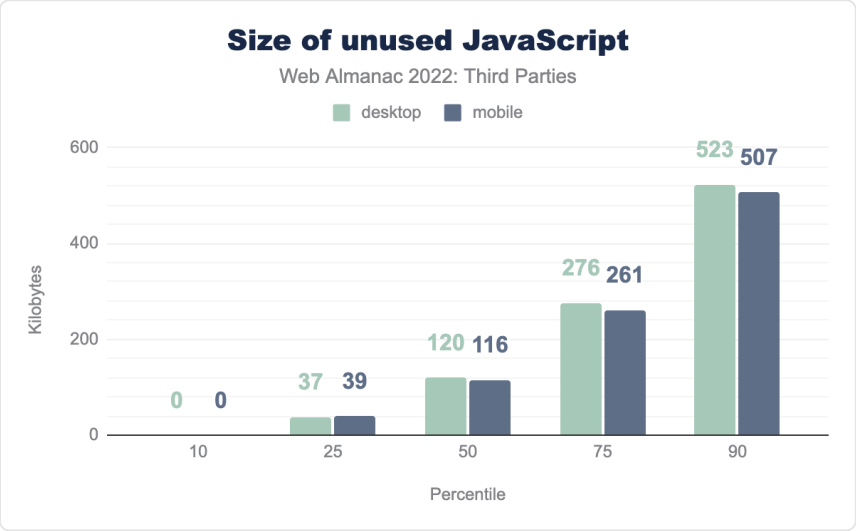



*図8.19. 未使用のサードパーティ製JavaScriptのサイズ。*

未使用のサードパーティ製JavaScriptの中央値は、約120KBです。サードパーティ製スクリプトを使用しているウェブサイトの25%では、261 KB以上となっています。

残念ながら、Webサイトの所有者には、サードパーティJavaScriptのバンドル方法を変更する可能性があるとは限りません。しかし、サードパーティの依存関係がセルフホストされている場合、最新のスクリプト バンドル手法[275]を採用することで開発中に最適化することができ、未使用コードの量を減らすのに役立つ可能性があります。

275. *https://web.dev/publish-modern-javascript*

## その他の最適化技術

サードパーティリソース管理の問題の1つは、時に開発チームを飛び越えて、適切なウェブパフォーマンス評価を行わずにタグ管理ツールを使って追加されることです。その結果、サードパーティーのスクリプトが、ページの読み込みや応答性の体験に制御不能な影響を与えることがあります。

近年、いくつかのモダンなサードパーティのロードと実行のソリューションが登場しています。たとえば、Partytown[276] は、ファーストパーティのコードのためにメインスレッドを解放するために、サードパーティのスクリプトをWebワーカーに再配置するライブラリです。現在、ライブラリの利用状況は初期導入段階であり、非常に低い状況です。2022年にデータセット全体から70のウェブサイトしか使用していません。しかし、Next.jsフレームワークがこのソリューション[277]の導入を開始したことで、Partytownの人気が高まるかもしれません。

前のセクションでは、サードパーティの悪影響に対する責任がファーストパーティとサードパーティの開発者の間で分担されていることを示しました。しかし、ブラウザもサードパーティ製リソースの読み込みを最適化することに関心を示しています[278]。提案には、より良いリアルユーザーのモニタリングと、サードパーティがウェブサイトに与える影響についてより多くのデータを提供する開発者向けツールが含まれています。

**25%**

*図8.20. Timing-Allow-Originヘッダーを持つサードパーティリクエストの割合*

サードパーティのウェブパフォーマンスデータの透明性を高めるために重要なTiming-Allow-Origin (TAO)ヘッダー[279]を提供するサードパーティのリクエストは全体の25%に過ぎないことから、この達成は困難かもしれません。

TAOヘッダーの普及率が以前と比較して改善されていない[280]ことを考慮し、ファーストパーティがこれらのリソースのパフォーマンスについてより正確に把握できるように、第三者プロバイダーがより積極的に利用することを推奨します。

276. *https://partytown.builder.io/*
277. *https://nextjs.org/docs/basic-features/script#off-loading-scripts-to-a-web-worker-experimental*
278. *https://developer.chrome.com/blog/third-party-scripts#proposed-approach*
279. *https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Timing-Allow-Origin*
280. *https://almanac.httparchive.org/ja/2021/third-parties#Timing-Allow-Originヘッダーの普及率*

## 結論

サードパーティは、しばしばウェブサイトのパフォーマンスに悪影響を及ぼすとされています。実際、レンダリングやメインスレッドのブロックにかかる時間は、とくにモバイルデバイスで顕著になります（モバイルデバイスはますます一般的になっています）。しかし、この章の主な目的は、サードパーティがウェブパフォーマンスに与える影響の責任は、サードパーティのプロバイダーとウェブサイトの所有者の間で共有されることを示すことです。ウェブサイト開発者には、サードパーティーの影響を軽減する機会がたくさんあります。将来的には、ブラウザがサードパーティのリソース最適化を自動的に適用するようになるかもしれません。

圧縮・最小化されたリソース、レガシーAPI、未使用のJavaScriptなど、さまざまなウェブパフォーマンスの推奨事項に関連するデータを分析しました。その結果をもとに、ウェブサイトやサードパーティの開発者がユーザー体験を向上させるために役立つ、以下のようなアクションポイントを実施しました。

- ミニ化・圧縮された本番環境に適したサードパーティ製リソースをロードします。
- とくに動画、地図、ライブチャットなどの「重い」コンテンツでは、レンダリングをブロックし、視覚コンテンツの初期表示時間に決定的な影響を与える可能性があるため、さまざまなサードパーティーのファサード技術を活用します。
- サードパーティーの候補を評価しながら、必要な場合を除き、レガシーなAPIを提供していないことを確認します。
- サードパーティーのコンテンツがページにとってどれだけ重要かを考慮し、レンダーブロッキングが発生しない場合は `defer` 属性を使って重要でないリソースをロードします。
- 最新のサードパーティーのロードと実行のストラテジーを探求します。
- gzipよりもBrotli圧縮を選択してください。

最適化の機会はまだまだたくさんあります！サードパーティが提供する機能がユーザー体験を損なうことなくウェブサイトを提供できるよう、ウェブデベロッパーにはそのような取り組みを推奨しています。

## 著者

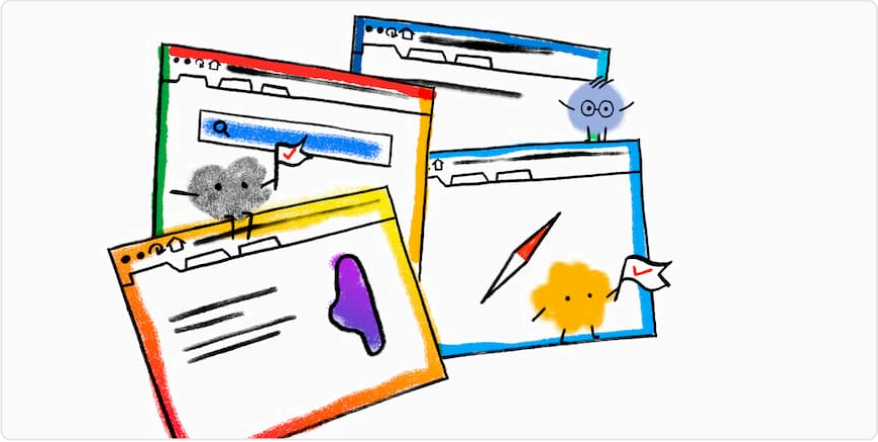

### Eugenia Zigisova

@jevgeniazi imeugenia https://github.com/imeugenia/speaking/blob/main/README.md


Eugeniaは、Webパフォーマンスとステートマシンに情熱を注ぐフロントエンドエンジニアであり、技術イベントのスピーカーです。N26やGorillasといったベルリンの急成長スタートアップで働いた経験があり、現在はRapidAPI[281]に入社しています。彼女はラトビアで数年間、Google Developer Groupを運営していました。

281. *https://rapidapi.com/*

# 部 I 章 9
# 相互運用性



***Brian Kardell* によって書かれた。**
*Eric A. Meyer* と *Philip Jägenstedt* によってレビュ。
*Rick Viscomi* と *Kevin Farrugia* による分析。
*Barry Pollard* 編集。
*Sakae Kotaro* によって翻訳された。

## 序章

2019年、Mozilla Developer Network（MDN）の製品諮問委員会は、173カ国の28,000人以上の開発者とデザイナーを対象とした重要な調査を実施しました。そこから得られた結果は、最初の ウェブ開発者ニーズ調査[282]（Web DNA）として発表されました。この調査では、重要な不満や苦痛のポイントのいくつかは、もっとも頻繁にブラウザ間の違いに関係していることが特定されました。2020年、これは MDNブラウザ互換性レポート[283] として知られるフォローアップにつながりました。

歴史的に、実装者の優先順位と焦点は独自に管理されてきました。しかし、この新しいデータを受けて、ブラウザ メーカーは *Compat 2021* と呼ばれる初の取り組みに協力しました。Compat 2021の開始当初、すべてのエンジンは、安定した出荷状態のブラウザにおいて、5つの分野で65〜70%の互換性しか得られませんでした。現在では、すべてのエンジンが90%

282. *https://insights.developer.mozilla.org/reports/pdf/MDN-Web-DNA-Report-2019.pdf*
283. *https://insights.developer.mozilla.org/reports/mdn-browser-compatibility-report-2020.html*

を超えています。2022年には、この取り組みが拡大され、 *Interop 2022* に改名されました。

どちらの取り組みも、本章で取り上げるべきいくつかの異なるものを提供しています。Compat 2021からもっとも改善されたものが出荷されてからほぼ1年が経ち、Interop 2022の多くのものはすでに出荷されているブラウザに導入されていますが、年末までにさらに多くのものが導入される予定です。

このような取り組みにおける興味深い疑問は、”どうすればうまくいった（あるいはいかなかった）とわかるのか“ということです。スコアの大幅な改善は有用ですが、開発者の採用がなければ不十分です。そこでWeb Almanacでは今年はじめて、このような疑問と格闘し何が変わり、何がもう一度見る価値があるのかについて、開発者に中心的な情報を提供するために新しい相互運用性の章を設けることにしました。

この章では、Compat 2021で行われた作業を要約し、私たちができることを測定します。また、Interop 2022で何が起きているかを調べ、私たちが長期的に追跡できる価値あるメトリクスがあるかどうかを検討します。これらの取り組みには非互換性やフラストレーションの程度の差こそあれ、安定したすでに有用な機能から、私たちが最初からセットアップしようとした真新しいものまで、さまざまなケースが混在しています。

## Compat 2021

Compat 2021は、5つの主要分野に重点を置いています。

- グリッド
- フレックスボックス
- スティッキーポジション
- トランスフォーム
- アスペクト比

2021年1月の時点では、すべての安定版／出荷版ブラウザがこれらの分野で65〜70%の互換性を獲得しており、各ブラウザで失敗しているテストは必ずしも同じ30〜35%ではありませんでした。

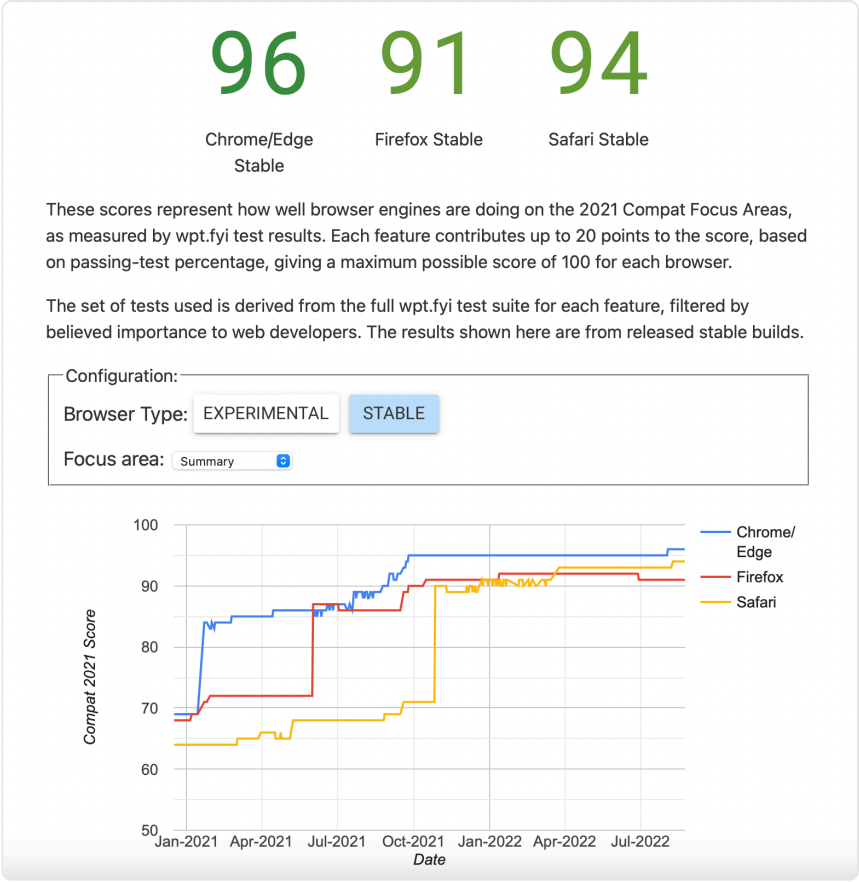



*図9.1. コンパチ2021ダッシュボード。*
*(ソースはこちら： ウェブ・プラットフォーム・テスト[284])*

今日、かなりのレベルで改善されていることがわかります。ChromeとEdgeは96%、Firefoxは91%、Safariは94%です。

## グリッド

CSSグリッドは、ここ数年もっとも普及している機能のひとつです。HTTP Archiveのデータを見ると、グリッドが登場して以来、年々倍増しています。グリッドはすでにかなり高い相互運用性を持っていましたが、サポートにはまだ細かな違いがいくつもありました。2021年と2022年を通して、グリッドの機能をテストするWeb Platform Testsの900以上のテスト

284. https://wpt.fyi/compat2021

の整合性を改善する作業が行われました。もしあなたが過去にグリッドで何かをしようとして頭痛の種になったことがあるなら、もう一度試してみてください。

そのよい例が、グリッドトラック、つまりグリッドの行と列をアニメーション化する機能です。しかし、この章の執筆中に、WebKit[285] と Chromium[286] の両方にグリッドトラックのアニメーションが追加されました。つまり、あなたがこれを読む頃には、3つの主要なエンジンすべてにグリッドトラックのアニメーションが追加されているはずです。

### フレックスボックス

Flexboxはさらに古く、より広く使われています。今年、その使用は再び拡大し、現在、モバイルページの75%、デスクトップページの76%に表示されています。Gridと同じようなテスト数で、非常に広く採用されているにもかかわらず、当初はもっとひどい状態でした。2021年に入ると、ボロボロのバグと未実装のままのサブ機能の組み合わせがありました。たとえば位置揃えキーワード値[287]（justify-contentとalign-contentに適用でき、justify-selfとalign-selfにも適用できる）は、サポートがボロボロで、いくつかの相互運用性の問題がありました。絶対位置のフレックスアイテムでは、これはさらに悪化していました。これらの問題は解決されました。




*図9.2. スタイルシートで `flex-basis: content` を使用しているデスクトップページ。*

これはフレックスアイテムのコンテンツに基づいて自動的にサイズを調整するために使用されます。 これは当初Firefoxに実装されましたが、2021年にはWebKitとChromiumへの実装が進められていました。今日、これらのテストはすべてのブラウザで一律にパスし、`flex-basis: content` はデスクトップで112,323ページ、モバイルで75,565ページ、ページの約1%に表示されています。ユニバーサルサポート1年目の機能としては悪くないスタートで、昨年の約2倍です。今後もこの指標から目が離せません。

285. https://webkit.org/blog/13152/webkit-features-in-safari-16-0/
286. https://groups.google.com/a/chromium.org/g/blink-dev/c/LI7br0giMk8/m/I4WNHdatBQAJ
287. https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Box_Alignment#positional_alignment_keyword_values

### Sticky positioning

**5.5%**

*図9.3. スタイルシートで `position: sticky` を使用しているデスクトップページ。*

スティッキーポジショニングは以前から存在していました。実際、フィーチャークエリの50%以上を占め、最も普及しているフィーチャークエリ[288]であることは注目に値します。たとえば、Chromeではヘッダーをテーブルに貼り付けることができません。`position:sticky` は、2022年にはデスクトップページの約5%、モバイルページの約4%で積極的に使用されています。これらの相互運用性の問題への対処が、時間の経過とともに採用にどのような影響を与えるか、今後しばらくこの指標から目が離せません。

### CSSトランスフォーム

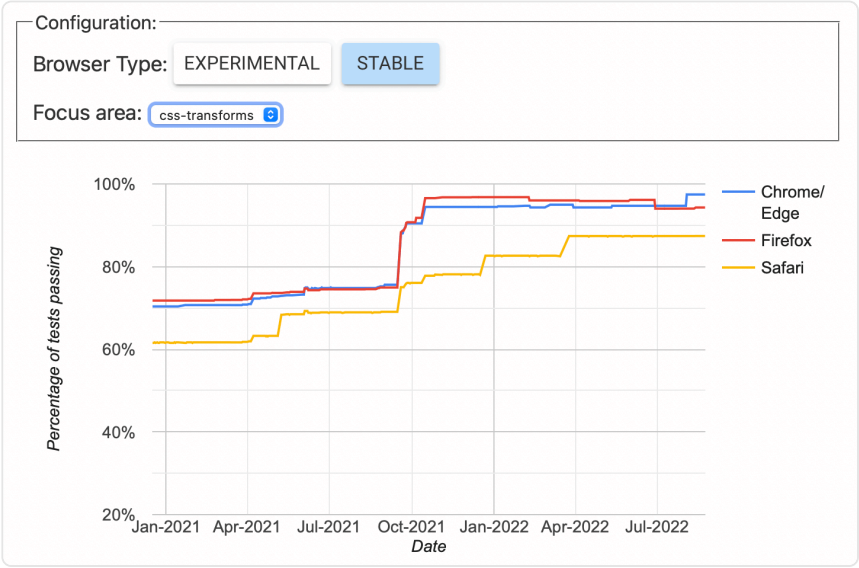



*図9.4. CSSトランスフォームWebページテストダッシュボード（安定版）。*
*(ソース: ウェブプラットフォームテスト[289])*

CSSトランスフォームは人気があり、長い間存在してきました。しかし、当初は多くの相互

288. https://almanac.httparchive.org/ja/2022/css#フィーチャークエリ
289. https://wpt.fyi/compat2021?feature=css-transforms&stable

運用性の問題があり、とくに `perspective:none` と `transform-style: preserve-3d` の問題がありました。そのため、多くのアニメーションが 不整合[290]という悩ましい問題を抱えていました。

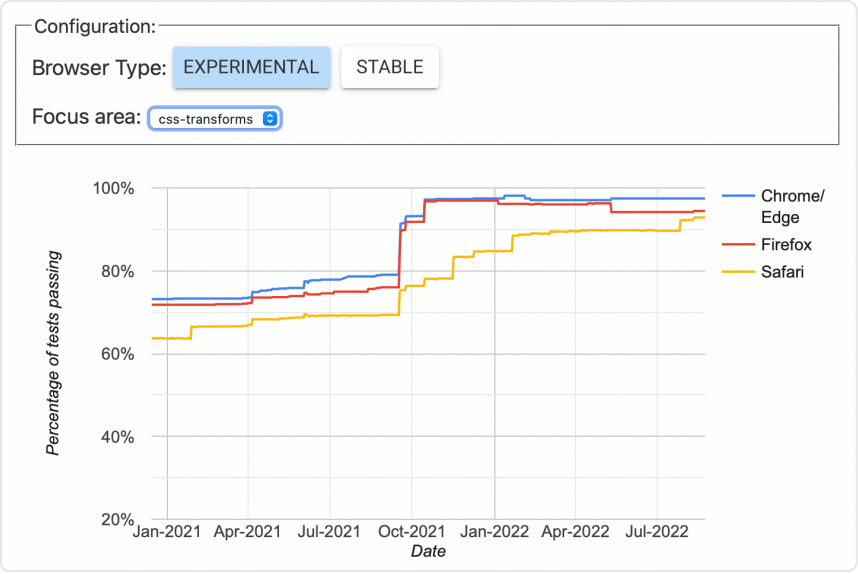



*図9.5. CSSトランスフォームWebページテストダッシュボード（実験的）。*
*(ソース: ウェブプラットフォームテスト[291])*

上記と同じCSSトランスフォームを実験的なブラウザで表示した最近のcompat 2021のグラフを見ると、すべてのブラウザが実験的なバージョンで90%以上のスコアを獲得しており、ブラウザの将来のバージョンを示しています。Interop 2022の一環として、Compat 2021の継続的な作業が含まれているため、これは安定したブラウザで大きく目に見える改善が続いている分野の1つです。

## アスペクト比

`アスペクト比` は2021年に開発された新機能です。その潜在的な有用性を考慮し、私たちは最初から高い相互運用性を目指すことにしました。

290. *https://web.dev/compat2021/#css-transforms*
291. *https://wpt.fyi/compat2021?feature=css-transforms*

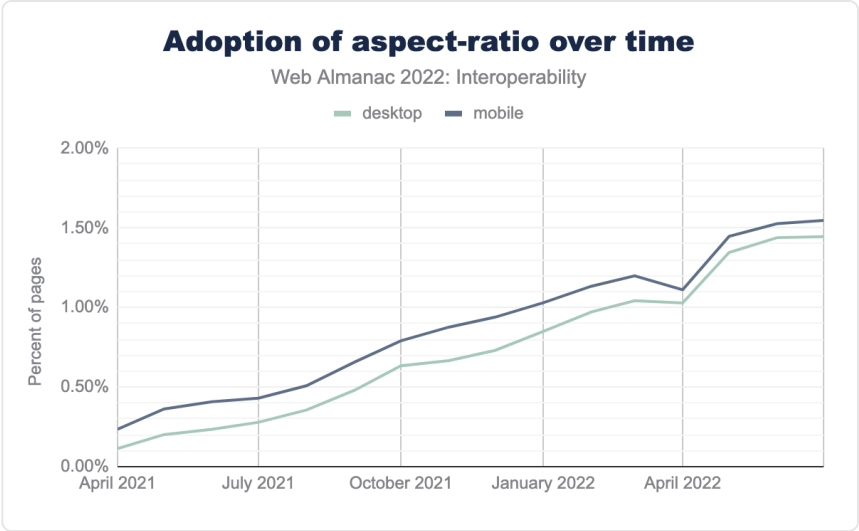



*図9.6. アスペクト比の経年変化。*
*(ソース: Chromeステータス[292])*

2022年、`アスペクト比`はすでにアーカイブクロールの2%のURLのCSSに登場しています。これらのページの2%が`アスペクト比`を使用しているという意味ではありません。これらのページでどのルールが適用されているかは別の問題で、デスクトップではページビューの1.55%、モバイルでは1.44%という控えめな結果になっています。それでも、成長グラフは着実に採用が増加していることを示しています。これは、今後追跡していく上で興味深い指標となるでしょう。

## Interop 2022

以前の *Compat* の取り組みと同様に、改名された *Interop* の取り組みではバグの収集から良い最終的な実装への着地、比較的新しいがすぐに出荷される機能まで、さまざまなものが混在しています。まずはバグから。。。

### バグ

多くの場合、成熟した機能であるにもかかわらず、さまざまなブラウザでボロボロのバグが報告されています。ボロボロのバグがあるということは、オーサリング体験が個々の合格率が意味するよりもずっと悪くなる可能性があるということです。たとえば、すべてのブラウ

292. https://chromestatus.com/metrics/css/timeline/popularity/657

ザが70%の合格率を報告しても、すべてのブラウザが異なる30%で不合格だった場合、実際の相互運用性はかなり低くなります。Interop 2022における私たちの焦点の大部分は、このような機能に関する実装の調整とバグの解消です。

### フォーム

ウェブの歴史のもっとも長い間、フォームはかなり重要な役割を果たしてきました。2022年には、デスクトップ ページの69% 以上が `<form>` 要素を含んでいます。これらは多くの投資を受けてきましたが、それにもかかわらず、開発者が仕様と異なるケースを見つけたり、時には微妙な方法で他の実装と異なるケースを見つけたりするため、多くのブラウザのバグの原因となっています。私たちは、合格率が非常に低い200のテスト[293]を特定しました。個々のスコアは、~62% (Safari) から ~91% (Chrome) までの範囲でしたが、やはり各ブラウザにはサポートに異なるギャップがありました。

私たちは実験的なリリースでこれらのギャップを埋めるためにかなり急進的な進歩を遂げました。HTTP Archiveのデータを使って、利用状況や採用状況を追跡できることはおそらくほとんどありませんが、開発者の苦痛やフラストレーションが軽減され、個々のブラウザで回避策を必要とすることが少なくなることを願っています。

293. *https://wpt.fyi/results/?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-forms*

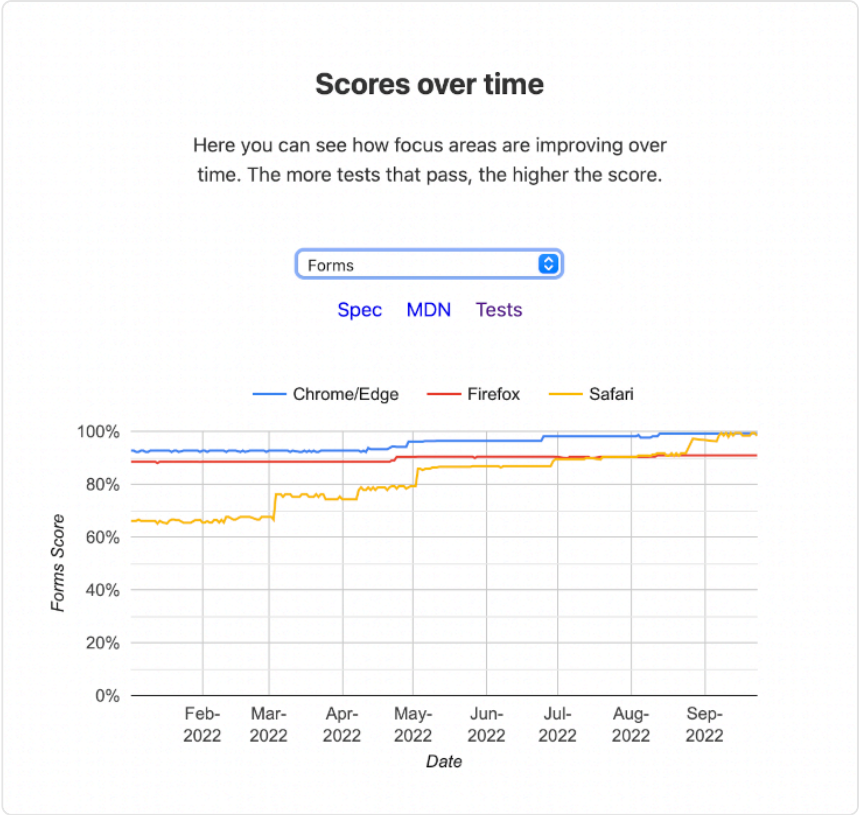



*図9.7. WPTダッシュボードのフォーム（実験的）。*
*(ソース: ウェブプラットフォームテスト[294])*

### スクロール

長年にわたり、私たちは新しいパターンを追加し、`scroll-snap`、`scroll-behavior`、`overscroll-behavior` などのスクロール体験に関する新しい能力を開発してきました。2022年には、これらの主要プロパティを含むCSSスタイルシートの数はこのようになっていました：

294. https://wpt.fyi/interop-2022?feature=interop-2022-forms&stable

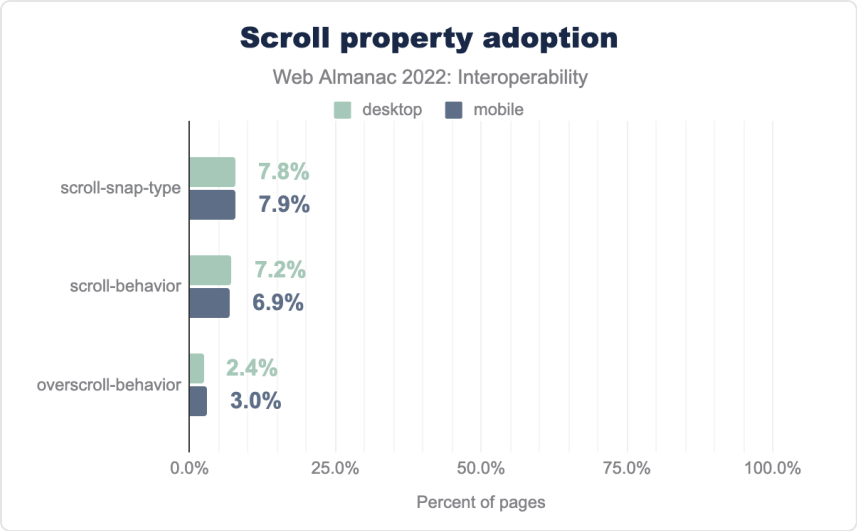



*図9.8. スクロールプロパティの採用。*

残念なことに、この分野には多くの非互換性が残っており、スクロールの非互換性に対処することは開発者を苦しめます。スクロールに関する ウェブプラットフォームテスト106[295] を特定しました。このプロセスの開始時点では、安定版（stable-release）のスコアは約70%（FirefoxとSafari）から約88%（Chrome）でした。ギャップが異なるため、実際の「相互運用性」交差点はこれらのどれよりも低くなっています。

295. *https://wpt.fyi/results/css?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-scrolling*

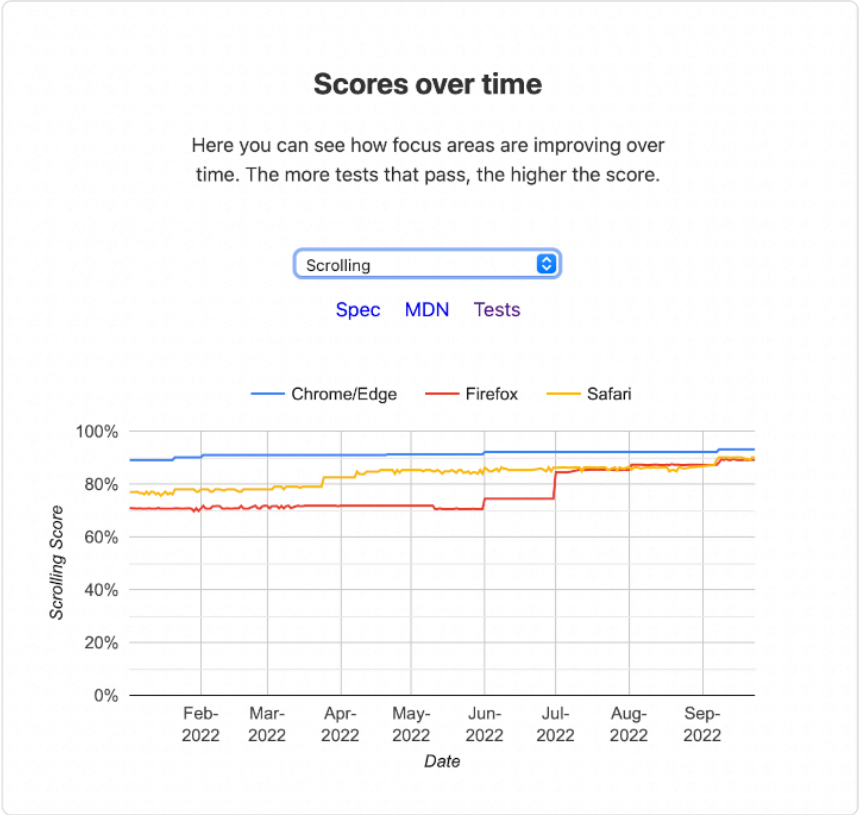



*図9.9. WPTのダッシュボードをスコーリング。*
*(ソース: ウェブプラットフォームテスト[296])*

これらの改良が時間とともに採用にどのような影響を及ぼすかを見積もるのは非常に困難ですが、私たちはこれらの指標を注視します。その間に、スクロール機能で相互運用性の問題を経験したことがある方は、もう一度見てみるとよいでしょう。このような改善が継続され、安定したブラウザのリリースに到達するにつれて、体験が大幅に改善されることを期待しています。

## タイポグラフィとエンコーディング

テキストのレンダリングはウェブの得意分野です。フォームのように、多くの基本的なアイデアは昔からありましたが、タイポグラフィやエンコーディングのサポートに関しては、多くのギャップや不整合が残っています。

296. https://wpt.fyi/interop-2022?feature=interop-2022-scrolling&stable

Interop 2022では、`font-variant-alternates`、`font-variant-position`、`ic`ユニット、CJKテキストエンコーディングに関する一般的な問題が取り上げられました。私たちは、ウェブプラットフォームテスト114のテスト[297]において、さまざまな種類のギャップを表す114のテストを特定しました。

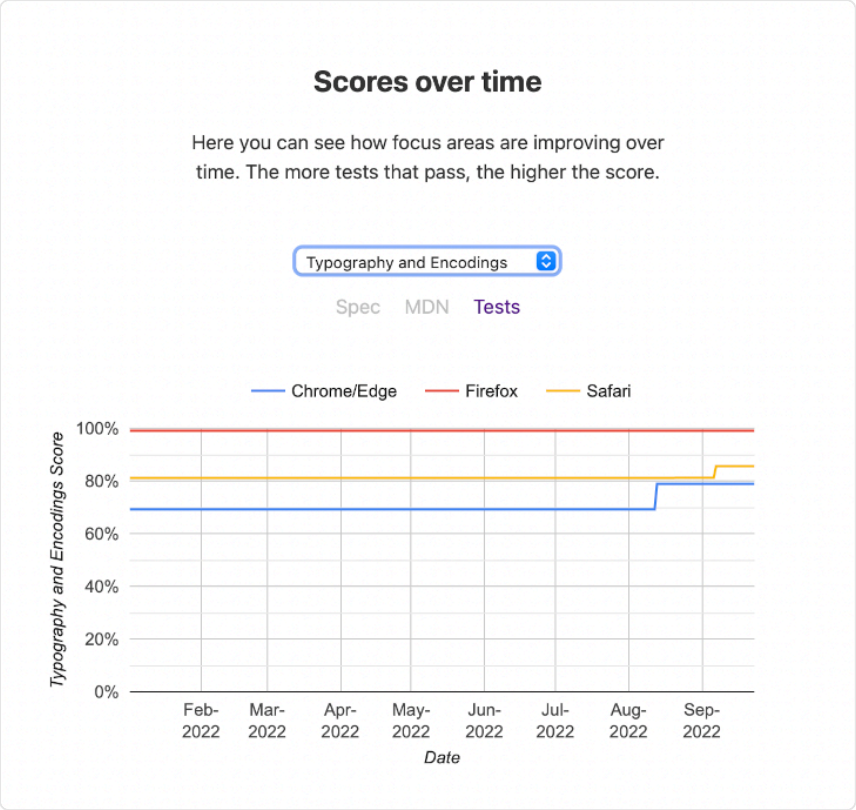



*図9.10. タイポグラフィとエンコーディングWPTダッシュボード。(ソース: ウェブプラットフォームテスト[298])*

Chromeは最近Safariとの差を縮め始めていますが、SafariとWebKitの両方がこの分野でFirefoxの完成度に追いつくにはまだ注意が必要です。

## 実装の完了

実装の調整はとくに困難です。実験と初期の実装体験の必要性と、作業がよく理解され、す

297. https://wpt.fyi/results/?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-text
298. https://wpt.fyi/interop-2022?feature=interop-2022-text&stable

べてのブラウザで実装を出荷する状態に達する可能性が非常に高いことを確実にするための十分な合意との間には微妙なバランスがあります。この調整には何年もかかることもあります。今年は、実装が完了し、少なくとも準備が整ったという合意が得られた3つの項目に焦点を当てました。 `<dialog>` 要素、CSS Containment、Subgridです。それぞれを見てみましょう。

## `<dialog>`

ダイアログ要素は、2014年8月のChrome 37ではじめて出荷されました。ダイアログを導入するには、”top-layer “や “inertness “など、サポートする多くの概念を導入し、定義する必要があります。また、多くの新しいアクセシビリティとUXの質問に答える必要があります。

さまざまなことが原因で、ダイアログの作業は長い間停滞していましたが、数年かけて回復しました。2017年4月には、Firefox Nightly 53にフラグが立てられました。それ以来、すべての質問に答えるために多くの作業が行われました。最終的な詳細が整理され、Interop 2022の一環として、まず良好な相互運用性を確保するために作業の優先順位がつけられました。私たちは88のテストを特定しました。これは、2022年3月にFirefox 98[299] と Safari 15.4[300] の両方の安定したブラウザでデフォルトで出荷され、すべてのブラウザで ~93% 以上のスコアを記録しました。

299. *https://developer.mozilla.org/ja/docs/Mozilla/Firefox/Releases/98*
300. *https://developer.apple.com/documentation/safari-release-notes/safari-15_4-release-notes*

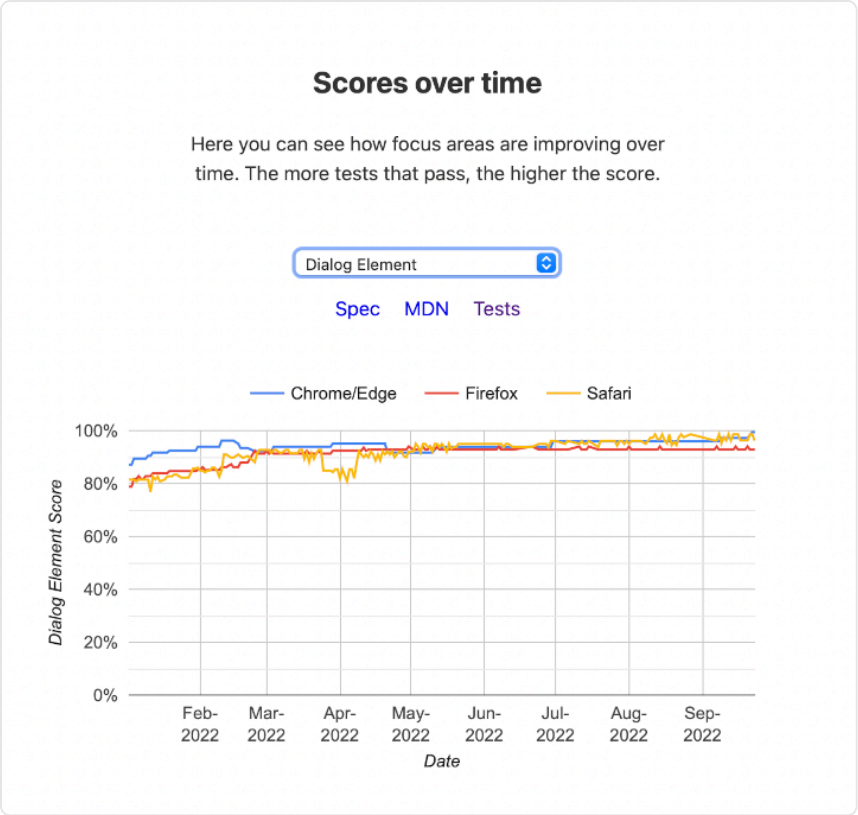



*図9.11. WPTダッシュボードのダイアログ要素。*
*(ソース: ウェブプラットフォームテスト[301])*

アーカイブがクロールするページのうち `<dialog>` を必要とするページの数を予測するのは難しいですが、その成長を追跡することは有益で興味深いでしょう。昨年、`<dialog>` をサポートしたブラウザは1つだけで、モバイルデータセットのページの~0.101%に表示されました。この章に使用したクロールの時点では、約5か月間全世界で出荷されており、~0.148%[302]で表示されていることがわかりました。まだ小さな数字ですが、これは昨年の今頃の ~146% です。来年もこの指標を追跡します。それまでの間、もしあなたが `<dialog>` を必要としているなら、朗報があります！

### CSSの格納

CSSコンテナメントは、CSSがどのように処理し、レンダリングすべきかという観点から、

301. *https://wpt.fyi/interop-2022?feature=interop-2022-dialog&stable*
302. *https://docs.google.com/spreadsheets/d/1grkd2_1xSV3jvNK6ucRQOOL1HmGTsScHuwA8GZuRLHU/edit#gid=2057119066*

ページのサブツリーを他の部分から分離するための概念を導入したものです。これは、パフォーマンスを向上させ、コンテナクエリ[303]を理解するための扉を開くために使用できるプリミティブとして導入されました。

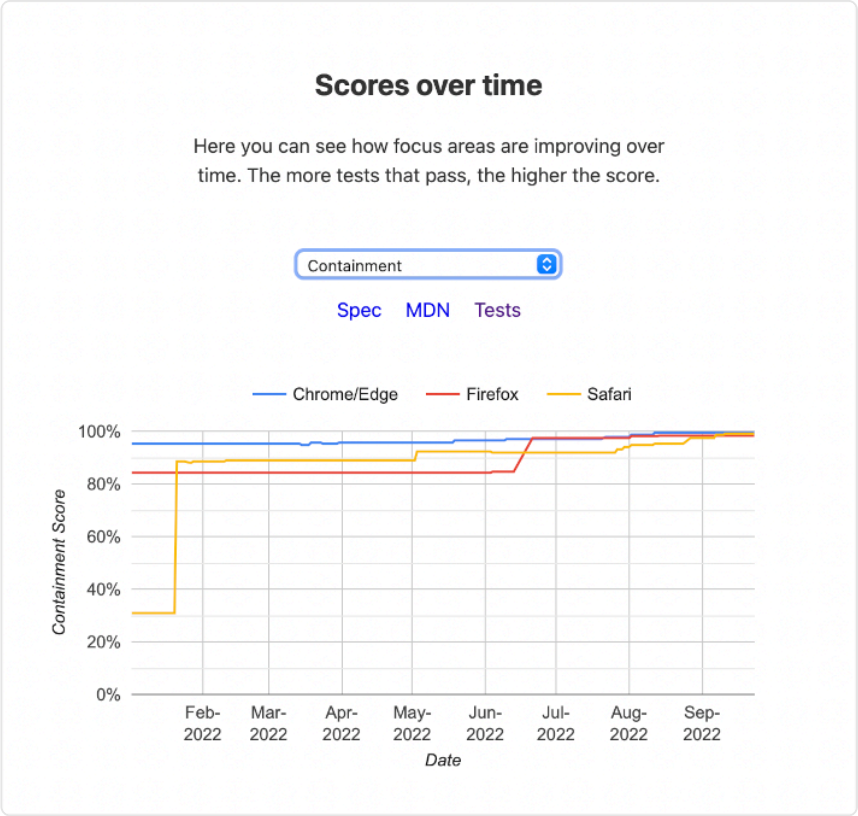



*図9.12. コンテインメントWPTダッシュボード。*
*(ソース: ウェブプラットフォームテスト[304])*

2016年7月にChrome安定版で初出荷。Firefoxは2019年9月に2番目の実装を出荷しました。今年、Interop 2022で取り上げられたのは、それが普遍的に利用可能になったときに、私たちが良い状態でスタートできるようにするためです。235テスト[305]を確認しました。Safariは2022年3月に安定版リリース15.4[306]でコンテントメント サポートを出荷しました。この1年を通して、各ブラウザはサポートを改善し、今では普遍的に利用できるようになりました。

303. *https://developer.mozilla.org/docs/Web/CSS/CSS_Container_Queries*
304. *https://wpt.fyi/interop-2022?feature=interop-2022-contain&stable*
305. *https://wpt.fyi/results/css/css-contain?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-contain*
306. *https://developer.apple.com/documentation/safari-release-notes/safari-15_4-release-notes*

**3.7%**

*図9.13. スタイルシートでコンテクメントを使用しているモバイルページの数。*

2022年のデータでは、モバイルでは3.7%、デスクトップでは3.1%のページで、スタイルシートにコンテントメントが表示されています。

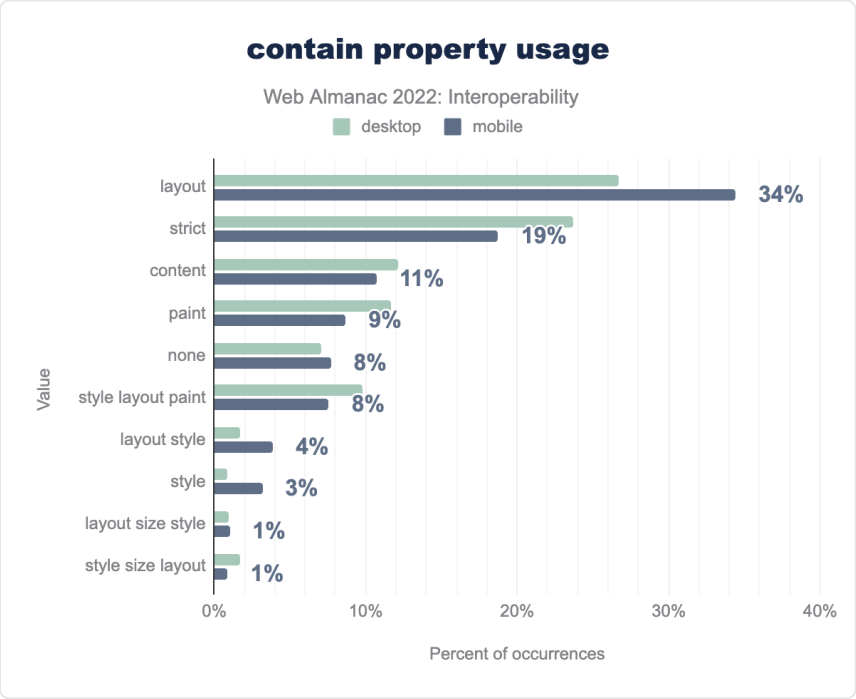



*図9.14. `contain` プロパティの使い方。*

上の図は、これらのページにおける値の相対的な出現を示しています。`layout` のコンテインはもっとも普及しており、`contain` の値の34%を占めています。

コンテナクエリは長年にわたって#もっともリクエストの多かったCSS機能第1位[307] であるため、これは長期にわたって追跡し続ける興味深い指標となるでしょう。コンテナクエリが普遍的に利用できるようになった今、基本的な概念に慣れ親しむ絶好の機会です。

なおChromeとSafariでは、ある程度のコンテナクエリのサポートがすでに提供されており、

307. https://2021.stateofcss.com/en-US/opinions/#currently_missing_from_css_wins

ポリフィルも利用可能というわけで、すでに`@container`ルールセットが含まれているスタイルシートがどれくらいあるのかも調べてみることにしました。

**0.002%**

*図9.15. `@container`ルールセットを含むモバイルページの割合。*

まだまだ少ないようです！モバイルデータセットでクロールした約800万ページのうち、コンテナクエリを使用しているのはわずか238ページです。コンテナクエリが新しく、まだ完全に出荷されていないことを考えると、これは驚くべきことではありません。しかし、将来的に採用を追跡するための良いベースラインを与えてくれます。

### サブグリッド

CSSのグリッド・レイアウトでは、コンテナーが行、列、トラックで子のレイアウトを表現できますが、これには常に限界があります。子ではない子孫も同じグリッドレイアウトに参加させる必要がしばしばあります。サブグリッド[308]はこのような問題の解決策です。2019年12月にFirefoxの安定版リリースではじめてサポートされましたが、他の実装はすぐには続きませんでした。

この待望の機能に関する作業を調整し、優れた相互運用性を確保することも、Interop 2022の目標でした。私たちは ウェブ・プラットフォーム・テストにおける51のテスト[309]をマークしました。

308. *https://developer.mozilla.org/ja/docs/Web/CSS/CSS_Grid_Layout/Subgrid*
309. *https://wpt.fyi/results/css/css-grid/subgrid?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-subgrid*

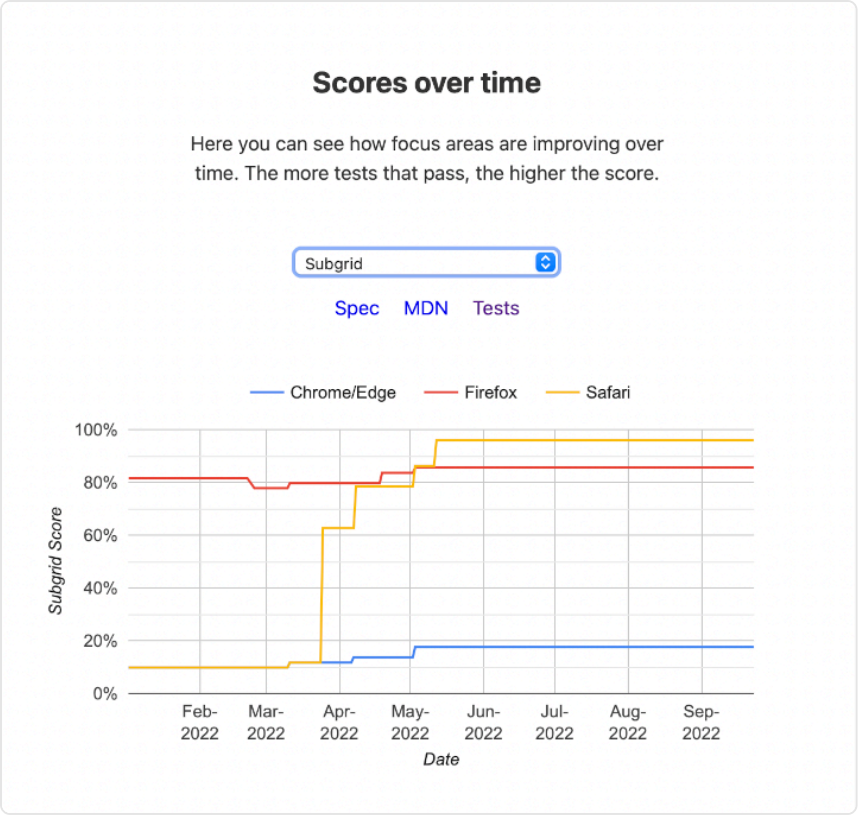



*図9.16. サブグリッドWPTダッシュボード。*
*(ソース: ウェブプラットフォームテスト[310])*

この記事を書いている時点では、非常に多くの進展があり（Safariは現在もっとも完成度が高い）、安定した出荷ブラウザには少なくとも2つの実装（SafariとFirefox）があります。年末までにChromeで急速な改善が見られることを期待しています。

**0.002%**

*図9.17. スタイルシートで `subgrid` を使用しているモバイルページの割合。*

これはまだすべての安定したブラウザで完全に利用できるわけではありませんが、データセットにはすでに少量の利用が含まれています。

310. *https://wpt.fyi/interop-2022?feature=interop-2022-subgrid&stable*

## 新機能

今年、CSSの範疇に入るすべての新機能と、それらに関するもっとも多くのデータは、CSSの章で扱われます。ここでは、主にいくつかのハイライトを取り上げます。

### 色空間と関数

ウェブ上の色は、常に魅力的な挑戦に満ちています。長年にわたり、私たちは作者に、結局は同じsRGB[311]色であるものを表現する多くの方法を与えてきました。つまり、色の名前（`red`）として書くことができます。単純なことです。

しかし、16進数の`(#FF0000)`を使うこともできます。人間は一般的に16進数で考えないので、`rgb()`色関数（`rgb(255,0,0)`）を追加しました。どちらも2つの異なる、同等の番号体系を使っていることに注意してください。また、これらはブラウン管ディスプレイで一般的だった、個々の光の強さの混合という観点から物事を表現するものです。

RGBの色の構成方法は、人間にとって非常に視覚化しにくいので、sRGBの色を（おそらく？）わかりやすく表現するために、`hsl(0, 100%, 50%)`や`hwb(0, 0%, 0%)`のような他の座標系を開発しました。ただし、繰り返しますが、これらはsRGB座標系です。

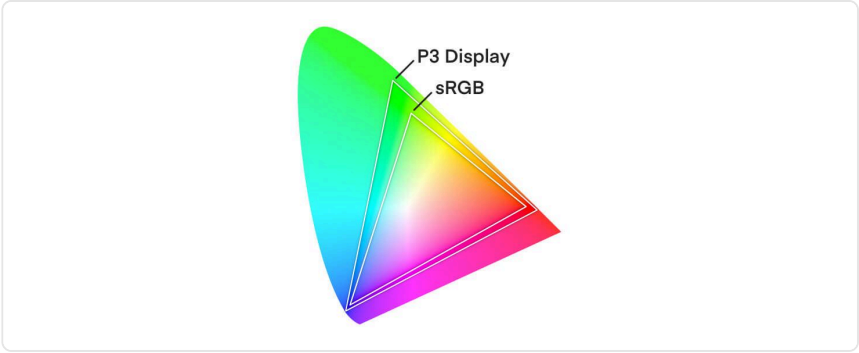



*図9.18. sRGBと比較したp3色空間。*

では、ディスプレイの能力が限界を超えるとどうなるのでしょうか？実際、広色域ディスプレイに見られるように、これは今日のケースです。

2017年にリリースされたSafari 10で、AppleはP3カラー画像のサポートを追加しました。CIELABモデル[312]に基づいて、利用可能な色域空間をすべてサポートするために、新しい`lab()`座標系と`lch()`座標系がCSSに追加されました。これらの座標系は知覚的に統一さ

311. *https://en.wikipedia.org/wiki/SRGB*
312. *https://en.wikipedia.org/wiki/CIELAB_color_space#Cylindrical_model*

れており、以前は表現できなかった色を表現できるようになりました（また、サポートが不足している場合にどうすべきかを定義しています）。これらのサポートは2021年9月にSafari 15ではじめて出荷されました。

`lab()`と`lch()`の色域空間が充実し知覚的な均一性が向上したことで、2つの色を受け取り、指定した色空間で指定した量だけ混合した結果を返す`color-mix()`のような新しい色関数にも、より簡単にフォーカスできるようになりました。

Interop 2022では、優れた相互運用性を優先することを目標に、これらの項目に関する 189 のテスト[313]を実施しました。FirefoxとChromeは着実に改善していますが、この分野ではまだかなり遅れています。1つの課題は、必然的に、多くの下位レベルのサポート -基礎となるグラフィックス ライブラリやレンダリング パイプラインなど全体を通して[314] - もsRGBで処理するように構築されているため、サポートを追加するのが簡単ではないということです。

313. *https://wpt.fyi/results/css/css-color?label=master&label=experimental&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-color*
314. *https://youtu.be/eHZVuHKWdd8?t=906*

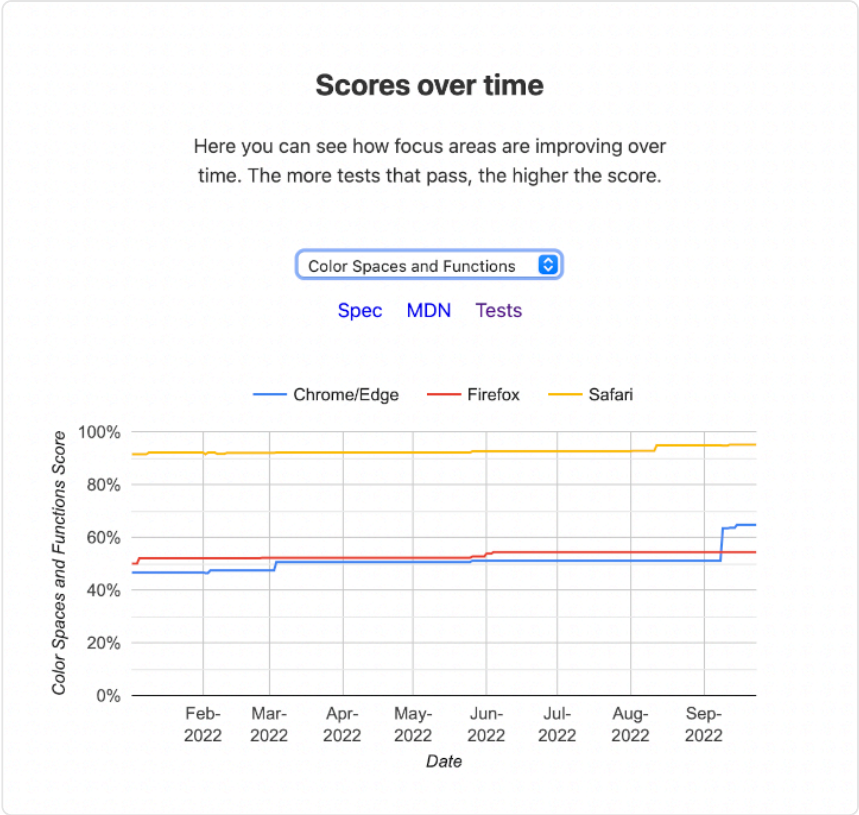



*図9.19. カラースペースと機能WPTダッシュボード。*
*(ソース: ウェブプラットフォームテスト[315])*

### ビューポートユニット

2020 MDNブラウザ互換性レポートでは、既存のvh/vw単位で報告されたビューポートのサイズで動作する機能は、共通のテーマでした[316]。ブラウザがさまざまなインターフェイスの選択を体験し、ウェブサイトがさまざまなデザインのニーズを持つようになったためCSS Working Groupはもっとも大きい（`lv*` 単位）、もっとも小さい（`sv*` 単位）、そして動的な（`dv*` 単位）ビューポート測定を測定するためにビューポート単位のいくつかの新しいクラスを定義しました。すべてのビューポート関連のメジャーは、同様の単位を含みます。

315. https://wpt.fyi/interop-2022?feature=interop-2022-color&stable
316. https://insights.developer.mozilla.org/reports/mdn-browser-compatibility-report-2020.html#findings-viewport

- 幅の1%（`vw`、`lvw`、`svw`、`dvw`）
- 高さの1%（`vh`、`lvh`、`svh`、`dvh`）
- インライン軸のサイズの1%（`vi`、`lvi`、`svi`、`dvi`）
- 最初に含まれるブロックのサイズの1%（`vb`、`lvb`、`svb`、`dvb`）
- 2つの次元のうち小さい方（`vmin`、`lvmin`、`svmin`、`dvmin`）
- 2つの次元のうち大きい方（`vmax`、`lvmax`、`svmax`、`dvmax`）

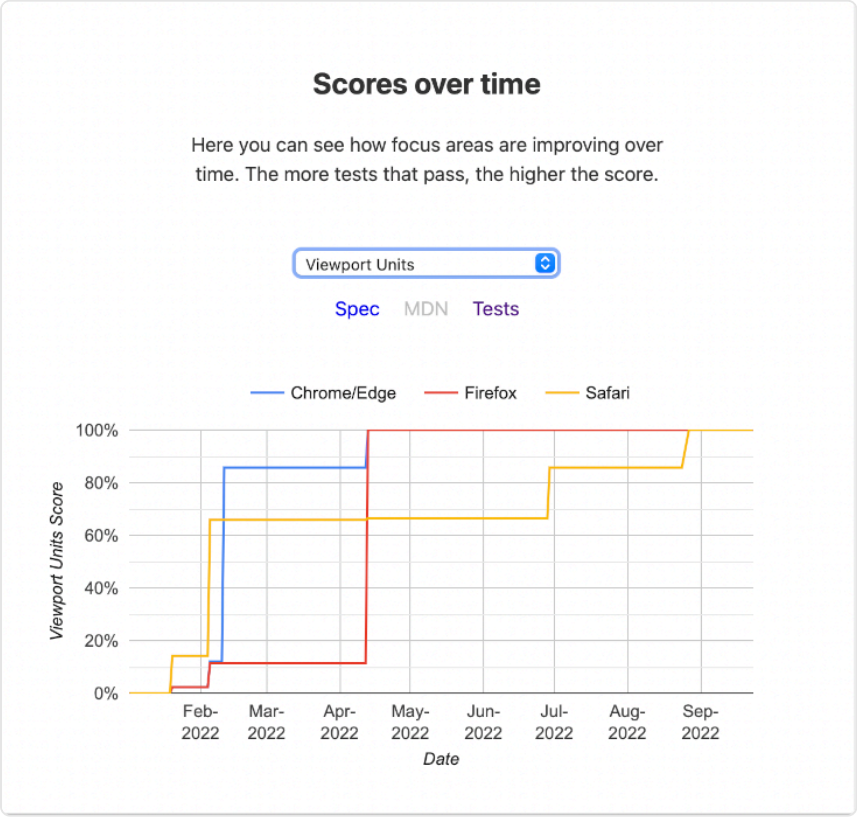



*図9.20. ビューポート・ユニットWPTダッシュボード（実験的）。*
*(ソース: ウェブプラットフォームテスト[317])*

Interop 2022では、これらのユニットのさまざまな側面を検証するための7つのテストが特

317. https://wpt.fyi/interop-2022?feature=interop-2022-viewport

定されました。Safariは2022年3月に、Firefoxは5月末に、これらのユニットをはじめてサポートしました。本稿執筆時点では、Chromiumの実験的ビルドでサポートされています。

この記事を書いている時点では、HTTP Archiveはまだこれらのユニットの使用を発見していませんが、非常に新しいものです。私たちは今後もこのユニットの採用を追跡します。

### カスケードレイヤー

カスケード・レイヤーはCSSの興味深い新機能で、CSSにずっと存在していた基本的なアイデアの上に構築されています。作者として、私たちがルールの重要性を表現する主な手段は、「具体性」でした。これは多くのことに有効ですが、デザインシステムやコンポーネントのアイデアを取り入れようとすると、すぐに扱いにくくなります。ブラウザはまた、UAスタイルシートと呼ばれるもので内部的にCSSを使います。しかし、UAスタイルシートでは、ふつうは特異性に関連した争いは起きないことにお気づきかもしれません。それは、CSSが機能する仕組みの中に、ルールの“レイヤー“が組み込まれているからです。カスケード・レイヤーは、作者が同じメカニズムにプラグインし、CSSと特異性の課題をより効果的に管理する方法を提供します。Miriam Suzanne[318]は、より詳しい説明とガイド[319]を書きました。

318. *https://twitter.com/TerribleMia*
319. *https://css-tricks.com/css-cascade-layers/*

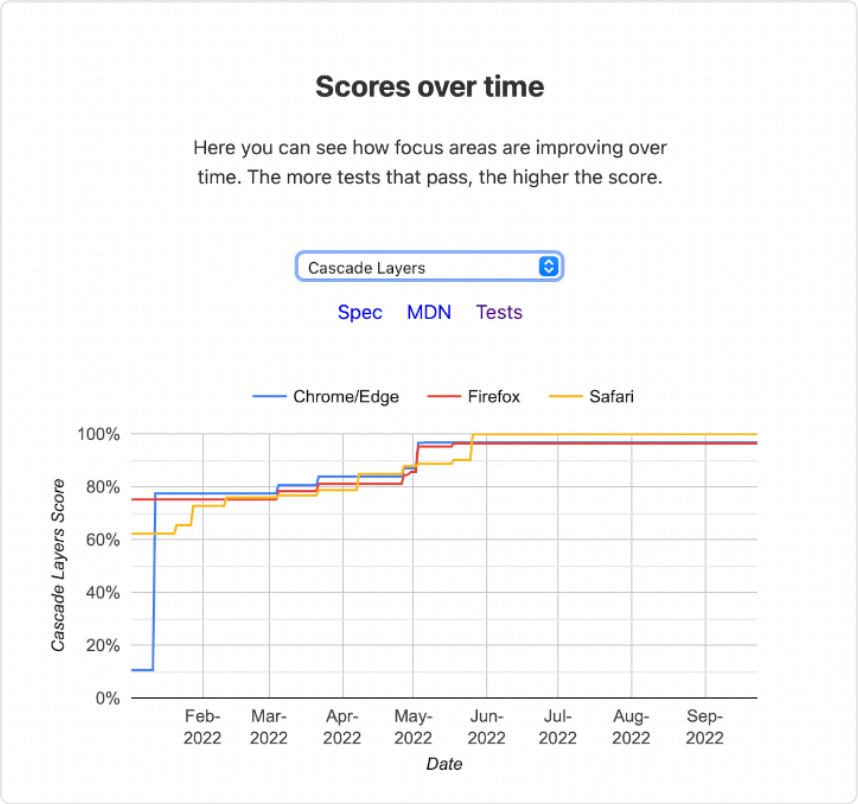



*図9.21. カスケードレイヤーWPTダッシュボード（実験的）。*
*(ソース: ウェブプラットフォームテスト[320])*

これをうまく開始するために、Interop 2022はウェブプラットフォームテストで 31個のテストを定義しました[321]。今年の初めには、安定したブラウザでのサポートは存在しませんでしたが、4月以降、3つのエンジンの安定したリリースで普遍的に実装されています。以下は開発の様子です。

**0.003%**

*図9.22. `@layer` ルールセットを含むモバイルページの割合。*

320. https://wpt.fyi/interop-2022?feature=interop-2022-cascade
321. https://wpt.fyi/results/css/css-cascade?label=experimental&label=master&product=chrome&product=firefox&product=safari&aligned&view=interop&q=label%3Ainterop-2022-cascade

今年のデータセットの時点では、レイヤーは野生のごく少数の場所で発生。

Web Almanacの今後のエディションでは、カスケード・レイヤーの採用と傾向を追跡します。整合性のある作業、緊密なリリース、優れた相互運用性への早期集中が、その潜在能力を発揮し、不満を軽減するのに役立つことを願っています。

どこでも出荷されている今こそ、Cascade LayersがCSSのコントロールにどのように役立つかを学ぶ絶好の機会です。

## 結論

相互運用性は標準の目標であり、最終的には大規模な採用の鍵となります。しかし、実際には、相互運用性に到達するためには、複雑な独立した作業、集中力、予算、優先順位の集大成が必要です。歴史的には、実装の着地と非互換性の間に何年ものギャップが生じることもあり、これは時として困難なことでした。ブラウザーベンダーはこのフィードバックを聞き、既存のギャップを埋めるための協調的な取り組みに重点を置き、新しい実装が非常に高い相互運用性を持って到着するまでの期間を短縮する努力を始めました。

私たちは、今年行われたこの作業のレビューが、開発者に情報を提供し、これらの機能の採用や注目を促す一助となることを願っています。私たちはできる限りの指標を追跡し続け、私たちがどのようにやっているかという感覚を伝え、どこにどのように焦点を当てるかに影響を与えるために、データをどのように利用できるかということに目を向けていきます。

### 著者

Brian Kardell

@briankardell bkardell https://bkardell.com

Brian Kardellは、Igalia[322]の開発者支持者であり、W3C諮問委員会代表、標準化貢献者、ブロガー[323]です。Extensible Web Community Group の創設者であり、The Extensible Web Manifesto[324] の共著者。

322. *https://igalia.com*
323. *https://bkardell.com*
324. *https://extensiblewebmanifesto.org*

# 部 II 章 10
# SEO

***Sophie Brannon*、 *Itamar Blauer* と *Mordy Oberstein* によって書かれた。**
*Patrick Stox*、 *Tushar Pol*、 *Mobeen Ali*、 *Dave Smart* と *John Murch* によってレビュ。
*Colt Sliva*、 *JR Oakes* と *Derek Perkins* による分析。
*Michael Lewittes* 編集。
*Sakae Kotaro* によって翻訳された。

## 序章

検索エンジン最適化（SEO）は、ウェブサイトやページの可視性を向上させ、検索エンジンの検索結果で上位に表示されるようにするためのデジタル技術です。多くの場合、技術的な設定、コンテンツの作成、リンクの獲得を組み合わせ、検索者のクエリと意図に対する関連性を向上させることを目的としています。SEOの人気は高まり続け、もっとも普及しているデジタルマーケティングチャネルのひとつとなっています。

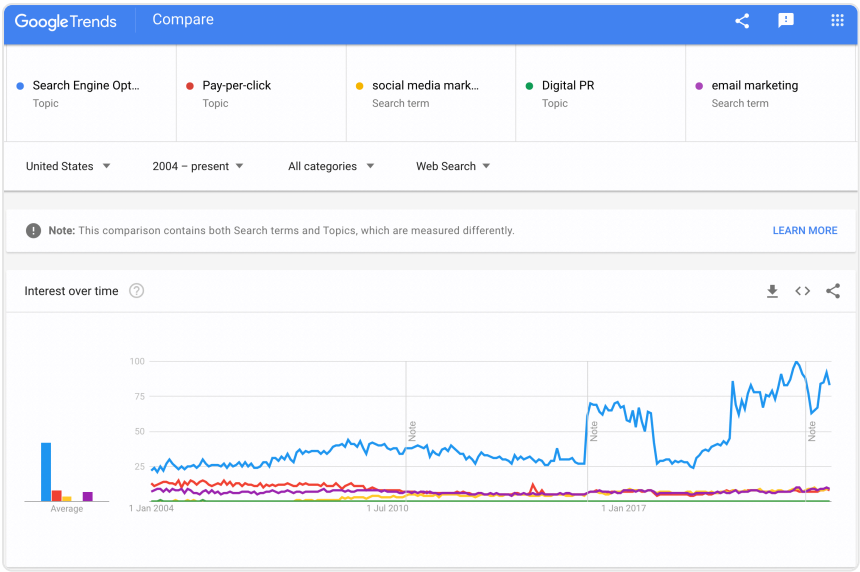



*図10.1. SEOとペイパークリック、ソーシャルメディアマーケティング、Eメールマーケティングのトピックの方向性検索人気を比較したGoogle Trends。*

私たちはこれまでにない新しい洞察を明らかにするカスタムメトリクスを使用して、ウェブ上の800万以上のホームページを分析し、2021[325]や、場合によっては2020[326]の結果と比較しました。注：私たちのデータ、とくにLighthouseとHTTP Archiveからのデータは、サイト全体のクロールではなく、ウェブサイトのホームページだけに限定されています。これらの制限については、Methodologyをご覧ください。

検索エンジンにやさしいウェブとは？

## クローラビリティと索引性

クロールとインデックスは、Googleや他の検索エンジンが最終的に検索結果ページに表示する内容のバックボーンです。クロールとインデックスがなければ、ランキングは成り立ちません。

プロセスの最初のステップは、クロールによるウェブページの発見です。数多くのページがクロールされますが、実際にインデックスされるページは少なく、基本的に検索エンジンのデータベースに保存され、分類されます。そして、検索者のクエリに基づいて、インデックスに登録されたページの中からマッチするページが提供されます。

325. *https://almanac.httparchive.org/ja/2021/seo*
326. *https://almanac.httparchive.org/ja/2020/seo*

このセクションでは、ウェブサイトをクロールしインデックスを作成するボットに関連するウェブの状態について説明します。サイトは検索エンジンのボットにどのような指示を与えているのでしょうか？Googleが正しいページを検索結果に表示するために、また重複したページを検索結果に表示しないために、サイトは何をしているのでしょうか？

クローラビリティとインデクサビリティに影響を与えるウェブとそのいくつかの側面を探ってみましょう。

## Robots.txt

robots.txtファイルは検索エンジンのクローラーを含むボットに対して、どこに行くことができどこに行くことができないか、つまり何をクロールすることができ、何をクロールすることができないかを指示します。

### Robots.txtのステータスコード

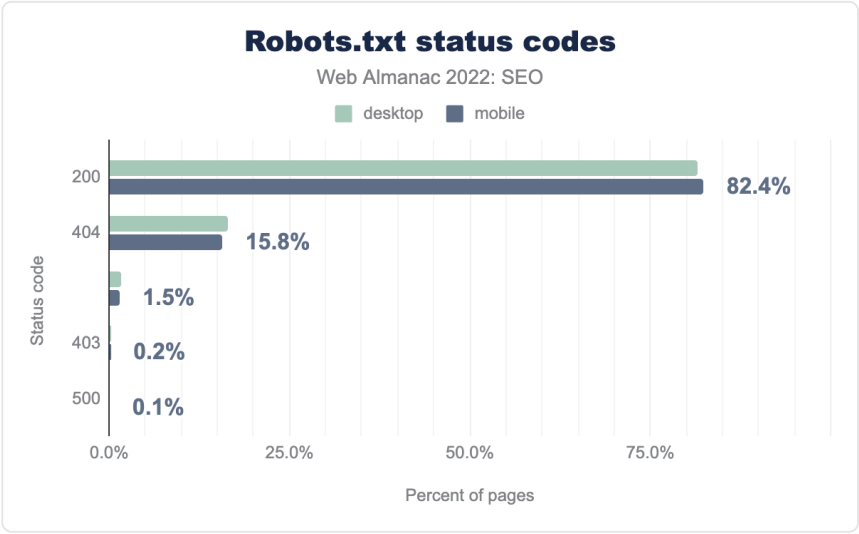



*図10.2. Robots.txtのステータスコード。*

2021年と比較して、2022年にはrobots.txtファイルが200ステータスコードを返すサイトの割合がわずかに増加しています。2022年には、デスクトップサイトのrobots.txtファイルの81.5%が200のステータスコードを返し、モバイルサイトの82.4%が同じコードを返しました。これは、2021年にデスクトップサイトとモバイルサイトのrobots.txtファイルがそれぞれ81%と81.9%で200のステータスコードを返したことと比較しています。

同時に、2022年には、404ステータスコードを返すrobots.txtファイルの割合が、2021年に比べてわずかに減少しました。昨年は、デスクトップサイトのrobots.txtファイルの17.3%が404を返し、モバイルサイトのrobots.txtファイルの16.5%が404を返していました。2022年、404ステータスコードを返すのは、デスクトップサイトのrobots.txtファイルの16.5%、モバイルサイトのrobots.txtファイルの15.8%です。

2021年同様、残りのステータスコードは、最小限の数のrobots.txtファイルに関連付けられています。

注：上記のデータは、robots.txtファイルがどれだけ最適化されているかを示すものではありません。200のステータスコードを返すファイルであっても、おそらくサイト全体の健全性にとって最善ではないディレクティブが含まれている可能性があります。

### Robots.txtのサイズ

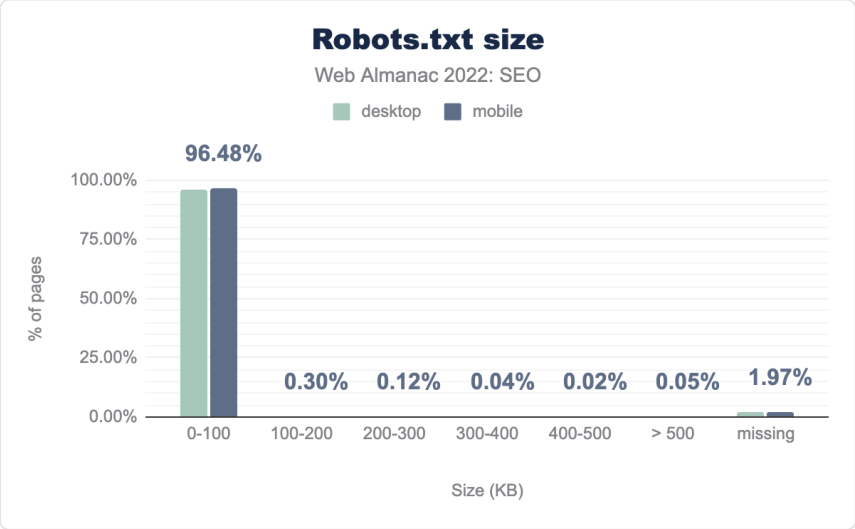



*図10.3. Robots.txtのサイズコード。*

予想通り、robots.txtファイルの圧倒的多数は0〜100KBと非常に小さいものでした。

Googleのrobots.txtファイルの上限は500KiBです。ファイルがこの制限に達した後に見つかったディレクティブは、検索エンジンによって無視されます。このカテゴリーに分類されるrobots.txtファイルはごく少数です。とくに、デスクトップとモバイルの両方のサイトのわずか0.005%が、Googleの最大制限を超えるrobots.txtファイルを含んでいます（これは2021年のデータと一致しています）。ファイルサイズが制限を超える場合、Googleはディ

レクティブを統合することを推奨しています[327]。

### Robots.txtのユーザーエージェントの使い方

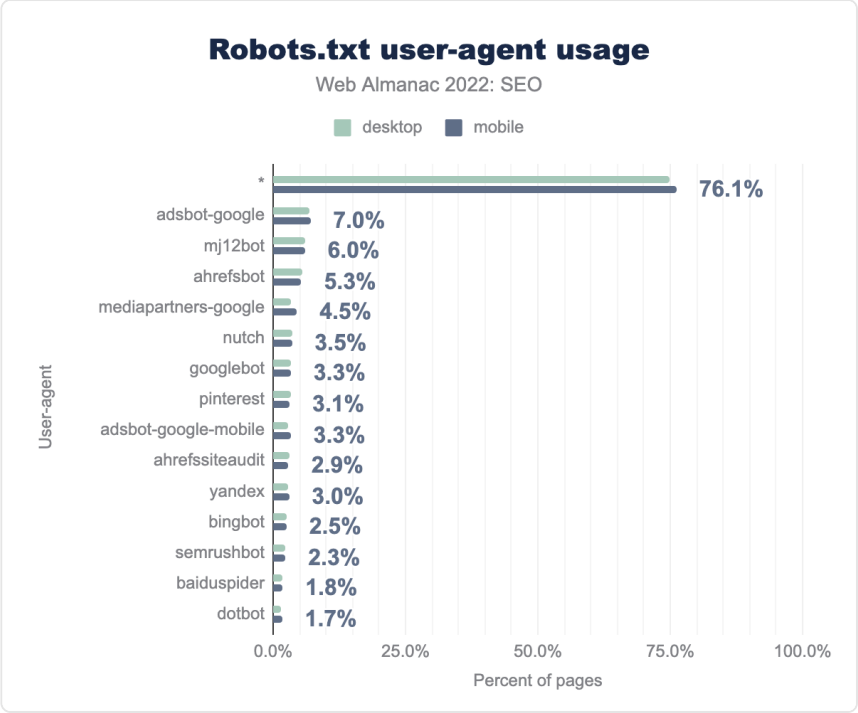



*図10.4. Robots.txtユーザーエージェントの使用。*

現在もっとも多くのウェブサイト（デスクトップ74.9%、モバイル76.1%）が、robots.txtファイル内で特定のユーザーエージェントを指定しておらず、これはファイル内のディレクティブがすべてのユーザーエージェントに適用されることを意味します。これは、デスクトップのrobots.txtファイルの74%、モバイルのrobots.txtファイルの75.2% が特定のユーザーエージェントを指定していなかった2020年のデータと一致しています。

興味深いことに、Bingbotはもっとも多く指定されたユーザーエージェントのトップ10には入っていません。SEOツールに関しては、2021年と同様、MajesticのボットとAhrefsのボットがもっとも多く指定されたユーザーエージェントのトップ5に入り、Semrushのボットがもっとも指定されたユーザーエージェントのトップ15に入りました。

検索エンジン別では、GooglebotがrobotsファイルのTxt3.3%でトップ、Bingbotは2.5%でし

327. https://developers.google.com/search/docs/advanced/robots/robots_txt

た。興味深いことに、モバイルサイトのrobots.txtファイルとBingbotを指定するデスクトップファイルの間には、2021年にはほぼ完全なパーセンテージポイントの差がありました。2022年にはこのような差はなく、基本的に一様です。

注目すべきは、2021年にはrobots.txtファイルのわずか0.5%にYandexbotが指定されていたことです。2022年には6倍に増加し、3%のファイルがYandexbotを指定しています。

## `IndexIfEmbedded` タグ

2022年1月、Googleは *indexifembedded* という新しいロボットタグを導入しました。このタグは、noindexタグが適用されている場合でも、ページ上のiframeにコンテンツが埋め込まれている場合にインデックスを制御できます。

まず、この新しいタグが適用される可能性のあるページの割合を決定することから始めましょう。

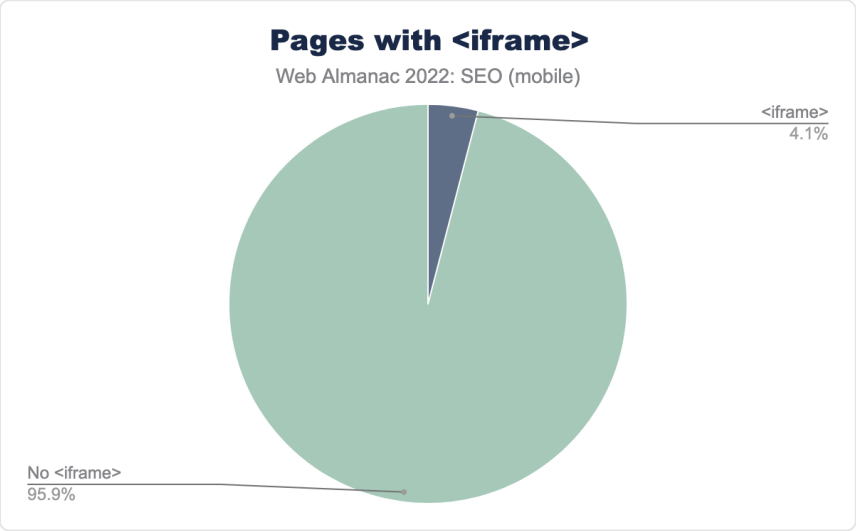



*図10.5. `<iframe>` を含むページ。*

ページの4%強が `<iframe>` 要素を含んでいます。 この要素を含むページの4.1%のうち、76%はiframeがnoindexされており、新しい `indexifembedded` タグの潜在的な使用例となっています。

しかし、 `indexifembedded` ロボットタグを採用しているサイトはごくわずかです。このタグは、調査対象となったページのわずか0.015%にしか見当たりません。

`indexifembedded` タグを採用しているページのうち、98.3%はヘッダーに実装しており、66.3%はHTMLを使用しています。

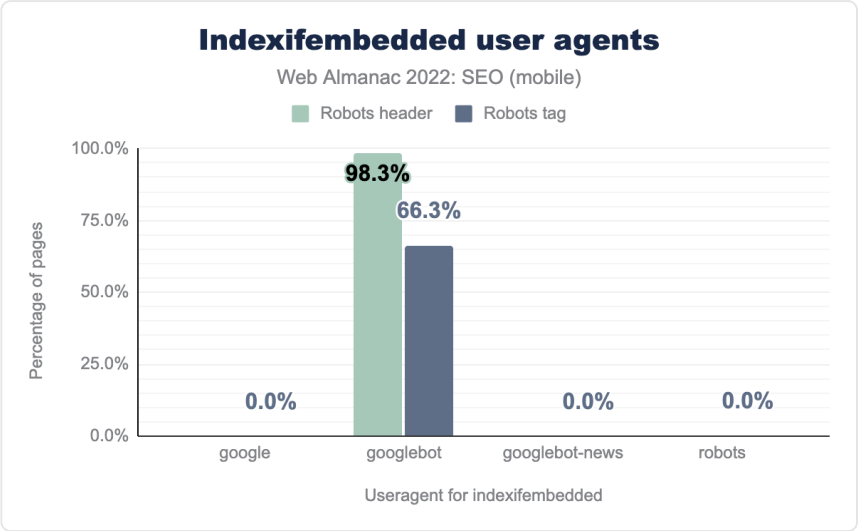



*図10.6. `Indexifembedded` ユーザーエージェント。*

## 無効なヘッド要素

`<head>` 要素はページのメタデータのコンテナーとして機能します。SEOの観点からは、ページのタイトルタグとメタディスクリプションは `<head>` 要素の中にあり、ロボットメタタグも同様です。

しかし、すべての要素が `<head>` 要素に属するわけではありません。Googleはページの `<head>` 内で無効な要素に出くわした場合、 `<head>` の最後に到達したとみなし、残りのコンテンツを発見することはありません[328]。

2022年のデータでは、デスクトップページの12.7%、モバイルページの12.6%に無効な要素が `<head>` に含まれています。

328. *https://developers.google.com/search/docs/advanced/guidelines/valid-html*

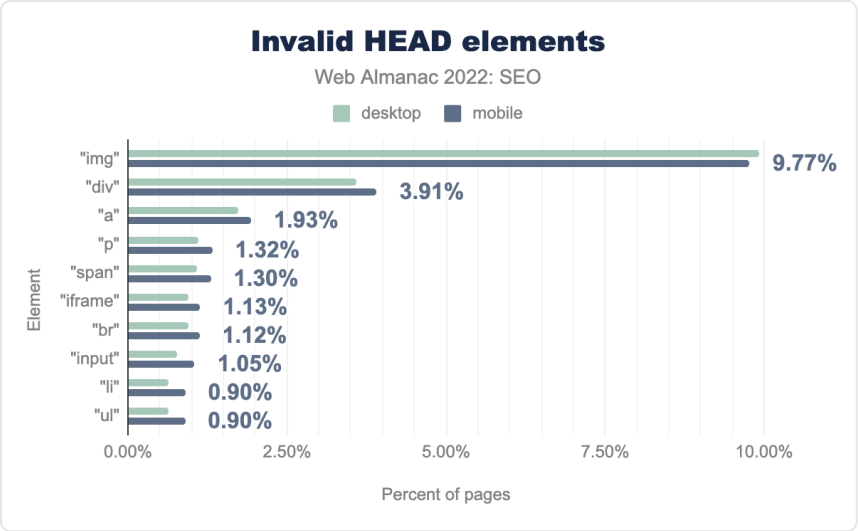



*図10.7. 無効な `<head>` 要素です。*

`<head>` にもっとも誤って適用されているのは、`<img>` 要素です。モバイルページの9.7%、デスクトップページの9.9%で、`<head>` 内に誤って配置されています。

`<div>` 要素は、2022年のデータセット内の3%以上のページで、`<head>` 内に誤って配置されている唯一の要素です。デスクトップページの3.5%、モバイルページの3.9%で、`<head>` に誤って適用されています。

## 正規化タグ

正規化タグは、重複コンテンツのページを定義するときや、検索エンジンが優先順位をつけるときに伝統的に使用されています。これはHTMLコードのスニペット（`rel="canonical"`）で、ウェブマスターは検索エンジンに対して、どのページが「優先される」バージョンであるかを定義できます。これらはディレクティブではなく、「ヒント」として機能します。そのため、Googleなどの検索エンジンは、そのページがユーザーにとってどの程度有用であるかに基づいて、そのページの正規化バージョンを決定します。正規化タグは、リンクなどの他のシグナルを統合したり、トラッキングの指標を簡素化したり、シンジケートされたコンテンツをよりよく管理するためにも使用できます。

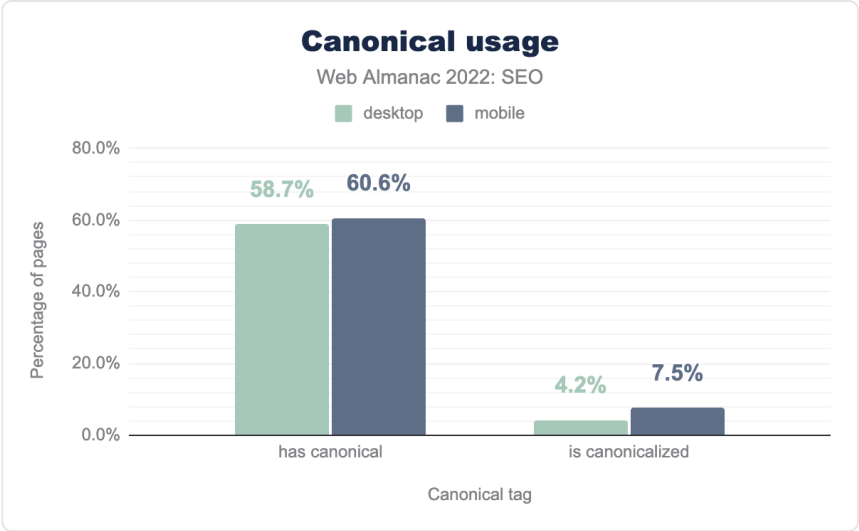



*図10.8. 正規の使用法。*

データから、正規タグの使用率が年々増加していることがわかります。2019年には、モバイルページの48.3%が正規を使用していました。2020年には53.6%に増加。2021年には58.5%とさらに増加。そして2022年には60.6%に増加しています。

モバイルはデスクトップよりも正規のアトリビューションの割合が高く（60.6%対58.7%）、これはモバイルでのURLの単一使用の直接的な結果と考えられます。この章のデータセットはホームページに限定されているため、これがモバイルの正規アトリビューションが高い理由であると考えるのが妥当でしょう。Googleのガイドライン[329]によると、独立したモバイルサイトを持つことは推奨されていません。

## HTMLとHTTPの正規表現

正規化タグの実装には2つの方法があります。

1. HTMLの `<head>` 内
2. HTTPヘッダー内（`Link` HTTPヘッダー）

329. *https://developers.google.com/search/mobile-sites/mobile-seo/separate-urls*

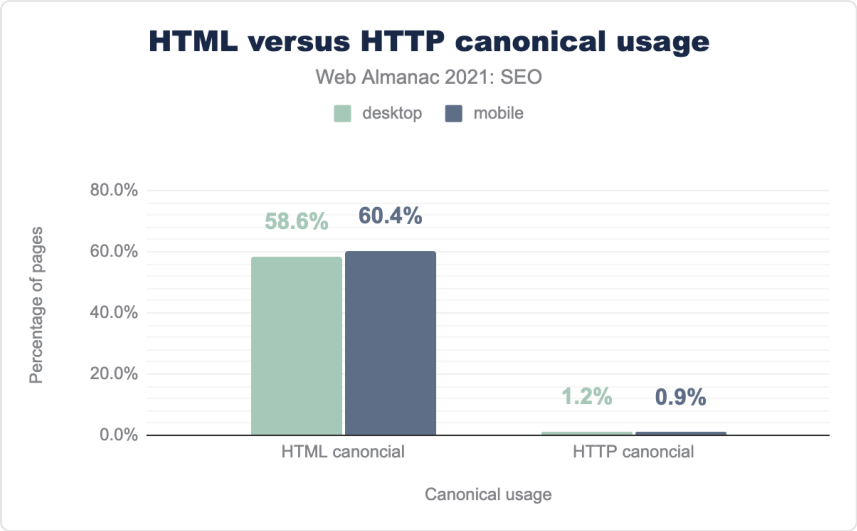



*図10.9. HTMLとHTTPの正規化の使い分け。*

デスクトップ、モバイルともにもっとも一般的な使用法はHTMLで、それぞれ58.6%、60.4%。これは、実装が簡単なためと思われます。一方は基本的なHTMLの知識が必要ですが、もう一方（HTTPヘッダーを使用）はより技術的なスキルが必要です。

### 未加工とレンダリングの比較

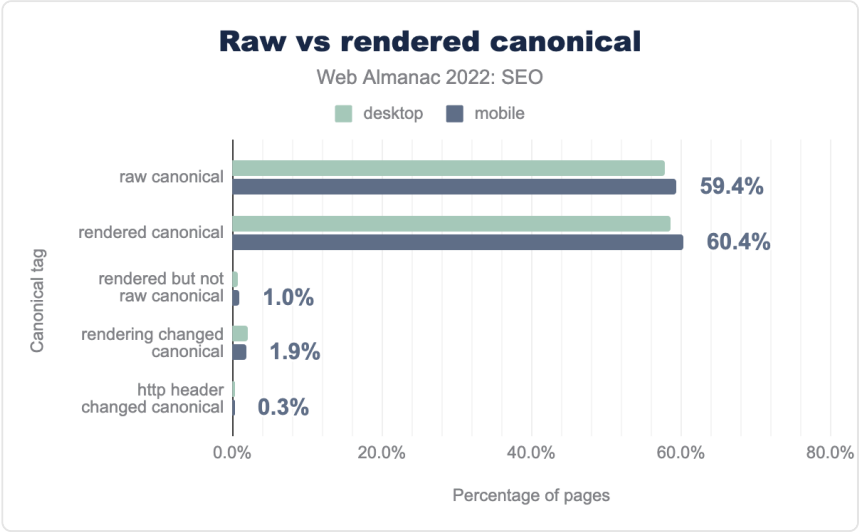



*図10.10. 未加工とレンダリング基準*

2021年の未加工のcanonical使用率は57.7%、レンダリングされたcanonical使用率は58.4%であったのに対し、2022年には若干の成長が見られ、未加工のcanonical使用率は59.4%、レンダリングされたcanonical使用率は60.4%に上昇しました。これは、カノニカルの使用率全体の伸びと相関しています。

## ページ体験

この章では、ページ体験のさまざまな要素と、2021年版Web Almanac以降の進化について見ていきます。

### HTTPS

2021年、Googleがコアウェブバイタルアップデートの序章を発表した後、サイトスピードとページ体験への注目が高まりました。HTTPSがランキング要因であるという体験は2014年[330]までさかのぼりますが、コアウェブバイタルの発表以降、ページ体験への全体的なフォーカスがウェブ全体のHTTPSの採用に影響を与えたと考えられます。

330. *https://developers.google.com/search/blog/2014/08/https-as-ranking-signal*

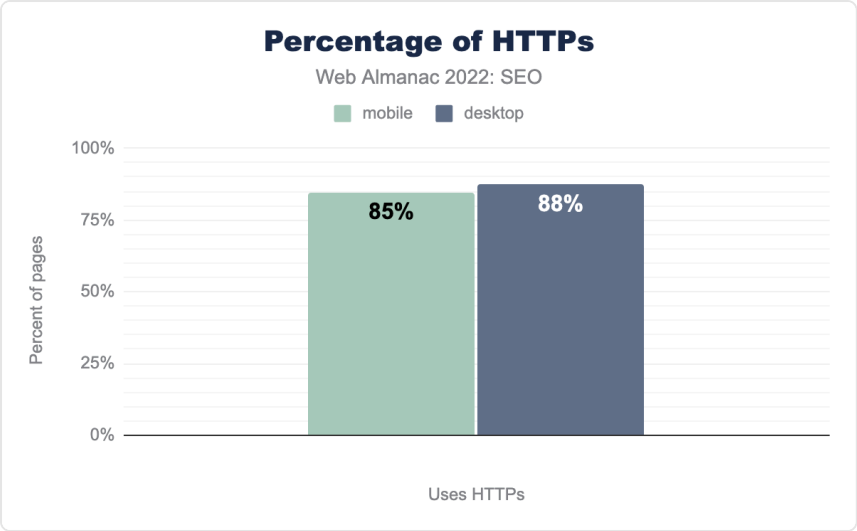



*図10.11. HTTPSをサポートしているウェブサイトの割合。*

データから、より多くのサイトがクロール時に安全な証明書（HTTPS）を使用していることがわかります（これらの証明書の有効期限を考慮）。2021年にはデスクトップページの84.3%がHTTPを使用していましたが、2022年には87.71%に上昇しました。モバイルでは、2021年の81.2%から2022年には84.75%に増加しました。2020年のコアウェブバイタル更新の発表から現在まで、モバイルでは約11%、デスクトップでは10%増加しています。

## モバイルフレンドリー

モバイルフレンドリーは、レスポンシブデザインの実装とダイナミックサービングを比較することで判断できます。これを特定するために、レスポンシブデザインで一般的に使用されるviewportメタタグの使用と、variable： User-Agentヘッダーの使用状況を調べ、ウェブサイトがダイナミックサービングを使用しているかどうかを判断しました。

## ビューポートメタタグ

**92%**

*図10.12. viewport metaタグをサポートしているサイト。*

2021年にはモバイルページの91.1%がviewport metaタグを使用していましたが、現在では

92%に増加しています。2020年には89.2%でした。

## Varyヘッダーの使い方

varyヘッダーは、異なるデバイス上の異なるユーザーに異なるコンテンツを提供できるようにするHTTPヘッダーです。これは動的な提供として知られており、まったく同じコンテンツを異なるデバイスに提供するレスポンシブデザインとは正反対です。

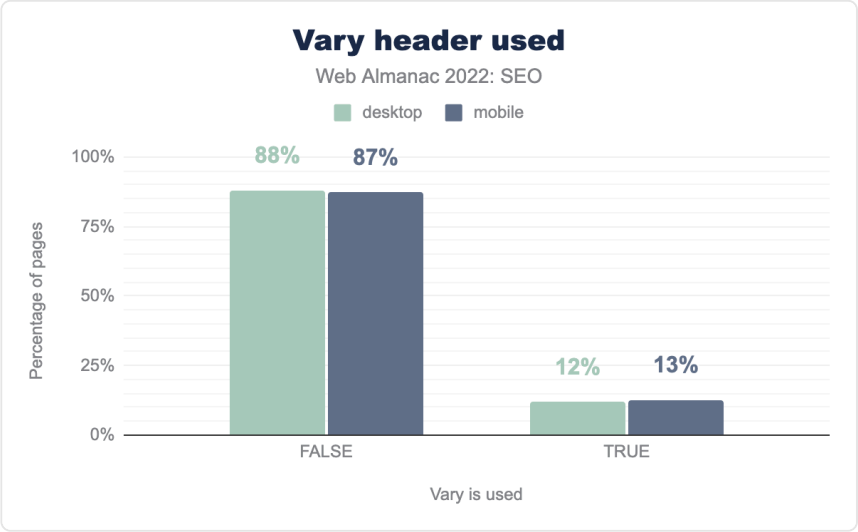



*図10.13. Varyヘッダーの使い方*

Varyヘッダーの使用率は、ここ数年比較的変化していません。2021年には、デスクトップページの12.6%、モバイルページの13.4%がこのフットプリントを使用していました。2022年のデータもほぼ同じで、デスクトップが12%、モバイルが13%です。

## 読みやすいフォントサイズ

2021年には、モバイルページの13.5%が読みやすいフォントサイズを使用していませんでした。Googleがすべてのデバイスでユーザー体験を重視するようになったおかげで、以前よりも多くのページで読みやすいフォントサイズが使用されるようになりました。読みやすいフォントサイズを使用していないモバイルページはまだ11%に過ぎません。


*図10.14. 読みやすいフォントサイズを使用していないサイト。*

## コアウェブバイタル（CWV）

コアウェブバイタルは、Googleが同年6月にページ体験アップデートの実施を発表した後、2021年を通してSEOのホットトピックでした。今年も引き続き関心が寄せられ、CWVのパフォーマンスに注目するサイトが増えています。

コアウェブバイタルは、開発者やSEO担当者がユーザーがページをどのように体験しているかをより理解するのに役立つ、標準化された一連のメトリクスです。主な指標は以下のとおりです。

- 最大のコンテントフルペイント (LCP)は、ウェブページのメインコンテンツの読み込み速度を測定します。
- 最初の入力までの遅延 (FID)は、ユーザーがウェブページとインタラクション（ボタンをクリックするなど）してから、ブラウザが応答できるようになるまでの時間を測定します。
- 累積レイアウトシフト（CLS）は、視覚的な安定性と、ページがビューポート内で移動するかどうかを測定します。

これら3つの指標はすべて、ユーザー体験とウェブページの安定性にとって非常に重要です。

Core Web Vitalsのデータは、Chromeユーザー体験レポート（CrUX）から得ています。このレポートは、実際の（オプトインした）ユーザーの公開データセットから作成されており、数百万のウェブサイトから取得されています（ラボのデータとは異なり、シミュレーションされたものです）。

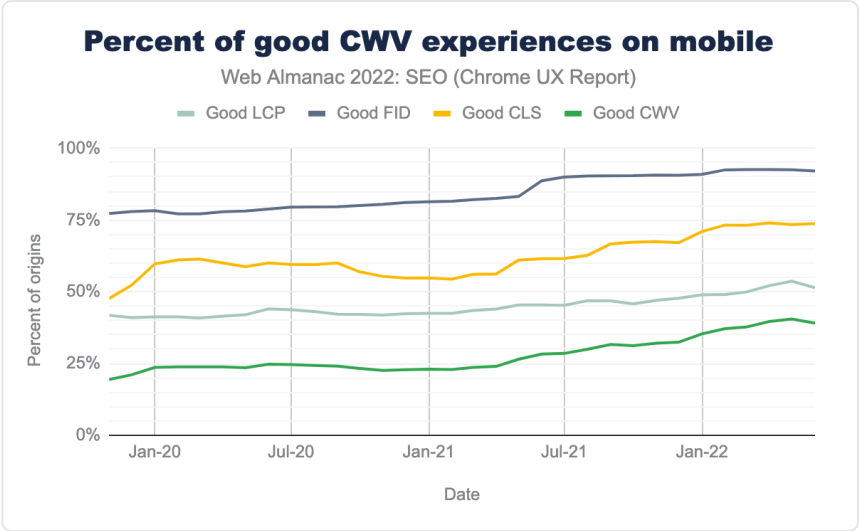



*図10.15. モバイルで良いCWV体験をした割合。*

モバイルでは、現在39%のサイトがCWVに合格しており、2021年の29%、2020年のわずか20%から上昇しています。また、現在92%のサイトがFIDに合格していますが、もっとも多くのサイトオーナーはLCPに苦戦しており、合格率は51%です。

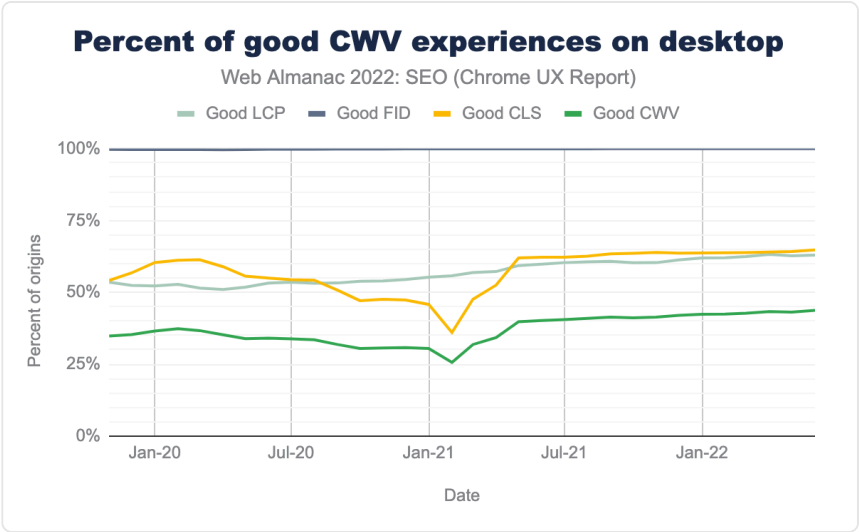



*図10.16. デスクトップでCWVを体験した人の割合。*

デスクトップでは、驚異的な100%のサイトがFIDに合格していますが、同様にLCPとCLSの合格には苦戦しています。注目すべきは、モバイル（39%）よりもデスクトップ（43%）の方がCWVに合格しているサイトが多いことです。

## `lazy`ローディングと`eager`ローディングのiframeの比較

レイジーローディングは、ウェブページ上の重要でない要素の読み込みを、必要な時点まで延期する技術です。これにより、ページが軽量化され、帯域幅やシステムリソースを節約できます。イージーローディングとは、関連するエンティティが同時にロードされ、一度にフェッチされることです。

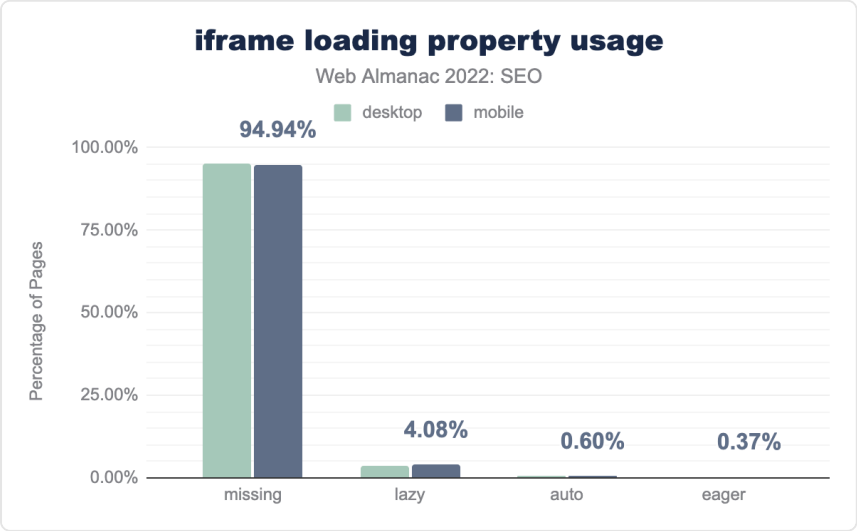



*図10.17. `iframe`ローディングプロパティの使用法。*

iframeだけを見ると、遅延ロードの方がイージーロードよりもはるかに好まれ、iframeの4.08%が遅延ロードであるのに対し、イージーロードは0.37%でした。

iframeのブラウザレベルの遅延読み込みがChrome[331]で標準化されたので、これはとくに興味深いことです。lazyやeagerを指定せずに`loading`属性を標準化したことが、94.4%の属性がlazyやeagerを含んでいないというデータを示している理由でしょう。

331. https://web.dev/iframe-lazy-loading/

# ページ上

関連性のシグナルを探すとき、検索エンジンはウェブページのコンテンツを見ます。SERP（検索エンジンの検索結果ページ）での順位や表示に影響を与えるさまざまなオンページSEOの要素があります。

## メタデータ

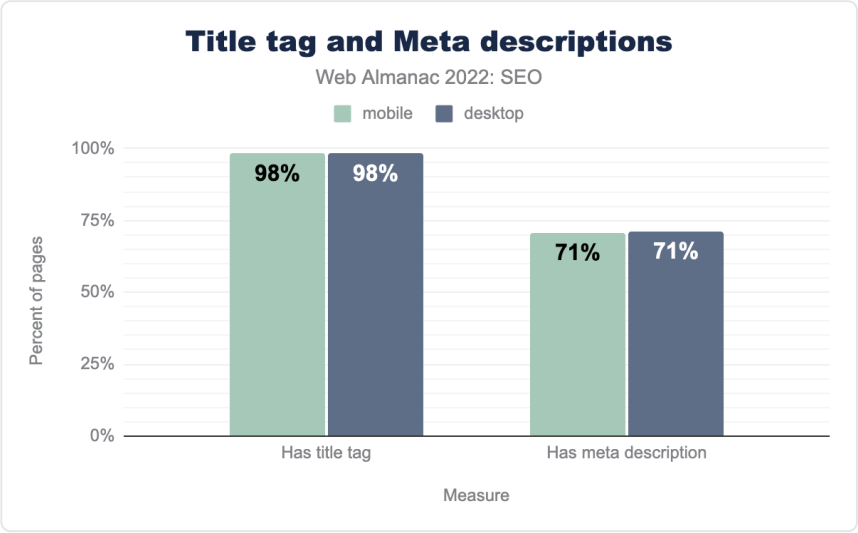



*図10.18. タイトルタグとメタディスクリプション*

2年連続で、デスクトップとモバイルページの98.8%に`<title>`要素がありました。また、2022年には、デスクトップとモバイルのホームページの71%に`<meta name="description">`タグがあり、昨年より0.1%減少しました。

### `<title>`要素

`<title>`要素は、ページの関連性に関する強いヒントを提供し、SERPに表示される可能性があるページ上のランキング要素です。2021年8月、Googleは検索結果でより多くのウェブサイトのタイトルを書き換え始めました[332]。

332. *https://developers.google.com/search/blog/2021/08/update-to-generating-page-titles*

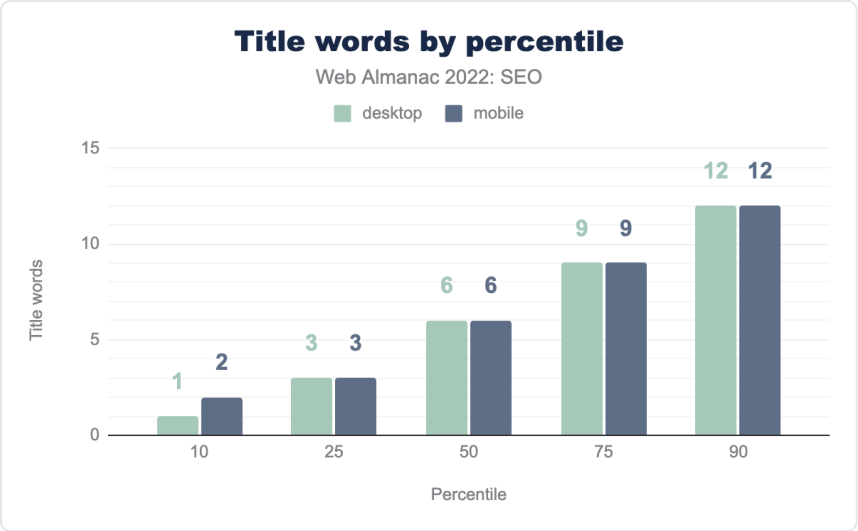



*図10.19. パーセンタイル別のタイトル語。*

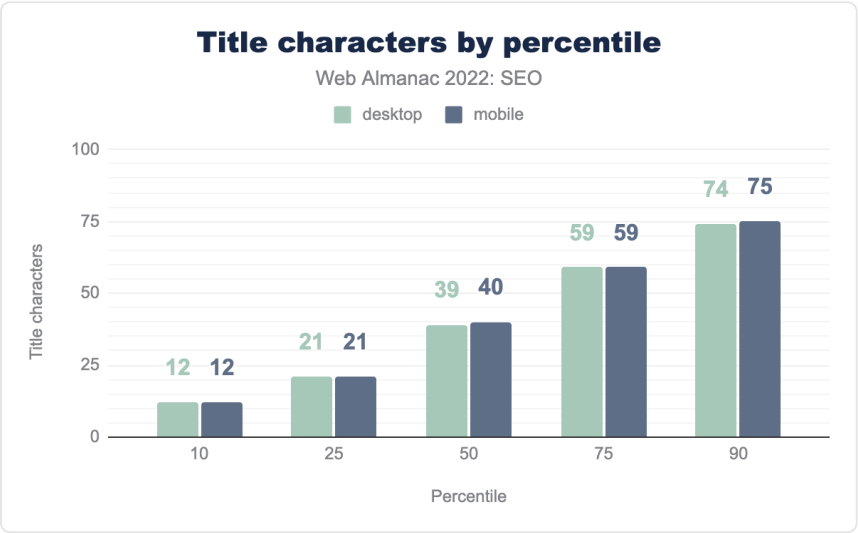



*図10.20. パーセンタイル別のタイトル文字。*

2022年に

- `<title>`の中央値は6語。
- ページの`<title>`の中央値は、デスクトップで39文字、モバイルで40文字でし

た。

- ページの10%に12語を含む `<title>` 要素がありました。
- デスクトップとモバイルのページの10%に、それぞれ74文字と75文字を含む `<title>` 要素がありました。

これらの統計は昨年と変わっていません。注：ホームページのこれらのタイトルは、深いページで使用されるものよりも短い傾向があります。

### メタディスクリプションタグ

`<meta name="description">` タグはランキングに直接影響しません。しかし、SERP上にページの説明として表示されることがあり、クリック率に影響を与えることがあります。

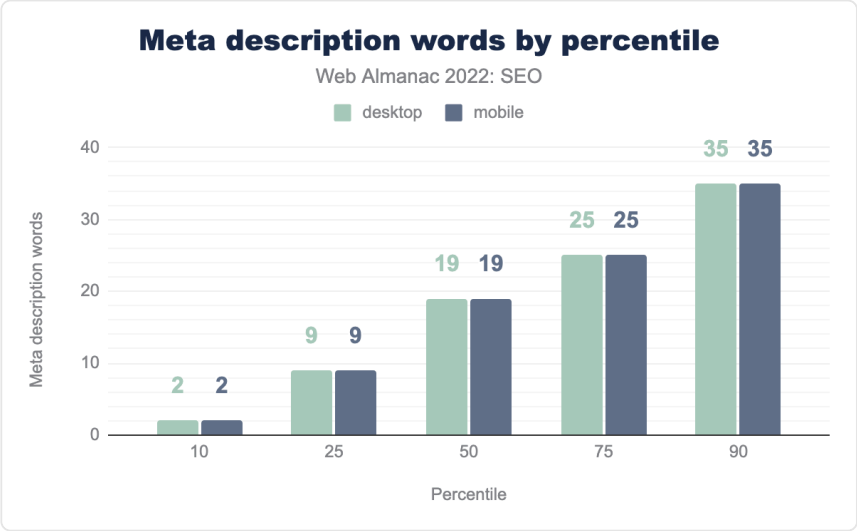



*図10.21. パーセンタイル別のメタディスクリプションワード*

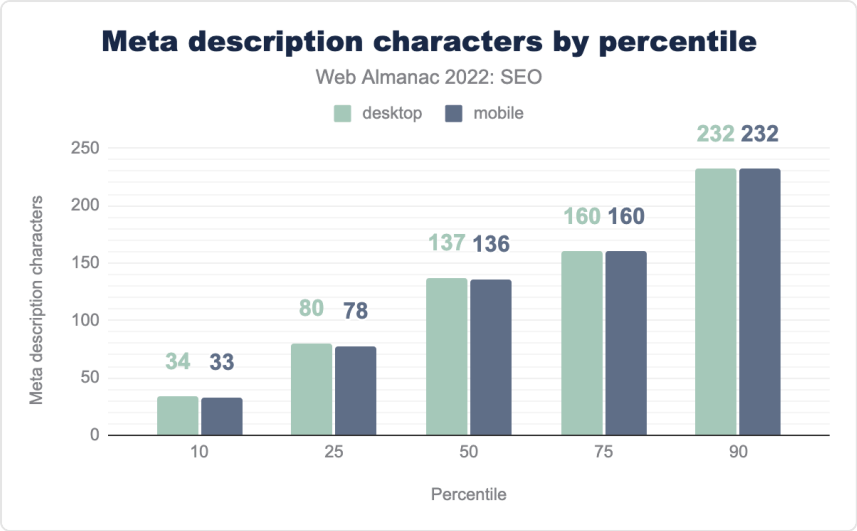



*図10.22. パーセンタイル別のメタディスクリプション文字数*

2022年に

- デスクトップとモバイルページの `<meta name="description">` タグに含まれる単語の中央値は19語でした。
- デスクトップページとモバイルページの `<meta name="description">` タグに含まれる文字数の中央値は、それぞれ137文字と136文字でした。
- デスクトップとモバイルのページの10%に、35語を含む `<meta name="description">` タグがありました。
- デスクトップとモバイルのページの10%に、232文字を含む `<meta name="description">` タグがありました。

もっとも、これらのスタッツは昨年と比較的変わりません。

### ヘッダータグ

見出し要素（`<h1>`、`<h2>`...）は、ページ上のコンテンツを整理するのに役立つため、ページ構造の重要な部分です。見出し要素は直接的なランキング要因ではありませんが、Googleがページ上のコンテンツをより理解するのに役立ちます。

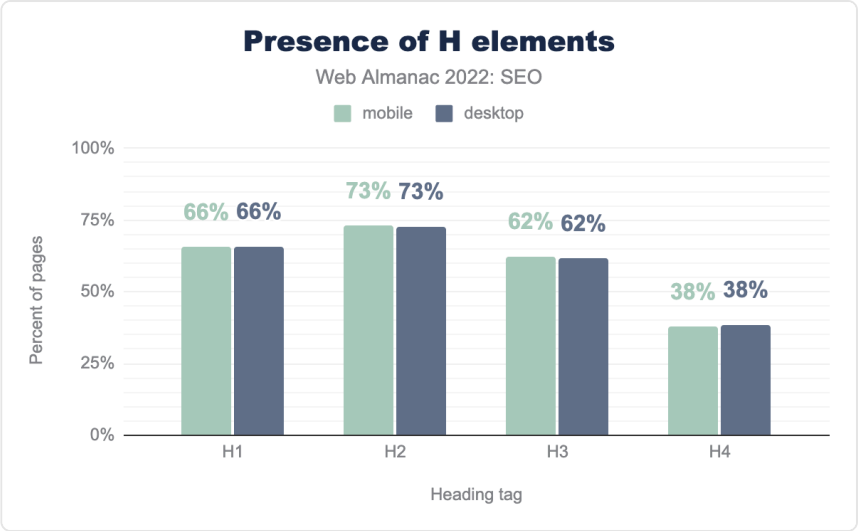



*図10.23. H元素の存在。*

2022年のタイプ別見出しの実装に関する傾向は、わずかな違いはあるものの、2021年とほぼ一致しています。たとえば、2021年にはモバイルページの71.9%がh2を使用していましたが、2022年には73.02%がh2を使用しています。

もう1つの傾向は、h1とh2の使い方の違いです。デスクトップページの72.7%がh2を使用しているのに対し、h1を使用しているのは65.8%のみです（モバイルでも同様の数字が反映されています）。

これについての明確な説明はありませんが、考えられる理由のひとつは、h1がどのコンテンツよりも上に配置されることが多いからです。h1はコンテンツの自然な流れにとって必須ではありません。しかし、h2がないと、構造化されていないコンテンツが長く続く可能性があります。

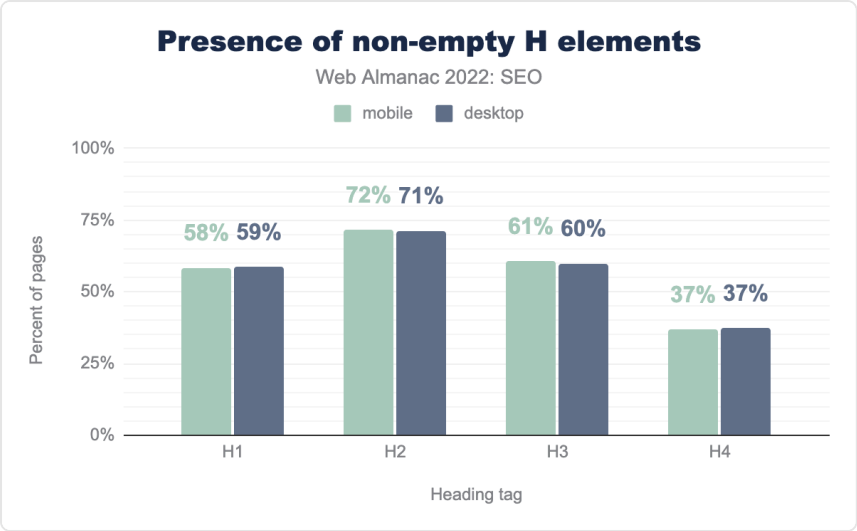



*図10.24. 空でないH要素の存在。*

全体的に、2021年の統計と同様に、ページ上に空のH要素は比較的少ないことがわかります。さらに、デスクトップとモバイルのデータにはほとんど食い違いがありません。

しかし、h1については乖離があります。 65.8%のページがh1要素を含んでいたのに対し、58.5%は空ではないh1要素を含んでいました。これは7.3ポイントの差です。h2との差はわずか1.5ポイントです。 2021 Web Almanacにあるように、これはホームページのh1要素にロゴ画像を挿入している多くのウェブサイトの結果かもしれません。

## イメージ属性

`<img>`要素のalt属性の主な目的はアクセシビリティです。alt属性はまた、検索エンジンが画像検索でとくにアセットをランク付けするのを助けます。

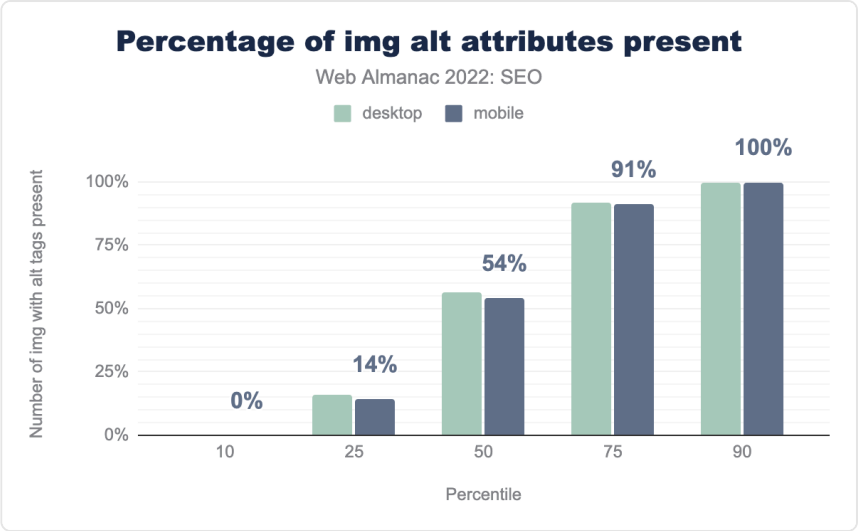



*図10.25.* `img` `alt` *属性が存在する割合。*

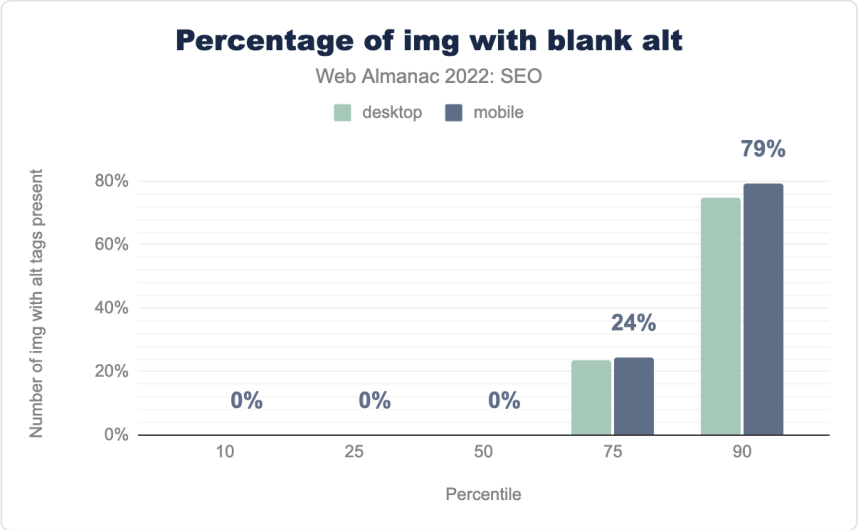



*図10.26. 空白の* `alt` *を持つ* `img` *の割合。*

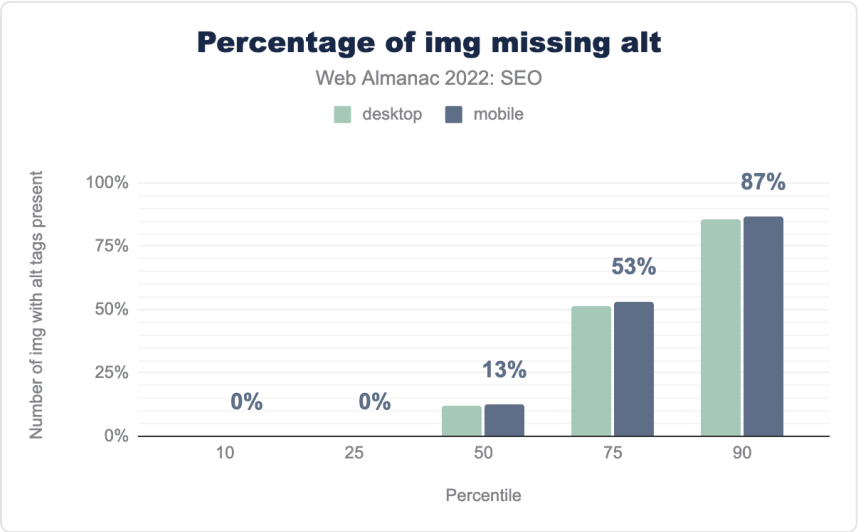



*図10.27. `alt` がない `img` の割合。*

発見したこと

- デスクトップページの中央値では、`<img>` タグの56.25%がalt属性を持っています。これは、2021年の56.5%からわずか4分の1ポイントのごくわずかな減少です。
- モバイルページの中央値では、`<img>` タグの54.9%がalt属性を持ちます。これは、2021年のalt属性を持つタグの54.6%からわずかな増加です。
- alt属性が空白の `<img>` タグを含むデスクトップとモバイルページの中央値は、2021年と比較して顕著な変化が見られます。昨年、デスクトップ ページとモバイル ページの中央値には、それぞれ10.5%と11.8%の `<img>` タグと空白の `alt` 属性が含まれていました。2022年には、この数字はデスクトップで12.1%、モバイルで12.5%に上昇。
- alt属性がない `<img>` タグを含むデスクトップとモバイルページの中央値は0%という傾向が続いています。2021年のデスクトップページの中央値では、`<img>` タグの1.4%が空白属性でした。2022年には0%に減少しました。

## 画像 `loading` プロパティの使用法

ユーザーエージェントが画像のレンダリングと表示をどのように優先するかは、`<img>` 要

素に適用されるloading属性によって影響を受けます。この実装はユーザー体験とパフォーマンス時間に影響を与え、SEOの成功とコンバージョンの両方に影響を与える可能性があります。

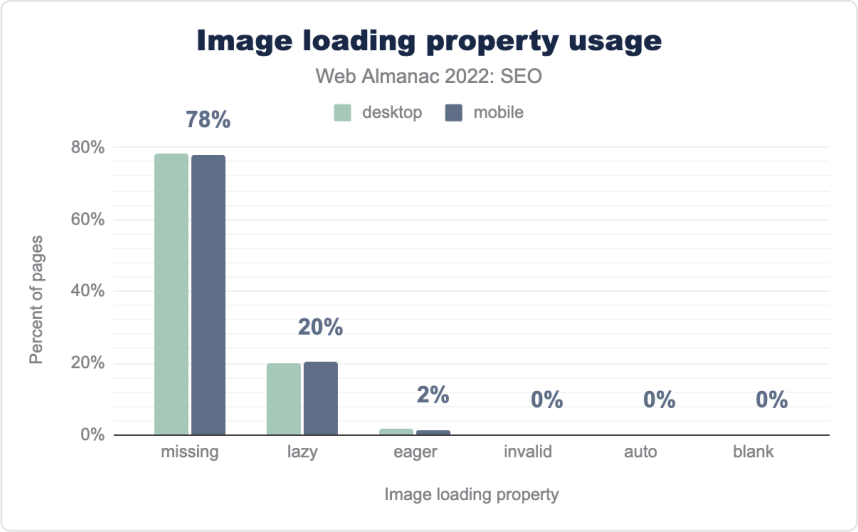



*図10.28. 画像 `loading` プロパティの使用法。*

発見したこと

- 画像読み込みプロパティを使用しないページが大幅に減少しています。2021年には、デスクトップページの83.3%、モバイルページの83.5%が画像読み込みプロパティをまったく利用していませんでした。2022年には、デスクトップページの78.3%、モバイルページの77.9%となっています。
- 逆にloading="lazy"の実装は増加しています。2021年には、デスクトップとモバイルページの15.6%がloading="lazy"を実装していました。2022年には19.8%（デスクトップ）、20.3%（モバイル）に増加しています。
- ブラウザの読み込み方法をデフォルトにするページ数は2022年に減少しています。デスクトップでは0.07%、モバイルでは0.08%のページがloading="auto"を利用しています。2021年には0.01%のページがloading="auto"を利用していました。

## 単語数

コンテンツの長さはランキング要因ではありませんが、ページの平均単語数を評価すること

は価値があります。

### レンダリング語数

まず、レンダリングされたページに含まれる単語の数から見てみましょう。

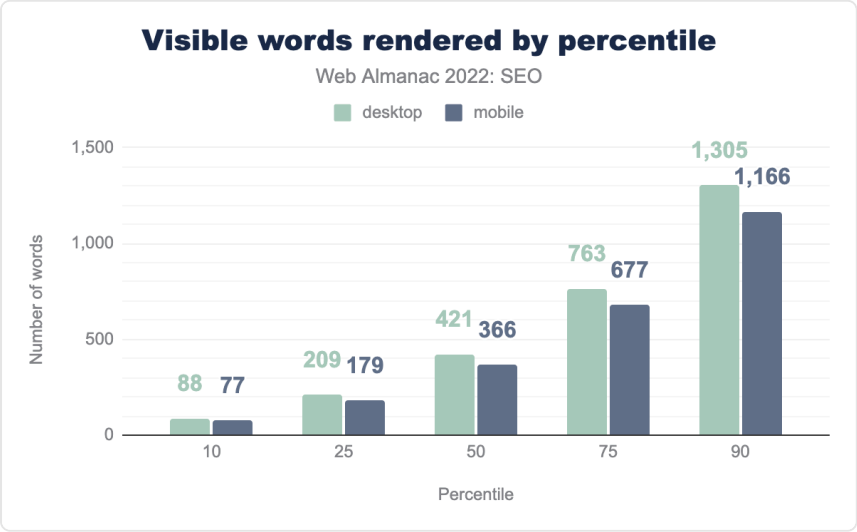



*図10.29. パーセンタイル別に表示される単語。*

2022年のデスクトップ・ページの中央値は421語。これは2021年の425語に近い数字です。しかし、これは2020年のデスクトップページの中央値402ワードと比べると、パーセンテージ的には大きな飛躍です。2021年にレンダリング語数が増加した原因が何であれ、それは2022年まで続いているようです。

同様に、2022年のモバイルのレンダリング語数の中央値は366語で、これも2021年のデータと同様の割合です。文脈上、デスクトップページの方がモバイルページよりも多くの単語が含まれています。デスクトップページの中央値は、50パーセンタイル内のモバイルページよりも15%多くの単語を含んでいます。Googleは数年前にモバイルファーストインデックスを採用しており、モバイル版ページにないコンテンツは検索エンジンにインデックスされないリスクがあるため、これは重要なことです。

### 未加工の単語数

ブラウザがJavaScriptコードを実行したり、DOMやCSSOMに変更を加えたりする前に、ページのソースコードに含まれる単語の数を調べてみましょう。

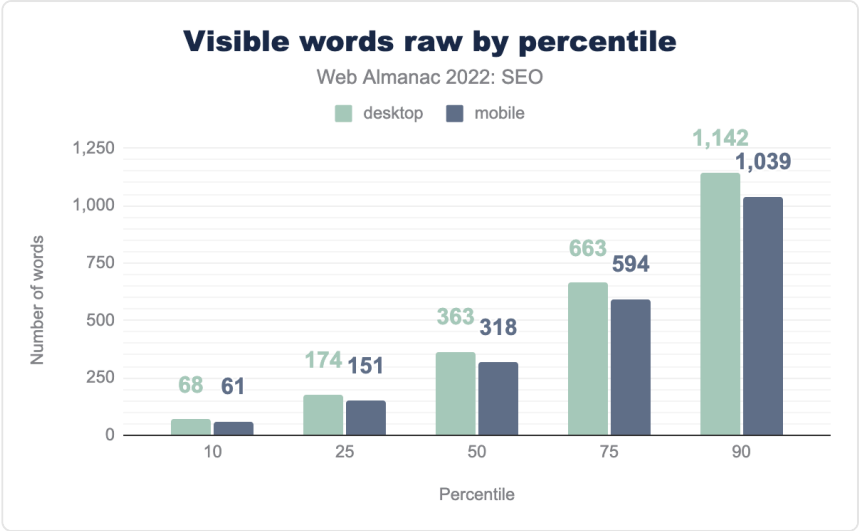



*図10.30. パーセンタイルごとに未加工のまま表示される単語。*

レンダリング語数と同様に、2022年のデータと2021年のデータにはわずかな違いがあります。たとえばデスクトップページの未加工ワード数の中央値は、2021年の369ワードに対し、2022年は363ワード、モバイルページの未加工ワード数の中央値は318ワードで、中央値が321ワードだった2021年をわずかに下回っています。

ここでも、モバイルページはデスクトップページよりも全体的に少ない単語数です。モバイルページのワード数の中央値は、デスクトップよりも12.39%少なくなっています。前述の通り、これはGoogleのモバイルファーストインデックスのために重要です。

## 構造化データ

構造化データの実装は、Google SERP上のリッチリザルトがより目立つようになったため、注目されるようになりました。

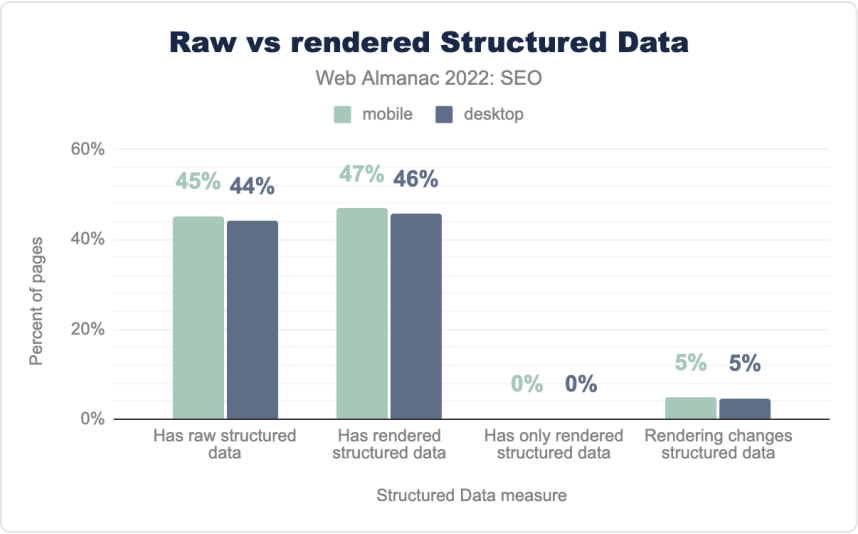



*図10.31. 未加工データとレンダリングされた構造化データ。*

ページのHTMLへの構造化データの実装は増加の一途をたどっています。2021年には、デスクトップページの42%、モバイルページの43%が構造化データを使用していました。2022年には、HTML内に構造化データを持つデスクトップページの44%、モバイルページの45%に上昇しています。

これは、デスクトップとモバイルページの両方で2ポイントの増加を反映しています。多くのコンテンツ管理システムがページに自動構造化データマークアップを追加したことと、前述のように構造化データがGoogle SERPsで重要な役割を果たすようになったことの2つが、採用増加の理由として考えられます。

また、モバイルページとデスクトップページの両方で、最初のHTMLレスポンスに含まれていないJavaScript経由で構造化データが追加されているページが大幅に減少しています。2021年には、モバイルページの1.7%、デスクトップページの1.4%が、最初のHTMLレスポンスに含まれていないJavaScript経由で構造化データを追加していました。現在はデスクトップでわずか0.15%、モバイルで0.13%です。

## もっとも普及している構造化データ形式

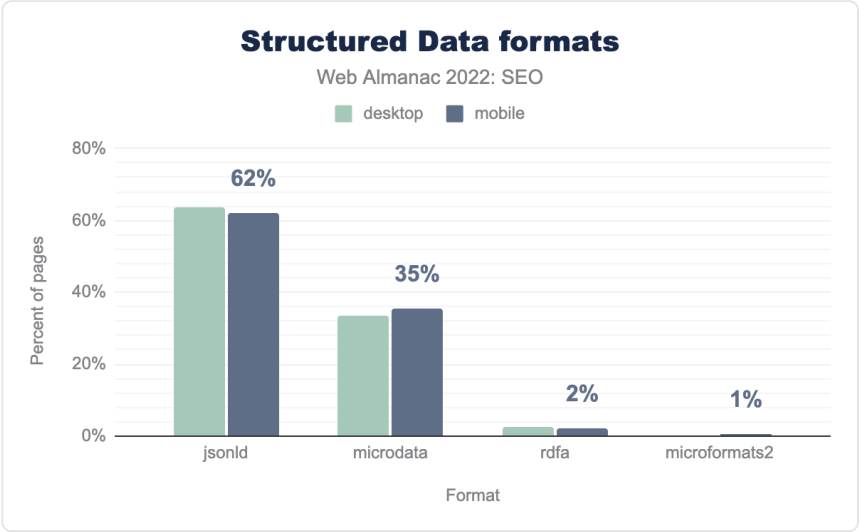



*図10.32. 構造化データ形式。*

構造化データは、ページ上にさまざまな方法で実装できます。しかし、JSON-LDは、Googleが推奨する実装方法と一致しており、もっとも普及しています。

2021年の数字と比較すると、2022年のデータでは、JSON-LDによる実装は名目上増加し、マイクロデータによる構造化データの実装はわずかに減少しています。この数字は、とくにモバイルで顕著です。2021年には、モバイルページの60.5%が構造化データの実装にJSON-LDを使用していました。2022年に構造化データの追加にJSON-LDを使用したモバイルページの数は、2.3%増の61.9%です。逆に、2021年には、36.9%のモバイルページがマイクロデータによる構造化データを利用していました。2022年には4.3%減少し、35.3%になりました。

## もっとも普及しているスキーマタイプ

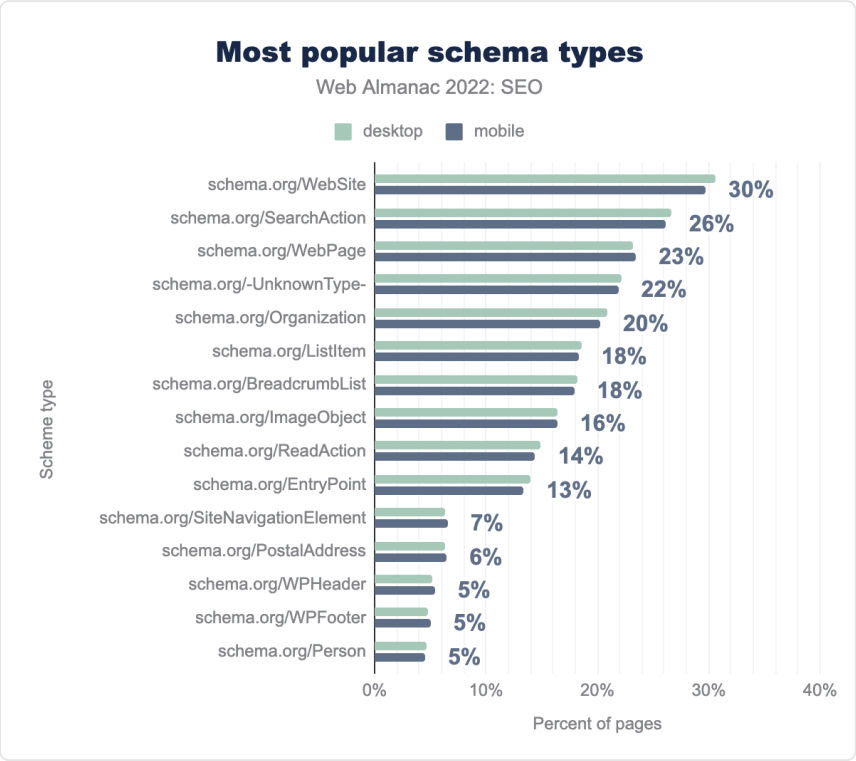



*図10.33. もっとも普及しているスキーマタイプ。*

2021年と2022年のホームページでもっとも普及しているスキーマの種類には強い相関関係があります。

Web Almanacの以前の版で述べたように、`WebSite`, `SearchAction`, `WebPage`, `SearchAction`は、サイトリンク検索ボックス[333][上の図を参照]を動かすものです。

2021年と2022年を比較すると、もっとも普及しているスキーマが軒並み大幅に増加しています。実際、2022年には、注目されるすべてのスキーマタイプで採用が増加しています。もっとも注目すべきは、2021年から22.8%上昇したBreadcrumbsListのスキーマと、12.3%上昇したImageObjectのスキーマです。

もっとも普及しているスキーマの実装という点では、モバイルページとデスクトップページの割合の差は比較的小さい。

333. *https://developers.google.com/search/docs/advanced/structured-data/sitelinks-searchbox*

構造化データについて詳しくは、構造化データの章をご覧ください。

## リンク

検索エンジンはリンクを利用して新しいページを発見し、ページの重要性を判断するPageRankを渡します。リンクはまた、あるページから別の（おそらく関連性のある）ページへの参照としても機能します。

### 説明的でないリンクテキスト

アンカーテキストとは、リンクに使用されるクリック可能なテキストのことで、検索エンジンがリンク先のページの内容を理解するのに役立ちます。ライトハウスでは、使用されているアンカーテキストが有用であるか、文脈に沿ったものであるか、または“詳しくはこちら“や“ここをクリック“のような一般的で非記述的なものかをチェックするテストを実施しています。2022年のテストでは、モバイルとデスクトップでそれぞれ15%と17%のリンクに説明的なアンカーテキストがありませんでした。

### 発信リンク

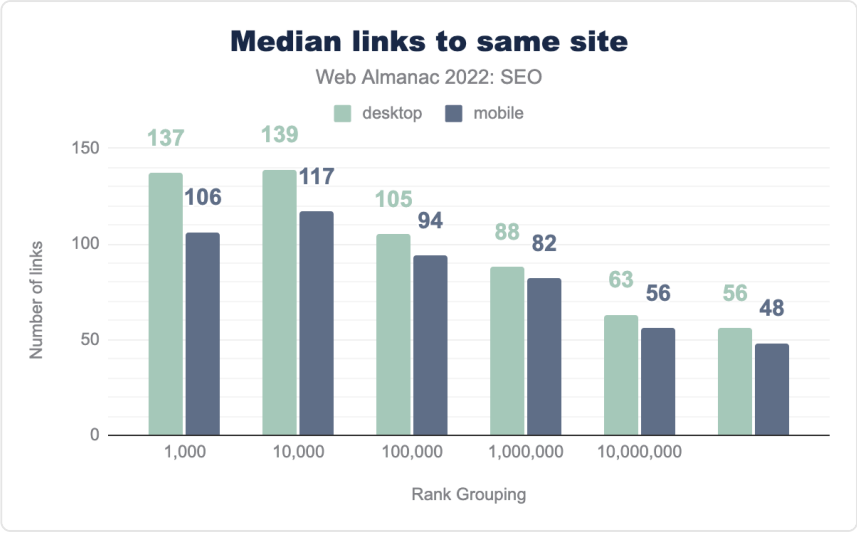



*図10.34. 同じサイトへの中央リンク*

内部リンクとは、同じウェブサイトの他のページへのリンクのことです。昨年と同様、

2022年の数字も、デスクトップ版に比べてモバイル版のリンク数が少ないことを示唆しています。

内部リンクの数の中央値は、デスクトップが56%、モバイルが48%と、モバイルよりも16%高くなっています。これは、開発者が小さな画面でも使いやすいように、モバイルのナビゲーションメニューやフッターを最小限に抑えた結果と考えられます。

CrUXのデータによると、もっとも普及している1,000のウェブサイトは、そうでないサイトよりも発信内部リンクが多く、デスクトップでは合計137本、モバイルでは106本でした。これは中央値の2倍以上です。これは、一般的にページ数の多い大規模サイトでメガメニューが使用されていることが原因と考えられます。

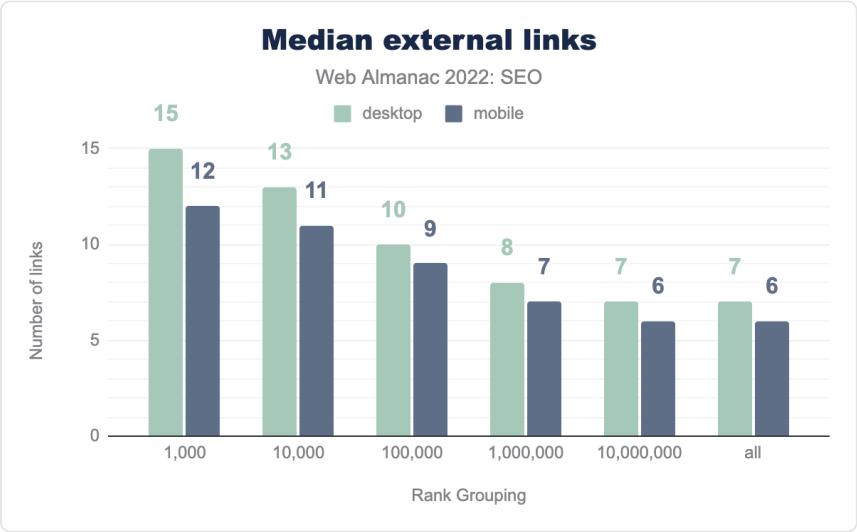



*図10.35. 中央の外部リンク*

外部リンクとは、別のウェブサイトの他のページへのリンクのことです。ここ数年一貫しているデータですが、デスクトップ版に比べ、モバイル版のページには外部リンクが少ないことが指摘されています。Googleは数年前にモバイルファーストインデックスを導入したにもかかわらず、ウェブサイトはモバイル版をデスクトップ版と同等にはしていません。

## アンカーrel属性の使用

2019年9月、Googleはパブリッシャーがリンクを *sponsored* または *user-generated content* に分類できる属性を導入しました[334]。これらの属性は、以前2005年に導入された[335]

334. *https://webmasters.googleblog.com/2019/09/evolving-nofollow-new-ways-to-identify.html*
335. *https://googleblog.blogspot.com/2005/01/preventing-comment-spam.html*

`rel=nofollow` に加えています。新しい属性の `rel=ugc` と `rel=sponsored` は、リンクに追加情報を加えます。

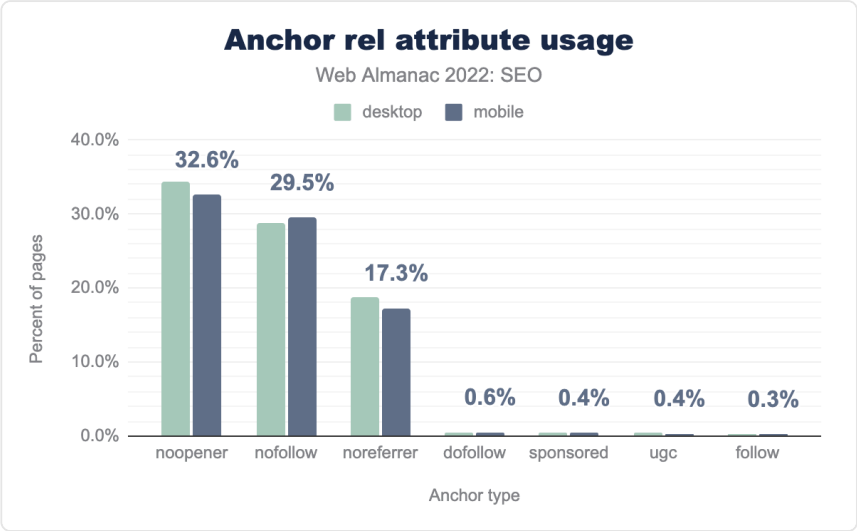



*図10.36. アコー `rel` 属性の使用法。*

新しい属性の採用という点ではあまり変化はなく、2022年には `rel=ugc` がデスクトップとモバイルページの0.4%、`rel=sponsored` がデスクトップの0.5%、モバイルページの0.4%に表示されます。

`rel="dofollow"` は、再び `rel="ugc"` や `rel="sponsored"` よりも多くのページに表示されました。これは技術的には問題ではありませんが、Googleは `rel="follow"` と `rel="dofollow"` を無視します。

実際の属性である `rel="nofollow"` は、2022年にはモバイルページの29.5%で見つかり、昨年より1.2%減少しました。Googleは `nofollow` をヒントとして扱い、検索エンジンがその属性を尊重するかどうかを選択できることを意味します。

## AMP

AMPは2015年のローンチ以来、ランキングに直接的な影響があるかどうかがSEO関係者の間で議論され、物議を醸してきました。その後、Googleはさらなる明確化のため、ドキュメントの中でこのような声明（下記）を発表しました。

> *AMP自体はランキング要因ではありませんが、スピードはGoogle検索のランキング要因です。Google検索は、ページを構築するために使用された技術に関係なく、すべてのページに同じ基準を適用します。*
>
> *– Google検索セントラル*[336]

Core Web Vitalsのローンチ以来、AMPの将来は変化しているようです。以前AMPを導入した主な理由は、ページ速度の改善以外に、トップカルーセルに掲載するために必要だったからです。2021年、Googleはその要件を更新し、AMPの有無にかかわらず、どのページもトップカルーセルに掲載される資格があると概説しました。

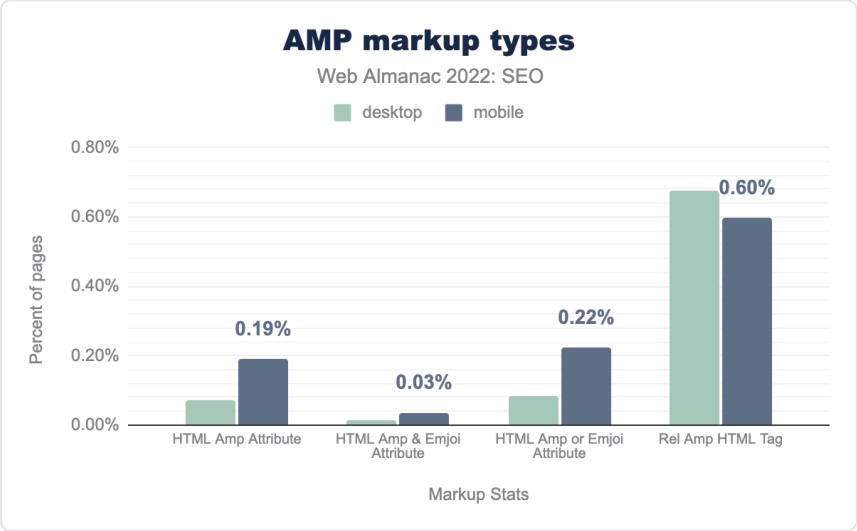



*図10.37. AMPマークアップタイプ。*

デスクトップの利用率は2021年と比較して2022年には0.09%から0.07%に低下し、モバイルの利用率は同じ期間に0.22%から0.19%に低下しています。

## 国際化

SEOにおける国際化とは、検索エンジンに適切にクロールされ、インデックスされるように、複数の国や複数の言語をターゲットとしたベストプラクティスに沿ってウェブサイトを最適化するプロセスのことです。

336. *https://developers.google.com/search/docs/advanced/experience/about-amp*

## Hreflang の使用法

Hreflangタグは、GoogleやBingやYandexなどの検索エンジンが、指定されたページの主要言語を理解するのに役立ちます。これは、主に国際的なSEOキャンペーンで、ウェブサイトの異なるバージョン間で複数の異なる言語が使用されている場合に使用されます。

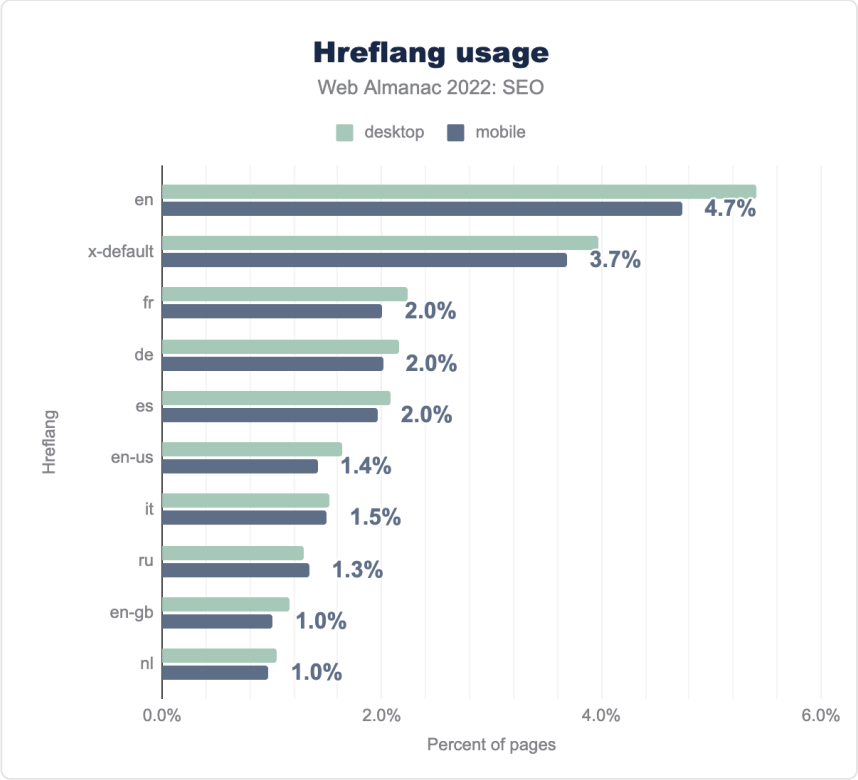



*図10.38. Hreflangの使い方。*

現在、hreflangタグをデスクトップで使用しているサイトは9.6%、モバイルで使用しているサイトは8.9%です。これは、9.0%のサイトがデスクトップでhreflangsタグを使用し、8.4%のサイトがモバイルでhreflangsタグを実装していた2021年からわずかに増加しています。

2022年にもっとも普及しているhreflangタグは `en` [English]で、デスクトップで5.4%、モバイルで4.7%となっています。この割合は前年とほぼ同じです。

“フォールバック”バージョンであるx-defaultの次に（そして2番目によく採用される）、フランス語、ドイツ語、スペイン語のhreflangタグがもっともよく使われます。

hreflangタグを実装するには、`<head>`、リンクヘッダー、XMLサイトマップの3つの方法があります。注：このデータはホームページのみを見ているため、XMLサイトマップは含まれていません。

## コンテンツの言語使用

Googleはhreflangタグを使う傾向がありますが、Bingなどの他の検索エンジンは`content-language`属性を好みます。これは2つの方法で実装できます。

1. HTML
2. HTTPヘッダー

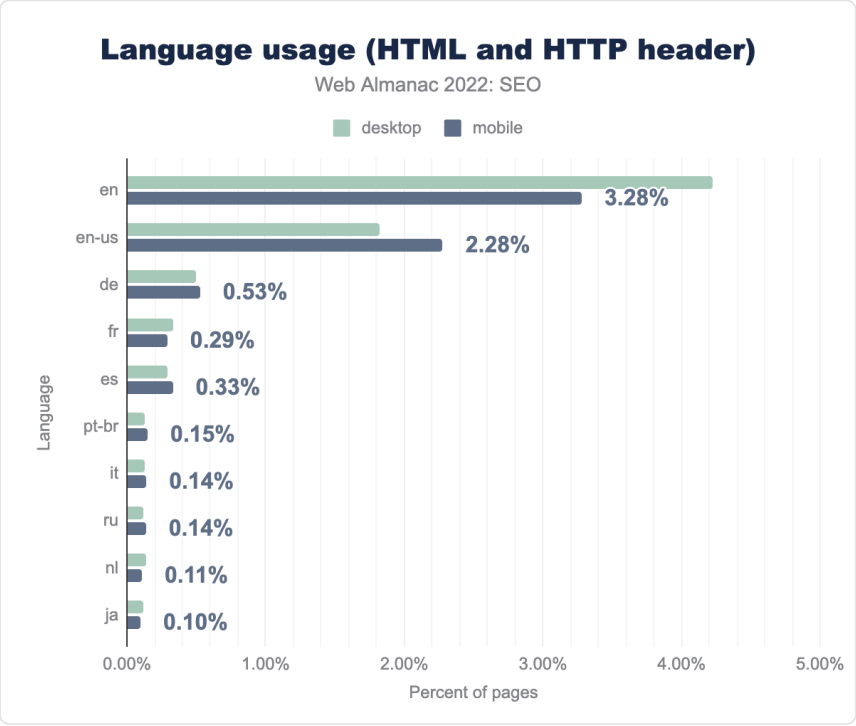



*図10.39. 言語使用（HTMLとHTTPヘッダー）。*

2022年、content-languageの実装方法としてもっとも普及しているのはHTTPサーバーレスポンスで、モバイルサイトの8.27%、デスクトップサイトの8.82%が利用しています。しかし、モバイルサイトの9.3%が使用していた2021年と比較すると、モバイルでの採用は減少しています。逆にデスクトップでは、8.7%のサイトが使用していた2021年と比較して、わずかに増加しています。

一方、HTMLは2022年にデスクトップで2.98%、モバイルで3.01%の採用率です。しかし、モバイルサイトの3.3%がHTMLタグを使用していた2021年と比較すると、やはりモバイルでの使用率は低下しています。

## 結論

2021年[337]、2020年[338]、2019年[339]のデータのパターンと同様に、分析したサイトの大半は、インデックス可能でクローラブルなページを持つことを含むSEOのさまざまな基本に関して、わずかではあるが一貫した改善を見せています。

また、コアウェブバイタルなどのパフォーマンス要素への注目も高まっており、アップデートが最初に発表された2020年にはわずか20%だったのに対し、現在では39%のサイトが合格点を獲得しています。これは、サイトがGoogleの指導をより心に留めていることを示しているようです。それでも、ウェブ全体でもっと取り組む必要があります。

indexifembeddedタグのような新しい序章は、ゆっくりとピックアップを見ています。このことは、ベストプラクティスの継続的な採用の必要性と、SEO、検索エンジンフレンドリー、そしてウェブ全般の状態における成長の機会がいかに大きいかを強調しています。

### 著者

Sophie Brannon

@SophieBrannon SophieBrannon

Sophieは英国を拠点とするエージェンシーAbsolute Digital Mediaのクライアント・サービス・ディレクターで、医療や金融など競争の激しい業界におけるSEO戦略とコンテンツ・マーケティングを専門としています。Sophieは、カンファレンスのスピーカーや業界ブロガーであり、地域、国、国際的な規模で受賞歴のあるキャンペーンを戦略化し、提供した実績があります。

337. https://almanac.httparchive.org/ja/2021/seo
338. https://almanac.httparchive.org/ja/2020/seo
339. https://almanac.httparchive.org/ja/2019/seo


## Itamar Blauer

@ItamarBlauer itamarblauer https://www.itamarblauer.com/

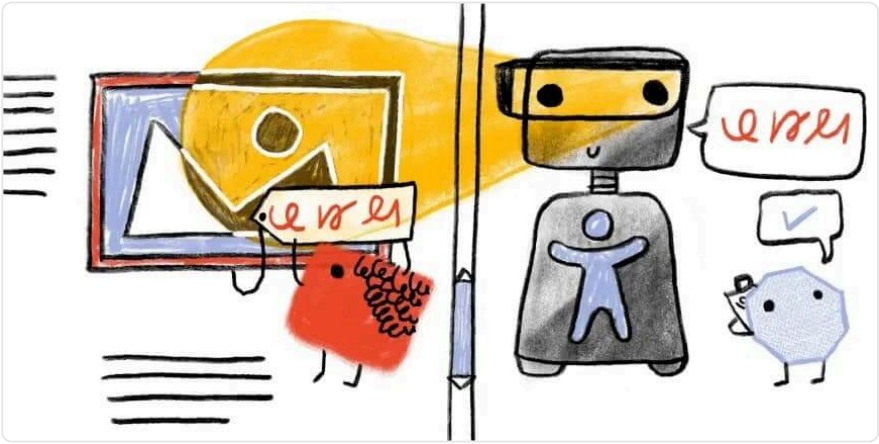

Itamar Blauerはロンドンを拠点とするSEOエキスパートです。UXを重視し、データに裏打ちされたクリエイティブなSEOでランキングを上げた実績があります。


## Mordy Oberstein

mordy-oberstein


モーディ・オーバースタインは、WixのSEOブランディング責任者です。また、SEMrushのコンサルタントを務め、SERP's Upポッドキャストを含む複数のSEOポッドキャストのマイクを握っています。

# 部 II 章 11
# アクセシビリティ



***Saptak Sengupta、 Thibaud Colas と Scott Davis によって書かれた。***
*Shaina Hantsis によってレビュ。*
*Thibaud Colas による分析。*
*Kirsty Simmonds 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

グローバルオンライン人口の27%がモバイルで音声検索を使用しています[340]。Facebookの動画の85%が音声をオフにしてクローズドキャプションをオンで視聴されています[341]。Siri、Alexa、Cortanaなどの音声アシスタントに質問すると、通常、パーソナルコンピューターが存在して以来のスクリーンリーダー技術[342]を使ってウェブページから答えを読み上げます。

ソフトウェアの機能が「アクセシビリティ機能」でなくなり、単に私たち全員が使用する「機能」となるのはいつですか？次にスマートフォンをサイレント/バイブレートモードにするとき、とくに聴覚障害/難聴コミュニティのメンバーでない場合、自分自身にその質問をしてみてください。

良いアクセシビリティは、障害を持つ人だけでなく、すべての人にとって有益です。これは

340. *https://www.gwi.com/hubfs/Downloads/Voice-Search-report.pdf*
341. *https://idearocketanimation.com/18761-facebook-video-captions/*
342. *https://www.theverge.com/23203911/screen-readers-history-blind-henter-curran-teh-nvda*

ユニバーサルデザイン[343]の核心原則の1つです。ティム・バーナーズ＝リーは「ウェブの力はその普遍性にあります。障害の有無にかかわらず、誰もがアクセスできることが本質的な側面です」と述べています。COVID-19パンデミックが始まって以来、ますます多くの人々がインターネットに依存するようになりました。同様に、アクセシビリティも改善する必要があります。そうでなければ、多くの人々を疎外するリスクがあります。

Lighthouseアクセシビリティ監査データの全体的なサイトスコアの中央値は、2020年の80%から2021年には82%へ、そして2022年には83%へと上昇しました。この増加が正しい方向へのシフトを表していることを願っています。

ウェブアクセシビリティの現状はまだ望ましいレベルには達していませんが、今年はサイトのアクセシビリティが全体的に改善されたことが見られました。Lighthouseアクセシビリティ監査データの全体的なサイトスコアの中央値は、2020年の80%から2021年には82%へ、そして2022年には83%へと上昇しました。Lighthouseの結果を監査ごとに見ることで、どのような具体的な改善が行われたかを把握できます。

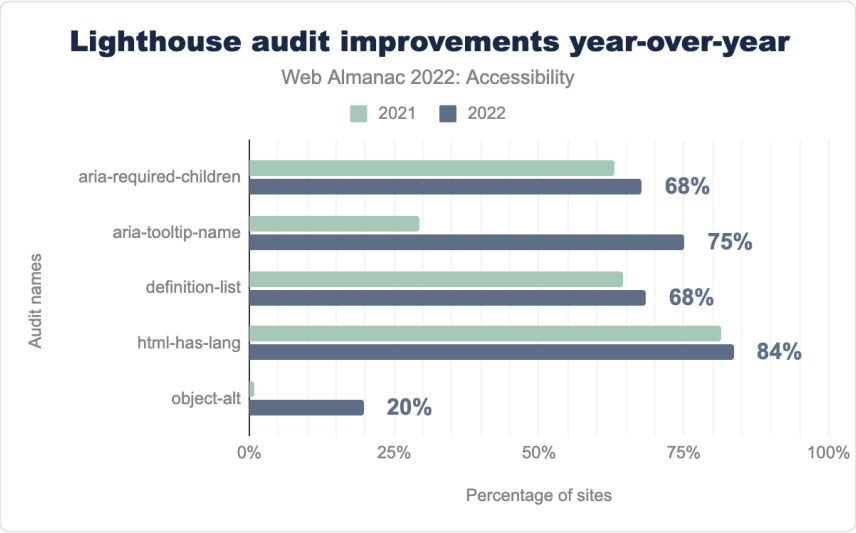



*図11.1. 年ごとのLighthouse監査の改善。*

Lighthouse監査の報告結果を見ると、41個の自動チェックのうち、35個が2022年に2021年に比べてより多くのサイトで成功裏に合格しました。11個の監査が1%以上の改善を示し、`aria-required-children`、`aria-tooltip-name`、`definition-list`、`html-has-lang`、`object-alt`がとくに顕著な増加を示しています。この増加が正しい方向へのシフトを表していることを願っています。

343. https://ja.wikipedia.org/wiki/%E3%83%A6%E3%83%8B%E3%83%90%E3%83%BC%E3%82%B5%E3%83%AB%E3%83%87%E3%82%B6%E3%82%A4%E3%83%B3

ウェブにおけるアクセシビリティの改善に向けて、実践的なリンクと解決策を提供する形で本章を書こうと試みました。一貫性を保つために、本章では「障害を持つ人々」という人第一の用語を使用していますが、アイデンティティ第一の用語「障害のある人々」も使われていることを認識しています。私たちが用語を選んだことは、どちらの用語がもっとも適切かを指示するものではありません。

## 読みやすさ

ウェブ上の情報やコンテンツの可読性は極めて重要です。コンテンツの可読性に貢献するウェブサイトの要因は数多くあります。これらの要因は、インターネット上の誰もがコンテンツを消費するだけでなく、コンテンツのどの側面によっても害を受けないことを保証します。

### カラーコントラスト

カラーコントラストとは、テキスト、図表、アイコノグラフィー、その他の情報を含む前景がセクションの背景からどれだけ際立っているかを指します。高いカラーコントラストは通常、人々がコンテンツを区別しやすくなることを意味します。

Web Content Accessibility Guidelines[344]（WCAG）によって定義される通常サイズのテキスト（24pxまで）の最小コントラスト要件は、AA適合性で4.5:1、AAA適合性で7:1です。ただし、大きなフォントサイズの場合、コントラスト要件は3:1のみであり、大きなテキストは低いコントラストでも可読性が向上します。

344. *https://www.w3.org/WAI/standards-guidelines/wcag/*

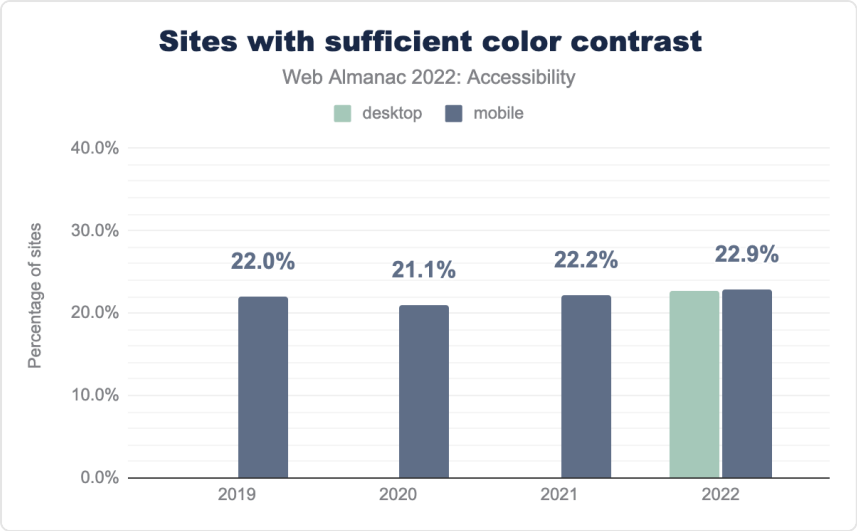



*図11.2. 十分なカラーコントラストを持つサイト。*

22.9%のモバイルサイトが十分なテキストのカラーコントラストを持っていることが分かりました。前年比でほぼ1%増加しています。2022年には、デスクトップサイトのデータもあり、22.7%が自動テキストコントラストチェックに合格しています。テキストベースのカラーコントラストに関する問題は、ウェブサイトの構築を開始する前に非常に簡単に検証できるものです。開発者やデザイナーがテキストやグラフィカル要素のカラーコントラストをチェックするのに役立つ複数のツールがあります。以下はその一部です。

- ウェブカラーコントラストチェッカー（WebAIM製）[345]
- Figmaプラグイン（Stark製）[346]

プロジェクトの初めや問題に取り組む際に、カラーコントラスト要件を満たすカラースキームを選択し、ウェブサイト全体で使用するのは良いアイデアです。また、ユーザーが選択できるように、ダークモード、ライトモード、高コントラストモードなどの他のカラーモードを提供することもできます。

## ズームとスケーリング

ズームは、視力が低いユーザーがウェブサイトのテキストをより見やすく表示するためによく使用する機能の1つです。ブラウザにはシステム設定があり、ユーザーがウェブサイトを

345. *https://webaim.org/resources/contrastchecker/*
346. *https://www.figma.com/community/plugin/733159460536249875*

ズームイン/アウトしてスケーリングすることを可能にするツールもあります。Adrian Roselli[347]は、ズームを無効にしないべきさまざまな理由について詳細に説明[348]しています。

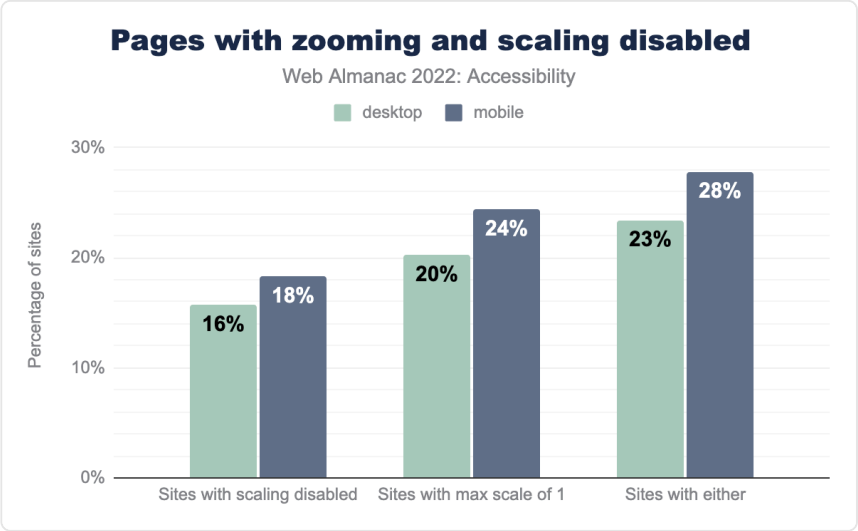



*図11.3. ズームインとスケーリングが無効になっているページ。*

WCAGは、ウェブサイト内のテキストを最低でも200%までリサイズできるようにすることを要件としています。私たちは、デスクトップホームページの23%とモバイルホームページの28%がズームを無効にしようとしていることを発見しました。

開発者がズームを無効にした方法は、`<meta name="viewport" >` タグに `content` 属性内に `maximum-scale`、`minimum-scale`、`user-scalable=no`、または `user-scalable=0` のような値を追加することです。したがって、これらの値を持つウェブサイトがある場合は、ズームを有効にするために `content` 属性から特定の値を削除してください。

347. *https://twitter.com/aardrian*
348. *https://adrianroselli.com/2015/10/dont-disable-zoom.html*

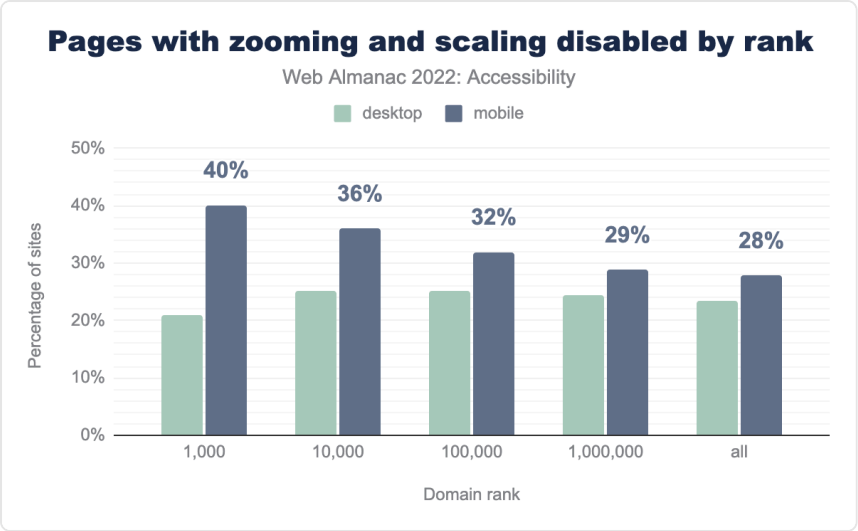



*図11.4. ランク別にズームとスケーリングが無効になっているページ。*

トップ1,000のもっとも訪問されるサイトのうち、デスクトップサイトの21%とモバイルサイトの40%は、ユーザーのズームやスケーリングを無効にしようとするコードを使用して構築されています。これは、ズームが無効になっているサイトの割合がモバイルではデスクトップと比較してほぼ倍になっていることを意味します。どのデバイスでもズームを無効にしないことは非常に重要です。この問題に対処するために、ブラウザは開発者がズームを無効にしようとする試みを上書きし始めています。Manuel Matuzović[349]は、ブラウザでのズームの無効化とユーザー設定に関する懸念について語った記事を書いています。詳細はこちら[350]をご覧ください。

349. https://twitter.com/mmatuzo
350. https://www.matuzo.at/blog/2022/please-stop-disabling-zoom/

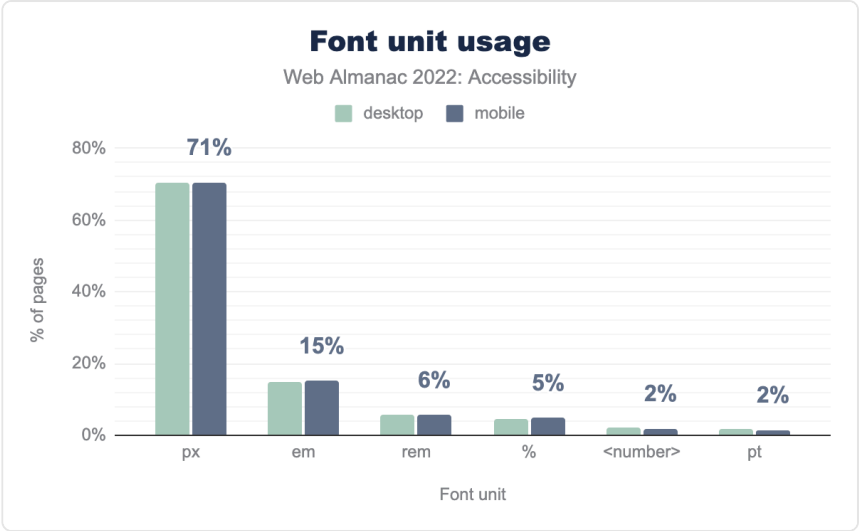



*図11.5. フォント単位の使用。*

フォントサイズに選択する単位についても考慮すべき点があります。デスクトップのページの71%が `px` を使用しており、`em` と `rem` はそれぞれ15%と6%しか使用していません。したがって、デスクトップでの `px` の使用率は前年比で2%増加し、一方で `em` の使用率は2%減少しました。`font-size` に関しては、ユーザーがブラウザの設定で明示的に大きなまたは小さなデフォルトフォントサイズを選択した場合、`px` を使用すると スケーリングされない可能性があるため[351]、`em` または `rem` などの相対単位を使用することが賢明とされています。

## 言語の識別

`lang` 属性を使用した言語の識別は、より良いスクリーンリーダーサポートを提供するために重要であり、自動ブラウザ翻訳にも役立ちます。これは、障害を持つ人々を含むすべての人々に役立つ機能のもう1つの良い例です。`lang` 属性がない場合、Chromeの自動ブラウザ翻訳がテキストを誤って翻訳することがよくあります。Manuel Matuzović[352]は、`lang` 属性の不足による自動翻訳の失敗の一例を示しています。

351. *https://adrianroselli.com/2019/12/responsive-type-and-zoom.html#Update02*
352. *https://www.matuzo.at/blog/lang-attribute/*

*図11.6. モバイルサイトには有効な `lang` 属性があります。*

83%のモバイルウェブサイトには `lang` 属性が存在することは励みになります。さらに、そのグループの中で99%以上が有効な値を持っています。ただし、これはWCAG 2.1のLevel A適合の問題であるため、改善の余地があります。この成功基準を満たすには、`lang` 属性を `<html>` タグに設定し、既知の主要言語タグ[353]を使用できます。`lang` 属性はグローバル属性であり、ウェブページに複数の言語のコンテンツがある場合に他のタグにも設定できます。ウェブサイトの正しい言語を定義することは重要です。ウェブサイトを作成するためにテンプレートをコピーする場合、ウェブサイトのコンテンツで使用される言語とコードで使用される `lang="en"` 属性との間に不一致があることがあります。

## ユーザーの設定

CSS Media Queries Level5仕様[354]には、ユーザーのアクセシビリティに関するさまざまな設定に使用できる特定のユーザー設定メディアクエリがあります。これには、ユーザーに適したカラースキームやコントラストモードを選択することから、前庭障害を持つ人々に役立つページ上のアニメーションを削減することなどが含まれます。

353. https://www.w3.org/WAI/standards-guidelines/act/rules/bf051a/#known-primary-language-tag
354. https://www.w3.org/TR/mediaqueries-5

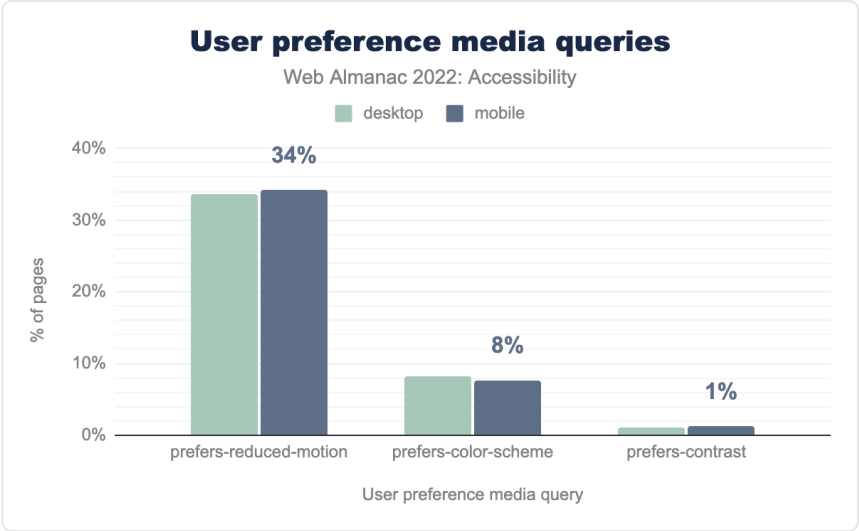



*図11.7. ユーザー設定メディアクエリ。*

モバイルウェブサイトの34%がprefers-reduced-motionを使用していることが判明しました。動きを主要な要素とするウェブサイトは、前庭障害を持つ方にとって問題を引き起こす可能性があるため、prefers-reduced-motionメディアクエリを利用して、これらのアニメーションを調整または削除することが重要です。アクセシビリティに関する素晴らしい リソース[355] も多くあり、アクセス可能なアニメーションの設計[356] に関する情報も豊富です。

デスクトップとモバイルのウェブサイトのうち、8%が `prefers-color-scheme` メディアクエリを使用し、さらに1%が `prefers-contrast` を使用していることが判明しました。これらのメディアクエリは、ユーザーの好みに基づいて ディスプレイモード[357] を調整することで、コンテンツの読みやすさを向上させます。 `prefers-color-scheme` はブラウザがユーザーのシステムカラーを検出するのを可能にし、ウェブ開発者はこの情報を使用して明暗モードを提供できます。 `prefers-contrast` は、低視力や光過敏症のあるユーザーにとって高コントラストモードが役立つ場合に便利です。

## 強制カラーモード

「強制カラーモード」は、テキストの可読性をカラーコントラストを通じて向上させるためのアクセシビリティ機能です。強制カラーモードでは、ユーザーのオペレーティングシステムがほとんどの色に関連するスタイルの制御を引き継ぎます。一般的なパターンとして背景画像などが完全に無効化されるため、テキストと背景のコントラストが予測可能になりま

355. https://alistapart.com/article/designing-safer-web-animation-for-motion-sensitivity/
356. https://www.a11yproject.com/posts/design-accessible-animation/
357. https://www.a11yproject.com/posts/operating-system-and-browser-accessibility-display-modes/

す。そのもっともよく知られた実装はWindowsの *ハイコントラストモード* であり、Windows 11では *コントラスト・テーマ* に改名されました。これらのテーマは、代替の低コントラストと高コントラストのカラーパレットを提供し、利用可能なシステムカラー[358]のどれかをカスタマイズする能力も提供します。

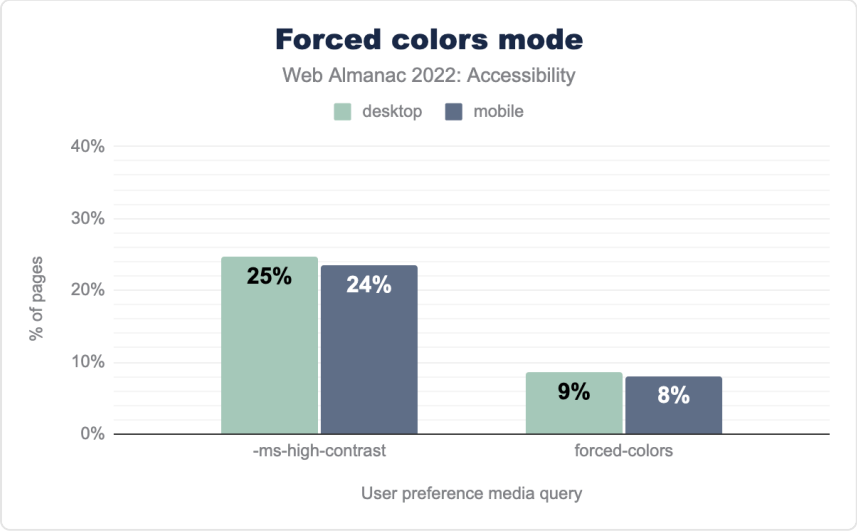



*図11.8. 強制色モード。*

他のユーザー好みのメディアクエリと同様に、多くのウェブサイトが強制色モードに基づいて調整を行っています。モバイルサイトの8%とデスクトップサイトの9%が `forced-colors` メディアクエリを使用してスタイルを変更していますが、古いIE11専用の `-ms-high-contrast` メディアクエリの使用率はモバイルとデスクトップの両方で20%以上です。 これによってサイトがどの程度強制色モードをサポートしているかは分かりませんが、`forced-colors` メディアクエリが主要なブラウザでサポートされているのは2020年から[359]であり、Windows以外のデバイスで強制色モードのエミュレート[360]をサポートするのは2022年2月からであることを考えると、データはそれでもなお励みになります。

## ナビゲーション

ウェブサイトをナビゲートする際に重要なのは、ユーザーがさまざまな方法や入力デバイスを使用する可能性があることを覚えておくことです。マウスでページをスクロールする人もいれば、キーボードやスイッチコントロールデバイスを使用する人もいます。また、スクリ

358. https://developer.mozilla.org/ja/docs/Web/CSS/system-color
359. https://caniuse.com/mdn-css_at-rules_media_forced-colors
360. https://developer.chrome.com/docs/devtools/renderingemulate-css?hl=ja

ーンリーダーを使って異なる見出しレベルをブラウズする人もいます。ウェブサイトを作成する際は、人々が選択するデバイスや支援技術に関係なく、ウェブサイトが全員にとって機能するようにすることが重要です。

## フォーカス表示

キーボードナビゲーションやスイッチコントロールデバイスに主に依存している人々にとって、フォーカス表示は非常に重要です。これらのツールは、運動能力が限られている人々によく使用されます。スイッチコントロールデバイスには、単一スイッチ[361]からシップ・アンド・パフデバイス[362]までさまざまな種類があります。目に見えるフォーカススタイルと適切なフォーカス順序は、そのようなユーザーがページ上の位置を視覚的に知るために不可欠です。

### フォーカススタイル

WCAGは、ページをトラバースする際にどの要素がキーボードフォーカスを持っているかを知るために、すべての対話型コンテンツに目に見えるフォーカスインジケーターを要求しています。事実上、WCAG 2.2[363]では（2022年12月に公開予定）、これはAAレベルからレベルAに昇格されました[364]。

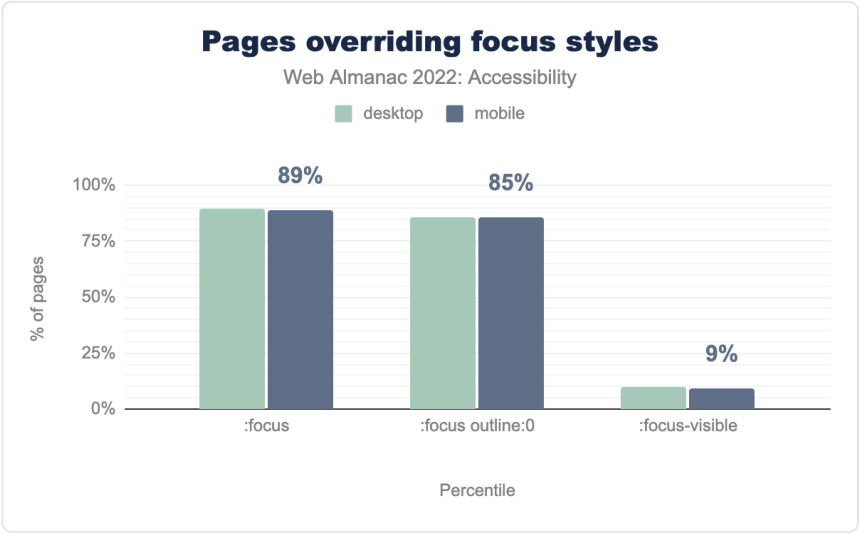



*図11.9. フォーカススタイルを上書きするページ。*

361. https://www.24a11y.com/2018/i-used-a-switch-control-for-a-day/
362. https://accessibleweb.com/assistive-technologies/assistive-technology-focus-sip-and-puff-devices/
363. https://w3c.github.io/wcag/guidelines/22/
364. https://w3c.github.io/wcag/guidelines/22/#focus-visible

私たちは、ウェブサイトの86%が `:focus {outline: 0}` を追加していることを発見しました。これは、フォーカスされた対話型要素にブラウザが使用するデフォルトのアウトラインを削除します。いくつかの場合では、カスタムスタイリングを使って上書きされていますが、常にそうではありません。これにより、ユーザーはどの要素がフォーカスされているかを判断できず、ナビゲーションが妨げられます。Sara Soueidan[365]には、WCAG準拠のフォーカスインジケーターをデザインする方法についての素晴らしい記事[366]があります。しかし、昨年の0.6%と比較して、9%のウェブサイトが `:focus-visible` を持っているのを見るのは興味深いことです。これは間違いなく正しい方向への一歩です。

## `tabindex`

`tabindex` は、要素がフォーカスを受け取ることができるかどうかを制御するために追加できる属性です。その値によって、要素はキーボードフォーカスまたは「タブ」順序内で整理されることもあります。

私たちは、モバイルウェブサイトの60%とデスクトップウェブサイトの62%が `tabindex` を使用していることを発見しました。`tabindex` 属性はいくつかの異なる目的に使用され、アクセシビリティの問題を引き起こす可能性があります：

- `tabindex="0"` を追加すると、要素はシーケンシャルなキーボードフォーカス順序に追加されます。対話型であることを意図したカスタム要素やウィジェットは、明示的に割り当てられた `tabindex="0"` が必要です。
- `tabindex="-1"` は、要素がキーボードフォーカス順序にはないが、JavaScriptを使用してプログラム的にフォーカスできることを意味します。
- 正の値の `tabindex` は、キーボードフォーカス順序を上書きするために使用され、ほとんどの場合、WCAG 2.4.3 - フォーカス順序[367]の失敗につながります。

キーボードフォーカス順序に非対話型要素を配置することは、低視力ユーザーに混乱を招く可能性があるため、避けるべきです。

365. *https://twitter.com/SaraSoueidan*
366. *https://www.sarasoueidan.com/blog/focus-indicators/*
367. *https://www.w3.org/TR/UNDERSTANDING-WCAG20/navigation-mechanisms-focus-order.html*

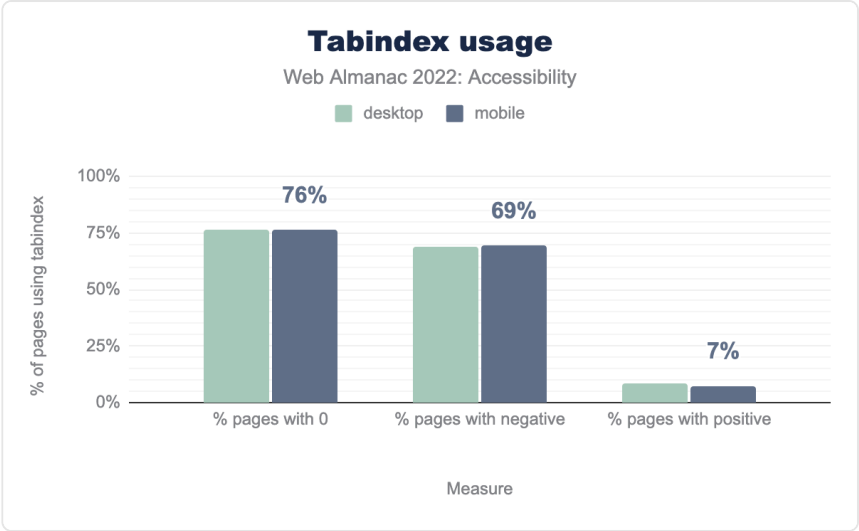



*図11.10. `tabindex` の使用。*

すべてのウェブサイトの中で `tabindex` 属性を持つものは、そのうち7%が正の値を持っています。`tabindex` に正の値を使用することは一般的に良くない慣行であり、通常のナビゲーションを妨げる可能性があります。Karl Groves[368] は、この概念について詳しく説明する素晴らしい記事[369] を書いています。

## ランドマーク

ランドマークは、ウェブページをテーマごとの領域に分けるのに役立ち、支援技術を使用するユーザーがページの構造を理解しやすくし、ウェブサイトをナビゲートしやすくします。たとえば、ローターメニュー[370]を使用して異なるページのランドマーク間をナビゲートできますし、スキップリンク[371]を使用して `<main>` を含むランドマークをターゲットに￥できます。ランドマークは、さまざまなHTML5要素を使用して作成できますが、ARIAの最初のルールにしたがって、可能な限りネイティブなHTML5要素を優先すべきです。

368. *https://twitter.com/karlgroves*
369. *https://karlgroves.com/2018/11/13/why-using-tabindex-values-greater-than-0-is-bad*
370. *https://www.afb.org/blindness-and-low-vision/using-technology/cell-phones-tablets-mobile/apple-ios-iphone-and-ipad-2*
371. *https://webaim.org/techniques/skipnav/*

| HTML5 要素 | ARIA ロール相当 | 要素を持つページ | ロールを持つページ | 要素またはロールを持つページ |
|---|---|---|---|---|
| `<main>` | `role="main"` | 31% | 17% | 38% |
| `<header>` | `role="banner"` | 63% | 13% | 65% |
| `<nav>` | `role="navigation"` | 63% | 22% | 67% |
| `<footer>` | `role="contentinfo"` | 65% | 11% | 66% |

*図11.11. ランドマーク要素とロールの使用（デスクトップ）。*

ほとんどのウェブページに期待される一般的なランドマークは、`<main>`、`<header>`、`<nav>`、`<footer>` です。デスクトップとモバイルページの31%だけがネイティブの HTML `<main>` 要素を持っており、デスクトップページの17%が `role="main"` を持つ要素を持っており、38%のページが `<main>` または `role="main"` のいずれかを持っています。ネイティブ要素の使用が増加しているのは良いことです。Scott O' Hara[372] のランドマークに関する記事[373] は、より良いアクセシビリティを確保するために心に留めておくべきすべての詳細をカバーしています。

## 見出し階層

見出しは、支援技術を使用するすべてのユーザーを含む、すべてのユーザーがウェブサイトをナビゲートするのに役立ちます。支援技術を使用するユーザーは、関心のある特定のセクションにナビゲートできます。Marcy Sutton[374] の見出しとセマンティック構造に関する記事[375] によると、見出しは特定のコンテンツエリアに移動するためにナビゲートできる目次と考えることができます。


*図11.12. 適切に順序付けられた見出しに合格したモバイルサイト。*

58%のウェブサイトは、レベルを飛ばさない正しく順序付けられた見出しのテストに合格しており、これは昨年と同じです。来年は、WHATWG標準のドキュメントアウトラインの例が更新されている[376] ため、この数字が増加することを期待しています。非常に重要なこと

372. https://twitter.com/scottohara
373. https://www.scottohara.me/blog/2018/03/03/landmarks.html
374. https://twitter.com/marcysutton
375. https://marcysutton.com/how-i-audit-a-website-for-accessibility#Headings-and-Semantic-Structure
376. https://github.com/whatwg/html/pull/7829

は、見出しレベルは特定の要素の実際のスタイル（または重要性）を表す必要はないということです。見出しは主に階層目的で使用されるべきであり、要素のスタイリングにはCSSを使用できます。ページ内の見出しの構造についての非常に良い記事は、Steve Faulkner[377] による “How to mark up subheadings, subtitles, alternative titles and taglines”[378] です。

## 二次ナビゲーション

WCAGでは、ヘッダー内のプライマリナビゲーションメニュー以外にも、異なるページ間をナビゲートするための複数の方法が必要とされています。これは、Success Criterion 2.4.5: Multiple Ways[379]に記載されています。たとえば、多くのページがあるウェブサイトでは、認知的制限を持つ人を含む多くの人が、ページを見つけるために検索機能を使用することを好みます。

モバイルのウェブサイトでは23%、デスクトップでは24%が検索入力を持っています。二次ナビゲーションのためのもう1つの推奨される方法は、ウェブサイトにサイトマップを含めることです。サイトマップの存在についてのデータはありませんが、W3Cの技術ガイド[380]では、それらが何であるか、およびそれらを効果的に実装する方法を詳しく説明しています。

## スキップリンク

スキップリンクは、キーボードやスイッチコントロールデバイスのユーザーが、すべてのフォーカス可能なアイテムを通過することなく、ページの異なるセクションをスキップできるようにする機能です。もっとも一般的にスキップされるセクションの1つは、とくにウェブサイトのプライマリナビゲーションに多くのインタラクティブなアイテムがある場合、プライマリナビゲーションから `<main>` セクションへ移動することです。


*図11.13. スキップリンクを持つ可能性があるモバイルおよびデスクトップページ*

デスクトップおよびモバイルページの21%にスキップリンクが存在する可能性があることがわかりました。これにより、ユーザーはページコンテンツの一部をバイパスできます。この数字は実際にはもっと高い可能性があります。なぜなら、私たちの検出はページの早い段階でのスキップリンクの存在のみをチェックするためです（例：ナビゲーションをスキップするため）。スキップリンクはページの一部をスキップするためにも使用できます。

377. *https://twitter.com/stevefaulkner*
378. *https://stevefaulkner.github.io/Articles/How%20to%20mark%20up%20subheadings,%20subtitles,%20alternative%20titles%20and%20taglines.html*
379. *https://www.w3.org/WAI/WCAG21/Understanding/multiple-ways.html*
380. *https://www.w3.org/WAI/WCAG21/Techniques/general/G63*

## ドキュメントタイトル

ページ、タブ、ウィンドウ間でナビゲートする際、記述的なページタイトルが役立ちます。新しいページのタイトルは支援技術によって読み上げられ、ユーザーが現在どこにいるのかを把握するのに役立ちます。

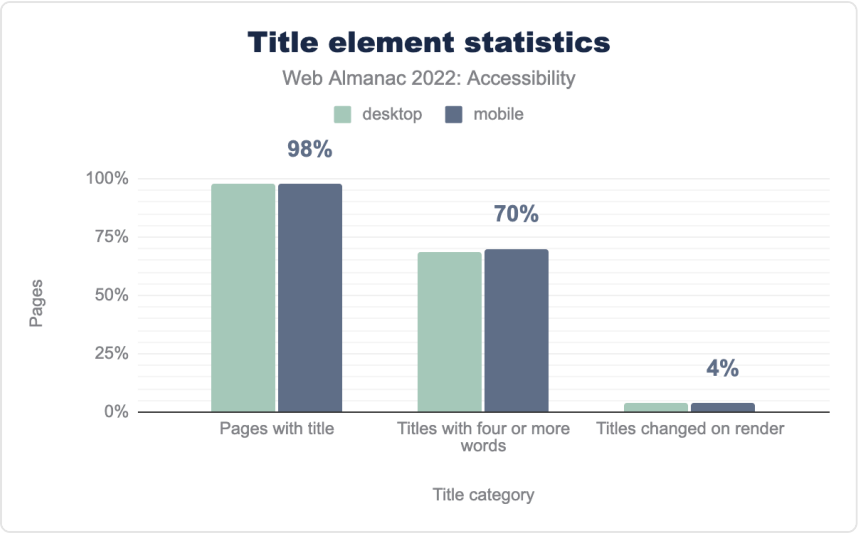



*図11.14. タイトル要素の統計。*

モバイルウェブサイトの98%にドキュメントタイトルがありますが、そのうち70%のみが4語以上のタイトルを持っています。私たちはウェブサイトのホームページのみをスキャンしているため、ウェブサイトの内部ページが `<title>` タグでページを詳しく説明するテキストを使用しているかどうかを判断することはできません。理想的には、タイトルにはウェブサイトのタイトルとページのコンテキストに関するタイトルの両方が含まれているべきで、より良いナビゲーションのために役立ちます。

## テーブル

テーブルは、二軸を使用してデータとデータ間の関係を表現するのに役立ちます。テーブルは、適切な要素とマークアップを備えた整然とした構造を持つべきであり、支援技術を使用するユーザーがテーブルに表されたデータを簡単に理解し、テーブルを通じてナビゲートするのに役立ちます。テーブルのキャプション、適切なヘッダー、そして各行の適切なヘッダーセルは、支援技術を使用するユーザーがデータを理解するのに役立つ重要な要素です。

| | テーブルサイト | | 全サイト | |
|---|---|---|---|---|
| | デスクトップ | モバイル | デスクトップ | モバイル |
| キャプション付きテーブル | 5.4% | 4.7% | 1.3% | 1.2% |
| プレゼンテーションテーブル | 1.2% | 0.9% | 0.6% | 0.5% |

*図11.15. アクセシブルなテーブルの使用。*

テーブルにキャプションを提供する場合、`<caption>` 要素はスクリーンリーダーユーザーにもっともコンテキストを提供するための正しいセマンティックな選択です。ただし、テーブルのラベリングの代替方法[381]もあります。テーブルのキャプションは、テーブルの情報を要約する見出しとして機能します。テーブル要素を持つデスクトップおよびモバイルサイトの1.3%が `<caption>` を使用しています。

テーブルはページのレイアウトにも使用されることがありますが、CSSのFlexboxやGridプロパティの登場により、視覚的なフォーマットにテーブルを使用することは避けるべきです。ただし、他に選択肢がない場合、テーブルは `role="presentation"` を設定できます。この回避策を使用するテーブルが1%観察されました。

## フォーム

フォームは、ユーザーが情報を提出し、ウェブサイトと対話するもっとも一般的な方法の1つです。ログイン、ソーシャルメディアへの投稿の作成、または電子商取引サイトでの購入など、これらのユーザージャーニーはいずれも何らかの段階でフォームを必要とします。適切なフォームのアクセシビリティがなければ、障害を持つ人々は適切に対話することができず、その結果、タスクを完了することができず、非障害者のユーザーと同じ情報や機能の平等性を達成することができません。

フォームのアクセシビリティに関しては、特定の点を念頭に置くべきです。

### `<label>` 要素

`<label>` 要素は、フォーム内の入力フィールド（またはフォームコントロール[382]）にアクセシブルな名前を提供するもっとも効果的な方法です。`for` 属性を使用して `<label>` をフォームコントロールにプログラム的にリンクできます。`for` 属性には、リンクしたいフォームコントロール要素の `id` 属性の値を含める必要があります。たとえば：

381. https://www.w3.org/WAI/tutorials/tables/caption-summary/
382. https://developer.mozilla.org/ja/docs/Learn/Forms/Basic_native_form_controls

```
<label for="emailaddress">Email</label>
<input type="email" id="emailaddress">
```

`for` 属性は重要です。これがないと、`<label>` は対応するフォームコントロールにプログラム的にリンクされません。これはフォームの使いやすさに影響を与える可能性があり、他の方法が使用されない限り、フィールドには意味的にリンクされたラベルがない可能性が高くなります。

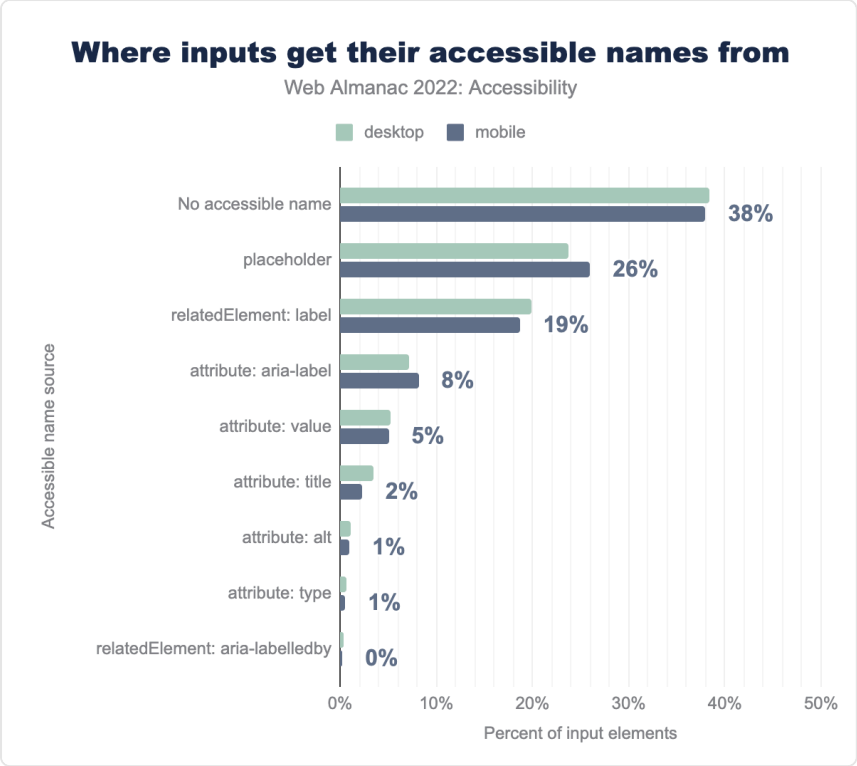



*図11.16. 入力がアクセシブルな名前を取得するソース。*

入力の38%にアクセシブルな名前がなく、わずか19%が `<label>` を使用しています。適切なアクセシブルな名前がないと、スクリーンリーダーのユーザーや音声認識のユーザーは入力が収集しようとしているデータを特定できません。多くのウェブサイトには目に見えるラベルがない入力があり、すべてのユーザーに問題を引き起こすことがあります。また、入力の目的が視覚的にもプログラム的にも明確に定義されていない場合もあります。検索フィールドなど、ラベルを視覚的に省略する場合でも、アクセシブルな名前を提供するためにスク

リーンリーダー専用の `<label>` を追加する必要があります。

## `placeholder` 属性

フォームコントロール内の `placeholder` 属性の目的は、そのフォームコントロールが受け入れるデータや形式の例を提供することです。たとえば、`<input type="text" id="credit-card" placeholder="1234-5678-9999-0000">` は、カード番号を4桁ごとにダッシュで区切って入力する必要があることをユーザーに知らせます。

しかし、`<label>` 要素とは異なり、`placeholder` 属性は誰かがタイピングまたはデータを入力し始める瞬間に消えます。これは、認知障害のあるユーザーが入力しようとしていたデータについて混乱する原因となります。また、すべてのスクリーンリーダーがアクセシブルな名前のための `placeholder` 属性をサポートしているわけではないため、問題が生じる可能性があります。そのため、アクセシブルな名前のために `placeholder` 属性を使用すると、多くのアクセシビリティ問題[383]が生じる可能性があり、避けるべきです。

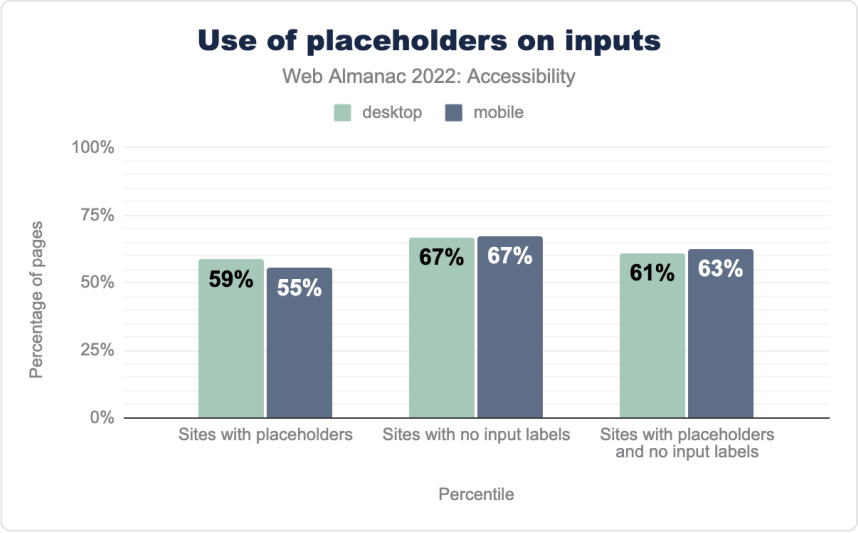



*図11.17. 入力にプレースホルダーを使用する。*

調査対象のウェブサイトの62.7%は、`<label>` 要素にリンクされていない `placeholder` 属性のみを持つ入力を持っており、これは非常に問題です。HTML5仕様[384]では、「placeholder属性は、ラベルの代わりとして使用されるべきではない」と明確に記されています。すべてのユーザーのアクセシビリティを向上させるためには、`<label>` を提供する

383. https://www.smashingmagazine.com/2018/06/placeholder-attribute/
384. https://html.spec.whatwg.org/#the-placeholder-attribute

ことが重要です。

> *ラベルの代わりとしてプレースホルダー属性を使用すると、高齢者や認知障害、移動障害、微細な運動技能障害、視覚障害を持つユーザーを含むさまざまなユーザーのコントロールのアクセシビリティと使いやすさが低下する可能性があります。*
>
> *– W3Cのプレースホルダー研究*[385]

## 情報の必要性

ウェブサイトがユーザーから入力を収集する際には、どの情報がオプションで、どの情報が提出に必要なのかを明確に示す方法が必要です。たとえば、フォームでメールアドレスは必須フィールドである可能性がありますが、ミドルネームはオプションのフィールドになる可能性があります。HTML5が2014年に `<input>` フィールドに `required` 属性を導入する前は、必須入力フィールドのラベルにアスタリスク（*）を付けるという一般的な方法がありました。しかし、アスタリスクのみを使用することは視覚的な指標に過ぎず、フィールドが必須であることを支援技術に十分に伝える検証や情報を提供しません。

385. https://www.w3.org/WAI/GL/low-vision-a11y-tf/wiki/Placeholder_Research

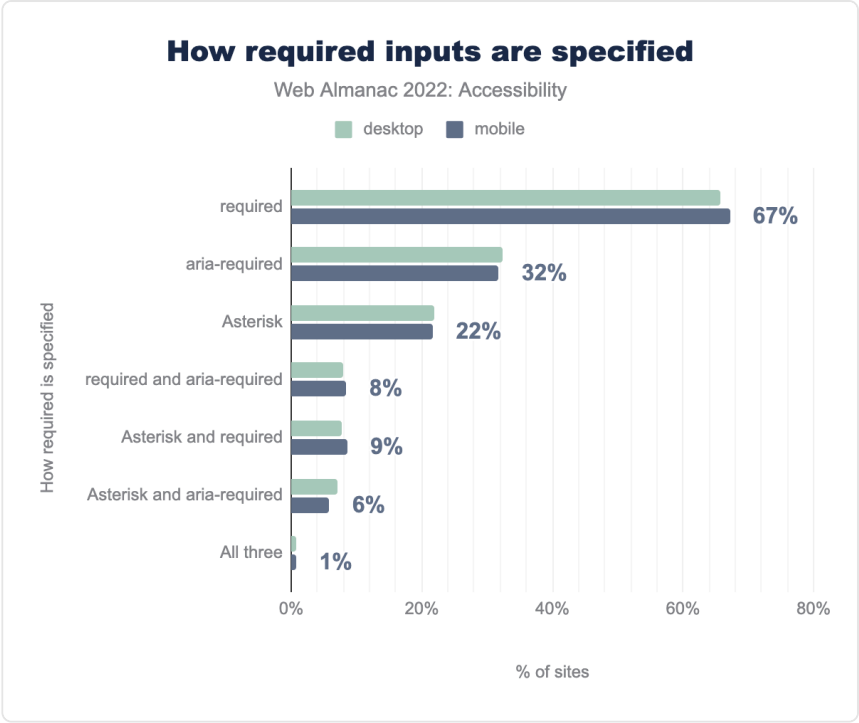



*図11.18. 必須入力が指定されている方法。*

`required` 属性と `aria-required` 属性は、入力フィールドがオプションではないことを支援技術に伝える2つの方法です。`required` 属性は入力なしでのフォーム送信を防ぎますが、`aria-required` は支援技術に情報を伝えるだけで入力の検証は行いません。私たちの調査では、サイトの67%が `required` 属性を使用し、32%が `aria-required` を使用していることがわかりました。しかし、まだ22%のウェブサイトが、フィールドが必須であることを示すためにアスタリスク（*）のみを使用しています。これは、`required` と `aria-required` と一緒に使用されない限り、絶対に避けるべきです。

## キャプチャ

ウェブサイトはしばしば、訪問者が人間であり、ウェブサイトを巡回するためのさまざまな目的でプログラムされたボットではないことを確認したいと考えます。たとえば、Web Almanacは毎年、ウェブサイトから情報を収集するために同様のウェブクローラーを送り出して作成されています。このような「人間のみ」のテストはCAPTCHA -「完全自動化された公開チューリングテスト、コンピューターと人間を区別するためのテスト」と呼ばれます。

*図11.19. CAPTCHAを使用しているモバイルサイト*

検出可能な2つのキャプチャ実装のうち、19%のモバイルウェブサイトが使用しています。このタイプのテストはすべての人にとって解決が難しいかもしれません（CAPTCHAs Have an 8% Failure Rate[386]を参照）が、とくに低視力者やその他の視覚または読解に関連する障害のある人々にとってはさらに困難です。また、このようなテストはWCAG 3.3.7 Accessible Authentication[387]の下でWCAG 2.2がリリースされたら失敗する可能性があります。実際、W3Cには視覚的なチューリングテストの代替案に関する論文[388]があり、それは非常にオススメです。

## ウェブ上のメディア

ウェブ上でのメディア消費に関しては、アクセシビリティの考慮が非常に重要になります。多くのメディアは、障害のある人々が代替方法が提供されない限り消費できないような方法で設計されています。たとえば、盲目の人や視力障害のある人には、画像やビデオに対する音声説明が必要です。スクリーンリーダーは、画像やビデオを説明する代替テキストが存在する場合にのみ、音声説明を作成できます。同様に、聴覚障害のある人々にとって、ビデオのキャプションやオーディオのテキストトラックは、素材にアクセスするために不可欠です。

### 画像




*図11.20. altテキストを持つ画像に対してLighthouse image-alt監査に合格したモバイルページ*

ウェブ上の画像は、画像の代替テキスト説明を提供する `alt` 属性を持つことができます。スクリーンリーダーは、この情報を使用して、視覚障害のある人に画像の音声説明を作成できます。私たちは、altテキストを持つ画像に関するテストに合格したサイトが59%であることを発見しました。これは2021年から1%の小さな増加です。

386. *https://baymard.com/blog/captchas-in-checkout*
387. *https://w3c.github.io/wcag/understanding/accessible-authentication.html*
388. *https://www.w3.org/TR/turingtest/*

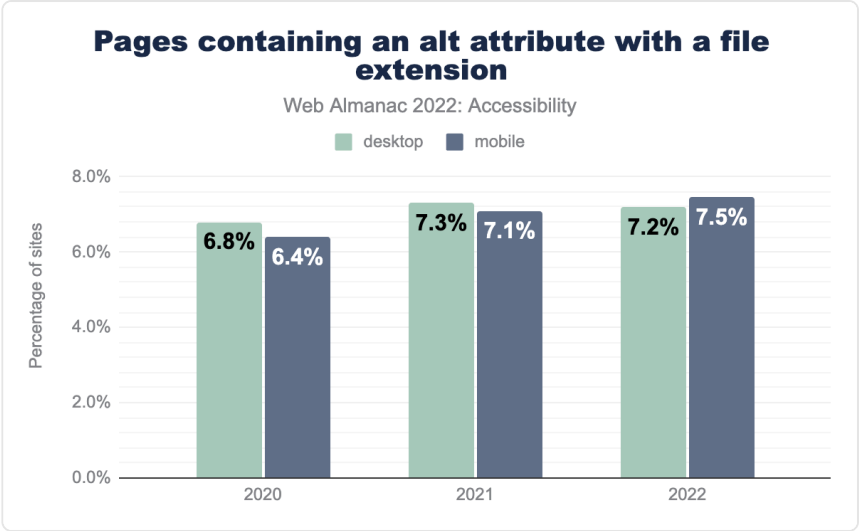



*図11.21. ファイル拡張子を含む `alt` 属性を持つページ。*

`alt` 属性のテキストは文脈に依存します。画像が装飾的で意味のある情報を提供しない場合は、`alt=""` が理想的です。しかし、画像がページの文脈にとって重要である場合は、適切なテキスト説明が不可欠です。画像がリンクの子である場合、理想的には `alt` 属性を使用してリンクをラベル付けし、ユーザーがリンク先を知ることができるようにすべきです。モバイルウェブページの7.5%とデスクトップページの7.2%で、`alt` 属性にファイル拡張子が割り当てられていることがわかりました。これはおそらく `alt` 属性が単に画像ファイル名を含んでいることを意味し、これはまったく役に立たない可能性が高く、すべての場合で避けるべきです。

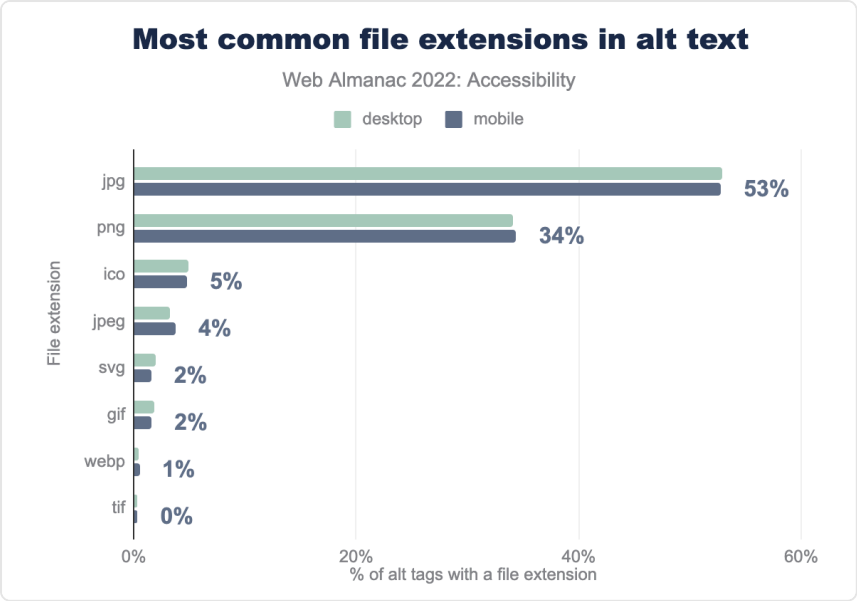



*図11.22. `alt` テキストでもっとも一般的なファイル拡張子。*

altテキストの値に明示的に含まれているトップ5のファイル拡張子（非空のalt値を持つ画像のサイトに限定）は、jpg、png、ico、jpeg、svgです。これは、CMSや他のコンテンツ管理方法が画像に代わるテキストを自動生成するか、コンテンツエディターに画像の説明を強制的に求めていることを反映している可能性があります。ただし、CMSが単に画像ファイル名を `alt` 属性に入れる場合、これはユーザーにとって何の価値も提供しないため、意味のあるテキスト説明が提供されることが重要です。

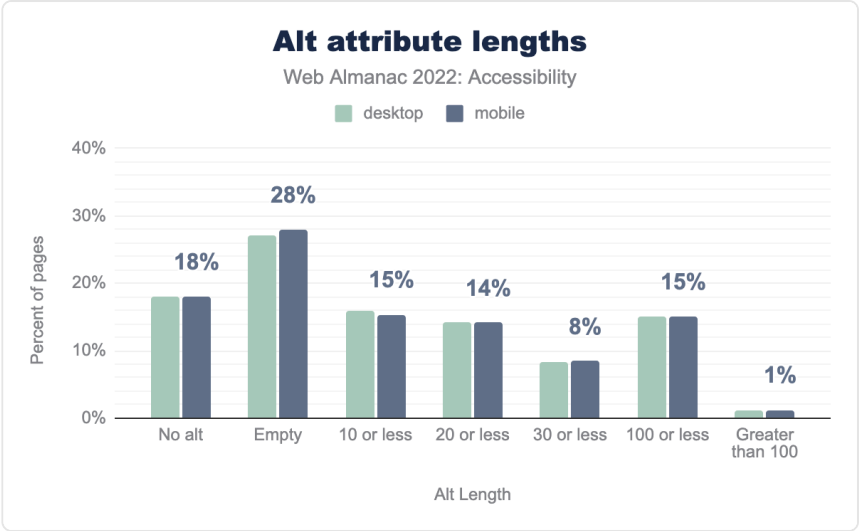



*図11.23. `alt` 属性の長さ。*

デスクトップおよびモバイルウェブサイトのaltテキスト属性の27%が空であることがわかりました。空の `alt` 属性は、画像が表現的でスクリーンリーダーや他の支援技術によって説明されるべきではない場合にのみ使用されるべきです。しかし、ウェブ上のほとんどの画像はウェブページのコンテンツに価値を加える[389]ため、適切なテキスト説明を持つべきです。15.3%が10文字以下であることがわかりましたが、これはほとんどの画像にとって奇妙に短い説明であり、情報の平等性が達成されていないことを示しています。ただし、リンクのラベル付けに使用される場合は問題ありません。

## オーディオとビデオ

`<track>` は `<audio>` および `<video>` 要素に対してタイミングを合わせたテキストコンテンツを提供するために使用されます。これは、字幕、キャプション、説明、又は章に使用できます。キャプションは、恒久的または一時的な聴覚障害を持つ人々がオーディオコンテンツを消費することを可能にします。説明は、盲目のスクリーンリーダーユーザーがビデオ内で何が起こっているかを理解するのに役立ちます。

389. https://www.smashingmagazine.com/2021/06/img-alt-attribute-alternate-description-decorative/


*図11.24. `<audio>` 要素を持つデスクトップウェブサイトで少なくとも1つの `<track>` 要素を持つもの*

`<track>` は1つ以上のWebVTTファイルをロードし、これによりテキストコンテンツを説明しているオーディオと同期させることができます。検出可能な `<audio>` 要素があるサイトのみを見ると、デスクトップのページの0.06%、モバイルのページの0.09%のみが少なくとも1つの `<track>` 要素を持っていることがわかります。すべての `<audio>` 要素を見ると、それぞれ0.03%と0.05%だけが `<track>` を含んでいます。


*図11.25. `<video>` 要素に対応する `<track>` 要素を持つデスクトップサイト*

`<track>` 要素は、対応する `<video>` 要素と一緒に使用されることは1%未満でした。デスクトップサイトでは0.71%、モバイルサイトでは0.65%です。これらのデータポイントには、ポッドキャストやYouTubeビデオなどのコンテンツに一般的な `<iframe>` 要素を介して埋め込まれたオーディオやビデオは含まれていません。また、ほとんどの一般的なサードパーティのオーディオおよびビデオ埋め込みサービスには、同期されたテキストの代替手段を追加する機能が含まれています。

## ARIAを使用した支援技術

Accessible Rich Internet Applications、またはARIA[390]は、障害のある人々のためにウェブコンテンツをよりアクセシブルにするためにHTML5要素に使用できる一連の属性を定義しています。しかし、ARIAの過剰使用は、アクセシビリティの改善よりも問題を引き起こす可能性があります。常に、HTML5だけでは完全にアクセシブルなエクスペリエンスを作成するのに十分でない場合にのみ、ARIA属性を使用することが推奨されます。それはネイティブのHTML5要素の代替として使用されるべきではなく、不必要に過剰使用されるべきではありません。

390. *https://www.w3.org/TR/using-aria/*

## ARIAの役割

支援技術が要素に遭遇すると、その要素の役割は、誰かがそのコンテンツとどのように相互作用するかに関する情報を伝えます。

たとえば、タブ付きインターフェイス[391]は、UIの構造を適切に伝えるために、さまざまなARIAの役割を明示的に定義する必要があるもっとも一般的に使用されるUI要素の1つです。アクセシブルなタブ付きインターフェイスの一般的な実装は、WAI-ARIAオーサリングプラクティスデザインパターン[392]で述べられています。タブリストウィジェットを作成する場合、コンテナー要素に `tablist` 役割を割り当てることができます。これは、ネイティブHTMLに同等のものがないためです。

HTML5は、役割を含む暗黙のセマンティクスを持つ多くの新しいネイティブ要素を導入しました。たとえば、`<nav>` 要素には暗黙の `role="navigation"` があり、この役割をARIAを介して明示的に追加する必要はありません。

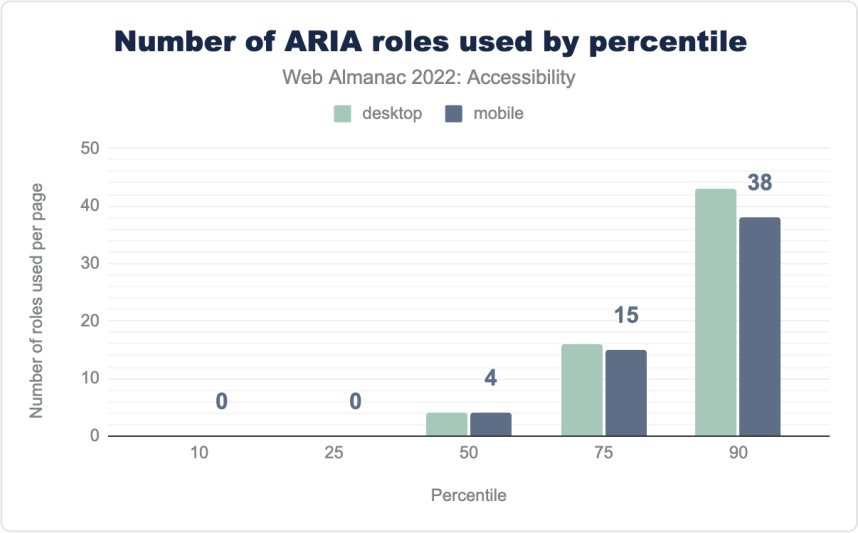



*図11.26. パーセンタイル別に使用されているARIAの役割数。*

現在、デスクトップページの72%に少なくとも1つのARIAの役割属性があります。中央値のサイトには `role` 属性のインスタンスが4つあります。

391. https://inclusive-components.design/tabbed-interfaces/
392. https://www.w3.org/TR/wai-aria-practices-1.1/#tabpanel

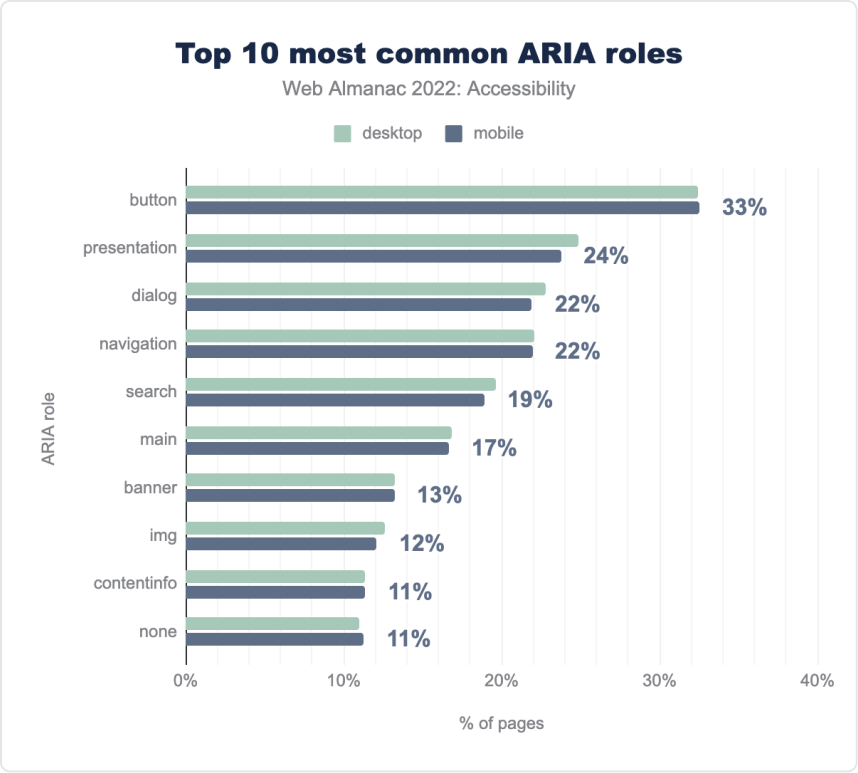



*図11.27. もっとも一般的なARIAの役割トップ10。*

2021年の29%、2020年の25%から上昇し、デスクトップおよびモバイルサイトの33%が明示的に `role="button"` を割り当てられた少なくとも1つの要素を持つホームページを持っていることがわかりました。この割合の増加はおそらく良いことではありません。これは、ウェブサイトが `<div>` や `<span>` を使用してカスタムボタンを作成しているか、または `<button>` 要素に冗長な役割を追加していることを示しています。これらはどちらも悪い実践であり、ARIAの最初のルール[393]に反しています。このルールにしたがって、この場合は `<button>` として、可能な場合は常にネイティブHTML要素を使用する必要があります。

**21%**

*図11.28. 少なくとも1つのリンクに `button` 役割を持つデスクトップウェブサイト*

393. https://www.w3.org/TR/using-aria/#rule1

ARIAの役割を追加すると、支援技術に要素は何であるかを伝えます。しかし要素はそのネイティブの同等物のように振る舞うための他の機能は提供しません。たとえば、少なくとも1つのリンクに `role="button"` を持つウェブサイトが21%ありました。この種のパターンは、リンクとボタンが異なる相互作用を持っているため、キーボードナビゲーションに問題を引き起こす可能性があります。リンクとボタンの両方がインタラクティブです、リンクはスペースキーでアクティベートされませんが、ボタンはそうです。

### presentationロールの使用

要素に `role="presentation"` が宣言されている場合、そのセマンティクスは取り除かれ、その子要素のセマンティクスも取り除かれます（子要素が必須の子要素である場合、たとえば `ul` 要素の下にネストされた `li` やテーブルの行やセルなど）。あと、親のテーブルまたはリスト要素に `role="presentation"` を宣言すると、その役割が必須の子要素にカスケードされ、それらのどれもテーブルやリストのセマンティクスを持たなくなります。


*図11.29. 少なくとも1つの要素に `role=presentation` を持つモバイルウェブサイト*

要素のセマンティクスを削除すると、その要素は振る舞いを失います。それは視覚的にのみ存在し、支援技術はその要素の構造を理解できず、そのメッセージをユーザーに伝えることができません。たとえば、`role="presentation"` を持つリストは、スクリーンリーダーにリスト構造に関する情報を伝えなくなります。デスクトップページの25%とモバイルページの24%に、少なくとも1つの `role="presentation"` を持つ要素があることがわかりました。


*図11.30. 少なくとも1つの要素に `role=none` を持つモバイルウェブサイト*

`role="none"` によるセマンティクスの削除も同じ効果があります。今年、`role="none"` の割合も11%に増加し、もっとも一般的なARIAの役割のトップ10に入っていることがわかりました。これが支援技術のユーザーにとくに役立つ使用例はほとんどありません。たとえば、レイアウト目的でのみ `<table>` 要素が使用されている場合です。しかし、それ以外では有害であり、慎重に使用する必要があります。

ほとんどのブラウザは、リンクや入力、または `tabindex` 属性が設定されたものなど、フ

ォーカス可能な要素に対する `role="presentation"` と `role="none"` を無視します。また、ブラウザは、要素に `aria-describedby` などのグローバルARIAの状態やプロパティが含まれている場合、役割の含まれることも無視します。

## ARIAを使用した要素のラベリング

DOMに平行して、アクセシビリティツリーと呼ばれる類似のブラウザ構造があります。これには、HTML要素に関する情報（アクセシブルな名前、説明、役割、状態など）が含まれています。この情報は、アクセシビリティAPIを介して支援技術に伝えられます。

アクセシブルな名前は、要素のコンテンツ（ボタンのテキストなど）、属性（画像の `alt` 属性の値など）、または関連する要素（フォームコントロールのプログラムで関連付けられたラベルなど）から派生できます。複数の潜在的なソースがある場合に、要素がそのアクセシブルな名前をどこから取得するかを決定するために、特定の優先順位付けが使用されます。Léonie Watson[394]の記事、What is an accessible name?[395]は、アクセシブルな名前について学ぶのに素晴らしい情報源です。

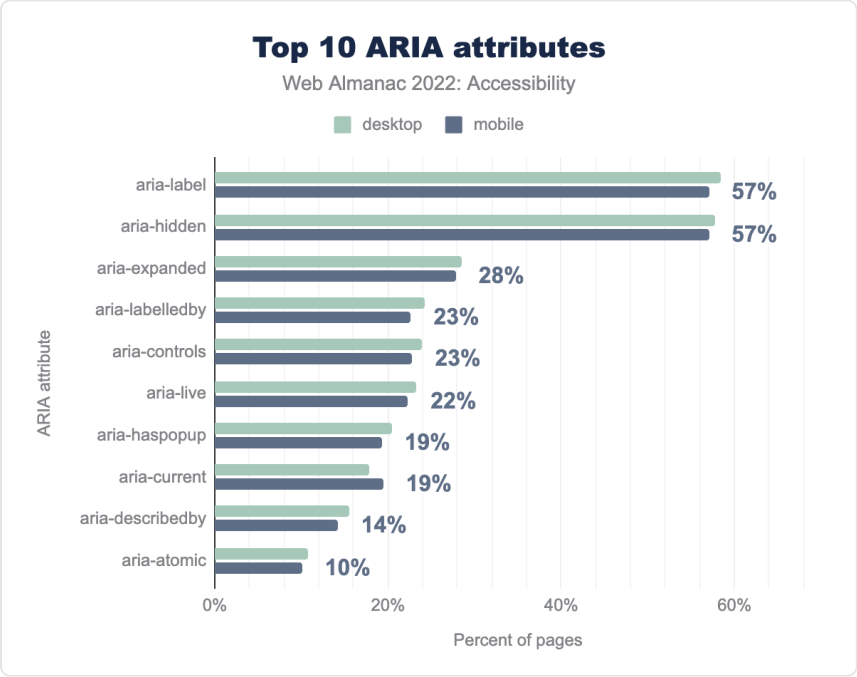



*図11.31. トップ10のARIA属性。*

394. *https://twitter.com/LeonieWatson*
395. *https://developer.paciellogroup.com/blog/2017/04/what-is-an-accessible-name/*

要素にアクセシブルな名前を提供するのに役立つ2つのARIA属性があります：`aria-label` と `aria-labelledby`。これらの属性は、ネイティブに派生するアクセシブルな名前よりも優先されるため、非常に慎重に、必要な場合にのみ使用する必要があります。スクリーンリーダーで得られたアクセシブルな名前をテストし、障害のある人々を巻き込んで、実際に役立つことを確認し、コンテンツをさらにアクセシブルでなくすることがないようにするのが常に良いアイデアです。

アクセシブルな名前の提供に `aria-label` 属性を使用する要素が少なくとも1つあるデスクトップページは58%、モバイルホームページは57%で、アクセシブルな名前を提供するためのもっとも人気のあるARIA属性であることがわかりました。`aria-labelledby` 属性を持つ要素が少なくとも1つあるデスクトップページは24%、モバイルページは23%でした。これは、より多くの要素がアクセシブルな名前を持つようになったことを意味するかもしれませんが、より多くの要素が視覚的なラベルを欠いていることを示している可能性もあります。これは、認知障害のある人々や音声からテキストへのユーザーにとって問題になる可能性があります（`<label>` 要素を参照）。

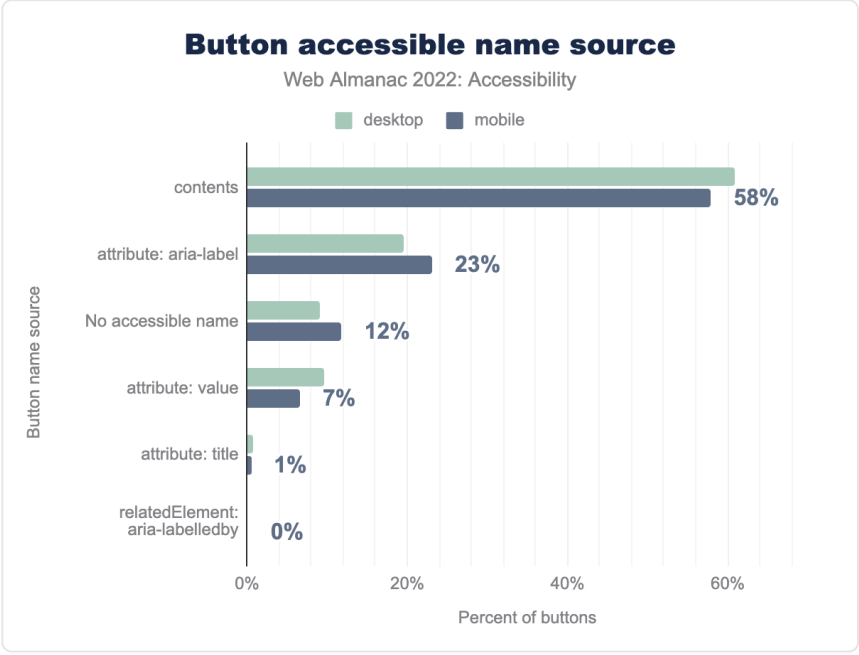



*図11.32. ボタンのアクセシブルな名前のソース。*

ボタンは通常、そのコンテンツまたはARIA属性からアクセシブルな名前を取得します。ARIAの最初のルールによれば、要素がARIAを使用せずにアクセシブルな名前を導出できる場合、これが望ましいです。したがって、`<button>` はARIA属性ではなく、そのテキスト

コンテンツからアクセシブルな名前を取得する必要があります。

デスクトップのボタンの61%とモバイルサイトの58%がコンテンツからアクセシブルな名前を取得していることがわかりました。これは良いことです。また、デスクトップサイトのボタンの20%とモバイルサイトのボタンの23%が `aria-label` 属性からアクセシブルな名前を取得していることもわかりました。

`aria-label` が役立ついくつかのケースがあります。たとえば、同じコンテンツを持つ複数のボタンがあるが、異なる目的がある場合、`aria-label` を使用してより良いアクセシブルな名前を提供できます。開発者は、ボタンに画像やアイコンのみがある場合に `aria-label` を使用することもあります。これが、モバイルサイトがコンテンツではなく `aria-label` を使用してアクセシブルな名前を定義する理由の1つかもしれません。

## コンテンツの非表示

時には視覚的インターフェイスに支援技術のユーザーにとって役に立たない冗長な要素が含まれることがあります。そのような場合には、`aria-hidden="true"` を使用してスクリーンリーダーからその要素を隠すことができます。しかし、そのような要素を省略するとスクリーンリーダーユーザーが視覚インターフェイスよりも少ない情報を得る場合には決して使用すべきではありません。支援技術からコンテンツを隠すことは、アクセシブルにするのが困難なコンテンツをスキップするために使用されるべきではありません。


*図11.33. 少なくとも1つの `aria-hidden` 属性を持つデスクトップウェブサイト*

`aria-hidden` 属性を持つ要素が少なくとも1つあるデスクトップページは58%、モバイルページは57%でした。コンテンツの非表示と表示は、現代のインターフェイスで一般的なパターンであり、すべての人のためにUIをすっきりさせるのに役立つことがあります。

適切なaria属性を使用して正しいメッセージを伝えることが重要です。たとえば、開示ウィジェットは `aria-expanded` 属性を使用して、コントロールがアクティブにされると何かが展開されて表示され、再度アクティブにされると隠されることを支援技術に示す必要があります。`aria-expanded` 属性を持つ要素が少なくとも1つあるデスクトップページは29%、モバイルページは28%でした。

## スクリーンリーダー専用テキスト

開発者がスクリーンリーダーユーザーに追加情報を提供するために採用する一般的な手法

は、テキストの一部を視覚的に非表示にしつつスクリーンリーダーで発見可能にするCSSを使用することです。このCSSコード[396]は、視覚的には隠されていますがアクセシビリティツリーに存在するように使用されます。


*図11.34.* `sr-only` *または* `visually-hidden` *クラスを持つデスクトップウェブサイト*

`sr-only` と `visually-hidden` は、スクリーンリーダー専用テキストを実現するために開発者やUIフレームワークによってもっとも一般的に使用されるクラス名です。たとえば、BootstrapやTailwindはこのような要素に `sr-only` クラスを使用しています。デスクトップページの15%とモバイルページの14%にこれらのCSSクラス名のいずれかまたは両方があることがわかりました。すべてのスクリーンリーダーユーザーが完全に視覚障害者であるわけではないため、スクリーンリーダー専用ソリューションの使用を過度に行わないようにすることが重要です。

### 動的にレンダリングされるコンテンツ

DOM内の新しいコンテンツや更新されたコンテンツがスクリーンリーダーに伝えられる必要があることがあります。たとえば、フォームの検証エラーは伝える必要がありますが、遅延読み込みされた画像はそうではないかもしれません。DOMへの更新も、中断を引き起こさないように行う必要があります。


*図11.35.* `aria-live` *を使用してライブリージョンを持つデスクトップページ*

ARIAライブリージョンを使用すると、DOM内の変更を監視し、更新されたコンテンツをスクリーンリーダーによってアナウンスできます。デスクトップページの23%（2021年の21%、2020年の17%から増加）とモバイルページの22%（2021年の20%、2020年の16%から増加）が `aria-live` を使用してライブリージョンを持っていることがわかりました。さらに、ページは暗黙の `aria-live` 値を持つライブリージョンARIAの役割[397]も使用しています：

396. https://css-tricks.com/inclusively-hidden/
397. https://www.w3.org/TR/wai-aria-1.1/#live_region_roles

| 役割 | 暗黙の aria-live 値 | デスクトップ | モバイル |
|---|---|---|---|
| status | polite | 5.6% | 5.1% |
| alert | assertive | 3.7% | 3.4% |
| timer | off | 0.6% | 0.6% |
| log | polite | 0.4% | 0.4% |
| marquee | off | 0.0% | 0.0% |

*図11.36. ライブリージョンARIAの役割とその暗黙の `aria-live` 値を持つページ*

`<output>` 要素も、その内容をエンドユーザーにアナウンスする暗黙のライブリージョン役割を持つ唯一のHTML要素として特筆に値します。私たちのデータセットでは、モバイルサイトで16,144回、デスクトップで12,120回使用されているか、要素の使用率の約0.0002%です。

ライブリージョンのバリエーションや使用法についての詳細は、MDNライブリージョンドキュメント[398]を参照するか、Dequeによるこのライブデモ[399]で遊んでみてください。

## アクセシビリティアプリとオーバーレイ

アクセシビリティオーバーレイは、ウェブサイトのアクセシビリティ問題を自動的に修正すると主張するツールです。

オーバーレイの事実[400]によると、これらは「ウェブサイトのアクセシビリティを改善することを目指す技術の幅広い用語です。これらはサードパーティのソースコード（通常はJavaScript）を適用し、ウェブサイトのフロントエンドコードを自動的に改善します」と定義されています。

これらのベンダーは一般的に、オンラインアクセシビリティに対して迅速かつ簡単な解決策を約束しています：1つのJavaScriptスニペットをサイトに統合して準拠させるだけです。このような箱から出してすぐに使えるソリューションでウェブアクセシビリティを達成することは単純に不可能です。自動化ツールは、そもそもアクセシビリティ問題の30〜50%しか検出できません[401]し、検出可能な問題に対しても、オーバーレイによる自動修正は常に信頼性を持って問題を修正できるわけではありません。

398. https://developer.mozilla.org/ja/docs/Web/Accessibility/ARIA/ARIA_Live_Regions
399. https://dequeuniversity.com/library/aria/liveregion-playground
400. https://overlayfactsheet.com/#what-is-a-web-accessibility-overlay
401. https://alphagov.github.io/accessibility-tool-audit/

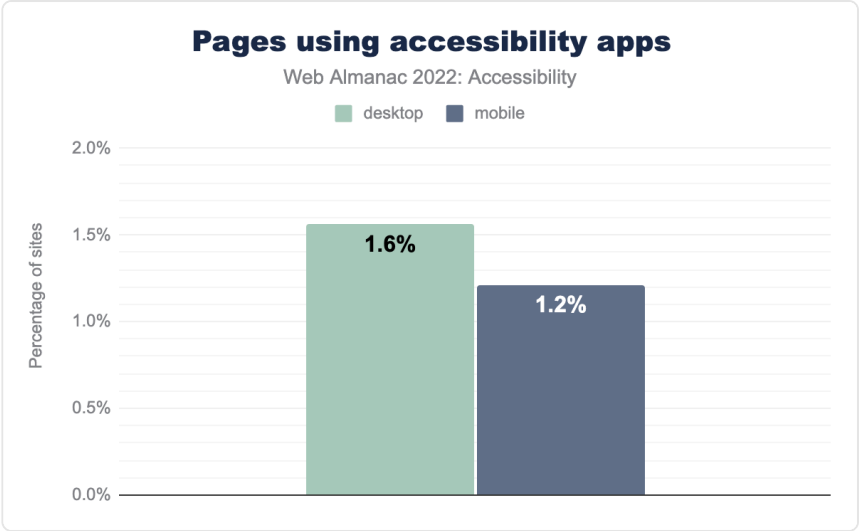



*図11.37. アクセシビリティアプリを使用するページ。*

2022年に検出可能な22の特定のアクセシビリティアプリを使用するデスクトップウェブサイトは1.6%で、これは2021年の約1%に比べて明らかな上昇です。

これらの製品のすべてがアクセシビリティオーバーレイであるわけではありませんが、検出可能な特定のオーバーレイも同様の上昇を示しています。

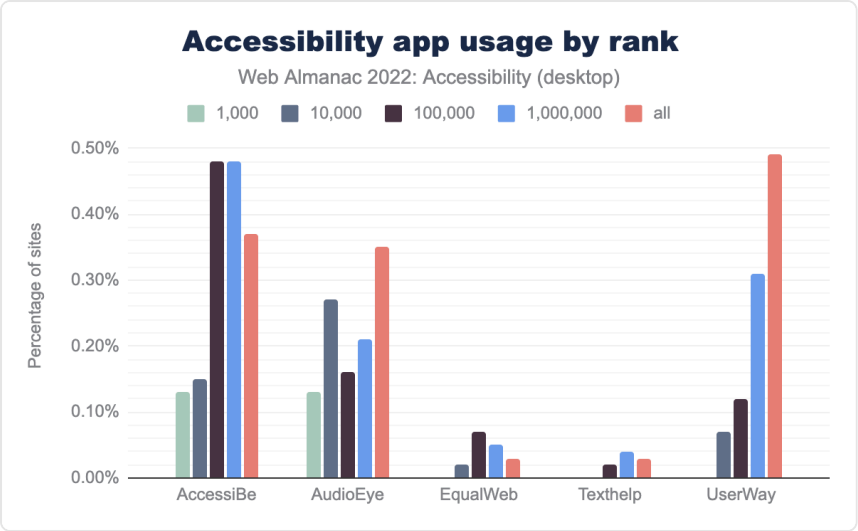



*図11.38. ランク別のアクセシビリティアプリの使用状況。*

データセットでもっとも人気のあるオーバーレイはUserWayで、デスクトップウェブサイトの0.49%とモバイルで0.39%で使用されています。これは、2021年のそれぞれ0.39%と0.33%と比較しています。

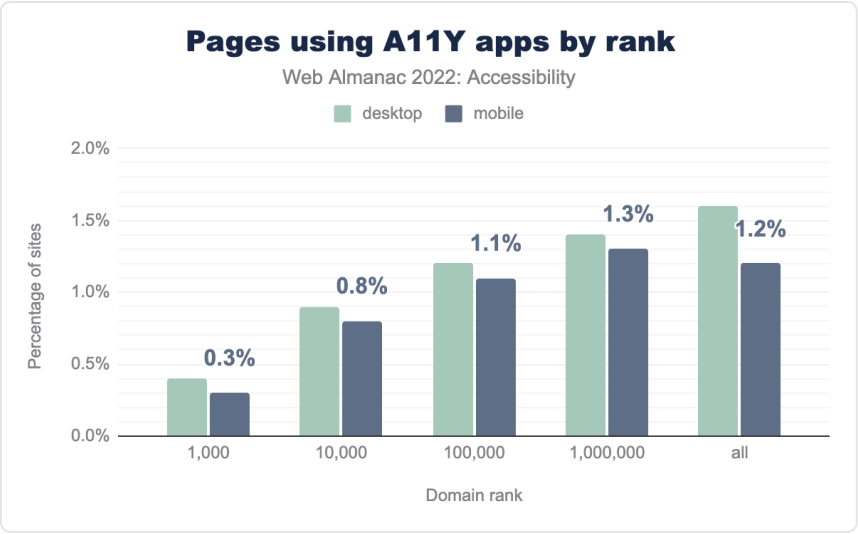



*図11.39. ランク別のアクセシビリティアプリを使用するページ。*

アクセシビリティアプリ一般に、オーバーレイの使用は高トラフィックのウェブサイトではまれです。訪問者数でトップ1,000にランクされたサイトでは、わずか0.3%、つまり3つのウェブサイトがオーバーレイを使用しています。

### オーバーレイに対する懸念

オーバーレイに対しては、アクセシビリティ擁護者[402]やユーザー[403]から多くの反発があります。National Federation of the Blindは、とくにAccessiBeの実践を非難しています[404]。AccessiBeは、偽のレビューを含む[405]欺瞞的なマーケティングで知られています：

> *[...] 当委員会は、accessiBeが現在、社会における盲人の進歩に有害な行動をとっていると考えています。[...] AccessiBeは、盲目の専門家や定期的なスクリーンリーダーユーザーが何がアクセシブルで何がアクセシブルでないかを知っていることを認めていないようです。国内の盲人はなだめられたり、脅迫されたり、買収されたりすることはありません。*
>
> *— National Federation for the Blind*[406]

AccessiBeは2022年にさらに3,000万ドルを調達しました[407]が、年々使用率が明確に上昇しているオーバーレイの1つです。2021年にはデスクトップサイトの0.27%とモバイルサイトの0.21%でしたが、2022年には0.37%と0.25%に増加しました。

Adrian RoselliのaccessiBe Will Get You Sued[408]は、このようなオーバーレイの使用に関連する法的リスクやプライバシー問題を含む実用的な意味についての優れたリソースです。

## 結論

私たちの分析によると、ウェブサイトで見られるアクセシビリティ問題にはあまり大きな変化がありませんでした。たとえば、`:focus-visible`の採用は今年約9%増加しましたが、カラーコントラストや画像の`alt`属性など、対処すれば大きな影響を与える可能性のある簡単な修正がまだたくさんあります。

実際には体験を損なうことが多いにもかかわらず、よりアクセシブルなものであるという錯覚を与える機能の誤用が多いことがわかります。たとえば、ウェブサイトの20%で`role=button`がアンカータグに使用されています。また、ウェブサイト全体での`alt`属

402. https://overlayfactsheet.com/
403. https://www.vice.com/en/article/m7az74/people-with-disabilities-say-this-ai-tool-is-making-the-web-worse-for-them
404. https://nfb.org/about-us/press-room/national-convention-sponsorship-statement-regarding-accessibe
405. https://www.joedolson.com/2021/02/accessibe-the-fake-wordpress-plug-in-reviews/
406. https://nfb.org/about-us/press-room/national-convention-sponsorship-statement-regarding-accessibe
407. https://www.geektime.com/accessibe-raised-30m/
408. https://adrianroselli.com/2020/06/accessibe-will-get-you-sued.html

性の2.2%にファイル拡張子が含まれており、これが画像の意味を伝える上でほとんど役に立たないことはほぼ確実です。

私たちの分析で見られる多くのアクセシビリティ問題は、デザイナーや開発者が最初からウェブアクセシビリティについて考え、最後の強化としてではなく、避けることができます。Anna E. Cookがかつて述べた[409]ように、「アクセシビリティなしにMVPはない」のです。ウェブコミュニティは、ウェブサイトがすべての人にとって素晴らしいユーザーエクスペリエンスを持つためには、そのユーザーエクスペリエンスがどのデバイスや支援技術を使用していても機能する必要があることを認識する必要があります。簡単に対処できる主要なメトリクスに焦点を当ててみましたが、2023年には数字が改善されることを期待しています。

## 著者


### Saptak Sengupta

@Saptak013 SaptakS https://saptaks.website

Saptak Sは、ウェブ開発において使いやすさ、セキュリティ、プライバシー、アクセシビリティを重視する、人権に焦点を当てたウェブ開発者です。彼は、The A11Y Project[410]、OnionShare[411]、Wagtail[412]など、様々なオープンソースプロジェクトのコントリビューター兼メンテナーです。彼のブログはsaptaks.blog[413]で見つけることができます。

### Thibaud Colas

@thibaud_colas thibaudcolas https://thib.me/

Thibaud Colasは、アクセシビリティのトピックに焦点を当てたウェブ開発者であり、オープンソースのコントリビューターです。彼は、Wagtail[414] CMSのコアコントリビューターであり、Django[415]のアクセシビリティチームのメンバーでもあります。


409. https://twitter.com/annaecook/status/1404615552883060737
410. https://www.a11yproject.com
411. https://onionshare.org/
412. https://wagtail.org/
413. https://saptaks.blog
414. https://wagtail.org/
415. https://www.djangoproject.com/

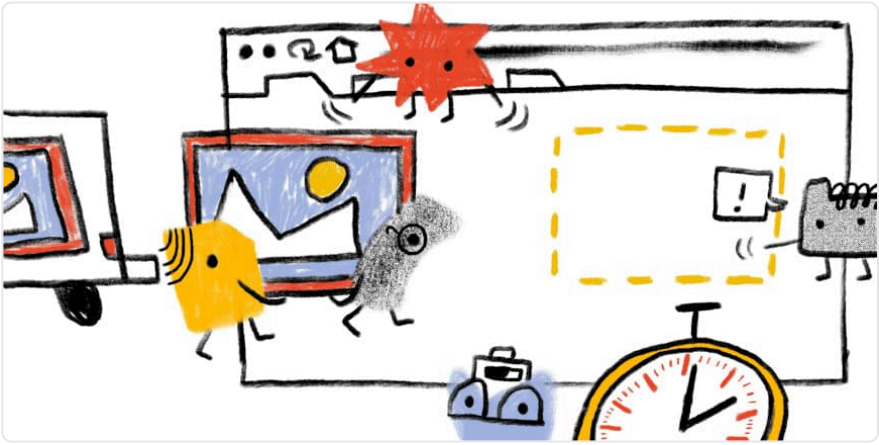

## Scott Davis

@scottdavis99 scottdavis99 http://thirstyhead.com


Scott Davisは、Thoughtworks[416]における著者兼デジタルアクセシビリティアドボケートです。彼は、ウェブ開発の最先端・革新的・新興・非伝統的な側面に焦点を当てています。”デジタルアクセシビリティは、単なるコンプライアンスチェックボックス以上のものです。アクセシビリティは革新のためのスプリングボードです。”

416. *https://www.thoughtworks.com/*

# 部 II 章 12
# パフォーマンス



*Melissa Ada と Rick Viscomi によって書かれた。*
*Barry Pollard、 Patrick Meenan、 Prathamesh Rasam、 Estelle Weyl と Kanmi Obasa によってレビュ。*
*Rick Viscomi、 Prathamesh Rasam、 Sia Karamalegos と Kanmi Obasa による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

ウェブパフォーマンスはユーザーエクスペリエンスにとって非常に重要です。私たちは皆、ロード時間が遅いためにサイトを離れた経験があるか、あるいはもっと悪いことに、重要な情報にアクセスできなかった経験があります。さらに、多くのケーススタディ[417]が、ウェブパフォーマンスの向上がビジネスのコンバージョンとエンゲージメントの向上につながることを示しています。驚くべきことに、ウェブパフォーマンスに対する業界の注目度は意外と低いのですが、なぜでしょうか？ウェブパフォーマンスは定義が難しく、さらに測定することがさらに困難だと言う人もいます。

そもそも定義が難しいものをどのように測定するのでしょうか？Sergey Chernyshev[418]（UX Captureの作成者）が言うように、*”パフォーマンスを測定する最良の方法は、ユーザーの脳*

417. *https://wpostats.com/*
418. *https://twitter.com/sergeyche*

*に組み込まれ、彼らがサイトを使用しているときに正確に何を考えているかを理解することです*"。私たちは、そして明らかにすべきではないですが、これを行うことはできません。では、私たちの選択肢は何でしょうか？

幸いにも、パフォーマンスのいくつかの側面を自動的に測定する方法があります！ブラウザがページのロードを担当しており、毎回チェックリストの一連のステップを実行することを私たちは知っています。ブラウザがどのステップにあるかに応じて、サイトがページロードプロセスのどこまで進んでいるかを知ることができます。便利なことに、ブラウザが特定のページロードステップに到達したときにタイムスタンプを発行するために使用される多数のパフォーマンスタイムラインAPI[419]があります。

これらの指標は、ユーザーエクスペリエンスを測定するための私たちの最善の推測に過ぎないことに注意することが重要です。たとえば、ブラウザが画面に要素が描画されたイベントを発火したとしても、それが常にその時点でユーザーに見えていたとは限りません。さらに、業界が成長するにつれて、より多くの指標が登場し、いくつかは非推奨になりました。どこから始めればよいか、パフォーマンス指標が私たちのユーザーについて何を伝えているのかを理解することは、とくにこの分野の新しい人々にとって複雑になる可能性があります。

この章では、問題に対するGoogleの解決策であるCore Web Vitals[420]（CWV）に焦点を当てています。これは、2020年に導入され、2021年に検索ランキングのシグナル[421]となったウェブパフォーマンス指標です。3つの指標はそれぞれ、ローディング、インタラクティビティ、視覚的安定性というユーザーエクスペリエンスの重要な領域をカバーしています。公開されているChrome UX Report[422]（CrUX）データセットは、CWVでウェブサイトがどのようにパフォーマンスを発揮しているかのChromeの視点を提供します。開発者側での設定は一切不要で、Chromeは対象となるウェブサイト[423]から、オプトインしたユーザーのデータを自動的に収集して公開しています。このデータセットを使用することで、時間をかけてウェブのパフォーマンスがどのように変化しているかについての洞察を得ることができます。

この章の焦点であるにもかかわらず、CWVがこの分野で比較的新しく、ウェブパフォーマンスを測定する唯一の方法ではないことに注意することが重要です。検索ランキングへの影響がほぼ1年前に有効であったため、これらの指標に焦点を当てることを選択しました。そして、今年のデータは、ウェブがこの業界の大きな変化にどのように適応しているか、そしてまだ機会が存在するかもしれない領域についての洞察を提供します。CWVは、サイト間のパフォーマンスを大まかに比較可能にする共通のベースラインを提供しますが、どの指標と戦略が自分のサイトに最適かを決定するのはサイト所有者次第です。私たちが望むとおりに、業界の全歴史やパフォーマンスを評価するためのすべての異なる方法を1章に収める方法はありません。

419. *https://developer.mozilla.org/ja/docs/Web/API/Performance*
420. *https://web.dev/articles/vitals?hl=ja*
421. *https://developers.google.com/search/blog/2020/11/timing-for-page-experience*
422. *https://developer.chrome.com/docs/crux*
423. *https://developer.chrome.com/docs/crux/methodology#eligibility*

CWVプログラムは、業界ではじめて、ユーザーが実際にパフォーマンスをどのように体験しているかを測定する明確に定義されたアプローチを提案しています。CWVは、ウェブをよりパフォーマントにするための答えでしょうか？この章では、現在のウェブがCWVとどのような関係にあるか、そして未来に向けての展望を検討します。

*開示: この章は、Core Web Vitals プログラムを作成したGoogleの従業員によって共同執筆されています。この章とその基礎となる分析は、Googleと無関係の他の人々によってレビューされ、承認されました。*

## Core Web Vitals

CWVがGoogle Searchのランキングシグナルとして追加されてから1年が経ちましたので、このプログラムがウェブ上のユーザーエクスペリエンスにどのような影響を与えたかを見てみましょう。

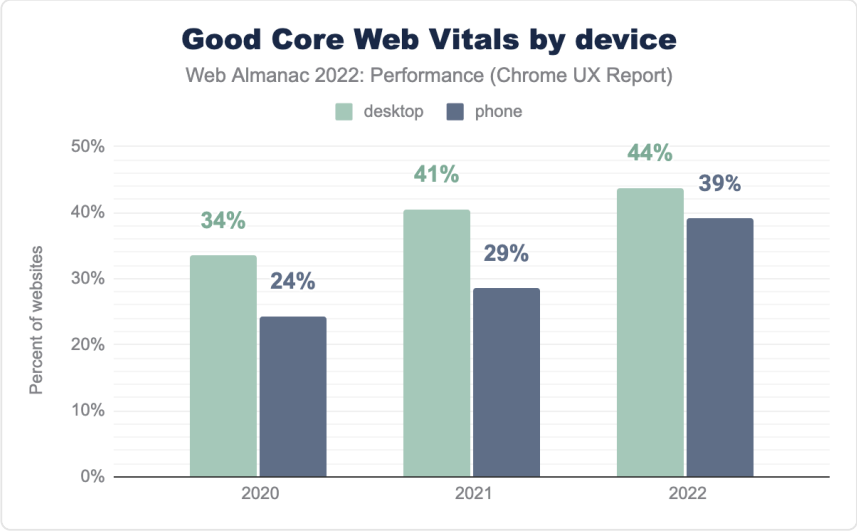



*図12.1. デバイスと年によって分けられた良好なCWVを持つウェブサイトの割合。*

2021年には、モバイルユーザー向けのウェブサイトの29%が良好なCWVと評価されました。これは2020年からの大幅なステップアップで、5ポイントの増加を表しています。しかし、2022年の進歩はさらに大きな前進で、モバイルで良好なCWVを持つウェブサイトは39%になり、さらに10ポイントの増加を表しました！

デスクトップで良好なCWVを持つウェブサイトは44%です。これはモバイルよりも良いですが、デスクトップ体験の改善率はモバイルほど速くはないため、ギャップは縮まっていま

す。

モバイル体験がデスクトップよりも悪い理由にはいくつかの説明があります。ポケットサイズのコンピューターの携帯性は大きな便利さですが、ユーザーエクスペリエンスに悪影響を及ぼす可能性があります。モバイルウェブ章で説明されているように、小さなフォームファクターは搭載できる処理能力の量に影響を与え、これはさらに、より強力な電話を所有するための高コストによって制約されます。処理能力が低いデバイスは、ウェブページをレンダリングするために必要な計算を実行するのにより長い時間がかかります。これらのデバイスの携帯性は、ウェブサイトのロード速度に影響を与える貧弱な接続性のエリアに持ち込むことができることも意味します。最後に、開発者がウェブサイトを構築する方法についての考慮事項があります。ページのモバイルフレンドリーなバージョンを作成するのではなく、一部のウェブサイトはデスクトップサイズの画像や不必要な量のスクリプト機能を提供しているかもしれません。これらのことはすべて、モバイルユーザーをデスクトップユーザーと比較して不利にする可能性があり、彼らのCWVパフォーマンスが低いことを説明するのに役立つかもしれません。

2022年には、2021年と比較して、はるかに多くのウェブサイトが良好なCWVと評価されました。しかし、その改善はウェブ全体でどれほど均等に分配されましたか？

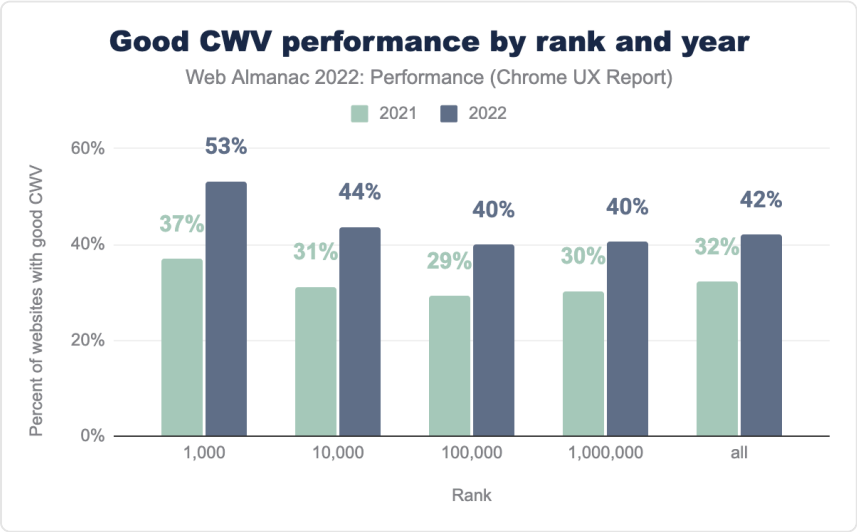



*図12.2. ランクと年によって分けられた良好なCWVを持つウェブサイトの割合。*

私たちは、相対的な人気（ランク）と年によってサイトを区分しましたが、デスクトップとモバイルの区別はしませんでした。興味深いことに、どうやら今年は人気のあるサイトに関係なく、全体的にウェブサイトがよりパフォーマントになったようです。もっとも人気のあるトップ1,000のウェブサイトがもっとも顕著に改善され、53%に16ポイントの増加を遂げ

ましたが、すべてのランクが10ポイント以上改善されました。人気のあるウェブサイトは、もっとも良いCWV体験を持っている傾向がありますが、これは、彼らがより大きなエンジニアリングチームと予算を持っていると仮定するとあまり驚くことではありません。

今年、モバイル体験がこれほど改善された理由をより深く理解するために、個々のCWV指標についてさらに詳しく掘り下げてみましょう。

## 最大のコンテントフルペイント（LCP）

最大のコンテントフルペイント[424]（LCP）は、ナビゲーションの開始からビューポート内で最大のコンテンツブロックが表示されるまでの時間です。この指標は、ユーザーがもっとも意味のあるコンテンツをどれだけ早く見ることができるかを表しています。

ウェブサイトのLCPが良好であるとは、すべてのページビューの少なくとも75%が2,500ミリ秒よりも速い場合を指します。3つのCWV指標の中で、LCPの合格率はもっとも低く、良好なCWV評価を達成する上でのボトルネックとなることがよくあります。

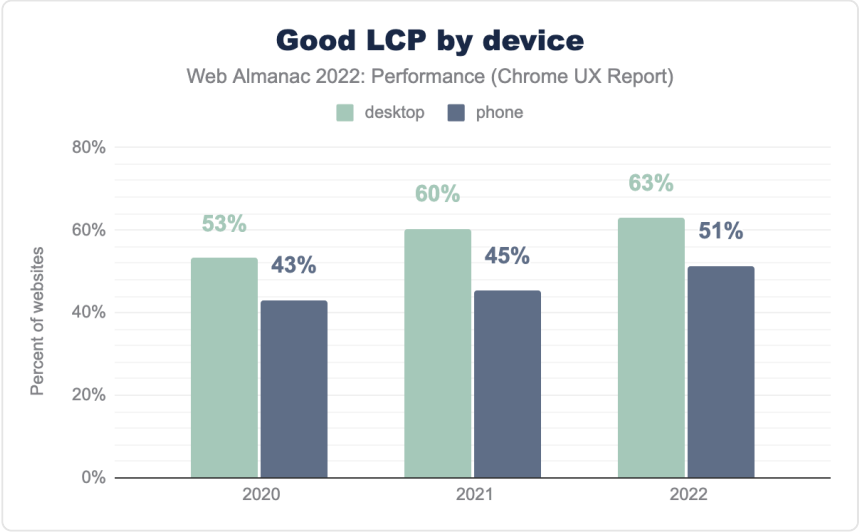



*図12.3. デバイスと年に分けられた良好なLCPを持つウェブサイトの割合。*

今年、モバイルで良好なLCP体験を提供するウェブサイトは51%で、デスクトップでは63%です。LCPは、ウェブサイトのモバイル体験が2022年に大幅に改善された主な理由の1つであり、今年は6ポイントの改善がありました。

424. https://web.dev/articles/lcp?hl=ja

なぜLCPは今年これほど改善されたのでしょうか？それを理解するために、いくつかのローディングパフォーマンス診断指標、TTFBとFCPを探ってみましょう。

## 最初のバイトまでの時間（TTFB）

最初のバイトまでの時間[425]（TTFB）は、ナビゲーションの開始からクライアントに返される最初のデータバイトまでの時間です。これはウェブパフォーマンスチェックリストの最初のステップであり、とくにネットワーク接続速度とサーバー応答時間に関して、LCPパフォーマンスのバックエンドコンポーネントを表しています。

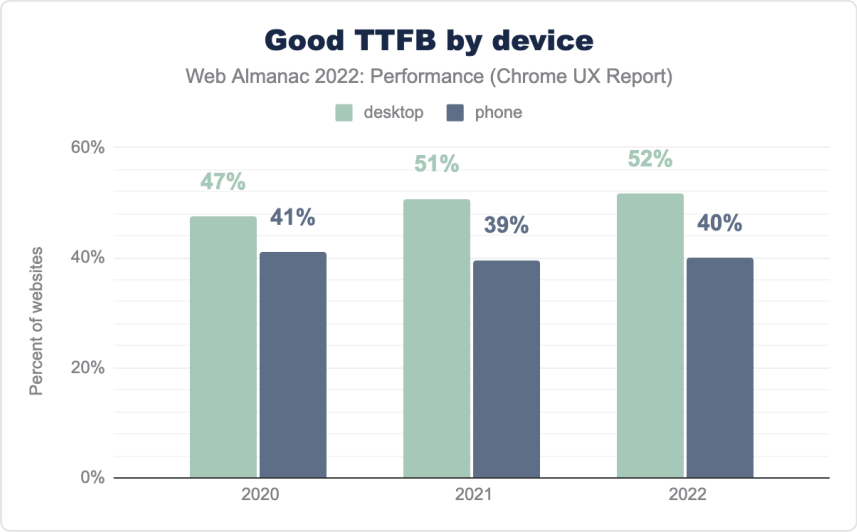



*図12.4. デバイスと年に分けられた良好なTTFBを持つウェブサイトの割合。*

余談ですが、Chromeは今年初めに「良好」なTTFBの閾値を500ミリ秒から800ミリ秒に変更[426]しました。上のグラフでは、比較目的でこの新しい閾値を使用して過去のデータすべてが示されています。

これを念頭に置くと、良好なTTFBを持つウェブサイトの割合は実際にはあまり改善していません。過去1年間で、ウェブサイトのデスクトップとモバイル体験は1ポイント改善されたに過ぎず、これは良いことですが、LCPに観察された利得を説明するものではありません。「必要な改善」と「不十分」のTTFB分布の両端に改善がなかったわけではありませんが、「良好」な端がもっとも重要です。

425. https://web.dev/articles/ttfb
426. https://developer.chrome.com/docs/crux/release-notes#202204

もう1つの複雑な点は、TTFBがCrUXで依然として実験的[427]な指標とみなされていることです。CrUXのドキュメントによると、TTFBは、プリレンダリングや戻る/進むナビゲーションなど、より高度なナビゲーションタイプを考慮に入れていません。これはある種の盲点であり、これらの領域で改善があった場合、それらは必ずしもTTFBの結果に影響を与えるとは限りません。

## 視覚コンテンツの初期表示時間（FCP）

視覚コンテンツの初期表示時間[428]（FCP）は、リクエストの開始から画面に最初の意味のあるコンテンツが描画されるまでの時間です。TTFBと同様に、この指標はレンダリングをブロックするコンテンツの影響を受けることがあります。「良好」なFCPの閾値は1,800ミリ秒です。

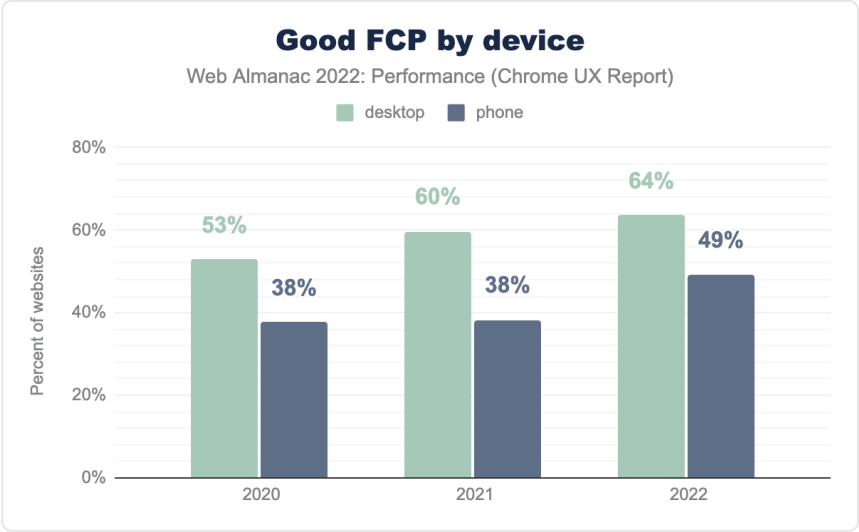



*図12.5. デバイスと年に分けられた良好なFCPを持つウェブサイトの割合。*

FCPは今年、劇的に改善されました。モバイルウェブサイトの49%が良好な体験を提供し、デスクトップでは64%です。これは、モバイルとデスクトップそれぞれについて、11ポイントと4ポイントの増加を表しています。

TTFBのデータが反対のことを示していない限り、これはフロントエンドの最適化、たとえばレンダリングをブロックするリソースの削除やリソースの優先順位付けの改善など、大幅な改善があったことを示しています。しかし、以下のセクションで見るように、今年のLCP

427. https://developer.chrome.com/docs/crux/methodology#experimental-metrics
428. https://web.dev/articles/fcp?hl=ja

の改善に感謝すべきはまったく別の何かかもしれません。

## LCPメタデータとベストプラクティス

これらのパフォーマンスの改善は、実際にはウェブサイト自体の変更によるものではないかもしれません。ネットワークインフラ、オペレーティングシステム、またはブラウザの変更も、このようにウェブスケールでLCPパフォーマンスに影響を与える可能性があります。そこで、いくつかのヒューリスティックに深く掘り下げてみましょう。

### レンダリングをブロックするリソース

ページにレンダリングをブロックするリソースがあるとは、リソースがページの初期描画（またはレンダリング）を遅らせる場合を指します。これは、ネットワーク経由でロードされる重要なスクリプトやスタイルにとくに当てはまります。Lighthouseには、これらのリソースをチェックする監査[429]が含まれており、CrUXの各ウェブサイトのホームページで実行しました。これらのページをどのようにテストしているかについては、Methodologyで詳しく学ぶことができます。

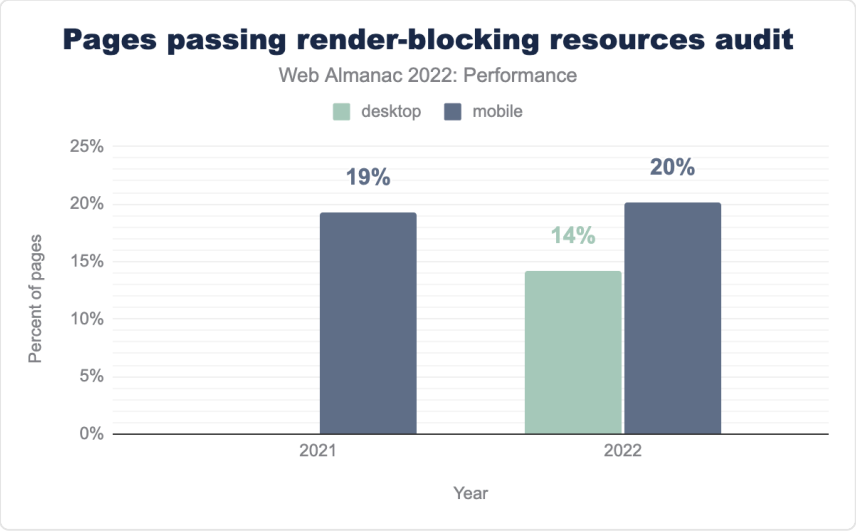



*図12.6. レンダリングをブロックするLighthouse監査に合格したページの割合、デバイスと年別。*

驚くべきことに、レンダリングをブロックするリソースを持つページの割合に劇的な改善は

429. https://developer.chrome.com/docs/lighthouse/performancerender-blocking-resources?hl=ja

ありませんでした。モバイルページのわずか20%が監査に合格しており、これは昨年からわずか1ポイントの増加に過ぎません。

2022年は、デスクトップのLighthouseデータを持つ最初の年です。したがって、以前の年と比較することはできませんが、デスクトップページがLCPとFCPのパフォーマンスがより良い傾向にあるにもかかわらず、モバイルに比べて監査に合格するデスクトップページがはるかに少ないことは興味深いことです。

### LCPコンテンツタイプ

LCP要素は、画像、見出し、またはテキストの段落など、さまざまなタイプのコンテンツである可能性があります。

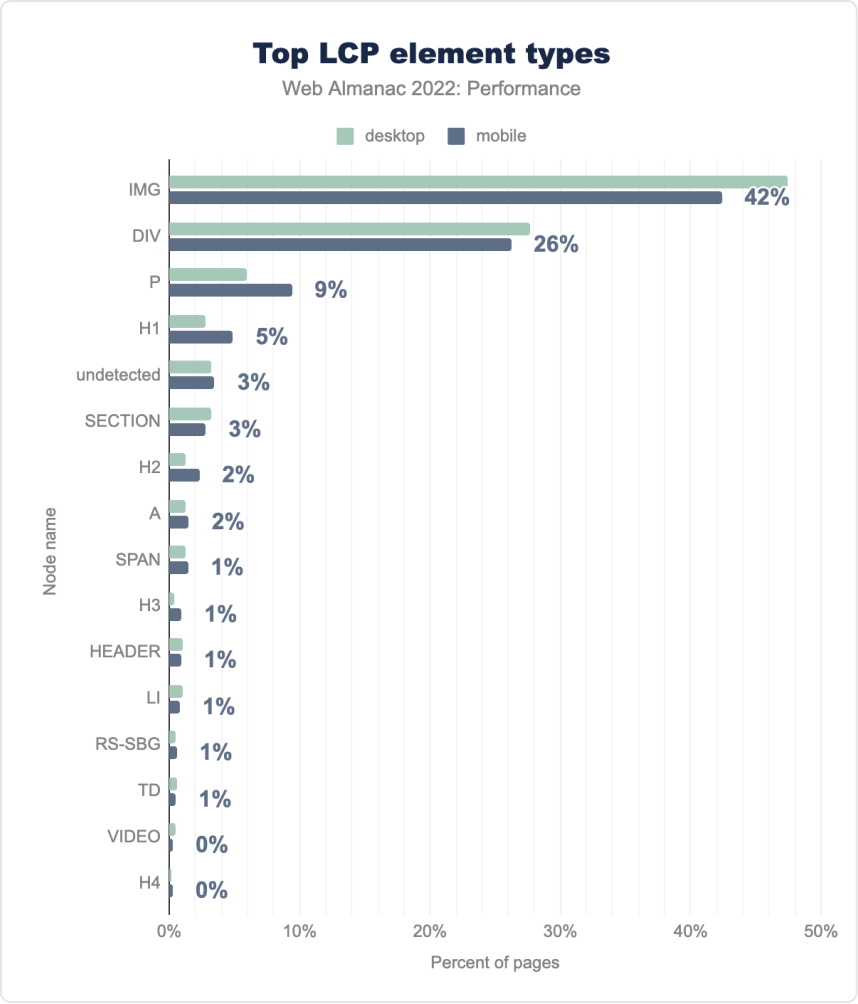



*図12.7. LCP要素として特定の要素を持つページの割合。*

画像はもっとも一般的なLCPコンテンツタイプであり、`img`要素はモバイルページの42%でLCPを表しています。モバイルページでは、見出しや段落の要素がLCPとなる可能性がデスクトップページよりもわずかに高く、デスクトップページでは画像要素がLCPとなる可能性が高くなっています。1つの可能な説明は、とくに縦向きのモバイルレイアウトでは、レスポンシブでない画像が小さく表示されるため、見出しや段落のような大きなテキストブロックがLCP要素になるというものです。

2番目に人気のあるLCP要素タイプは`div`です。これは、テキストや背景画像のスタイリングに使用される汎用的なHTMLコンテナーです。これらの要素が画像やテキストを含んでい

る頻度を詳しく評価するために、LCP API[430]の`url`プロパティを評価できます。仕様[431]によると、このプロパティが設定されている場合、LCPコンテンツは画像でなければなりません。

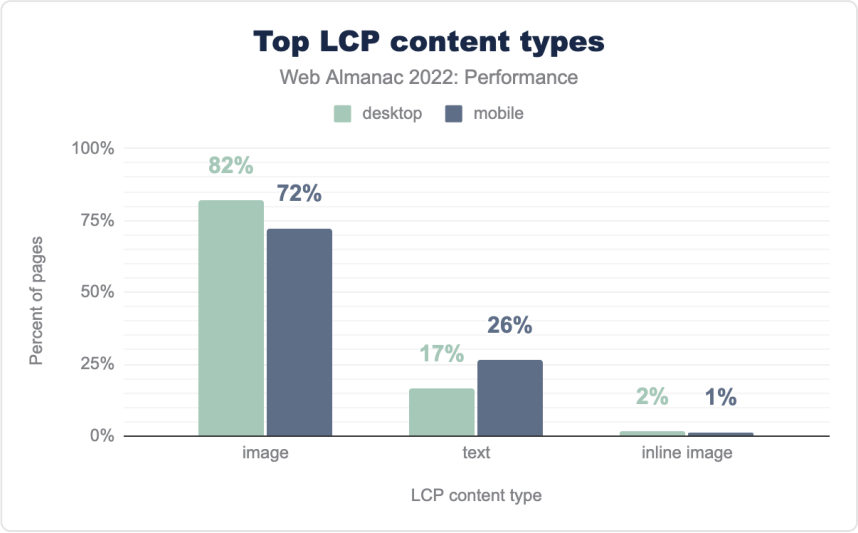



*図12.8. LCPコンテンツのタイプごとのページの割合。*

モバイルページの72%、デスクトップページの82%が画像をLCPとしています。たとえば、これらの画像は従来の`img`要素やCSSの背景画像である可能性があります。これは、前の図で見た`div`要素の大部分も画像であることを示唆しています。モバイルページの26%、デスクトップページの17%がテキストベースのコンテンツをLCPとしています。

ページの1%が実際にインライン画像をLCPコンテンツとして使用しています。これは、キャッシュや複雑さを中心に、さまざまな理由でほとんど常に悪いアイデアです。

### LCPの優先順位付け

HTMLドキュメントがロードされた後、LCPリソース自体をいかに早くロードできるかに影響を与える2つの主要な要因があります：発見可能性と優先順位付け。LCPの発見可能性については後で探りますが、まずLCP画像がどのように優先されるかを見てみましょう。

画像はデフォルトでは高優先度でロードされませんが、新しい優先度のヒント[432] APIのおかげで、開発者はLCP画像を高優先度でロードするように明示的に設定し、非必須リソースよりも優先させることができます。

430. *https://developer.mozilla.org/docs/Web/API/LargestContentfulPaint*
431. *https://www.w3.org/TR/largest-contentful-paint/#dom-largestcontentfulpaint-url*
432. *https://web.dev/articles/fetch-priority?hl=ja*


*図12.9. LCP要素に `fetchpriority=high` を使用しているページの割合。*

0.03%のページがLCP要素に `fetchpriority=high` を使用しています。逆効果的に、いくつかのページは実際にLCP画像の優先度を下げています：モバイルで77ページ、デスクトップで104ページです。

`fetchpriority` はまだ非常に新しく、どこでもサポートされているわけではありませんが、ほとんどの開発者のツールボックスになぜ含まれていないのかについてはほとんど理由がありません。APIを開発したPatrick Meenan[433]は、実装が相対的に簡単であることに対して潜在的な改善がどれほど大きいかを考えると、それを「チートコード」と表現しています[434]。

### LCPの静的発見可能性

LCP画像が早期に発見されることは、ブラウザができるだけ早くそれをロードするための鍵です。上で議論したprioritizationの改善でさえ、ブラウザが後でリソースをロードする必要があることを知らない場合には役立ちません。

LCP画像が*静的に発見可能*であるとは、そのソースURLがサーバーから送信されたマークアップから直接解析できる場合を指します。この定義には、 `picture` または `img` 要素内で定義されたソースおよび明示的にプリロードされたソースが含まれます。

1つの注意点は、この定義に基づくと、テキストベースのLCPコンテンツは常に静的に発見可能であるということです。しかし、テキストベースのコンテンツはクライアントサイドレンダリングやWebフォントに依存することがあるため、これらの結果を下限として考えてください。

```
<img data-src="kitten.jpg" class="lazyload">
```

カスタムの遅延読み込み技術のような上記の例は、画像が静的に発見可能でないようにする方法の1つです。これは、 `src` 属性を更新するためにJavaScriptに依存しているためです。クライアントサイドのレンダリングもLCPコンテンツを隠す可能性があります。

433. *https://twitter.com/patmeenan*
434. *https://twitter.com/patmeenan/status/1460276602479251457*

# 39%

*図12.10. LCP要素が静的に発見可能でなかったモバイルページの割合。*

モバイルページの39%が静的に発見可能でないLCP要素を持っています。この数字はデスクトップでは44%のページでさらに悪いです。これは、前のセクションの発見によるものかもしれません。それによると、LCPコンテンツがデスクトップページで画像である可能性が高いです。

これは、この指標を見る最初の年なので比較のための歴史的データはありませんが、これらの結果は、LCPリソースのロード遅延を改善する大きな機会があることを示唆しています。

### LCPのプリロード

LCP画像が静的に発見可能でない場合、プリロード[435]は、ロード遅延を最小限に抑える効果的な方法です。もちろん、リソースが最初から静的に発見可能であればより良いですが、それに対処するにはページのロード方法を複雑に再構築する必要があります。プリロードは比較的簡単な修正方法であり、単一のHTTPヘッダーまたは`meta`タグで実装できます。

# 0.56%

*図12.11. LCP画像をプリロードするモバイルページの割合。*

モバイルページの約200分の1だけがLCP画像をプリロードしています。この数字は、LCP画像が静的に発見可能でないページのみを考慮すると約400分の1（0.25%）に低下します。

静的に発見可能な画像をプリロードすることは過剰かもしれませんが、ブラウザはプリロードスキャナー[436]のおかげですでに画像について知っているはずです。しかし、プリロードはHTML内で早期に発見される他の静的に発見可能な画像よりも、重要な画像を早くロードするのに役立ちます。たとえば、ヘッダー画像やメガメニューの画像などです。これは、`fetchpriority`をサポートしていないブラウザ[437]にとくに当てはまります。

これらの結果は、圧倒的多数のウェブがLCP画像をより発見しやすくすることで恩恵を受ける可能性があることを示しています。LCP画像をより早くロードすること、つまり静的に発見可能にするかプリロードすることは、LCPパフォーマンスを大幅に改善するのに役立ちま

435. https://web.dev/articles/preload-critical-assets?hl=ja
436. https://web.dev/articles/preload-scanner?hl=ja
437. https://caniuse.com/?search=fetchpriority

す。しかし、パフォーマンスに関連するすべてのことと同様に、常に実験を行い、特定のサイトに最適なものが何かを理解してください。

### LCPイニシエーター

LCPリソースが静的に発見可能でない場合、それが発見されるためには、もっと複雑なプロセスが必要です。

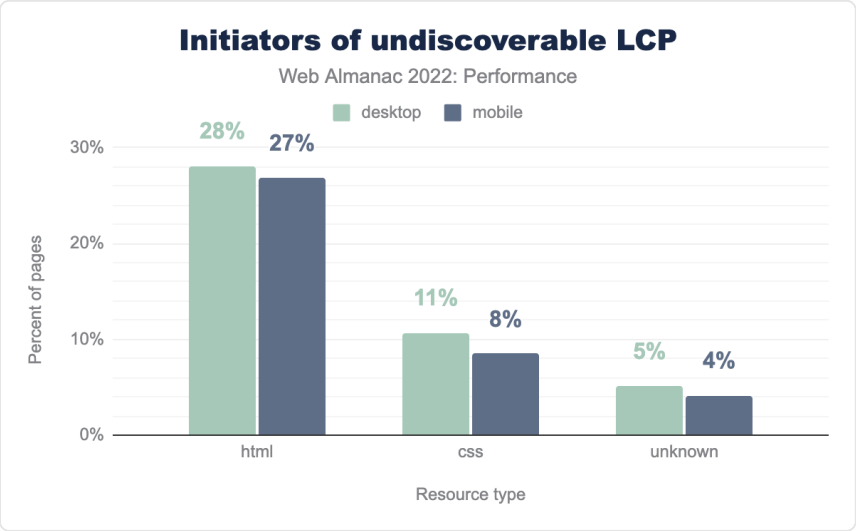



*図12.12. LCPが静的に発見可能でないページで、特定のリソースから開始された割合。*

モバイルページの27%が、プリロードスキャナーがすでに実行された後にHTML内で発見されたLCP画像を持っています。これは通常、スクリプトベースの遅延読み込みやクライアントサイドレンダリングによるものです。

モバイルページの8%は、外部スタイルシートに依存してLCPリソースを持っています。たとえば、`background-image`プロパティを使用します。これにより、リソースのクリティカルリクエストチェーンにリンクが追加され、スタイルシートがクロスオリジンでロードされる場合、LCPパフォーマンスがさらに複雑になる可能性があります。

モバイルページの4%が、私たちが検出できない未発見のLCPイニシエーターを持っています。これらはHTMLとCSSのイニシエーターの組み合わせかもしれません。

スクリプトとスタイルの両方に基づく発見可能性の問題はパフォーマンスに悪いですが、プリロードでその影響を緩和できます。

### LCP遅延読み込み

Lazy-loadingは、非重要リソースの読み込みを遅らせる（通常はビューポート内または近くになるまで）効果的なパフォーマンス技術です。貴重な帯域幅とレンダリングパワーを解放することで、ブラウザはページ読み込みの早い段階で重要コンテンツを読み込むことができます。問題は、Lazy-loadingをLCPイメージ自体に適用した場合、それがブラウザがそれをはるかに遅くまで読み込むことを防ぐことになる点です。


*図12.13. `loading=lazy` を設定しているモバイルページの割合*

`img` ベースのLCPを持つページの約10分の1が、ネイティブの `loading=lazy` 属性を使用しています。技術的にこれらの画像は静的に発見可能ですが、ブラウザはページをレイアウトしてビューポート内にあるかどうかを知るまで、それらの読み込みを開始するのを待つ必要があります。定義上、LCPイメージは常にビューポート内にありますので、実際にはこれらのイメージがLazy-loadedされるべきではありませんでした。LCPがビューポートのサイズやディープリンクナビゲーションからの初期スクロール位置によって異なるページでは、LCP候補を積極的に読み込むことで全体的なパフォーマンスが向上するかどうかをテストする価値があります。

このセクションのパーセンテージは、すべてのページではなく、 `img` 要素がLCPであるページのみに基づいています。参考までに、これはページの42%を占めることを思い出してください。


*図12.14. カスタム遅延ロードを使用している画像ベースのLCPを持つモバイルページの割合。*

先に示したように、サイトがネイティブの遅延ロード動作をポリフィルする方法の1つは、画像ソースを `data-src` 属性に割り当て、クラスリストに `lazyload` のような識別子を含めることです。その後、スクリプトがこのクラス名を持つ画像のビューポートに対する位置を監視し、 `data-src` の値をネイティブの `src` 値に切り替えて画像のロードを開始させます。

`img` ベースのLCPを持つページの8.8%がこの種のカスタム遅延ロード動作を使用しており、ネイティブの遅延ロードとほぼ同じ割合です。

LCP画像の遅延ロードの悪影響を超えて、ネイティブ画像の遅延ロードはすべての主要ブラウザによってサポートされています[438]ので、カスタムソリューションは不必要なオーバーヘッドを追加している可能性があります。私たちの意見では、カスタムソリューションが画像のロードタイミングをより細かく制御することができるかもしれませんが、開発者はこれらの余分なポリフィルを取り除き、ユーザーエージェントのネイティブな遅延ロードのヒューリスティクスに委ねるべきです。

ネイティブの遅延ロードを使用するもう1つの利点は、ChromeのようなブラウザがLCP候補である可能性が高い属性を無視するヒューリスティクスを使用することを実験している[439]ことです。これはネイティブの遅延ロードでのみ可能であり、カスタムソリューションはこの場合、改善から恩恵を受けることはありません。




*図12.15. ネイティブまたはカスタムの遅延ロードを使用している画像ベースのLCPを持つモバイルページの割合。*

どちらの技術を使用しているページを見ると、`img`ベースのLCPを持つページの18%がもっとも重要な画像のロードを不必要に遅延させています。

正しく使用された場合、遅延ロードは良いことですが、これらの統計はLCP画像からこの機能を取り除くことでパフォーマンスを大幅に改善する大きな機会があることを強く示唆しています。

WordPressはネイティブの遅延ロード採用の先駆者の1つであり、バージョン5.5から5.9の間、実際にはLCP候補から属性を省略しませんでした。したがって、WordPressがこのアンチパターンにどの程度まだ貢献しているかを探りましょう。




*図12.16. LCP画像にネイティブの遅延ロードを使用しているモバイルページでWordPressを使用している割合。*

CMSの章によると、35%のページがWordPressを使用しています。そのため、2022年1月にバージョン5.9で修正が利用可能になって以来、LCP画像にネイティブの遅延ロードを使用しているページの72%がWordPressを使用していることは驚きです。さらなる調査が必要な1

438. *https://caniuse.com/loading-lazy-attr*
439. *https://bugs.chromium.org/p/chromium/issues/detail?id=996963*

つの理論は、プラグインがWordPressコアに組み込まれたセーフガードを迂回して、ページに遅延ロード動作を持つLCP画像を注入している可能性があるということです。

**54%**

*図12.17. LCP画像にカスタム遅延ロードを使用しているモバイルページでWordPressを使用している割合。*

同様に、カスタム遅延ロードを使用しているページの割合がWordPressで作成されたページでは54%と不釣り合いに高いです。これはWordPressエコシステム内で遅延ロードの過剰使用に関するより広範な問題を示唆しています。これはWordPressコアに局所化された修正可能なバグではなく、このアンチパターンに貢献している何百もの別々のテーマやプラグインがある可能性があります。

### LCPサイズ

LCPリソースをロードする時間に大きく影響する要因の1つが、そのワイヤー上のサイズです。より大きなリソースは自然とロードに時間がかかります。では、画像ベースのLCPリソースはどれくらいの大きさなのでしょうか？

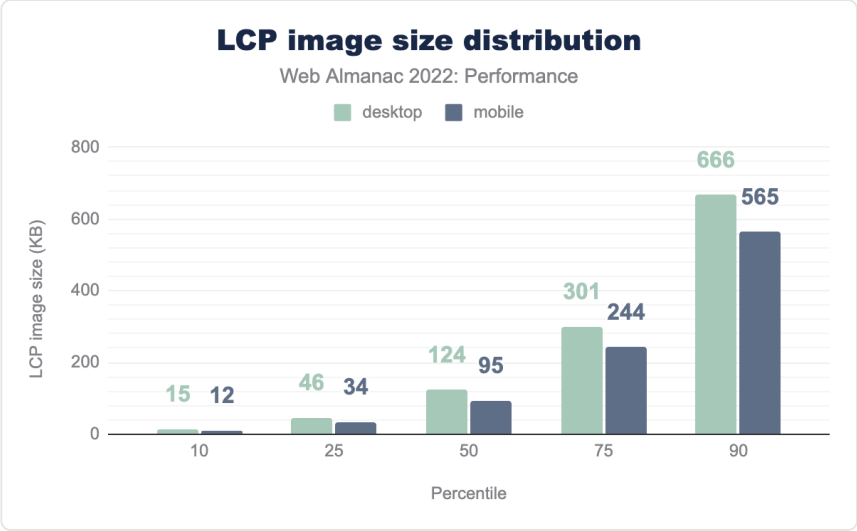



*図12.18. 画像ベースのLCPリソースのサイズ分布。*

モバイルの中央値のLCP画像は95KBです。正直、もっと悪いと予想していました！

デスクトップページは、分布全体でより大きなLCP画像を持つ傾向にあり、中央値のサイズは124KBです。

**114,285 KB**

*図12.19. 最大のLCP画像のサイズ。*

また、最大のLCP画像サイズを調査し、デスクトップで68,607KB、モバイルで114,285KBの画像を発見しました。これらの極端に大きな外れ値を見るのはおもしろいかもしれませんが、これらが実際のユーザーによって訪れられるアクティブなウェブサイトであるという不幸な現実を忘れないようにしましょう。データは常に無料ではなく、これらのようなパフォーマンスの問題は、計量されたモバイルデータプランのユーザーにとってアクセシビリティの問題になり始めます。これらは、これらのような明らかに過大な画像をロードするためにムダにされるエネルギーを考えると、サステナビリティの問題でもあります。

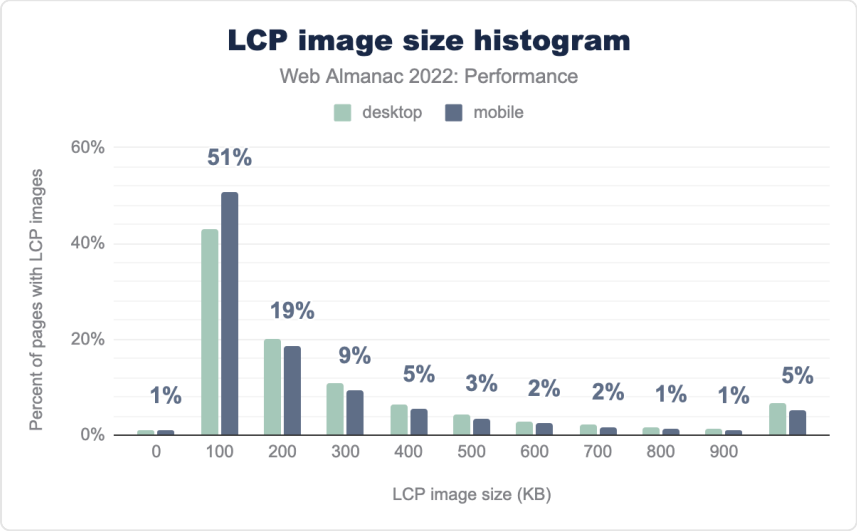



*図12.20. 画像ベースのLCPサイズのヒストグラム。*

異なる視点から見ると、上記の図は100KB刻みのヒストグラムとして分布を示しています。この視点からは、LCP画像サイズが主に200KB未満の範囲に落ちることがより明確に見えます。また、モバイルのLCP画像の5%が1,000KBを超えるサイズであることがわかります。

LCP画像が理想的にどれくらいの大きさであるべきかは、多くの要因に依存します。しかし、20のウェブサイトのうち1つが、私たちの360px幅のモバイルビューポートにメガバイ

トサイズの画像を提供しているという事実は、サイトの所有者がレスポンシブ画像[440]を採用する必要があることを明確に示しています。このトピックに関するさらなる分析については、メディアおよびモバイルウェブの章を参照してください。

### LCPフォーマット

LCP画像のフォーマット選択は、そのバイトサイズに大きな影響を及ぼし、最終的にはそのローディングパフォーマンスに影響します。WebPとAVIFは、JPG（またはJPEG）やPNGといった従来のフォーマットよりも効率的であることがわかっている比較的新しいフォーマットです。

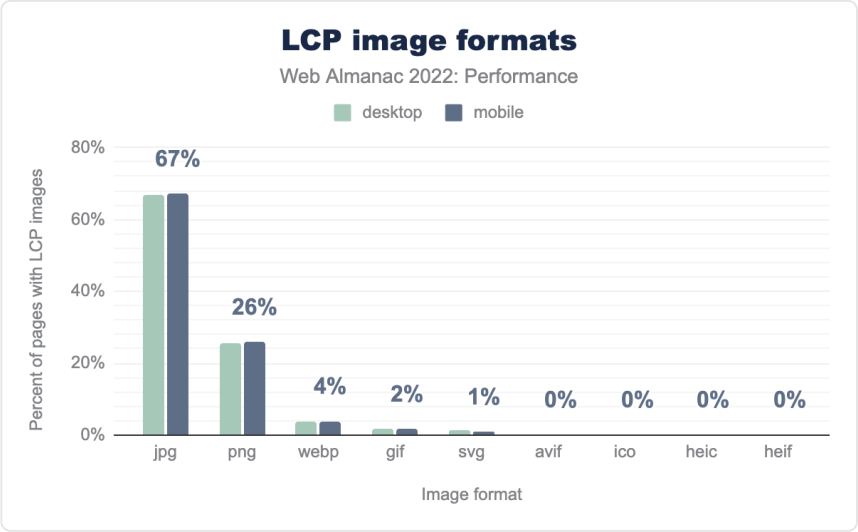



*図12.21. LCP画像に使用されるフォーマットのページ割合。*

メディアの章によると、JPGフォーマットはモバイルページでロードされるすべての画像の約40%を占めます。しかし、モバイルのすべてのLCP画像の67%をJPGが占めており、これは26%のPNGよりも2.5倍一般的です。これらの結果は、ページがLCPリソースとしてデジタルアートワークではなく写真品質の画像を使用する傾向があることを示唆しているかもしれません。なぜなら、写真はPNGと比較してJPGでよりよく圧縮される傾向があるからですが、これは単なる憶測です。

画像ベースのLCPを使用するページの4%がWebPを使用しています。これは画像効率にとって良いニュースですが、AVIFを使用しているのは1%未満です。AVIFはWebPよりもさらによく圧縮されるかもしれませんが、すべての現代のブラウザでサポートされていないため、

440. https://developer.mozilla.org/ja/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images

その採用率は低いです。一方、WebPはすべての現代のブラウザでサポートされているため、その低い採用率はLCP画像とそのパフォーマンスを最適化する大きな機会を示しています。

### LCP画像の最適化

前のセクションでは、LCPリソースに使用されるさまざまな画像フォーマットの人気を見てきました。開発者がLCPリソースを小さくし、より速くロードするための別の方法は、効率的な圧縮設定を利用することです。JPGフォーマットは、画質をあまり落とさずに不要なバイトを削減するために、損失圧縮できます。しかし、一部のJPG画像は十分に圧縮されていないかもしれません。

Lighthouseには、JPGを圧縮レベル85に設定した際のバイト節約を測定する監査[441]が含まれています。画像が結果として4KB以上小さくなる場合、監査は失敗し、最適化の機会と見なされます。

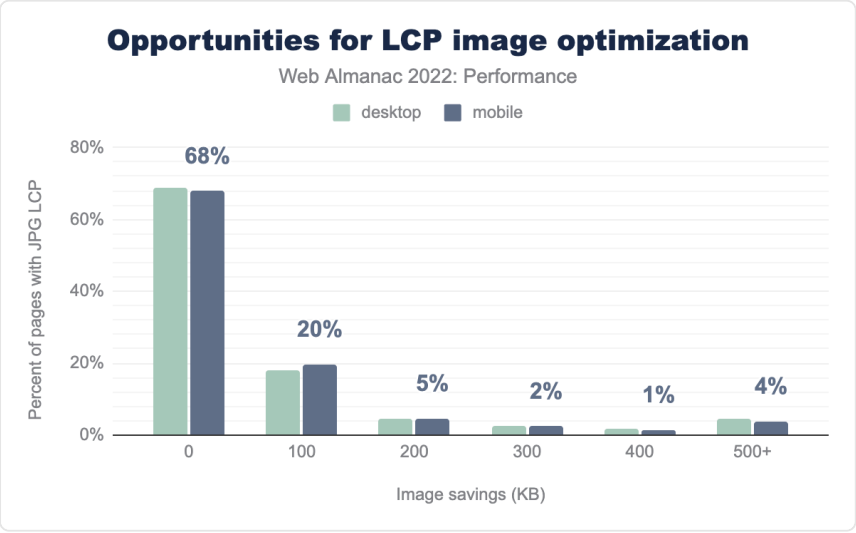



*図12.22. JPGベースのLCP画像のバイト節約のヒストグラム。*

Lighthouseによってフラグが立てられたJPGベースの LCP画像を持つページのうち、68%は損失圧縮のみでLCP画像を改善する機会がありません。これらの結果はやや驚きであり、多くの「ヒーロー」JPG画像が適切な品質設定を使用していることを示唆しています。それにもかかわらず、これらのページの20%は最大100KB、4%は500KB以上節約できます。LCP画像の大多数が200KB以下であることを思い出してください。これはかなりの節約です！

441. *https://developer.chrome.com/docs/lighthouse/performanceuses-optimized-images?hl=ja*

### LCPホスト

LCP画像自体のサイズと効率性に加えて、それがロードされるサーバーもそのパフォーマンスに影響を与える可能性があります。HTMLドキュメントと同じオリジンからLCP画像をロードする方が、オープンな接続を再利用できるため、通常は速くなります。

しかし、LCP画像はアセットドメインや画像CDNなど、他のオリジンからロードされることがあります。これが発生すると、追加の接続を設定する時間がLCPの許容時間から貴重な時間を奪うことになります。

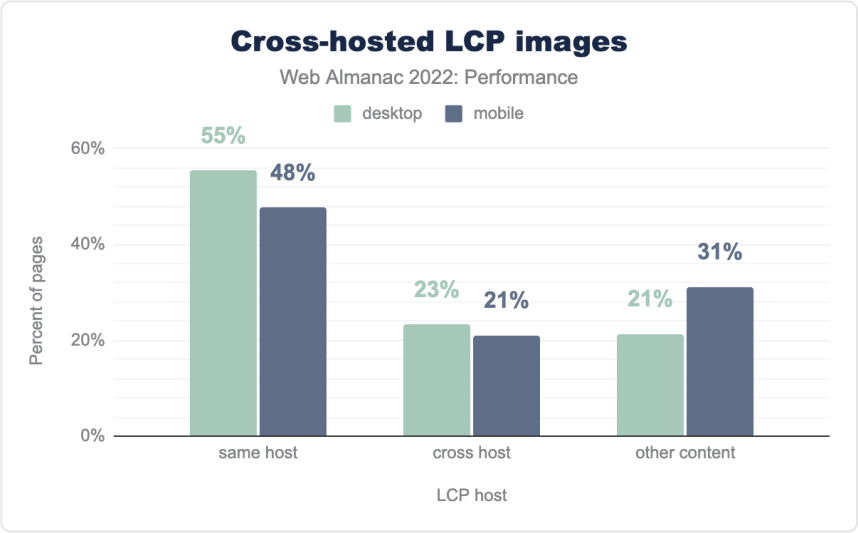



*図12.23. クロスホストされたLCP画像*

モバイルページの5分の1がLCP画像をクロスホストしています。これらのサードパーティホストへの接続を設定する時間は、LCP時間に不必要な遅延を追加する可能性があります。可能な限り、ドキュメントと同じオリジンにLCP画像を自己ホストすることがベストプラクティスです。リソースヒント[442]を使用してLCPオリジンに事前接続する、またはさらに良いことに、画像自体をプリロードできますが、これらの技術は非常に広く採用されていません。

### LCPの結論

LCPのパフォーマンスは、とくにモバイルユーザーにとって、今年大幅に改善しました。それがなぜ起こったのかについての決定的な答えはありませんが、上で提示されたデータはいくつかの手がかりを提供しています。

442. *https://almanac.httparchive.org/ja/2021/resource-hints*

これまでに見てきたことは、レンダリングをブロックするリソースがまだかなり普及しており、非常に少数のサイトが高度な優先順位付け技術を使用しており、LCP画像の3分の1以上が静的に発見できないということです。サイト所有者や大規模な出版プラットフォームが、これらのLCPパフォーマンスの側面を意図的に最適化していると示唆する具体的なデータがないため、OSやブラウザレベルでの最適化を見る他の場所があります。

2022年3月のChromiumブログ投稿[443]によると、Androidのローディングパフォーマンスは15%改善されました。投稿は詳細には触れていませんが、この改善は「*ブラウザのユーザーインターフェイススレッド上で重要なナビゲーションの瞬間を優先すること*」によるものとしています。これは、2022年にモバイルパフォーマンスがデスクトップパフォーマンスを上回った理由を説明するのに役立つかもしれません。

今年のLCPの6パーセンテージポイントの改善は、何十万ものウェブサイトのパフォーマンスが改善されたときにのみ起こり得ます。それがどのように起こったのかという魅力的な問いを脇に置いて、ウェブ上のユーザーエクスペリエンスが*改善されている*ことを一瞬、評価しましょう。それは困難な作業ですが、このような改善はエコシステムをより健全にし、祝う価値があります。

## 累積レイアウトシフト（CLS）

累積レイアウトシフト（CLS）は、画面上でコンテンツが予期せず動く量を表すレイアウト安定性メトリックです。ウェブサイトが良好なCLSを持っているとは、サイト全体のナビゲーションの少なくとも75%が0.1以下のスコアを持っている場合を言います。

443. *https://blog.chromium.org/2022/03/a-new-speed-milestone-for-chrome.html#:~:text=Chrome%20continues%20to%20get%20faster%20on%20Android%20as%20well.%20Loading%20a%20page%20now%20takes%2015%25%20less%20time%2C%20than*

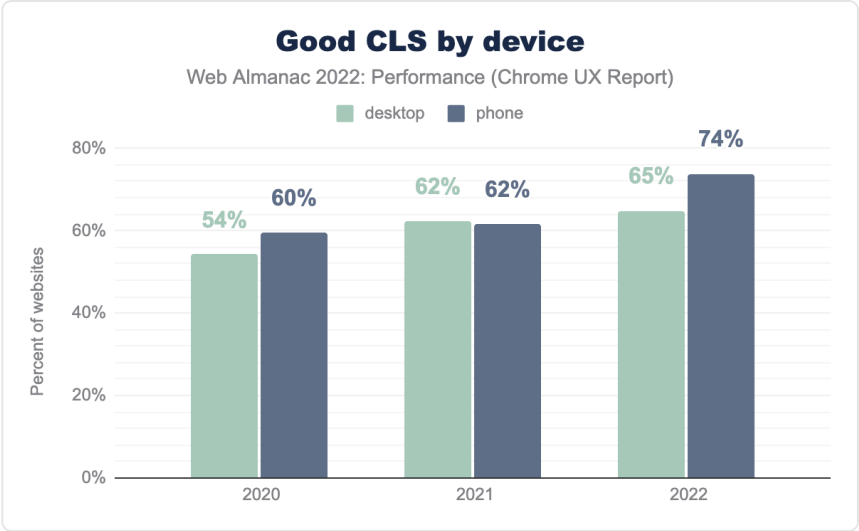



*図12.24. デバイス別の良好なCLS*

今年、モバイルデバイスでのウェブサイトの「良好な」CLSの割合は大幅に改善し、62%から74%へと向上しました。デスクトップでのCLSも3パーセンテージポイント向上し、65%になりました。

LCPがほとんどのサイトが全体的に良好なCWVと評価されるボトルネックである一方で、今年のモバイルCLSの大幅な改善がCWVの合格率にプラスの影響を与えたことは間違いありません。

モバイルCLSがこれほど大きく改善された理由は何だったのでしょうか？1つの可能性としては、2021年11月中旬にリリースされたChromeの新しく改善された戻る/進むキャッシュ[444]（bfcache）が挙げられます。この変更により、適格なページが戻ると進むナビゲーション中にメモリから綺麗に復元されるようになり、HTTPキャッシュからリソースをフェッチしたり、さらに悪いことにはネットワーク経由でフェッチしてページを再構築する「最初から始める」必要がなくなりました。

444. https://web.dev/articles/bfcache?hl=ja

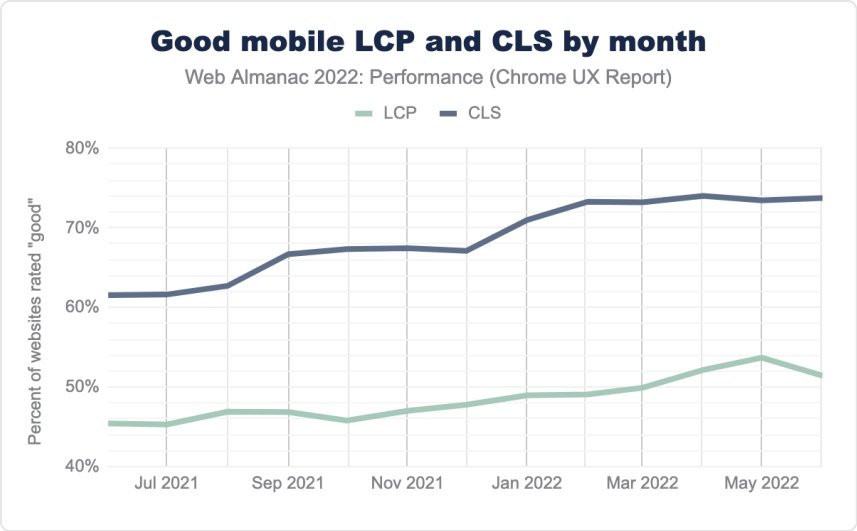



*図12.25. 良好なモバイルLCPとCLSを持つウェブサイトの月次時系列。*

このグラフは、月次の粒度でのLCPとCLSのパフォーマンスを示しています。2022年1月に始まる2か月間で、ChromeがbfcacheのアップデートをリリースしたCLSを持つウェブサイトの割合が以前よりもはるかに速く上昇し始めました。

しかし、bfcacheがCLSをこれほど改善した理由は何でしょうか？Chromeがページをメモリから即座に復元する方法により、そのレイアウトは確定しており、通常のローディング中に発生する初期の不安定さの影響を受けません。

LCP体験がそれほど劇的に改善されなかった理由の1つとして、戻る/進むナビゲーションは標準のHTTPキャッシングのおかげですでにかなり速かったためです。良好なLCPの閾値は2.5秒であり、任意のクリティカルリソースがすでにキャッシュにあると仮定すると、これはかなり寛大であり、「良好」な体験をさらに速くする追加ポイントはありません。

## CLSメタデータとベストプラクティス

CLSベストプラクティスにどれだけのウェブが準拠しているか探りましょう。

### 明示的な寸法

レイアウトシフトを回避するもっとも直接的な方法は、コンテンツのスペースを予約するために寸法を設定することです。たとえば、画像に`height`と`width`属性を使用するなどです。

# 72%

*図12.26. 少なくとも1つの画像に明示的な寸法を設定していないモバイルページの割合。*

モバイルページの72%にサイズが指定されていない画像があります。この統計だけでは全体像を示すことはできません。なぜなら、サイズが指定されていない画像が常にユーザーが認識するレイアウトシフトを引き起こすわけではないからです。たとえば、ビューポートの外でロードされる場合などです。それでも、サイト所有者がCLSベストプラクティスに十分に準拠していない可能性を示す兆候です。

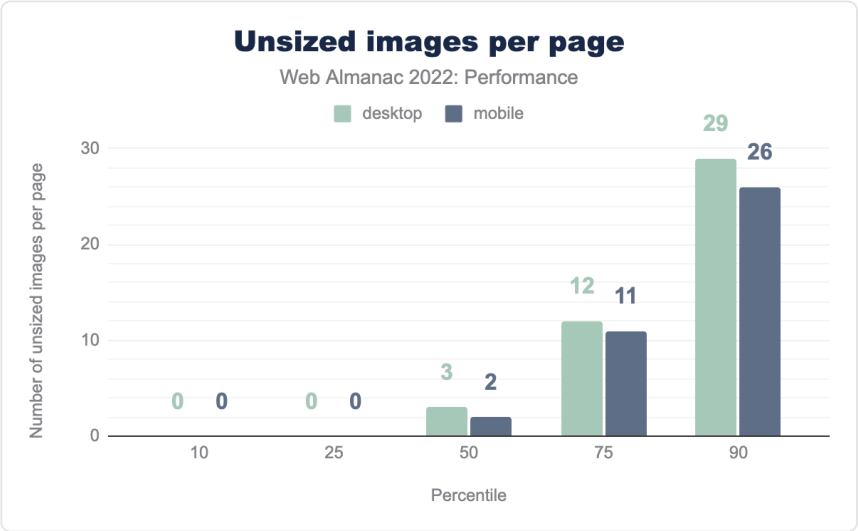



*図12.27. サイズが指定されていない画像のページごとの分布。*

中央値のウェブページには2枚のサイズが指定されていない画像があり、モバイルページの10%には少なくとも26枚のサイズが指定されていない画像があります。

ページ上にサイズが指定されていない画像があることはCLSにとってリスクですが、おそらくもっと重要な要因は画像のサイズです。大きな画像はより大きなレイアウトシフトに寄与し、CLSを悪化させます。

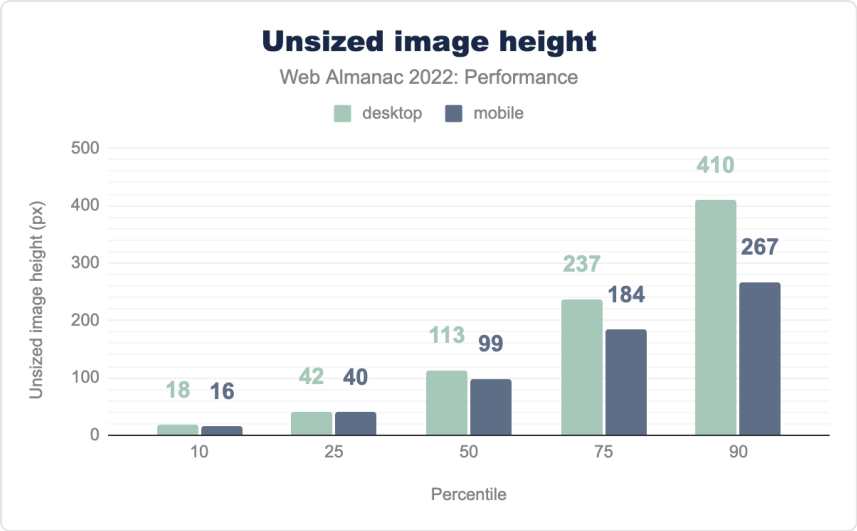



*図12.28. サイズが指定されていない画像の高さの分布。*

モバイルの中央値のサイズが指定されていない画像の高さは99ピクセルです。テストデバイスのモバイルビューポートの高さが512ピクセルであることを考えると、それはビューポート領域の約20%が下にシフトすることを意味します。仮にコンテンツの幅が全幅であるとします。その画像がロードされたときにビューポート内のどこにあるかによって、最大で0.2のレイアウトシフトスコア[445]を引き起こす可能性があり、「良好」なCLSの0.1の閾値を大幅に超えます。

デスクトップページでは、サイズが指定されていない画像がより大きい傾向があります。デスクトップの中央値のサイズが指定されていない画像の高さは113ピクセルで、90パーセンタイルでは高さが410ピクセルです。

**4,048,234,137,947,990,000px**

*図12.29. 最大のサイズが指定されていない画像の高さ。*

測定エラーであることを願うか、または深刻な間違いを犯したウェブ開発者であることを願うだけですが、私たちが見つけた最大のサイズが指定されていない画像は、信じられないほど4京ピクセルの高さがあります。その画像は非常に大きく、地球から月まで...300万回伸びることができます。それが何らかの1回限りの間違いであっても、次に大きいサイズが指定されていない画像でも高さが33,554,432ピクセルあります。いずれにせよ、それは大きなレイアウトシフトです。

445. *https://web.dev/articles/cls?hl=ja#layout-shift-score*

## アニメーション

一部の非合成[446] CSSアニメーションはページのレイアウトに影響を与え、CLSに貢献する可能性があります。ベストプラクティス[447]は、代わりに`transform`アニメーションを使用することです。

**38%**

*図12.30. 非合成アニメーションを持つモバイルページの割合。*

38%のモバイルページがこれらのレイアウトを変更するCSSアニメーションを使用し、CLSを悪化させるリスクを冒しています。使用されていない画像の問題と同様に、CLSにとってもっとも重要なのは、アニメーションがビューポートに対してどの程度レイアウトに影響を与えるかの度合いです。

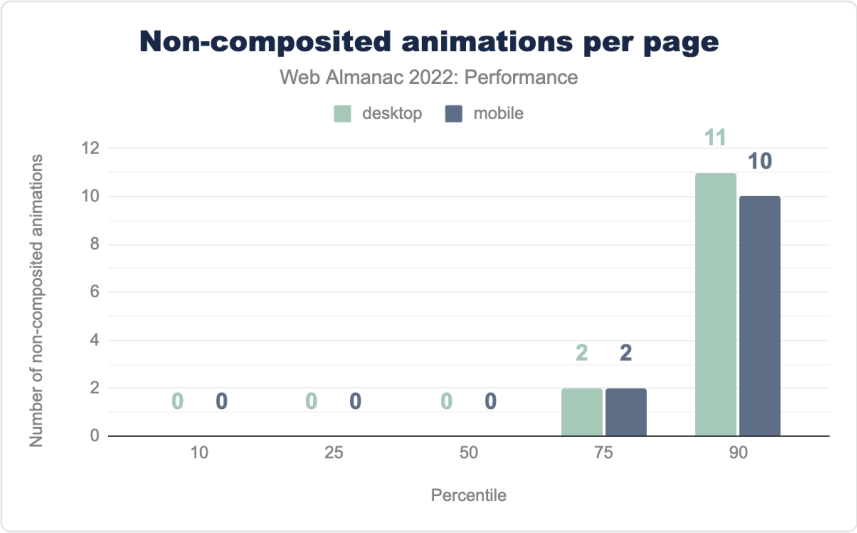



*図12.31. 非合成アニメーションのページごとの分布。*

上記の分布から、ほとんどのページはこのタイプのアニメーションを使用しておらず、使用しているページも控えめに使用していることがわかります。75パーセンタイルで、ページはこれらを2回使用します。

446. https://developer.chrome.com/docs/lighthouse/performancenon-composited-animations?hl=ja
447. https://web.dev/articles/optimize-cls?hl=ja#animations-%F0%9F%8F%83%E2%80%8D%E2%99%80%EF%B8%8F

## フォント

ページの読み込みプロセスではブラウザがWebフォントを発見し、リクエストし、ダウンロードし、適用するまでに時間がかかることがあります。これがすべて行われている間、テキストがすでにページ上にレンダリングされている可能性があります。Webフォントがまだ利用できない場合、ブラウザはシステムフォントでテキストをレンダリングすることをデフォルトとします。Webフォントが利用可能になったときに、システムフォントでレンダリングされた既存のテキストがWebフォントに切り替わると、レイアウトシフトが発生します。レイアウトシフトの量は、フォント同士の違いによって異なります。

**82%**

*図12.32. Webフォントを使用しているモバイルページの割合。*

ページの82%がWebフォントを使用しているため、このセクションはほとんどのサイトオーナーにとって非常に関連があります。

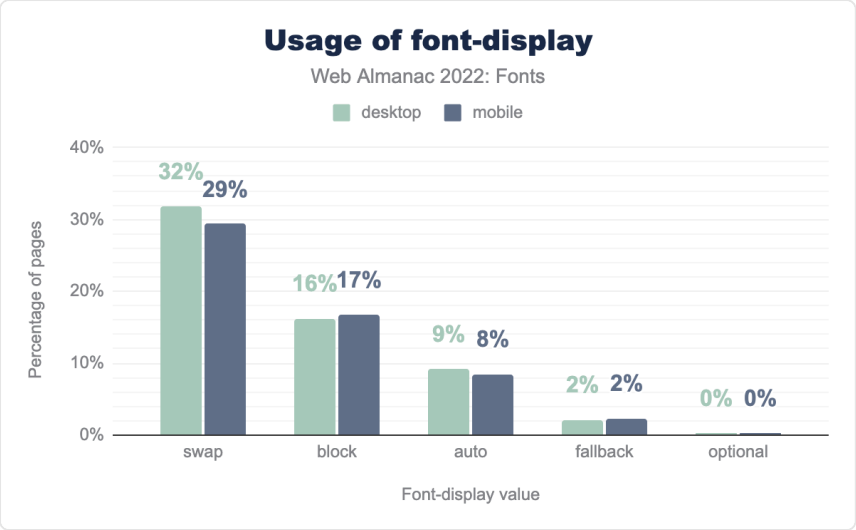



*図12.33. `font-display` 値の採用。*

フォントによるレイアウトシフトを回避する1つの方法は、`font-display: optional` を使用することで、これはシステムテキストがすでに表示された後にWebフォントを交換することは決してありません。しかし、フォントの章で指摘されているように、この指示を活用しているページは1%未満です。

`optional`はCLSには良いですが、UXのトレードオフがあります。サイトオーナーは、好ましいフォントをユーザーに表示できるということであれば、レイアウトの不安定さや目立つスタイル未適用テキストのフラッシュ（FOUT）を受け入れるかもしれません。

Webフォントを隠すのではなく、もう1つのCLSを軽減する戦略は、できるだけ早くそれらを読み込むことです。うまくいけば、システムテキストがレンダリングされる前にWebフォントが表示されるでしょう。

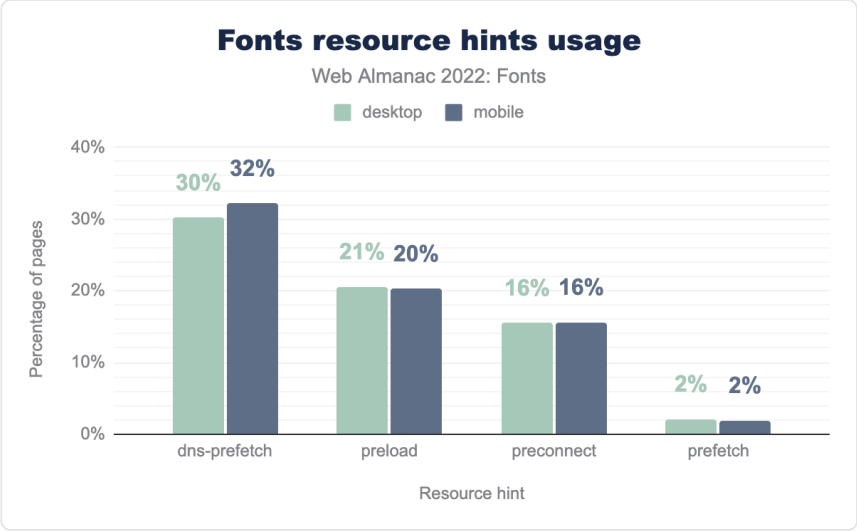



*図12.34. フォントリソースに対するリソースヒントの採用。*

フォントの章によると、モバイルページの20%がWebフォントをプリロードしています。フォントをプリロードする際の1つの課題は、Googleフォントのようなサービスを使用している場合、正確なURLが事前にわからないことがあります。フォントホストへのプリコネクトは、パフォーマンスにとって次善の選択肢ですが、その使用しているページは16%に過ぎず、DNSのプリフェッチを使用するページの半分です。

### bfcacheの適格性

CLSに対してbfcacheがどれほど影響力があるかを示しましたので、間接的なベストプラクティスとして適格性を考慮する価値があります。

特定のページがbfcacheに適格であるかどうかを確認する最良の方法は、DevToolsでテストする[448]ことです。残念ながら、100以上の適格性基準[449]があり、その多くはラボで測定するこ

448. https://web.dev/articles/bfcache?hl=ja#test-to-ensure-your-pages-are-cacheable
449. https://docs.google.com/spreadsheets/d/1li0po_ETJAIybpaSX5rW_IUN62upQhY0tH4pR5UPt60/edit?usp=sharing

とが難しい、または不可能です。したがって、bfcache適格性を全体として見るのではなく、より簡単に測定できるいくつかの基準を見て、適格性の下限を感じ取りましょう。

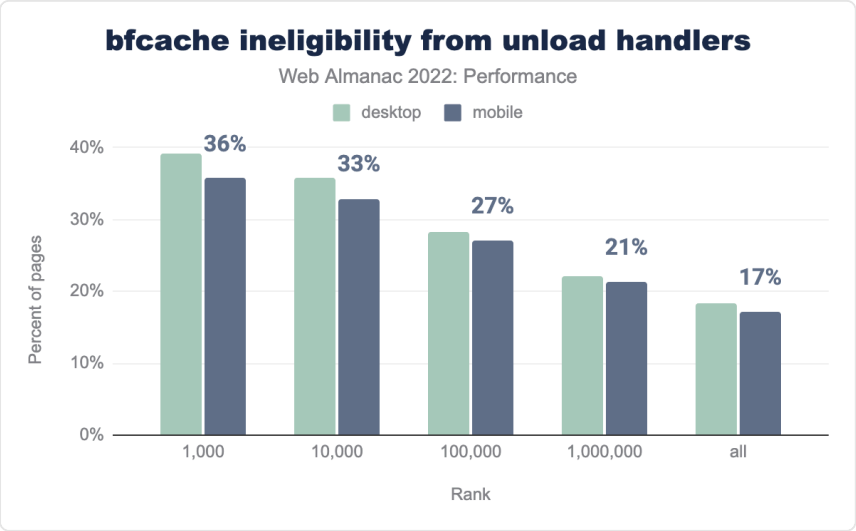



*図12.35. サイトのランク別の `unload` の使用。*

`unload` イベントは、ページが去る（アンロードされる）プロセスにあるときに作業を行うために推奨されない方法です。それを行うためのより良い方法[450]があることに加えて、それはページをbfcacheの対象外にする方法の1つでもあります。

すべてのモバイルページの17%がこのイベントハンドラーを設定していますが、ウェブサイトの人気度が高いほど状況は悪化します。上位1千のモバイルページでは、この理由でbfcacheの対象外となるページが36%あります。

450. *https://web.dev/articles/bfcache?hl=ja#never-use-the-unload-event*

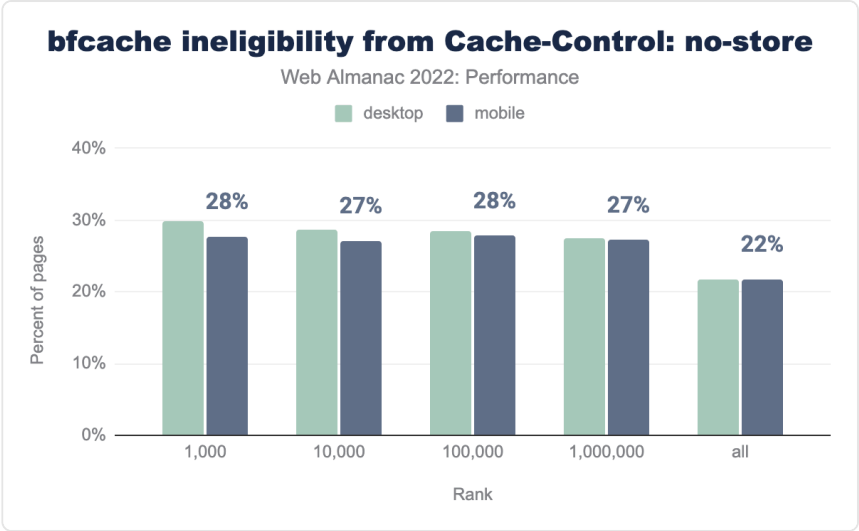



*図12.36. サイトのランク別の`Cache-Control: no-store`の使用。*

`Cache-Control: no-store`ヘッダーは、ユーザーエージェントに特定のリソースを決してキャッシュしないように指示します。メインHTMLドキュメントに設定された場合、この設定はページ全体をbfcacheの対象外にします。

すべてのモバイルページの22%がこのヘッダーを設定しており、上位1千のモバイルページでは28%です。`unload`基準とこの基準は排他的ではないため、組み合わせると、少なくとも22%のモバイルページがbfcacheの対象外であると言えます。

繰り返しますが、これらのことがCLSの問題を引き起こすわけではありません。しかし、これらを修正することで、ページをbfcacheの対象とすることができ、レイアウトの安定性を間接的にしかし強力に改善することが示されています。

## CLSの結論

CLSは2022年にもっとも改善されたCWVメトリックであり、全体的に「良好」なCWVを持つウェブサイトの数に顕著な影響を与えたようです。

この改善の原因は、Chromeのbfcacheの導入にあると思われます。これは2022年1月のCrUXデータセットに反映されています。しかし、少なくとも5分の1のサイトが、攻撃的な`no-store`キャッシングポリシーまたは`unload`イベントリスナーの推奨されない使用のために、この機能の対象外となっています。これらのアンチパターンを修正することは、パフォーマンスを改善するためのCLSの「奇妙なトリック」です。

サイトオーナーがCLSを直接改善するための他の方法もあります。画像に`height`および`width`属性を設定することがもっとも簡単な方法です。アニメーションのスタイルの最適化やフォントの読み込み方法の最適化も、考慮すべき他の2つを認めることはより複雑なアプローチです。

## 最初の入力までの遅延 (FID)

最初の入力までの遅延[451]（FID）は、クリックやタップのような最初のユーザーのインタラクションから、ブラウザが対応するイベントハンドラーの処理を開始するまでの時間を測定します。ウェブサイトが「良好」なFIDを持っている場合、サイト全体のナビゲーションの少なくとも75パーセントが100ミリ秒未満で完了します。

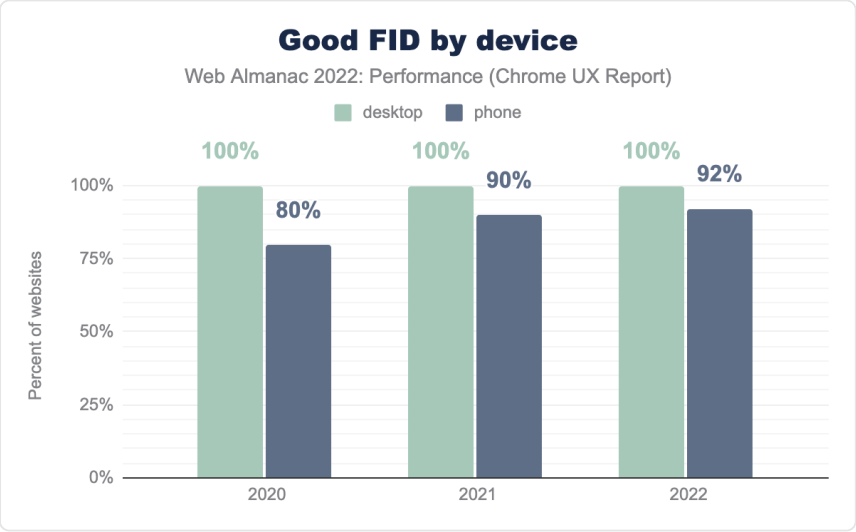



*図12.37. デバイス別の良好なFID*

事実上すべてのウェブサイトがデスクトップユーザーに対して「良好」なFIDを持っており、この傾向は年を経ても変わっていません。モバイルのFIDパフォーマンスも非常に高速で、92%のウェブサイトが「良好」なFIDを持っており、昨年に比べてわずかに改善されています。

これほど多くのウェブサイトが良好なFID体験を提供しているのは素晴らしいことですが、開発者は自己満足に陥らないよう注意が必要です。Googleは新しい応答性メトリックの実験を行っており[452]、それがFIDに取って代わる可能性があります。とくに、サイトはFIDに比べ

451. *https://web.dev/articles/fid?hl=ja*
452. *https://web.dev/blog/better-responsiveness-metric?hl=ja*

てこの新しいメトリックでかなり悪いパフォーマンスを示す傾向にあるため、これはとくに重要です。

## FIDのメタデータとベストプラクティス

ウェブ全体で応答性を改善する方法について、もっと深く掘り下げてみましょう。

### ダブルタップズームを無効にする

Chromeを含む一部のモバイルブラウザは、ユーザーがダブルタップしてズームする[453]ことを試みていないことを確認するため、タップ入力を処理する前に少なくとも250ミリ秒待ちます。”良好”なFIDの閾値が100ミリ秒であることを考えると、この振る舞いは評価を通過することが不可能になります。

```
<meta name="viewport" content="width=device-width, initial-
scale=1">
```

修正は簡単です。上記のような`meta`ビューポートタグをドキュメントのヘッドに含めることで、ブラウザにデバイスの幅と同じ幅でページをレンダリングするよう促し、テキストコンテンツをより読みやすくし、ダブルタップでのズームが不要になります。

これはレスポンシブネスを意味のある方法で改善するためのもっとも迅速で、簡単で、侵入性の低い方法の1つであり、すべてのモバイルページで設定されるべきです。


*図12.38. ビューポート`meta`タグを設定していないモバイルページの割合。*

モバイルページの7.3%が`meta`ビューポートディレクティブを設定していません。”良好”なFIDの基準を満たしていないモバイルウェブサイトが約8%あることを思い出してください。これは、サイトのレスポンシブネスを不必要に遅くしているウェブの大きな割合です。これを修正することは、FID評価に合格するか否かの違いを意味するかもしれません。

453. *https://developer.chrome.com/blog/300ms-tap-delay-gone-away?hl=ja*

## トータルブロッキングタイム(TBT)

トータルブロッキングタイム[454](TBT)は、視覚コンテンツの初期表示時間[455](FCP)と操作可能になるまでの時間[456](TTI)の間の時間で、メインスレッドがブロックされ、ユーザー入力に応答できなかった総時間を表します。

TBTは、シンセティックテストでユーザーインタラクションを現実的にシミュレートすることの課題を考慮して、FIDのラボベースの代理としてよく使用されます。

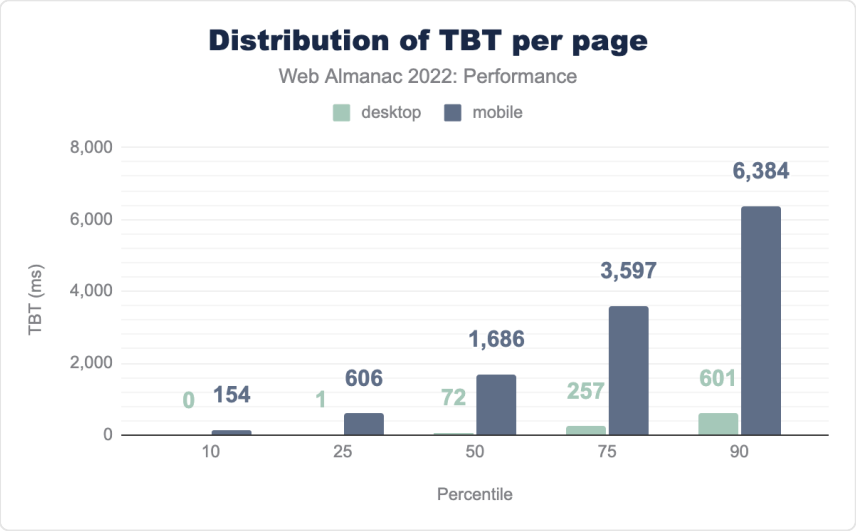



*図12.39. ページごとのラボベースTBTの分布。*

これらの結果は、HTTP Archiveデータセットのページのラボベースの TBTパフォーマンスから得られたものであることに注意してください。これは重要な区別です。なぜなら、これまでのところ、私たちはCrUXデータセットからの実際のユーザーパフォーマンスデータを見てきたからです。

それを念頭に置いて、モバイルページはデスクトップページよりもTBTが著しく悪いことがわかります。これは驚くことではありません。なぜなら、私たちのLighthouseモバイル環境は、低端のモバイルデバイスをエミュレートする方法でCPUを制限するように意図的に設定されているからです。それにもかかわらず、結果は中央値のモバイルページがTBTが1.7秒であることを示しており、これが実際のユーザー体験であれば、FCPから1.7秒以内のタップは反応しないことを意味します。90パーセンタイルでは、ユーザーはページが反応するまで6.3秒待たなければなりません。

454. *https://web.dev/articles/tbt?hl=ja*
455. *https://web.dev/articles/fcp?hl=ja*
456. *https://web.dev/articles/tti?hl=ja*

これらの結果はシンセティックテストから得られたものであるにもかかわらず、実際のウェブサイトが提供する実際のJavaScriptに基づいています。同様のハードウェアを持つ実際のユーザーがこれらのサイトのいずれかにアクセスしようとした場合、そのTBTはあまり変わらないかもしれません。それにもかかわらずTBTとFIDの主な違いは、後者がユーザーがページと実際に対話することに依存していることであり、彼らはTBTウィンドウの前、中、または後のいつでもこれを行うことができすべてが大きく異なるFID値につながります。

### 長時間タスク

長時間タスク[457]は、メインスレッドが入力に応答できない、少なくとも50ミリ秒のスクリプトによるCPUアクティビティの期間です。ユーザーがその時にページと対話しようとすると、どんな長時間タスクでも応答性の問題を引き起こす可能性があります。

上記のTBT分析のように、このセクションはラボベースのデータからの情報を引用しています。その結果、ページのロード観測ウィンドウ中にのみ長時間タスクを測定できます。このウィンドウは、ページがリクエストされた時点から始まり、60秒後またはネットワークの非活動状態が3秒続いた後のどちらか早い方で終了します。実際のユーザーは、ページの全生存期間中に長時間タスクを経験する可能性があります。

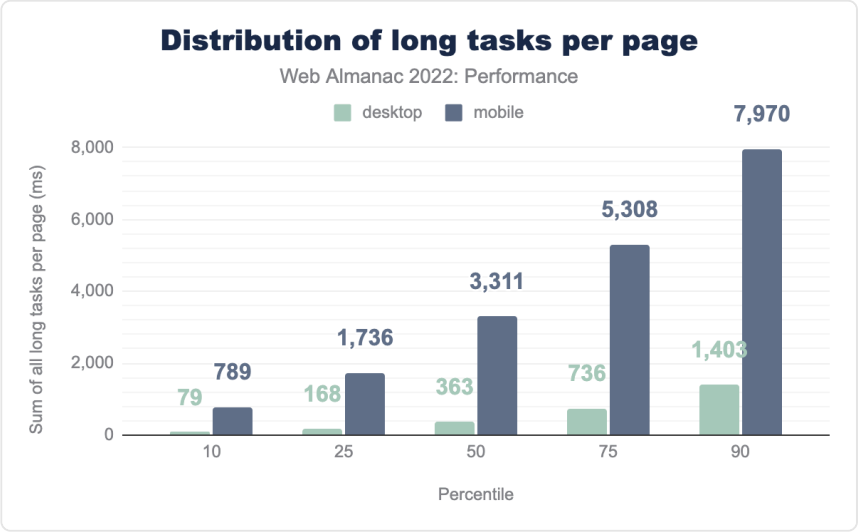



*図12.40. ページごとのラボベースの長時間タスクの分布。*

中央値のモバイルウェブページは、長時間タスクで3.3秒分を有していますが、デスクトップページは0.4秒分のみです。再び、これはCPU速度が応答性のヒューリスティックスに大

457. https://web.dev/articles/long-tasks-devtools?hl=ja

きな影響を与えていることを示しています。90パーセンタイルでは、モバイルページは少なくとも8.0秒の長時間タスクを有しています。

これらの結果は、TBT時間の分布よりもかなり高いことに注意する価値があります。TBTはFCPとTTIによって制限され、FIDはCPUがどれだけ忙しいかとユーザーがページと対話しようとするタイミングの両方に依存します。これらのTTI後の長時間タスクも、最初のインタラクション中に発生しない限り、FIDによって表されないため、イライラする応答性の経験を作り出す可能性があります。これは、ページの全生存期間を通じてユーザーの経験をより包括的に表すフィールドメトリックが必要な理由の1つです。

## インタラクションから次の描画 (INP)

インタラクションから次の描画[458] (INP)は、ユーザーインタラクションに応答してブラウザが次のペイントを完了するまでの時間を測定します。このメトリックは、応答性を測定する方法を改善する提案についてGoogleがフィードバックを求めた[459]後に作成されました。多くの読者がこのメトリックについてはじめて聞くことになるので、それがどのように機能するかについてもう少し詳しく説明する価値があります。

この文脈でのインタラクションとは、ユーザーがWebアプリケーションに入力を提供し、次のフレームの視覚フィードバックが画面に描かれるのを待つユーザー体験を指します。INPに考慮される唯一の入力は、クリック、タップ、およびキープレスです。INPの値自体は、ページ上の最悪のインタラクションレイテンシーのうちの1つから取られます。INPのドキュメント[460]で、それがどのように計算されるかの詳細を参照してください。

FIDとは異なり、INPはページ上の最初のインタラクションだけでなく、すべてのインタラクションを測定します。また、イベントハンドラーが処理を開始するまでの時間のみを測定するFIDとは異なり、次のフレームが描かれるまでの全時間を測定します。これらの点において、INPはページ上の包括的なユーザー体験をより代表するメトリックです。

ウェブサイトは、INP体験の75%が200ミリ秒より速い場合、「良好」と評価されます。75パーセンタイルが500ミリ秒以上である場合、ウェブサイトは「悪い」と評価されます。それ以外の場合は、INPは「改善が必要」と評価されます。

458. *https://web.dev/articles/inp?hl=ja*
459. *https://web.dev/blog/responsiveness?hl=ja*
460. *https://web.dev/articles/inp?hl=ja#what-is-inp*

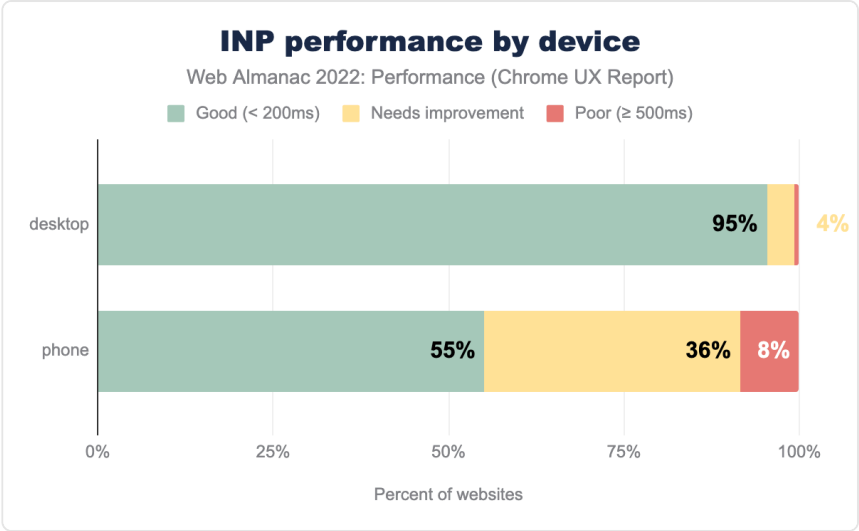



*図12.41. デバイス別のINPパフォーマンスの分布。*

モバイルでのウェブサイトの55%が「良好」なINPを持ち、36%が「改善が必要」で、8%が「悪い」INPを持っています。デスクトップのINPの話はFIDに似ており、ウェブサイトの95%が「良好」と評価され、4%が「改善が必要」で、1%が「悪い」と評価されています。

デスクトップとモバイルユーザーのINP体験の著しい格差は、FIDと比較してもはるかに広がっています。これは、ウェブサイトが行う圧倒的な作業量にモバイルデバイスが追いついていない程度を示しており、JavaScript[461]への依存度が増していることが主な要因であると指摘しています。

### ランク別INP

ウェブ全体でINPパフォーマンスがどれだけ不均等に分布しているかを見るために、ウェブサイトを人気ランキングでセグメント化すると便利です。

461. https://httparchive.org/reports/state-of-javascript?start=earliest&end=latest&view=grid#bytesJs

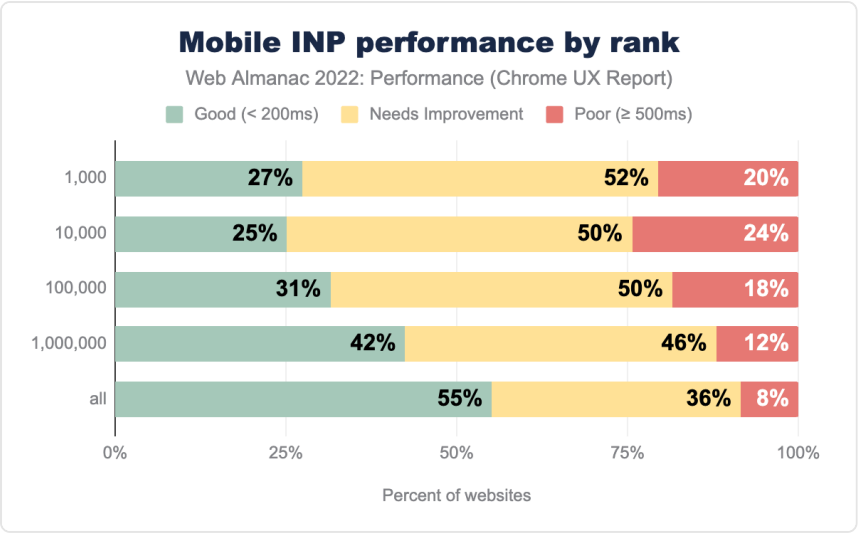



*図12.42. ランク別のモバイルINPパフォーマンス。*

もっとも人気のあるトップ1kウェブサイトの27%が良好なモバイルINPを提供しています。サイトの人気度が下がるにつれて、良好なモバイルINPを持つ割合は少し変わります。トップ10kランクで25%に悪化し、トップ100kで31%に改善し、トップ100万で41%になり、最終的には全ウェブサイトで55%になります。トップ1kを除いて、INPパフォーマンスはサイトの人気度に反比例するようです。

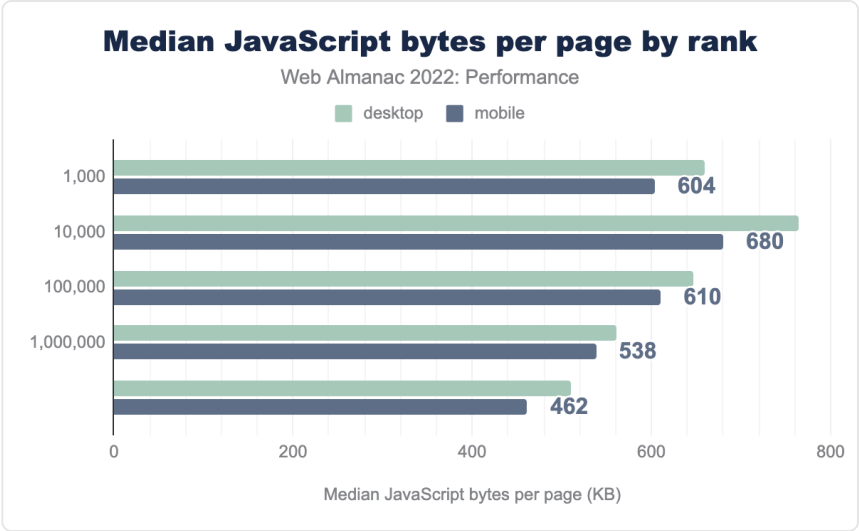



*図12.43. ランク別、ページあたりにロードされるJavaScriptの中央値。*

これらのランクごとにモバイルページがロードするJavaScriptの量を見ると、同じおもしろいパターンに従います！トップ1kのモバイルページの中央値は604KBのJavaScriptをロードし、トップ10kでは680KBに増加し、全ウェブサイトでは462KBまで下がります。これらの結果は、多くのJavaScriptをロードして使用することが必ずしも悪いINPを引き起こすわけではないが、確かに相関関係が存在することを示唆しています。

### INPが仮想のCWVメトリックとして

INPは公式のCWVメトリックではありませんが、GoogleのCWVプログラムのテックリードであるAnnie Sullivan[462]は、その将来についてコメント[463]しています。*"INPはまだ実験段階です！まだコアウェブバイタルではありませんが、FIDを置き換えることができればと思っています。"*

これは興味深い疑問を提起します：仮にINPが今日CWVメトリックであったら、合格率はどのように異なるでしょうか？

462. *https://twitter.com/anniesullie*
463. *https://twitter.com/anniesullie/status/1535208365374185474*

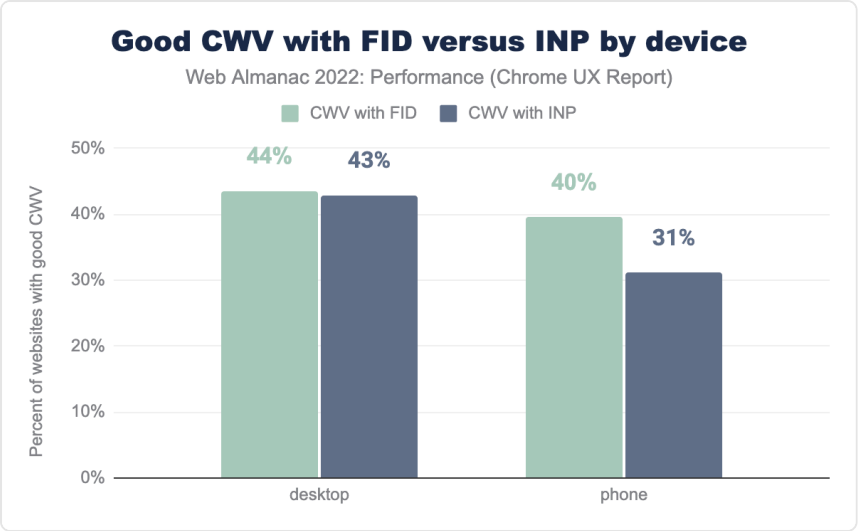



*図12.44. デバイス別、FIDとINPを使用した良好なCWVを持つウェブサイトの割合の比較。*

デスクトップ体験では、状況はあまり変わりません。ウェブサイトの43%がINPで良好なCWVを持つでしょうが、FIDでは44%です。

しかし、モバイル体験のウェブサイト間での差ははるかに劇的で、INPで良好なCWVを持つものは31%に減少し、FIDでは40%です。

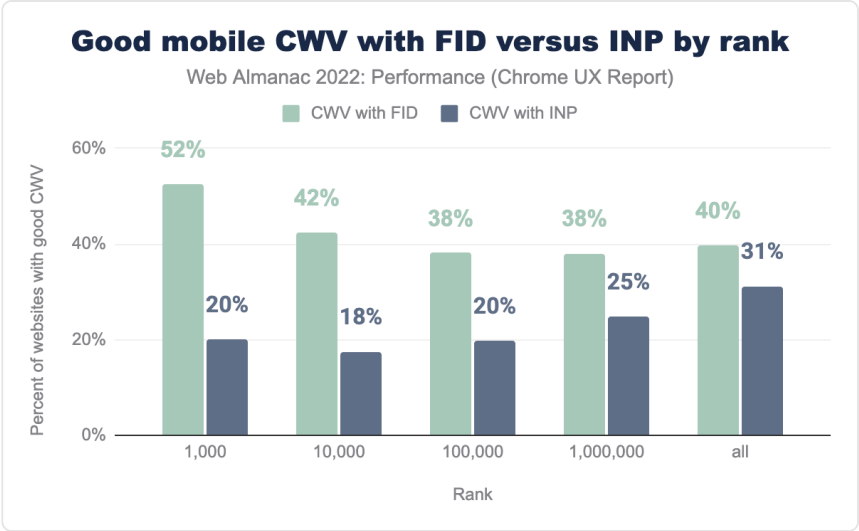



*図12.45. ランク別、モバイルウェブサイトのFIDとINPで良好なモバイルCWVを持つ割合の比較。*

サイトランクによるモバイル体験を見ると、状況はさらに厳しくなります。FIDで良好なCWVを持つトップ1kのウェブサイトは52%ですが、INPでは20%のみが良好なCWVを持つことになり、32パーセンテージポイントの減少となります。したがって、もっとも人気のあるウェブサイトはFIDと比較して全ウェブサイト（52%対40%）で過剰にパフォーマンスを発揮していますが、INPでは実際には*不十分*（20%対31%）です。

この話はトップ10kウェブサイトにも似ており、CWVとしてのINPによって24パーセンテージポイント減少します。このランクのウェブサイトは、INPで良好なCWVの最低率を持つことになります。前のセクションで見たように、これはJavaScriptの使用量がもっとも高いランクでもあります。人気が低いランクになるにつれて、FIDとINPの良好なCWV率は収束し、それぞれ18、13、9パーセンテージポイントの差に減少します。

これらの結果は、もっとも人気のあるウェブサイトがINPを改善するためにもっとも努力が必要であることを示しています。

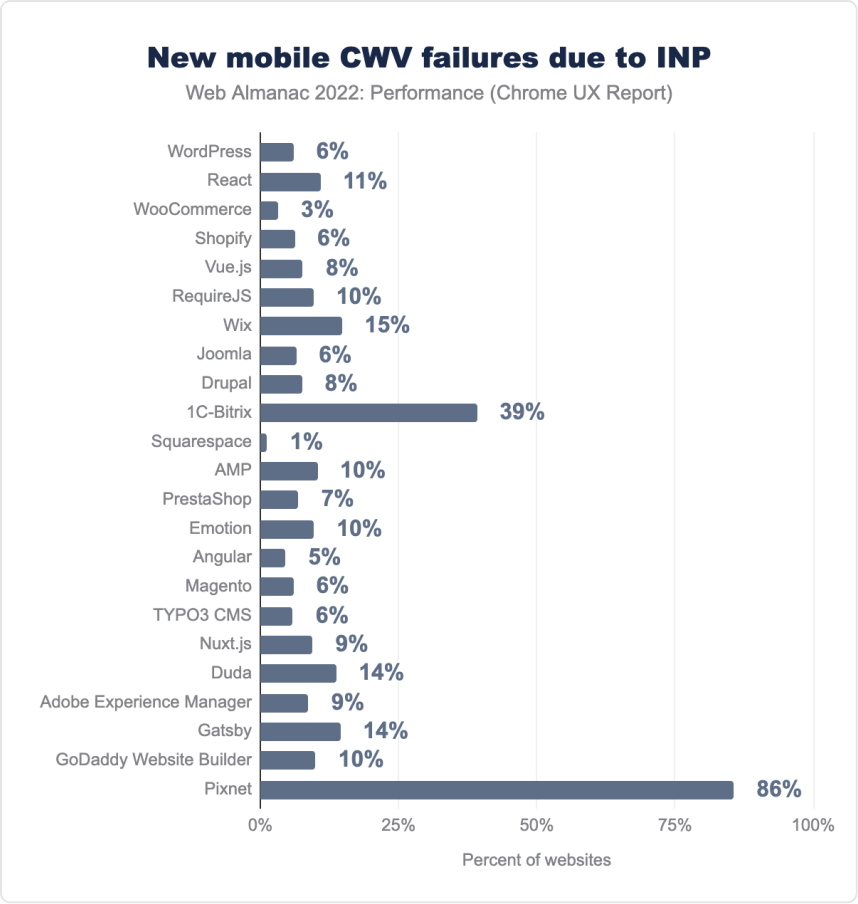



*図12.46. 技術別、FIDからINPへの切り替えによる良好なCWVを持つウェブサイトの割合の変化。*

このチャートでは、FIDがINPに置き換えられた場合に「良好」なCWVとみなされなくなる特定の技術のウェブサイトの割合を見ています。

このチャートで目立つのは、CMSである1C-BitrixとPixnetで、これらはINPでCWV評価に合格しなくなるウェブサイトの膨大な割合を持っています。Pixnetは、98%から13%に下がり、ウェブサイトの86%が失われる可能性があります！1C-Bitrixの合格率は、79%から40%に減少し、39%の差があります。

Reactフレームワークを使用するウェブサイトの11%がもはやCWVに合格しなくなります。Wixは現在2番目に人気のあるCMSで、Reactを使用しています。そのウェブサイトの15%が「良好」なCWVを持たなくなります。ただし、比例してWixウェブサイトは依然として

Reactウェブサイト全体よりもCWVに合格する割合が高くなります、それぞれ24%と19%ですが、INPはその差を縮めるでしょう。

リストの中でもっとも人気のある技術であるWordPressは、230万のウェブサイトの6%がもはや「良好」なCWVを持たなくなります。その合格率は、30%から24%に減少します。

Squarespaceは、この仮説的な変更による動きがもっとも少ないでしょう。ウェブサイトの「良好」なCWVを持つものが1%しか失われません。これは、Squarespaceのウェブサイトが最初のインタラクションが速いだけでなく、ページ体験全体を通じて一貫して速いインタラクションを持っていることを示唆しています。実際に、CWVテクノロジーレポート[464]は、INPで他のCMSを大きく上回っており、ウェブサイトの80%以上がINPの閾値をクリアしていることを示しています。

### INPとTBT

INPがFIDよりも優れた反応性メトリックである理由は何か？その質問に答える1つの方法は、フィールドベースのINPとFIDのパフォーマンスと、ラボベースのTBTパフォーマンスとの相関を見ることです。TBTはページがどれだけ反応しない可能性があるかの直接的な測定ですが、実際には「ユーザーが感じる体験」を測定しないため、CWV自体にはなり得ません。

このセクションは、2022年5月のデータセットを使用したAnnie Sullivanの研究[465]に基づいています。

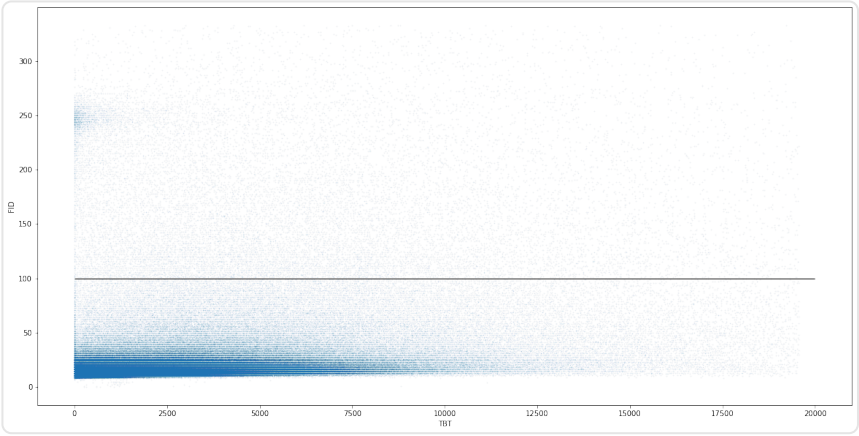



*図12.47. FIDとTBTの相関を視覚化する散布図。(出典[466])*

464. *https://datastudio.google.com/s/sM9D7EUjxU8*
465. *https://colab.sandbox.google.com/drive/12lJmAABgyVjaUbmWvrbzj9BkkTxw6ay2*
466. *https://colab.sandbox.google.com/drive/12lJmAABgyVjaUbmWvrbzj9BkkTxw6ay2*

このチャートは、ページのFIDとTBTの反応性の関係を示しています。100 msの固定水平線は「良い」FIDの閾値を表し、ほとんどのページはこの閾値の下に快適に収まります。

このチャートのもっとも注目すべき属性は、左下の角に密集した領域であり、TBT軸に沿って広がっているように見えます。この広がりの長さは、高いTBTと低いFIDを持つページを表し、FIDとTBTの間の相関の低さを示しています。

また、低いTBTと約250 msのFIDを持つページのパッチもあります。この領域は、`<meta name=viewport>` タグが欠けているためにタップ遅延[467]の問題を持つページを表しています。これらはこの分析の目的のために安全に無視できる外れ値です。

この分布のケンドール[468]とスピアマン[469]の相関係数はそれぞれ0.29と0.40です。

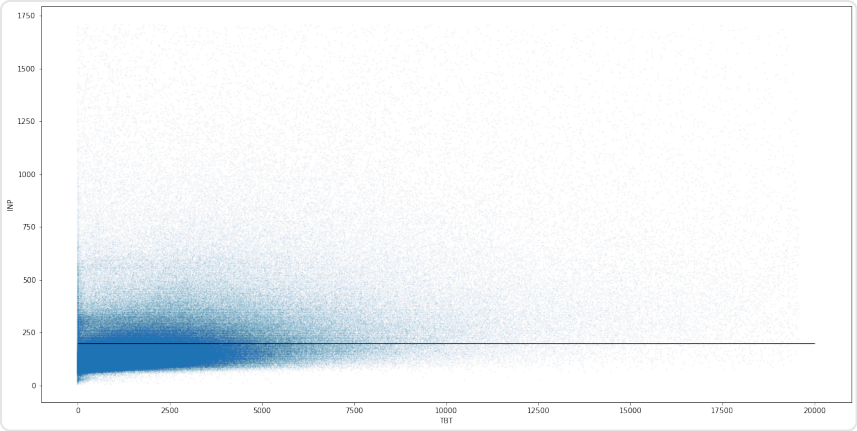



*図12.48. INPとTBTの相関を視覚化する散布図。(出典[470])*

これは同じチャートですが、FIDの代わりにINPです。「良い」INPのための200 msの閾値を表す固定水平線がここにあります。FIDと比較して、この線の上にあるページが多く、「良い」と評価されないページが多いです。

このチャートのページは、FIDとTBTの間の相関度が高いことを示す左下の角により密集しています。まだ広がりがありますが、それほど顕著ではありません。

この分布のケンドールとスピアマンの相関係数はそれぞれ0.34と0.45です。

467. *https://developer.chrome.com/blog/300ms-tap-delay-gone-away*
468. *https://ja.wikipedia.org/wiki/%E3%82%B1%E3%83%B3%E3%83%89%E3%83%BC%E3%83%AB%E3%81%AE%E9%A0%86%E4%BD%8D%E7%9B%B8%E9%96%A2%E4%BF%82%E6%95%B0*
469. *https://ja.wikipedia.org/wiki/%E3%82%B9%E3%83%94%E3%82%A2%E3%83%9E%E3%83%B3%E3%81%AE%E9%A0%86%E4%BD%8D%E7%9B%B8%E9%96%A2%E4%BF%82%E6%95%B0*
470. *https://colab.sandbox.google.com/drive/12IJmAABgyVjaUbmWvrbzj9BkkTxw6ay2*

> *まず、INPはTBTと相関していますか？FIDよりもTBTとの相関が強いですか？はい、そうです！*
>
> *しかし、両方のメトリックがTBTと相関しています。INPはメインスレッドをブロックするJavaScriptの問題をより多く捕捉していますか？「良い」閾値を満たすサイトの割合を分解することで、はい、そうです！*
>
> *—Annie SullivanのTwitter[471]にて*

Annieが指摘するように、両方のメトリクスはTBTと相関していますが、彼女はINPがより強く相関していると結論づけ、それをより良い反応性メトリックとしています。

### FIDの結論

これらの結果は、FIDによって描かれる明るい絵とは裏腹に、サイトには確かに反応性の問題があることを示しています。INPがCWVメトリックになるかどうかにかかわらず、今からそれを最適化し始めると、ユーザーは感謝するでしょう。

ほぼ10分の1のモバイルサイトが、ダブルタップズームを無効にすることを怠ることで、無料のパフォーマンスを放棄しています。これはすべてのサイトが行うべきことです。HTMLの一行だけで、FIDとINPの両方に利益をもたらします。ページでLighthouseを実行し、ビューポート[472]の監査が確実に行われていることを確認してください。

CWVとしてのINPを仮定して見ると、FIDレベルに戻るだけでなく、どれだけ多くの作業が必要であるかがわかります。とくにJavaScriptの（過度な）使用の結果として、もっとも人気のあるモバイルウェブサイトはこのような変更の影響を受けやすいです。CMSやJavaScriptフレームワークの中には、他よりも大きな影響を受けるものがあり、クライアント側で行う作業の量を集団で抑制するためには、エコシステム全体の努力が必要になります。

## 結論

業界がCWVについてさらに学び続けるにつれて、実装面でも全体的なトップレベルメトリック値でも、着実な改善が見られます。もっとも顕著なパフォーマンス最適化の進歩は、その影響が一度に多くのサイトで感じられるため、プラットフォームレベルで行われていま

471. *https://twitter.com/anniesullie/status/1525161893450727425*
472. *https://developer.chrome.com/docs/lighthouse/pwaviewport?hl=ja*

す。たとえば、Androidとbfcacheの改善などです。しかし、パフォーマンスパズルのもっとも捉えどころのない部分、つまり個々のサイトオーナーに注目しましょう。

CWVを検索ランキングの一部にするというGoogleの決定は、多くの企業、とくにSEO業界でパフォーマンスを優先事項のトップに押し上げました。個々のサイトオーナーは確かにパフォーマンスを向上させるために一生懸命に取り組んでおり、このスケールではそれぞれの努力が目立ちにくいかもしれませんが、昨年のCWVの改善に大きな役割を果たしています。

それでもなお、やるべき仕事はあります。私たちの研究は、LCPリソースの優先順位付けと静的な発見可能性を改善する機会を示しています。多くのサイトは依然として人工的なインタラクティビティの遅延を避けるためにダブルタップズームを無効にすることに失敗しています。INPに関する新しい研究は、FIDでは見過ごされやすかった反応性の問題を明らかにしました。INPがCWVになるかどうかにかかわらず、私たちは常に迅速で反応の良い体験を提供しようと努力すべきであり、データは私たちがもっとうまくできることを示しています。

結局のところ、常にやるべき仕事があるということです。そのため、もっとも影響力のあることは、ウェブパフォーマンスをよりアプローチしやすくすることを続けることです。これからの年月で、サイトオーナーへの「最後の一マイル」までウェブパフォーマンスの知識を伝えることを強調しましょう。

## 著者


### Melissa Ada

@mel_melificent  mel-ada  mel-ada

メル・アダはEtsyのウェブ・パフォーマンス・チームのソフトウェア・エンジニア。現在のコミュニティへの参加には、NYウェブ・パフォーマンス・ミートアップの共同主催や最近の作品についての講演が含まれる。

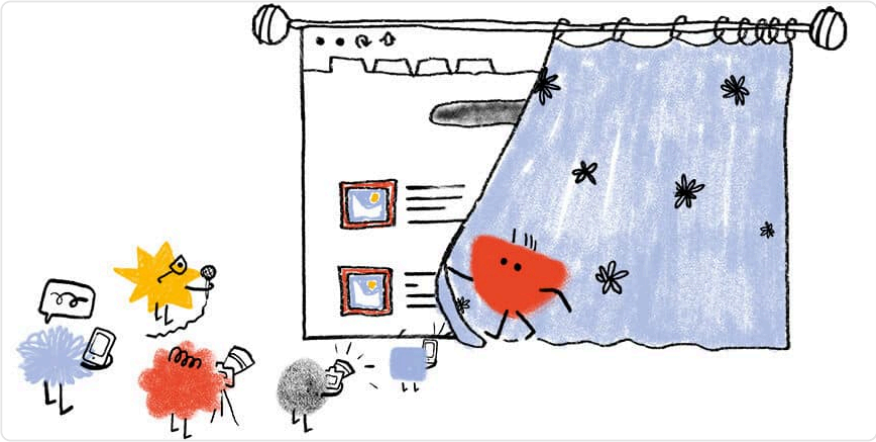
### Rick Viscomi

@rick_viscomi  rviscomi  https://rviscomi.dev/

リック・ヴィスコミはグーグルのDevRelエンジニアで、ウェブパフォーマンスを専門としている。ウェブ・パフォーマンス・テストに関する本、Using WebPageTest[473]の共著者。また、HTTP Archiveの共同管理者であり、Web Almanacの編集長でもある。

473. *https://usingwpt.com/*

# 部 II 章 13
# プライバシー



*Tom Van Goethem と Nurullah Demir によって書かれた。*
*Iskander Sanchez-Rola によってレビュ。*
*Max Ostapenko と Yana Dimova による分析。*
*Abel Mathew 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

最新のニュースを入手するため、オンラインソーシャルメディアを通じて友人と連絡を取り合うため、または素敵なドレスやセーターを購入するために、多くの人々がクリックするだけでこれらのサービスや情報を提供するウェブに依存しています。インターネットで平均して1日約7時間を過ごす[474]という副作用は、多くの閲覧活動、そして間接的には個人の興味やデータが、無数のオンラインウェブサービスや企業と共有されたり捕捉されたりすることです。

広告主はユーザーに最も関連性の高い広告（最も関与しそうなもの）を提供しようとするため、しばしばサードパーティの追跡を用いてユーザーの興味を推測します。本質的には、ユーザーのオンライン活動がステップバイステップで追跡され、特にウェブ上で最も普及している追跡者には、ユーザーの興味を推測するために必要ない情報が含まれている可能性のあ

474. *https://datareportal.com/reports/digital-2022-global-overview-report*

る情報の山が提供されます。その上、ユーザーはこれをオプトアウトする適切な選択肢を一般的に与えられていません。

この章では、プライバシーの観点から現在のウェブの状態を探ります。サードパーティの追跡の普及、このエコシステムを構成する異なるサービス、および特定の当事者がユーザーのプライバシーを守るために採用している保護措置（例えば、ブロックリストベースのアンチトラッカー）を回避しようとしている方法について報告します。さらに、ウェブサイトが訪問者のプライバシーを高めようとしている方法、つまり他の当事者と共有される情報を制限する機能を採用するか、GDPR[475]やCCPA[476]などのプライバシー規制に準拠することによっても調査します。

## オンライン追跡

**82%**

*図13.1. デスクトップで少なくとも1つのサードパーティトラッカーを含むウェブサイトの割合。*

追跡はウェブ上で最も広範にわたる技術の一つであり、デスクトップウェブサイトの82%（モバイルでは80%）が少なくとも1つのサードパーティトラッカーを含んでいることがわかります。オンラインでのユーザーの行動を追跡することにより、これらの追跡会社はユーザーのプロファイルを作成することができ、これはパーソナライズされた広告に使用されるほか、ウェブサイトの所有者に誰が彼らのウェブサイトを訪問しているかの洞察を与えた

### サードパーティによる追跡

オンライン追跡の最も一般的な形態の一つは、サードパーティのサービスを通じて行われます。通常、ウェブサイトの所有者は、サイトの分析を提供したり、訪問者に広告を表示するためのサードパーティのクロスサイトスクリプトを含めます。このスクリプトはサードパーティのクッキーを設定し、ユーザーが訪れたウェブサイトを記録することができます。ユーザーが同じサードパーティサービスを含む別のウェブサイトを訪問するたびに、このクッキーはトラッカーに送信され、彼らはユーザーを再識別し、両方のウェブサイト訪問を同じプロファイルにリンクすることができます。

含まれているサードパーティサービスの種類は多少異なりますが、それによってウェブサイ

475. https://gdpr.eu/
476. https://oag.ca.gov/privacy/ccpa

ト訪問者を追跡する機能が暗黙のうちに与えられています。そのようなトラッカーの2つの最も一般的なカテゴリは、WhoTracks.meによって定義されているように[477]、サイト分析スクリプト（モバイルで68%、デスクトップで73%）と広告（モバイルで66%、デスクトップで68%）です。これらに続いて、追跡と明確なリンクがないかもしれないいくつかのカテゴリがあります。例えば、顧客対話（顧客がウェブサイトの所有者に簡単にメッセージを送ることを可能にするサービス）、オーディオ/ビデオプレイヤー（例えば、YouTube埋め込みビデオ）、ソーシャル（例えば、Facebookの「いいね！」ボタン）などです。

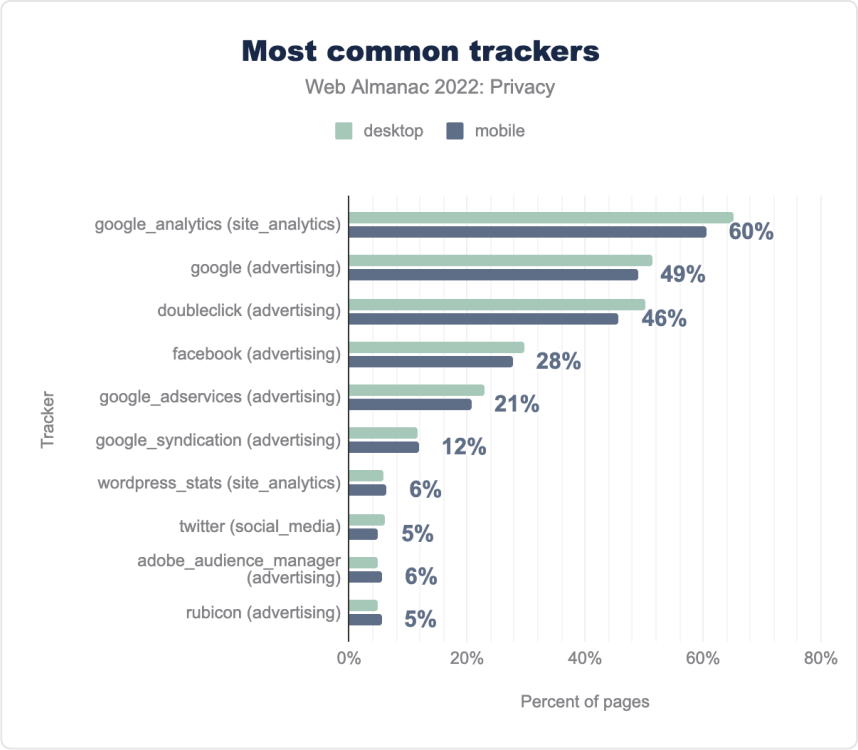



*図13.2. 最も一般的なトラッカー。*

トラッカーがユーザーのプロファイルを成功裏に作成するためには、多くのウェブサイトに含まれる必要があります。このようにしてユーザーのオンライン活動のかなりの部分を追跡することができます。最も一般的なトラッカーを見ると、これらは主に「いつもの容疑者」です。トップ10の最も一般的なトラッカーのうち、5つはGoogleに関連しています。このリストには、FacebookやTwitterなどの人気のソーシャルネットワークも含まれています。

477. *https://whotracks.me/blog/tracker_categories.html*

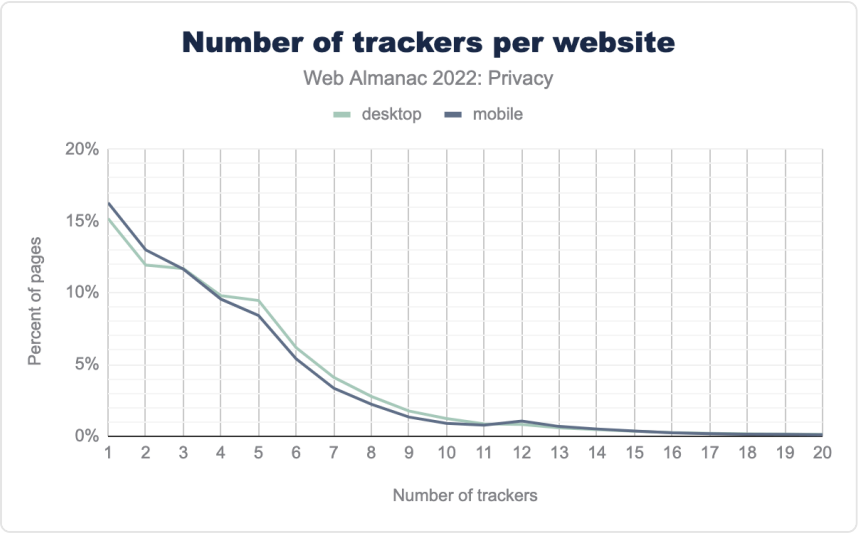



*図13.3. ウェブサイトごとのトラッカーの数。*

ウェブサイトは複数のサードパーティサービスを利用したいかもしれませんので、そのウェブサイトには複数のトラッカーが含まれることがあります（詳細については、ウェブ上に含まれるサードパーティに関するサードパーティ章をぜひご覧ください！）。私たちは、デスクトップサイトの約15%とモバイルサイトの16%が「ちょうど」1つのトラッカーを含んでいることを発見しました。残念ながら、これはウェブサイトが複数のトラッカーを含むことがより一般的であることを意味します。私たちは126個の異なるトラッカーを含むウェブサイトさえ見つけました！

## (再)ターゲティング

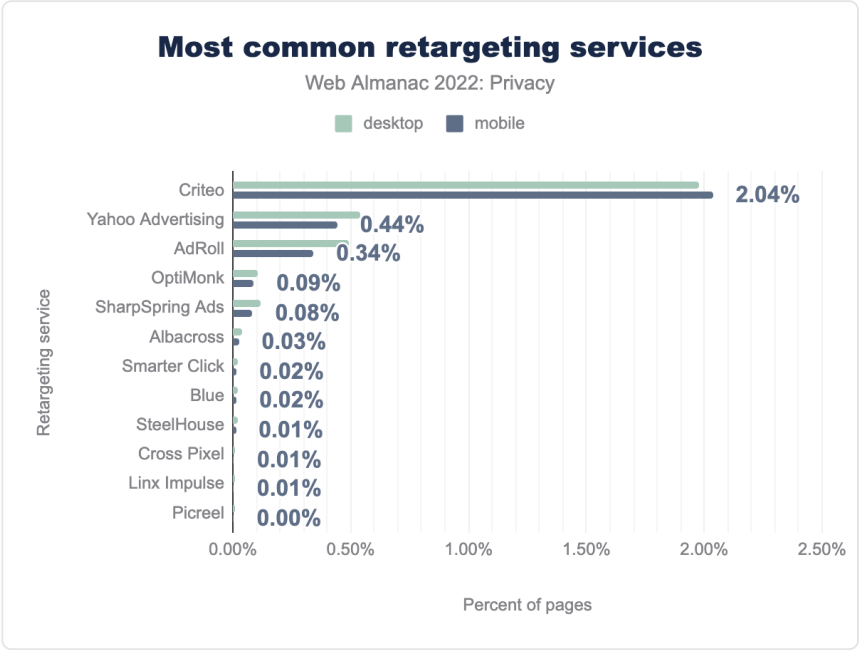



*図13.4. 最も一般的なリターゲティングサービス。*

ウェブを閲覧していると、最近調べた製品の広告に遭遇することがよくあります。その理由は広告のリターゲティングにあります。ウェブサイトがユーザーが特定の製品に興味を持っているかもしれないと検出すると、トラッカーや広告主にこれを報告し、後にユーザーが他の関連性のないウェブサイトを訪問する際に、ユーザーが興味を持っているとされる製品の広告を表示し、それによって購入へと誘導しようとします。

純粋にリターゲティングサービスを提供するトラッカーで最も普及しているのはCriteoで、デスクトップで1.98%、モバイルで2.04%の普及率を持っています。それに続くのはYahoo広告とAdRollで、合わせてCriteoの市場シェアの半分以下です。昨年[478]の最も広く使用されていたリターゲティングサービスであるGoogleタグマネージャーは、今回の結果には表示されていません。これは現在「タグマネージャー」のWappalyzerカテゴリに分類されているためです。このサービスはリターゲティングに使用されますが、リターゲティングタグの含まれることによって間接的に行われ、それらのタグは別々に検出されます。

478. *https://almanac.httparchive.org/ja/2021/privacy*

## サードパーティクッキー

前述の通り、異なるウェブサイト間でユーザーを追跡する最も確立された方法はサードパーティクッキーを使用することです。ブラウザのポリシーの最近の変更により、クッキーはデフォルトでクロスサイトリクエストに含まれなくなります。技術的な用語では、ほとんどのブラウザがクッキーの`SameSite`属性をデフォルト値の`Lax`に設定します。ウェブサイトはこれを明示的に自ら設定することで上書きすることができます。これは大規模に行われており、`SameSite`クッキーを設定するサードパーティクッキーの98%が`None`の値を設定しており、クロスサイトリクエストに含まれることを可能にしています。さらに、クッキーの有効期限もその有効期間を決定します。クッキーの中央値の寿命は365日です。クッキーとクッキー属性についての詳細は、セキュリティ章を参照してください。

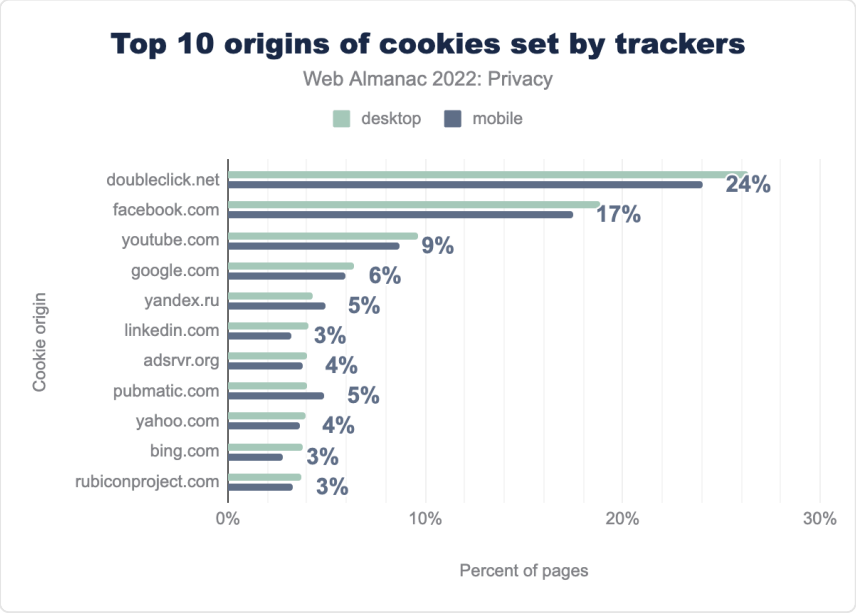



*図13.5. トラッカーによって設定されたクッキーの上位10の発生元。*

大部分において、クッキーを設定するサードパーティのトラッカーはウェブサイトに含まれるサードパーティと一致しています。しかし、最も人気のあるサードパーティのトラッカーであるGoogleアナリティクスはここではそれほど普及していません。これは、Googleアナリティクスがファーストパーティクッキー（`_ga`）を設定し、それが彼らの定義[479]によると「特定のプロパティに固有であるため、関連性のないウェブサイト間で特定のユーザーやブラウザを追跡するために使用することはできません」とされているためです。それにもかかわらず、サードパーティクッキーを設定する最も一般的な追跡ドメインは、依然として

479. *https://policies.google.com/technologies/cookies?hl=en-US*

Googleに関連する`doubleclick.net`です。リスト上の他のドメインはソーシャルメディアと広告に関連しています。

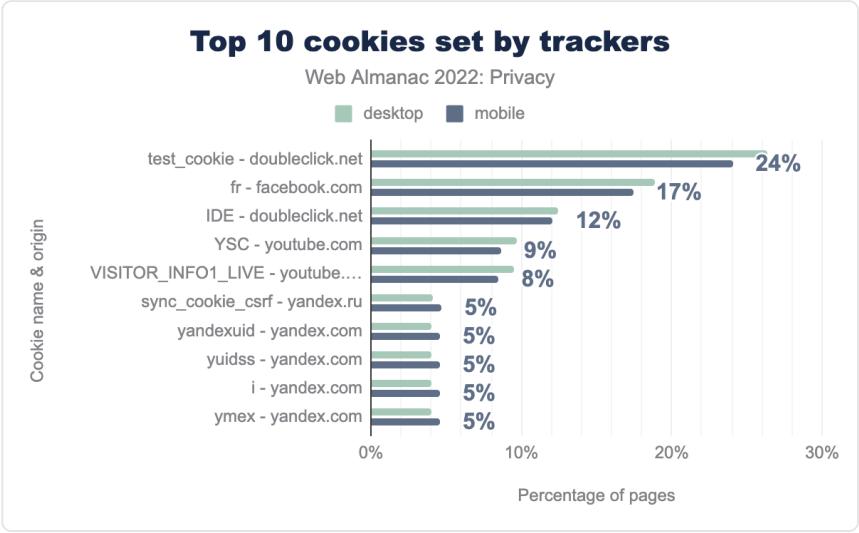



*図13.6. トラッカーによって設定された上位10のクッキー。*

最も一般的なサードパーティクッキーを見ると、再びいくつかの追跡ドメインが現れます。`doubleclick.net`からの`test_cookie`が先導しており、これは機能目的で使用されるクッキーで、寿命は15分です。このクッキーに続いて、`facebook.com`によって設定された`fr`クッキーがあります。これは「広告の配信、測定、および関連性の向上のために使用されるクッキーで、寿命は90日です」とその定義[480]によるとされています。最も普及している10個のサードパーティクッキーの残りはYouTubeとYandexによって設定されています。

480. *https://www.facebook.com/policy/cookies/*

## 回避技術：フィンガープリンティング

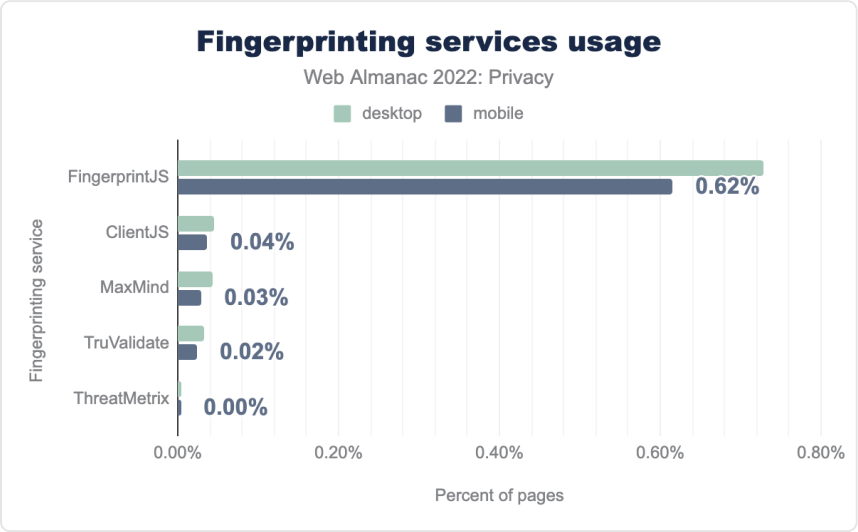



*図13.7. フィンガープリンティングサービスの使用状況。*

クッキーベースのトラッキングに対する対策が増えるにつれて、サードパーティクッキーのブロックをユーザーがより制御できるようになる中、一部のトラッカーはこれらの保護を回避しようとします。その一つの技術がフィンガープリンティングで、ブラウザ固有の機能（例：インストールされたブラウザ拡張機能）、OS固有の機能（例：インストールされたフォント）、ハードウェア固有の機能（例：使用されているGPUに基づく複雑な構成のレンダリングの違い）を利用してユーザーのユニークなフィンガープリントを作成します。このフィンガープリントにより、トラッカーは異なる関連性のないウェブサイト間で同じユーザーを再識別することができます。

私たちの分析では、5つの異なる、知られているフィンガープリンティングライブラリを調べ、ウェブ上でフィンガープリンティングを実行するために最も普及しているライブラリはFingerprintJS[481]であり、全てのウェブサイトの0.62%で見つかりました。これはライブラリがオープンソースで、無料版があるためであると考えられます。昨年の測定[482]と比較して、フィンガープリンティングの使用はほぼ変わっていないことがわかります。

## 回避技術：CNAMEトラッキング

ほとんどのトラッキング対策がサードパーティクッキーのブロックまたは無効化に焦点を当

481. *https://github.com/fingerprintjs/fingerprintjs*
482. *https://almanac.httparchive.org/ja/2021/privacy*

てている中、これらの保護を回避するもう一つの方法は、ファーストパーティクッキーを代わりに使用することです。ここでは、トラッカーがウェブサイトのサブドメインにCNAMEレコードを使用して偽装されます。トラッカーがクッキーを設定すると、それはファーストパーティクッキーと見なされます。CNAMEベースのトラッキングの制限は、特定のウェブサイト内でのユーザーの活動のみを追跡できるという点ですが、トラッカーはまだcookie syncing[483]を利用して複数のサイト間の訪問を一致させることができます。

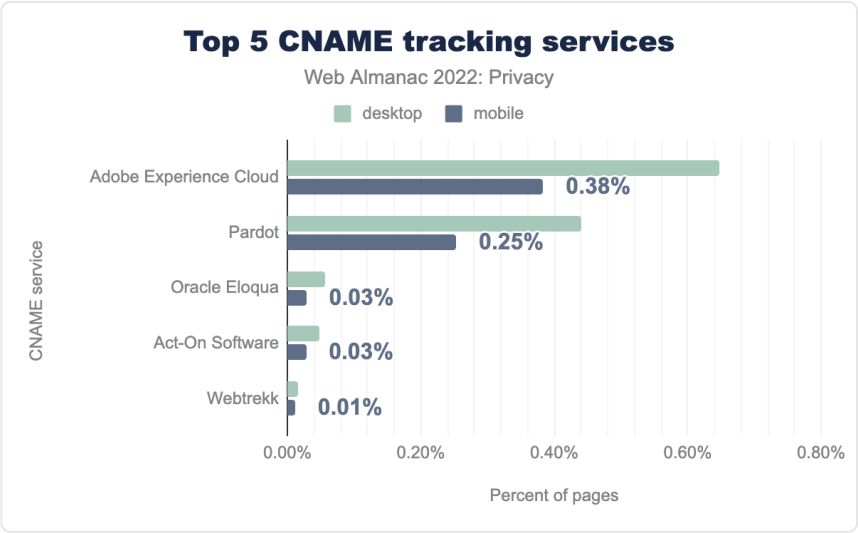



*図13.8. 上位5つのCNAMEトラッキングサービス。*

さまざまなCNAMEトラッカーを分析すると、市場シェアは主に2つの主要サービス、Adobe Experience Cloud（デスクトップで0.65%、モバイルで0.38%）とPardot（デスクトップで0.25%、モバイルで0.44%）に集中していることがわかります。興味深いことに、CNAMEトラッキングの採用はモバイルで訪問されたウェブサイトよりもデスクトップブラウザで訪問されたウェブサイトで著しく高いです。これは、モバイルブラウザにはプライバシーを保護するメカニズムが少ないためと思われます。たとえば、モバイルの人気ブラウザのほとんどは拡張機能をサポートしていません。

483. *https://adtechexplained.com/cookie-syncing-explained/*

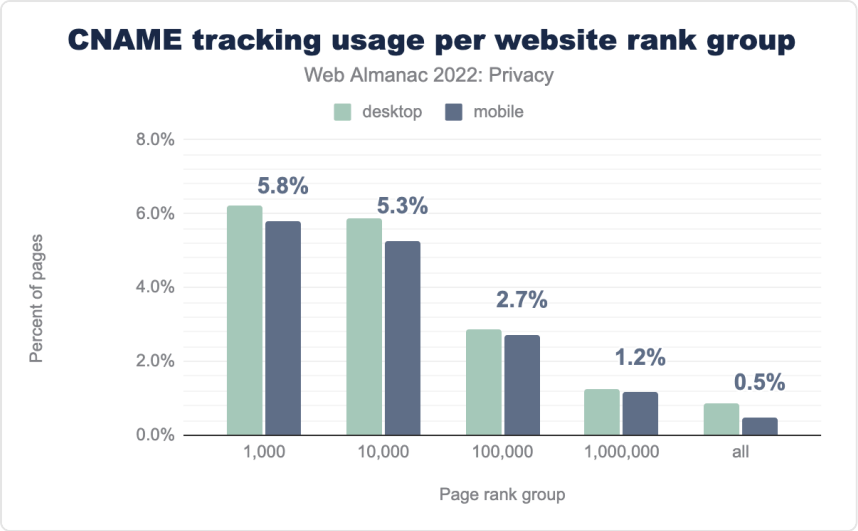



*図13.9. ウェブサイトランク別のCNAMEトラッキングの使用状況。*

CNAMEベースのトラッキングの全体的な普及率は非常に高くはないかもしれません（デスクトップウェブサイトで0.9%、モバイルサイトで0.5%）が、その採用は主に非常に人気のあるウェブサイトに集中しています。訪問されたウェブサイトのトップ1,000では、デスクトップサイトの6.2%とモバイルサイトの5.8%がCNAMEトラッカーを組み込んでいます。これは、ユーザーがウェブを閲覧する際にこのようなトラッカーに遭遇する可能性が高いことを意味します。

## ブラウザからの（機密）データへのアクセス

ブラウザには数多くのAPIがあり、開発者がさまざまなコンポーネントと任意の方法でやり取りするための便利なメカニズムを提供しています。これらのAPIのいくつかは、ユーザーのデバイスに接続されているセンサーやその他の周辺機器から情報を抽出するためにも使用できます。ほとんどのAPIは限定的な情報（例えば、画面の向き）を提供しますが、他のAPIは非常に詳細な情報（例えば、加速度計やジャイロスコープ）を提供し、これをデバイスのフィンガープリンティングや、モバイルデバイスで行う動作に基づいてユーザーが入力するパスワードを推測するために使用することができます。

## センサーイベント

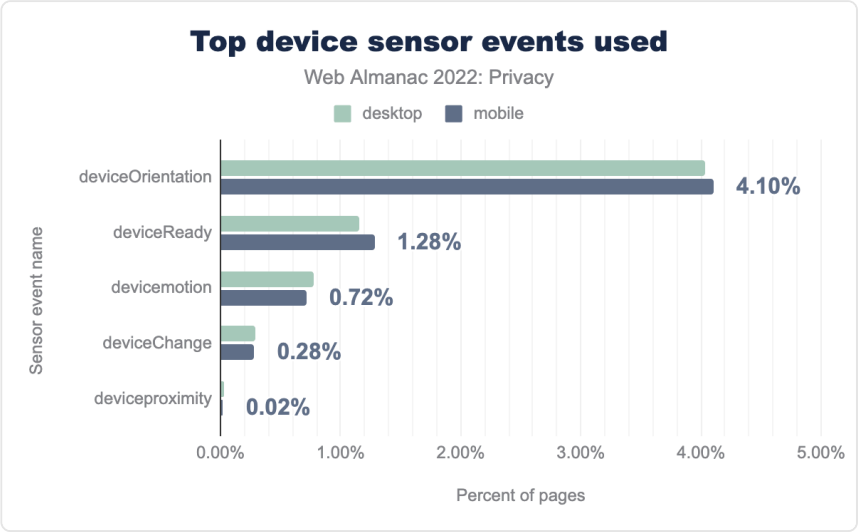



*図13.10. 使用されている主なデバイスセンサーイベント。*

ウェブサイトが最も注目しているセンサーイベントは`deviceOrientation`イベントであり、デバイスがポートレートモードからランドスケープモードに変わる、またはその逆の場合に発生します。デスクトップウェブサイトの4.0%、モバイルウェブサイトの4.1%で使用されています。使用率が（比較的）高いのは、デバイスの向きが変わるとウェブサイトがレイアウトの要素を更新したいと考えるかもしれないからです。

## メディアデバイス

**0.59%**

*図13.11. メディアデバイスを列挙するデスクトップページの割合。*

MediaDevices API[484]を使用すると、ウェブ開発者は`enumerateDevices()`メソッドを使ってユーザーのデバイスに接続されているすべてのメディアデバイスのリストを取得できます。この機能は、ビデオ通話を開始するためにユーザーがカメラやマイクロフォンを接続しているかどうかを判断するのに便利ですが、フィンガープリンティング目的でシステムの環

484. https://developer.mozilla.org/ja/docs/Web/API/MediaDevices

境に関する情報を収集するためにも使用されることがあります。デスクトップウェブサイトの0.59%、モバイルサイトの0.48%が接続されているメディアデバイスのリストにアクセスしようとしていることがわかります。興味深いことに、このAPIの使用は昨年以来かなり減少しています[485]。メディアデバイスのリストにアクセスするサイトの普及率は12倍高かったです。これはおそらく、もはやAPIを呼び出さない人気のあるライブラリが原因です。

## ジオロケーション

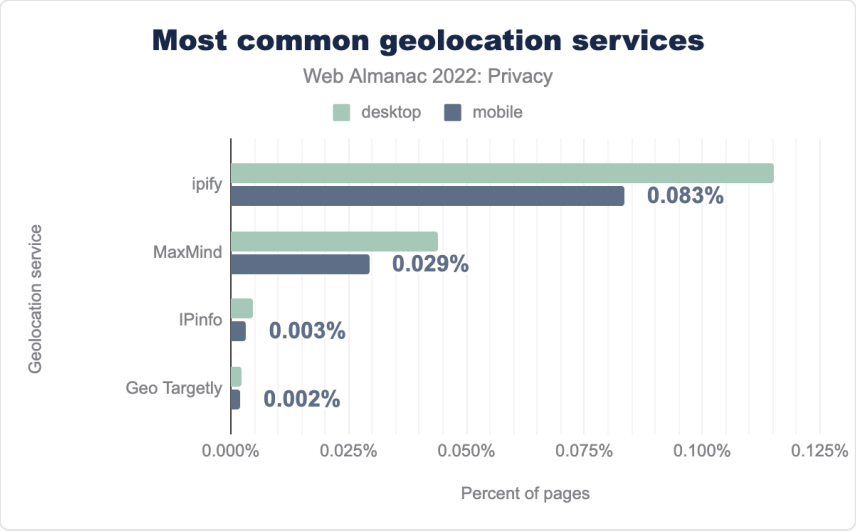



*図13.12. 最も一般的なジオロケーションサービス。*

私たちがウェブサイトを訪れる場所に基づいてローカライズされたコンテンツが多く提供されています。ウェブ開発者が訪問者の位置を特定するためには、サードパーティのジオロケーションサービスを利用できます。これらはIPアドレスに基づいてユーザーの位置を特定します。このジオロケーションは通常バックエンドで使用されますが、フロントエンドでもいくつかの使用例が見られます：デスクトップサイトの0.115%とモバイルサイトの0.083%が、ユーザーのIP位置を特定するためにipifyに連絡しています。

485. *https://almanac.httparchive.org/ja/2021/privacy#メディアデバイス*

0.65%

*図13.13. ブラウザのジオロケーションにアクセスしようとするデスクトップページの割合。*

IPベースのジオロケーションサービスは、特にユーザーがVPNを利用して元のIPアドレスを隠している場合には正確でないことがあります。そのため、ウェブサイトはより詳細な位置情報をジオロケーションAPI[486]を通じて要求することがあります。もちろん、この（プライバシー侵害的な）APIへのアクセスは、ユーザーが手動で提供する必要がある許可によって守られています。それでも、デスクトップサイトの0.65%とモバイルサイトの0.61%がホームページを訪れた際に、ユーザーの現在の位置情報にアクセスしようとしています。興味深いことに、安全でない接続でページがロードされた際にこの機能にアクセスしようとする574のデスクトップサイトを発見しました（昨年は900サイトでした）。この機能が提供するデータの性質がデリケートであるため、ほとんどのブラウザは安全な起源での使用に制限を設けています。

## 訪問者のプライバシーを改善するための確立されたコントロール

ウェブサイトは完全に信頼していないサードパーティ（スクリプト、プラグインなど）から多くのコンテンツを含むため、これらのサードパーティからユーザーのプライバシーを保護することが望まれます。次に、サードパーティがアクセスできる機能やデータを制限するために使用できるさまざまなコントロールを探ります。また、ウェブサイトがユーザーからどの情報を取得したいかを明確にする方法も調査します。

486. https://developer.mozilla.org/ja/docs/Web/API/Geolocation_API

## パーミッションポリシー

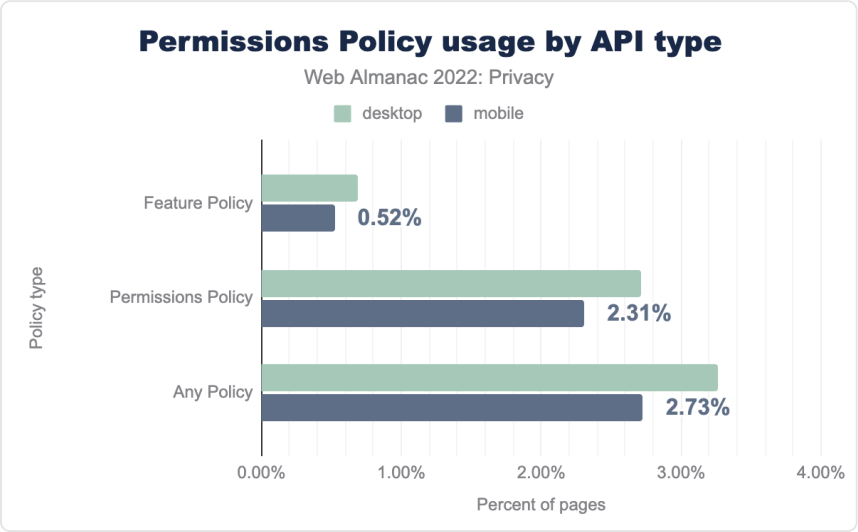



*図13.14. APIタイプ別のパーミッションポリシーの使用状況。*

デフォルトでは、サードパーティのスクリプトは、組み込まれているウェブサイトと同じブラウザ機能にアクセスできます。ウェブサイトが有効にする機能を制限するために、ウェブサイトはパーミッションポリシー[487]を利用できます。HTTPレスポンスヘッダーを通じて、ウェブサイトはどの機能を許可するかを示すことができます。例えば、`microphone`機能がこのリストに含まれていない場合、ウェブページに組み込まれているスクリプトはそれを使用することができません。このポリシーは比較的新しいものですが、デスクトップサイトの2.71%、モバイルサイトの2.31%で採用が見られます。

パーミッションポリシーは、フィーチャーポリシー[488]を置き換えるもので、デスクトップサイトの0.69%、モバイルサイトの0.52%で依然として見られます。 デフォルトでは、パーミッションポリシーによって規制されるほとんどの機能は、クロスオリジンのiframeで無効になっていますが、`allow`属性を通じて明示的に有効にすることができます。デスクトップサイトの15.18%、モバイルサイトの14.32%がこの機能を利用しています。iframe上の`allow`属性の使用に関する詳細な分析については、[セキュリティ](./security#パーミッションポリシー )章を参照してください。

487. *https://developer.chrome.com/docs/privacy-securitypermissions-policy?hl=ja*
488. *https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Permissions-Policy*

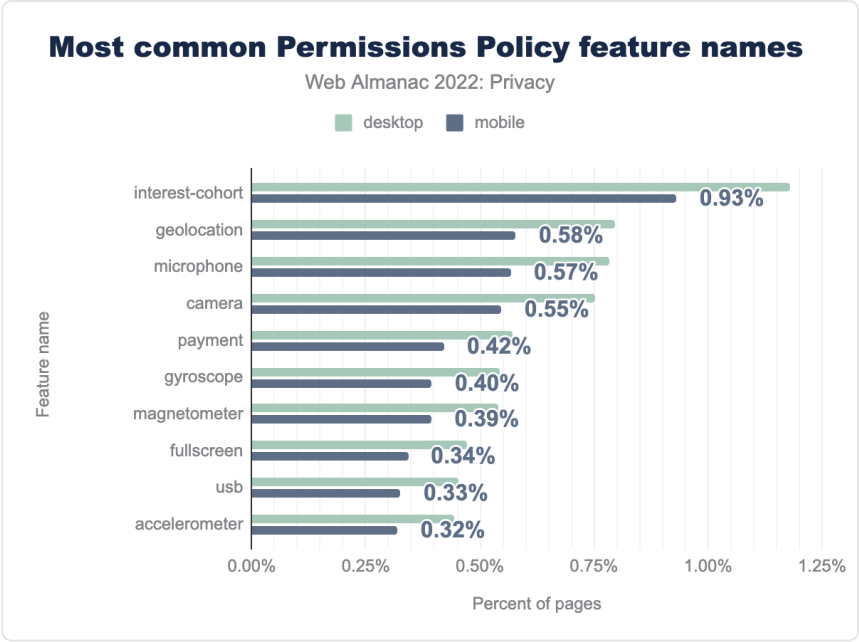



*図13.15. 最も一般的なパーミッションポリシー機能名。*

パーミッションポリシーで使用されるディレクティブを見ると、昨年[489]と比較して類似の使用状況が見られますが、2022年に最も広く使用されたのは`interest-cohort`ディレクティブです。このディレクティブは、現在は機能していないFLoC APIへのアクセスを制限するために使用されます。この増加は、FLoCのさまざまな欠点（フィンガープリンティングの表面を増加させる、ユーザーについての潜在的にセンシティブな情報を明らかにするなど）に対し、ウェブサイトの所有者や提供者、ライブラリがユーザーのプライバシーを保護するために積極的な対策を講じたためと思われます。

## リファラーポリシー

12%

*図13.16. ドキュメント全体のリファラーポリシーを設定するデスクトップサイトの割合。*

デフォルトでは、ほとんどのユーザーエージェントは`Referer`ヘッダーを含めます。簡単

489. https://almanac.httparchive.org/ja/2021/privacy

に言うと、これは第三者に対して、どのウェブサイトやページからリクエストが開始されたかを明らかにします。これはウェブページに埋め込まれたリソースに対しても、リンクをクリックした後に開始されたリクエストに対しても当てはまります。もちろん、これは望ましくない副作用として、これらの第三者が特定のユーザーが訪れていたウェブサイトやウェブページを知ることができるというものです。リファラーポリシー[490]を利用することで、ウェブサイトは`Referer`ヘッダーがリクエストに含まれるインスタンスを制限し、ユーザーのプライバシーを改善することができます。私たちは、12%のデスクトップサイトと10%のモバイルサイトが、主にHTTPレスポンスヘッダーを介して、そのようなドキュメント全体のポリシーを設定していることを発見しました。

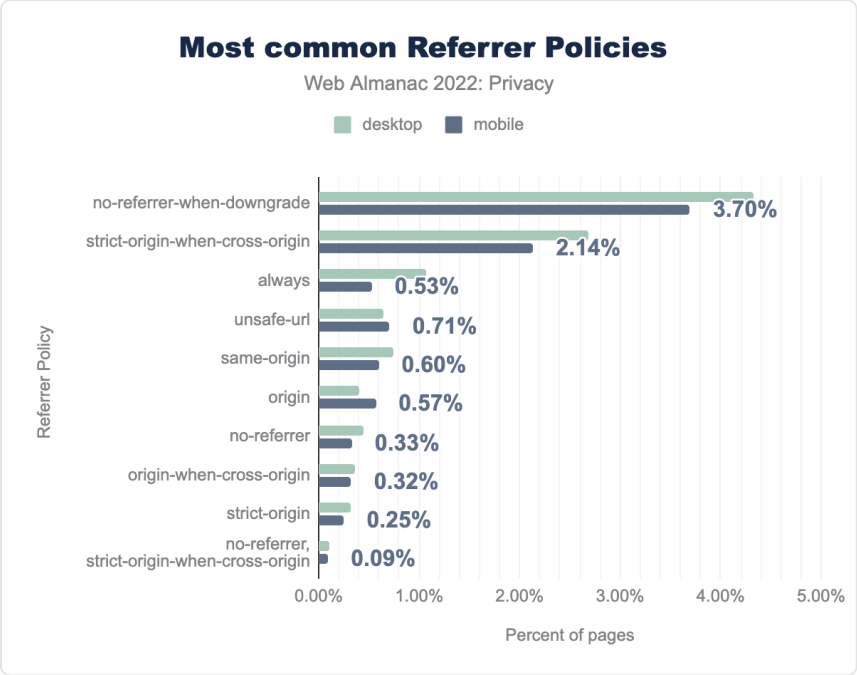



*図13.17. 最も一般的なリファラーポリシー。*

リファラーポリシーの使用状況を見ると、`Referer`ヘッダーをダウングレードリクエストに含めないことが最も一般的です。つまり、HTTPSが有効なページで開始されたHTTPリクエストには含まれません。残念ながら、これはほとんどのシナリオでHTTPSリクエストにおいてユーザーが訪れたページが漏れることを意味します。しかし、デスクトップサイトの2.7%とモバイルサイトの2.1%が`strict-origin-when-cross-origin`ポリシーを用いて、ユーザーが訪れた具体的なウェブページを隠そうとしています。これは現在、ポリシーが指定されていない場合のほとんどのブラウザのデフォルトです。

490. *https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Referrer-Policy*

## ユーザーエージェント・クライアント・ヒント

ブラウザ環境に関する情報、特に`User-Agent`文字列について公開される情報を減らすために、User-Agent Client Hints[491]メカニズムが導入されました。この機能を通じて、ユーザーのブラウジング環境（ブラウザのバージョン、オペレーティングシステムなど）について特定の情報を取得したいウェブサイトは、最初のレスポンスでヘッダー（`Accept-CH`）を設定する必要があります。これにより、ブラウザは後続のリクエストで要求されたデータを送信します。この機能は、フィンガープリンティングの表面を減らす他、ブラウザが特定のデータの送信を介入することを可能にします（例えば、Privacy Budget[492]提案を通じて）。

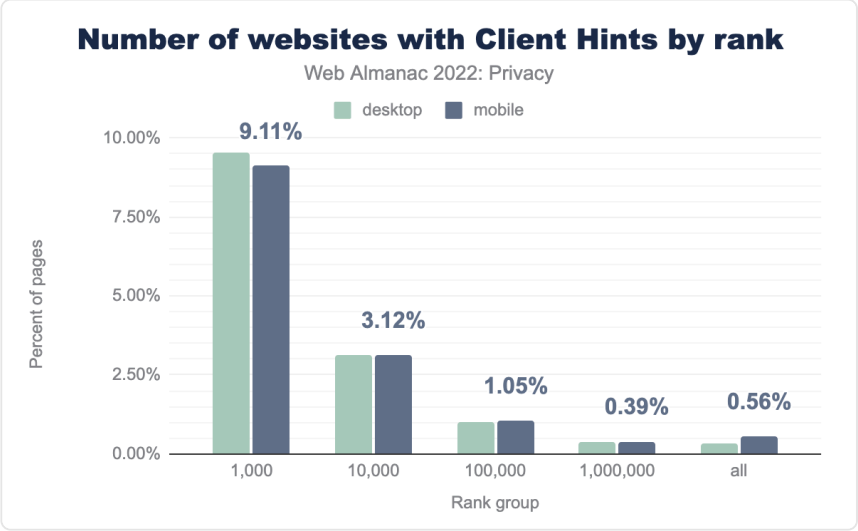



*図13.18. ランクグループ別のクライアントヒントを使用するウェブサイトの数。*

昨年（トップ1k：3.56%、トップ10k：1.44%）と比較して、`Accept-CH`ヘッダーで応答するサイトの採用を見ると、特に最も人気のあるサイトでの採用がほぼ3倍に増加しています。この採用の増加は、Chromiumが`User-Agent`文字列で共有される情報を減らしている事実（User-Agent Reduction plan[493]を通じて）に関連していると考えられます。

ユーザーエージェント・クライアント・ヒントを利用するサイトは、一般的に比較的多くのプロパティへのアクセスを要求しており、ブラウザがUser-Agent Reductionなどの努力を通じて達成しようとしている利益を制限しています。近い将来、ブラウザがユーザーのブラウジング環境に関する多くの情報を取得する慣行をどのように制限するかを見るのが興味深いでしょう。

491. https://wicg.github.io/ua-client-hints/
492. https://github.com/mikewest/privacy-budget
493. https://www.chromium.org/updates/ua-reduction/

# ブラウザによるプライバシーの改善のための新たな取り組み

過去数年間で、一般のウェブユーザーはオンラインでのプライバシーに対してますます意識が高まっています。一方で、ますます大きくなるデータ漏洩が頻発しており、ほとんどの人が影響を受けていません。また、サードパーティクッキーを通じたユーザーの広範な追跡が一般によく知られるようになっています。その結果、ますます多くのユーザーが自分のブラウザにプライバシーを保護し、オンライン行動の追跡をより制御することを期待しています。ブラウザベンダー、オンラインパブリッシャー、広告技術会社は、プライバシーの改善に対するこの需要を聞き、Google Chromeが主導するプライバシーサンドボックスというイニシアチブを提案しています。

## プライバシーサンドボックスオリジントライアル

本年のWeb Almanacの公開時点では、プライバシーサンドボックスの機能は一般に利用可能ではありません。しかし、ウェブサイトやウェブサービス（例えば、iframe内で表示される広告など）は、オリジントライアル[494]を利用してプライバシーサンドボックス機能の初期テストに参加することができます。この機能をサポートしているブラウザを使用しているユーザーに限ります—プライバシーサンドボックス機能はChromeにのみ実装されており、この記事の執筆時点ではデフォルトで無効になっています。これにより、ウェブサービスはプライバシーサンドボックス関連のAPIであるTopics[495]、FLEDGE[496]、およびAttribution Reporting[497]にアクセスできます。

494. *https://developers.google.com/privacy-sandbox/blog/privacy-sandbox-unified-origin-trial?hl=ja*
495. *https://developers.google.com/privacy-sandbox/relevance/topics/developer-guide?hl=ja*
496. *https://developers.google.com/privacy-sandbox/relevance/protected-audience?hl=ja*
497. *https://developers.google.com/privacy-sandbox/relevance/attribution-reporting?hl=ja*

| 機能を要求するオリジン | デスクトップ | モバイル |
|---|---|---|
| https://www.googletagmanager.com | 12.53% | 10.99% |
| https://googletagservices.com | 11.05% | 10.52% |
| https://doubleclick.net | 11.04% | 10.51% |
| https://googlesyndication.com | 11.04% | 10.51% |
| https://googleadservices.com | 2.50% | 2.29% |
| https://s.pinimg.com | 1.49% | 1.21% |
| https://criteo.net | 0.64% | 0.41% |
| https://criteo.com | 0.59% | 0.37% |
| https://imasdk.googleapis.com | 0.10% | 0.07% |
| https://teads.tv | 0.04% | 0.03% |

*図13.19. プライバシーサンドボックスAPIオリジントライアルにアクセスを要求するオリジンの普及率。*

プライバシーサンドボックスのオリジントライアル中にテストを行うウェブ上で最も普及しているサービスは、Google Tag Manager、Doubleclick、Google Syndication、そしてGoogle Ad Servicesです。これらはデスクトップとモバイルサイトの両方でトップ5に入っています。これに続くのはソーシャルメディアサイトのPinterestや、他のトラッカーや広告主であるCriteo、Google Ads SDK、Teadsです。

## プライバシーサンドボックス実験

プライバシーサンドボックスイニシアティブは、さまざまな側面に関わる多くの異なる機能で構成されており、サードパーティクッキーが段階的に廃止される際に、ユーザーがウェブ上で行う一般的なアクションを依然としてサポートすることを目指しています。ほとんどの機能はまだ積極的に開発中であり、ウェブサイトがそれらを採用しているわけではありません（`PrivacySandboxAdsAPIs`オリジントライアルにオプトインしているサービスを除く）。

一時期、プライバシーサンドボックス機能のオリジントライアルは、各機能ごとに分けられていました。これらの試用は現代のブラウジング環境では効果がありませんが、一部のウェブサービスはそれらにオプトインし、`Origin-Trial`レスポンスヘッダーの削除を忘れていました。

例えば、34,128のサイトでウェブサービスが `ConversionMeasurement` オリジントライアルにオプトインしているのが見られます。このトライアルは、かつてAttribution Reporting API[498]（以前はConversion Measurement APIと呼ばれていました）へのアクセスを提供していました。このAPIは、例えば広告をクリックしたユーザーの購入への変換を追跡するために使用されます。

また、有効期限が切れているTrustTokens[499]オリジントライアルにも、まだ6,005のサイトでウェブサービスがオプトインしています。このメカニズムは、一つのブラウジングコンテキスト（例えば、サイト）が別のコンテキストに限定された情報を伝達することを可能にすることで、詐欺と戦うことを目的としています。

興味深いことに、30,000以上のウェブサイトで、ウェブサービスがユーザーの興味グループのFLoCへのアクセスを提供する `InterestCohort` オリジントライアルにまだオプトインしています。しかし、APIのプライバシー上の懸念から、この取り組みは追求されなくなり、開発は中止されました。これに取って代わるのは、リマーケティングやカスタムオーディエンスのための「デバイス上での広告オークションを提供する」FLEDGE API[500]と、クロスサイト追跡を必要とせずにユーザーの興味に基づいて広告を配信することを目的とするTopics API[501]です。

## プライバシー規制への準拠

データプライバシー規制の領域は立法の最新のフロンティアとして拡大し続けています。これらの規制は、組織に対してユーザーのデータ処理についてより透明性を持たせ、そのデータを保護することを要求しています。一般データ保護規則（GDPR）[502]やIABの透明性と同意フレームワーク（TCF）v2.0[503]などの重要なデータプライバシー規制の登場を受けて、ウェブサイト提供者は訪問中に処理されるデータについてユーザーに情報を提供し、追跡や広告など非機能的な目的でのデータ処理についてもこれらのユーザーから同意を得るために行動を起こしました。これにより、ウェブサイト提供者が主に（クッキー）同意バナーを通じてユーザーに通知したり同意を求めたりするため、ウェブサイト上でクッキーバナーをより頻繁に見るようになりました。

ほとんどの場合、ユーザーはこのような同意バナーと対話し、どのデータを処理するかを設定することができます。しかし、ますます複雑になる現代の洗練されたウェブ上でこのようなタスクを管理することは容易ではありません。このため、ウェブサイト運営者はこのタスクをサードパーティであるいわゆる同意管理プラットフォーム（CMP）に委ねようとしています。CMPは、該当するウェブサイト上でクッキーが法律に従って使用されることを保証します。次に、CMPの使用とプライバシーポリシーの通知について議論します。

498. *https://developers.google.com/privacy-sandbox/relevance/attribution-reporting?hl=ja*
499. *https://developers.google.com/privacy-sandbox/protections/private-state-tokens?hl=ja*
500. *https://developers.google.com/privacy-sandbox/relevance/protected-audience?hl=ja*
501. *https://developers.google.com/privacy-sandbox/relevance/topics/developer-guide?hl=ja*
502. *https://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf*
503. *https://www.iabeurope.eu/*

## 同意管理プラットフォーム（CMP）

すでに議論したように、同意管理プラットフォーム（CMP）の使用は、特にクッキーの扱いに関して、ウェブサイトが法的に適合する方法で運用されることを保証すべきです。

この時点で、CMPサービスの統合がウェブサイトが法的に適合する状態を常に保証するわけではないことも指摘しておくべきです。この分野の研究（例えば、Santos et al.[504]やFouad et al.[505]）が示しています。

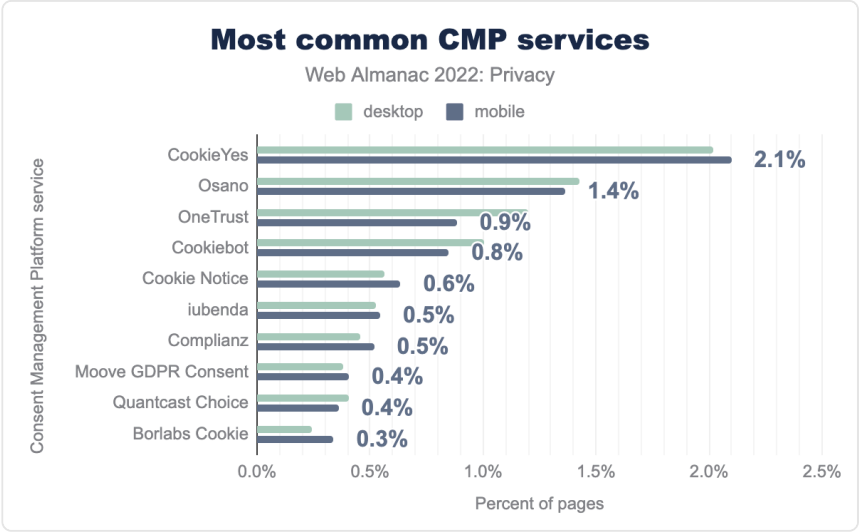



*図13.20. 最も一般的な同意管理プラットフォーム（CMP）サービス。*

私たちの分析によると、CMPの使用は昨年から11%へと7%から増加し、ほぼ60%の増加を記録しました。また、今年はモバイルよりもデスクトップの方が関与が少ないことが見て取れますが、その差は最小限です。 また、CookieYes（18%増）、OneTrust（64%増）、Cookiebot（56%増）といったプロバイダーが昨年から市場シェアを増やしていることも確認できます。

## IAB同意フレームワーク

GDPRと比較して、IAB Europe透明性および同意フレームワーク（TCF）[506]は、グローバルベンダー[507]が関与する業界標準です。その目的は、ユーザーの同意と広告主との間のコミュニケーションを確立することです。TCFは、ヨーロッパのウェブサイトがGDPRに準拠してい

504. *https://arxiv.org/abs/2104.06861*
505. *https://ieeexplore.ieee.org/document/9229842*
506. *https://iabeurope.eu*
507. *https://iabeurope.eu/vendor-list/*

ることを保証します。IAB Tech Lab USが開発したアメリカ合衆国プライバシー技術仕様（USP）[508]は、TCFと同じコンセプトを使用してアメリカ合衆国向けに設計されました。

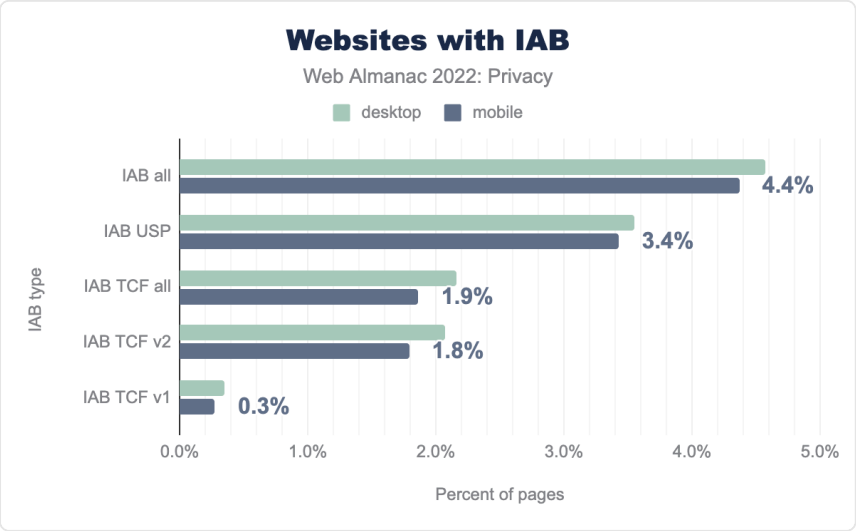



*図13.21. IABを使用しているウェブサイト。*

デスクトップウェブサイトの4.6%で何らかのIABが使用されており、そのうち3.5%がUSPを、2.2%がIAB TCFを使用しています。これは昨年からの両仕様の使用増加を示しています。ここで注記すべきは、私たちの測定は米国ベースであるため、TCFによれば、EU外の訪問には同意バナーが必要ないため、USPを使用しているウェブサイトが多いことが理由の一つと考えられます。

508. *https://iabtechlab.com/standards/ccpa/*

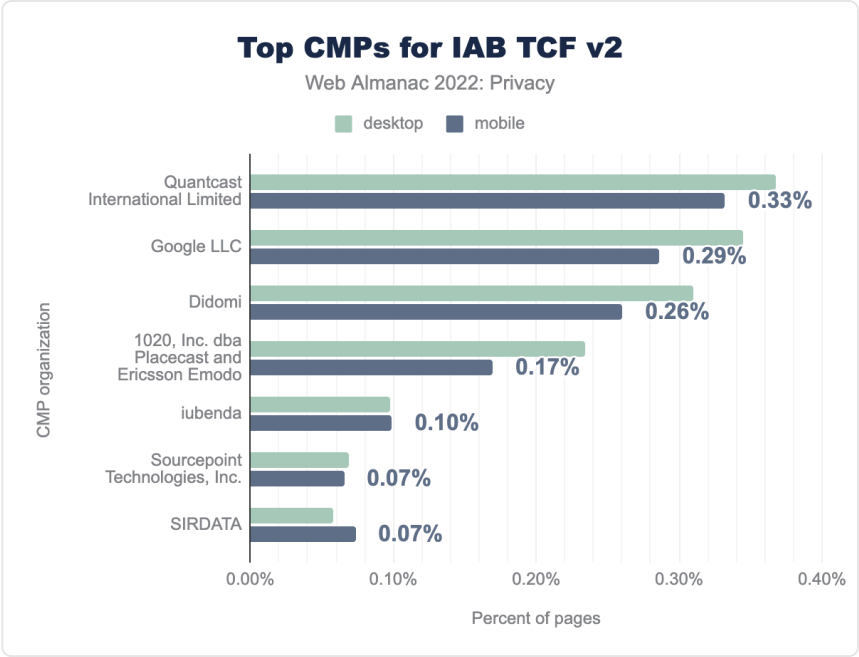



*図13.22. IAB TCF v2のためのトップCMP。*

Quantcast International Limited (0.37%)、Google LLC (0.34%)、およびDidomi (0.31%) がIAB TCF v2のための人気のあるCMPプロバイダーです。

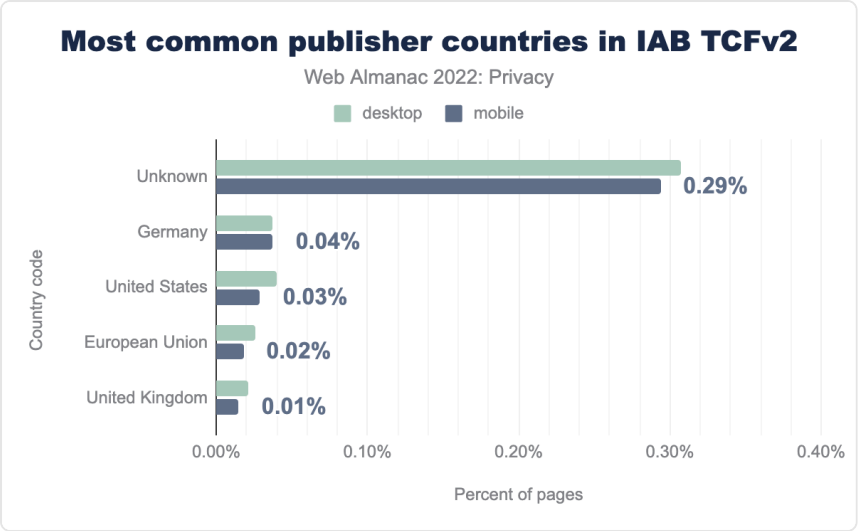



*図13.23. IAB TCF v2を使用しているパブリッシャーの最も一般的な国。*

私たちの分析によると、特定された最も一般的なパブリッシャーはドイツ、米国、およびEUからです。

## プライバシーポリシー

データ処理に関する通知は、同意バナーを介して行われるだけではありません。これらは通常、そのようなバナーと比較して別のページでより詳細に説明されます。このようなページでは、統合されたサードパーティ、どのデータがどの目的で処理されるかなどの情報を見つけることができます。このようなサイトを特定するために、ある研究[509]からのプライバシーに関連するシグネチャを使用しました。この方法を使用して、デスクトップウェブサイトの45%（モバイルでは41%）がホームページにプライバシー関連ページへのリンクを含んでいることが判明しました。以下の図は、トッププライバシーリンクキーワードの分布を示しています。

509. https://github.com/RUB-SysSec/we-value-your-privacy/blob/master/privacy_wording.json

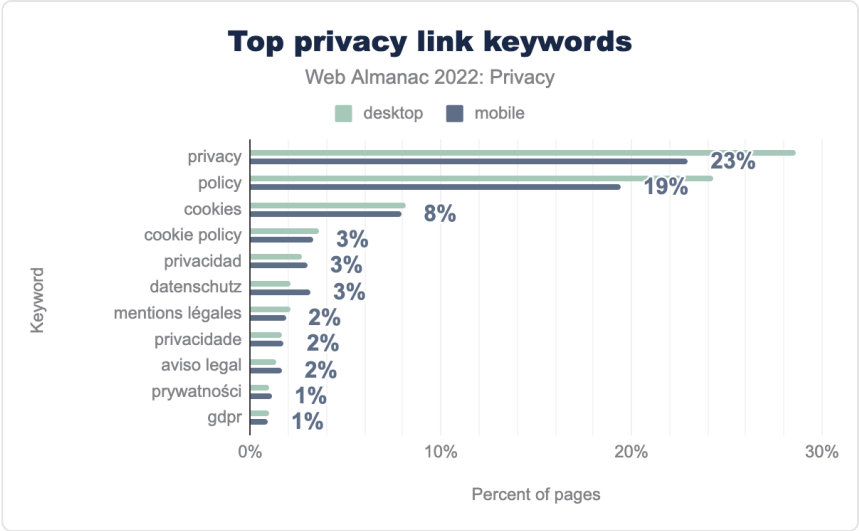



*図13.24. プライバシーリンクの主要キーワード。*

プライバシー（29%）、ポリシー（24%）、クッキー（8%）がこのようなリンクのトップキーワードです。

## 結論

この章では、ウェブ上のオンラインプライバシーに関連するさまざまな側面を探求しました。過去1年間にプライバシーに影響を与える多くの変化があったことは明らかであり、今後もこの進展が続くことが期待されます。簡単に言うと、私たちの前には興味深い時代が広がっています。一方で、いつかウェブの遺産として参照できることを願っている不幸な進化も見つかりました。第三者トラッキングは、主に第三者クッキーによって推進され、82%以上のウェブサイトに少なくとも1つのトラッカーが含まれており、依然として普及しています。さらに、反トラッキング対策を回避するための逃避的な手法を採用しているウェブサイトやウェブサービスも少なくありません。

より前向きでプライバシーを尊重する方向で、ブラウザAPIから潜在的にセンシティブな情報にアクセスしようとするサイトが減少していることがわかります。この傾向が、定期的にブラウザに導入される新しいAPIでも続くことを願っています。

一般的に、ウェブサイトはユーザーのプライバシーを尊重するという呼びかけに応じ始めているようです。この呼びかけはますます大きくなっています。ますます多くのサイトが、第三者に送信される情報を制限するブラウザ機能を採用しています。さらに、GDPRやCCPAなどのプライバシー規制に主に動機づけられて、同意管理プラットフォーム（CMP）の採用

が明確に増加しており、ユーザーが共有したい情報をよりコントロールできるようになっています。

最後に、ブラウザの側でも、オンラインプライバシーの管理を強化するための強力な進化が見られます。プライバシーに焦点を当てたいくつかのブラウザが内蔵ソリューションとして提供する機能に加えて、プライバシーサンドボックスイニシアティブも、クロスサイトトラッキングの悪影響なしに、ウェブ上での現在の機能（ターゲット広告、詐欺防止、購入の帰属など）を提供し続けることを目指しています。開発はまだかなり初期段階にありますが、多くのウェブサイトでウェブサービスがオリジントライアルにオプトインしていることが見られます。この機能は広範囲にテストされており、ウェブの恒久的な部分になる可能性が高いです。

完全にそこに到達するまでには数年かかるかもしれませんが、ユーザーがどの当事者とどの情報を共有したいかをよりコントロールできるウェブへと移行しています。この収束はスペクトルの両側で見られます：一方ではウェブサイトによって、もう一方ではブラウザによって強制されます。近い将来、私たちが共有するデータは意図したデータであり、日常的に行うウェブ上の旅が、現在遭遇している数多くのトラッカーによって収集、共有、分析される必要がなくなることを期待できます。これは、明日のための敬意を表しています。

## 著者


### Tom Van Goethem

@tomvangoethem　tomvangoethem　https://tom.vg/

Tom Van Goethemは最近、GoogleのChromeプライバシーチームに参加しました。それ以前には、ベルギーのルーヴェン大学のDistriNetグループで博士課程に在籍していました。彼の研究興味は、ウェブセキュリティとプライバシーの分野における幅広いトピックをカバーしており、特にサイドチャネル攻撃に重点を置いています。脅威を明らかにし、緩和策を提案することで、Tomはウェブを少しずつでもより良い場所にすることを目指しています。

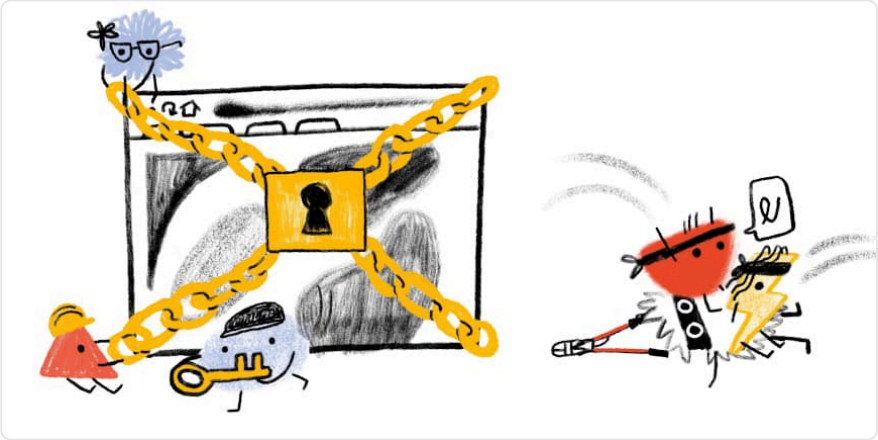
### Nurullah Demir

@nrllah　nrllh　https://www.internet-sicherheit.de/team/demir-nurullah.html

Nurullah Demirは、サイバーセキュリティ研究者であり、インターネットセキュリティ研究所[510]およびインテリジェントシステムセキュリティ、KASTELセキュリティ研究ラボ[511]の博士課程の学生です。彼の研究はウェブセキュリティとプライバシー、そしてウェブ計測に焦点を当てています。

510. *https://www.internet-sicherheit.de/en/*
511. *https://intellisec.de*

# 部 II 章 14
# セキュリティ



***Saptak Sengupta、 Liran Tal と Brian Clark によって書かれた。***
*Kushal Das と Barry Pollard によってレビュ。*
*Victor Le Pochat、 Vik Vanderlinden と Gertjan Franken による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

人々の個人情報がさらにデジタル化するにつれて、インターネット全体でセキュリティとプライバシーが非常に重要になっています。ウェブサイトのオーナーには、ユーザーから収集したデータを安全に保つ責任があります。そのため、ユーザーをマルウェアが悪用可能な脆弱性から保護するために、すべてのセキュリティベストプラクティスを採用することが不可欠です。

過去の年同様[512]、私たちはウェブコミュニティによるセキュリティ方法とベストプラクティスの採用と使用を分析しました。私たちは、すべてのウェブサイトが採用すべき基本的なセキュリティ対策に関連する指標を分析しました。これには、トランスポートセキュリティや適切なクッキー管理などが含まれます。また、異なるセキュリティヘッダーの採用と、それらがコンテンツの取り込みやさまざまな悪意のある攻撃の防止にどのように役立つかについ

512. *https://almanac.httparchive.org/ja/2021/security*

ても議論しました。

私たちは、セキュリティ対策の採用を地域、技術スタック、ウェブサイトの人気度との相関関係を調べました。このような相関関係が、すべての技術スタックがデフォルトでより良いセキュリティ対策を目指すことを促進することを願っています。また、Web Application Security Working Groupの標準とドラフトに基づいて、脆弱性開示やその他のセキュリティ関連の設定を支援するよく知られたURIについても議論します。

## トランスポートセキュリティ

トランスポート層セキュリティは、ユーザーとウェブサイト間のデータとリソースの安全な通信を保証します。HTTPS[513]は、クライアントとサーバー間のすべての通信を暗号化するためにTLS[514]を使用します。


*図14.1. デスクトップでHTTPSを使用するリクエスト。*

デスクトップでの総リクエストの94%、モバイルでの総リクエストの93%がHTTPS経由で行われています。すべての主要ブラウザーには、ウェブサイトがHTTPSではなくHTTPを使用している場合に警告を表示するHTTPS-only モード[515]があります。

513. *https://developer.mozilla.org/ja/docs/Glossary/https*
514. *https://www.cloudflare.com/ja-jp/learning/ssl/transport-layer-security-tls/*
515. *https://support.mozilla.org/ja/kb/https-only-prefs*

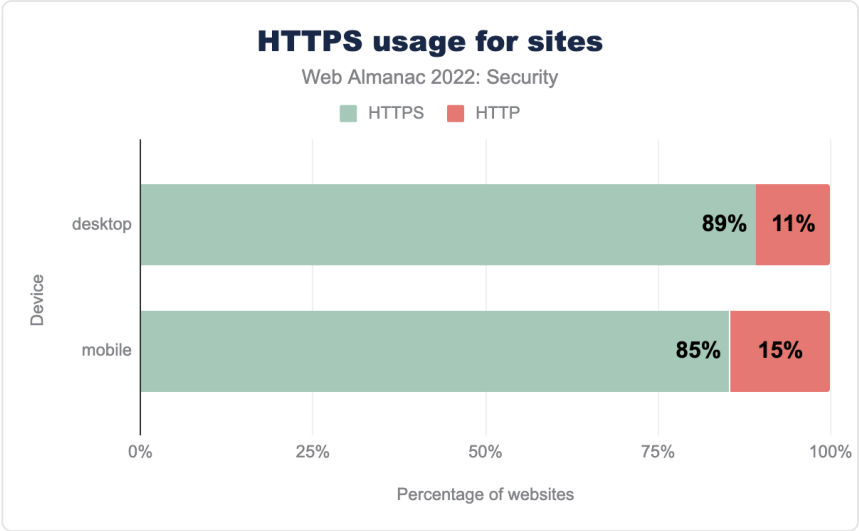



*図14.2. サイトにおけるHTTPSの使用状況。*

総リクエストに比べてHTTPS経由で提供されるホームページの割合は低いままです。なぜなら、ウェブサイトへの多くのリクエストがフォントやCDNなどのサードパーティサービスによって占められており、これらはHTTPS採用率が高いからです。昨年に比べてHTTPS経由で提供されるホームページの割合はわずかに増加しています。デスクトップでは昨年の84.3%から89.3%に、モバイルでは昨年の81.2%から85.5%に増加しています。

## プロトコルバージョン

HTTPSを使用するだけでなく、最新のTLSバージョンを使用することも重要です。TLSワーキンググループ[516]は、複数の弱点があったため、TLS v1.0およびv1.1を非推奨としました。昨年の章以降、FirefoxはUIを更新[517]し、Firefoxバージョン97のエラーページからTLS 1.0と1.1を有効にするオプションが削除されました。Chromeもバージョン98以降、TLS 1.0と1.1のエラーページのバイパスを許可しなくなりました[518]。

TLS v1.3は最新のバージョンで、2018年8月にIETFによってリリースされました。TLS v1.3はTLS v1.2よりもはるかに高速で、より安全です[519]。TLS v1.2の主要な脆弱性の多くは、古い暗号化アルゴリズムに関連しており、TLS v1.3はこれらを排除します。

516. https://datatracker.ietf.org/doc/rfc8996/
517. https://support.mozilla.org/ja/kb/secure-connection-failed-error-message#w_an-quan-najie-sok-gadekimasendeshita
518. https://chromestatus.com/feature/5759116003770368
519. https://www.cloudflare.com/ja-jp/learning/ssl/why-use-tls-1.3/

*図14.3. サイトにおけるTLSバージョンの使用。*

上のグラフでは、モバイルのホームページの70%、デスクトップのホームページの67%がTLSv1.3で提供されており、昨年から約7%増加しています。したがって、TLS v1.2からTLS v1.3への一定のシフトが見られます。

## 暗号スイート

暗号スイート[520]は、クライアントとサーバーがTLSを使用して安全に通信を開始する前に合意する必要がある暗号化アルゴリズムのセットです。

520. *https://learn.microsoft.com/ja-jp/windows/win32/secauthn/cipher-suites-in-schannel*

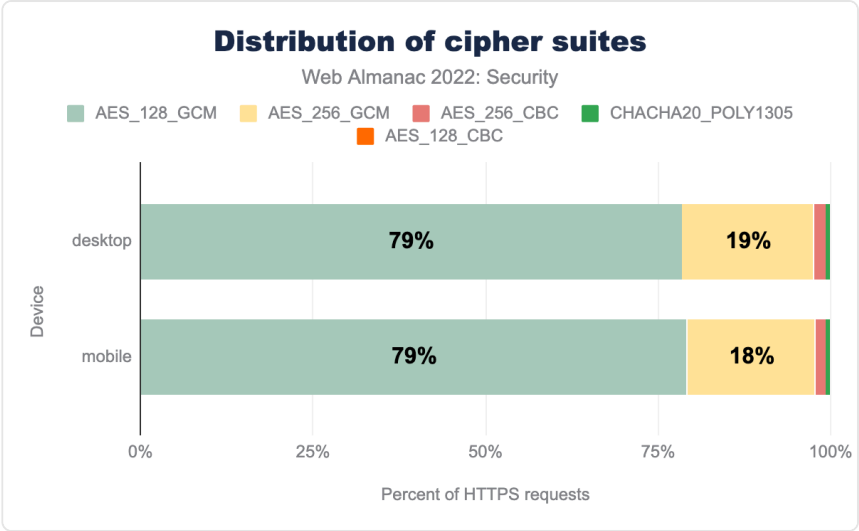



*図14.4. 暗号スイートの分布。*

現代のGalois/Counter Mode (GCM)[521]暗号モードは、パディング攻撃[522]に弱くないため、はるかに安全とされています。TLS v1.3はモダンなブロック暗号モードのみをサポートしており[523]、より安全です。また、暗号スイートの順序付けの問題も解消しています[524]。暗号と復号に使用される鍵のサイズも暗号スイートの使用を決定する要因の1つです。依然として128ビット鍵サイズが広く使用されているため、昨年のグラフと大きな違いは見られません。`AES_128_GCM`が依然としてもっとも使用されている暗号スイートで、使用率は79%です。

521. *https://ja.wikipedia.org/wiki/Galois/Counter_Mode*
522. *https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities*
523. *https://datatracker.ietf.org/doc/html/rfc8446#page-133*
524. *https://go.dev/blog/tls-cipher-suites*

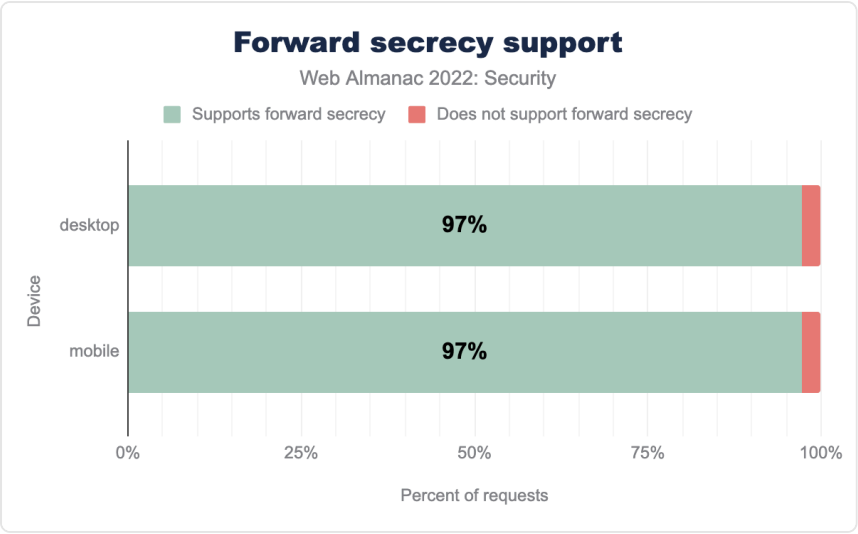



*図14.5. フォワードシークレシーの使用。*

TLS v1.3は、フォワードシークレシー[525]を義務付けています。モバイルとデスクトップのリクエストの97%でフォワードシークレシーが使用されています。

## 証明書認証局

証明書認証局[526]（CA）は、ウェブサイトにTLS証明書を発行し、ブラウザに認識され、ウェブサイトとの安全な通信チャネルを確立することができる企業または組織です。

525. *https://ja.wikipedia.org/wiki/Forward_secrecy*
526. *https://www.ssl.com/faqs/what-is-a-certificate-authority/*

| 発行者 | デスクトップ | モバイル |
|---|---|---|
| R3 | 48% | 52% |
| Cloudflare Inc ECC CA-3 | 13% | 12% |
| Sectigo RSA Domain Validation Secure Server CA | 7% | 8% |
| cPanel, Inc. Certification Authority | 5% | 5% |
| Amazon | 3% | 3% |
| Go Daddy Secure Certificate Authority - G2 | 3% | 2% |
| DigiCert SHA2 Secure Server CA | 2% | 1% |
| RapidSSL TLS DV RSA Mixed SHA256 2020 CA-1 | 1% | 1% |
| E1 | 1% | 1% |

*図14.6. ウェブサイトのトップ10証明書発行者。*

R3（Let's Encrypt）[527]は、デスクトップのウェブサイトの48%、モバイルのウェブサイトの52%が彼らによって発行された証明書を使用しており、引き続きチャートをリードしています。非営利組織であるLet's EncryptはHTTPSの採用に重要な役割を果たしており、引き続き多くの証明書を発行しています[528]。また、残念ながら最近亡くなったその創設者の一人、Peter Eckersly[529]を記念する瞬間を持ちたいと思います。

Cloudflare[530]も、顧客に無料で証明書を提供することで2位を維持しています。CloudflareのCDNは、RSA証明書よりも小さく効率的な楕円曲線暗号（ECC）[531]証明書の使用を増やしていますが、古いクライアントに非ECC証明書も提供する必要があるため、展開が困難なことが多いです。Let's EncryptとCloudflareの割合が増加するにつれて、他のCAの使用率が少し減少しているのが見られます。

## HTTP Strict Transport Security

HTTP Strict Transport Security (HSTS)[532]は、ブラウザに対してHTTPを使用してサイトにアクセスしようとするすべての試みをHTTPSリクエストに自動的に変換するように指示するレスポンスヘッダーです。

527. https://letsencrypt.org/
528. https://letsencrypt.org/stats/#daily-issuance
529. https://community.letsencrypt.org/t/peter-eckersley-may-his-memory-be-a-blessing/183854
530. https://developers.cloudflare.com/ssl/ssl-tls/certificate-authorities/
531. https://www.digicert.com/faq/ecc.htm
532. https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Strict-Transport-Security

# 25%

*図14.7. HSTSヘッダーがあるモバイルリクエスト。*

モバイルレスポンスの25%とデスクトップレスポンスの28%にHSTSヘッダーがあります。

HSTSは`Strict-Transport-Security`ヘッダーを使用して設定され、`max-age`、`includeSubDomains`、`preload`の3つの異なるディレクティブを持つことができます。`max-age`はブラウザがサイトにHTTPSを使用してのみアクセスすることを覚えておくべき時間（秒単位）を示すのに役立ちます。`max-age`はヘッダーに必須のディレクティブです。

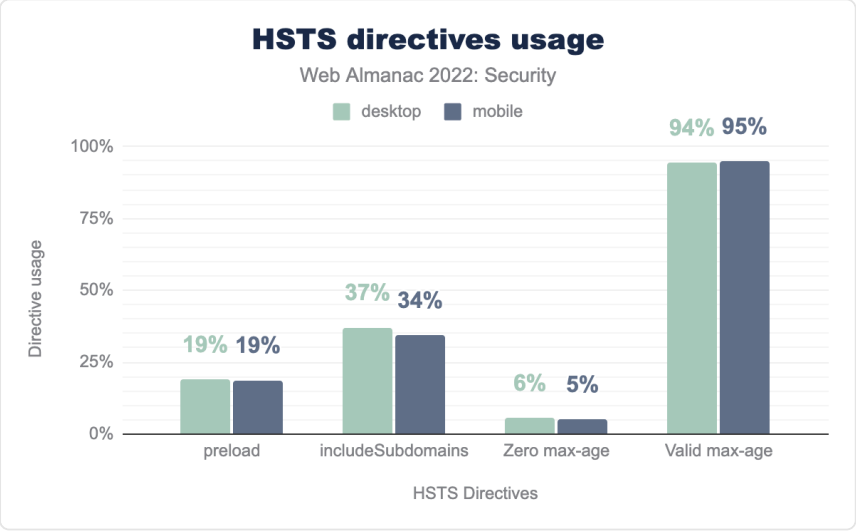



*図14.8. 異なるHSTSディレクティブの使用。*

デスクトップのサイトの94%とモバイルのサイトの95%には、ゼロではなく空ではない`max-age`が設定されています。

モバイルのリクエストレスポンスの34%とデスクトップの37%がHSTS設定に`includeSubdomain`を含んでいます。HSTS仕様の一部ではない`preload`ディレクティブを含むレスポンスは少なく、少なくとも31,536,000秒（または1年）の`max-age`と`includeSubdomain`ディレクティブの存在が必要です。

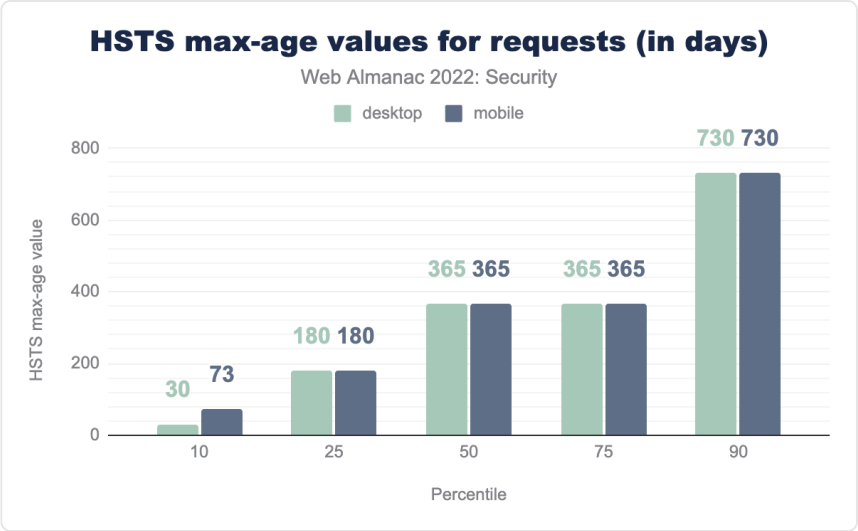



*図14.9. すべてのリクエストのHSTS `max-age` 値（日数）。*

HSTSヘッダーの `max-age` 属性のすべてのリクエストに対する中央値は、モバイルとデスクトップの両方で365日です。hstspreload.org[533]は、HSTSヘッダーが適切に設定されて問題がないことが確認された場合、 `max-age` を2年間に設定することを推奨しています。

## クッキー

HTTPクッキー[534]は、サーバーがブラウザに送信するユーザーに関するデータのセットです。クッキーは、セッション管理、パーソナライゼーション、追跡、その他のユーザーに関連する状態情報を異なるリクエストに渡って使用できます。

クッキーが適切に設定されていない場合、セッションハイジャック[535]、クロスサイトリクエストフォージェリ(CSRF)[536]、クロスサイトスクリプトインクルージョン(XSSI)[537]、その他のクロスサイトリーク[538]の脆弱性など、多くの異なる形態の攻撃に対して脆弱になる可能性があります。

533. *https://hstspreload.org/*
534. *https://developer.mozilla.org/ja/docs/Web/HTTP/Cookies*
535. *https://owasp.org/www-community/attacks/Session_hijacking_attack*
536. *https://owasp.org/www-community/attacks/csrf*
537. *https://owasp.org/www-project-web-security-testing-guide/v41/4-Web_Application_Security_Testing/11-Client_Side_Testing/13-Testing_for_Cross_Site_Script_Inclusion*
538. *https://xsleaks.dev/*

## Cookie属性

上記の脅威に対抗するために、開発者はクッキーに3つの異なる属性を使用できます：`HttpOnly`、`Secure`、そして`SameSite`です。`Secure`属性は`HSTS`ヘッダーと同様で、クッキーが常にHTTPS経由で送信されることを保証し、Manipulator in the Middle攻撃[539]を防ぎます。`HttpOnly`は、クッキーがJavaScriptコードからアクセスできないようにすることで、クロスサイトスクリプティング攻撃[540]を防ぎます。

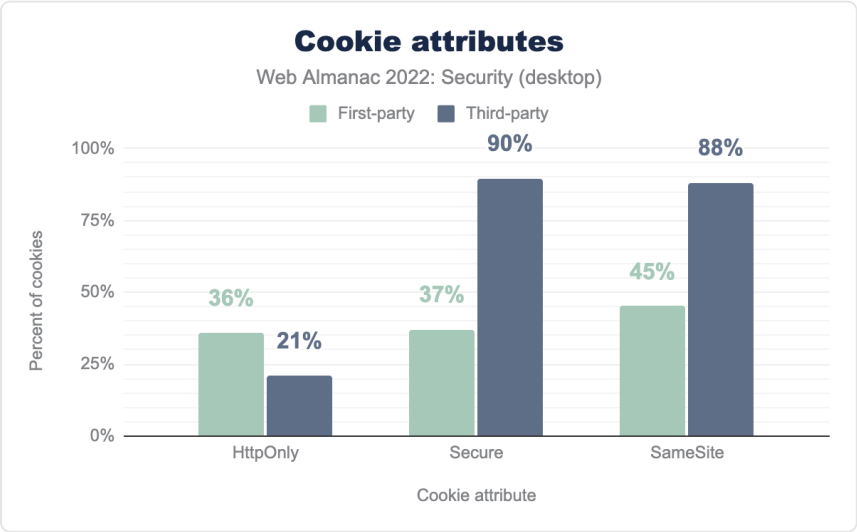



*図14.10. クッキー属性（デスクトップ）。*

クッキーにはファーストパーティとサードパーティの2種類があります。ファーストパーティのクッキーは通常、訪問している直接のサーバーによって設定されます。サードパーティのクッキーはサードパーティのサービスによって作成され、しばしばトラッキングや広告配信に使用されます。デスクトップでのファーストパーティクッキーの37%に`Secure`があり、36%に`HttpOnly`が設定されています。しかし、サードパーティクッキーでは、クッキーの90%に`Secure`があり、21%に`HttpOnly`が設定されています。サードパーティのクッキーでは`HttpOnly`の割合が少ないのは、JavaScriptコードによるアクセスを許可したいためです。

539. *https://owasp.org/www-community/attacks/Manipulator-in-the-middle_attack*
540. *https://owasp.org/www-community/attacks/xss/*

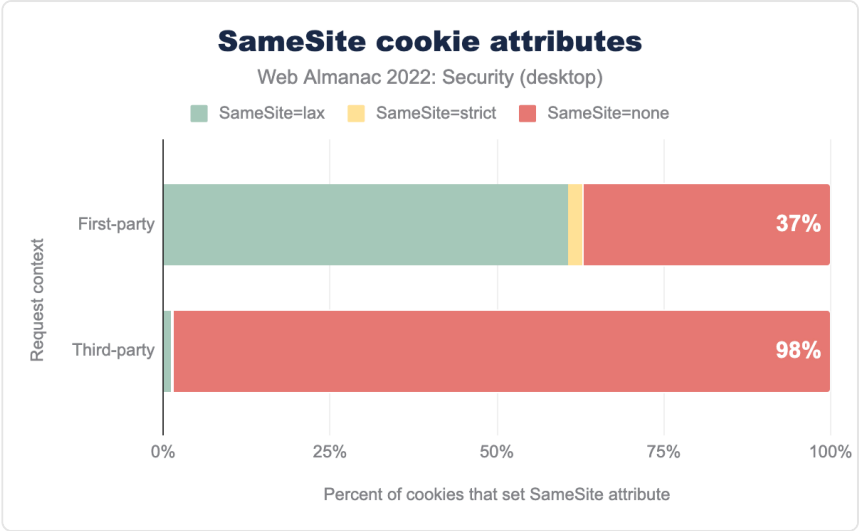



*図14.11. Same site cookie属性。*

`SameSite`属性は、ブラウザにクロスサイトリクエストにクッキーを送信するかどうかを指示することで、CSRF攻撃を防ぐために使用できます。`Strict`値はクッキーが元々のサイトからのみ送信されることを許可し、`Lax`値はユーザーがリンクをたどって元のサイトにナビゲートする場合にのみ、クロスサイトリクエストにクッキーを送信することを許可します。`None`値の場合、クッキーは発信元とクロスサイトのリクエストの両方に送信されます。`SameSite=None`が設定されている場合、クッキーの`Secure`属性も設定されている必要があります（そうでない場合、クッキーはブロックされます）。`SameSite`属性を持つファーストパーティクッキーの61%が`Lax`値を持っています。`SameSite`属性がない場合、ほとんどのブラウザはデフォルトで`SameSite=Lax`を使用するため、これがチャートを支配し続けるのを見ています。サードパーティのクッキーでは、`SameSite=None`がデスクトップでのクッキーの98%で非常に高いままです。これは、サードパーティのクッキーがクロスサイトリクエストで送信されたいと考えているためです。

## Cookieの寿命

クッキーが削除されるタイミングを設定するには、`Max-Age`と`Expires`の2つの方法があります。`Expires`はクライアントに関連する特定の日付を使用してクッキーが削除されるタイミングを決定し、`Max-Age`は秒単位の期間を使用します。

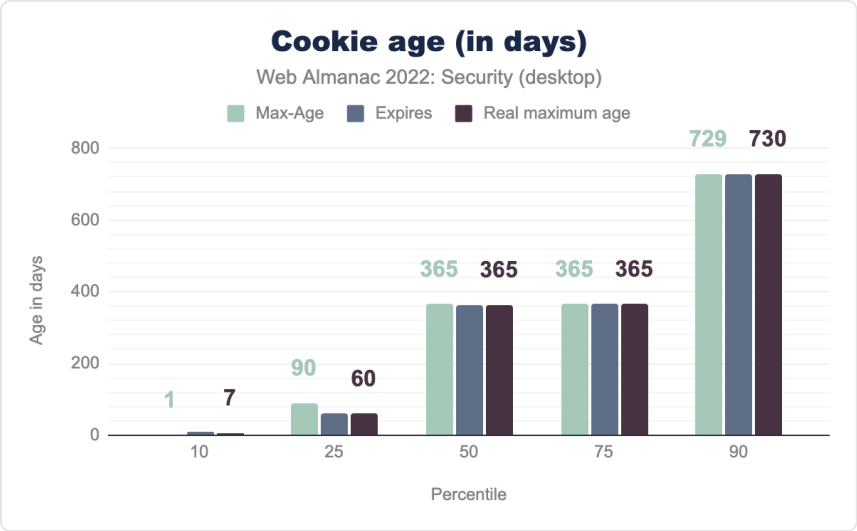



*図14.12. デスクトップでのCookieの寿命の使用（日数）。*

昨年と異なり、昨年は`Max-Age`の中央値が365日である一方で、`Expires`の中央値が180日でしたが、今年は両方とも約365日です。したがって、今年は実際の最大寿命の中央値が180日から365日に上がりました。`Max-Age`が729日で`Expires`が730日の90パーセンタイルにもかかわらず、Chromeは`Max-Age`と`Expires`の両方に400日の上限を設定する予定[541]です。

| % | Expires |
|---|---|
| 1.8% | *"Thu, 01-Jan-1970 00:00:00 GMT"* |
| 1.2% | *"Fri, 01-Aug-2008 22:45:55 GMT"* |
| 0.7% | *"Mon, 01-Mar-2004 00:00:00 GMT"* |
| 0.7% | *"Thu, 01-Jan-1970 00:00:01 GMT"* |
| 0.3% | *"Thu, 01 Jan 1970 00:00:00 GMT"* |

*図14.13. デスクトップでもっとも一般的なクッキーの有効期限の値。*

541. https://chromestatus.com/feature/4887741241229312

| % | Expires |
|---|---|
| 1.2% | "Fri, 01-Aug-2008 22:45:55 GMT" |
| 0.9% | "Thu, 01-Jan-1970 00:00:00 GMT" |
| 0.7% | "Mon, 01-Mar-2004 00:00:00 GMT" |
| 0.6% | "Thu, 01-Jan-1970 00:00:01 GMT" |
| 0.2% | "Thu, 31-Dec-37 23:55:55 GMT" |

*図14.14. モバイルでもっとも一般的なクッキーの有効期限の値。*

もっとも一般的な`Expires`には興味深い値があります。デスクトップでもっとも使用されている`Expires`値は「1970年1月1日00:00:00 GMT」です。クッキーの`Expires`値が過去の日付に設定されると、クッキーはブラウザから削除されます。1970年1月1日00:00:00 GMTはUnixエポックタイムであり、クッキーを期限切れにするか削除するためによく使用されます。

## コンテンツの組み込み

ウェブサイトのコンテンツは多様な形を取り、CSS、JavaScript、フォントや画像などのメディア資産といったリソースが必要とされます。これらは通常、コンテンツ配信を効率化するために、クラウドネイティブインフラのリモートストレージサービスやコンテンツ配信ネットワーク（CDN）からロードされます。

しかし、ウェブサイトに組み込むコンテンツが改ざんされていないことを保証することは非常に重要であり、その影響は壊滅的なものになる可能性があります。とくに最近ではサプライチェーンのセキュリティへの意識が高まり、Magecart攻撃[542]など、ウェブサイトのコンテンツシステムを狙った横断的なマルウェアの注入を通じて、クロスサイトスクリプティング（XSS）の脆弱性などを利用する事例が増加しているため、コンテンツの組み込みはさらに重要になっています。

### コンテンツセキュリティポリシー

コンテンツの組み込みに関連するセキュリティリスクを軽減するための効果的な手段として、コンテンツセキュリティポリシー（CSP）を採用できます。これは、クロスサイトスクリプティングによるコード注入やクリックジャッキングなどの攻撃を軽減するために、防御の深さを増すセキュリティ標準です。

542. https://www.imperva.com/learn/application-security/magecart/

これは、事前に定義された信頼できるコンテンツルールのセットが遵守され、制限されたコンテンツのバイパスや組み込みの試みが拒否されることを保証することで機能します。たとえば、ブラウザで実行されるJavaScriptコードを、それが提供された同一のオリジンとGoogle Analyticsからのみ許可するコンテンツセキュリティポリシーは、`script-src 'self' www.google-analytics.com;`として定義されます。`<a img=x onError=alert(1)>`のようなクロスサイトスクリプティング注入の試みは、設定されたポリシーを強制するブラウザによって拒否されます。


*図14.15. 2021年からのContent-Security-Policyヘッダーの採用率の相対的な増加。*

2021年のデータの12.8%から2022年のデータの14.6%へと、`Content-Security-Policy`ヘッダーの採用率が14%相対的に増加しています。これは開発者とウェブセキュリティコミュニティの間で採用傾向が高まっていることを示しています。これはポジティブな兆候ですが、依然としてこのより高度な機能を使用しているサイトは少数派です。

CSPは、HTMLレスポンス自体に提供される場合にもっとも有効です。ここでは、2年前に7.2%、昨年に9.3%、今年にはモバイルホームページの合計11.2%でCSPヘッダーが提供されていることから、採用が着実に増加していることがわかります。

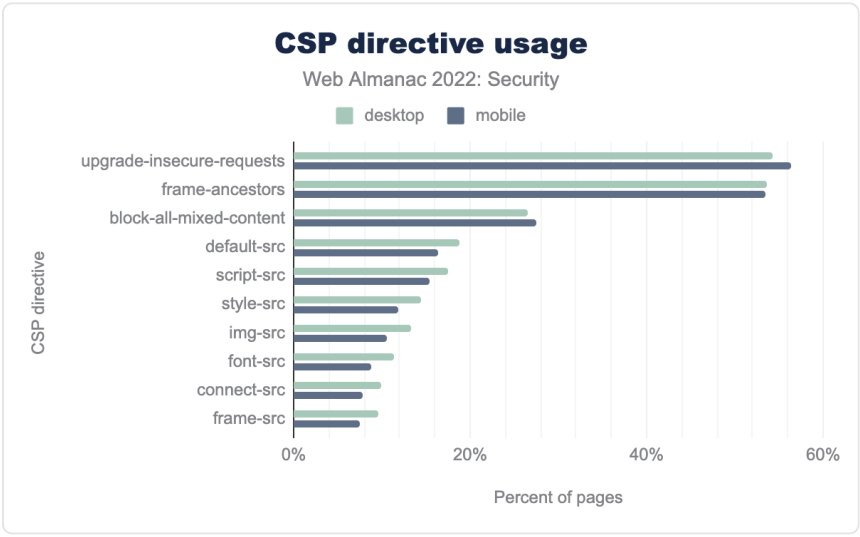



*図14.16. CSPでもっとも一般的に使用されるディレクティブ。*

デスクトップとモバイルのHTTPリクエストの四分の一以上で提供されている上位3つのCSPディレクティブは、`upgrade-insecure-requests`が54%、`frame-ancestors`が54%、`block-all-mixed-content`ポリシーが27%です。その他のポリシーには`default-src`、`script-src`、`style-src`、`img-src`などがあります。

`upgrade-insecure-requests`ポリシーの高い採用率は、TLSリクエストの高い採用が事実上の標準となっていることに起因する可能性があります。ただし、この日付時点で`block-all-mixed-content`が非推奨とされているにもかかわらず、高い採用率を示しています。これは、CSP仕様が急速に進化しており、ユーザーが最新の状態に追いつくのが困難であることを示しているかもしれません。

クロスサイトスクリプティング攻撃の軽減に関連して、Googleのセキュリティイニシアチブである Trusted Types[543]があります。これはDOMインジェクションクラスの脆弱性を防ぐための手法を採用するために、ネイティブブラウザAPIのサポートが必要です。これはGoogleエンジニアによって積極的に提唱されていますが、まだW3Cでドラフト提案[544]の段階にあります。それでも、リクエストの0.1%で関連するCSPセキュリティヘッダー`require-trusted-types-for`および`trusted-types`が記録されています、これは多くはありませんが、採用の成長傾向を示しているかもしれません。

定義済みのルールセットからのCSP違反が発生しているかどうかを評価するために、ウェブサイトは`report-uri`ディレクティブを設定できます。このディレクティブでは、ブラウザがJSON形式のデータをHTTP POSTリクエストとして送信します。`report-uri`リクエストはCSPヘッダーを持つすべてのデスクトップトラフィックの4.3%を占めていますが、現在は非推奨のディレクティブであり、`report-to`に置き換えられており、デスクトップリクエストの1.8%を占めています。

厳密なコンテンツセキュリティポリシーを実装する上での最大の課題の1つは、イベントハンドラーやDOMの他の部分に一般的に設定されるインラインJavaScriptコードの存在です。チームがCSPセキュリティ標準を段階的に採用できるようにするために、ポリシーは`unsafe-inline`または`unsafe-eval`を`script-src`ディレクティブのキーワード値として設定することがあります。これにより、一部のクロスサイトスクリプティング攻撃ベクトルを防ぐことができず、ポリシーの予防措置として逆効果になります。

チームは、インラインJavaScriptコードにノンスまたはSHA256ハッシュを使用して署名することで、より安全な態勢を取ることができます。これは次のような形式になります：

```
Content-Security-Policy: script-src 'nonce-4891cc7b20c'
```

そして、HTML内でそれを参照するには：

543. https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Content-Security-Policy/trusted-types
544. https://w3c.github.io/trusted-types/dist/spec/

```
<script nonce="nonce-4891cc7b20c">
  …
</script>
```

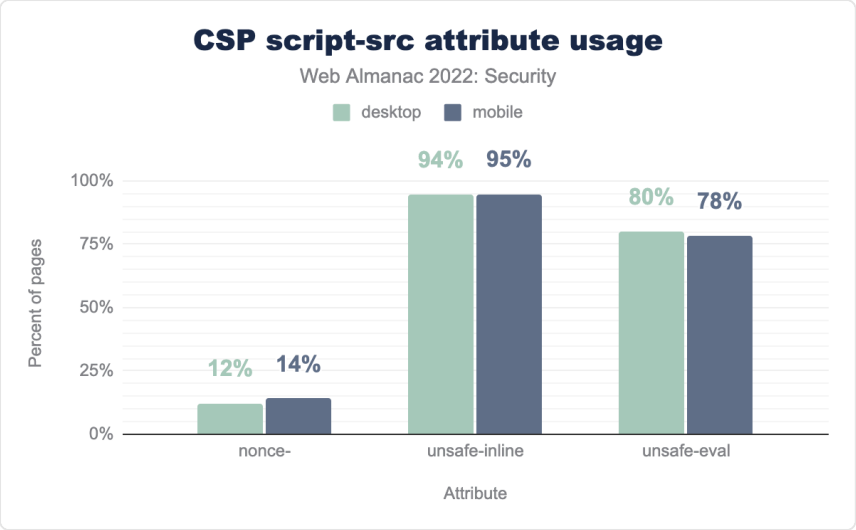



*図14.17. CSP `script-src` 属性の使用状況*

デスクトップのHTTPリクエストの統計によると、`script-src` 値の94%で `unsafe-inline`、80%で `unsafe-eval` が存在します。これは、ウェブサイトのアプリケーションコードをロックダウンし、インラインJavaScriptコードを避けることの実際の課題を示しています。さらに、上述のリクエストのうち14%だけが、安全でないインラインJavaScriptコードの使用を保護するのに役立つ `nonce-` ディレクティブを採用しています。

コンテンツセキュリティポリシーの定義の高い複雑さを示すかもしれないのが、CSPヘッダーの長さに関する統計です。

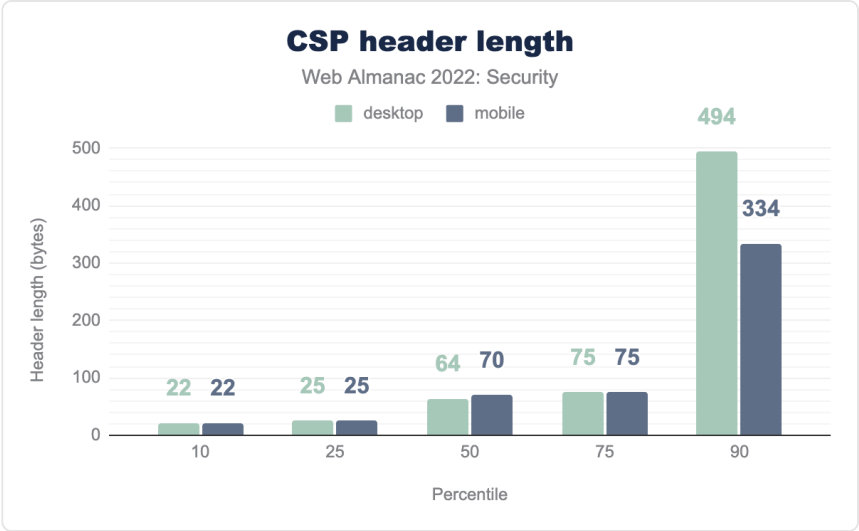



*図14.18. CSPヘッダーの長さ*

中央値で見ると、リクエストの50%がデスクトップで70バイトのサイズです。これは昨年のレポートからわずかに減少しており、昨年はデスクトップとモバイルのリクエストが75バイトでした。90番目のパーセンタイル以上のリクエストは、昨年のデスクトップリクエストの389バイトから、今年は494バイトに増加しています。これは、より複雑で完全なセキュリティポリシーへのわずかな進歩を示しています。

コンテンツセキュリティポリシーの完全な定義を観察すると、依然として単一のディレクティブがすべてのリクエストの大部分を占めています。すべてのデスクトップリクエストの19%が `upgrade-insecure-requests` のみに設定されています。8%が `frame-ancestors 'self'` に設定され、23%が `block-all-mixed-content; frame-ancestors 'none'; upgrade-insecure-requests;` の値に設定されています。これは、もっとも一般的なCSPディレクティブの上位3つを組み合わせたものです。

コンテンツセキュリティポリシーは、しばしばフォント、広告関連のスクリプト、一般的なコンテンツ配信ネットワークの使用をサポートするために、自身の起源以外のコンテンツを許可する必要があります。そのため、リクエスト全体でもっとも一般的な起源のトップ10は次のとおりです：

| 発行者 | デスクトップ | モバイル |
|---|---|---|
| https://www.google-analytics.com | 0.39% | 0.26% |
| https://www.googletagmanager.com | 0.37% | 0.25% |
| https://fonts.gstatic.com | 0.27% | 0.19% |
| https://fonts.googleapis.com | 0.27% | 0.18% |
| https://www.google.com | 0.24% | 0.17% |
| https://www.youtube.com | 0.21% | 0.15% |
| https://stats.g.doubleclick.net | 0.19% | 0.13% |
| https://connect.facebook.net | 0.18% | 0.13% |
| https://www.gstatic.com | 0.17% | 0.12% |
| https://cdnjs.cloudflare.com | 0.16% | 0.11% |

*図14.19. CSPポリシーでもっとも頻繁に許可されるホスト。*

上記のホストは昨年報告されたランクの位置づけとほぼ同じですが、使用率はわずかに上昇しています。

CSP（Content Security Policy）セキュリティ標準は、Webブラウザだけでなく、コンテンツ配信ネットワークやコンテンツ管理システムにも広くサポートされており、Webセキュリティの脆弱性を防御するためのウェブサイトやWebアプリケーションに強く推奨されるツールです。

## サブリソースインテグリティ

もう1つの防御の深さを提供するツールは、サブリソースインテグリティであり、コンテンツの改ざんに対するウェブセキュリティ防御層を提供します。コンテンツセキュリティポリシーがどの種類やソースのコンテンツが許可されるかを定義する一方で、サブリソースインテグリティメカニズムは、そのコンテンツが悪意のある目的で変更されていないことを保証します。

サブリソースインテグリティを使用する参考事例は、サードパーティのパッケージマネージャーからJavaScriptコンテンツをロードする場合です。これらの例には、unpkg.comやcdnjs.comなどがあり、どちらもJavaScriptライブラリのコンテンツソースを提供しています。

CDNプロバイダーによるホスティング問題、またはプロジェクトの貢献者やメンテナンス担当者による問題でサードパーティライブラリが危険にさらされる可能性がある場合、実質的に他人のコードを自分のウェブサイトにロードしていることになります。

CSPの `nonce-` の使用と同様に、サブリソースインテグリティ（SRIとも呼ばれます）は、ブラウザが提供されたコンテンツが暗号学的に署名されたハッシュと一致するかを検証し、コンテンツの改ざんを防ぐために送信中またはそのソースでの改ざんを防ぎます。


*図14.20. デスクトップサイトでのSRIの使用。*

デスクトップのウェブページ要素のうち、約5つに1つ（20%）がサブリソースインテグリティを採用しています。これらのうち、83%がデスクトップの `<script>` タイプ要素で使用され、17%がデスクトップリクエストの `<link>` タイプ要素で使用されています。

ページ単位でのカバレッジでは、SRIセキュリティ機能の採用率は依然としてかなり低いです。昨年、モバイルおよびデスクトップの中央値は3.3%でしたが、今年は2%減の3.23%になりました。

サブリソースインテグリティは、SHA256、SHA384、またはSHA512の暗号関数のいずれかで計算されたハッシュのbase64文字列として指定されます。使用事例の参照[545]として、開発者は以下のようにそれらを実装できます：

```
<script src="https://example.com/example-framework.js"
  integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K/
uxy9rx7HNQlGYl1kPzQho1wx4JwY8wC"
        crossorigin="anonymous"></script>
```

545. *https://developer.mozilla.org/ja/docs/Web/Security/Subresource_Integrity*

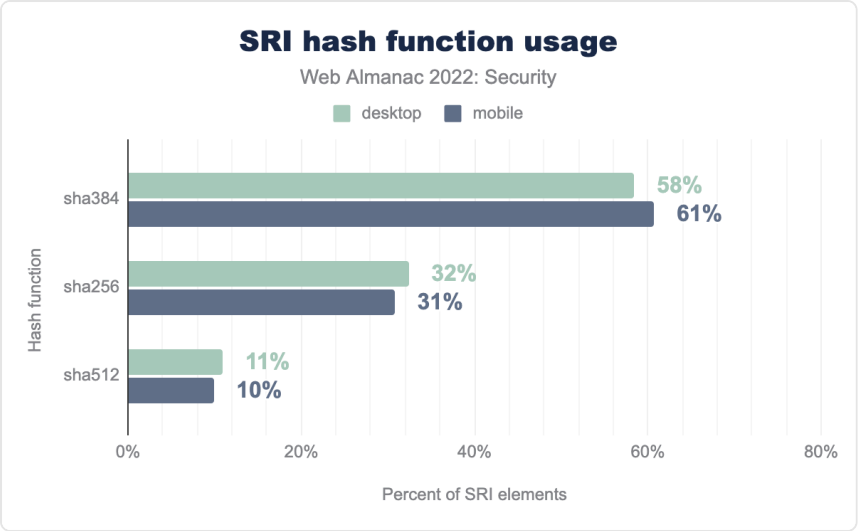



*図14.21. SRIハッシュ機能*

昨年の報告と一致して、SHA384は利用可能なすべてのハッシュ関数の中でもっとも多くのSRIハッシュタイプを示しています。

CDNはサブリソースインテグリティに慣れており、ページ上で提供されるコンテンツのURL参照にサブリソースインテグリティ値を含むことで、消費者に安全なデフォルトを提供しています。

| ホスト | デスクトップ | モバイル |
|---|---|---|
| www.gstatic.com | 39% | 40% |
| cdn.shopify.com | 22% | 23% |
| cdnjs.cloudflare.com | 8% | 7% |
| code.jquery.com | 7% | 7% |
| static.cloudflareinsights.com | 5% | 4% |
| cdn.jsdelivr.net | 3% | 3% |
| t1.daumcdn.net | 3% | 1% |
| stackpath.bootstrapcdn.com | 2.7% | 2.7% |
| maxcdn.bootstrapcdn.com | 2.2% | 2.3% |

*図14.22. SRI保護スクリプトが含まれるもっとも一般的なホスト*

上記のリストは、サブリソースインテグリティ値が観察されたトップ10のもっとも一般的なホストを示しています。昨年からの注目すべき変化は、Cloudflareのホストが4位から3位に上昇し、jsDelivrが7位から6位に上昇し、Bootstrapのホストのランキングを上回ったことです。

## パーミッションポリシー

時間とともにブラウザはますます強力になり、ウェブサイトに利用可能なさまざまなハードウェアや機能セットをアクセスして制御するためのより多くのネイティブAPIが追加されています。これらは、マイクをオンにしてデータを収集する悪意のあるスクリプトや、デバイスのジオロケーションを指紋認証して位置情報を収集するなど、特定の機能の悪用を通じてユーザーに潜在的なセキュリティリスクをもたらす可能性があります。

以前は `Feature-Policy` として知られ、現在は `Permissions-Policy` と名付けられているこのAPIは、ブラウザがアクセス可能な多くの機能の許可リストと拒否リストを制御するための実験的なブラウザAPIです。

`Permissions-Policy` の使用とHTTPS対応接続（97%）、`X-Content-Type-Options`（82%）、および `X-Frame-Options`（78%）との間に高い相関関係が見られます。すべての相関関係はデスクトップリクエストにわたっています。もう1つの高い相関関係は、特定の技術の交差点で観察され、Google My Businessモバイルページ（99%）と、次に近いAcquiaのCloud Platform（67%）で見られます。すべての相関関係はモバイルリクエストにわたっています。


*図14.23. 2021年からの `Permissions-Policy` の採用率の相対的な増加。*

2021年のデータ（1.3%）から2022年のデータ（2.4%）にかけて、モバイルリクエストにおける `Permissions-Policy` の採用率が85%の相対的な増加を見ています。デスクトップリクエストにおいても同様の傾向があります。廃止された `Feature-Policy` は昨年のデータと今年のデータの間で1パーセントポイントの僅かな増加を示しており、これはユーザーがウェブブラウザの仕様変更に追いついていることを示しています。

HTTPヘッダーとして使用されるだけでなく、この機能は以下のように `<iframe>` 要素内で使用できます：

```
    <iframe src="https://example.com" allow="geolocation 'src'
  https://example.com'; camera *"></iframe>
```

モバイルの1,150万フレームのうち18.9%に `allow` 属性が含まれており、許可または機能ポリシーを有効にしています。

以下は、フレームで検出されたトップ10の `allow` ディレクティブのリストです：

| ディレクティブ | デスクトップ | モバイル |
|---|---|---|
| encrypted-media | 75% | 75% |
| autoplay | 48% | 49% |
| picture-in-picture | 31% | 31% |
| accelerometer | 26% | 27% |
| gyroscope | 26% | 27% |
| clipboard-write | 21% | 21% |
| microphone | 9% | 9% |
| fullscreen | 8% | 7% |
| camera | 6% | 7% |
| geolocation | 5% | 6% |

*図14.24. iframeの`allow`ディレクティブの普及率。*

興味深いことに、上記のリストに入っていない11位、12位、13位のモバイル用`allow`ディレクティブは、それぞれ`vr`（6%）、`payment`（2%）、`web-share`（1%）です。これらは、仮想現実（メタバースとも言われます）、オンライン決済、フィンテック業界を取り巻く業界の成長傾向を示唆しているかもしれません。最後に、これは過去数年の在宅勤務の習慣の増加により、Webベースの共有のサポートが向上していることを示していると考えられます。

## Iframeサンドボックス

ウェブサイトで`iframe`要素を使用することは、リッチメディア、クロスアプリケーションコンポーネント、広告などのサードパーティコンテンツを簡単に埋め込むために、開発者が長年にわたって行ってきた実践です。一部の人々は、`iframe`要素が埋め込んでいるウェブサイトとソースウェブサイトの間にセキュリティ境界を形成すると考えるかもしれませんが、それは正確ではありません。

HTMLの`<iframe>`要素は、デフォルトでトップレベルページの機能（ポップアップやトップページのブラウザナビゲーションとの直接的なやりとり）にアクセスできます。たとえば、以下のコードは`iframe`要素のソースに埋め込まれ、アクティブなユーザーのジェスチャーを利用して、`iframe`をホスティングしているウェブサイトが

`https://example.com` に新しいURLへとナビゲートするようにします：

```
function clickToGo() {
  window.open('https://example.com')
}
```

この問題は一般に「クリックジャッキング攻撃」として知られており、`iframe` に埋め込まれる多くのセキュリティリスクの1つです（言葉遊びを意図）。

これらの懸念を軽減するために、HTML仕様（バージョン5）では `iframe` 要素に適用可能な `sandbox` 属性を導入しました。これは許可リストとして機能し、空のままであれば、`iframe` 要素内の任意の機能を基本的には有効にしません。これにより、ポップアップのようなページの対話性へのアクセス、JavaScriptコードの実行権限、クッキーへのアクセスが一切なくなります。

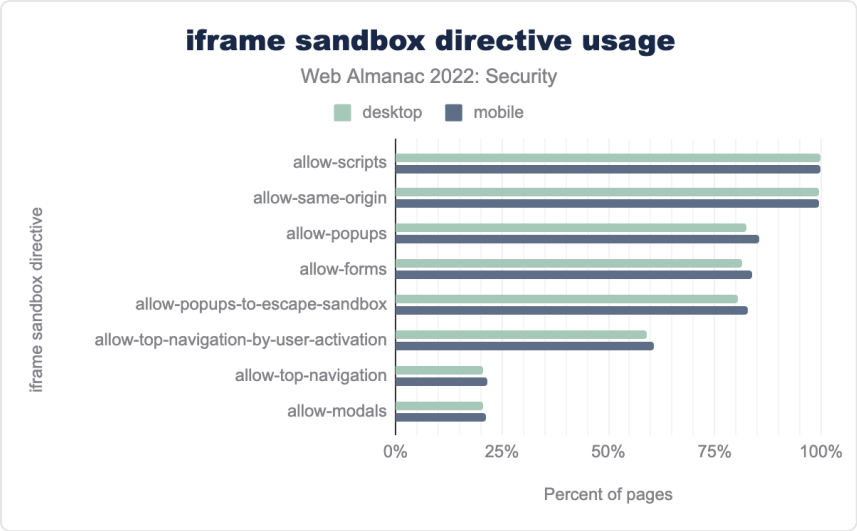



*図14.25. フレーム上のsandbox指令の普及率。*

2022年の上記のグラフは、`sandbox` 属性を持つウェブサイトの99%以上が `allow-scripts` と `allow-same-origin` の権限を有効にしていることを示しています。

iframeを埋め込むデスクトップウェブサイトの35.2%が `sandbox` 属性も含んでいます。

モバイルの `Content-Security-Policy` ヘッダーに `sandbox` 指令を含むものはわずか0.3%（デスクトップも同様に0.4%）であり、この属性がページ内でiframeコンテンツを埋め

込む際にケースバイケースで適用されることが多く、事前のコンテンツセキュリティポリシー定義を通じて計画することは少ないことを示しています。

## 攻撃の予防

ウェブサイトを悪用するさまざまな攻撃があり、ウェブサイトを完全に保護することはほとんど不可能です。しかし、ウェブ開発者はこれらの攻撃の多くを防ぐため、またはユーザーへの影響を限定するために多くの対策を講じることができます。

### セキュリティヘッダーの採用

セキュリティヘッダーは、トラフィックとデータフローの種類を制限することにより攻撃を防ぐもっとも一般的な方法の1つです。ただし、これらのセキュリティヘッダーのほとんどはウェブサイト開発者によって手動で設定されなければなりません。したがって、セキュリティヘッダーの存在は、そのウェブサイトの開発者が従うセキュリティ衛生の良い指標となることがよくあります。

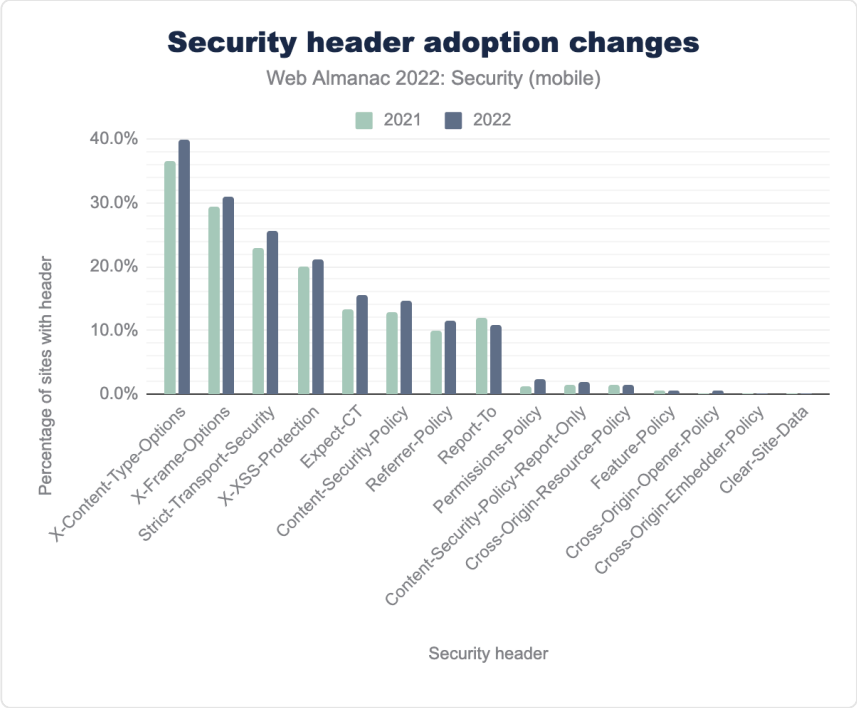



*図14.26. モバイルページでのサイトリクエストにおけるセキュリティヘッダーの採用。*

もっとも広く使用されているセキュリティメカニズムは依然としてX-Content-Type-Optionsヘッダーで、モバイルでクロールしたウェブサイトの40%でMIMEスニッフィング攻撃から保護するために使用されています。このヘッダーの次に多いのはX-Frame-Optionsヘッダーで、すべてのサイトの30%で有効になっています。昨年のデータからは大きな違いは見られず、すべてのセキュリティヘッダーの採用が徐々に増加していますが、使用率のランキングは同じです。

## CSPを使用した攻撃の防止

Content Security Policy（CSP）の主な用途は、安全にコンテンツをロードできる信頼できるソースを決定することです。これにより、CSPはクリックジャッキング、クロスサイトスクリプティング攻撃、ミックスコンテンツの含有など、さまざまな種類の攻撃を防ぐのに非常に役立つセキュリティヘッダーになります。


*図14.27. モバイルにおけるCSPを持つウェブサイトの割合で、frame-ancestorsディレクティブが含まれているもの。*

クリックジャッキング攻撃を防ぐ一般的な方法の1つは、ブラウザがウェブサイトをフレーム内でロードするのを防ぐことです。CSPヘッダーの`frame-ancestors`ディレクティブを使用して、他のドメインがページコンテンツをフレーム内に含むのを制限できます。モバイルでCSPを持つウェブサイトの53%に`frame-ancestors`ディレクティブが含まれていることがわかりました。これはクリックジャッキング攻撃を防ぐために非常に有効です。`frame-ancestors`ディレクティブの値を`none`または`self`に設定することがもっとも安全です。`none`はどのドメインもコンテンツをフレーム内に含むことを許可せず、`self`は元のドメインのみがコンテンツをフレーム内に含むことを許可します。モバイルでCSPヘッダーを持つウェブサイトの8%が`frame-ancestors 'self'`のみを持っており、これはCSPヘッダーの値としては3番目に一般的です。

| キーワード | デスクトップ | モバイル |
|---|---|---|
| `strict-dynamic` | 6% | 5% |
| `nonce-` | 12% | 14% |
| `unsafe-inline` | 94% | 95% |
| `unsafe-eval` | 80% | 78% |

*図14.28. CSPキーワードの普及率。`default-src`または`script-src`ディレクティブを定義するポリシーに基づく。*

ウェブサイトの`script-src`ディレクティブを制限的に設定することにより、XSS攻撃に対する防御メカニズムの1つとなります。これにより、JavaScriptは信頼できるソースからのみロードされ、攻撃者は悪意のあるコードを注入することができなくなります。`strict-dynamic`は、HTML内のルートスクリプトが他のスクリプトファイルを動的にロードまたは注入する場合に役立ちます。これはルートスクリプトにnonceやhashを使用し、`unsafe-inline`や個々のリンクなどの他の許可リストを無視します。これはIEを除くすべての現代のブラウザでサポートされています。また、昨年と比較して`unsafe-inline`と`unsafe-eval`の使用が約2%減少しています。これは正しい方向への一歩です。

## クロスオリジンポリシーを使用した攻撃の防止

クロスオリジンポリシーは、クロスサイトリークのようなマイクロアーキテクチャ攻撃を防ぐために使用される主要なメカニズムの1つです。XS-Leaksはクロスサイトリクエストフォージェリと似ていますが、ウェブサイト間の相互作用中に露呈されるユーザーに関する小さな情報を推測します。

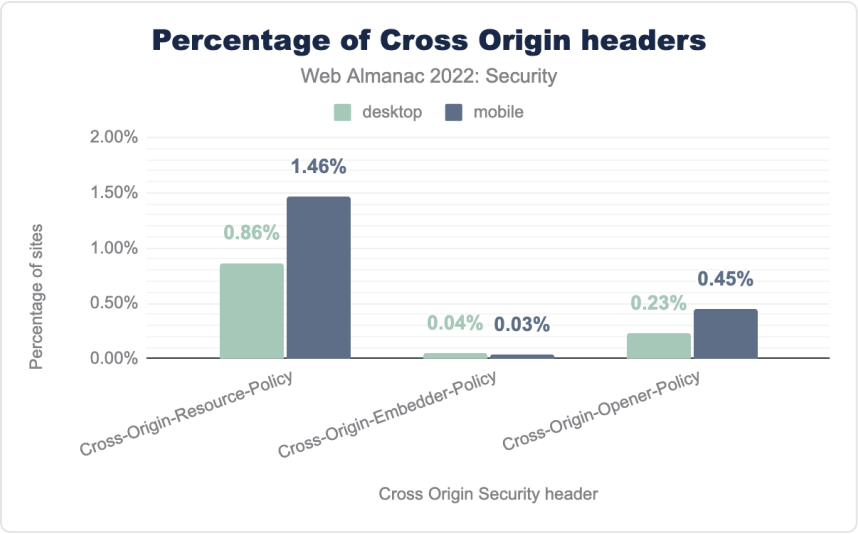



*図14.29. クロスオリジンヘッダーの割合。*

`Cross-Origin-Resource-Policy`はモバイルの114,111(1.46%)のウェブサイトに存在し、もっとも使用されているクロスオリジンポリシーです。これは、リソースがクロスオリジンから含まれるべきかどうかをブラウザに示すために使用されます。`Cross-Origin-Embedder-Policy`は、昨年の911ウェブサイトから比較して現在2,559ウェブサイトに存在しています。`Cross-Origin-Opener-Policy`の採用も同様に成長しており、昨年の15,727サイトから現在モバイルでは34,968サイトがこのヘッダーを持っています。したがって、すべてのクロスオリジンポリシーの採用は安定して成長しており、これはXS-Leak攻撃を防ぐのに非常に役立つため、素晴らしいことです。

## Clear-Site-Dataを使用した攻撃の防止

Clear-Site-Data[546]は、ウェブ開発者が自分のウェブサイトに関連するユーザーデータのクリアをよりコントロールできるようにします。たとえば、ウェブ開発者は、ユーザーが自分のウェブサイトからサインアウトするときに、そのユーザーに関連するすべてのクッキー、キャッシュ、ストレージ情報を削除できるように決定できます。これにより、必要ないときにブラウザに保存されるデータ量を制限し、攻撃の影響を限定するのに役立ちます。これは比較的新しいヘッダーで、HTTPSで提供されるサイトにのみ制限されており、機能の一部のみがブラウザによってサポートされています[547]。2021年にはモバイルで`Clear-Site-Data`ヘッダーを持つサイトは75件でしたが、今年は428件に増加しました。ウェブアルマナックのデータでは追跡されていないログアウトページでのみこのヘッダーを使用するウェブサイト

546. https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Clear-Site-Data
547. https://developer.mozilla.org/ja/docs/Web/HTTP/Headers/Clear-Site-Data#browser_compatibility

が多いことに注意する価値があります。

| CSDヘッダー | デスクトップ | モバイル |
|---|---|---|
| cache | 65% | 63% |
| * | 9% | 8% |
| “cache” | 7% | 7% |
| cookies | 3% | 6% |
| “storage” | 2% | 1% |
| cache,cookies,storage | 1% | 1% |
| “cache”, “storage” | 1% | 1% |
| "*" | 1% | 2% |
| “cache”, “cookies” | 1% | 0% |
| “cookies” | 1% | 1% |

*図14.30. `Clear-Site-Data` ヘッダーの普及率。*

`cache` はClear-Site-Dataでもっとも普及しているディレクティブで、モバイルウェブサイトの63%で使用されていますが、これは多くの開発者がユーザーのプライバシーやセキュリティよりも、新しい静的ファイルをロードするためにこのセキュリティヘッダーを使用していることを意味するかもしれません。しかし、ディレクティブは引用文字列文法[548]に従うべきであり、そのためこのディレクティブは無効です。9%のモバイルウェブサイトがこのヘッダーを使用し、”*”を使っているのを見るのは素晴らしいことです。これは、ブラウザに保存されているすべてのユーザーデータをクリアするよう指示しています。3番目に使用されているディレクティブは再び `"cache"` であり、今回は適切に使用されています。

## `<meta>` を使用した攻撃の防止

ウェブサイトのHTMLコード内で `<meta>` タグを使用して `Content-Security-Policy` や `Referrer-Policy` を設定できます。たとえば、 `Content-Security-Policy` は以下のコードを使用して設定できます: `<meta http-equiv="Content-Security-Policy" content="default-src 'self'">`。モバイルでこの方法でCSPとReferrer-Policyを有効にしているウェブサイトはそれぞれ0.47%と2.60%であることが分かりました。

548. https://datatracker.ietf.org/doc/html/rfc7230#section-3.2.6

| メタタグの値 | デスクトップ | モバイル |
|---|---|---|
| include Referrer-Policy | 3.11% | 2.60% |
| include CSP | 0.55% | 0.47% |
| include note-allowed headers | 0.08% | 0.06% |

*図14.31. `<meta>` タグで使用されるセキュリティヘッダー。*

`<meta>` タグを使用して攻撃を防ぐ際の問題点は、他のセキュリティヘッダーを設定した場合、ブラウザはそのセキュリティヘッダーを無視することです。たとえば、2,815のサイトが `<meta>` タグに `X-Frame-Options` を持っていました。開発者は `<meta>` タグを追加したことでウェブサイトが特定の攻撃に対して安全だと期待しているかもしれませんが、実際にはそのセキュリティヘッダーが追加されていないのです。ただし、この数は昨年の3,410サイトから減少しているので、ウェブサイトが `<meta>` タグの誤用を修正している可能性があります。

## Web Cryptography API

Web Cryptography API[549]は、ランダム数の生成、ハッシュ化、署名、暗号化、復号など、ウェブサイト上で基本的な暗号操作を行うためのJavaScript APIです。

549. https://www.w3.org/TR/WebCryptoAPI/

| 機能 | デスクトップ | モバイル |
|---|---|---|
| CryptoGetRandomValues | 69.6% | 65.5% |
| SubtleCryptoDigest | 0.6% | 0.6% |
| CryptoAlgorithmSha256 | 0.5% | 0.3% |
| SubtleCryptoImportKey | 0.2% | 0.1% |
| SubtleCryptoGenerateKey | 0.2% | 0.2% |
| SubtleCryptoEncrypt | 0.2% | 0.1% |
| SubtleCryptoExportKey | 0.2% | 0.1% |
| CryptoAlgorithmAesGcm | 0.1% | 0.1% |
| SubtleCryptoSign | 0.2% | 0.2% |
| CryptoAlgorithmAesCtr | 0.1% | < 0.1% |

*図14.32. もっとも使用されている暗号APIトップリスト。*

昨年からのデータにはほとんど変化がありません。`CryptoGetRandomValues`は引き続きもっとも採用されている機能であり（昨年から約1%減少していますが）、強力な擬似乱数を生成するために使用されているためです。その高い使用率は、Google Analyticsなどの一般的なサービスで使用されていることに起因しています。

## ボット保護サービス

現代のインターネットはボットで溢れており、悪質なボット攻撃が常に増加しています。Impervaによる2022年の悪質ボットレポート[550]によると、インターネットトラフィックの27.7%が悪質なボットによるものでした。悪質なボットとは、データをスクレイピングして悪用しようとするものです。レポートによると、2021年末にはlog4jの脆弱性を利用したボットによる攻撃が急増したとされています。

550. https://www.imperva.com/resources/reports/2022-Imperva-Bad-Bot-Report.pdf

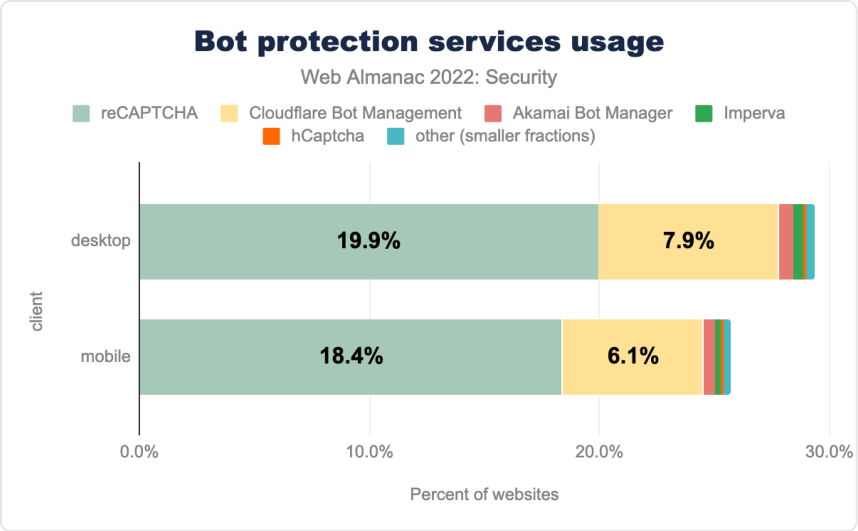



*図14.33. ボット保護サービスのプロバイダー別使用状況。*

分析によると、デスクトップのウェブサイトの29%、モバイルのウェブサイトの26%が悪質なボットと戦うためのメカニズムを使用しており、昨年（それぞれ11%と10%）から大幅に増加しています。この増加はCloudflareボット管理[551]などのキャプチャフリーのソリューションが原因かもしれません。昨年のデータクロールではこの点が追跡されていませんでしたが、今年はそれが識別され、モバイルのウェブサイトの6%で使用されていることがわかりました。デスクトップとモバイルの両方でreCaptchaの使用も昨年から約9%増加しています。

## セキュリティメカニズムの導入要因

ウェブサイトがより多くのセキュリティ対策を採用するための主な要因はいくつかあります。もっとも重要な3つは以下の通りです：

- 社会的：一部の国ではセキュリティ指向の教育が進んでいる、またはデータ侵害の場合により厳しい処罰を規定する法律がある
- 技術的：特定の技術スタックでセキュリティ機能を採用することが容易である、または特定のベンダーがデフォルトでセキュリティ機能を有効にしている
- 人気：広く人気のあるウェブサイトは、あまり知られていないウェブサイトより

551. https://www.cloudflare.com/en-gb/products/bot-management/

もターゲットとされる攻撃を多く受ける可能性がある

## ウェブサイトの所在地

ウェブサイトの開発者が拠点を置いている場所やウェブサイトがホストされている場所は、セキュリティ機能の採用に影響を与えることがよくあります。これは開発者の意識が異なるためかもしれませんが、国の法律が特定のセキュリティ対策の採用を義務付けているためかもしれません。

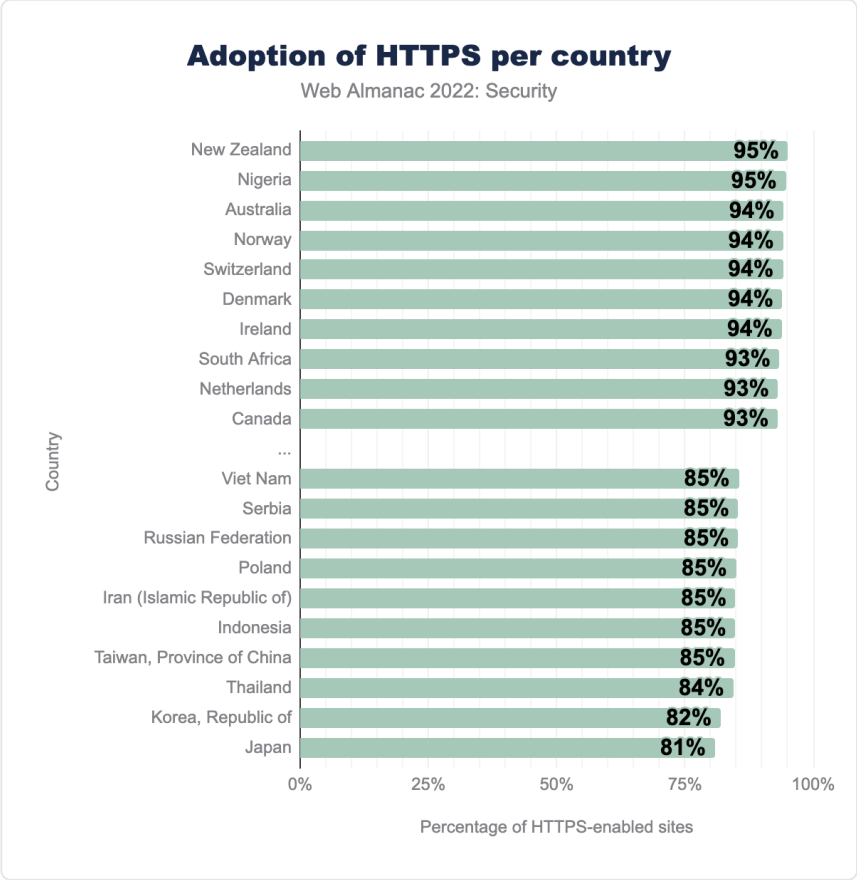



*図14.34. 国別HTTPSの採用状況。*

ニュージーランド、ノルウェー、デンマークなどの新しい国々がHTTPSの採用で急速にトップに上り詰めています。新しい国々も広範なセキュリティ対策を採用しているのを見るのは

良い兆候です。また、HTTPSの最低採用率と最高採用率の差が縮小していることから、ほぼすべての国がデフォルトでウェブサイトにHTTPSを有していることを目指していることがわかります。

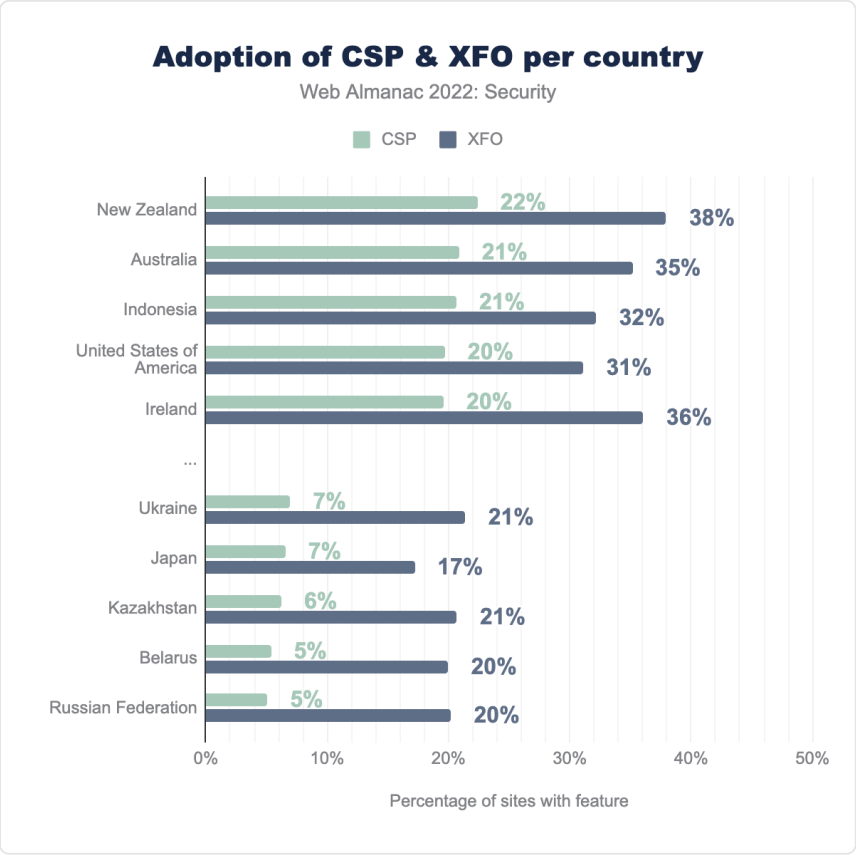



*図14.35. 国別CSPとXFOの採用状況。*

CSPと`X-Frame-Options`（XFO）の採用は昨年と非常に似ています。驚くべきことに、インドネシアのウェブサイトはCSPの採用が非常に進んでいますが、HTTPSの採用は依然として低いままです。CSPの採用は国によって非常にバラバラですが、XFOの採用差は徐々に縮小しています。CSPは多岐にわたる攻撃から保護する非常に重要なセキュリティ機能であるため、より多くの国がCSPの採用を増やす必要があります。

## テクノロジースタック

ウェブサイトの構築に使用されるテクノロジースタックも、特定のセキュリティ機構の採用

に大きな影響を与える要因の1つです。場合によっては、コンテンツ管理システムなどで、セキュリティ機能がデフォルトで有効になっていることもあります。

| テクノロジー | セキュリティ機能 |
| --- | --- |
| Blogger | `Content-Security-Policy` (99%), `X-Content-Type-Options` (99%), `X-Frame-Options` (99%), `X-XSS-Protection` (99%) |
| Wix | `Strict-Transport-Security` (99%), `X-Content-Type-Options` (99%) |
| Drupal | `X-Content-Type-Options` (77%), `X-Frame-Options` (77%) |
| Squarespace | `Strict-Transport-Security` (91%), `X-Content-Type-Options` (98%) |
| Google Sites | `Content-Security-Policy` (96%), `Cross-Origin-Opener-Policy` (96%), `Referrer-Policy` (96%), `X-Content-Type-Options` (97%), `X-Frame-Options` (97%), `X-XSS-Protection` (97%) |
| Plone | `X-Frame-Options` (60%) |
| Wagtail | `X-Content-Type-Options` (51%), `X-Frame-Options` (72%) |
| Medium | `Content-Security-Policy` (75%), `Expect-CT` (83%), `Strict-Transport-Security` (84%), `X-Content-Type-Options` (83%) |

*図14.36. さまざまなテクノロジーによるセキュリティ機能の採用。*

上記は一般的なCMSやブログサイトの一部です。Wix、Squarespace、Mediumなど、カスタマイズをあまり提供せず、コンテンツ編集に重点を置いているサイトでは、`Strict-Transport-Security`などの基本的なセキュリティ機能がデフォルトで有効になっている傾向があります。Wagtail、Plone、Drupalなどのコンテンツ管理システムは、開発者がウェブサイトを設定するためによく使用されるため、セキュリティ機能の追加は開発者により大きく依存しています。また、Google Sitesを使用するウェブサイトでは、多くのセキュリテ

ィ機能がデフォルトで有効になっていることがわかります。

### ウェブサイトの人気度

多くの訪問者を持つウェブサイトは、潜在的に機密データを持つユーザーが多いため、ターゲットとされる攻撃を受けやすい可能性があります。そのため、広く訪問されるウェブサイトでは、ユーザーを保護するためにセキュリティへの投資がより多くなると考えられます。

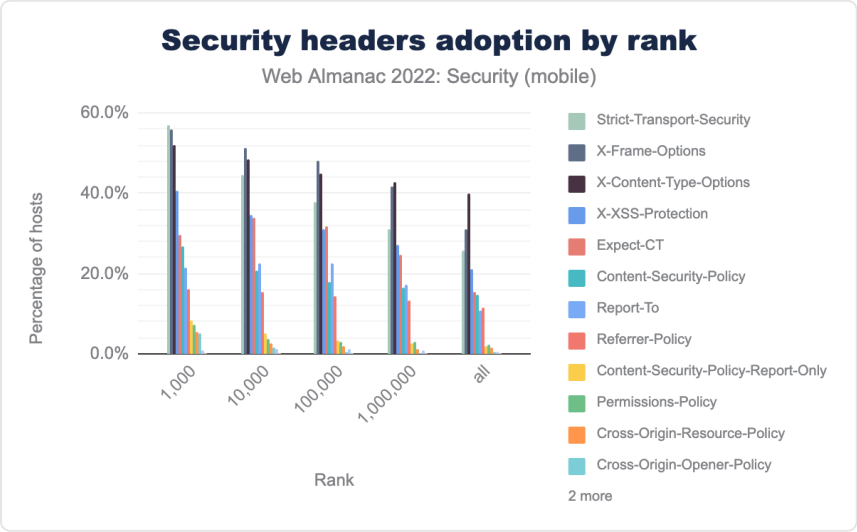



*図14.37. ランク別に設定されているセキュリティヘッダーの普及率。*

調査によると、`Strict-Transport-Security`、`X-Frame-Options`、および`X-Content-Type-Options`は、人気のあるウェブサイトほど採用率が高いことがわかりました。モバイルのトップ1000ウェブサイトの56.8%がStrict-Transport-Securityを採用しており、これらのウェブサイトはコンテンツとデータをHTTPS経由でのみ提供することを重視しています。人気のないウェブサイトもHTTPSを有効にしているかもしれませんが、ウェブサイトが常にHTTPS経由で提供されるように`Strict-Transport-Security`ヘッダーを追加することはしばしば見られません。今年の数字は昨年の結果とほぼ一致しています。

## ウェブ上の悪用

今年も暗号通貨の人気が高まり、さまざまな用途で利用できる新しいタイプの暗号通貨が増加しました。その継続的な成長と既存の経済的インセンティブを背景に、サイバー犯罪者は

cryptojacking[552]を利用して利益を得ています。しかし、昨年から全体的に暗号通貨マイナーの使用は減少しています。攻撃者がデスクトップとモバイルのシステムに暗号通貨マイナーを注入することを可能にする特定の脆弱性イベントが発生すると、その使用が急増する傾向があります：

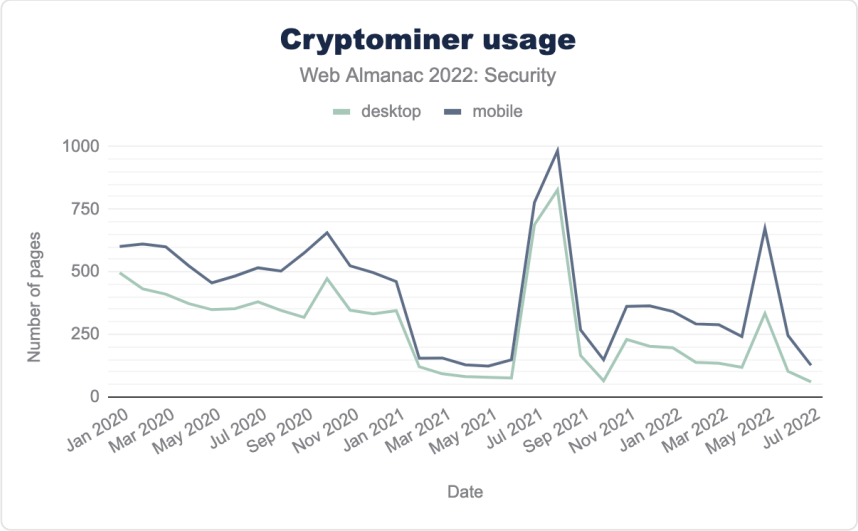



*図14.38. Cryptominerの使用状況。*

例として、2021年の7月と8月には、いくつかの暗号通貨ジャッキングキャンペーンと脆弱性が報告され、その時期にウェブサイトで見つかった暗号通貨マイナーの急増が発生しました。さらに最近の2022年4月には、ハッカーがSpringShell脆弱性を利用して暗号通貨マイナーを設定し実行しようと試みました[553]。

デスクトップとモバイルのウェブサイトで使用されている暗号通貨マイナーの具体的な情報を見ると、マイナー間のシェアが昨年から広がっています。たとえば、Coinimpのシェアは昨年から約24%縮小し、Minero.ccは約11%増加しました。

552. https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AA%E3%83%97%E3%83%88%E3%82%B8%E3%83%A3%E3%83%83%E3%82%AD%E3%83%B3%E3%82%B0
553. https://arstechnica.com/information-technology/2022/04/hackers-hammer-springshell-vulnerability-in-attempt-to-install-cryptominers/

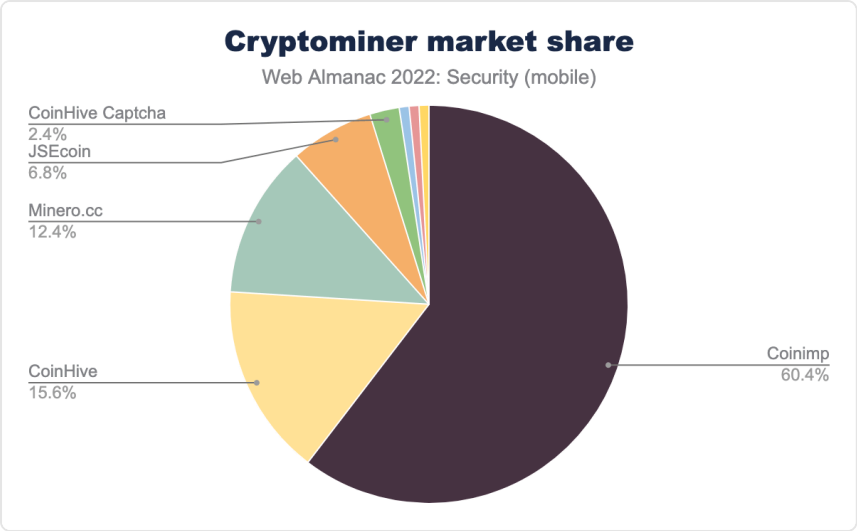



*図14.39. モバイルにおけるCryptominerの市場シェア。*

これらの結果から、cryptojackingは毎年深刻な攻撃手法であり、新たに出現した脆弱性を利用して使用が急増することが示されています。したがって、この分野のリスクを軽減するために適切な注意が依然として必要です。

これらのウェブサイトすべてが感染しているわけではありません。ウェブサイト運営者は広告を表示する代わりにこの技術を使用してウェブサイトを資金調達することがあります。しかし、この技術の使用は技術的、法的、倫理的にも大きく議論されています。

また、私たちの結果がcryptojackingに感染したウェブサイトの実際の状態を示しているとは限らないことにも注意してください。私たちのクローラーは月に一度実行されるため、暗号通貨マイナーを実行しているすべてのウェブサイトを発見することはできません。たとえば、ウェブサイトがX日間だけ感染しており、私たちのクローラーが実行された日には感染していない場合がこれに該当します。

## よく知られたURI

よく知られたURI[554]は、ウェブサイト全体に関連するデータやサービスを特定の場所に指定するために使用されます。よく知られたURIは、パスコンポーネントが`/.well-known/`という文字で始まるURI[555]です。

554. https://datatracker.ietf.org/doc/html/rfc8615
555. https://datatracker.ietf.org/doc/html/rfc3986

## security.txt

security.txtは、ウェブサイトが脆弱性報告の標準を提供するためのファイルフォーマットです。ウェブサイト提供者は、このファイルに連絡先の詳細、PGPキー、ポリシー、その他の情報を提供できます。ホワイトハットハッカーやペネトレーションテスターは、これらの情報を使用してウェブサイトのセキュリティ分析を行い、脆弱性を報告できます。

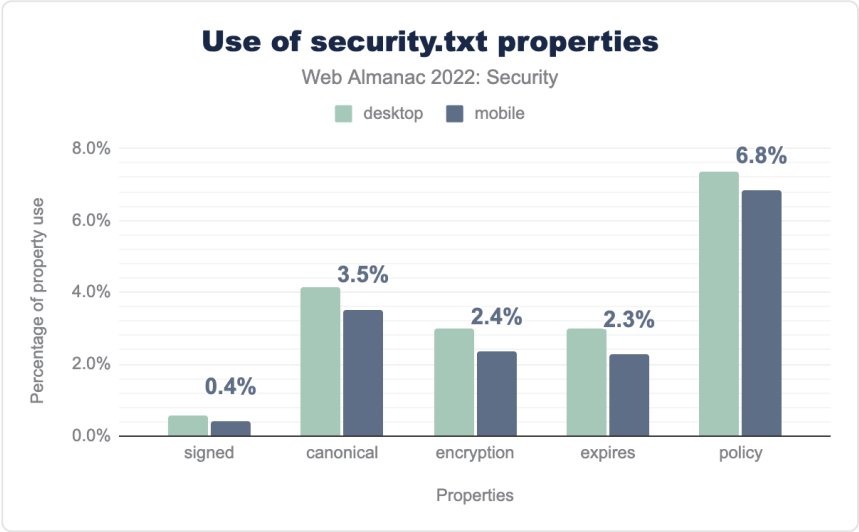



*図14.40. security.txtプロパティの使用。*

今年はsecurity.txt URIのexpiresプロパティを持つ割合が0.7%から2.3%に増加しました。expiresプロパティは標準に基づいて必要なプロパティであるため、より多くのウェブサイトが標準にしたがっているのを見るのは良いことです。policyはsecurity.txt URIでもっとも人気のあるプロパティのままです。policyは、セキュリティ研究者が脆弱性を報告するために従うべき手順を説明しているため、security.txt URIにとって非常に重要です。

## change-password

change-passwordは、現在編集ドラフト状態にあるW3CのWebアプリケーションセキュリティ作業グループの仕様です。この特定のよく知られたURIは、ユーザーやソフトウェアがパスワード変更用のリンクを簡単に特定できる方法として提案されました。

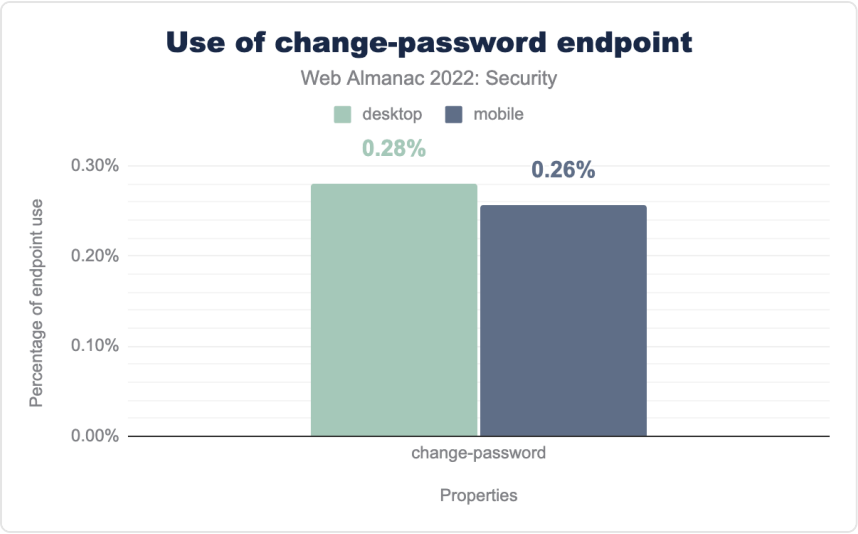



*図14.41. change-passwordエンドポイントの使用。*

このよく知られたURIの採用率はまだ非常に低いです。仕様はまだ作業中であるため、多くのウェブサイトがこれを採用していないのは理解できます。また、サインインシステムを持たないウェブサイトには、パスワード変更フォームがない場合もあります。

## ステータスコードの信頼性検出

この特定のよく知られたURIは、ウェブサイトのHTTPレスポンスステータスコードの信頼性を決定します。このURIもまだ編集ドラフト[556]の状態にあり、将来変更される可能性があります。このよく知られたURIの背景にある考え方は、ウェブサイトに存在すべきではないというものです。したがって、このよく知られたURIはok-status[557]で応答すべきではありません。リダイレクトして"ok-status"を返す場合、そのウェブサイトのステータスコードは信頼できないという意味です。

556. *https://w3c.github.io/webappsec-change-password-url/response-code-reliability.html*
557. *https://fetch.spec.whatwg.org/#ok-status*

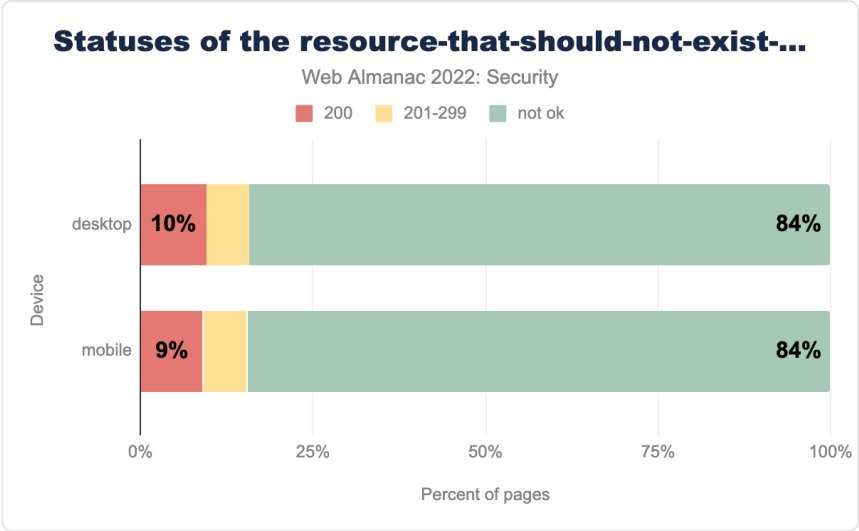



*図14.42. 存在しないはずのリソースのステータスコードの検出。*

このよく知られたURIに対して、モバイルとデスクトップのウェブサイトの84%がnot-okステータスで応答しています。この仕様の良い点は、ウェブサイトが正しく設定されていれば、ウェブサイト開発者が特定の変更を行う必要がなく、自動的に機能することです。

## 結論

今年の分析によると、過去数年にわたって見られたように、ウェブサイトはセキュリティ機能の向上を続けています。また、ウェブセキュリティの採用が遅れていた多くの国々がその使用を増やしていることも、興味深い観察点です。これは、ウェブセキュリティに対する一般的な意識が高まっていることを意味しているかもしれません。

ウェブ開発者が新しい標準を徐々に採用し、古いものを置き換えていることもわかりました。これは間違いなく正しい方向への一歩です。インターネット上でのセキュリティとプライバシーの重要性は日々高まっています。ウェブは多くの人々の生活に不可欠な部分となっており、そのため、ウェブ開発者はウェブセキュリティ機能の使用を続けて増やすべきです。

より厳格なコンテンツセキュリティポリシーを設定する上で、まだ多くの進歩が必要です。クロスサイトスクリプティングは依然としてOWASP Top 10[558]に含まれています。このような攻撃を防ぐために、より厳格な`script-src`ディレクティブの広範な採用が必要です。ま

558. https://owasp.org/Top10/

た、より多くの開発者がWeb Cryptography APIを活用することを検討することもできます。security.txtのようなよく知られたURIを採用するためにも同様の努力が必要です。これは、セキュリティ専門家がウェブサイトの脆弱性を報告する方法を提供するだけでなく、開発者がウェブサイトのセキュリティを重視しており、改善に積極的であることを示しています。

過去数年にわたるウェブセキュリティの使用における継続的な進歩を観察することは励みになりますが、ウェブが成長を続け、セキュリティがより重要になるにつれて、ウェブコミュニティはさらに多くのセキュリティ機能の研究と採用を続ける必要があります。

## 著者


### Saptak Sengupta

@Saptak013 SaptakS https://saptaks.website

Saptak Sは、ウェブ開発における使いやすさ、セキュリティ、プライバシー、アクセシビリティを中心に据えた人権重視のウェブデベロッパーです。彼は、The A11Y Project[559]、OnionShare[560]、Wagtail[561]など、さまざまなオープンソースプロジェクトの貢献者兼メンテナです。彼のブログはsaptaks.blog[562]で読むことができます。

### Liran Tal

@liran_tal lirantal https://twitter.com/liran_tal

オープンソースおよびJavaScriptセキュリティの取り組みで知られるLiran Tal[563]は、受賞歴のあるソフトウェア開発者であり、セキュリティ研究者、そしてJavaScriptコミュニティでのオープンソースのチャンピオンです。彼は国際的に認められたGitHub Star[564]であり、オープンソースへの支援で知られ、Node.jsのセキュリティに対する彼の功績でOpenJS FoundationからPathfinder for Security[565]を受賞しています。彼の開発者セキュリティ教育への貢献には、OWASPプロジェクトのリーダーシップ、サプライチェーンセキュリティツールの構築、CNCFおよびOpenSSFイニシアティブへの参加、そして「O'Reilly's Serverless Security」のような書籍の執筆が含まれます。彼はSnyk.ioでデベロッパーアドボカシーチームを率いており、より優れたアプリケーションセキュリティスキルで開発者を支援することを使命としています。


559. https://www.a11yproject.com
560. https://onionshare.org/
561. https://wagtail.org/
562. https://saptaks.blog
563. https://www.lirantal.com/
564. https://stars.github.com/profiles/lirantal/
565. https://openjsf.org/announcement/2022/06/07/first-ever-javascriptlandia-awards-celebrate-community-leaders/

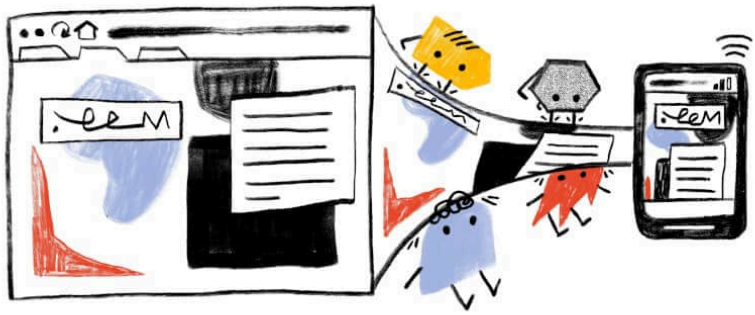
## Brian Clark

@_clarkio clarkio https://www.clarkio.com

Brianはアプリケーションセキュリティに関する深い経験を持つウェブデベロッパーです。彼はSnyk.ioのデベロッパーアドボケートとして、開発者が安全なウェブアプリケーションを構築できるよう支援しています。彼はフルスタックプロジェクト全般にわたる経験がありますが、とくにバックエンドサービス、API、および開発者ツールに焦点を当てています。Brianは、自身のキャリアを通じて得た成功と失敗から学んだことを開発者に教えることに情熱を注いでいます。彼の週間ライブストリーム[566]やPluralsightのコース[567]のいずれかで、その活動を見ることができます。

566. *https://clarkio.live*
567. *https://www.pluralsight.com/authors/brian-clark*

# 部 II 章 15
# モバイルウェブ



***Cindy Krum によって書かれた。***
*Dave Smart、 David Fox と Hemanth HM によってレビュ。*
*Sia Karamalegos と Rick Viscomi による分析。*
*Rick Viscomi 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

Webコンテンツへのモバイルアクセスは、インターネットアクセス全体の重要な側面です。実際、多くの状況や地域において、これはインターネットへのアクセスのデフォルト手段[568]です。また、デスクトップ、ネイティブアプリ、ウェブアプリを使用してデバイス間の行動を促進し、消費者が好みのアクセス方法を使用できるようにするプラットフォームで、コミュニケーションの中核[569]としてもしばしば機能します。これにより、オンラインでの情報とコミュニケーションがさらに簡素化され、民主化されます。

この章では、モバイルデバイスからアクセスされたときの2022年のウェブの状態を概説します。場合によっては、多くの人がより慣れ親しんでいるデスクトップデータとモバイルデータを比較します。この比較は重要です。多くの人がデスクトップデータに焦点を当てるかもしれませんが、現在、世界中でデスクトップよりもモバイルウェブトラフィックの方が多

568. https://www.gsma.com/mobileeconomy/wp-content/uploads/2022/02/280222-The-Mobile-Economy-2022.pdf
569. https://www.investopedia.com/is-having-a-smartphone-a-requirement-in-2021-5190186

く、これは約2016年か2017年からのことであり、ソースによって異なります。

## 世界的な接続性

いつものように、今年はこれまでに経験したことのないほど接続された世界に生きています。モバイル技術とモバイルウェブの進化は、COVID-19パンデミックによって2年以上もデジタル中心の商取引が増加したことだけでなく、5G通信ネットワーク、クラウドおよびハイブリッドクラウドコンピューティング環境の成長と進化、そしてデジタルおよび音声アシスタント、キャスティング技術、IoTの普及によっても燃え上がっています。

新しい世代がソーシャルメディアに関わり、以前よりも早くモバイル技術にアクセスしており、これまで以上に社会的に受け入れられやすくなっています。したがって接続の成長は目に見える終わりなく進行しており、今日の子供たち、ティーンエイジャー、若者たちは、デジタルに接続された世界に生まれたことから「デジタルネイティブ」と呼ばれており、彼らは間違いなくモバイル技術、モバイルウェブ、接続の進化を新たな高みに押し上げるでしょう。この進歩は、未来の技術の基礎となる最新技術に見えるでしょう。

### モバイルとデスクトップからのトラフィック

方法論に従い、このレポートの主要なデータソースはHTTP ArchiveとChrome UXレポート（CrUX）です。データソースからタブレットデータが別の測定値として含まれている場合、それは省略されました。これは、タブレットが主要なモバイルまたはデスクトップの分類に適切に収まらず、モバイルとデスクトップの情報を解釈または対比する際に混乱や複雑さを増加させる可能性があるためです。詳しくは、CrUXドキュメント[570]を参照してください。

570. *https://developer.chrome.com/docs/cruxmethodology?hl=ja#user-eligibility*

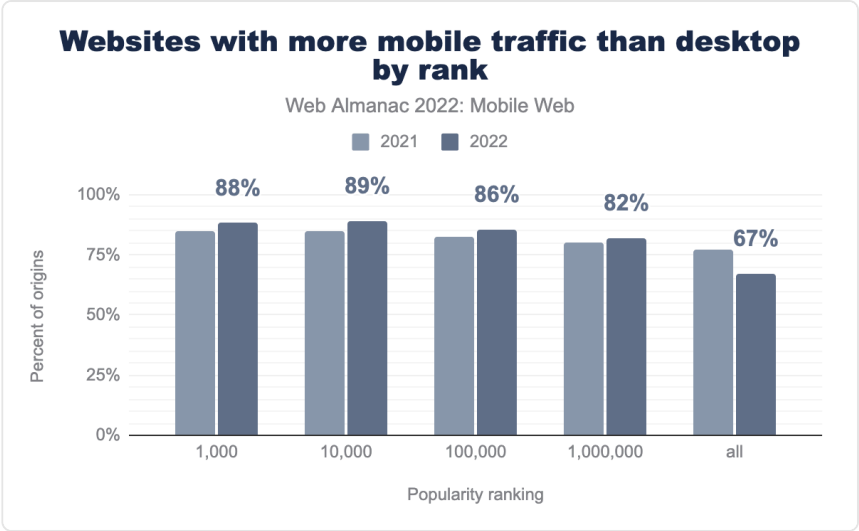



*図15.1. モバイルトラフィックがデスクトップを上回るウェブサイトの割合の年次比較、人気度による区分。*

人気ランクが高いサイトでは、モバイルトラフィックがデスクトップトラフィックを上回るサイトの割合が昨年[571]に比べて増加しています。2022年には、もっとも人気のある1,000のサイトのうち88%がデスクトップよりもモバイルデバイスからのトラフィックが多く、2021年の約85%から増加しています。そして、もっとも人気のある10,000のサイトの中で89%がモバイルデバイスからのトラフィックが多く、これは2021年の約86%から増加しています。これは、これらのトップ人気ランキンググループの両方で、デスクトップよりもモバイルトラフィックが多いサイトが約3%増加したことを意味します。

Oberloによると、2022年のウェブトラフィックの約58%[572]がモバイルデバイスからのものです。これらの統計の一貫した成長と普及は、モバイルウェブアクセスとインタラクションの全体的な評価での明らかな重要性の増大を明確に示しています。

## モバイルウェブからのコミュニケーション

モバイルウェブの価値は、人々を他の人々や情報にシームレスにつなげる能力に大きく依存しています。これは常に持ち運べる形で、馴染み深く、目立たない方法で行われます。電話は常に主にコミュニケーションのために使用されてきましたが、モバイルウェブを携帯電話に追加することで、電話の見られ方、使用法、評価が根本的に変わりました。このレポートのこのセクションでは、モバイルウェブからのコミュニケーションのもっとも重要な側面を

571. *https://almanac.httparchive.org/ja/2021/mobile-web#トラフィック利用の人気順*
572. *https://www.oberlo.com/statistics/mobile-internet-traffic*

どのように認識できるか、そしてコミュニケーションの可能性がモバイルウェブサイトにどのように反映されているかを検討します。

## 代替プロトコルリンク

モバイルデバイスが人々の日常的なコミュニケーションにとって非常に重要であるため、モバイルウェブに存在するもっとも一般的なリンク形式の評価が興味深いものになります。これは「代替プロトコルリンク」と呼ばれるものです。これらの代替プロトコルは、ウェブサイトの一ページから別のページへのリンクではなく、ウェブブラウジング以外の活動へのリンクを行います。

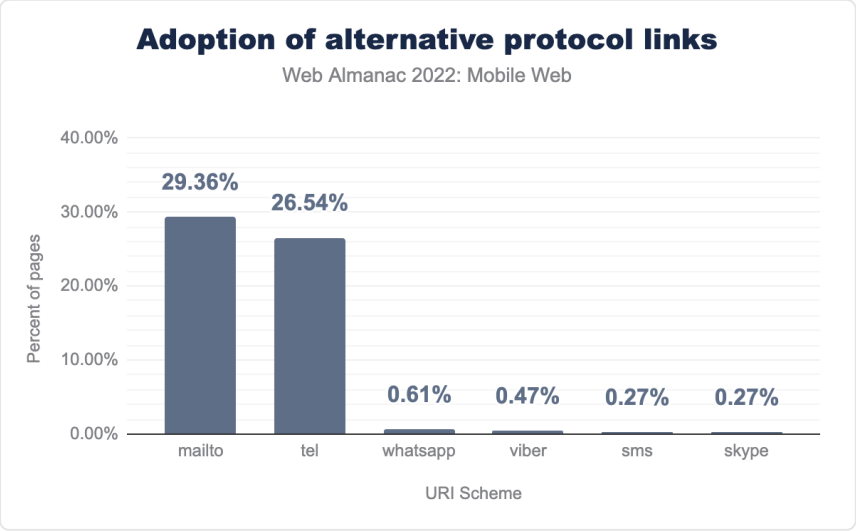



*図15.2. モバイルウェブページで使用される代替プロトコルの採用状況。*

上記の図は、2022年にウェブでどのように異なる代替プロトコルリンクが使用されているかを概説しています。もっとも人気のあるオプションは、メールのための `mailto`、電話をかけるための `tel`、テキストメッセージを送るための `sms`、WhatsApp、Skype、Viberなどの異なるメッセージングサービスのアプリケーション固有のプロトコルです。

`mailto` プロトコルは、モバイルページの29%で見つかります。それに続いて `tel` プロトコルがあり、27%のページで見つかります。これらのプロトコルがモバイルページでより多く見られることは、電話をかけるなど、デスクトップで実行するのがより複雑な、より多様なモバイル体験を奨励する機能でウェブサイトが変更されている可能性を示しているかもしれません。

*図15.3. `sms` リンクプロトコルを使用するモバイルページの割合。*

モバイルウェブサイトでの `sms` プロトコルの採用は非常に低く、0.27%にとどまっています。これは興味深いことです。なぜなら、人々に事前にフォーマットされたSMSメッセージへのリンクを提供することは、ユーザーを長期にわたってブランドとより関与させることができるSMSロイヤルティプログラムへの登録を奨励する効果的な方法になり得るからです。しかし、サイトを構築する際にデザイナーや開発者、マーケターがこれを考慮に入れていないようです。

メッセージングアプリケーションのリンクは、比較的非常にまれに使用されており、採用率は1%未満です。これは、モバイルウェブ上での「ディープリンキング」の成長によって説明される可能性があります。ディープリンキングでは、アプリを直接開いてアプリ内の深いセクションにナビゲートできます。

## 入力フィールド

良いモバイルウェブ体験には、エンゲージメントとインタラクティビティが不可欠ですが、多くの開発者がモバイルユーザーのために特定のタイプ宣言を設定することを長年見過ごしてきました。これは、ユーザーが最終的に対話を決定したときに、すぐに正しいキーボードが表示されるようにするためです。正しい設定は、ユーザーが電話番号を入力する際には数字キーボードを、メールアドレスを入力する際には `@` 記号が含まれるキーボードを返します。その他の入力フィールドのタイプには、送信機能、検索、チェックボックス、パスワードフィールド、ラジオボタン、その他のボタン、または非表示要素が含まれます。

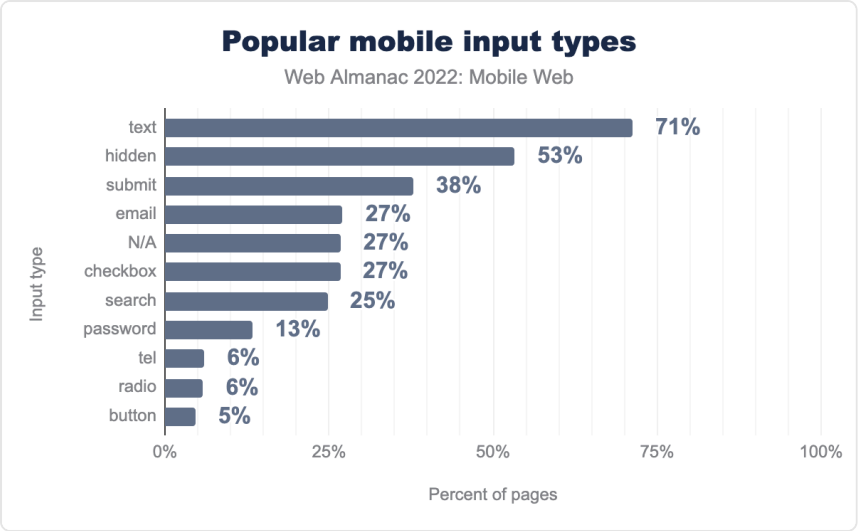



*図15.4. 入力タイプの採用。*

モバイルではデスクトップよりも入力フィールドによるキーボードの変調がより重要ですので、彼らのウェブサイトアーカイブを通じてもっとも人気のあるモバイル入力タイプを追跡しています。2022年では、`text`が依然としてもっとも人気のある入力タイプで、71%のページで使用されており、これは昨年[573]の73%からわずかに減少しています。次に一般的な入力タイプは、53%のページで`hidden`、そして38%で`submit`です。

もっとも一般的な入力タイプの後には、約四分の一のページに現れる入力タイプのクラスターがあります。これには、`email`、入力フィールドなし（`N/A`）、`checkbox`が27%で、`search`が25%で続きます。

昨年と比較して、`checkbox`、`email`、`search`の普及率が少し増加しています。これは、以前の年よりもモバイルのユースケースに対する注意が増えたことを示している可能性があります。その際、より多くの入力フィールドが単に`text`として無差別にまとめられていた可能性があります。

### 高度な入力タイプ

高度な入力タイプには、`color`、`date`、`datetime-local`、`email`、`month`、`number`、`range`、`reset`、`search`、`tel`、`time`、`url`、`week`、`datalist`が含まれます。これらは、ブラウザでの入力方法を変更するためにHTML 5仕様で導入されたため、*高度*と見なされます。これは一般的に、ユーザーがモバイルサイトで対話できるオプ

573. *https://almanac.httparchive.org/ja/2021/mobile-web#type-declarations*

ションの表示を変更します。


*図15.5. 高度な入力タイプを使用するモバイルページの割合。*

少なくとも1つの入力を持つページの47%が1つ以上の高度な入力タイプを使用しており、これは2021年の45%から増加しています。

一般的に、特殊な入力フィールドとプロトコルの使用は、訪問者の対話をよりエンゲージングで有益、効率的にするのに役立ちます。モバイルウェブで入力フィールドや関連機能がしばしば見過ごされることは残念ですが、まったく落胆することではありません。これらのタイプのモバイルコントロールは多くの年月を経ていますが、時代遅れになっているか、現在はウェブアプリの機能やJavaScriptによって実行されている可能性があります。

同様に、ネイティブアプリ内の深い画面を直接開き、ナビゲートすることができるディープリンクの進化に伴い、特殊な入力フィールドが必要なくなるかもしれません。ブランドがより良い体験を提供できると信じている場合や、ウェブサイトよりも全体的なコンバージョン統計が優れていると考える場合、訪問者をウェブサイトからアプリへと移行させようとすることがあります。これはとくに、アプリが会社の主な焦点である場合や、アプリがウェブサイトよりも積極的に更新されている場合に当てはまります。技術的に必要なことはほとんどありませんが、ブランドの好みや独自の内部成功指標については主に関係しています。

入力フィールドのバリエーションは一般的に、企業にとってより利益をもたらすウェブインタラクションを作り、ユーザーにとってより効率的なものにするために含まれています。これらはしばしばデジタルチェックアウトプロセスに関連しており、そのため、直接的に利益に影響を与える可能性があります。いくつかのブランドにとって、この機能の重要性は、開発者が購入を完了するためにユーザーをネイティブアプリに押し進める可能性が高くなるかもしれません。これは、処理を合理化し、保存されたユーザーロイヤリティ情報を活用し、取引の完了を速めるためです。この種のインタラクションにネイティブアプリが優れているという認識は誤解を招く可能性がありますが、現在ほとんどの消費者はウェブからアプリに移行するか、その逆を強制されることなく、どこで始めてもスムーズでシームレスな体験を期待しています。可能であれば、アプリとウェブの機能の間の同等性を最優先事項とし、高度な入力方法はとくにチェックアウトプロセスなど、利益に影響を与えるものに役立つことがあります。

伝統的な入力フィールド（リンクプロトコル、入力タイプ、高度な入力タイプであるかどうか）は、アプリへのディープリンクによって処理されることがあり、これらは異なる方法で処理されます。アプリへのリンクのプロトコルは一般にディープリンキング、iOSでは *Universal Links*、Androidでは *Android App Links* と呼ばれます。コード内では、これらのリン

クは通常のウェブリンクのように見え、アプリの起動はウェブサイトのルートにある `/well-known` ディレクトリにホストされているウェブアプリマニフェストファイルによって処理されます。それを踏まえて、これらの数値の変化とバリエーションが実際の解釈で何を意味するかについての仮定を立てるのは困難です。損失と利得が絶対的であるか、またはこれらのモバイルウェブの側面において技術と規範の移行が進行中であるかどうかはわかりません。

このトピックを別の視点から見ると、開発者がサイト上のすべてのものに対して入力タイプとバリエーションをとくにコーディングするという期待が高すぎる可能性もあります。モバイルブラウザがコードからの明白な手がかりや以前のユーザーの対話に基づいて、特定のリンクや入力フィールドに対する適切なアクションやキーボードレイアウトを決定するために、より多くの重労働を行うことが可能です。ブラウザは、この研究を活用してそのような体験を最適化することさえできるかもしれません。

たとえば、特定の入力フィールドをクリックするすべてのユーザーが数字キーボードに切り替えたり、数字のみを送信したりする場合、ブラウザはそのような対話とメタデータを使用して、デフォルトで数値キーボードをヒューリスティックに有効にする方法を見つけるかもしれません。ブラウザがこれまで複雑で非効率的だったタスクを簡素化する傾向を続けることを願っています。これにより、開発者は負担を軽減されます。ブラウザがインタラクティビティの側面でより多くの責任を負うと、サイトの所有者に良いユーザー体験のすべての責任を押し付けるのではなく、その名前の役割である *ユーザーエージェント* としての役割を果たすことができます。これらのヒューリスティックなデフォルトは、機能の強化された潜在能力が利用可能な場合にアプリへのディープリンクとともに、ウェブ専用サイトの理想的な進むべき道のようです。

## モバイルウェブにおけるアクセシビリティ

モバイルデバイスは、コンピューターよりも安価で軽量で持ち運びやすいため、技術によって歴史的に無視されたり疎外されたりしてきた人々を支援する大きな可能性を秘めています。Googleによると[574]、ウェブ上での真のアクセシビリティは「サイトのコンテンツが利用可能であり、その機能を文字通り誰もが操作できる」ということを意味します。さらに詳しく説明すると、Googleは以下のように述べています。

> *アクセシビリティとは、狭い範囲の「典型的な」ユーザーの範囲外にいるかもしれないユーザーの経験を指します。彼らは予想とは異なる方法で物事にアクセスしたり、対話したりするかもしれません。具体的には、何らかの障害や不自由を抱えるユーザーを懸念しており、その経験は非物理的または一時的である可能性があります。*

574. *https://web.dev/articles/accessibility?hl=ja#what-is-accessibility*

ウェブアクセシビリティとモバイルウェブアクセシビリティは、ウェブ全体の対話と一般的な意識の中でその重要性と存在感を進化させています。よりアクセシブルなサイトを構築することで、モバイルウェブがより多くのユーザーや潜在的な顧客に到達することを可能にします[575]。また、情報と関連ブランドを包括的で現状に合ったものとして位置づけるのにも役立ちます。これは、「平均的なユーザー」のみを考慮することで無神経さを助長し、さらなる疎外を引き起こすのではなく、 *すべてのユーザー* のニーズに触れることを意味します。

アクセシビリティの懸念は、状況的、一時的、永続的の3つのタイプに分けることができます。ほとんどの場合、ウェブアクセシビリティの側面が永続的な障害を持つ人にとって重要であると同時に、状況的または一時的な障害を持つ人々にとっても非常に役立つと想像することが可能です。Google検索結果でのオーガニックランキングの向上[576]という副次的な利点を持ちながら、ウェブアクセシビリティ基準を満たすことは、アクセシビリティの側面を明示的に評価することではありません。

W3C[577]は、ウェブサイトを *perceivable*（見たり聞いたりできるもの）、*operable*（使用できるボタンやジェスチャー）、*understandable*（レイアウトや表現に関する懸念）、*robust*（フォームや情報を入力して送信する能力）にするためのアクセシビリティの基本的な原則を4つのグループにまとめています。ここで焦点を当てる要素は主に *perceivable* と *operable* のグループに含まれ、色のコントラスト、タップターゲット、ズームやスケーリングが含まれます。

モバイルアクセシビリティへの注目が高まることで、モバイルおよびデジタルコミュニケーションの機能と重要性が再評価されるかもしれません。とくに、デジタル技術とコミュニケーションの人間性を再確認するのに役立つ場合です。デジタル接続が増えることで実際には孤立や現実の世界との接続の欠如を助長している文化では、これはとくに重要です。モバイルウェブをよりアクセシブルにすることは、モバイル技術と社会全体に直接的ならびに間接的な好影響をもたらすべきです。

## 色のコントラスト

色のコントラストは、多くの理由からモバイルウェブアクセシビリティにおいて重要な側面であり、モバイルサイトをレビューする際に見つけやすく更新しやすいアクセシビリティニーズの1つです。モバイルデバイスでは、画面が常に小さく、通常よりもフォントが小さいため、良好な色のコントラストがあると、コンテンツの可読性に大きな違いをもたらすことがあります。色覚異常、緑内障、白内障などの一般的な目の問題がある場合、モバイルフォンでウェブコンテンツを見ることは困難または不可能です。

完璧な視力を持つ人々でさえ画面が汚れていたり、まぶしさが多かったり、ユーザーが明るい日光の下にいる場合など、画面のモバイルウェブコンテンツを消費するのに苦労すること

575. *https://moz.com/blog/seo-and-accessibility-introduction*
576. *https://searchengineland.com/seo-accessibility-tips-deaf-disabled-386577*
577. *https://www.w3.org/TR/mobile-accessibility-mapping/*

があります。モバイルサイトの色のコントラスト比が高いと、悪条件下でも、また完璧な視力を持つ人々にとっても、使用しやすく感じられるようになります。

それを踏まえて、多くの人々が完璧な視力を持っておらず、眼鏡やコンタクトレンズを使用して視力を補助しています。完璧な視力を持つ人々でも、年齢とともにある程度視力が低下すると予想されるため、このアクセシビリティ基準は、最終的にはすべての人に影響を与えると考えられます。


*図15.6. 十分な色のコントラストを持つモバイルページの割合。*

リスクを考えると、今年のモバイルサイトのわずか23%が適切な色のコントラストを持っているというのは残念です。これは2021年の統計（22%）からわずか1ポイント増加したもので、改善はありますが、大きな改善ではありません。さらに遡ると、この統計は2019年からほとんど改善されていないので、ウェブ上でもっとも影響力のあるアクセシビリティ基準と広く考えられているものについては失望感があります。

米国一般サービス管理局[578]によると、サイト所有者は「テキストと背景のコントラストが小さいテキストで4.5:1以上、大きいテキストで3:1以上であることを確認すべきです。」W3C[579]もこの比率を支持しており、さらに「少なくとも7:1（または大規模なテキストの場合は4.5:1）のコントラスト」を求める拡張ガイドラインを追加しています。明らかに、*拡張されたアクセシビリティ* の基準を満たすモバイルサイトはもっと少なく、その統計は23%よりもはるかに低い可能性があります。

サイトの知覚性を向上させ、実際に使用可能にするための色のコントラストを改善するというインセンティブが十分でない場合、テキストと背景色の間に最低限のコントラストを持つことは、Google検索でのランキングにとっても長い間重要な要素でした。これは、スパマーがウェブサイトに白いテキストを白い背景の上に含めるのを防ぐ手段として始まりました。その後、Googleはこれを最初のモバイルフレンドリー[580]ガイドラインの一部に移行し、現在は単にユーザーにとって良いウェブサイトを設計するための一般的なガイドラインの一部として含まれています。

とくに新しい色の機能がブラウザに追加されたことを考慮すると、ウェブマスターに色のコントラストに焦点を当てることを奨励したいと考えています。たとえば、`dark-mode` のような機能です。サイトの更新を行い、`dark-mode` でのサイトの互換性を向上させる場合、理想的にはそのプロジェクトの一環として全体的な色のコントラストも改善する時間を取るべきです。この機能は一部の人にとって使用しやすいので、この組み合わせた努力を二重の

578. *https://accessibility.digital.gov/visual-design/color-and-contrast/*
579. *https://www.w3.org/TR/mobile-accessibility-mapping/#h-contrast*
580. *https://developers.google.com/search/mobile-sites/get-started*

利益として数えることができます！

## タップターゲット

タップターゲットは、ページ上のクリック可能な要素を効果的に表します。多くのデザイナーや開発者はまだ、ウェブサイトをデスクトップのインタラクション用に設計されたものと考えがちですが、 *モバイルファースト* ではなく、タッチスクリーンのインタラクションに十分なスペースを確保することが見落とされがちです。モバイルデバイスでは、小さい画面に合わせて要素がリサイズされたり再配置されたりするため、リンクされた要素が互いに近接して配置されることが一般的です。これが問題となるのは、ナビゲーションメニューやフッターに見られるように、複数のクリック可能な要素が集まっている場合です。

適切なサイズのタップターゲットを持つことで、ユーザーが誤って間違ったボタンやリンクをクリックする可能性が低くなり、もう一度クリックを試みるために戻る必要がありません。また、ユーザーにモバイル画面をズームインして正しいボタンをクリックすることを期待するのも理想的ではありません。アクセシビリティの観点から見ると、適切なサイズのタップターゲット[581]を持つことは、運動障害があるユーザーや微細な運動技能に問題があるユーザーにとっても重要です。

**42%**

*図15.7. 十分なタップターゲットを持つモバイルページの割合。*

タップターゲットの最小サイズは一般的に48ピクセル×48ピクセル未満であってはならず、これはタッチスクリーンで使用される指のサイズのおおよその推定値です。また、タップターゲットは、Googleの評価を通過するためには、互いに最低8ピクセル離れている必要があります。私たちの調査によると、モバイルサイトの42%が十分なタップターゲットを持っており、適切なサイズのタップターゲットを普遍的に持っているサイトが半数以下であることは残念です。

581. *https://www.w3.org/WAI/WCAG21/Understanding/target-size.html#benefits*

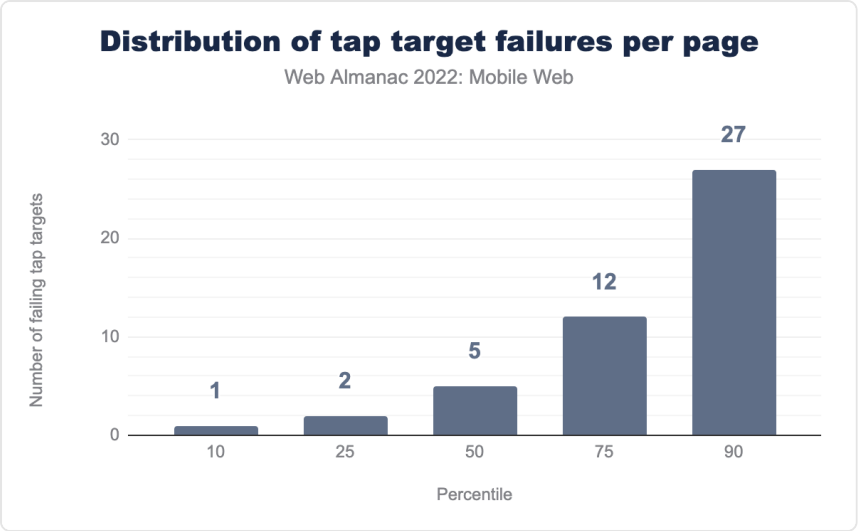



*図15.8. ページごとのタップターゲットの失敗数の分布。*

監査に失敗したページは、通常、複数のリンクが失敗しています。失敗したタップターゲットの中央値は5ですが、サイトが失敗する場合、失敗したタップターゲットの数は非常に多いことがあります。最悪の10%のサイトでは、少なくとも27の失敗したタップターゲットがあります。

## ズームとスケーリング

モバイルデバイスは多くの人々の日常生活の大きな部分となっており、モバイルウェブコンテンツとの対話は迅速かつ簡単であることが期待されています。ウェブサイトがズームとスケーリングをどのように扱うかは、モバイル上での対話を改善するための長い道のりとなる可能性があります。これには異なる見解があり、ほとんどの人はモバイルユーザーのために適切な初期スケールをビューポートに設定する必要があることに同意していますが（`<meta name="viewport" content="width=device-width, initial-scale=1">`）、ズームやスケーリングを無効にすべきかどうかを制御するビューポート設定の第二部分については、一致した見解はありません（`…user-scalable=no">` もしくは `…user-scalable=yes">`）。W3Cを含むほとんどの機関[582]は、スケーリングとズームを制限することはユーザー体験を損ない、アクセシビリティに悪影響を及ぼす可能性があるため、避けるべきであると提案しています。`minimum-scale` と `maximum-scale` の設定も可能であり、開発者が制限が必要だと考える場合、これらはしばしばより安全な限界です。

ズームは、視覚障害があるユーザーや、単に読書用メガネが手元にないときに必要な人にと

582. *https://www.w3.org/WAI/standards-guidelines/act/rules/b4f0c3/proposed/*

って良い回避策になることがあります。一方で、モバイル上で広範囲にわたってうまくスケールするサイトを構築することは困難です。対応する必要のある多くの異なるフォントサイズ設定があり、間違えるとサイトとの対話が非常に困難になります。このため、一部のデザイナーはページがスケーリングやズームの影響を受けず、予測可能な方法でレンダリングされることを保証するために、スケーリングやズームを防ぐことを好むことがあります。これは事実ですが、ズームやスケーリングを無効にすることはモバイルサイトの使いやすさを妨げるため、アクセシビリティのために避けるべきです。

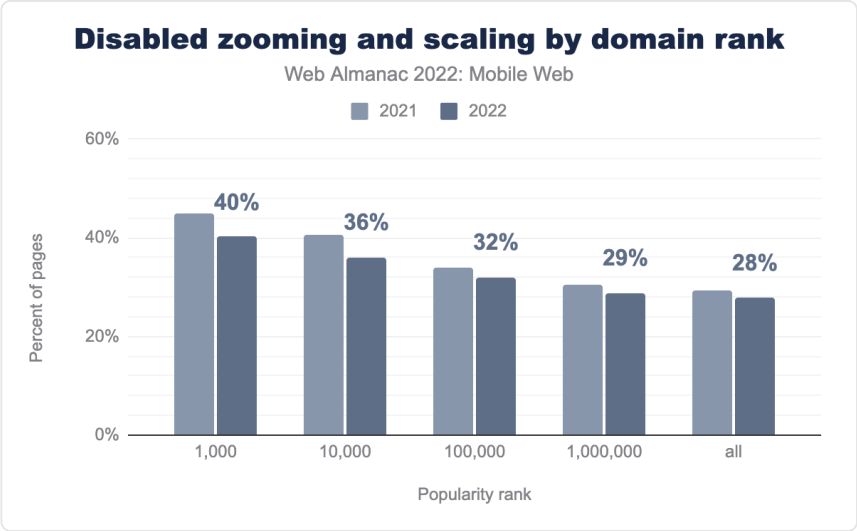



*図15.9. ズームとスケーリングを無効にしているウェブサイトの割合の年次比較、人気度による区分。*

上位1,000のモバイルウェブサイトのうち、40%がズームとスケーリングを無効にしており、これは2021年の約45%から減少しています。上位10,000を見ると、モバイルサイトの36%がスケーリングとズームを無効にしており、これは2021年の約41%から減少しています。データセットに含まれるすべてのサイトを含むもっとも広いランキンググループでは、28%のサイトがズームとスケーリングを防止しており、これは2021年の約29%から減少しています。全体として、このアクセシビリティを制限する設定の普及率はとくにもっとも人気のあるサイトで減少していることがわかりますが、まだ半数以上のサイトがこのタイプの制限的な設定を使用しているという事実は残念です。

全体として、アクセシビリティの懸念はなくなることはなく、時間が経つにつれてアクセシビリティ基準を満たすことが基本的な期待となるでしょう。とくにモバイルデバイスでは、モバイルインタラクティビティのユースケースはデスクトップと比較して広範囲にわたるため、ユーザーの期待は高くなっています。しかし、ウェブモバイルコンテンツの開発制約が実際に達成することをより困難にしています。それにもかかわらず、アクセシビリティは優

れたウェブデザインの重要な要素となりつつあり、より包括的なモバイルウェブを作成するために受け入れるべきです。基本的なアクセシビリティガイドラインへの遵守が増えているのを見るのは良いことですが、まだ改善の余地は大いにあります。

Googleは、一定の基本要件を満たすウェブサイトにより良いランキングを与えることで、長年にわたりウェブに肯定的な影響を与えてきました。彼らは読み込み時間、パフォーマンス、セキュリティ、モバイルフレンドリー性に対してこれを行ってきましたが、まだアクセシビリティに対しては行っていません。Googleは公式コミュニケーションでウェブアクセシビリティの進歩を多く書いてサポートしていますが、もっとできることがあります。多くのアクセシビリティ更新は自然にウェブサイトのランキングに肯定的な影響を与えますが、Googleが基本的なアクセシビリティ基準へのコンプライアンスの一定レベルを明示的に良いランキングでインセンティブを与える時期かもしれません。これは、彼らがウェブサイトの意味論的理解を強化できるだけでなく、単純にウェブをすべての人にとってより良い場所にするためです。

## モバイルパフォーマンス

サイトオーナーがモバイルWebで対処しなければならないもっとも複雑な問題の1つがパフォーマンスです。これは、ユーザーが小型デバイスでウェブサイトをリクエストまたは操作する際に発生する遅延として経験されます。モバイルデバイスはかなり進化していますが、一般的には大型デバイスよりも処理能力が劣り、より遅くて信頼性の低いインターネット接続を使用することが多いです。さらに、プランの制限を超えた場合にモバイルデータリクエストのコストが発生する可能性があるため、効率性はユーザーエクスペリエンスや速度への影響を超えて、実用的な問題としても重要です。

### コアウェブバイタル

Core Web Vitals[583]は、Googleがさまざまなウェブサイトを評価するためにまとめたパフォーマンス指標のコレクションで、とくに、ウェブサイト上の異なるページグループ[584]がモバイルおよびデスクトップページ設定でどのように機能しているかを説明します。Core Web Vitalsの要素には、ローディング、インタラクティビティ、ならびにレイアウトの安定性が含まれます。

これらの3つは、ページのパフォーマンスをどのようにユーザーが認識するかに関係し、ユーザーのローディング体験を助けたり妨げたりすることがあります。この種の評価は2020年5月に始まり、これらの指標は、ページエクスペリエンス[585]の評価の一環として具体的にGoogleのランキングアルゴリズムで考慮され、したがって指標は「良い」や「改善が必要」

583. *https://web.dev/articles/vitals?hl=ja*
584. *https://support.google.com/webmasters/answer/9205520#page_groups*
585. *https://developers.google.com/search/docs/appearance/page-experience?hl=ja*

または「悪い」とされるパフォーマンスの閾値を中心に組織されています。サイトが「良い」とみなされるためには、訪問の75%が各Core Web Vitals指標の規定された閾値を満たしている必要があります。

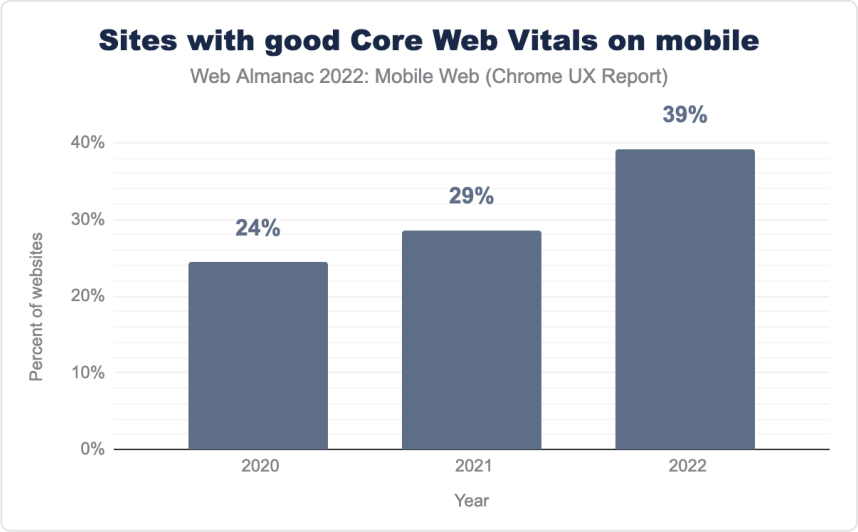



*図15.10. モバイルで良好なCore Web Vitalsを有するウェブサイトの割合の年次比較。*

上記の図は、Core Web Vitalsが2020年に導入されて以来、ウェブ全体のパフォーマンスがどのように変化したかを示しています。全体として、モバイルウェブサイトは年々着実に改善されています。2022年には、ウェブサイトの39%がモバイルデバイス上で良好なCore Web Vitals体験を提供しています。今年の顕著な変化の原因については、パフォーマンスの章でさらに詳しく調べることができます。

## ローディングパフォーマンス指標

モバイルWeb上でのロード時間は、他のどの文脈でも測定される方法と同じです。モバイルデバイスはサイトを素早くロードするのが困難な負担をより多く抱えているにもかかわらず、研究[586]によると、モバイルビジターはより不耐性があり、実際には大型デバイスよりも電話での体験をより速くしたいと期待しています。

### 最初のバイトまでの時間（TTFB）

最初のバイトまでの時間 (TTFB)[587]、しばしばTTFBと略される、はナビゲーションの開始から

586. https://unbounce.com/page-speed-report/#:~:text=Young%20folks%20have,on%20a%20cellphone.
587. https://web.dev/articles/ttfb?hl=ja

リクエストに対する最初のデータバイトが受信されるまでの経過時間を測定します。TTFBは、ページの構築を開始するために必要なサーバーやその他のネットワークリソースの反応性を説明します。

TTFB自体はコアウェブバイタルの指標ではありませんが、すべてのローディングパフォーマンス指標に直接影響を与えるため、しばしばすべてのコアウェブバイタル要素と一般的なサイトパフォーマンスの最適化の一部として議論されます。とくにもっとも重要なコンテンツの描画です。

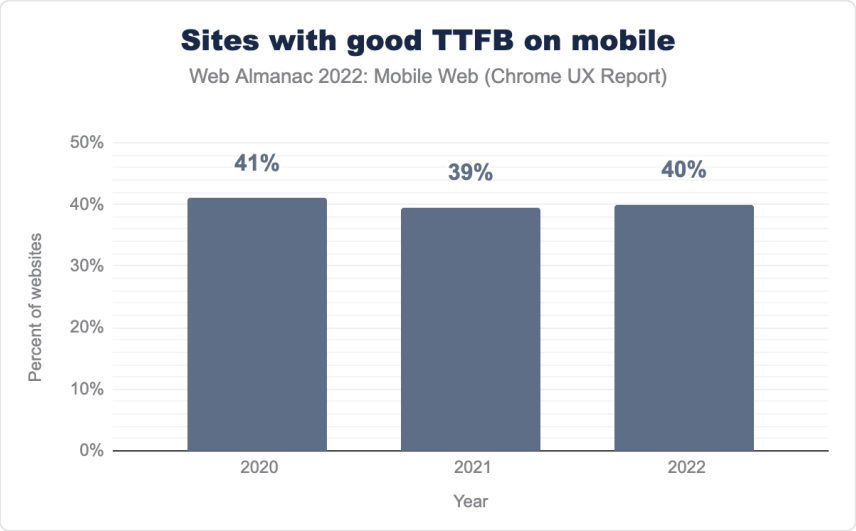



*図15.11. モバイルで良好なTTFBを有するウェブサイトの割合の年次比較。*

上記のように、2020年から2022年までの間にモバイルサイトが「良好」とされる割合にはわずかな変動があります。2020年の41%から2021年には39%に下がり、2022年には再び40%に上がりました。

### もっとも重要なコンテンツの描画（LCP）

もっとも重要なコンテンツの描画 (LCP)[588]、またはLCPは、ウェブサイトがユーザーに最初に表示される意味のあるコンテンツの最大部分をロードするのにかかる時間を説明する指標で、しばしばヘッダー画像やデザインのサイズとパフォーマンスの機能です。LCPは重要です。なぜなら、それがユーザーにページが消費を開始する準備ができたことを知らせるからです。

588. *https://web.dev/articles/lcp?hl=ja*

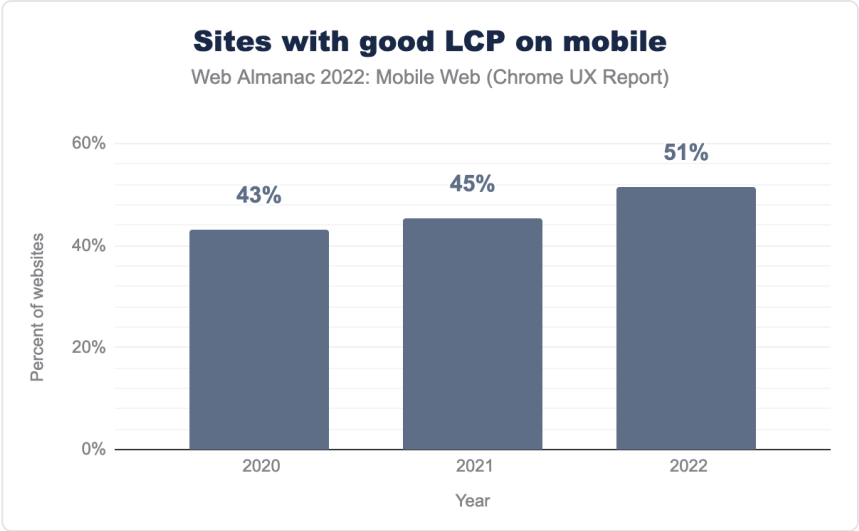



*図15.12. モバイルで良好なLCPを有するウェブサイトの割合の年次比較。*

LCPのパフォーマンスは改善しています。2020年には、モバイルサイトの43%が「良好」と評価されたLCPを持っていました。2021年にはこの数値が45%に改善し、2022年には顕著な跳躍があり、51%のモバイルサイトが良好なLCPパフォーマンスを有していました。パフォーマンスの章では、これがなぜ起こったのかについてのいくつかの可能性のある説明を探求します。

## 画像

画像は優れたモバイル体験を提供するために大いに寄与できますが、適切に設定されていない場合は、パフォーマンスの低下や悪いローディング体験にも大きく寄与することがあります。このセクションでは、サイトオーナーが画像のパフォーマンスへの影響をどのように扱っている（または扱っていない）かを探ります。

### 適切なサイズの画像

モバイルデバイスに適切なサイズの画像を使用することは、誰にとってもモバイルのロード時間を改善するもっとも簡単な方法の1つです。モバイルWebの初期には、サイトオーナーはしばしば、最終的にモバイルブラウザがページのモバイルレンダリングに合わせて画像をスケールおよびリサイズするため、デスクトップユーザーと同じ画像をモバイルユーザーに送信していました。残念ながら、これはうまく機能しませんでした。なぜなら、デスクトップ視聴体験に適した豊かで高品質の画像を送信するために多くの追加データが必要とされたからです。

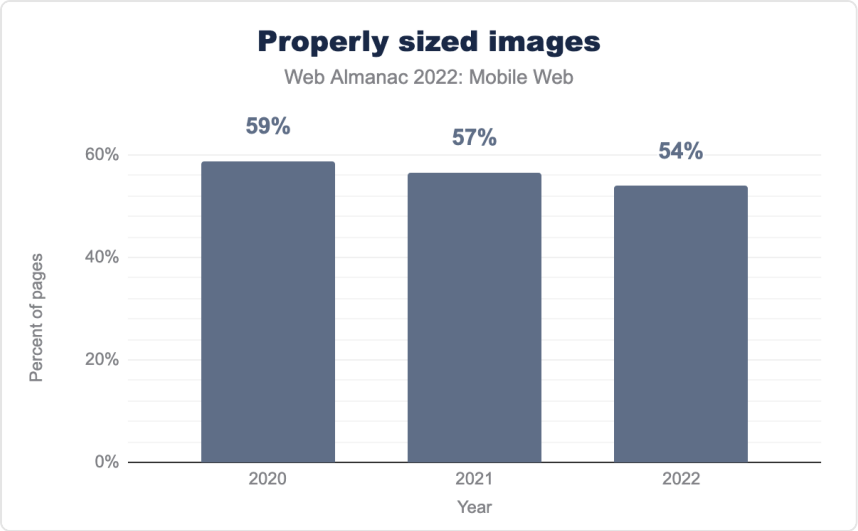



*図15.13. 適切なサイズの画像を持つウェブサイトの割合の年次比較。*

GoogleがCore Web Vitalsプログラムで良好なパフォーマンスを報酬化することに注力しているため、より多くのサイトが画像を最適化すると期待されます。しかし、時間が経つにつれて最適化された画像が減少していることは興味深いです。上の図は、2020年に59%のサイトが適切なサイズの画像を持っていたのに対し、2022年にはその数が54%に減少したことを示しています。

### レスポンシブ画像

異なる画面サイズに対応する画像を作成することは、モバイル画像サイズを扱う一般的な方法です。レスポンシブ画像を使用することは広いスクリーンのテレビでウェブサイトを見る、デジタルアシスタントに接続してウェブサイトを見る、あるいは小型の機能電話や携帯型ゲームシステムで見るなど、もっともユニークなプレゼンテーションシナリオでもウェブサイトが対応する素晴らしい方法です。

画面に画像を埋め込む主な方法は2つあります：`img`要素と、より拡張された`picture`要素です。`picture`要素は、特定の基準に基づいて画像を含めるいくつかの可能性を提供します。両方の要素で利用可能な`srcset`属性は、画面サイズや表示密度などに基づいて条件付きで画像を含めることを可能にします。一部のブラウザは、適切な画像を選択する際に帯域幅も考慮する場合があります。

`picture`要素はこれらの機能をさらに拡張し、モバイルの縦画面には4:3比の画像、デスク

トップや横画面には16:9の画像を指定するなど、アートディレクション[589]を可能にします。さらに異なる画像形式を指定する使用法もあります。たとえば、ブラウザはAVIF画像をサポートしている場合にはそれをロードし、そうでない場合はWebPやPNG画像にフォールバックします。ブラウザが運用条件に基づいて賢明な選択をすることで、通常はパフォーマンスが向上し、したがってユーザー体験も向上します。

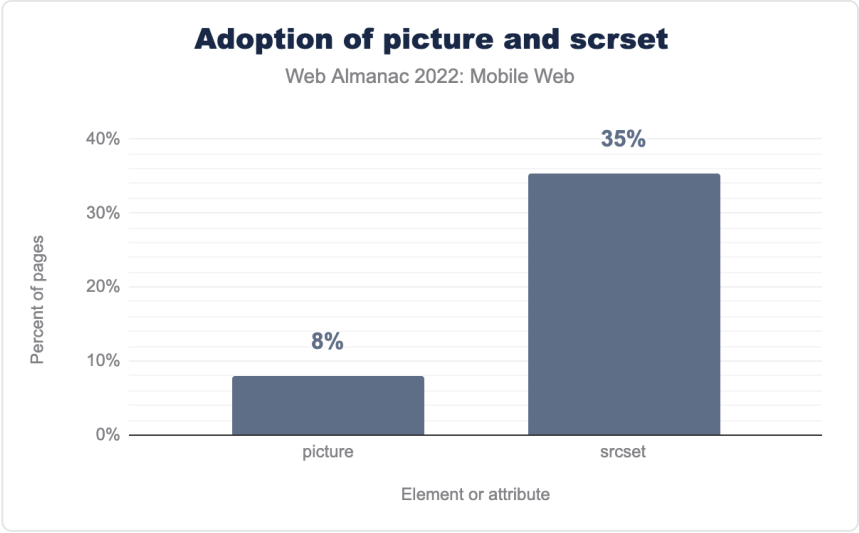



*図15.14. `picture` 要素と `srcset` 属性の採用。*

`picture` 要素の採用はモバイルページのわずか8%（昨年の6%から増加）ですが、`srcset` は35%に達しています。2021年には `srcset` の使用が約32%だったので、約3ポイントの成長を見ています。

全体的に、レスポンシブ画像技術の使用の増加を見るのは素晴らしいことです。アートディレクションと形式のフォールバックサポートでより柔軟性を提供するにもかかわらず、`picture` の採用が比較的少ないのは、より多くの作業が必要であり、WordPressのような人気のあるCMSシステムではデフォルトでサポートされることが少ないためかもしれません（現時点では[590]）。

### レイジーローディング

レイジーローディングは、ページ上の要素に異なるローディング優先度レベルを割り当てるプロセスです。レイジーローディングがなければ、ページ上のすべての要素と画像が最終的

589. https://developer.mozilla.org/ja/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images#art_direction
590. https://github.com/WordPress/performance/issues/21

にはロードされますが、レイジーローディングを使用すると、ユーザーがページ上でスクロールした位置に基づいて画像のロードを遅延させることができます。とくにモバイルデバイスにおいてレイジーローディングは重要であり、一般的なレスポンシブデザインパターンではほぼ常にコンテンツがモバイルレンダリング用に積み重ねられます。モバイル画面の狭い特性は、ページの多くの積み重ねられた要素が画面下部に押し下げられ、とくにユーザーがトップナビゲーションのリンクをクリックするだけの場合には、すぐには必要ない可能性があります。レイジーローディングは、その不要なデータ転送とそれに関連するロード時間を排除します。


*図15.15. 画像に `loading="lazy"` を使用しているモバイルページの割合。*

ネイティブレイジーローディング[591]は2019年から利用可能で、ブラウザがもっとも効率的な方法で複雑な計算を行うことを可能にし、サイトオーナーは画像に `lazy` または `eager` をタグ付けするだけで済みます。このシンプルなタグ付けは、ページとサイトのパフォーマンスを大いに向上させることができ、自前のレイジーローディングコードを維持する際の多くの時間と労力を節約できます。ただし、ページのトップにあるLCP画像を誤ってレイジーロードしない[592]限り、これは簡単な勝利ですが、現在 `loading=lazy` 属性を使用しているサイトは25%に過ぎません。

## レイアウトの安定性

レイアウトの安定性は、GoogleがCore Web Vitalsを導入したことで最近注目されているパフォーマンスの重要な部分です。Core Web Vitalsの要素は、ページのローディング体験を測定および評価するために設計されており、ページのレイアウトの安定性はその重要な部分です。ページがロードされている間に常に動いたり再描画されたりすると、ページがロードされるのにかかる時間が実際よりもはるかに長く感じられます。一度ロードされると、物事がロードされた場所に正確に留まるより予測可能で安定した体験であれば、そうではないでしょう。

コンテンツのローディング順序は異なる条件や異なるブラウザでは異なる可能性があるため、レイアウトの安定性を考慮して構築および計画することは、どのような状況下でもスムーズなローディング体験を保証する良い方法です。レイアウトの安定性を評価するためのもっとも関連性の高い指標は累積レイアウトシフト（CLS）ですが、画像のサイズやレイジーローディングなどのパフォーマンスの側面もレイアウトの安定性に影響を与える可能性があります。

591. *https://web.dev/articles/browser-level-image-lazy-loading?hl=ja*
592. *https://web.dev/articles/lcp-lazy-loading?hl=ja*

## 累積レイアウトシフト（CLS）

累積レイアウトシフト（CLS）[593]は、ページが使用中にどの程度安定しているかを表す指標です。

低いCLSは視覚的に安定したレイアウトを表し、ユーザーにとってイライラすることが少ない体験を提供します。高いCLSを持つページは、画像がロードされ始めてテキストが画像の周りにフィットするためにレンダリングされるとき、またはフォントファイルがロードされてフォントによって引き起こされる違いを収容するためにページが描かれなければならないとき、動きが生じることがよくあります。これは *reflow* として説明されることもあります。また、大きなヘッダー画像がロードされてページ上のすべてのコンテンツを移動させることもこれに含まれます。これらの出来事は俗に *jank* と表現されていました。

ページをリクエストするデバイスの画面サイズは、要素がどのように配置され、シフトを引き起こす要素がロードされたときにどれだけ移動するかに大きな影響を与える可能性があります。

CLSはスコアとして測定され、ページの寿命中の任意のセッションウィンドウでの最大の動きのインスタンスが測定されます。これは2021年に変更されました[594]。CLSは以前、ページ上のすべての個々のシフトスコアの合計として測定されていました。Googleは0.1以下のスコアを「良好」と考え、0.25を超えるスコアを「悪い」と考えています。

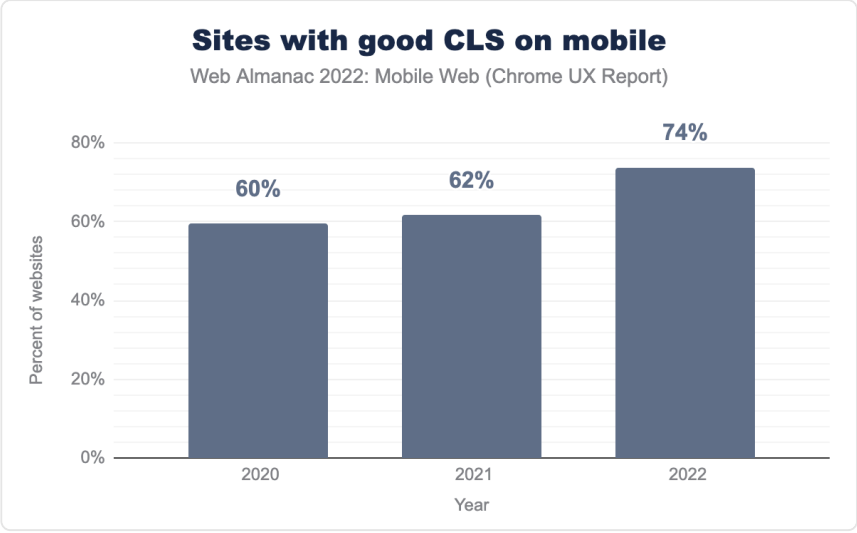



*図15.16. モバイルで良好なCLSを有するウェブサイトの割合の年次比較。*

593. https://web.dev/articles/cls?hl=ja
594. https://web.dev/articles/cls-web-tooling?hl=ja

モバイルで「良好」なCLSを持つウェブサイトの割合は、昨年の62%から大幅に改善され、74%になりました。

## レスポンシブ性

モバイルシナリオではレスポンシブ性が常に良いものですが、それには複数の意味があります。一般的に、レスポンシブな技術は良いものであり、与えられた手がかりに対して効率的かつ意味のある方法で反応します。デザインの観点からモバイルのレスポンシブ性について話すとき、私たちは異なる画面サイズのデバイスに要求されたときにそのレイアウトを調整するコンテンツを指します。もっと広い意味では、レスポンシブ性はページがユーザーのインタラクションにどれだけ迅速で効率的に反応するかをも示します。このタイプのレスポンシブ性はレイアウトについてではなく、迅速なインタラクションに関するものです。非常にレスポンシブなサイトを持つことは良いことです。なぜなら、ユーザーがサイトとインタラクトしたときにすぐに認識される効率的なユーザーインタラクションが作成されるからです。この種のレスポンシブ性を評価するために使用される指標は、ファーストインプット遅延（FID）とインタラクションから次のペイントまで（INP）で、このセクションで取り上げます。

### ファーストインプット遅延（FID）

ファーストインプット遅延（FID）[595]は、サイトがユーザーがページ要素をはじめてクリックした後、サイトが応答するまでにかかる時間、とくにそれがどれだけ長くかかるかに関連して、サイトのレスポンシブ性を説明します。低いFIDは望ましいですが、とくにモバイルウェブインタラクションでは、モバイルサイトのレスポンシブ性が理想的には比較可能なネイティブアプリのそれと競合し、インタラクションが同じくらい流れるように感じさせることが重要です。Googleは、体験の75%以上が100ms未満である場合、サイトに「良好な」FIDがあると考えています。

595. *https://web.dev/articles/fid?hl=ja*

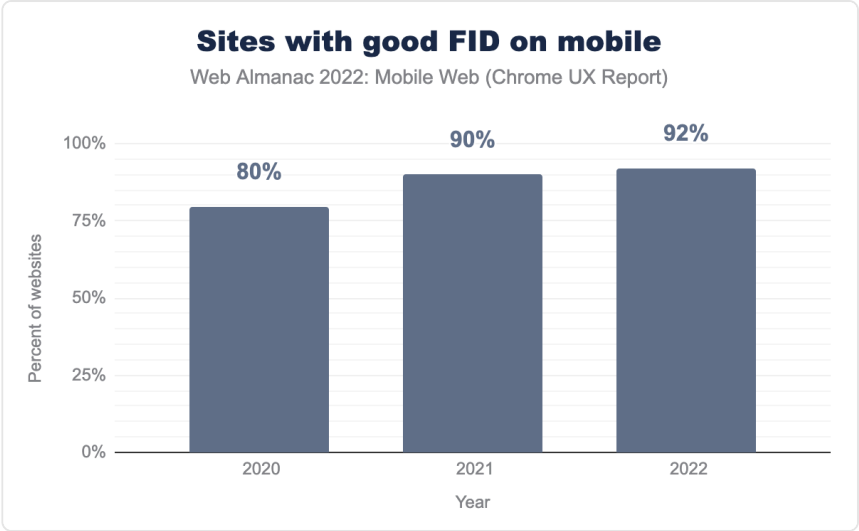



*図15.17. モバイルで良好なFIDを有するウェブサイトの割合の年次比較。*

過去3年間で「良好な」FIDを持つモバイルサイトの割合には一貫した成長があり、2020年の80%から2021年の90%、2022年には92%に達しました。

### インタラクションから次のペイントまで（INP）

インタラクションから次のペイントまで（INP）[596]は、Googleが実験的に使用している指標で、ユーザーがそれと対話するときにページのレスポンシブ性と応答時間を測定するために使用されます。低いINPは望ましいです。これは、ページがユーザーの対話に対して迅速に応答でき、要求された後のコンテンツの描画を待つための大幅な遅延がなかったことを意味します。「良い」INPは200ms以下であり、悪いものは500msを超えるものです。最終的には、INPは公式のCore Web Vitals指標に追加される可能性がありますが、現在はそれが信頼できる一貫した指標であり、サイトオーナーにとって有用であることを確認するためにまだテストされています。


*図15.18. モバイルで良好なINPを有するウェブサイトの割合。*

INPデータが利用可能になったのは今年がはじめてで、歴史的な文脈はありませんが、モバ

596. https://web.dev/articles/inp?hl=ja

イルウェブサイトの55%が良好なINPを持っていることがわかります。これは、とくにモバイルウェブがINPでFIDに比べてどれだけ悪いパフォーマンスをしているかを考えると興味深いです。INPがFIDをCore Web Vitalとして置き換える場合、レスポンシブ性ははるかに顕著な問題になります。

## 結論

2020年と2021年のデータと2022年のデータを比較すると、モバイルWebの使用と期待に多くの進化が見られます。レイアウトの安定性のようなパフォーマンスの側面は、以前はオプションであったり、過度に技術的であったり、ニッチであったりしたものが現在ではモバイルとデスクトップの両方に焦点を当てるサイトオーナーにとって主流の懸念事項となっています。優れたユーザー体験を提供するための複雑さがモバイルデバイスにおいてより顕著になり、期待されるものが複雑になる一方で、達成がより困難になっています。

モバイルWebとそのユーザーエクスペリエンスを気にかける人々にとっての良いニュースは、異なるデバイス間でうまく機能するサイトを構築するための多くの困難な作業が、個々のサイトやCMS開発チームからブラウザにオフロードされつつあることです。これはレイジーローディング、レスポンシブ画像用の `srcset` 要素、およびパフォーマンスの他の側面で起こっていますが、このタイプの取り組みが他の方法でもブラウザによって支援されることを期待しています。モバイルWebのアクセシビリティを改善する機会はまだ多く残っており、異なるタイプの電話機能間のインタラクティビティや、将来的にはキャスティング、共有、その他の接続された家庭やIoTシナリオとの相互作用も考えられます。Googleや他の検索エンジンも、Webに広範な肯定的な影響を与えるために、モバイルアクセシビリティの問題に対する注目を明示的に奨励できます。

このレポートのトラフィックと人気の統計は、モバイルWebが基本的には単に *Web* と同義であることを明確にしています。ほとんどのシナリオでモバイルウェブサイトのインタラクションの期待はデフォルトであるべきです。これは、私たちが協力する技術的でないチーム、部門、グループにこの現実を意識させ続ける必要があることを意味します。モバイルファーストの設計と開発の概念に口先だけで同意するだけでは不十分です。これらの概念は引き続き受け入れられ、必要に応じて推進される必要があります。また、ビジネスプランニング、マーケティング、戦略、コミュニケーションのようなWebシナリオ外のビジネスの大きな要素においても成長を経験する必要があります。

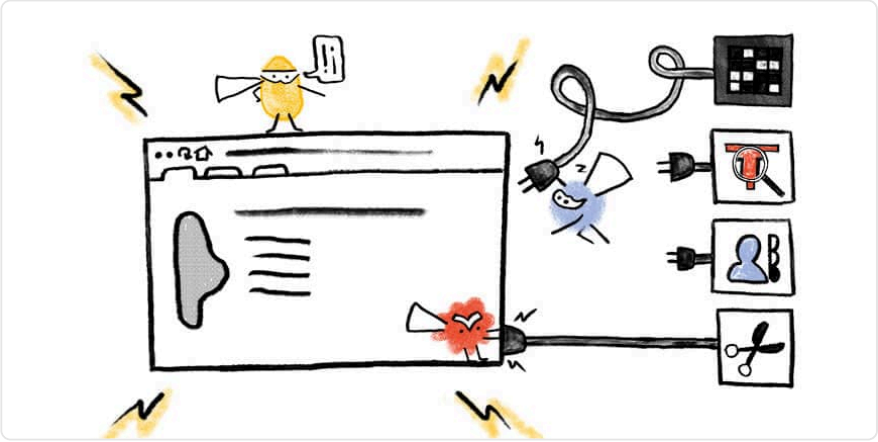
## 著者


### Cindy Krum

@suzzicks Suzzicks https://mobilemoxie.com/

Cindy KrumはMobileMoxieのCEO兼創設者です。彼女はモバイルSEO、アプリSEO（ASO）、そしてGoogleの発表前に変更を予測し説明することを専門としています。

# 部 II 章 16
# ケイパビリティ



***Michael Solati** によって書かれた。*
*Thomas Steiner と Christian Liebel によってレビュ。*
*Barry Pollard による分析 と 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

魅力的なウェブ体験は、ブラウザの基本的な機能に限定されるものではありません。基盤となるオペレーティングシステムの機能にアクセスできます。ウェブプラットフォームAPIは、これらの機能を公開し、Progressive Web Apps (PWA)の基礎となります。PWAは、プラットフォーム固有のアプリケーションと同じ高品質の体験を提供できるウェブアプリケーションです。

さらに、ウェブプラットフォーム上のいくつかの機能は、ファイルシステム[597]、位置情報[598]、クリップボード[599]へのアクセス、さらにゲームパッド[600]の検出など、低レベルの機能にアクセスできます。

597. https://developer.mozilla.org/ja/docs/Web/API/File_System_API
598. https://developer.mozilla.org/ja/docs/Web/API/Geolocation_API
599. https://developer.mozilla.org/ja/docs/Web/API/Clipboard_API
600. https://developer.mozilla.org/ja/docs/Web/API/Gamepad_API

## 方法論

この章では、HTTP Archiveの数百万ページにわたる公開データセットを使用しました。これらのページは、デスクトップとモバイルの両方で訪問されたかのようにアーカイブされており、デバイスによって異なるコンテンツを提供するサイトもあります。

HTTP Archiveのクローラーは、これらのページのソースコードを解析し、どのAPIが（潜在的に）使用されているかを特定しました。例えば、正規表現 `/navigator\.share\s*\(/g` を使用して、具体的なケースで Web Share API[601] がソースコードに含まれているかどうかをテストします。

この方法には2つの大きな問題があります。第一に、ミニファイなどにより、`navigator` が `n` に縮小された場合など、使用されている一部のAPIを検出できないため、報告が少なくなることがあります。さらに、APIが実際に使用されているかどうかを確認するためにコードを実行しないため、APIの出現を過報告することがあります。これらの限界にもかかわらず、この方法はウェブ上で使用されている機能の良い概要を提供すると考えています。

サポートされている機能のための合計75個の正規表現があります。使用されたすべての表現を見るには、こちらのソースファイル[602]を参照してください。

この章の使用データは、2022年6月のクロールからのものです。生データは、Capabilities 2022 Results Sheet[603]で見ることができます。

また、この章では、昨年のAPI使用状況と比較します。前年の生データは、Capabilities 2021 Results Sheet[604]で確認できます。

## 非同期クリップボードAPI

最初のAPIである非同期クリップボードAPI[605]は、システムのクリップボードへの読み書きアクセスを可能にします。

非同期クリップボードAPIは、クリップボードにアクセスするための廃止された `document.execCommand()` APIに取って代わります。

### 書き込みアクセス

クリップボードにデータを書き込むためには、`writeText()` メソッドと `write()` メソッ

601. https://developer.mozilla.org/ja/docs/Web/API/Web_Share_API
602. https://github.com/HTTPArchive/custom-metrics/blob/5d2f74fbdc580e76da5d1dad738fca8381429b9a/dist/fugu-apis.js
603. https://docs.google.com/spreadsheets/d/13S9FRj8OPRtoMPb94jFh6pPNz3lNS9yztIaorZYe288/edit?usp=sharing
604. https://docs.google.com/spreadsheets/d/1b4moteB9EiLYkH1Ln9qfi1tnU-E4N2UQ87uayWytDKw/edit#gid=2077755325
605. https://developer.mozilla.org/ja/docs/Web/API/Clipboard_API

ドがあります。`writeText()`メソッドはString引数を取り、Promiseを返します。一方、`write()`メソッドは`ClipboardItem`オブジェクトの配列を取り、同様にPromiseを返します。`ClipboardItem`オブジェクトは、画像などの任意のデータを保持できます。

ブラウザがClipboards API仕様によってサポートする必要があるデータタイプのリストがあります。このW3Cのリスト[606]を参照してください。残念ながら、すべてのベンダーが完全なリストをサポートしているわけではありません。可能な場合はブラウザ固有のドキュメントを確認してください。

```
await navigator.clipboard.writeText("hello world");

const blob = new Blob(["hello world"], { type: "text/plain" });
await navigator.clipboard.write([
  new ClipboardItem({
    [blob.type]: blob,
  }),
]);
```

### 読み取りアクセス

クリップボードからデータを読み取るためには、`readText()`メソッドと`read()`メソッドがあります。これらの方法はいずれもPromiseを返し、クリップボードからのデータで解決されます。`readText()`メソッドはStringとして解決され、`read()`メソッドは`ClipboardItem`オブジェクトの配列として解決されます。

```
const item = await navigator.clipboard.readText();
const items = await navigator.clipboard.read();
```

ユーザーデータを安全に保つためには、クリップボードからデータを読み取るためにPermissions API[607]の`"clipboard-read"`権限が付与されていなければなりません。

クリップボードへの読み書きアクセスは、Chrome、Edge、Safariの現代のバージョンで利用可能です。Firefoxは`writeText()`のみをサポートしています。

606. https://www.w3.org/TR/clipboard-apis/#mandatory-data-types-x
607. https://developer.mozilla.org/ja/docs/Web/API/Permissions_API

### 非同期クリップボードAPIの成長

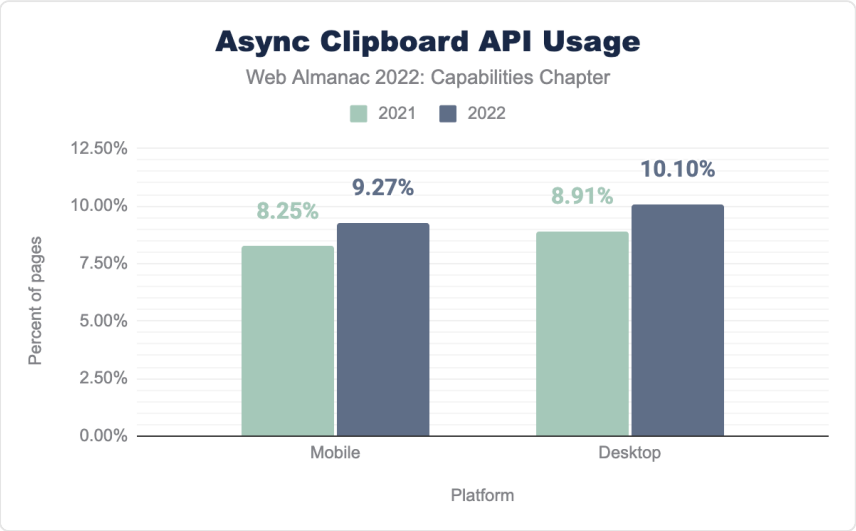



*図16.1. 2021年から2022年にかけてのデスクトップとモバイルでの非同期クリップボードAPIの使用状況。*

非同期クリップボードAPIは、デスクトップで2021年の8.91%から2022年には10.10%へと、またモバイルでは2021年の8.25%から2022年には9.27%へと使用率が増加しました。その結果、今年の非同期クリップボードAPIは、デスクトップとモバイルの両方で最も多く使用されたAPIとなり、昨年最も使用されたWeb Share APIを上回りました。

## Web Share API

Web Share API[608]は、デバイスのプラットフォーム固有の共有メカニズムを呼び出し、テキスト、URL、またはWebアプリケーションからのファイルなどのデータをメールクライアント、メッセージングアプリケーションなどの他のアプリケーションと共有できます。

データを共有するために呼び出されるメソッドは`navigator.share()`です。`navigator.share()`メソッドは共有するデータを含むオブジェクトを受け取り、Promiseを返します。ただし、共有できるファイルタイプは限られており、`navigator.canShare()`メソッドは、ブラウザがデータオブジェクトを共有できるかどうかをテストできます。MDNで共有可能なファイルタイプのリスト[609]を見ることができます。

608. https://developer.mozilla.org/ja/docs/Web/API/Web_Share_API
609. https://developer.mozilla.org/ja/docs/Web/API/Navigator/share#%E5%85%B1%E6%9C%89%E5%8F%AF%E8%83%BD%E3%81%AA%E3%83%95%E3%82%A1%E3%82%A4%E3%83%AB%E5%9E%8B

`navigator.share()` を呼び出した後、ブラウザはプラットフォーム固有のシートを開き、ユーザーがデータを共有するアプリケーションを選択します。

さらに、Web Share APIはユーザーのページとのインタラクション、例えばボタンのクリックなどによってのみトリガーでき、実行されたコードによって任意に呼び出すことはできません。

```
const data = {
  url: "https://almanac.httparchive.org/en/2022/capabilities",
};

if (navigator.canShare(data)) {
  try {
    await navigator.share(data);
  catch (err) {
    console.error(err.name, err.message);
  }
}
```

Web Share APIは、現代のバージョンのChrome、Edge、およびSafariで利用可能です。ただし、ChromeではWindowsとChromeOSでのみサポートされています。

### Web Share APIの成長

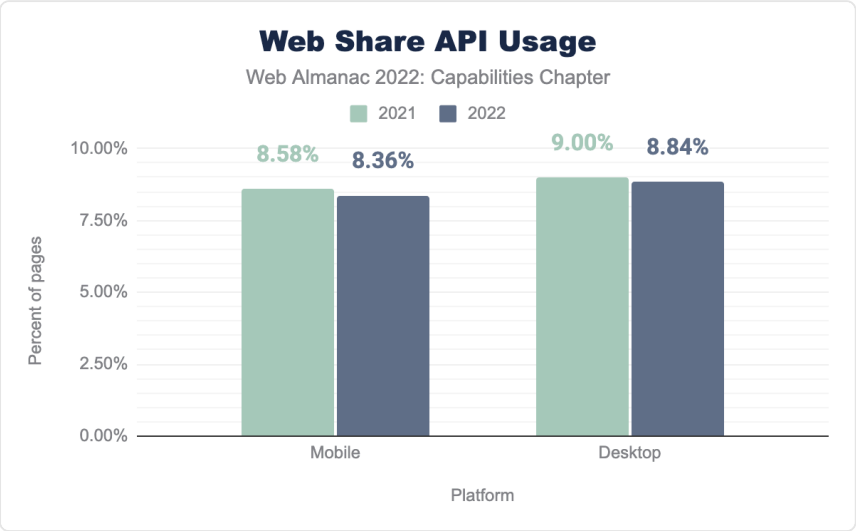



*図16.2. 2021年から2022年にかけてのデスクトップとモバイルでのWeb Share APIの使用状況。*

Web Share APIは、デスクトップで2021年の9.00%から2022年には8.84%へ、モバイルでは2021年の8.58%から2022年には8.36%へと使用率が減少しました。その結果、今年、Web Share APIはデスクトップとモバイルの両方で二番目に多く使用されたAPIとなり、昨年の二番目に多く使用された非同期クリップボードAPIに次ぐものとなりました。

多くのサイトでWeb Share APIが使用されています。例えば、ソーシャルメディアプラットフォーム、ドキュメントサイトなどがコンテンツを共有するための素晴らしい方法として使用しています。APIが使用されている例としては、web.dev[610]やtwitter.com[611]があります。

*Web Share APIを使用してTwitterプロフィールを共有する。*

*図16.3. Web Share APIを使用してTwitterプロフィールを共有する。*

## ホームスクリーンへの追加

昨年のCapabilitiesレポートでは取り上げなかった機能の一つに、デバイスのホームスクリーンにWebアプリケーションを追加する機能があります。どのサイトがこの機能を持っている

610. https://web.dev/
611. https://twitter.com/

かを計算するために、`beforeinstallprompt`イベントのリスナーがあるかどうかをテストしました。

`beforeinstallprompt`イベントはChromium専用のAPIであり、現在WICG内でインキュベート中[612]です。

`beforeinstallprompt`イベントは、ユーザーが「インストール」する前にトリガーされます。`beforeinstallprompt`イベントのイベントリスナーを使用することは、Webアプリをデバイスのホームスクリーンに追加するために必要ではありませんので、実際の使用率はもっと高いと考えられます。しかし、この方法論により、どれほど人気のある機能かを把握することができます。

ホームスクリーンにアプリケーションを追加する機能は、PWAの重要な特徴です。この機能を使用するには、Webアプリケーションが以下の基準[613]を満たしている必要があります：

- Webアプリは既にインストールされていないこと。
- ユーザーはページを閲覧してから少なくとも30秒経過していること。
- ユーザーはページで少なくとも一度クリックまたはタップしていること。
- WebアプリはHTTPS経由で提供されること。
- WebアプリにはWebアプリマニフェスト[614]が含まれていること：
    - `short_name`または`name`。
    - `icons`（192×192pxおよび512×512pxのアイコンを含む必要があります）。
    - `start_url`。
    - `display`（`fullscreen`、`standalone`、または`minimal-ui`のいずれかである必要があります）。
    - `prefer_related_applications`（存在しないか、`false`である必要があります）。
- Webアプリは`fetch`ハンドラーを持つサービスワーカーを登録していること。

インストールされたアプリケーションは、スタートメニュー、デスクトップ、ホームスクリーン、アプリケーションフォルダー、デバイス上でのアプリケーションの検索、コンテンツ共有シートなどに表示されることがあります。

612. *https://wicg.github.io/manifest-incubations/index.html#installation-prompts*
613. *https://web.dev/install-criteria/#criteria*
614. *https://developer.mozilla.org/ja/docs/Web/Manifest*

ホームスクリーンに追加する機能は、iOSおよびiPadOSのChrome、Edge、Safariの現代のバージョンでのみ利用可能です。

### ホームスクリーンへの追加の使用状況

**7.71%**

*図16.4. モバイルでのホームスクリーンへの追加の使用率。*

前述の通り、昨年はホームスクリーンへの追加機能は測定されていませんでした。しかし、後世と詳細な報告のために、`beforeinstallprompt`イベントはデスクトップページの8.56%、モバイルページの7.71%で使用されており、デスクトップとモバイルの両方で三番目に多く使用される機能となっています。

`beforeinstallprompt`イベントを活用することで、開発者はユーザーがWebアプリケーションをインストールする方法にカスタマイズされた体験を提供することができます。例としては、YouTube TVがそのアプリケーションをインストールすることで、より迅速かつ簡単にアクセスできるようにユーザーを招待するケースがあります。

*beforeinstallpromptイベントによって提供されるアプリ内プロンプトからYouTube TVをインストールする。*

*図16.5. `beforeinstallprompt`イベントによって提供されるアプリ内プロンプトからYouTube TVをインストールする。*

## メディアセッションAPI

メディアセッションAPI[615]を使用すると、開発者はWeb上のオーディオまたはビデオコンテンツのカスタムメディア通知を作成することができます。このAPIには、ブラウザがキーボード、ヘッドセット、およびデバイスの通知エリアやロック画面のソフトウェアコントロールにアクセスするために使用できるアクションハンドラが含まれています。メディアセッションAPIは、ユーザーがアクティブにページを閲覧していなくても、Webページで何が再生されているかを知り、制御することを可能にします。

ページがオーディオまたはビデオコンテンツを再生すると、ユーザーはモバイルデバイスの通知トレイまたはデスクトップのメディアハブにメディア通知が表示されます。ブラウザはタイトルとアイコンを表示しようとしますが、メディアセッションAPIを使用すると、タイ

615. https://developer.mozilla.org/ja/docs/Web/API/Media_Session_API

トル、アーティスト名、アルバム名、アルバムアートワークなどの豊富なメディアメタデータを持つカスタマイズされた通知を提供することができます。

```
navigator.mediaSession.metadata = new MediaMetadata({
  title: "Creep",
  artist: "Radiohead",
  album: "Pablo Honey",
  artwork: [
    {
      src: "https://via.placeholder.com/96",
      sizes: "96x96",
      type: "image/png",
    },
    {
      src: "https://via.placeholder.com/128",
      sizes: "128x128",
      type: "image/png",
    },
    {
      src: "https://via.placeholder.com/192",
      sizes: "192x192",
      type: "image/png",
    },
    {
      src: "https://via.placeholder.com/256",
      sizes: "256x256",
      type: "image/png",
    },
    {
      src: "https://via.placeholder.com/384",
      sizes: "384x384",
      type: "image/png",
    },
    {
      src: "https://via.placeholder.com/512",
```

```
        sizes: "512x512",
        type: "image/png",
      },
    ],
  });
```

メディアセッションAPIは、現代のバージョンのChrome、Edge、Firefox、およびSafariで利用可能です。

### メディアセッションAPIの使用状況


*図16.6. モバイルでのメディアセッションAPIの使用率。*

メディアセッションAPIは昨年測定されていませんでした。追跡の初年度において、このAPIはデスクトップページの8.37%、モバイルページの7.41%で使用され、デスクトップとモバイルの両方で四番目に多く使用される機能となりました。

YouTube、YouTube Music、SpotifyなどのWebアプリケーションはメディアセッションを活用し、再生されるビデオやオーディオのための豊富なコントロールを提供しています。

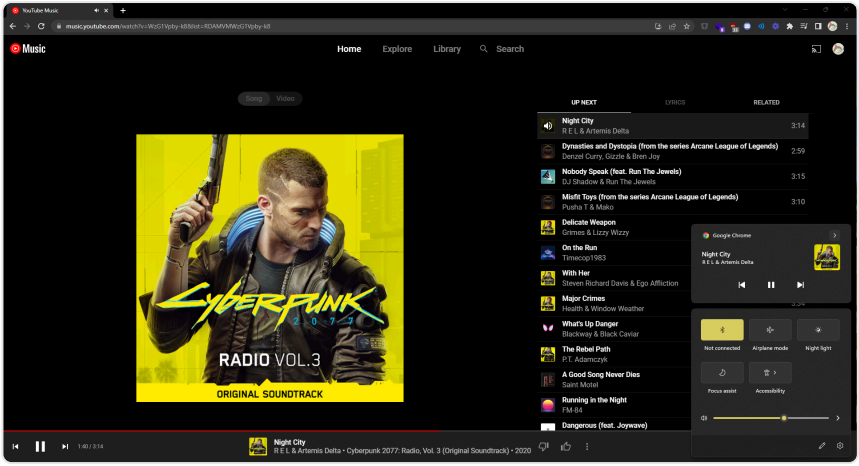



*図16.7. Windowsのタスクバーを通じてYouTube Musicのコントロールと情報にアクセスする。*

Web上のビデオ使用についてさらに詳しく知りたい場合は、メディア章をご覧ください。

## デバイスメモリAPI

デバイスの性能は、ネットワーク、CPUコア数、利用可能なメモリ量など、いくつかの要素に依存します。デバイスメモリAPI[616]は、`Navigator`インターフェース上の読み取り専用プロパティ`deviceMemory`を提供することで、利用可能なメモリについての情報を提供します。このプロパティは、浮動小数点数としてデバイスのメモリ量をギガバイト単位でおおよそ返します。

返される値は、ユーザーのプライバシーを保護するために不正確です。値は2のべき乗に切り捨てられた後、その数を1,024で割って計算されます。また、数値は上下限の範囲内で調整されています。したがって、`0.25`、`0.5`、`1`、`2`、`4`、`8`（ギガバイト）の数値が予想されます。

```
const memory = navigator.deviceMemory;
console.log('This device has at least ', memory, 'GiB of RAM.');
```

デバイスメモリAPIは、現代のバージョンのChromeとEdgeでのみ利用可能です。

616. *https://developer.mozilla.org/ja/docs/Web/API/Device_Memory_API*

### デバイスメモリAPIの使用状況


*図16.8. モバイルでのデバイスメモリAPIの使用率。*

デバイスメモリAPIは昨年は測定されていませんでした。追跡の初年度において、このAPIはデスクトップページの6.27%、モバイルページの5.76%で使用され、デスクトップとモバイルの両方で五番目に多く使用される機能となりました。

2019年のFacebookのリデザインリリース、FB5では、実際のハードウェアに基づいてロード内容を調整することで、アダプティブローディングを積極的に統合しました。例えば、デスクトップでは、利用可能なCPUコア（`navigator.hardwareConcurrency`）とデバイスメモリ（`navigator.deviceMemory`）に基づいてユーザーのバケットを定義しました。

Chrome Dev Summit 2019のこのビデオ[617]を24:03からご覧ください。Nate Schlossがデバイスメモリなどの機能を使用してFacebookがアダプティブローディングをどのように取り扱っているかを共有しています。

## サービスワーカーAPI

サービスワーカー[618]はプログレッシブウェブアプリの核心的なコンポーネントの一つです。サービスワーカーはクライアント側のプロキシとして機能し、開発者がシステムのキャッシュやリソース要求への応答方法を制御できるようにします。重要なリソースを事前にキャッシュすることで、ネットワークへの依存を排除し、即時かつ信頼性の高い体験を保証できます。

リソースをキャッシュすることに加えて、サービスワーカーはサーバーからアセットを更新したり、プッシュ通知を可能にしたり、バックグラウンド及び周期的なバックグラウンド同期APIへのアクセスを許可することができます。

サービスワーカーは主要なブラウザに広く採用されていますが、サービスワーカーの全ての機能がすべてのブラウザで利用可能なわけではありません。現在サポートされていない機能の一例としては、SafariのPush APIがあります。Safariは2022年のmacOS Ventura[619]および2023年のiOS 16[620]とiPadOS 16でPush APIをサポートする予定です。

617. *https://www.youtube.com/watch?v=puUPpVrIRkc&t=1443s*
618. *https://developer.mozilla.org/ja/docs/Web/API/Service_Worker_API*
619. *https://www.apple.com/macos/macos-ventura-preview/features/*
620. *https://www.apple.com/ios/ios-16/features/*

サービスワーカーAPIは、現代のバージョンのChrome、Edge、Firefox、およびSafariで利用可能です。

### サービスワーカーAPIの成長

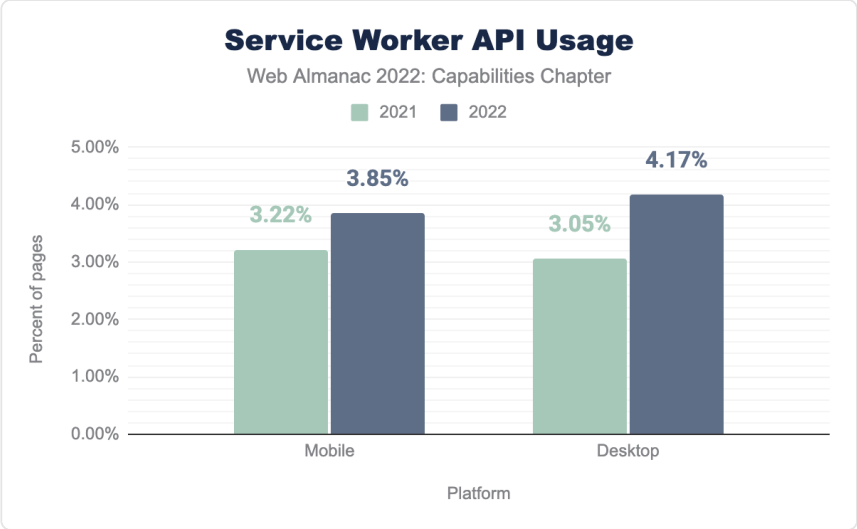



*図16.9. 2021年から2022年にかけてのデスクトップとモバイルでのサービスワーカーAPIの使用状況。*

昨年のCapabilities章ではサービスワーカーAPIは測定されていませんでしたが、昨年のPWA章のデータを使用して、APIの使用はデスクトップで3.05%から4.17%へ、モバイルページで3.22%から3.85%へと増加し、デスクトップで六番目に、モバイルで七番目に多く使用される機能となりました。

PWA章でのサービスワーカーの使用測定方法はCapabilities章での測定方法と異なります。また、昨年のPWA章のデータパイプラインにバグが見つかり、サービスワーカーの使用が少なく見積もられていたことがあります。

Web上でのサービスワーカーの使用についてさらに深く掘り下げるには、2022年のWeb AlmanacのPWA章をご覧ください。

## ゲームパッドAPI

*ゲームパッドAPI*は、Webアプリケーションがゲームパッドやその他のゲームコントローラ

ーからの入力にどのように対応するかを定義しています。このAPIには3つのインターフェースがあります。1つはデバイスに接続されたコントローラーを表し、もう1つは接続されたコントローラーのボタンを表し、最後にゲームパッドが接続または切断されたときに発火されるイベント用のものです。

```
window.addEventListener("gamepadconnected", (e) => {
  const gp = navigator.getGamepads()[e.gamepad.index];
  console.log(`Controller connected at index ${gp.index}`);
});
```

ゲームパッドAPIは、現代のバージョンのChrome、Edge、Firefox、およびSafariで利用可能です。

### ゲームパッドAPIの成長

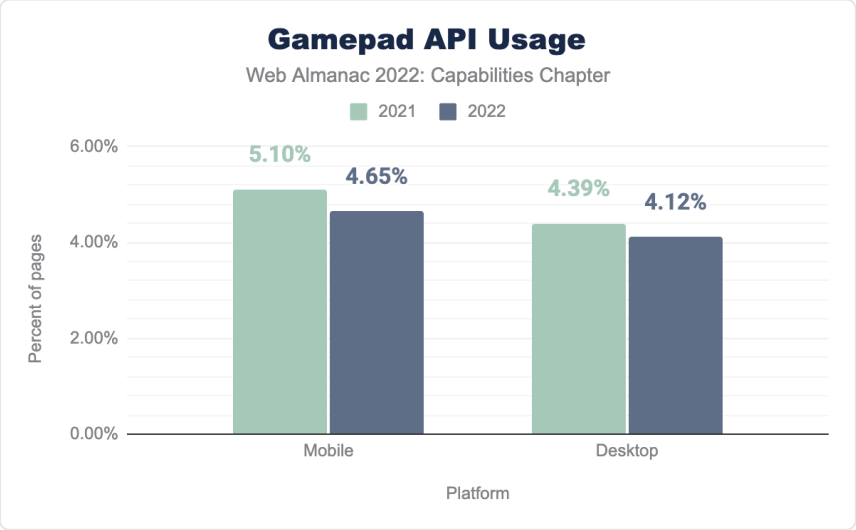



*図16.10. 2021年から2022年にかけてのデスクトップとモバイルでのゲームパッドAPIの使用状況。*

ゲームパッドAPIの使用は、デスクトップで2021年の4.39%から2022年には4.12%へと減少しました。モバイルでは、2021年の5.10%から2022年には4.65%へと減少しました。その結果、今年ゲームパッドAPIはデスクトップで七番目に、モバイルで六番目に多く使用される機能となりました。

GoogleのStadia、NVIDIAのGeForce Now、MicrosoftのXbox Cloud GamingなどのWebアプリケーションは、クラウド上でローカルデバイスやゲームコンソールでゲームを実行する体験と同等のゲーム体験を提供します。ゲームパッドAPIのおかげで、これらのWebアプリケーションはユーザーにキーボードとマウスだけでなく、従来のコンソールゲームコントローラーを使用することを可能にします。

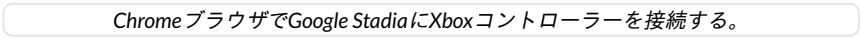



*図16.11. Chromeブラウザで Google Stadia に Xbox コントローラーを接続する。*

## プッシュAPI

*プッシュAPI* を使用すると、アプリケーションがフォアグラウンドにない場合でも、Webアプリケーションがサーバーからメッセージを受信できます。開発者は非同期通知や更新をユーザーに送信でき、意味のある更新を提供し、アプリケーションとの再エンゲージメントを促すことができます。

プッシュ通知をサーバーから受信するためには、Webアプリケーションにサービスワーカーが必要です。サービスワーカー内から、`PushManager.subscribe()` メソッドを使用してプッシュ通知を購読することができます。

プッシュAPIは、現代のバージョンのChrome、Edge、およびFirefoxで利用可能です。

### プッシュAPIの使用状況


*図16.12. モバイルでのプッシュAPIの使用率。*

プッシュAPIは昨年測定されていませんでした。追跡の初年度において、このAPIはデスクトップページの2.03%、モバイルページの1.86%で使用され、デスクトップとモバイルの両方で八番目に多く使用される機能となりました。

## プロジェクトフグ

多くのユーザーがプラットフォーム固有のアプリケーションに属していると期待する機能は、Web上にも存在します。しかし、Capabilities Project（多くの人々にプロジェクトフグ

として知られている）のおかげで、これらの機能はWeb上に存在しています。プロジェクトフグは、iOS、Android、またはデスクトップアプリができることを考慮して、Webアプリケーションに機能の同等性をもたらすための企業間の取り組みです。プロジェクトフグは、ユーザーのセキュリティ、プライバシー、信頼、およびWebの他の核心的な原則を維持しながら、プラットフォーム固有の機能をWebに公開する作業に取り組んでいます。

プロジェクトフグには、Microsoft、Intel、Samsung、Googleなど多くのグループや個人が参加しています。

Capabilities Projectについてもっと学ぶために、Chrome Developersブログのこの投稿[621]をチェックしてください。

## 結論

Capabilitiesは、Web上で開発者が活用できる新しい可能性と機能を解き放ちます。この章では、現在Web上で使用されている最も人気のあるWebプラットフォームAPIの8つを共有しました。また、異なるWebアプリケーションで使用されているいくつかの機能も紹介しました。Webの美しさは、デバイスにインストールする必要がなく、追加のライブラリやプラグインも必要ないということです。

Webの機能を活用するエキサイティングな体験には、What Web Can Do Today?[622]（WWCDT）やDiscourse[623]があります。WWCDTは、追跡している機能の38を使用し、各APIのライブデモを示して多くのWeb APIを紹介しています。Discourseは、コミュニティにWebフォーラムを提供し、追跡している機能の14を使用しています。たとえば、Badging APIを使用すると、ユーザーは未読通知の数を確認できます。

プロジェクトフグ、Capabilities Projectにより、アプリケーションはWebに移行し、プラットフォーム固有のアプリケーションに関連するいくつかの障壁を取り除くことができます。ネイティブコードを書く必要がなく、ユーザーが最新の更新にアクセスしているかを心配する必要がなく、ユーザーにアプリストアで検索してダウンロードするよう促す必要もありません。Webとその機能は、ユーザーに魅力的な体験を提供するための新たな可能性を開きます。

621. *https://developer.chrome.com/blog/fugu-status*
622. *https://whatwebcando.today/*
623. *https://www.discourse.org/*

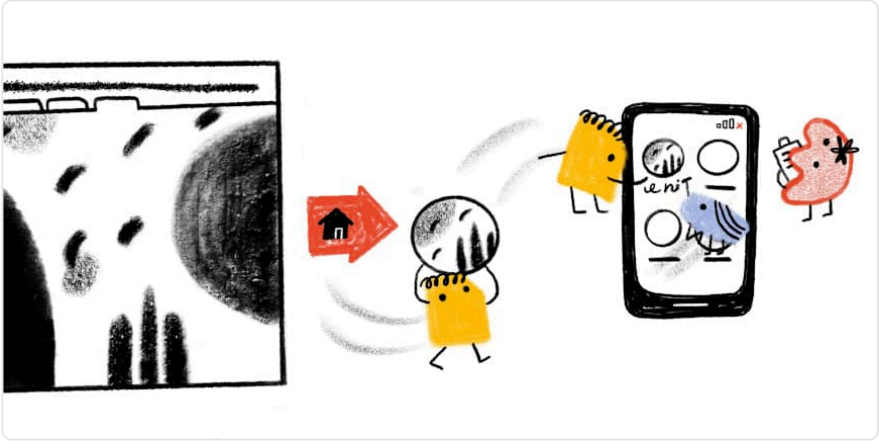
## 著者


### Michael Solati

@MichaelSolati MichaelSolati https://michaelsolati.com

MichaelはAmplicationのDeveloper Advocateで、開発者がAPIを構築し、IPAを飲むのを支援することに専念しています。さらに、彼はWeb GDEであり、ウェブ上で魅力的な体験を創造することとウェブの魔法のような方法についての愛を見つけました。

# 部 II 章 17
# PWA



*Diego Gonzalez によって書かれた。*
*Aaron Gustafson、 Adriana Jara、 Maxim Salnikov、 Kai Hollberg と Beth Pan によってレビュ。*
*Beth Pan による分析。*
*Siwin Lo と Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

プログレッシブウェブアプリ（PWA）の初期段階において、現代の高度なウェブアプリケーションの可能性を活用した2つの重要な機能がありました：オフラインサポートとデバイスのホーム画面に直接アイコンを表示すること。

これらの概念は、PWAをインストールした後に有効になり、通常はブラウザのURLバーに表示される「アンビエントバッジ」をタップすることでインストールプロセスが始まります。このバッジはユーザーにウェブサイトをインストールするよう促します。Samsung InternetやMozilla Firefoxなどのモバイルブラウザは、この振る舞いを明確にサポートした最初のブラウザの一部であり、一般的には“ホーム画面に追加”[624]として知られています。

5年前、これは画期的なアイデアでした。ウェブサイトがホーム画面から直接起動し、ユーザーがデバイスにインストールした他のアプリケーションと並んでリストされるようになり

624. https://developer.mozilla.org/ja/docs/Web/Progressive_web_apps/Guides/Making_PWAs_installable

ました。これは、ウェブアプリとOS固有の経験の間のギャップを縮小するための進歩の始まりでした。

A2HSシナリオは、モバイルおよびデスクトップの両方のコンテキストでホストOSに完全にインストールされ、深く統合されるウェブアプリへと進化しました。過去12か月間には、ブラウザがデスクトッププラットフォームとの緊密な統合を確実にするための重要なステップを踏み出しており、今年のアルマナックに追加された多くの新機能がこれらの変更を反映しています。これが2022年のPWAの状況です。

注：PWAはウェブプラットフォームの他の部分と孤立しているわけではありません。Capabilitiesに専用の章がありますが、今年はPWA内で使用されるこれらの新しい高度な機能の交差点を調査しました。

## サービスワーカー

サービスワーカー[625]はPWAの中核技術の1つであり、オフラインアプリ、プッシュ通知の受信、バックグラウンド処理の実現者です。これらは、アプリケーションから期待されるほとんどの高度な体験の基盤となっています。また、データの更新を定義するためや、PWA技術に基づくウィジェット[626]などの近代的な機能のためにも使用されています。

主要なブラウザ間でサービスワーカーの機能サポートに一貫性はありませんが、Webkitがプッシュ通知[627]のサポートを追加したことは大きな節目でした。今年初めに、Appleがデスクトッププラットフォーム でPush API[628]、Notifications API[629]の関連部分をサポートし、サービスワーカー[630]がWebプッシュを有効にする変更を行ったと発表されました。また、この機能が2023年にモバイルプラットフォームにも導入される予定です。

### サービスワーカーの使用

比較のために、昨年と同じクエリを実行して、サービスワーカーの使用の進化を理解しようとしました。昨年の章では、サービスワーカーの実際の使用状況を見つけることが簡単ではない理由[631]を説明しましたが、今年も同様です。

2つの指標を見てみましょう：

- Lighthouseは、すべてのウェブサイトの1.6%（モバイル）および1.7%（デスクトップ）がサービスワーカーを使用していることを検出しています。これは、

625. *https://developer.mozilla.org/ja/docs/Web/API/Service_Worker_API*
626. *https://github.com/aarongustafson/pwa-widgets#rich-widgets*
627. *https://caniuse.com/push-api*
628. *https://developer.mozilla.org/ja/docs/Web/API/Push_API*
629. *https://developer.mozilla.org/ja/docs/Web/API/Notifications_API*
630. *https://developer.mozilla.org/ja/docs/Web/API/Service_Worker_API*
631. *https://almanac.httparchive.org/ja/2021/pwa#サービスワーカーの利用状況*

Lighthouseが考慮する追加チェック[632]のため、実際の世界の割合よりも低いと予想されます。

- 昨年導入された同じメトリックス[633]にしたがって、ウェブサイトでのサービスワーカーの使用率はデスクトップで1.63%、モバイルで1.81%です。

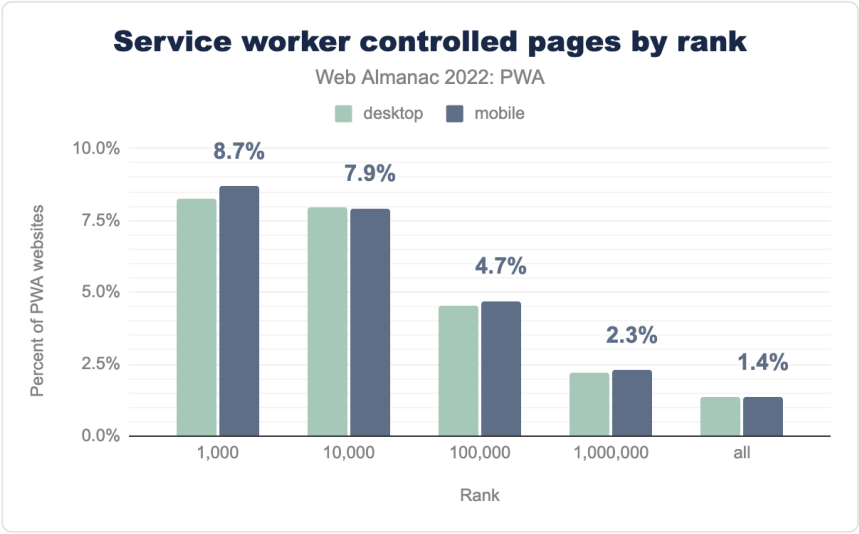



*図17.1. ランク別のサービスワーカー制御ページ。*

上位1,000のサイトにおけるサービスワーカー制御ページに顕著な変化はありませんでした。デスクトップではわずかな減少、モバイルではさらに小さな増加がありましたが、他のすべてのカテゴリでは増加が見られました。昨年の推論[634]に従えば、大きなウェブサイトが先に高度な技術を採用するという仮説、他のカテゴリでの成長を見るのは理にかなっています。より多くのトラフィックを持つ企業からの事例研究や例を共有することで、小さな企業や開発者が技術を学び、採用したと思われます。

## サービスワーカーのイベント

サービスワーカーは、ウェブアプリ、ブラウザ、ネットワークの間に位置するプロキシサーバーとして機能します。サービスワーカーをインストールするには、フェッチして登録する必要があります。これが成功すると、サービスワーカーはメインスレッドから切り離され、DOMへのアクセスがない特別なワーカーコンテナ[635]で実行されます。この時点でイベントが

632. *https://web.dev/service-worker*
633. *https://github.com/HTTPArchive/legacy.httparchive.org/blob/master/custom_metrics/pwa.js*
634. *https://almanac.httparchive.org/ja/2021/pwa#fig-2*
635. *https://developer.mozilla.org/ja/docs/Web/API/ServiceWorkerGlobalScope*

処理されます。

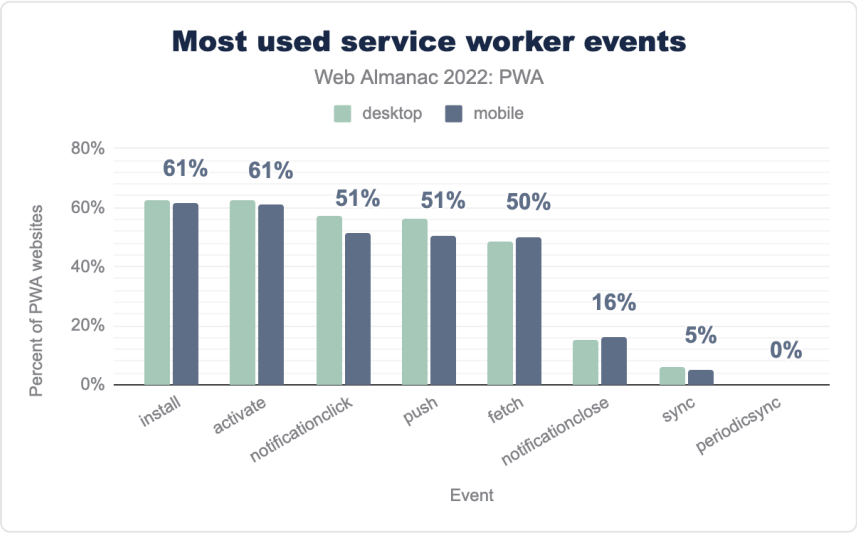



*図17.2. もっとも使用されているサービスワーカーのイベント。*

前述のグラフは、もっとも使用されているサービスワーカーのイベントに関する情報を示しています。これらのイベントは以下のカテゴリに分類されます：

- ライフサイクルイベント
- 通知関連イベント
- バックグラウンド処理イベント

### ライフサイクルイベント

`install` と `activate` はライフサイクルイベントです。インストール時にオフラインでアプリを実行するためのアセットのキャッシュを作成するのが一般的な方法です。アクティベーションは通常、以前のサービスワーカーに関連付けられた古いキャッシュをクリーンアップするために使用されます。


*図17.3. モバイルにおける `install` イベントのサービスワーカーイベント。*

サービスワーカーは、fetchやpushなどのイベントを受け取るためにアクティブである必要があります。これにより、`install`のデスクトップでの使用率63%、モバイルでの使用率61%、そして`activate`も同様に、サービスワーカーの要となります。

サービスワーカーを持つ残りの約40%のサイトは、これらのイベントを積極的に使用していないかもしれません。これは、通知のためにサービスワーカーを使用しているか、またはサイトが稼働しているときにのみリクエストのキャッシュを行う技術、いわゆるランタイムキャッシング[636]を利用している可能性があります。

これらが依然としてもっとも使用されているイベントである一方で、他のタイプのイベントの使用増加は、サービスワーカーが事前キャッシュのためだけに（のみで）使用されているわけではない、という仮説を導くものです。

### 通知関連イベント


*図17.4. デスクトップにおける`notificationclick`イベントのサービスワーカーイベント。*

プッシュ通知イベントは、もっとも使用されているサービスワーカーメソッドの次に来ます。

- `notificationclick`はデスクトップで57%（昨年のデータから▲11%）、モバイルで51%（▲5%）です。
- `push`はデスクトップで56%（▲12%）、モバイルで50%（▲5%）です。
- `notificationclose`は現在、デスクトップで15%（▲9%）、モバイルで16%（▲10%）です。

ここでのいくつかのポイントは、デスクトップでのPWAの勢いが今年も続いており、プッシュ通知も例外ではないということです。通知に関連するイベントの使用は約11%増加しています。さまざまなプラットフォームで多くの調整と修正が行われ、これらのUXがホストOSと完全に統合されるようになっています。新たに発表されたWebkitにおけるWebプッシュのサポート[637]にしたがって、これらの数字が増加し続けることが期待されます。これは多くの開発者から長い間要望されていた機能であり、ついにmacOSでサポートされ、そして間もなくiOSデバイスでもサポートされることが開発者にAPIの使用を促すかもしれません。

636. https://web.dev/runtime-caching-with-workbox/
637. https://webkit.org/blog/12945/meet-web-push/

### バックグラウンド処理イベント


*図17.5. デスクトップにおける `fetch` イベントのサービスワーカーイベント。*

チャートの残りのイベントはバックグラウンド処理イベントを表しています：

- `fetch` は、リクエストがサーバーに送信されると発生し、該当のリクエストを傍受してカスタムまたはキャッシュされたアセットで応答することができ、PWAのオフラインサポートを可能にします。Fetchの使用率はデスクトップで49%、モバイルで50%です。
- `sync` は、UAがユーザーが接続を持っていると考えるときに発火し、デスクトップで6%、モバイルで5%の使用率です。
- `periodicsync` は、ウェブアプリケーションが定期的にバックグラウンドでデータを同期することを可能にし、現在、デスクトップとモバイルプラットフォームの両方で0.01%です。 `periodicsync` は最大12時間に1回と制限されています。この制限が機能の使用を人為的に抑制している可能性があります。

## その他の人気のあるSW機能

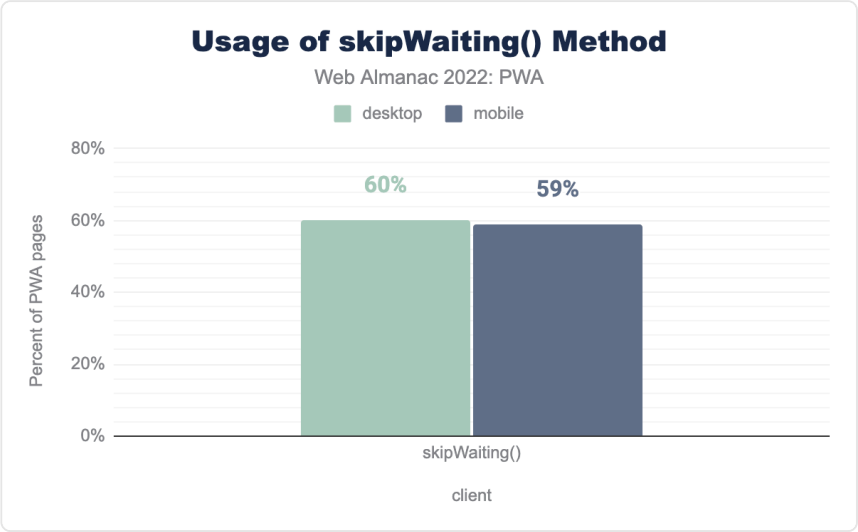



*図17.6. `skipWaiting()` メソッドの使用状況。*

昨年の統計[638]と同様に、サービスワーカーを直ちにアクティブにするために使用される `skipWaiting` メソッドは、依然として開発者の間で非常に人気があり、デスクトップの60%、モバイルのウェブアプリの59%で使用されています。

これらはもっとも使用されているサービスワーカーオブジェクトのトップです：

638. https://almanac.httparchive.org/ja/2021/pwa#その他、人気のあるサービスワーカーの機能

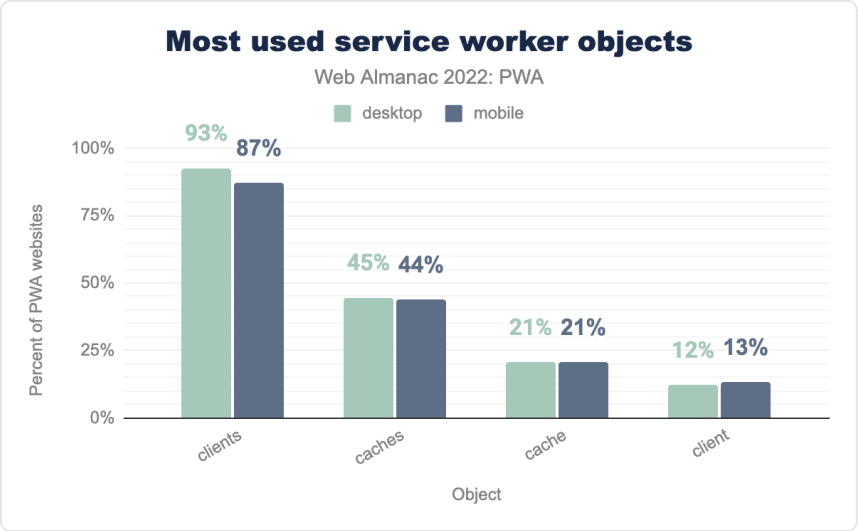



*図17.7. もっとも使用されているサービスワーカーオブジェクト。*

これらはもっとも使用されているメソッドです：

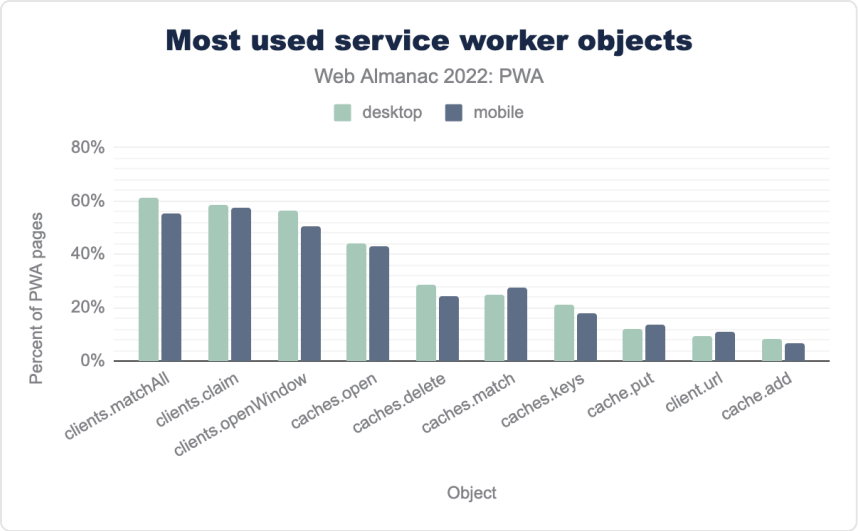



*図17.8. もっとも使用されているサービスワーカーオブジェクトのメソッド。*

# Webアプリマニフェスト

*Web*アプリマニフェスト ファイルは、アプリケーション自体に関する情報を含むJSONファイルです。マニフェストファイルは、PWAを定義するもう1つの主要なコア技術です。キーと値のペアに含まれるデータには、アプリをホストOSに表示、促進、統合するための関連情報があります。

Webアプリのマニフェストを完全に記述しておくことは、オンラインリポジトリへの高度な発見性を利用したり、アプリケーションストアへの提出、さらには最近ではアプリのシェアターゲットやファイルハンドリングなどの高度な機能にアクセスする方法としても不可欠です。PWA技術に基づいたウィジェットを有効にする[639]ための最先端の作業もマニフェストに根ざしており、さらなる高度なプラットフォーム統合のためのファイル自体の多様性を証明しています。

**95%**

*図17.9. デスクトップで解析可能なマニフェストファイルの割合。*

ほとんどの場合、デスクトップで95%、モバイルで94%我々が見つけたマニフェストはJSONとして解析可能です。これは、マニフェストを使用するほぼすべてのWebアプリが正しく形成されていることを示しています。

これは、Webアプリのインストールに寄与する特定のフィールドの完全性や最小限の可用性を示すものではありません。実際、マニフェストファイルには現在必要なプロパティがありません。技術的には空のファイルでも有効なマニフェストファイルです。

マニフェストファイルは、ブラウザがインストールを促すための信号を送る上で重要な役割を果たしますが、プロンプトのトリガー方法はブラウザによって異なります[640]が、常にマニフェストファイルのサブセットが関与しています。

以下は、マニフェストファイルとサービスワーカーの使用状況の数値です。これら2つが一緒に使用されることは、一般的にインストール可能性を意味します。

639. *https://github.com/aarongustafson/pwa-widgets#how-widgets-are-represented-in-these-apis*
640. *https://web.dev/installable-manifest/#in-other-browsers*

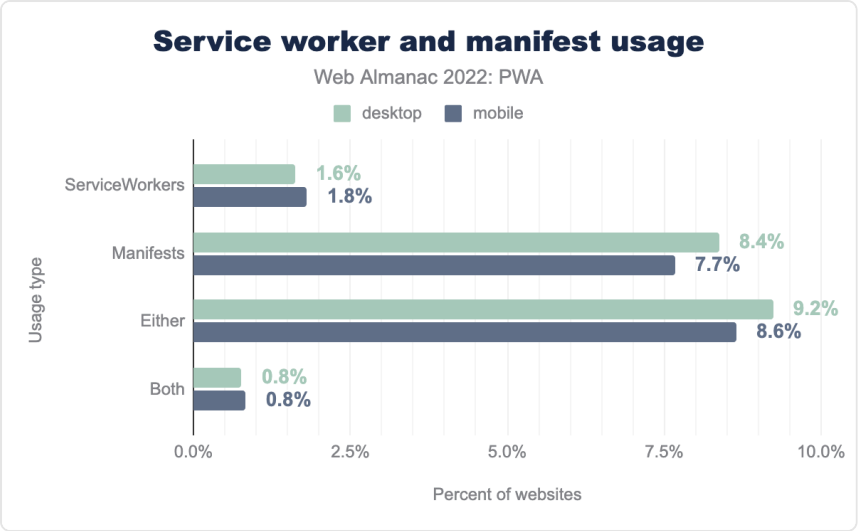



*図17.10. サービスワーカーとマニフェストの使用状況。*

データからは、Webアプリケーションがマニフェストファイルを持っている可能性がサービスワーカーよりも約5倍高いことがわかります。多くのプラットフォーム、たとえばコンテンツ管理システム（CMS）がウェブサイトのマニフェストファイルを自動生成するため、このような状況が生じています。

デスクトップとモバイルのウェブサイトのわずか0.8%がサービスワーカーとマニフェストファイルの両方を実装しており、これは伝統的なアプリのようにデバイスにインストールできるウェブサイトが1%未満であることを意味します。

この章では、主にサービスワーカーとマニフェストの両方を持つサイトに興味がありますので、とくに注記されていない限り、この章に提示されているマニフェストデータはPWAサイトのものです。

## マニフェストのプロパティ

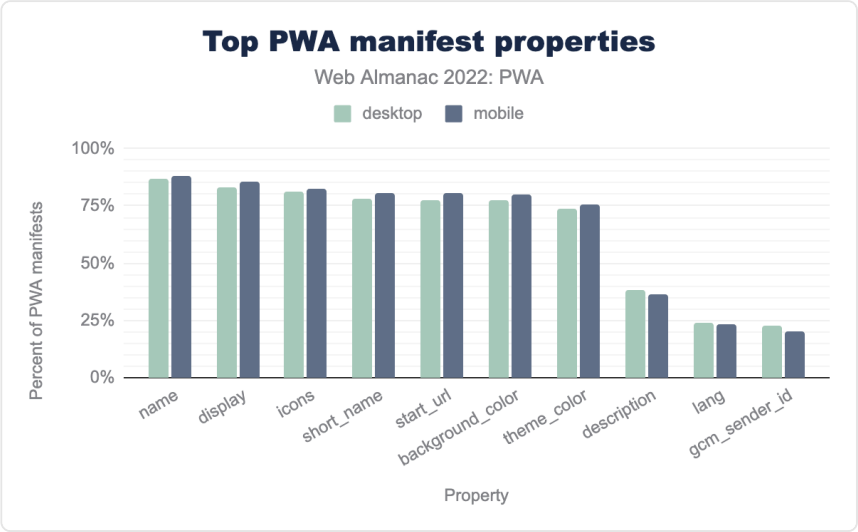



*図17.11. PWAマニフェストの主要なプロパティ。*

昨年と比較して、マニフェストファイルで使用される主要なプロパティに顕著な変化はありません。

`gcm_sender_id`は標準化されたプロパティではなく、Google Developer Consoleがアプリを識別し、古いバージョンのChromeがGCMサービスに依存するWebプッシュを実装するのに使用されます。

ほとんどのPWAでは、デスクトップで80%、モバイルで79%が優先的な向きを定義していません。設定されている場合、もっとも頻繁に使用される値は「portrait」で、デスクトップで13%、モバイルウェブサイトで15%がその値をマニフェストで定義しています。

### `display`プロパティ

`display`プロパティをさらに詳しく見ると、以下の値があります：

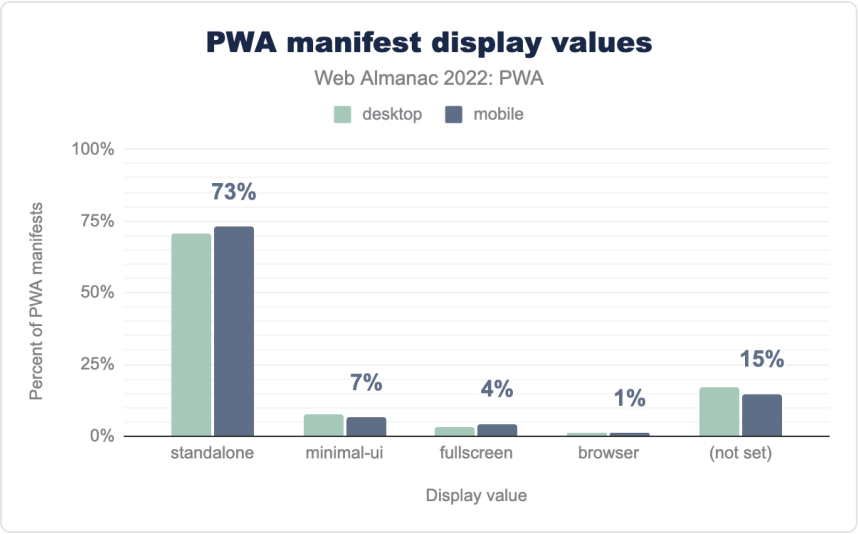



*図17.12. PWAマニフェストのdisplay値。*

`display: standalone`モードは、もっとも一般的な表示モードで、表示モードを定義するウェブサイトの約3/4が使用しています。これは、アプリがインストール可能になる表示モードの1つでもあります。

### `icons` プロパティ

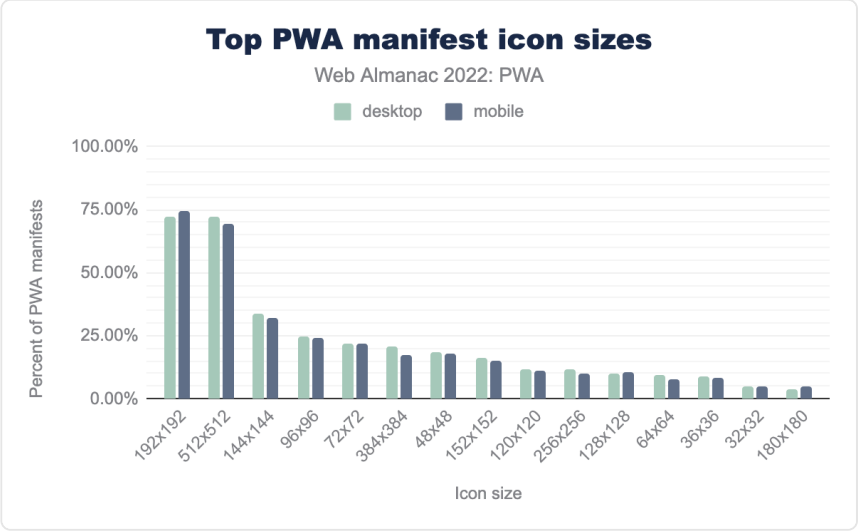



*図17.13. PWAマニフェストのアイコンサイズのトップ。*

PWAは、アプリが広告されたり配置されるさまざまな場所に対応するために、異なるアイコンサイズを生成する必要があります。異なるデスクトップおよびモバイル環境に必要な多数のアイコンを生成するための多くのツールが存在します。マニフェストファイルに存在する2つのもっとも一般的なアイコンサイズは `192x192` と `512x512` です。これらのサイズは分析されたマニフェストファイルの約70%に登場します。

### インストールと発見性のプロパティ

Webアプリマニフェストファイルには、アプリケーションの説明に役立つデータが含まれている場合があります。これらのプロパティは、アプリケーションを促進するためにストアやその他の配布メカニズムによって使用されることがあります。より豊かなブラウザベースのインストールダイアログ[641]の成長もこれらのフィールドをより顕著に利用しています。マニフェストファイルのアプリケーション情報サプリメントの一部として見つかる関連フィールドは以下の通りです：

- `description`：このプロパティは、デスクトップの36%とモバイルの34%のWebアプリマニフェストに存在します。説明はアプリケーションの機能を説明するために重要であり、通常はストア向けにアプリに関する情報を提供するために使用

641. https://developer.chrome.com/blog/richer-pwa-installation

されます。現在、インストール可能なPWAの約3分の1がこの情報を提供しています。

- `screenshots` : このプロパティは、アプリストアやブラウザのインストールプロンプトで使用する1つ以上のスクリーンショットのURLを提供します。この機能を利用するマニフェストを持つPWAは、デスクトップで1.12%、モバイルデバイスで1.19%です。
- `categories` : カタログの組織化のためのヒントとして使用されます。
- `iarc_rating_id` : WebアプリのIARC認証コード[642]を表す文字列です。デスクトップとモバイルアプリの0.05%が、アプリやゲームの評価を広告するためにこのフィールドを利用しています。

### マニフェストのカテゴリ

カテゴリについてもう少し詳しく見てみましょう。

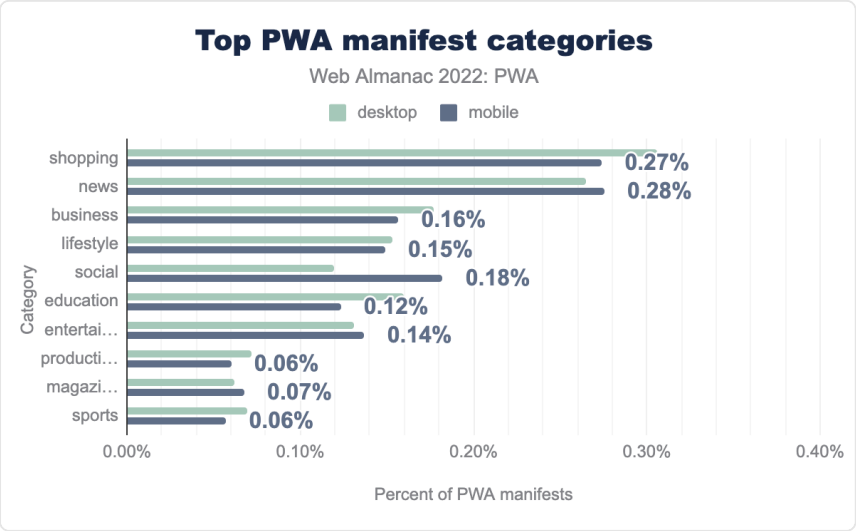



*図17.14. PWAマニフェストのトップカテゴリ。*

また、PWAサイトだけでなく、すべてのウェブサイトについても同じデータを示します。これは私たちが「PWAサイト」として使用しているサービスワーカーを持つウェブサイトの定義です：

642. *https://www.globalratings.com/how-iarc-works.aspx*

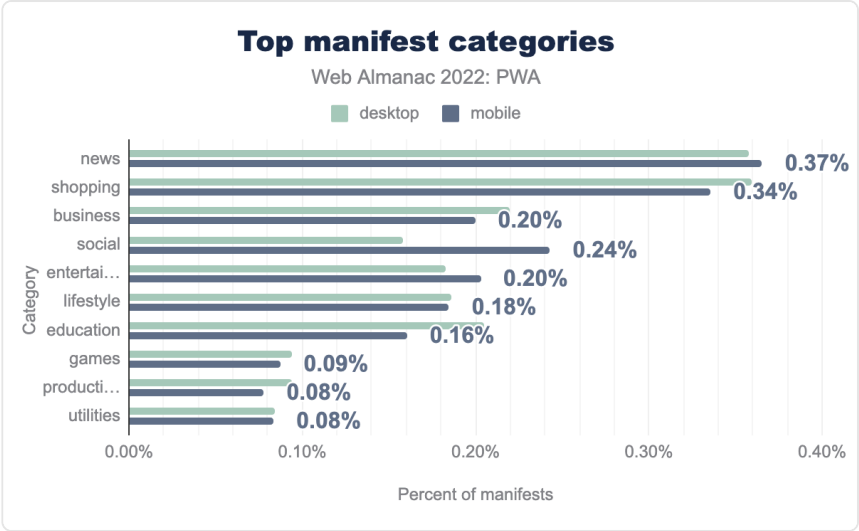



*図17.15. マニフェストのトップカテゴリ。*

PWAとウェブサイトのトップカテゴリは同じですが、それぞれがわずかに異なります。トップカテゴリはショッピング、ニュース、ビジネスです。

### 高度な機能

マニフェストファイルは、最新のプラットフォーム機能の活用も可能にします。これらの機能により、高度なウィンドウ機能やホストOS内での動作の登録が可能になります。これらの機能の多くは最近プラットフォームに導入されたばかりであり、これらの新しいAPIの導入の始まりをこのデータが示していることを期待しています。

これらはあまり使われていない、より高度な機能であるため、以前のマニフェストプロパティのトップのグラフには表示されませんが、その使用状況を見る価値があります：

- `shortcuts`：デスクトップで6.2%、モバイルで4.3%のPWAがアプリ内の深いリンクへのショートカットを使用しています。
- `file_handlers`：インストールされたPWAが特定のファイル拡張子のハンドラーとして自身を登録することを可能にします。デスクトップで0.01%、モバイルで0.02%が`file_handlers`を使用しています。
- `protocol_handlers`：PWAは、事前定義されたプロトコルまたはカスタムプロトコルのハンドラーとして登録できます。現在の使用率はデスクトップで0%、モバイルウェブサイトで0.01%です。

- `share_target` : デスクトップで5.3%、モバイルで3.1%のPWAが共有データを受信するために自身を登録する能力を持っています。
- ウィンドウコントロールオーバーレイ[643]: タイトルバーが通常占める領域を解放する機能はデスクトップ専用の機能です。この機能は最近Chromium 105で導入され、デスクトップPWAのマニフェストの0.01%がこれを利用しています。
- `note_taking` : デスクトップの0%とモバイルサイトの0.01%が、クイックノートを取るための便利な方法として特別な統合を利用しています。

#### ネイティブを優先するマニフェスト




*図17.16. モバイルの関連アプリケーションフィールドを持つマニフェストファイル*

マニフェストには、`related_applications` フィールドにリストされたアプリケーションがウェブアプリケーションよりも優先されるべきかどうかを指定するプロパティがあります。これにより、ユーザーエージェントはウェブアプリの代わりに関連アプリのインストールを提案する可能性があります。分析されたすべてのマニフェストファイルの中で、デスクトップで2.3%、モバイルで2.0%のマニフェストがこのプロパティを設定しています。

## Fugu API

PWAは高度なウェブ機能と密接に関連しています。これらの機能は一般的にプロジェクトFuguの一部であり、これはChromiumプロジェクト内で孵化中の新しいウェブプラットフォーム機能のコレクションのコードネームです。




*図17.17. デスクトップでもっとも使用されているFugu API*

ウェブプラットフォームに追加された機能の増加するリストから、ウェブで使用されているトップのAPIは以下の通りで、PWAにとって有用です：

643. *https://wicg.github.io/window-controls-overlay/*

| API | デスクトップ | モバイル |
|---|---|---|
| Web Share | 8.8% | 8.4% |
| Add to Home Screen | 8.6% | 7.7% |
| Service worker | 4.2% | 3.9% |
| Push | 2.0% | 1.9% |

*図17.18. もっとも使用されているFugu API。*

これらについては詳しく掘り下げませんが、Capabilitiesの章でそれらをカバーしています。

## LighthouseによるPWAの洞察

Lighthouse[644]はオープンソースで自動化されたツールで、Webページの品質を向上させるために使用されます。ウェブサイトに対して多数の監査を実行でき、PWA監査の専用カテゴリがあります。利用可能なデータからは、過去12か月間のPWAの状況について興味深い事実が明らかにされています。

644. https://developer.chrome.com/docs/lighthouse

## Lighthouse監査

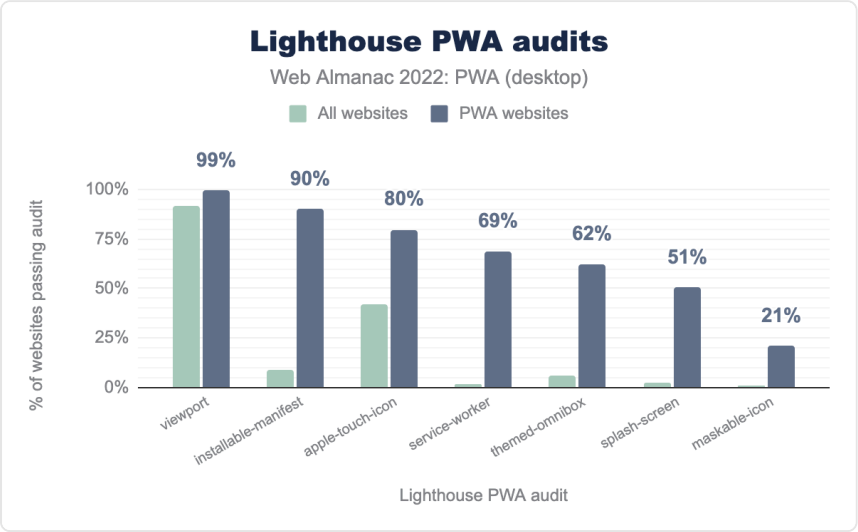



*図17.19. デスクトップのLighthouse PWA監査。*

一般的なWebサイトよりもPWAサイトでPWA監査に合格する頻度がはるかに高いのは驚くことではありませんが、viewportメタタグ[645]やapple-touch-iconメタタグ[646]の存在など、PWAサイトでなくても適用される（そして使用される）監査も多くあります。

645. *https://web.dev/viewport/#how-to-add-a-viewport-meta-tag*
646. *https://web.dev/apple-touch-icon/#how-the-lighthouse-apple-touch-icon-audit-fails*

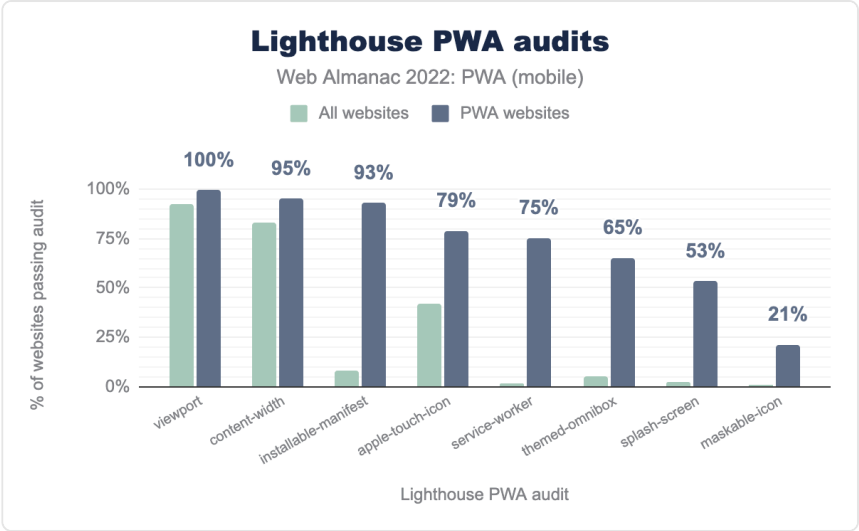



*図17.20. モバイルのLighthouse PWA監査。*

モバイルサイトのLighthouseデータを見ると、似たような統計が見られますが、モバイルのみのcontent-widthメタタグ[647]もここに表示され、両方によってうれしいことに合格されています。

viewportメタタグの存在は重要であり、ダブルタップでズームインするために300-350msの遅延を取り除くために役立ちます。さらに、モバイルデバイスでの利点として、アプリをデバイスの画面サイズに最適化します。ほぼすべてのウェブサイト（PWAかどうかにかかわらず）がこれを含んでいるのは驚くことではありません。

インストール可能なマニフェストも両方のトップ3リストに登場しています。予想通り、これはPWAサイトにとって非常に高い値を持っており、デスクトップで90.2%、モバイルで95.2%ですが、すべてのウェブサイトでは非常に低い値を持っています。これは、開発者がこれらをインストールすることを意図していないためと思われます。

最後に、`apple-touch-icon`はPWA関連のLighthouse監査で3番目に位置しています。iOS 1.1.3以来、Safari for iOSは開発者がホーム画面でウェブサイトまたはアプリを表す画像を指定する方法をサポートしています。これは主にモバイルデバイスで関連しています。

647. *https://developer.chrome.com/docs/lighthouse/pwacontent-width?hl=ja*

# Lighthouseスコア

Lighthouseインサイトセクションを締めくくるために、監査に基づいてPWAサイトの全体的なLighthouse PWAスコアを見てみましょう。

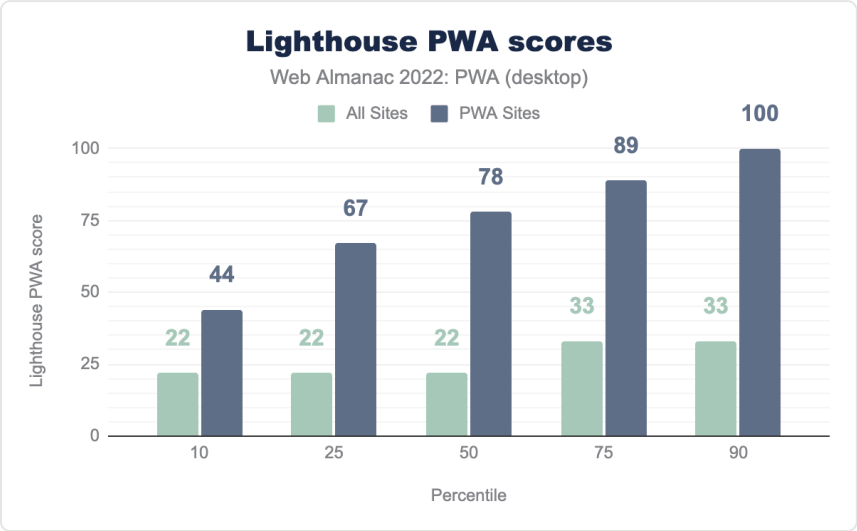



*図17.21. デスクトップのLighthouseスコア。*

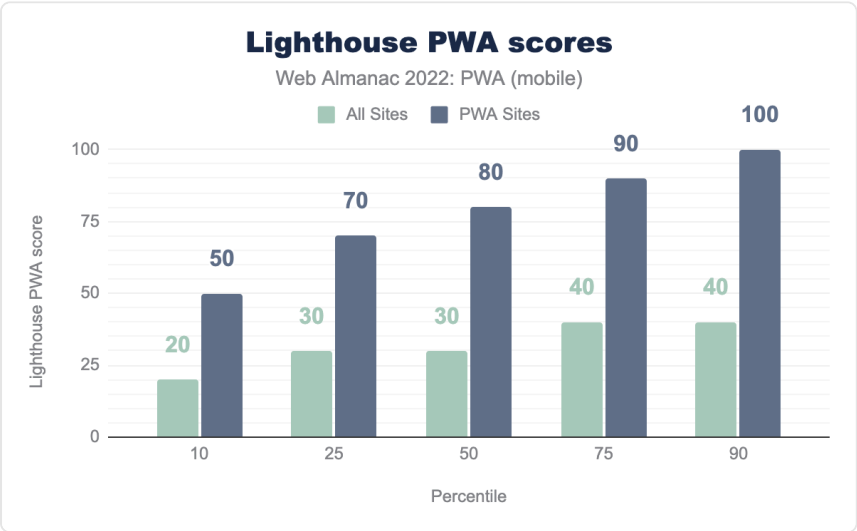



*図17.22. モバイルのLighthouseスコア。*

予想通り、PWAサイトはPWA監査のスコアがかなり高くなっています。これらの監査は、ドキュメント[648]で詳細に説明されているように、速度、信頼性、インストール可能性、その他のPWA要件を調べます。

また注目すべきは、PWAサイトの監査スコアの範囲（50-100）で、これは存在するPWAの違いを表しています。対照的に、他のウェブの範囲は比較的一貫しており（20-40）、これは以前に議論されたほとんどのサイトに関連する主要な2つの監査（ビューポートとアイコン）を反映しています。

## サービスワーカーのライブラリ

サービスワーカーは非常に強力なツールであり、そのAPIによって開発者は以前は不可能だったアプリ体験を作成できるようになります。たとえば、独自のオフライン体験を作成したり、パフォーマンスを向上させるためにアセットをキャッシュしたりします。しかし、ウェブアプリとネットワークの関係を扱うコードを作成することには複雑さや注意点が伴います。ここでライブラリが開発者の生活をより良くすることができる場所であり、サービスワーカーAPIを取り巻く高レベルの抽象化を提供します。

### Workboxの使用

Workbox[649]は、開発者がサービスワーカーの使用を容易にするために作られたライブラリセットです。基本から他のWorkboxライブラリで再利用されるworkbox-core[650]まで、キャッシング戦略、バックグラウンド同期、プリキャッシングなど、より具体的なタスクまで幅広いライブラリが含まれています。

*648. https://developer.chrome.com/docs/lighthouse/pwa*
*649. https://developer.chrome.com/workbox/*
*650. https://developer.chrome.com/docs/workbox/modules/workbox-core*

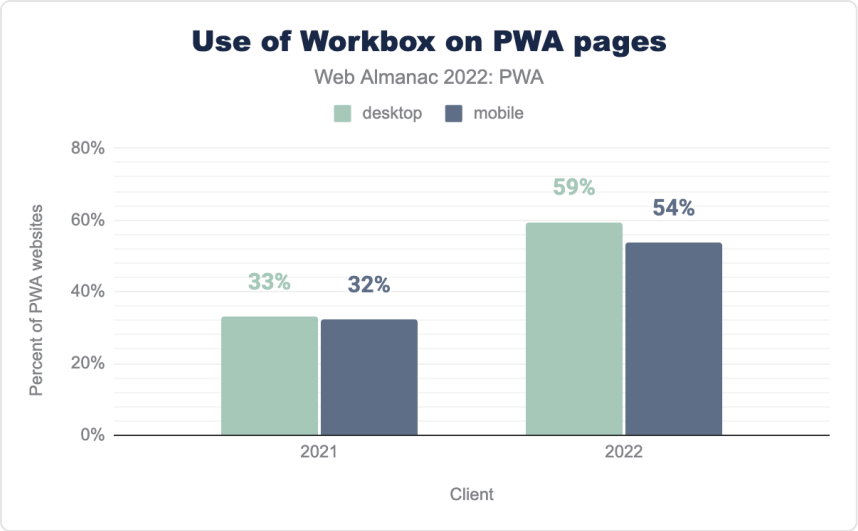



*図17.23. PWAページでのWorkboxの使用。*

昨年と比較してWorkboxの使用に大きな増加が見られます。昨年はモバイルで33%だったのに対し、今年は54%に増加しており、デスクトップPWAのほぼ60%が何らかの形でWorkboxを使用しています。

サービスワーカーによって制御されるページ数の増加が最上位1,000のウェブサイトではなく、より詳細なカテゴリで見られたこと、そしてWorkboxの使用の増加から、Workboxの採用はトップのウェブサイトで技術が採用されるのを待っていた企業やウェブサイトの内部で起こっていると推測できます。または、完全にカスタムなサービスワーカーの実装が必要なく、Workboxのテスト済みパターンを最大限に活用できる場合もあります。

## Workboxパッケージ

Workboxは、開発者がサイトのニーズに応じてどの部分をプロジェクトに追加するか選択できるように構成されています。以下に示す使用状況は、現在の開発者がどのPWA機能を実装しているかを文書化するのに役立ちます。

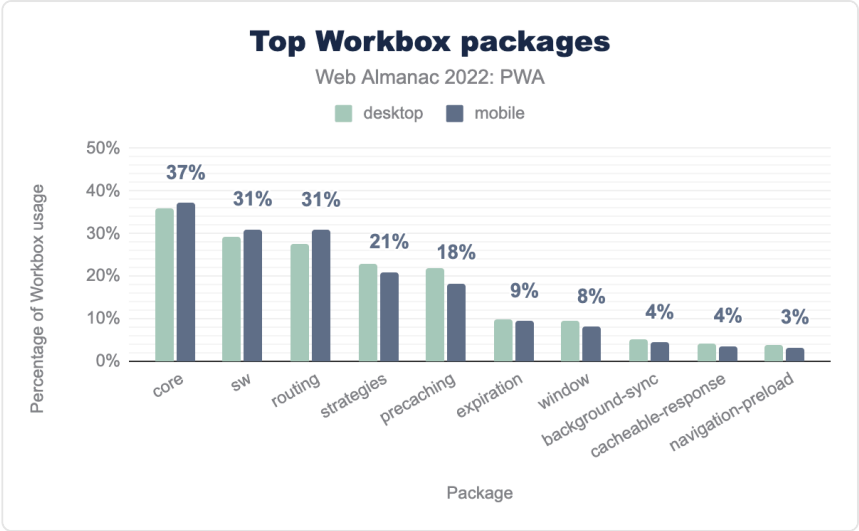



*図17.24. トップWorkboxパッケージ。*

ここでも使用率の全体的な増加が見られ、`workbox-core`の基本ライブラリの使用が14%増加しました。`workbox-core`、`workbox-routing`、`workbox-strategies`は、異なるアーティファクトをWebアプリに提供するキャッシング戦略を作成するために使用されます。これらがすべてトップに位置しているのは、サービスワーカーの主要な利点を有効にするためです。

`workbox-precaching`の使用にもかなりのジャンプが見られます。プリキャッシングは、パッケージ化されたアプリが使用するモデルをエミュレートするために使用できます。`workbox-precaching`を使用すると、サービスワーカーのインストール時にキャッシュされるアセットを選択し、その後の訪問でこれらのアセットをより速く読み込むことができます。

驚くべきことに、`workbox-sw`の使用が増えていますが、これはWorkbox 5[651]からWorkboxチームが開発者に`importScripts()`を使用して`workbox-sw`（ランタイム）をロードするのではなく、Workboxランタイムのカスタムバンドルを作成することを奨励しているためです。Workboxチームは*workbox-sw*のサポートを続けますが、新しい技術が現在推奨されています。実際に、ビルドツールのデフォルトはその方法を優先するように変更されています。

この増加は、古いバージョンのWorkboxを使用しているライブラリから来ている可能性があります。たとえば、`create-react-app`のバージョン3などです。

651. *https://github.com/GoogleChrome/workbox/releases/tag/v5.0.0*

## Webプッシュ通知

通知は、ユーザーとの再エンゲージメントのための強力な方法です。また、プラットフォーム固有のアプリケーションから期待される特徴の1つでもあります。通知はタイムリーで関連性があり、正確な情報を提供する完璧な方法であり、WebプッシュAPIによって提供されています。

### Webプッシュ通知の受け入れ率

Web通知の実装が開発者やユーザーにとってもっともスムーズではなかったことを認めることができますが、それがどれほど有用なツールであるかも重要です。カレンダーの通知や購読の更新、ゲームなど、重要なのはユーザーがいつそれらをオンにするかを選択できることです。

通知が有用であるためには、タイムリーで、正確で、関連性がある必要があります[652]。通知の許可を求めるプロンプトを表示する際には、ユーザーがサービスの価値を理解している必要があります。開発者は、ブラウザの許可ダイアログを表示する前に、ユーザーが特定の通知から得られる利点を共有することで、ユーザーを通知に引き込むチャンスがあります。

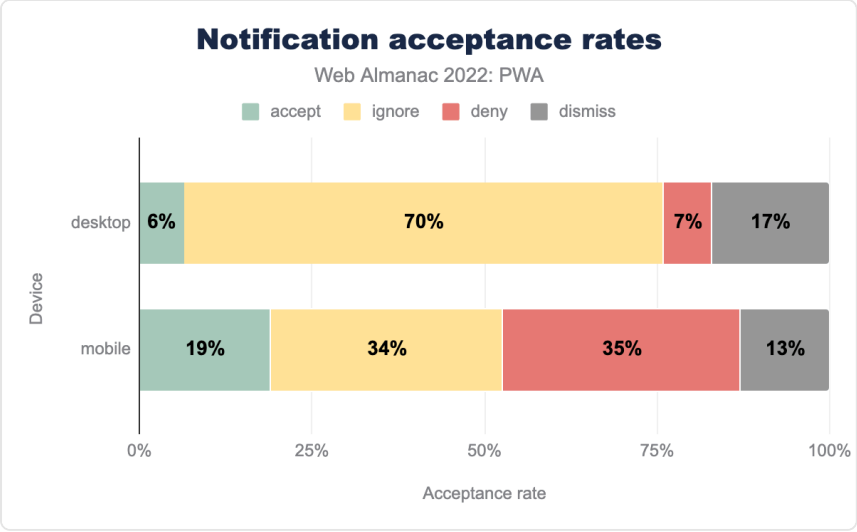



*図17.25. 通知の受け入れ率。*

異なるプラットフォームでのUXの改善と通知サポートの成長にもかかわらず、通知の受け入れに大きな変化はありません。2020年初頭以来、デスクトップでの受け入れ率は約6%、

652. https://developers.google.com/web/fundamentals/push-notifications

モバイルで20%です。

デスクトップとモバイルの通知受け入れ率は共通の傾向を共有しており、通知を無視する傾向があります。無視の合計は、2020年2月の48%から2022年6月には70%まで上昇しました。モバイルプラットフォームでは、2020年2月の1.88%から今年の6月には34%まで驚異的に増加しました。通知の疲れと、セキュリティ、プライバシー、高度な機能に対するプロンプトの増加が部分的に原因であり、これを解決し、異なるプラットフォームでより良い統一されたUXを提供するための作業が行われています。

## 結論

2022年はPWAにとって素晴らしい年でした。デスクトッププラットフォームとの統合を可能にする機能の増加は、業界の大手企業によるこの技術の採用を推進しました。昨年は、プロトコルハンドラー、ウィンドウコントロールオーバーレイ、OSログイン時の実行などの高度な機能が導入され、PWAをアプリケーション開発の重要な技術として位置づけるようになりました。これは励みになりますが、ウェブプラットフォーム全体を代表するものではありません。サービスワーカーの使用率は、2021年のデータと比較して半分に減少しましたが、PWA技術を使用して構築された大規模なアプリケーションの台頭が見られました。

マニフェストファイルは引き続き健全な状態にあり、昨年からわずかに増加してデスクトップで95%に達しました。これらのファイルの正確さは素晴らしいですが、完全性にはまだ改善の余地が多く残されています。現在、すべてのウェブサイトの約0.8%しかインストール可能と見なされていません。`shortcuts`や`share_target`などの多くの高度な機能が、約5%のPWAで徐々に採用され始めています。`protocol_handlers`やウィンドウコントロールオーバーレイなどの他の機能は新しすぎて、データに影響を与えるには至っていません。今年はこれらのFugu APIの多くについての初期のスナップショットも提供しています。

通知の疲れは理解できることですが、ユーザーは正当な通知の使用例を要求し、評価しています。ブラウザベンダーは、侵入的でない許可リクエストの実験を行っており、Webプッシュ通知はプラットフォーム全体で一貫した体験を提供する利点があり、ユーザーが使用しているデバイスに関係なくリクエストした通知を提供します。

この情報があなたのPWAの旅に光を当て、開発者がAPI採用の現在の技術トレンドを理解するのに役立つことを願っています。

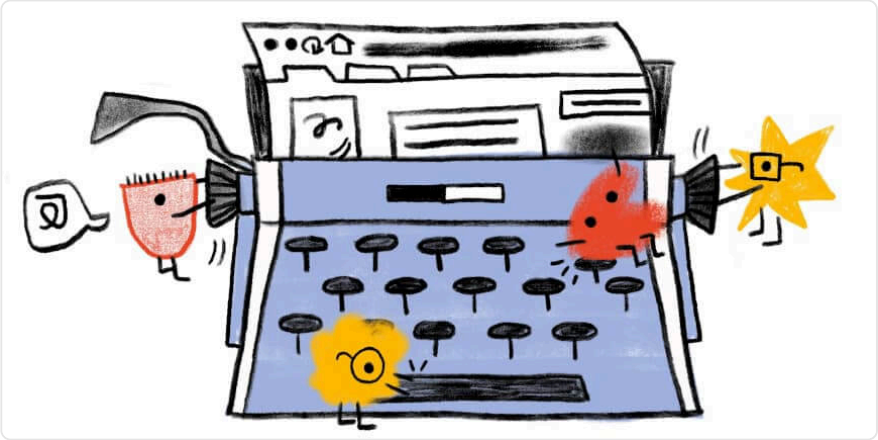
## 著者


### Diego Gonzalez

@diekus diekus https://diek.us


Diego Gonzalezは、コスタリカ出身のコンピュータエンジニアで、Microsoft EdgeブラウザのPWAプラットフォーム機能のプロジェクトマネージャーとして働いています。

# 部 III 章 18
# CMS



***Jonathan Wold によって書かれた。***
*Alex Denning、 Alon Kochba と Dan Knauss によってレビュ。*
*Colt Sliva による分析。*
*Dan Knauss 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

この章では、コンテンツ管理システム（CMS）エコシステムの現状と、それらがウェブ上でのコンテンツ体験におけるユーザー認識を形成する上で果たしている役割について理解を深めようとしています。私たちの目標は、CMSの全体像と、これらのシステムによって作成されたウェブページの特性を探求することです。

私たちは、CMSが高速で強固なウェブを構築するための共同の取り組みにおいて重要な役割を果たしていると信じています。現在の状況を理解し、疑問を投げかけ、将来の研究に向けた問いを立てることが、この目標を達成するための道筋です。

私たちのチームは、今年のデータに対して好奇心を持って取り組み、その好奇心を複数の人気CMSに関する個人的な専門知識と結びつけました。CMS間の違いやその上で扱われるコンテンツの種類の違いを考慮しながら、私たちの分析を読んでいただくことをオススメします。

## CMSとは何ですか？

コンテンツ管理システム（CMS）という用語は、個人や組織がコンテンツを作成、管理、公開できるようにするシステムを指します。とくにウェブコンテンツのCMSは、ウェブ上で体験されるコンテンツを作成、管理、公開するためのシステムです。

各CMSは、ユーザーがコンテンツを中心にウェブサイトを構築できるようにするさまざまなコンテンツ管理機能と対応するメカニズムを実装しています。また、CMSはコンテンツの追加と管理を容易にするための管理機能も提供します。

CMSは、サイトを構築するために提供するアプローチが大きく異なります。いくつかのCMSは、ユーザーコンテンツを補完するすぐに使えるテンプレートを提供する一方で、他のCMSはサイト構造を設計し構築するためにユーザーの関与を必要とします。

このWeb Almanacの章では、ホスティングプロバイダー、拡張機能開発者、開発エージェンシー、サイトビルダーなど、CMSプラットフォームを取り巻くエコシステムを構成するすべての要素を考慮しようとしました。このため、CMSに言及する際には、通常そのプラットフォーム自体だけでなく、その周辺のエコシステムも意図しています。

Wappalyzerの定義[653]に基づく私たちのデータセットでは、270以上の個別のCMSが特定されました。欠けているCMSを知っていますか？Wappalyzerに貢献[654]してください。

データセットの中には、WordPressやJoomlaのようなオープンソースのCMSもあれば、WixやSquarespaceのようなプロプライエタリなCMSもあります。一部のCMSは「無料」でホスティングされたり、セルフホストされたプランで使用でき、また、エンタープライズレベルまで利用できる上位プランのオプションもあります。

CMS全体の領域は、離散的でありながら相互に関連するCMSエコシステムの複雑で分散型の世界です。

## CMSの採用

私たちの分析では、デスクトップとモバイルのウェブサイトが含まれています。調査したURLの大部分は両方のデータセットに含まれていますが、一部のURLはデスクトップまたはモバイルデバイスのみでアクセスされます。これによりデータに違いが生じる可能性があるため、デスクトップとモバイルの結果を別々に検討しました。

653. *https://www.wappalyzer.com/technologies/cms*
654. *https://github.com/wappalyzer/wappalyzer/blob/7ac625c34432cb35d01abd683f88d3bfadca4cca/CONTRIBUTING.md*

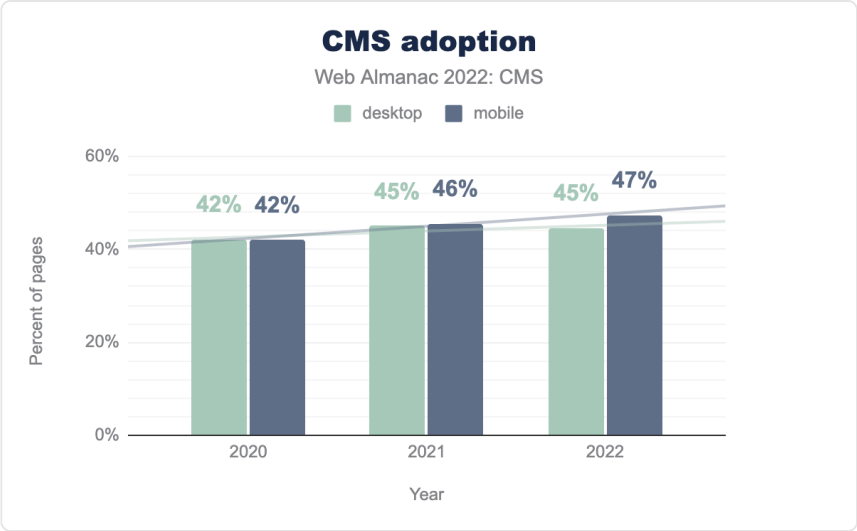



*図18.1. CMSの採用状況。*

2022年6月の時点で、Web Almanacのデスクトップデータセット内のウェブサイトの45%がCMSによって運営されており、2021年[655]と同様の利用状況を示しています。モバイルデータセットでは、2021年の46%から2022年には47%に増加しています。デスクトップの生データを詳しく見ると、絶対数および割合の両方でわずかな減少が見られますが、この減少はCMSの使用率の低下を示すものではなく、割り当ての小さな変動によるものである可能性が高いです。デスクトップサイトの数がHTTP Archive（ソースであるCrUXデータセット）で640万サイトから540万サイトに大幅に減少した一方で、モバイルサイトの数は750万サイトから約40万サイト増加して790万サイトに達しました。この増加は、デスクトップの代わりにモバイルデバイスの使用が引き続き成長していることを反映していると考えられます。

これらの数値を、W3Techs[656]のような一般的に使用されている他のデータセットと比較することは有益です。W3Techsは、2021年6月の時点で、ウェブサイトの64.6%がCMSを使用して作成されていると報告しています。これは、2020年6月の59.2%から9%以上の増加です。

私たちの分析とW3Techsの分析の違いは、調査方法やCMSの定義の違いによって説明できます。

W3Techsの定義は次のとおりです：*"コンテンツ管理システムは、ウェブサイトのコンテンツを作成および管理するためのアプリケーションです。このカテゴリーには、ウィキ、ブログエンジン、ディスカッションボード、静的サイトジェネレーター、ウェブサイトエディタ、またはウェブサイトコンテンツを提供するあらゆる種類のソフトウェアも含まれま*

655. https://almanac.httparchive.org/ja/2021/cms#CMSの採用
656. https://w3techs.com/technologies/history_overview/content_management/all/q

*す。”*

前述のように、WappalyzerのCMSの定義は私たちよりも厳しいものです。Wappalyzerは、W3Techsのレポートに登場するいくつかの主要なCMSを除外しています。私たちのCMSの定義について詳しくは、Methodologyページをご覧ください。

## 地域別のCMSの採用

CMSは世界中で使用されていますが、国によってばらつきがあります。

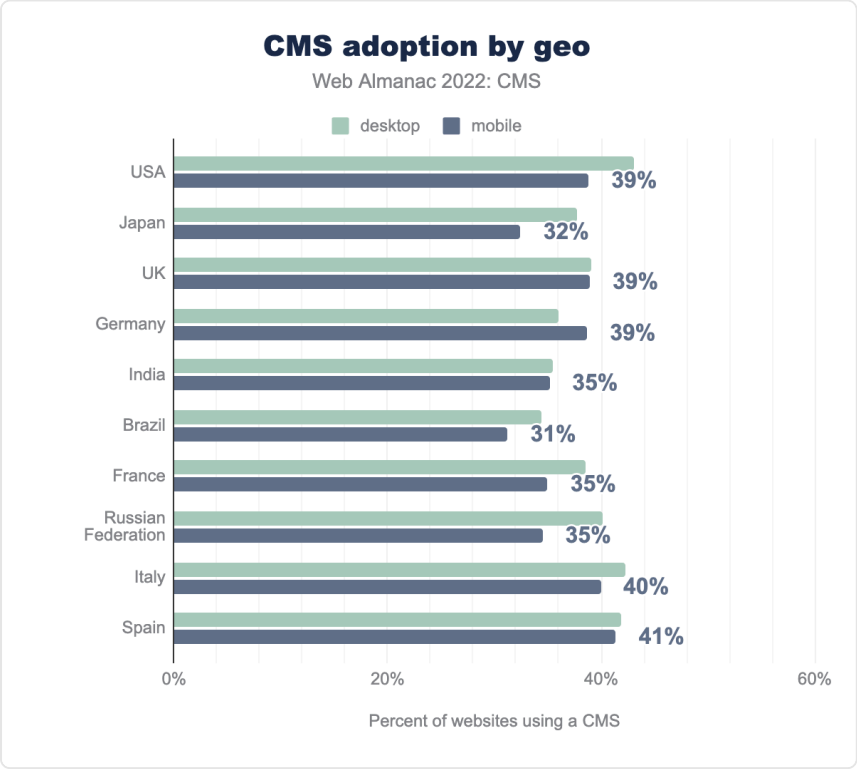



*図18.2. 地域別のCMS採用。*

ウェブサイト数がもっとも多い国の中で、CMSの採用率がもっとも高いのはイタリアとスペインで、モバイルサイトの40%〜41%がCMSで構築されています。ブラジルと日本では、採用率がそれぞれ31%と32%ともっとも低いです。

とくに注目すべきは、個々の国を考慮すると、2021年のデータセット[657]と比較して全体的に減少していることです。モバイルの結果を年ごとに比較すると、インドを除くすべての国で減少が見られ、英国とドイツでは4%減少し、米国とイタリアでは8%減少しています。地理的に一貫して減少していることを考慮すると、CMSの採用が大幅に減少したというよりも、割り当てのばらつきが原因である可能性が高いです。次年度の分析ではこの点をさらに評価することをオススメします。

## ランク別のCMSの採用

私たちは、データセットに含まれるサイトの推定ランク別にCMSの採用を調査しました。

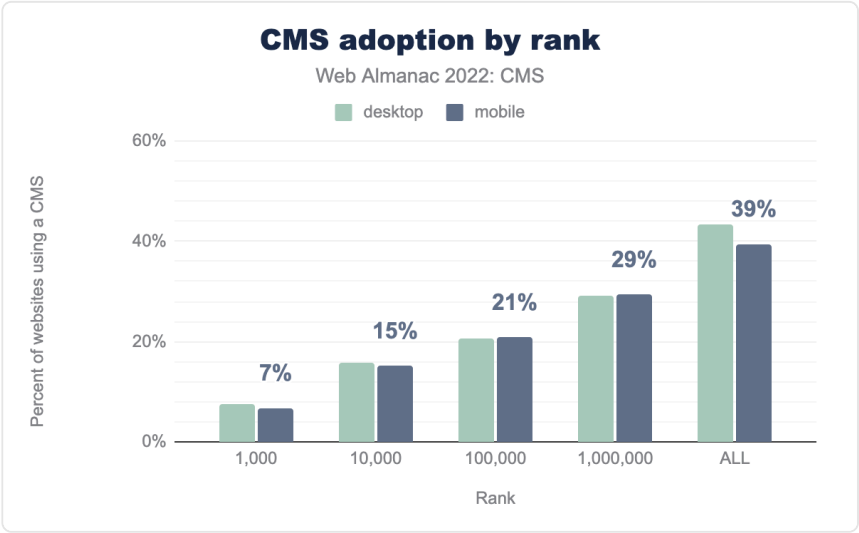



*図18.3. ランク別のCMS採用。*

データセットによると、トップ1,000のウェブサイトのうちCMSを使用しているのはデスクトップとモバイルの両方で7%未満ですが、データセット内のすべてのモバイルサイトの47%がCMSを使用しています。この明らかな違いの説明として、昨年も述べたように、ウェブサイトを持つ中小企業は使いやすさから人気のCMSを利用する傾向があり、そのCMSが容易に特定できるということが考えられます。しかし、よりランクの高いウェブサイトを持つ大企業は、特注のCMSソリューションを使用していることが多く、それらは私たちには特定できません。

もう1つの説明は、開発に多くのリソースを割り当てられるランクの高いサイトは、セキュリティ上の理由からCMSの特定を隠す傾向があるということです。トップ1,000の90%以上

657. *https://almanac.httparchive.org/ja/2021/cms#地域別のCMS導入状況*

がCMSを完全に使用しないというのはあり得ないことですので、単にデータセットに反映されていないだけである可能性が高いです。

関連する傾向としては、「ヘッドレス」CMSの採用と、コンテンツ（およびそれを支えるCMS）をエンドユーザーに提供するフロントエンド体験から分離する動きがあります。

データセット全体への自信は依然として高いものの、来年度のレポートでランク別のデータセットをさらに調査し、より多くのCMSを検出および特定して結果の全体的な精度を向上させることができるかどうかを確認することに興味があります。

## もっとも人気のあるCMS

特定可能なCMSを使用しているすべてのウェブサイトの中で、WordPressサイトは相対的な市場シェアの大部分を占めており、モバイルでの採用率は35%以上です。次いでWix（2%）、Joomla（1.8%）、Drupal（1.6%）、Squarespace（1.0%）が続いています。

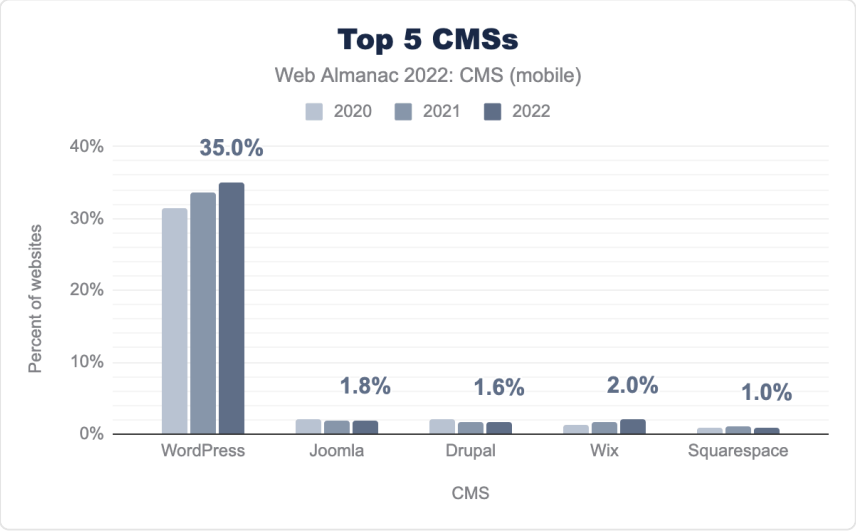



*図18.4. 過去1年間での上位5つのCMS。*

年ごとに比較すると、DrupalとJoomlaは市場シェアが引き続き減少している一方で、Squarespaceは安定しており、Wixは成長しています。WordPressは引き続き上昇しており、モバイルでは2021年に比べて1.4%増加し、デスクトップでは2021年に比べて0.2%増加しています。

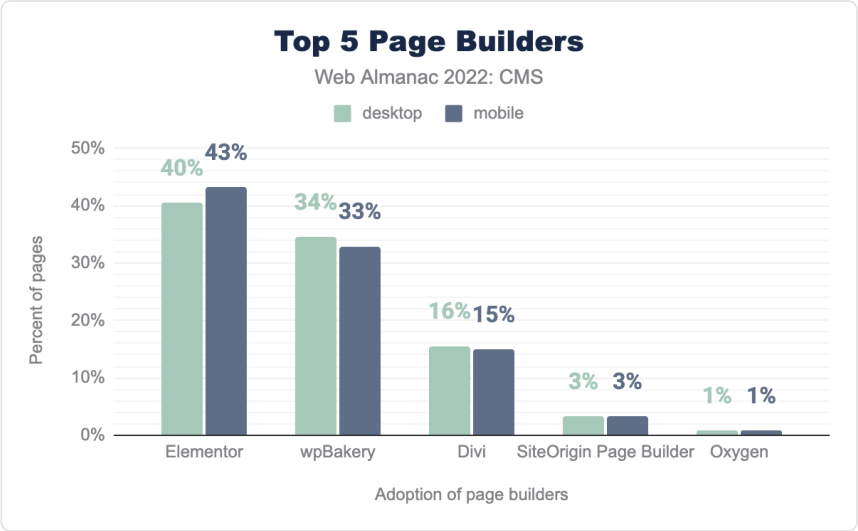



*図18.5. WordPressページビルダーの採用。*

WordPressでは、ユーザーがコンテンツ管理用のインターフェイスを提供する「ページビルダー」をよく利用します。今年はWappalyzerの検出方法が向上したため、ページビルダーの採用状況を調査しました。その結果、ページビルダーが割り当てられているWordPressサイト（私たちのデータセット内のすべてのWordPressサイトの約34%）の中で、ElementorとWP Bakeryが明確な勝者であり、Divi、SiteOrigin、Oxygenがそれに続いていることがわかりました。

今日の時点で、ページビルダーはサイトのパフォーマンスに大きな影響を与えています。歴史的に、ページビルダーはパフォーマンスが悪いことの経験的な指標とされてきました。たとえば、私たちのデータセットでは、複数のページビルダーがインストールされているウェブサイトも珍しくなく、これによりサイトが読み込むリソースが大幅に増加しています。

ページビルダーデータの追跡を開始したことで、今後の版では、年ごとのページビルダーの採用状況の変化を評価し、それらの変化がWordPress全体のCMSとしてのパフォーマンスにどのように関連しているかを調べる機会が得られます。

## CMSユーザーエクスペリエンス

CMSの重要な機能の1つは、これらのプラットフォーム上で構築されたサイトを訪問するユーザーに提供するユーザーエクスペリエンスです。私たちは、Chrome User Experience

Report[658] (CrUX) を通じたリアルユーザーメジャーメント (RUM) と、Lighthouse を使用したシンセティックテストを通じて、これらのエクスペリエンスを調査しようと試みています。

## Core Web Vitals

Core Web Vitals Technology Report[659]を使用して、利用可能なデータを詳しく調べ、月ごとに更新されるプラットフォームの進捗を確認できます。

このセクションでは、Web Almanac全体で提示されたデータの時間枠を一貫させるため、2022年6月のデータに焦点を当てます。私たちは、Chrome User Experience Report[660] が提供する3つの重要な指標を調査し、CMSによって運営されているウェブページがどのように体験されているかを理解する手がかりを探ります：

- Largest Contentful Paint[661] (LCP)
- First Input Delay[662] (FID)
- Cumulative Layout Shift[663] (CLS)

これらの指標は、優れたウェブユーザーエクスペリエンスの技術的基礎をカバーすることを目的としています。パフォーマンス の章ではこれらの指標についてさらに詳しく説明していますが、ここではCMSに関連してとくにそれらを調べています。

まず、もっとも多くのオリジンを持つ10のCMSプラットフォームを確認し、それぞれのプラットフォームで「合格」評価を持つサイトの割合を調査します。合格評価とは、上記の各指標が各サイトで「良好」（緑）範囲内にあることを意味します：LCPが2.5秒以下、FIDが100ms以下、CLSが0.1以下です。

658. *https://developers.google.com/web/tools/chrome-user-experience-report*
659. *https://httparchive.org/reports/cwv-tech*
660. *https://almanac.httparchive.org/ja/2021/methodology#chrome-ux-report*
661. *https://web.dev/articles/lcp?hl=ja*
662. *https://web.dev/articles/fid?hl=ja*
663. *https://web.dev/articles/cls?hl=ja*

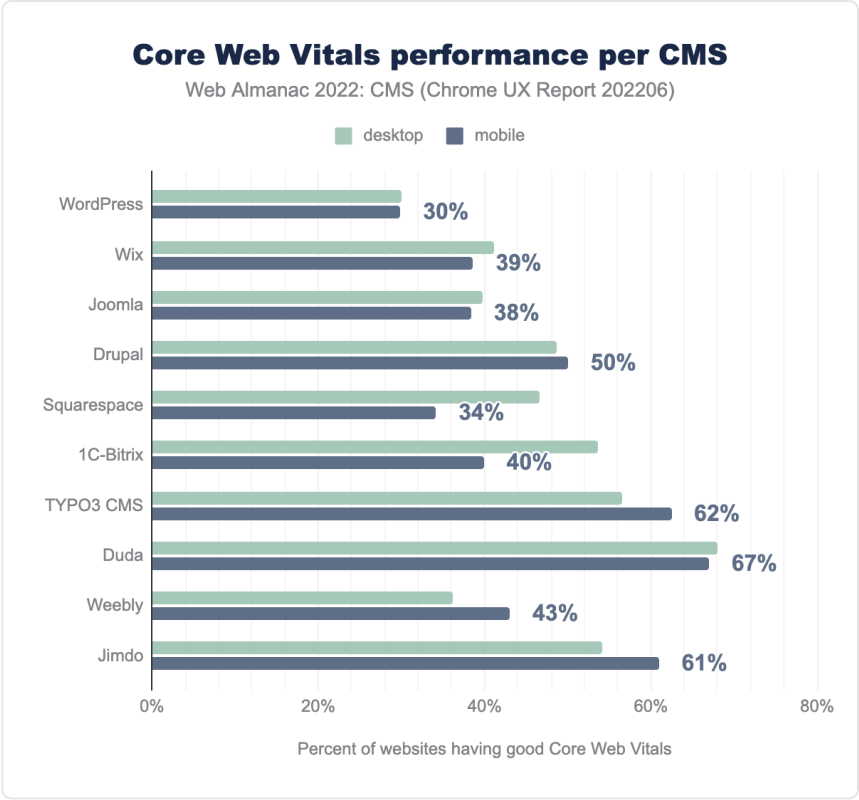



*図18.6. CMSごとのCore Web Vitalsパフォーマンス。*

デスクトップ訪問者の方がモバイル訪問者よりも高いスコアを獲得していることがわかります。これは、モバイルデバイスのリソース制限や接続の悪さによって説明できます。一部のプラットフォームでモバイルとデスクトップのパフォーマンスに大きな差があることは、ユーザーが使用するデバイスによって提供されるページが非常に異なることを示唆しています。

6月のモバイルデバイスの場合、Dudaがもっとも高い合格サイト割合を持っており、67%のモバイルサイトがすべてのCWVに合格しています。WordPressは30%のサイトが合格しており、これはもっとも低い割合ですが、それでも2021年のデータ（WordPressサイトの19%が合格）から大幅に増加しています。

デスクトップデバイスのエクスペリエンスは、ほとんどのCMSでモバイルよりも優れています。Dudaがもっとも高い合格率を持ち、68%です。WordPressは合格サイトの割合がもっとも低く、30%です。

また、これらのCMSプラットフォームのモバイルデバイス上でのパフォーマンスの進捗を、昨年のデータと比較して評価できます。

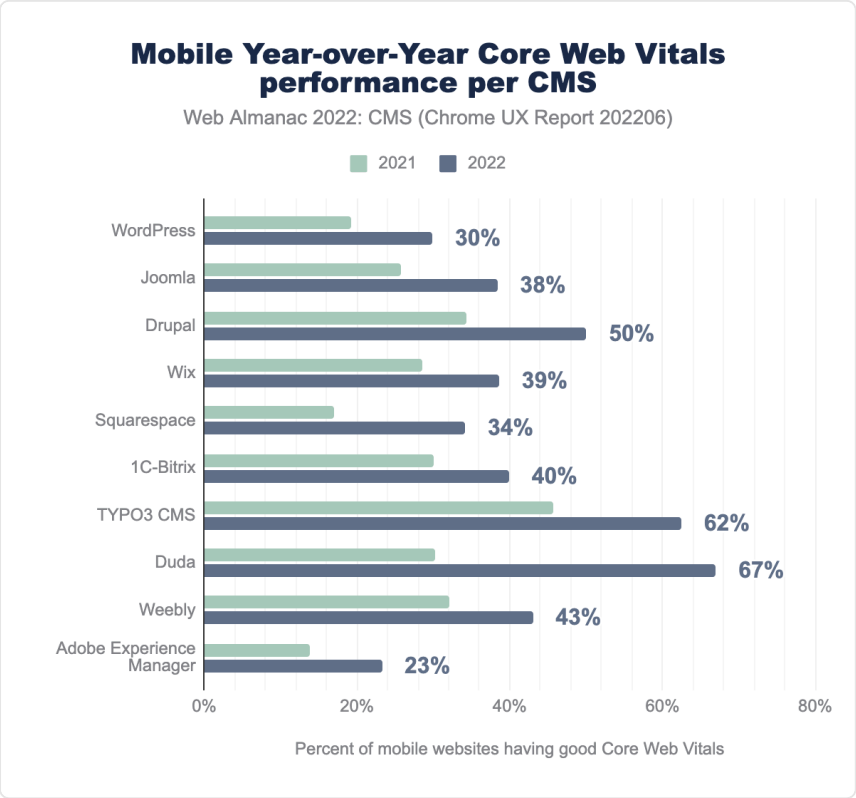



*図18.7. 年ごとのCore Web Vitalsのモバイル合格率。*

これらのCMSはすべて、2021年6月以降、良好なCWVを持つオリジンの割合が改善されています。

次に、3つのCore Web Vitalsを詳しく見て、それぞれのプラットフォームが改善の余地がある箇所と、昨年からもっとも改善された指標を確認しましょう。

### Largest Contentful Paint (LCP)

Largest Contentful Paint（LCP）は、ページの主要なコンテンツが読み込まれた時点を測定し、その時点でページがユーザーにとって有用であると見なされます。LCPは、ビューポート内に表示される最大の画像またはテキストブロックのレンダリング時間を測定することで評価されます。

「良好な」LCPは2.5秒未満とされています。

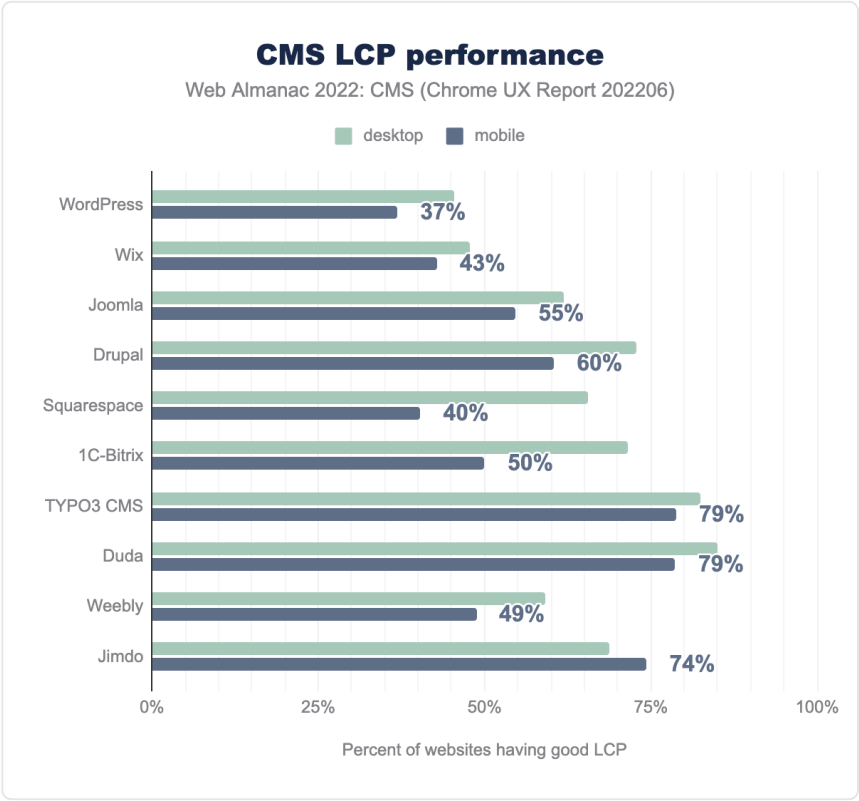



*図18.8. CMS別の良好なLCPを持つサイトの割合。*

TYPO3とDudaはもっとも良いLCPスコアを持ち、起源の79%が「良好な」LCP体験をしています。WordPressとSquarespaceはもっとも悪いLCPスコアを持ち、それぞれ起源の37%と40%が良好なLCPスコアを持っています。

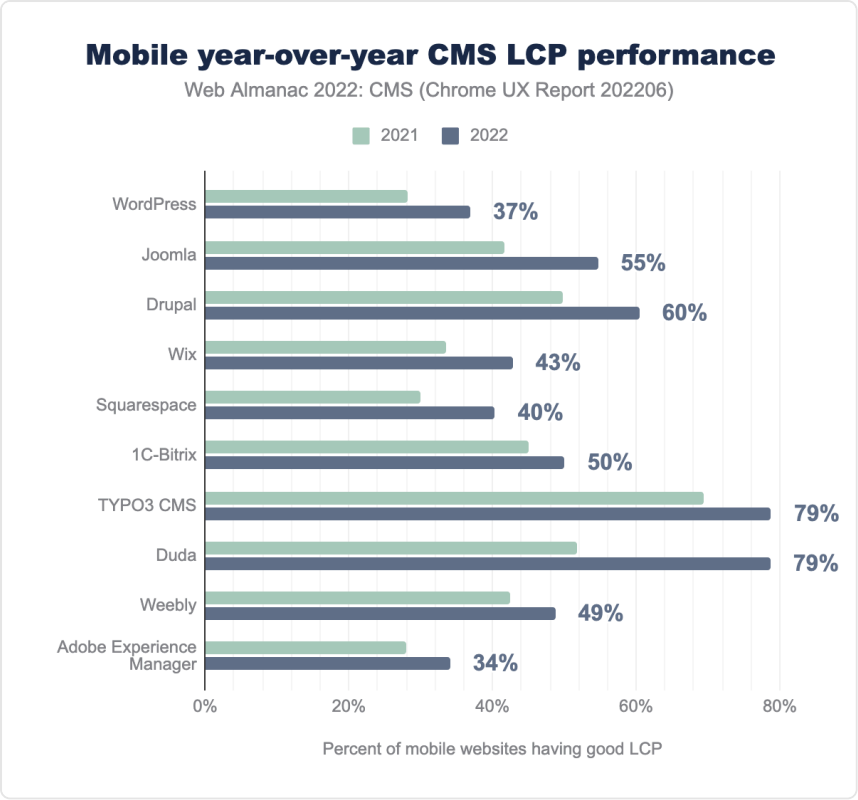



*図18.9. 年ごとのLCPモバイル。*

2021年のデータセットと比較して、すべてのCMSはLCPの向上を示しました。Joomlaは13%改善されました。Drupal、Squarespace、TYPO3は10%改善されました。WordPressは9%改善されました。

これらの改善は、ほとんどのCMSで数字が低いにもかかわらず、前向きな兆候です。良好なLCPスコアを達成することは困難である可能性があり、その理由はLCPが画像/フォント/CSSのダウンロードとそれに続く適切なHTML要素の表示に依存しているからです。すべてのデバイスタイプと接続速度で2.5秒未満でこれを達成することは挑戦的です。LCPスコアの改善には、キャッシング、プリロード、リソースの優先順位付け、および他の競合リソースの遅延読み込みの正しい使用が通常含まれます。

### First Input Delay (FID)

First Input Delay（FID）は、ユーザーがページとはじめてインタラクションする（つまりリ

ンクをクリックする、ボタンをタップする、またはカスタムのJavaScript駆動のコントロールを使用する）時からブラウザがそのインタラクションを処理できるようになるまでの時間を測定します。ユーザーの観点から「速い」FIDは、停止した経験ではなく、ほぼ即座にフィードバックが得られることを意味します。

遅延は痛点であり、ユーザーがサイトと対話しようとする間にサイトの他の部分が読み込まれることと相関がある可能性があります。「良好な」FIDは100ミリ秒未満とされています。

2021年のレポートでは、ほぼすべてのプラットフォームが良好なFIDを提供しているという事実が、この指標の厳しさについて疑問を投げかけました。Chromeチームは記事を公開しました[664]。これは2022年の5月に更新され、新しい指標であるInteraction to Next Paint (INP)[665]に言及しています。この執筆時点でそのベータ版であるため、来年のレポートでの可能な拡張に備えて、この参照に限定しています。

664. https://web.dev/responsiveness/
665. https://web.dev/inp/

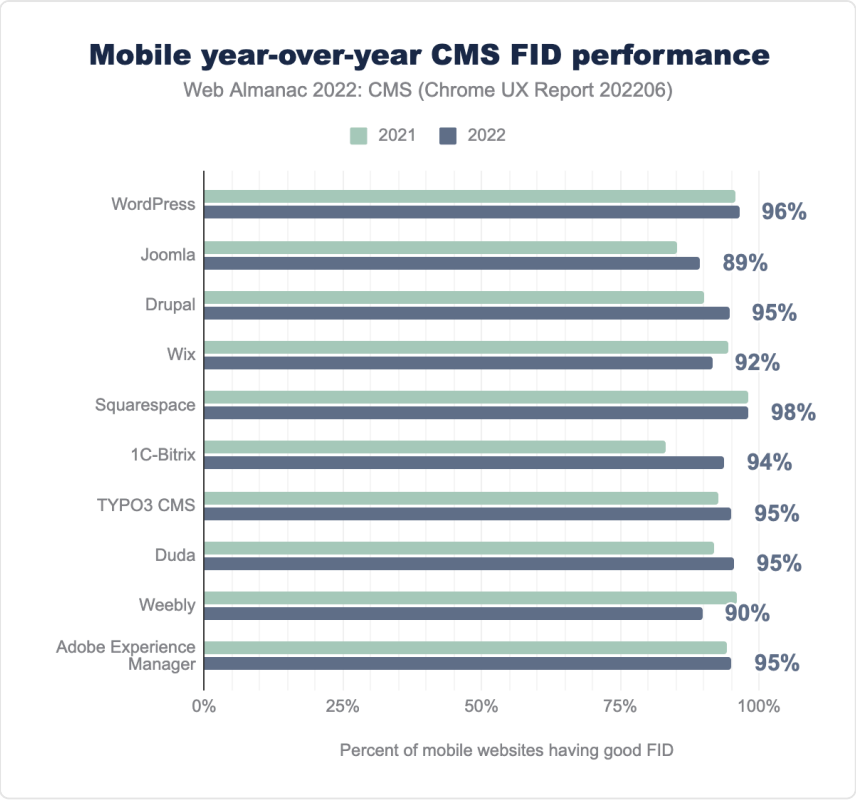



*図18.10. 年ごとのFIDモバイル。*

年次データは、ほとんどのCMSが過去1年間でFIDを改善したことを示しています。WixとWeeblyは前年のデータよりもいくつかのパーセンテージポイント後退しました。

### Cumulative Layout Shift (CLS)

Cumulative Layout Shift（CLS）は、ウェブページ上のコンテンツの視覚的安定性を測定し、ユーザーの直接的なインタラクションによって引き起こされないページのライフスパン中に発生する予期しないレイアウトシフトの最大のバーストを測定します。

レイアウトシフトは、表示される要素が1つのレンダリングフレームから次のフレームへと位置を変えるたびに発生します。CLSメトリックは2021年に進化しました[666]。主に長期間存在するページやシングルページアプリケーション（SPA）に公平であるためにセッションウィンドウの概念が導入されました。

666. *https://web.dev/evolving-cls/*

0.1以下のスコアは「良好」と測定され、0.25以上は「悪い」とされ、その間は「改善が必要」とされます。

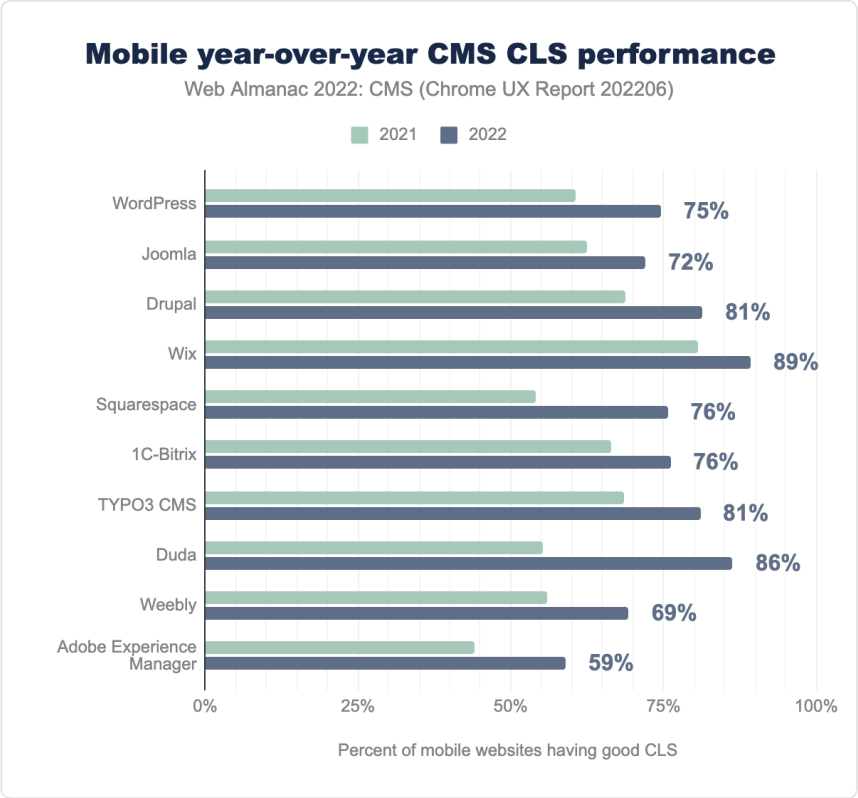



*図18.11. 年ごとのCLSモバイル。*

年次データを比較すると、すべてのCMSが進歩していることがわかります。とくにWordPress、Squarespace、Duda、Adobe Experience Managerは顕著な向上を示しています。

## Lighthouse

Lighthouse[667]は、ウェブページの品質を向上させるためのオープンソースの自動化ツールです。このツールの重要な側面の1つは、パフォーマンス、アクセシビリティ、SEO、ベストプラクティスなどの観点からウェブサイトの状態を評価する一連の監査を提供することです。Lighthouseレポートはラボデータを提供し、開発者がウェブサイトのパフォーマンスを

667. https://developers.google.com/web/tools/lighthouse/

改善する方法についての提案を得る方法ですが、LighthouseスコアはCrUX[668]によって収集された実際のフィールドデータに直接的な影響はありません。ラボスコアとフィールドデータの相関関係[669]については、さらに詳しく読むことができます。

HTTP Archiveは、そのモバイルウェブページでLighthouseを実行しており、遅い4G接続をエミュレートしCPUを遅くする設定もされています。また、今年からデスクトップでも実行を開始しました。

これらの合成テストの結果を使用して、CrUXで追跡されていない指標も含め、CMSのパフォーマンスについて別の視点から分析できます。

### パフォーマンススコア

Lighthouseのパフォーマンススコア[670]は、いくつかのスコア指標の加重平均です。

668. *https://developers.google.com/web/tools/chrome-user-experience-report*
669. *https://web.dev/lab-and-field-data-differences/*
670. *https://web.dev/performance-scoring/*

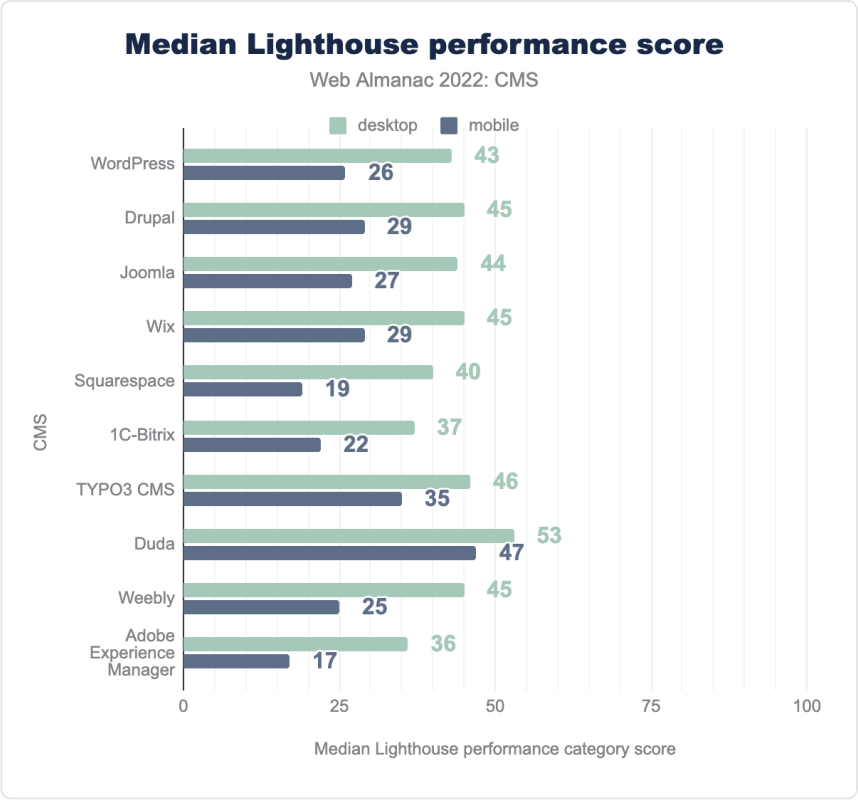



*図18.12. Lighthouseの中央値パフォーマンススコア。*

ほとんどのプラットフォームのモバイルでの中央値パフォーマンススコアは低く、約19から35の範囲です。Dudaの47は例外です。Philip Waltonが2021年に指摘したように、これがモバイルフィールドデータの悪い結果を直接示すわけではありません[671]が、すべてのプラットフォームが改善の余地があり、とくにLighthouseがエミュレートしようとしているようなネットワーク接続を持つ低性能デバイスでの改善が必要です。

WordPress、Joomla、Drupal、1C-Bitrixは昨年の結果から変化はありませんでした。Wixは30%から29%に下がりましたが、他は改善を示しています。

デスクトップスコアは全体的に良好で、すべてのCMSが10から20ポイントの改善を見せています。これは、デスクトップに利用可能なより高速なCPUとネットワークを考慮すると驚くことではありません。

671. *https://philipwalton.com/articles/my-challenge-to-the-web-performance-community/*

### SEOスコア

検索エンジン最適化（SEO）は、ウェブサイトを改善して検索エンジンでより簡単に見つけられるようにする実践です。これについては私たちのSEOの章でより詳しく扱っていますが、CMSとも関連があります。CMSとその上のコンテンツは、検索エンジンのクローラーにできるだけ多くの情報を提供し、検索エンジンの結果でサイトコンテンツを適切にインデックスできるように設定されています。カスタムビルドのウェブサイトと比較すると、CMSは良好なSEO機能を提供すると期待されますが、このカテゴリーのLighthouseスコアは適切に高いです。

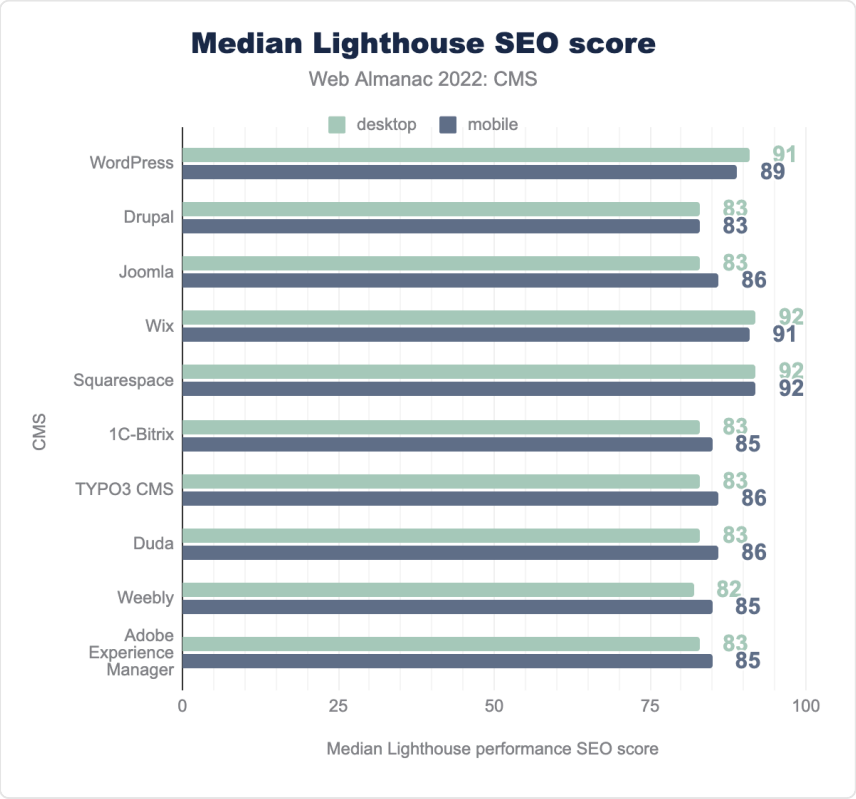



*図18.13. Lighthouseの中央値SEOスコア。*

トップ10のプラットフォームの中央値SEOスコアは83から92の範囲で、2021年の84から95からわずかに減少しました[672]。デスクトップスコアは似ており、いくつかの場合はわずかに良く、他の場合はわずかに悪くなっています。

672. *https://almanac.httparchive.org/ja/2021/cms#SEOスコア*

## アクセシビリティスコア

アクセシブルなウェブサイトは、障害を持つ人々が使用できるように設計および開発されたサイトです。ウェブアクセシビリティは、遅いインターネット接続などの障害を持たない人々にも利益をもたらします。詳細はアクセシビリティ章で読むことができます。

Lighthouseは一連のアクセシビリティ監査を提供し、それらすべての加重平均を返します。各監査がどのように加重されているかの完全なリストについては、スコアリング詳細[673]を参照してください。

各アクセシビリティ監査は合格または不合格ですが、他のLighthouse監査とは異なり、ページがアクセシビリティ監査を部分的に合格してもポイントは得られません。たとえば、一部の要素がスクリーンリーダーに優しい名前を持っているが、他の要素が持っていない場合、そのページはスクリーンリーダーに優しい名前の監査でゼロを取得します。

673. *https://web.dev/accessibility-scoring/*

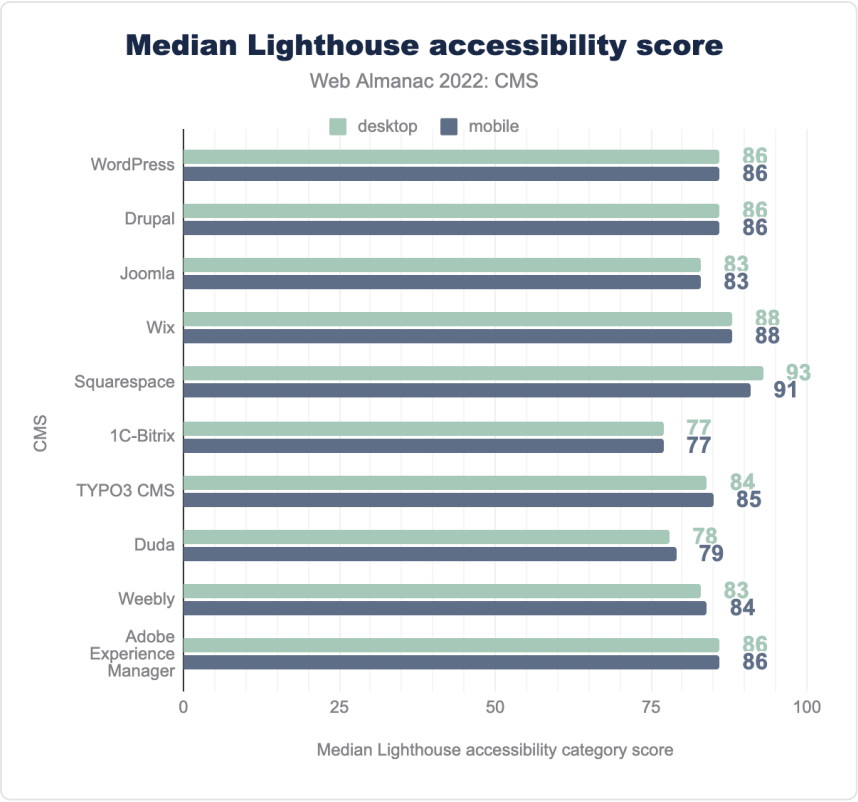



*図18.14. Lighthouseのアクセシビリティスコアの中央値。*

トップ10のCMSのLighthouseアクセシビリティスコアの中央値は77から91の範囲です。Squarespaceが最高スコアの91を持ち、1C-Bitrixが最低のアクセシビリティスコアを持っています。デスクトップスコアはモバイルとほぼ同一であり、デスクトップとモバイルデバイスに同じサイトが配信されていることを反映している可能性があります。

## ベストプラクティス

Lighthouseのベストプラクティス[674]は、HTTPSのサポート、コンソールにエラーが記録されていないことなど、さまざまな指標に対してウェブページがウェブのベストプラクティスにしたがっていることを確認しようとします。

674. *https://web.dev/lighthouse-best-practices/*

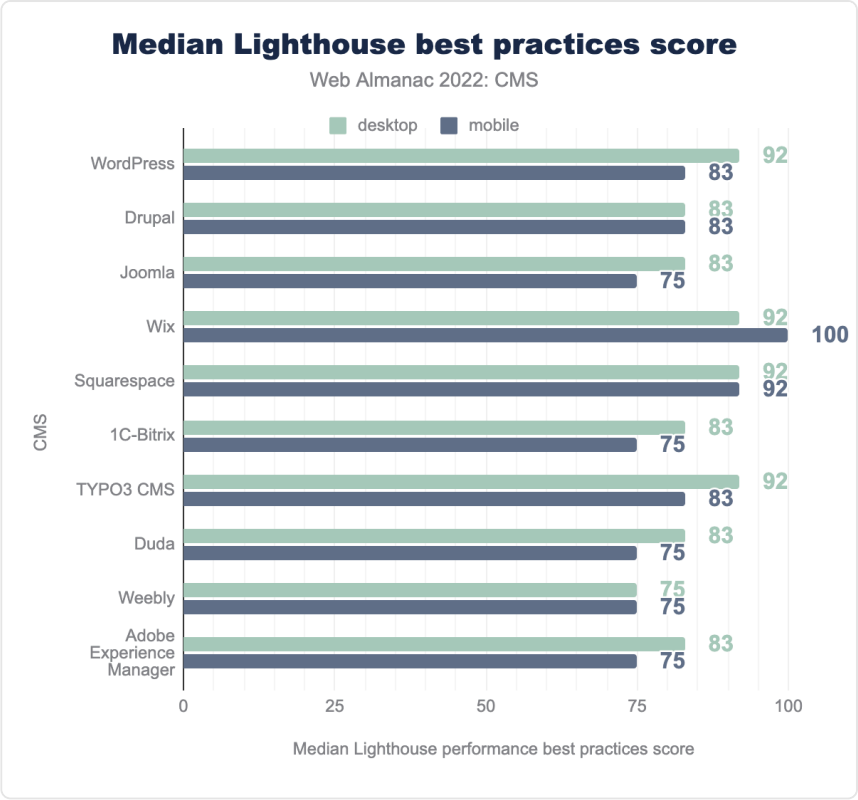



*図18.15. Lighthouseのベストプラクティススコアの中央値。*

Wixは中央値のベストプラクティススコアが100でもっとも高く、他のトップ10のプラットフォームは最低スコアが75を共有しています。再び、デスクトップの結果は非常に似ており、いくつかのケースでは大きくなっています。これは、このカテゴリーの他の監査がプラットフォームベースであるため、モバイルページで画像のアスペクト比が正しくないことを反映している可能性があります。

## リソースの重さ

また、さまざまなプラットフォームで使用されるリソースの重さを分析するためにHTTP Archiveのデータを使用しました。これは、パフォーマンス改善の機会を浮き彫りにするためです。ページの読み込みパフォーマンスはダウンロードされるバイト数だけに依存するわけではありませんが、ページを読み込むために必要なバイト数が少なければコストが削減され、炭素排出量が少なくなり、とくに接続速度が遅い場合にはパフォーマンスが向上する可

能性があります。

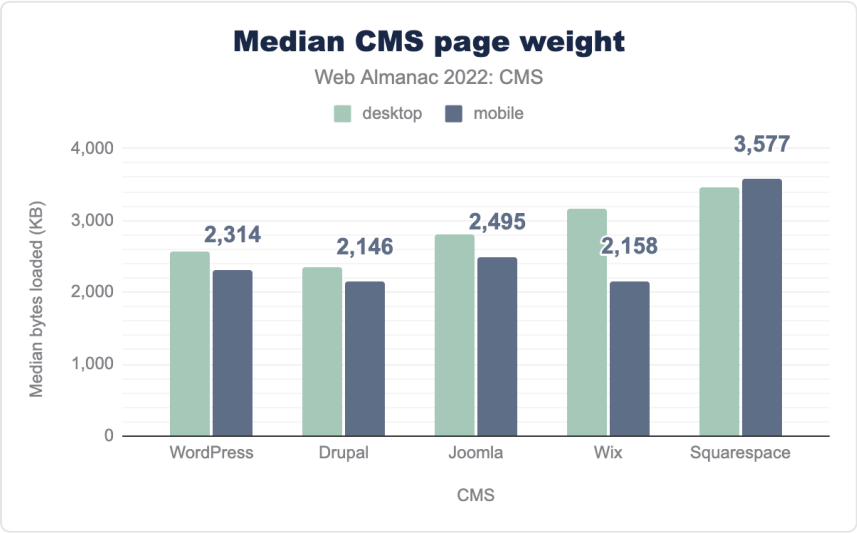



*図18.16. CMS別のリソースの重さの中央値。*

上位5つのCMSのほとんどは、約2 MBの中央ページ重量を提供していますが、Squarespaceは約3.5 MBと大きくなっています。Joomlaを除くすべてが2021年のデータからページ重量の増加[675]を示しています。

675. *https://almanac.httparchive.org/ja/2021/cms#ページ重量の内訳*

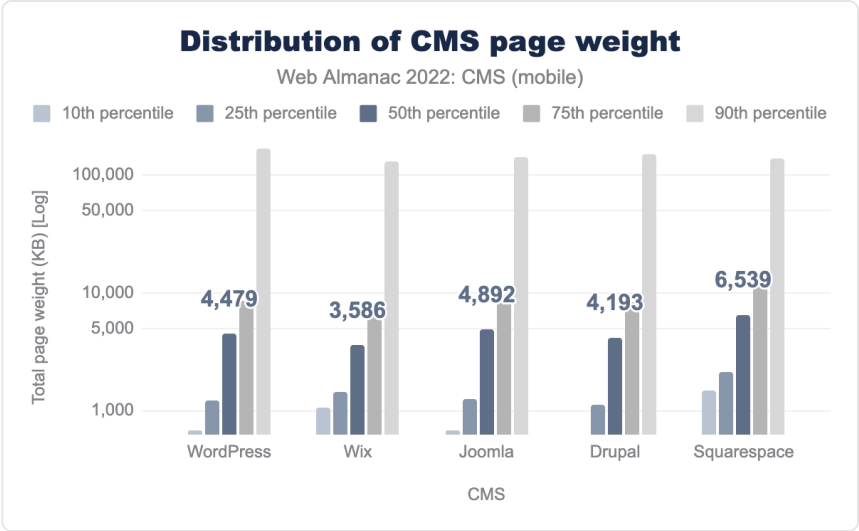



*図18.17. CMS別のモバイルページ重量の分布。*

各プラットフォームのパーセンタイルでのページ重量の分布は著しいです。ページ重量は、ウェブページ間のユーザーコンテンツの違い、使用される画像の数、インストールされたプラグインなどに関連している可能性があります。プラットフォームごとに提供される最小のページはWordPressからで、昨年のデータからの顕著な改善が見られます。今年、WordPressは訪問の10パーセンタイルで673 KBしか送信していません。最大のページはSquarespaceからで、訪問の90パーセンタイルで約11.4 MBを配信しており、昨年のデータから約2 MBの増加です。

## ページ重量の内訳

ページ重量は、ページで使用されるリソースのキロバイトの合計です。異なるCMSでこれらの異なるリソースサイズを評価しようとできます。

### 画像

通常、ウェブページで読み込まれるもっとも重いリソースである画像は、リソース重量の大部分を占めます。

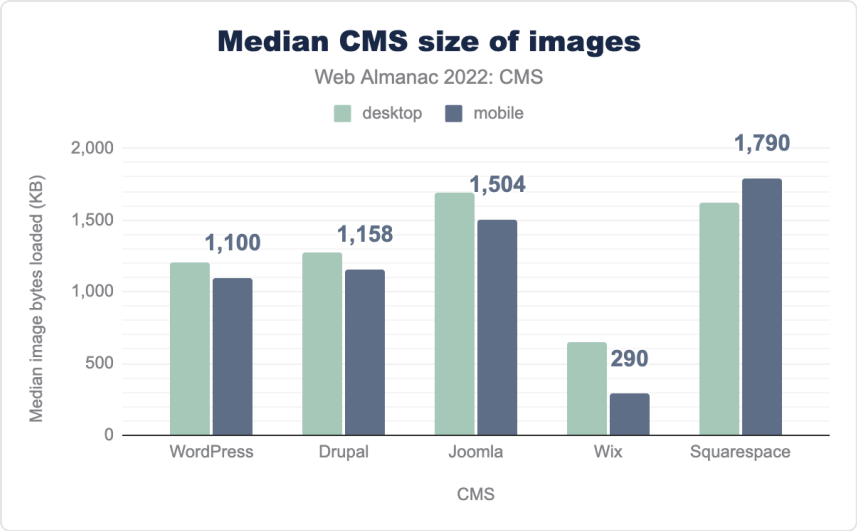



*図18.18. CMS別の画像サイズの中央値。*

Wixはモバイルビューの中央値で290 KBの画像バイトのみを提供し、これは画像圧縮と遅延画像読み込みの良好な使用を示唆しています。他のトップ5のプラットフォームは1 MB以上の画像を提供し、Squarespaceは最大約1.7 MBを提供しています。

進んだ画像フォーマットは圧縮に大きな改善をもたらし、リソースの節約とサイトの高速化を可能にします。WebPは現在、すべての主要ブラウザで一般的にサポートされており、95%以上のサポート[676]があります。さらに、AVIF[677]やJPEG-XL[678]など、新しい画像フォーマットも人気を集めており、採用が進んでいます。

トップCMSでの異なる画像フォーマットの使用状況を調査できます：

676. https://caniuse.com/webp
677. https://caniuse.com/avif
678. https://jpegxl.info

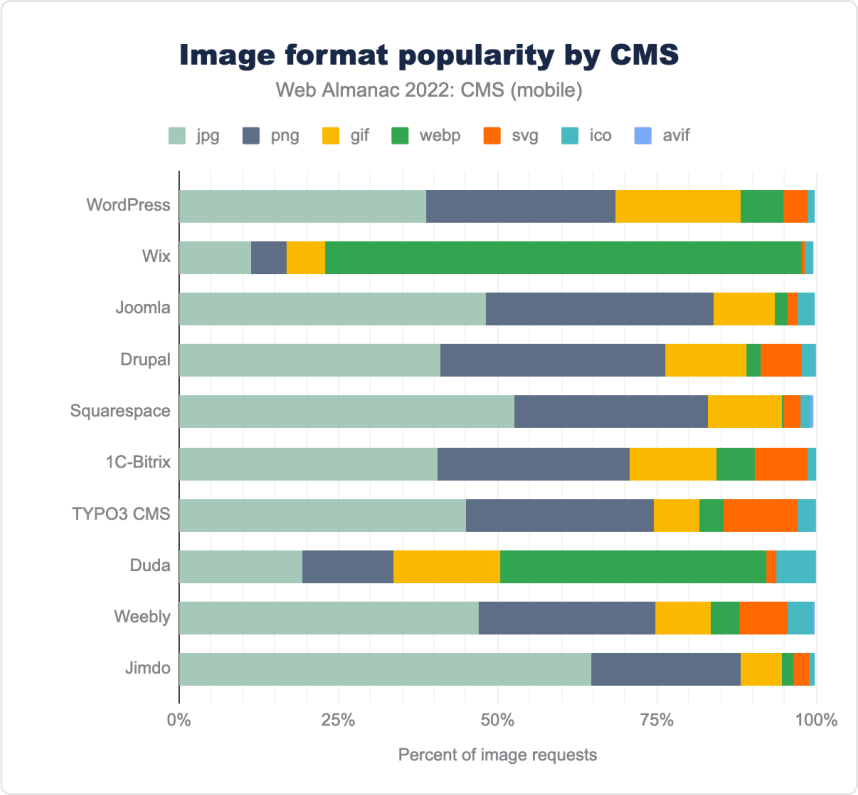



*図18.19. CMS別の画像フォーマットの人気度。*

WixとDudaはWebPの使用がもっとも多く、それぞれ約75%と42%の採用率ですが、他はわずかな増加を示しています。

WebPのサポートが拡大する[679]中で、すべてのプラットフォームは、画像品質を損なうことなく古いJPEGおよびPNGフォーマットの使用を減らすために取り組むべき課題があります。

WordPressは2021年6月にリリースされたWordPress 5.8でWebPサポートを導入しました。WebPサポートはWordPress 6.1でデフォルトで含まれる予定でした[680]が、この決定は延期されました。最終的に、WordPressを通じてWebPの採用が大幅に増加することが期待され、その結果は2023年の結果に明らかになるかもしれません。

679. https://caniuse.com/webp
680. https://make.wordpress.org/core/2022/06/30/plan-for-adding-webp-multiple-mime-support-for-images/

### JavaScript

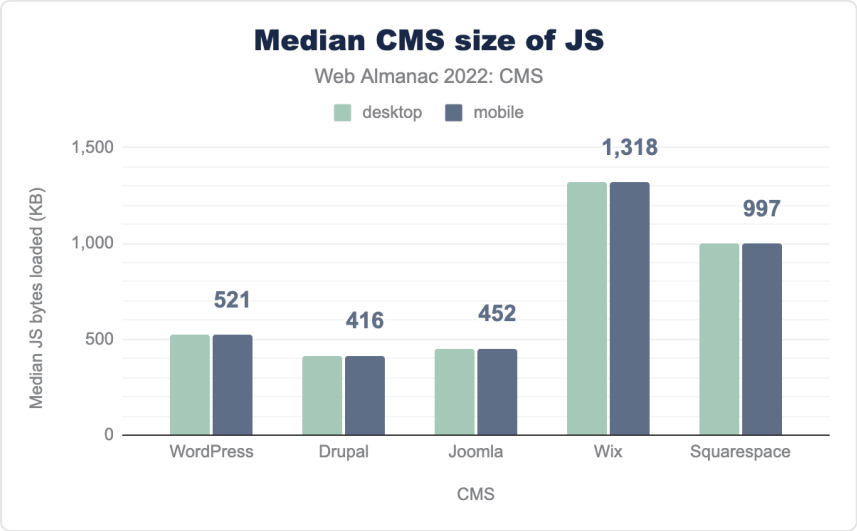



*図18.20. CMS別のJavaScriptリソースの中央値。*

上位5つのCMSはすべてJavaScriptに依存するページを提供しており、Drupalがもっとも少ないJavaScriptバイトを提供しています：モバイルで416 KBです。Wixは1.3 MB以上のJavaScriptバイトを提供し、もっとも多くなっています。

## HTMLドキュメント

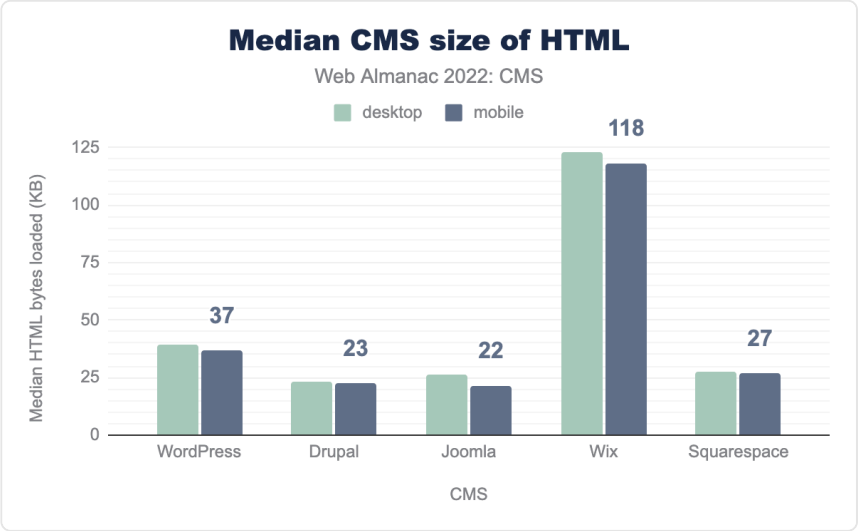



*図18.21. CMS別のHTMLサイズの中央値。*

HTMLドキュメントサイズを調べると、上位のCMSのほとんどが中央値のHTMLサイズを約22 KB～37 KBで提供していることがわかります。例外はWixで、約118 KBを提供しており、2021年の結果に比べてわずかな改善が見られます。これはインラインリソースの広範な使用を示唆しており、さらに改善できる領域を示しています。

## CSS

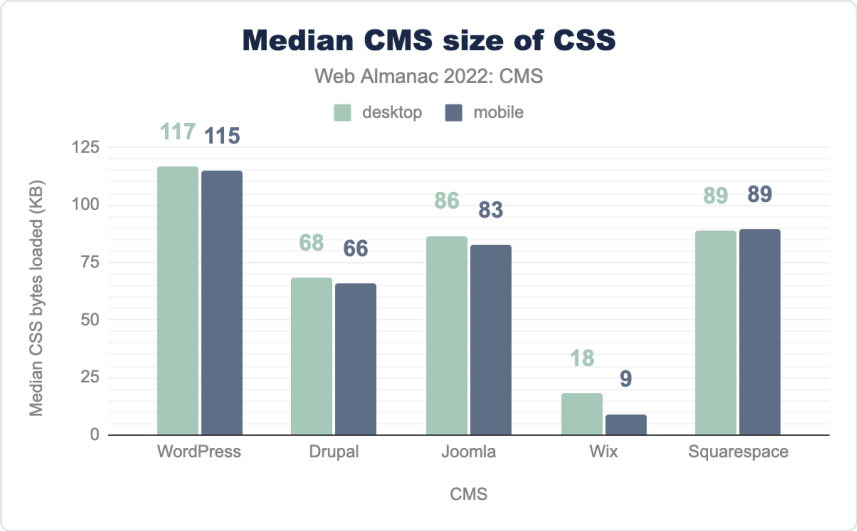



*図18.22. CMS別のCSSサイズの中央値。*

次に、ダウンロードされる明示的なCSSリソースの使用を調査します。ここでは、CSSのインライン化についての議論を強化するプラットフォーム間で異なる分布が見られます。Wixはもっとも少ないCSSリソースを提供し、モバイルビューで約9 KBのみを送信します。WordPressは約115 KBでもっとも多くを提供します。

フォント

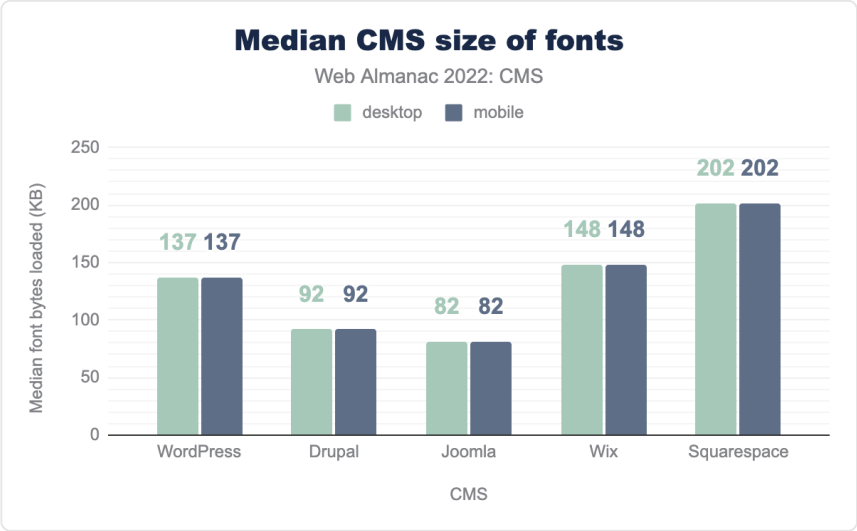



*図18.23. CMS別のフォントサイズの中央値。*

テキストを表示するために、ウェブ開発者はしばしばさまざまなフォントを使用します。Joomlaはモバイルビューでもっとも少ないフォントバイトをSquarespaceはもっとも多く202 KBを提供します。

## 2022年のWordPress

WordPressは現在、もっとも一般的に使用されているCMSです。CMSを使用して構築されたサイトの約3分の2がWordPressを使用しており、これについてさらに議論することが必要です。

WordPressは2003年から存在するオープンソースプロジェクトです。WordPress上で構築された多くのサイトは、Elementor、WP Bakery、Diviなどのページビルダーを通じてさまざまなテーマやプラグインを使用しています。

WordPressエコシステムは、CMSおよび追加機能に必要なカスタムサービスや製品（テーマやプラグイン）を維持しています。このコミュニティは比較的少数の人々によってCMS自体と追加機能の両方が維持されており、この影響は大きいです、WordPressがほとんどの種類のウェブサイトに対応できるほど十分に強力で柔軟であるため、市場シェアを説明する際にはこの柔軟性が重要ですが、WordPressベースのサイトのパフォーマンスについての議論を複雑にもします。

2021年には、WordPressコミュニティの貢献者がパフォーマンスの現状を認め、この提案[681]でパフォーマンスに特化したコアチームを作ることが提案されました。これにより、平均的なWordPressサイトのパフォーマンスが向上することを期待しています。

今年は昨年の結果と比較し、地理的な採用と地理的なCore Web Vitalsの合格率に焦点を当て、平均リソース使用量を見ています。

## 地理別の採用率

まず、私たちは2021年の結果と比較して、データセット内のすべてのサイトを対象に地理的なWordPressの採用を調査しました。

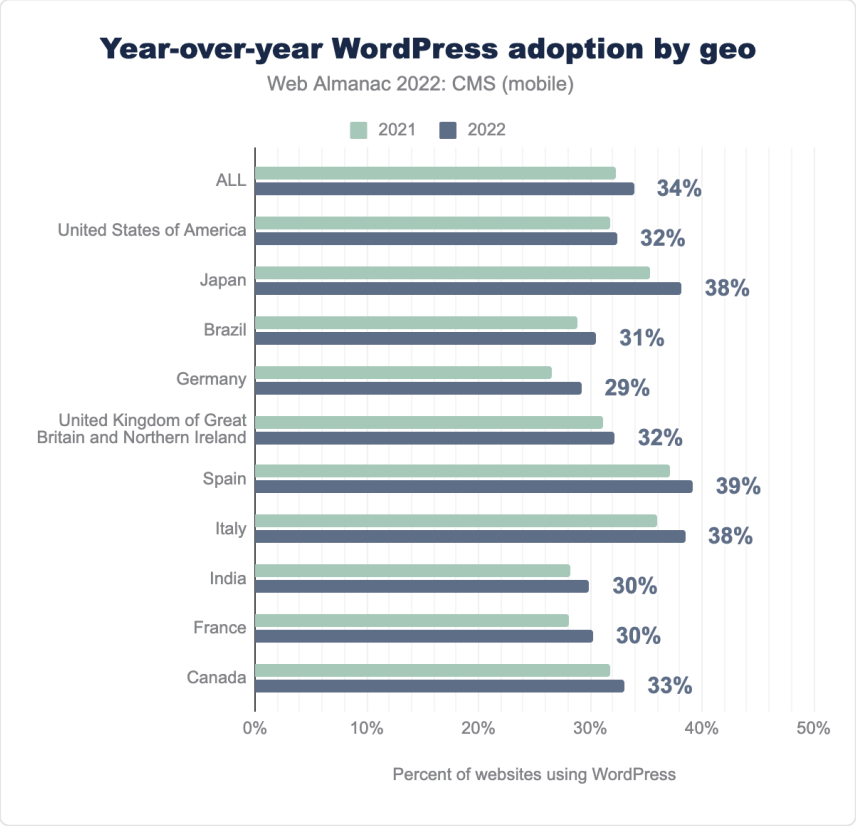



*図18.24. 地理別のWordPressの採用率（年次比較）。*

681. https://make.wordpress.org/core/2021/10/12/proposal-for-a-performance-team/

私たちのデータセットによると、主要な地域でWordPressの採用は著しく増加しています。

### 地理別のCore Web Vitals合格率

次に、地理的な分布、モバイルデバイスの使用、および2021年の結果との比較に基づいて、Core Web Vitalsの合格スコアを持つWordPressのオリジンの数を調べました。

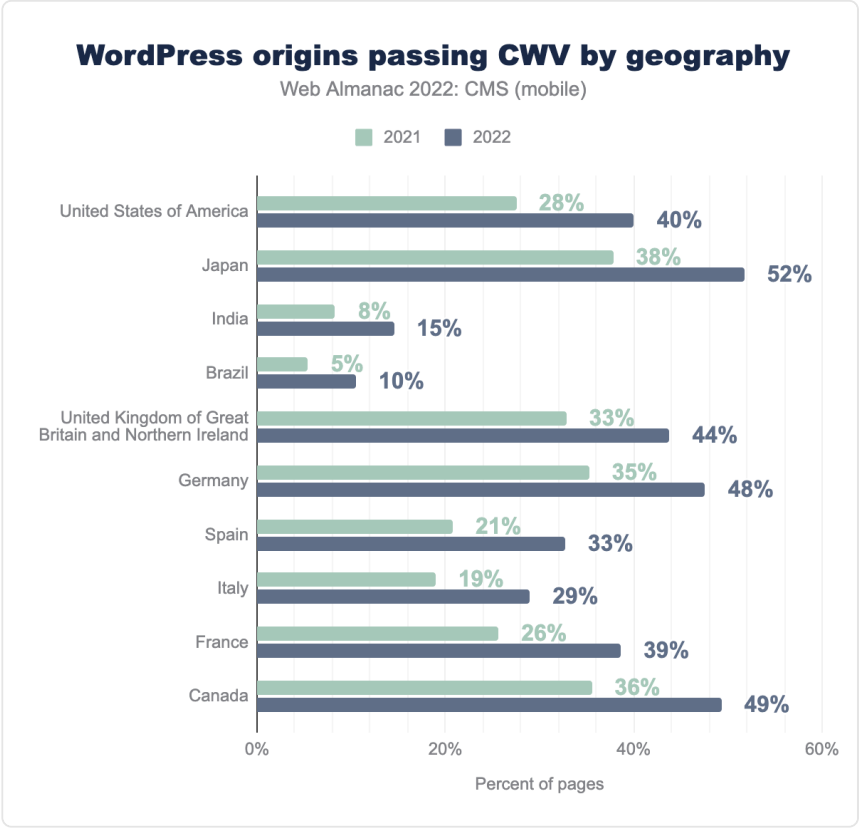



*図18.25. 地理別のWordPress Core Web Vitals（年次比較）。*

すべての地域で改善が見られ、ブラジルの全体的な5%の増加から日本の14%の増加までの範囲です。ブラジルは10%と比較的低いですが、日本は52%と高く、地域間で大きな格差があります。低い側のブラジルは年間で100%の改善が見られ、来年のデータセットを評価する際には、低いパフォーマンスの原因と改善の機会をさらに調査する価値があるかもしれません。

### プラグイン

WordPressサイトが外部リソースをどのように使用しているかを探求し、それらをプラグイン、テーマ、WordPressコア（wp-includes）に含まれるリソースに分けて、2021年の結果と比較しました。

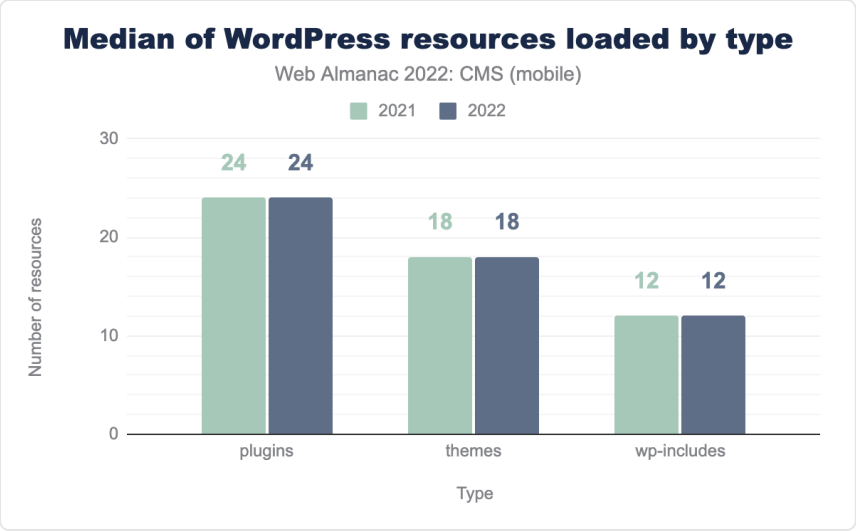



*図18.26. 年次比較のWordPressリソース。*

年を追うごとに使用されるリソースの数に目立った変化はありません。来年も再度調査を行い、人気のあるテーマやプラグインのパフォーマンスへの影響をさらに詳しく見ていくかもしれません。

## 結論

CMSプラットフォームは年々成長し続け、より一般的になっています。インターネット上でコンテンツを作成し消費するためには欠かせないものであり、とくにますます多くの人々やビジネスがオンラインプレゼンスを確立するにつれて、その重要性は増しています。

Core Web Vitalsの導入とパフォーマンスデータの可視性の向上により、ウェブパフォーマンスはCMSが使用されるあらゆる場所で優先事項となっています。この章の洞察が、ウェブの現状をより良く理解するのに役立ち、最終的にウェブをより良い場所にすることを願っています。

CMSは素晴らしい仕事をしており、インフラを強化し、新しい標準との統合や実験を進め、

ベストプラクティスに従うことによって大規模にウェブ上でのユーザーエクスペリエンスをさらに向上させる機会を持っています。

一方で、Core Web Vitalsとしての基準もまだ進化の余地があります。上記で触れたより良い反応性の指標[682]についてのいくつかのアイデアがあります。さらに、サイト内のページ間のナビゲーションは、シングルページアプリケーション（SPA）とマルチページアプリケーション（MPA）[683]のアーキテクチャ間の構造的な違いを考慮して、より適切に追跡されるべきです。

来年の結果を楽しみにしており、データセットを拡大し、方法論を改善することを望んでいます。その間、前進し続け、ウェブをより良くしていきましょう。

### 著者


## Jonathan Wold

@sirjonathan sirjonathan https://jonathanwold.com

Jonathan Woldは、WordPressエコシステムに17年以上取り組んできたオープンウェブの支持者です。WordPressへの愛に加えて、彼は幅広い読書、戦略ゲーム、演技、ロッククライミング、そして時々三人称で文章を書くことを楽しんでいます。


682. *https://web.dev/responsiveness/*
683. *https://web.dev/vitals-spa-faq*

# 部 III 章 19
# Jamstack

***Laurie Voss と Salma Alam-Naylor によって書かれた。***
*Barry Pollard によってレビュ。*
*Laurie Voss と Barry Pollard による分析。*
*Abel Mathew 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

Jamstackについて書く際の最大の問題の1つは、Jamstackが具体的に何であるかを定義することです。ここに3つの異なる定義があります（いくつかの単語を強調しました）。

1. ファイルを事前にレンダリングしてCDNから直接配信することにより、Webサーバーの管理や運用の必要をなくし、迅速かつ安全なサイトとアプリを提供します。

2. Jamstackは、Webをより **高速** に、より安全に、スケールしやすくするために設計された **アーキテクチャ** です。開発者が愛用する多くのツールやワークフローを活用し、最大限の生産性を引き出します。

3. Jamstackは、Web体験レイヤーをデータやビジネスロジックから **切り離す** という **アーキテクチャアプローチ** であり、柔軟性、スケーラビリティ、**パフォーマ**

ンス、および保守性を向上させます。

上記の定義はすべてJamstack.org[684]からのもので、それぞれ2020年[685]、2021年[686]、2022年[687]の定義です。Jamstackの定義に関してより権威ある情報源は考えにくいので、定義が移り変わるものであると言えます。

しかし、強調された単語が示すように、明らかに一貫性があります。サイトは迅速であるべきで、事前にレンダリングされ、データを取得する場所とデータをレンダリングする方法を切り離すアーキテクチャアプローチを使用すべきです。正確な辞書の定義は得にくいですが、Jamstack開発者は「Jamstack」と言うと何を意味しているかを理解しています。それは、非常に迅速にロードされ、ビルド時に一度多くの有用なコンテンツをレンダリングし、必要に応じてJavaScriptを介して追加データを取得するサイトです。

*開示事項: このレポートの二人の著者はNetlifyの社員でした。NetlifyはJamstackという用語を発明し、Jamstack.orgを所有しています。このレポートとその基になる分析は、Netlifyと無関係の他者によってレビューおよび承認されました。*

## Jamstackを定量化する: 何をカウントすべきか？

しかし、2022年のWeb Almanacをまとめようとするとき、問題はもっと複雑になります。何百万ものWebサイトを扱う際、「見たらわかる」という定義では不十分です。私たちはJamstackをどのように定量化すべきでしょうか？どのようにして正確に識別して、それについて学ぶことができるでしょうか？まず、自分たちに一連の質問を投げかけて始めました。

### すべての静的サイトはJamstackサイトですか？

それは明らかに「はい」と思われるかもしれません：もしページが一度にすべてレンダリングされるフラットなHTMLであれば、確かにJamstackのように聞こえます。しかし、JavaScriptが流行する前の90年代に人々が構築したすべてのページはどうでしょうか？それらはJamstackですか？そう感じられませんでした。すべての静的サイトがJamstackサイトというわけではありません。その理由を考えてみました。

私たちはJamstackの初期定義の「CDN」の側面に着地しました。特定のCDNプロバイダーである必要はありませんが、定義の確かな部分は「事前レンダリング」であり、これは何かをレンダリングしてからキャッシュすることを意味します。したがって、Jamstackサイトはキャッシュされるべきです（自分でキャッシュするか、CDNを使用するかは関係ありません）。

684. https://jamstack.org/
685. https://web.archive.org/web/20200331214426/https://jamstack.org/
686. https://web.archive.org/web/20210924115533/https://jamstack.org/what-is-jamstack/
687. https://web.archive.org/web/20220809174445/https://jamstack.org/

これにより別のエッジケースが生じます。多くのサイトがキャッシュされています！完全に動的なサイトでさえ、負荷のピークを避けるために数分間物をキャッシュすることがよくあり、現在では多くのサイトがCDNによって提供されていますが、これはJamstackのパターンには何も借りていないサイトのアーキテクチャであっても本質的にキャッシュです。通常のキャッシュサイトとJamstackサイトの違いは何でしょうか？キャッシュ可能性は一部ですが、他には何がありますか？

## JamstackサイトはJavaScriptを使用する必要がありますか？

Jamstackサイトが必ずしもJavaScriptを使用するわけではないと決めました。もちろん、多くのJamstackサイトは使用します。結局のところ、「J」は元々「JavaScript」の略でした。しかし、Jamstackの最初の定義でも、JavaScriptの使用は任意であり、JavaScriptがなく完全に静的なサイトも常にJamstackでした。

## Jamstackを使用すると特定のフレームワークが必要ですか？

Jamstackについて考えるとき、人々が思い浮かべるいくつかのフレームワークが確かにあります。2020年と2021年のWeb Almanacでは、Jamstackを使用するフレームワークだけで定義していました[688]。これにはStatic Site Generators（SSG）に焦点を当てていました。これは十分に理にかなっていますが、いくつかの問題があると思いました。

まず、簡単に検出できないフレームワークはどうでしょうか？例として、Eleventy[689]は、Jamstackで成長して人気のある選択肢であり、生成されたHTMLには跡形もありません。それはデフォルトではエンドユーザーには見えませんが、必要に応じてgenerator tag[690]を追加できます。礼儀正しく邪魔にならないフレームワークを数えないのは間違っているように思えます。

第二にフレームワークが多すぎます！大きなものが何十個もあり、小さなものが数千個あります。検出できるものについても、常に良い検出方法があるわけではありません。さらに、フレームワークをまったく使用せずに「Jamstack的」に感じるサイトを構築することも確かに可能です。プレーンHTMLは確かにJamstackです。

第三にJamstack開発者に人気のあるフレームワークを使用しても、構築するサイトがJamstackサイトになるとは限りません。建築上の理由でレンダリングが非常に遅い場合や、すべてのページを動的にレンダリングする場合は、多くのJamstackサイトが使用している同じフレームワークを使用していても、Jamstackサイトにはなりません。すべてのサイトがJamstackである必要はありませんし、それで問題ありません。

688. https://almanac.httparchive.org/ja/2021/jamstack
689. https://www.11ty.dev/
690. https://www.11ty.dev/docs/data-eleventy-supplied/#eleventy-variable

したがって、今回はフレームワークを定義の一部として無視することにしましたが、あとで見るように、予想されるフレームワークが結果に現れました。

### Jamstackサイトはパフォーマンスが高い必要がありますか？

Jamstackの3つの定義に共通している唯一の特徴は **パフォーマンス** でした。しかし、「速い」という言葉はWebサイトに関してはあいまいです。Web Almanacを読んだことがあれば、Webサイトのパフォーマンスを測定するために使用できる多くの指標があることを知っているでしょうし、それらすべてについての良い議論がたくさんあります。

したがって、Jamstackサイトの感じ方について真剣に考えました。最初に、Jamstackサイトは初期コンテンツを非常に迅速にレンダリングします。使用できる指標がたくさんある中で、その考えをもっとも明確に捉えたのは、Largest Contentful Paint[691]、またはLCPでした。これは、ページ上の最大のアイテムがレンダリングされるまでの時間を測定します。APIを介して追加のコンテンツを取り込むことも、しないこともありますが、Jamstackであるためには、何か重要なものを迅速にレンダリングする必要があります。

## 指標の定義

HTTP Archiveでクエリとして表現できるJamstackの正確な定義を選んだ方法の詳細に興味がない場合は、次の2つのセクションをスキップして安全にJamstackの成長まで進むことができます。ここで行動に移す洞察を得るために、私たちの方法論を理解することは重要ではありません。

私たちは測定したいことを知っていました。ほとんどのコンテンツを非常に迅速にロードし、キャッシュ可能なサイトです。これらのことを測定するさまざまな方法を多く実験した後、いくつかの具体的な指標を思いつきました。

**Largest Contentful Paint (LCP)**：すべてのページのすべてのLCP時間の分布を取得し、Chrome UX Report[692]から実際のユーザーデータの中央値を選び、「もっともコンテンツを迅速にロードした」とみなされる任意のサイトとしました。これはモバイルデバイスで2.4秒、デスクトップデバイスで2.0秒でした。

**Cumulative Layout Shift (CLS)**：非常に迅速にスケルトンをロードするが、実際のコンテンツのロードに長い時間がかかるサイトを避けたいと思いました。それにもっとも近いものはCumulative Layout Shift[693]で、ページのレイアウトがロード中にどれだけ跳ねるかを測定します。CLSを「操作」する方法がありますが、私たちはそれが測定しようとしているものの合理的な代理であるとまだ信じています。私たちはこの測定が好きでした、なぜなら「跳ね

691. *https://web.dev/lcp/*
692. *https://developer.chrome.com/docs/crux*
693. *https://web.dev/cls/*

る」サイトもまた「Jamstack的」ではないと感じられるからです。「Jamstack的」という言葉を多く使うことになるでしょう。再び、Chrome UX Reportのデータの中央値を選びました。

*Chrome UX ReportのデータはCLSデータをもっとも近い0.05に丸めますが、それは残念なことです。なぜなら「実際の」中央値は約0.02-0.03のようで、モバイルではゼロに丸められ、デスクトップでは0.05に丸められるからです。0は膨大な数のページを除外するため、私たちはモバイルとデスクトップの両方に最適な閾値として0.05を使用することにしました。*

**キャッシュ**：これはとくに定量化が難しいものでした。なぜなら、Jamstackサイトであっても、ほとんどのホームページが実際には長時間キャッシュされているにもかかわらず、再検証を要求するからです。私たちは`Age`、`Cache-Control`、`Expires`を含むHTTPヘッダーの組み合わせを使用しました。これらは長時間キャッシュされる可能性のあるページで一般的に見られました。

当初、私たちは「小さな」サイトを除外するための別の測定が必要だと思っていました。つまり、実際のところ誰も訪れない「まもなく公開」や「Hello world」ページを非常に迅速にロードするサイトです。しかし、HTTP ArchiveのデータはChromeのユーザー訪問による人気[694]に基づいて定義されており、それらのサイトがサンプルに入るほど十分に訪問されるものはほとんどありません（ただしexample.comは入っています！）。

良い質問は、これらの指標にCore Web Vitals[695]（CWV）の数値を使用しない理由は何かということです。LCPについては、私たちの数値はCWVの数値とほぼ同じです。CLSについては、CWVチームが要件を緩和しました[696]（彼らの閾値は中央値の2倍以上です）が、私たちはそれがJamstack体験を代表していないと考えました。したがって、両方に中央値を選ぶのが公平だと決めました。そしてCWVには「キャッシュ可能性」の指標はありません。

## “Jamstack的”免責事項

はっきりさせておきたいのは、これは「Jamstack」の非常に非常に曖昧な定義です。実際、この作業を行っているときに「Jamstack的」という言葉を使い始めました。

最大の誤差の源泉は基本的なものです。Jamstackの定義はアーキテクチャに関するものですが、アーキテクチャは生成されたHTMLをクロールすることで、非常に広い意味でしか判断できません。HTMLの山を見て、フロントエンドとバックエンドが切り離されているかどうかを判断する方法は単純にありません。したがって、私たちの測定は最善の努力であり、大まかな見積もりであり、それを誤解されたくありません。

この方法論はJamstackサイトを過少評価も過大評価もします：

694. *https://developer.chrome.com/docs/crux/methodology#popularity-eligibility*
695. *https://web.dev/vitals/*
696. *https://web.dev/defining-core-web-vitals-thresholds/#achievability-3*

- あなたのサイトが静的であるが切り離されていない場合（たとえば、SquareSpace[697]とWix[698]のサイトは明らかにバックエンドと密接に結びついています）、パフォーマンスが良ければ過大評価されます。
- あなたのJamstackサイトが切り離されているが、コンテンツを非常に頻繁に更新する場合、キャッシュ戦略が私たちが求めるものと異なるかもしれないので、過少評価されます。
- あなたのJamstackサイトが切り離されているが非常に遅くレンダリングされる場合、LCP数値が高くなり、過少評価されます。
- 逆に、あなたの非Jamstack動的サイトが非常に速い場合、それをJamstackと誤って判断し、過大評価するかもしれません。

これらすべての注意事項にもかかわらず今年の「Jamstack的」サイトの見積もりは、厳密にSSGに焦点を当てた定義を改善しており、結局のところAlmanacの目標である、今日のWebが実際にどこにあるのかについてのより良い感覚を与えます。

## Jamstackの成長

新しい基準を適用して、HTTP Archiveで「Jamstack」として分類されるサイトの割合を測定しました。2020年と2021年の測定方法が非常に異なっていたため、2022年の定義を使用してそれらのサンプルを再測定しました。

697. *https://www.squarespace.com/*
698. *https://www.wix.com/*

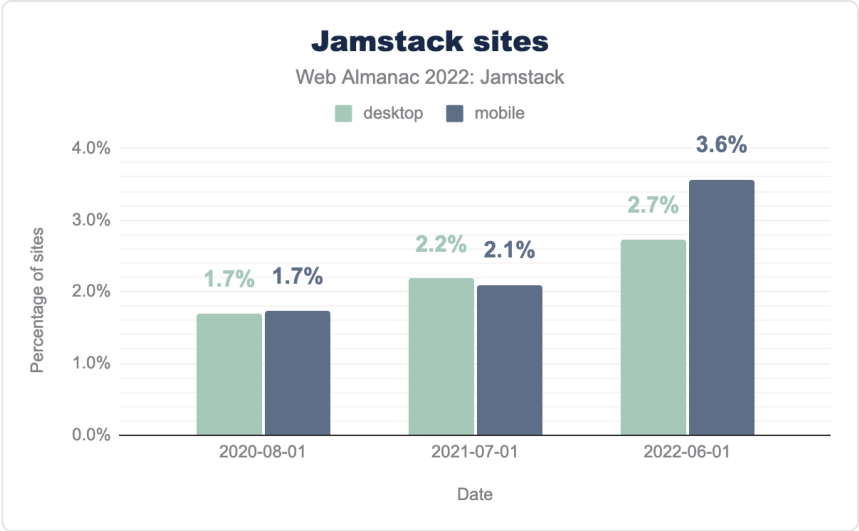



*図19.1. Jamstackサイト。*

主な発見は、2022年にはモバイルWebサイトの3.6%が「Jamstack的」であり、2020年以来100%以上成長していることです。デスクトップでは2.7%のサイトがJamstack的であり、この数値も成長していますが、この2つのグループ間の違いは主にデバイスによるレイアウトの違いによってCLS閾値を満たすサイトの数が異なるためです。再び、これがどれほど近似的であるかについての多くの注意事項を上で見てください。

新しい定義での歴史的な数値は、昨年測定されたものよりかなり高い[699]です。これは使用されている技術に基づいて採用を完全に判断したときのことです。Jamstackと見なされていなかった技術も含む一部の技術を検出する限界を考慮すると、これはおそらく驚くべきことではありません。

私たちが無作為に選んだサイトのセットをサンプリングしたとき、私たちが判断できる限り、見た目や感じがJamstackサイトのように思えるサイトを見つけていることにほとんど満足していました。

自分自身で判断し、私たちを正直に保つために、ここに完全に無作為に選ばれた10の「Jamstack的」サイトがあります。どのようなキュレーションも行っていません。

- www.cazador-del-sol.de
- snpbooks.org[700]

699. *https://almanac.httparchive.org/ja/2021/jamstack#SSGの採用*
700. *https://snpbooks.org/*

- eikounoayumi.jp[701]
- rochesteronline.precollegeprograms.org[702]
- surveyforcustomers.com[703]
- www.shopmate.eu
- docs.saleor.io[704]
- www.wildeyebrewing.ca
- 360insurancegroup.com[705]
- 144onthehill.co.uk[706]

Webの3.6%（またはデスクトップの2.7%）が正確にJamstackであるかどうかは、Jamstackの定義が検証できないアーキテクチャの選択に依存しているため、私たちは確実に言うことはできませんが、Jamstackが成長していることは確かです。私たちの基準を満たすサイトの割合は年々着実に増加しています。Webはより「Jamstack的」になっています。

もちろん、私たちの定義が2つのパフォーマンス指標とキャッシング指標であるため、間違っている可能性の1つは、Webが一般的によりパフォーマンスが向上している場合です。それを確認するために、指標を再び分割しました（これはモバイルデータです。デスクトップデータは大きく異なりませんでした）。

701. *https://eikounoayumi.jp/*
702. *https://rochesteronline.precollegeprograms.org/*
703. *https://surveyforcustomers.com/*
704. *https://docs.saleor.io/*
705. *https://360insurancegroup.com/*
706. *https://144onthehill.co.uk/*

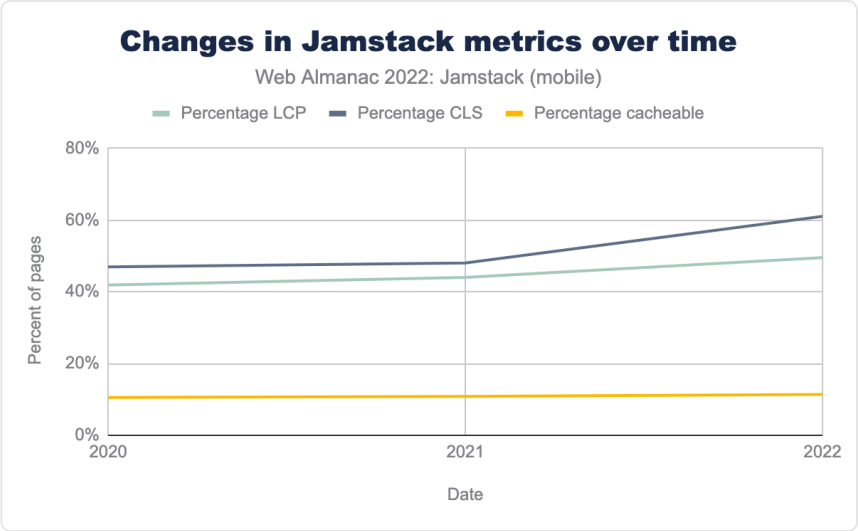



*図19.2. モバイルでの時間の経過とともにJamstack指標の変化。*

ご覧のとおり、2020年から2022年までの私たちの指標にはわずかな改善がありました。しかし、ここでもっとも小さな数値である「私たちのキャッシング基準を満たすサイトの割合」は、年とモバイルまたはデスクトップを問わず、Webの11-14%です。Jamstackサイトの私たちのセットはこれらのグループの交差点です。これら3つの基準を同時に満たすサイトのセットは、個々のグループよりもはるかに小さいです。

したがって、私たちは本当にサイトの独特なサブセットを見ており、そのセットはWeb全体のパフォーマンスの改善よりもはるかに速く成長しています。私たちは単に「どれだけの高速サイトがあるか」を測定しているわけではありません。

## Jamstack-y フレームワーク

Jamstack-yサイトが実際に存在し、識別可能であることを確認したので、これからいくつかの質問をしてみましょう。ここからが楽しいところです。Jamstack-yの定義にフレームワークが含まれていないため、私たちはサイトを見て、Jamstackでもっとも頻繁に現れるフレームワークを見ることができます。

Wappalyzerが提供するフレームワークの識別を使用しましたが、これには（以前に言及したように）Eleventyのような「見えない」フレームワークは含まれておらず、数えたり分析したりすることはできません。

Wappalyzerには、「ウェブフレームワーク」と「JavaScriptフレームワーク」というやや任

意的な区別があります。こちらが全体のウェブでトップ10のJavaScriptフレームワークです。

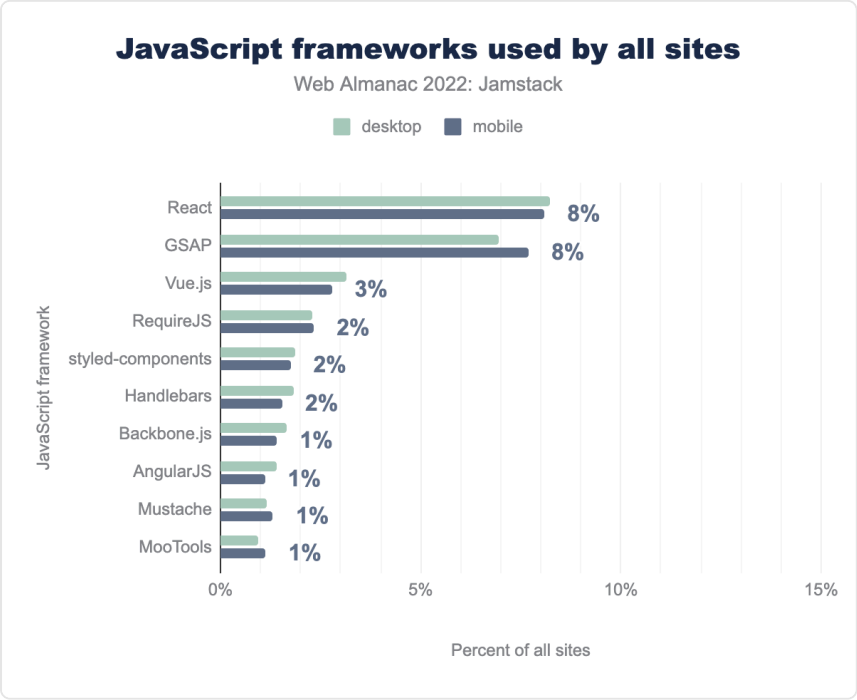



*図19.3. 全サイトで使用されているJavaScriptフレームワーク。*

Jamstackサイトでのトップ10は以下の通りです。

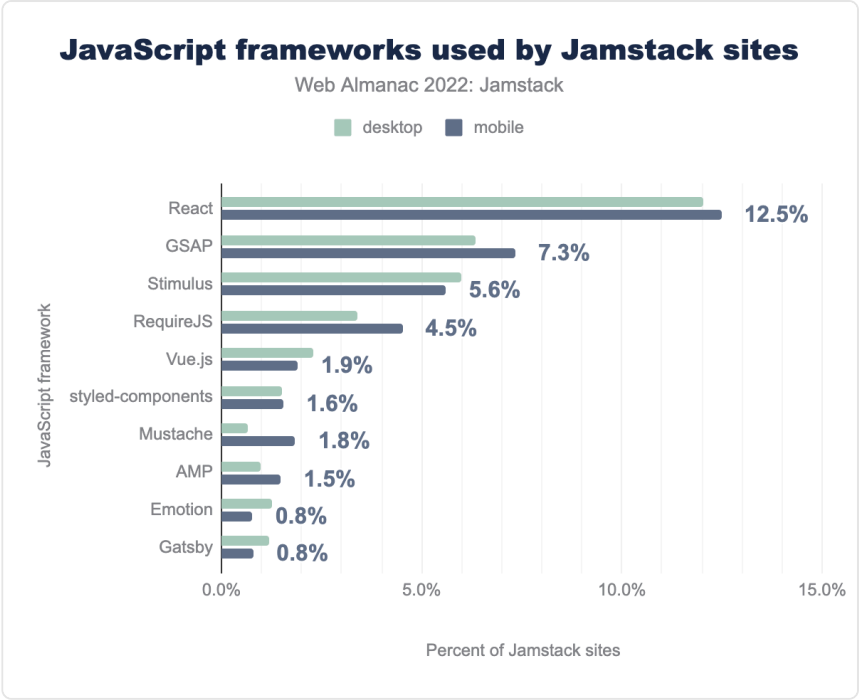



*図19.4. Jamstackサイトで使用されているJavaScriptフレームワーク。*

一般的なウェブと比較して、JamstackではReactやGatsbyがより人気があることがわかります。次に、「ウェブフレームワーク」について見てみましょう。これもWappalyzerによってやや任意的に定義されています。

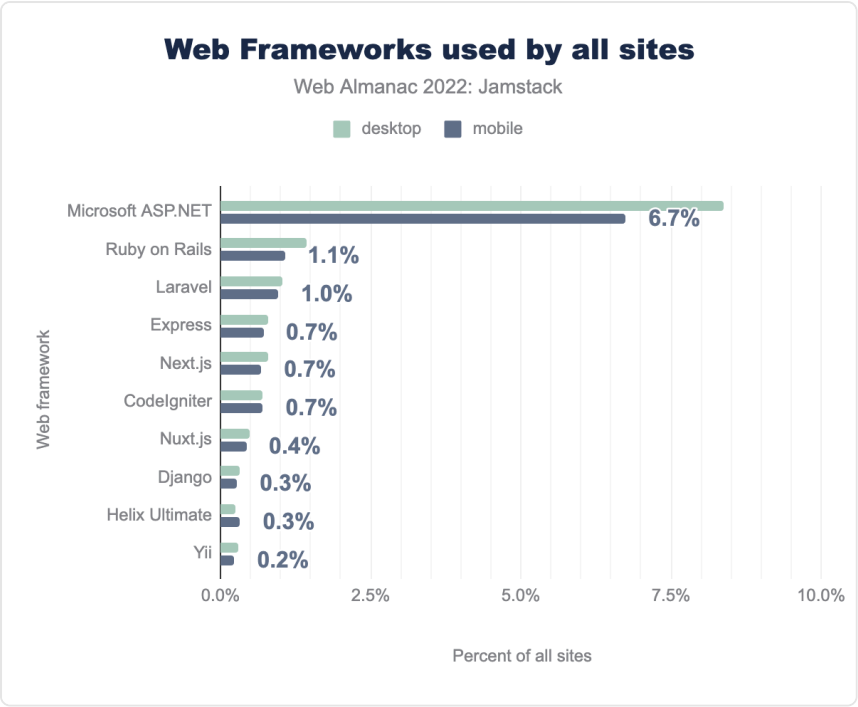



*図19.5. 全サイトで使用されているウェブフレームワーク。*

質問はおもしろいです。Next.jsとNuxt.jsがウェブフレームワークとみなされ、Vue.jsとReactはJavaScriptフレームワークとみなされる理由は何でしょうか？さておき、MicrosoftのASP.Netフレームワークがweb全体で非常に人気がありますし、Ruby on Railsも同様です。Jamstackではどうでしょうか？

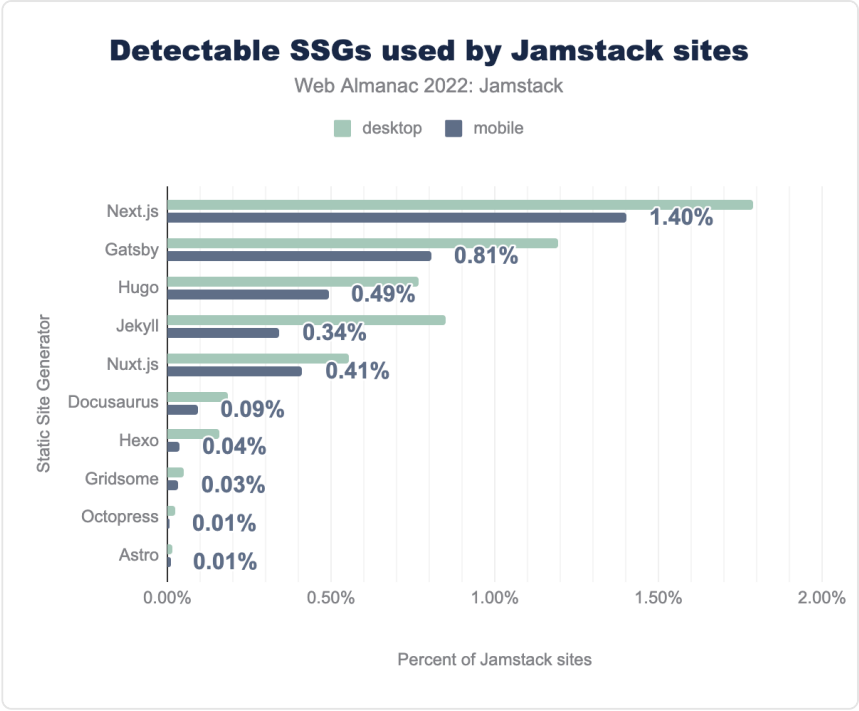



*図19.6. Jamstackサイトで使用されているウェブフレームワーク。*

ご覧の通り、JamstackではASP.Netが依然として第一のウェブフレームワークですが、Ruby on Railsはそれほど人気がありません。代わりにJamstackのお気に入りであるNext.jsが第五位から第三位に、Nuxt.jsは第七位から第五位に上昇しました。驚くべき追加はSymfonyで、一般的なサイトでは11位ですが、私たちのJamstackセットでは第二位まで上昇しました。

Next.jsとNuxt.jsがJamstackコミュニティでもっとも大きなフレームワークであることから、これは大きな驚きではありませんが、フレームワークに依存しない定義が「Jamstack-y」サイトを正しく特定しているのを見るのは再び良いことでした。

一見すると、ASP.NetがまだJamstack-yグループで第1位であり、PHPベースのSymfonyが第2位になるのはさらに驚くかもしれません。しかし、ASP.NETやPHPを使用しても、パフォーマンスの高い、現代的なサイトを構築する理由はありません。Jamstackは特定の技術リストではなく、建築的アプローチですので、流行に乗っていない技術スタックで働く人々にとっては励みになる結果だと思います。

SSGについてはどうでしょうか？Wappalyzerはそれらを別のカテゴリとしています。ここに両方のサイトでの数値があります（注：私たちはNuxt.jsとNext.jsを手動でこのリストに追加しました。WappalyzerはそれらをSSGとはみなしませんが、どちらもそのように使用するこ

とができるので、それらを考慮するのは有用だと思いました）。全サイトに対するそれらの数値です。

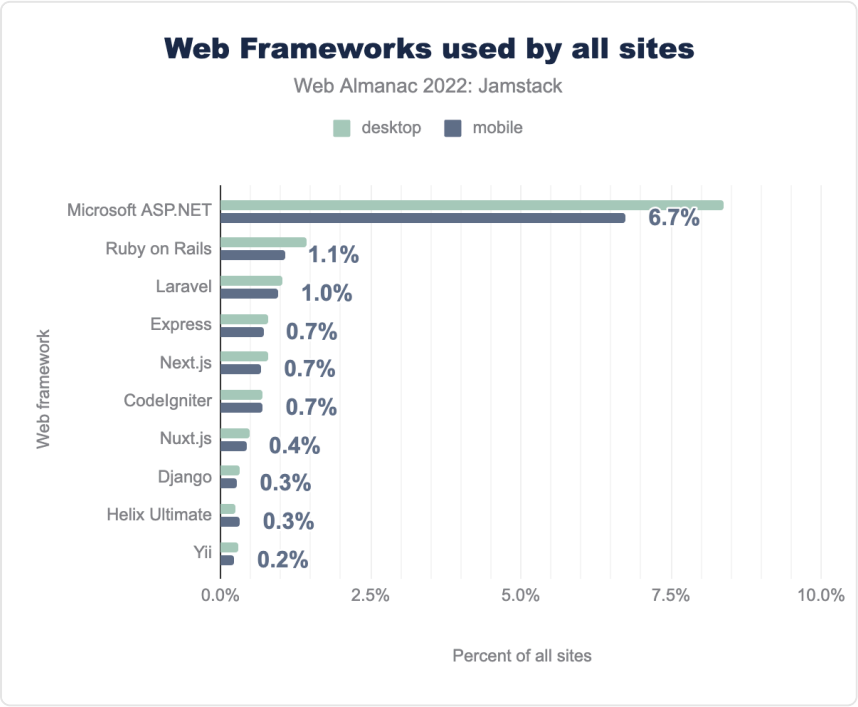



*図19.7. すべてのサイトで検出可能なSSG。*

Jamstackサイトについては、こちらです。

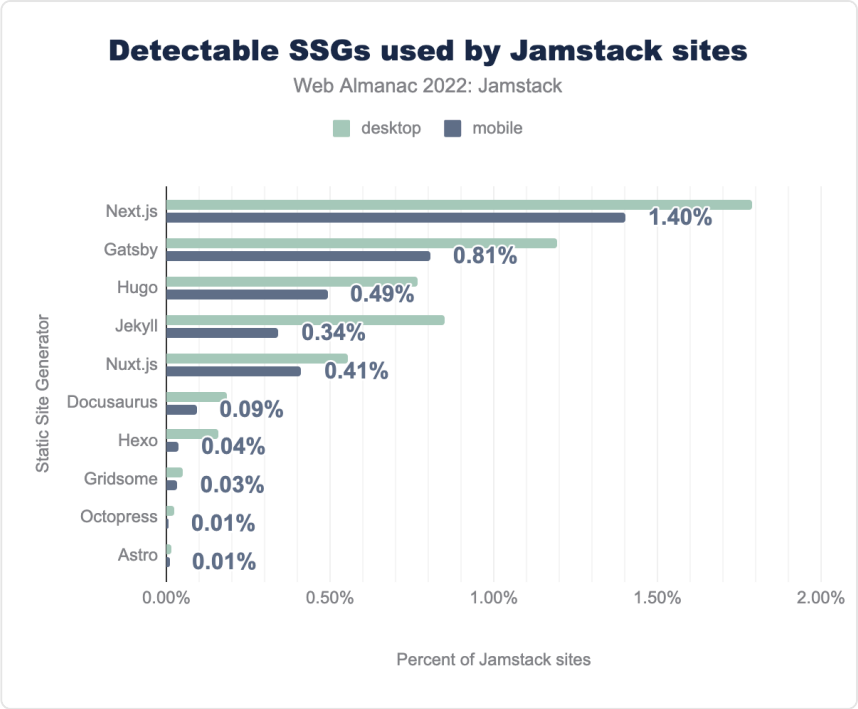



*図19.8. Jamstackサイトで検出可能なSSG。*

ご覧のとおり、ほぼ同じリストで、ほぼ同じ順序ですが、Nuxtはいくつかの位置を下げています。これは直感的に理解できます。SSGによって生成されたサイトはJamstack-yと資格を得ることが期待されますが、それがその建築目標を達成する唯一の方法では明らかにありません。

SSGは、すべてのサイトではなく、すべてのJamstackサイトのはるかに大きな割合を占めており、SSGを使用することはJamstackサイトを入手するためのかなり良い方法であることを示しています。しかし、SSGを使用してもJamstackサイトが作成されるとは限りません。こちらは私たちのサンプルのいくつかのフレームワークの総数です。

| SSG | すべてのサイト | Jamstackサイト | Jamstack % |
|---|---|---|---|
| Next.js | 39,928 | 2,651 | 7% |
| Nuxt.js | 24,600 | 824 | 3% |
| Gatsby | 12,014 | 1,765 | 15% |
| Hugo | 5,071 | 1,135 | 22% |
| Jekyll | 3,531 | 1,259 | 36% |

*図19.9. デスクトップでのJamstackサイトにおけるSSGの割合。*

すべてのSSGで、私たちの定義によるJamstack-yと資格を得たサイトの割合は、そのフレームワークの総サイト数よりも少なかったです。Jekyllがもっとも良い結果を出し、Jekyllのサイトの3分の1以上が私たちの基準を満たしていました。NextとNuxtはとくに低い割合で、これは予想されることです。これらはSSGとして使用できますが、しばしば動的サイトを作成するために使用され、私たちはどのモードであるかを判断する方法がありません。

## Jamstack-yホスティング

私たちは、人々がJamstack-yサイトをどこでホストしているかにも興味がありました。パターンはあるでしょうか？再び、私たちはWappalyzerのデータを使用して技術を特定しましたが、今回は彼らのPlatform as a Service (PaaS)カテゴリーを使用しました。

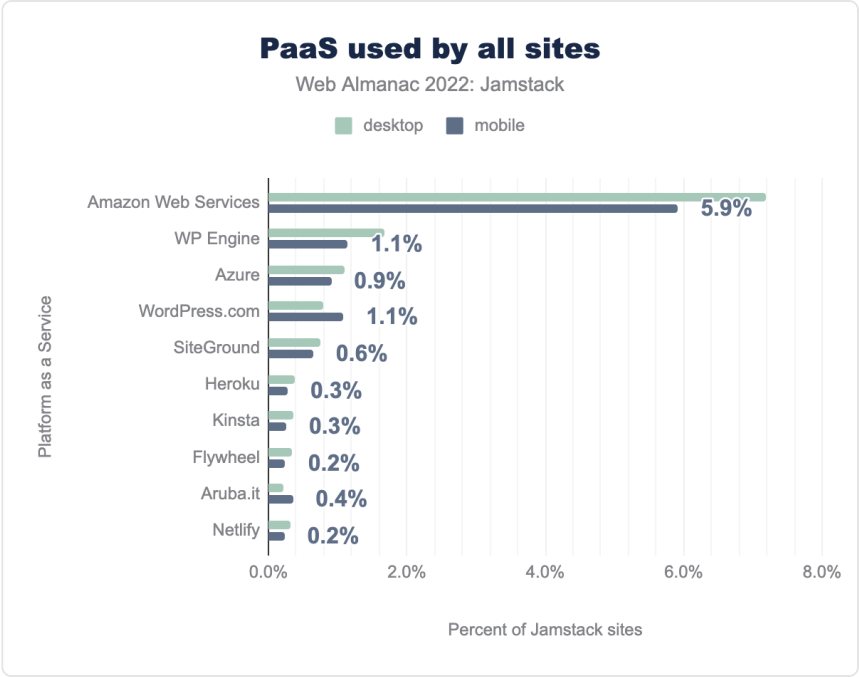



*図19.10. すべてのサイトで使用されるPaaS。*

そしてJamstackサイトについてはこちらです。

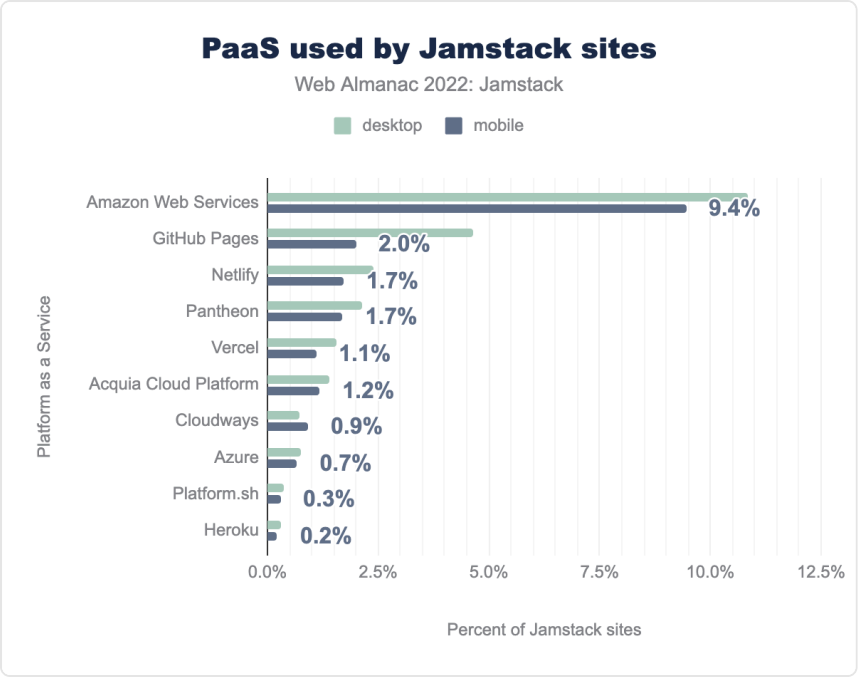



*図19.11. Jamstackサイトで使用されるPaaS。*

ウェブの巨人であるAmazon Web Servicesは両方のセットで支配的ですが、全体的な好みとJamstack-y開発者の好みの間にはいくつかの重要な違いがあります。

WP Engine、Azure、WordPress.comはWeb全体で非常に人気がありますが、Jamstackコミュニティ内での人気は大幅に低下しています。GitHub Pagesは広いウェブでは11位ですが、Jamstackセットでは2位です。一方、Jamstack開発者のお気に入りであるNetlifyとVercelは、それぞれ3位と5位を占めており、大きなウェブではNetlifyは10位、Vercelは14位（表示されていません）です。PantheonとAcquia Cloud Platformは、全体でトップ10に入っていませんが、それぞれ11位から4位、12位から6位に大幅に跳ね上がりました。

これらのホストの相対的な人気が広いウェブに対してどのように変化したかは、それらがすべて世帯名ではないため、少し驚くかもしれませんが、2021年から2022年にかけて私たちのセットでプラットフォームの好みがどのように変わったかを見る価値があると思いました。モバイルデータを使用して、2021年から2022年にかけてJamstackサイトでさまざまなプラットフォームを使用する割合がどのように変わったかです。

| PaaS | 2021 | 2022 | Change |
|---|---|---|---|
| Amazon Web Services | 7.00% | 9.45% | 2.45% |
| GitHub Pages | 2.62% | 1.99% | -0.63% |
| Pantheon | 1.97% | 1.70% | -0.27% |
| Netlify | 1.68% | 1.72% | 0.04% |
| Acquia Cloud Platform | 1.37% | 1.18% | -0.20% |
| Vercel | 0.50% | 1.10% | 0.60% |
| Cloudways | | 0.91% | N/A |
| Azure | | 0.67% | N/A |
| Platform.sh | 0.27% | 0.29% | 0.02% |
| Heroku | 0.28% | 0.22% | -0.05% |

*図19.12. JamstackサイトのSSGs割合（デスクトップ）。*

GitHub Pages、Pantheon、Acquia Cloud Platform、Herokuは人気が下がっている一方で、AWS、Netlify、Vercel、Platform.shはより人気が出ています。2021年のPaaSデータにCloudwaysやAzureが含まれていないため、これらとの比較はできません。AWS、Netlify、Vercelが人気を集めているのは、単なるホスティングではなく、開発者のワークフローに役立つツールのスイートを提供しているからと推測できます。

すべてのプラットフォームリストから欠けているのは、Webの巨人であるCloudflareですが、WappalyzerはCloudflareをPaaSではなくCDNとして分類しています。ただし、Cloudflareには非常にJamstack的なPaaSオファリングであるCloudflare Pagesがありますが、Wappalyzerのデータは「CDN上でホストされている」と「そのCDN上に一部のアセットをホストしている」とを区別していないため、この分析には含めることができませんでした。著者は、Cloudflareが非常に人気のあるJamstackホスティングオプションであると考えていますが、これを確認するための良いデータはありません。

## 結論

今年の分析からもっとも重要な洞察は、HTTP Archiveのデータだけを見てJamstackを測定することは困難であるということです。それでも、プラットフォームやフレームワークに依存しない測定アプローチを使用して、結果のデータから「既知」のJamstackプラットフォームやフレームワークと強い相関関係を見出すことができたのは、ArchiveでJamstackサイトを

確実に特定できるようになったことの励みになりました。

Webの何パーセントがJamstackであるか正確にはわかりませんが、Webの約3%がJamstack的であり、このグループは過去3年間で強く成長していると言えます。これはJamstackコミュニティにとって良い兆候です。

また、一部のフレームワークやホスティングプラットフォームは、広いWebよりもJamstackでより人気があることがわかりました。これは、技術的に当社の基準を達成するのに優れているためか、または単にJamstack開発者が特定のスタックを好むコミュニティの好みがあるためかもしれません。

もちろん、Jamstackコミュニティが特定のプラットフォームやフレームワークを好むと、それは自己強化のトレンドとなります。これにより、それらのツールを使用してJamstackサイトを実現する方法に関するドキュメントやチュートリアルが増え、これらのツールを使ってJamstackサイトを構築することが容易になります。したがって、任意のテクノロジースタックでJamstack的な結果を達成できると信じていますが（データもそれを示しています）、Jamstackサイトを達成するのに役立つツールやプラットフォームを特定するのにこのデータが役立つことを願っています。

また、Jamstackサイトを構築する際に目指すべきより確かな目標を持つことが、「Jamstackとして数えるもの」を定量化するこの試みから得られる最終的な有用な洞察であると信じています。特定のフレームワークを選んでそれを忘れるというわけではありません。結果についてです。LCPやCLSの時間を分析することで、努力が「Jamstack的」であるかどうかを定量化できることは、自動化できる有用なことです。

## 著者

### Laurie Voss

seldo  http://seldo.com

Laurieは1996年からWeb開発者として活動しており、awe.sm（2010年）awe.sm[707] や npm（2014年）npm[708] のような会社を設立するために時々休憩をとっています。現在は、Netlify[709]でデータエバンジェリストとして働いています。彼はWebをより大きく、より良いものにすることに情熱を注いでいます。そのための最良の方法の1つは、より多くの人々にWeb開発を奨励し、既存の技術を教え、Web開発を容易にするためのツールやサービスを構築することにより、彼らが多くを学ぶ必要がないようにすることだと彼は考えています。

707. *https://www.crunchbase.com/organization/snowball-factory*
708. *https://npmjs.com/*
709. *https://netlify.com*

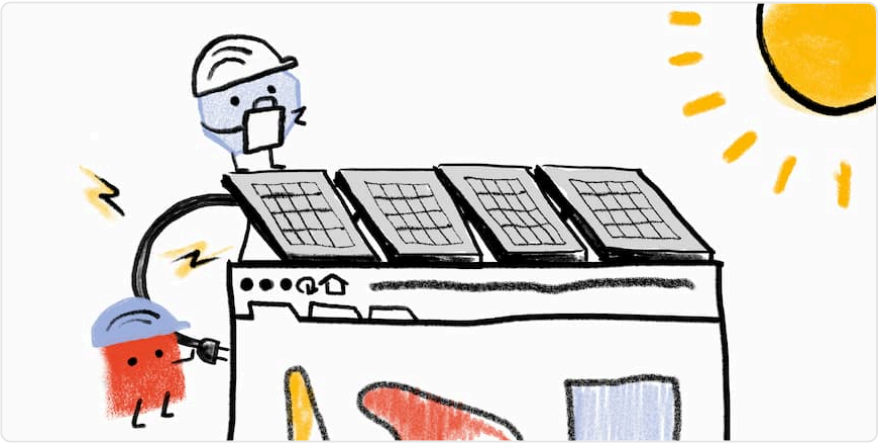

## Salma Alam-Naylor

@whitep4nth3r whitep4nth3r https://whitep4nth3r.com/


Salmaはあなたのためにコードを書きます。彼女はライブストリーマーで、ソフトウェアエンジニア、開発者教育者でもあり、人々がテクノロジー分野に進むのを手助けすることが好きです。音楽教師およびコメディアンとしてのキャリアの後、2014年にテクノロジー分野に転身し、スタートアップ、エージェンシー、グローバルなeコマースのためのフロントエンド開発とテクノロジーリーダーシップを専門としています。現在はデベロッパーリレーションズで働いています。TwitchでSalmaを見つけて[710]、彼女が何を作っているのかを確認してください。

710. *https://twitch.tv/whitep4nth3r*

# 部 III 章 20
# 持続可能性



***Laurent Devernay、 Gerry McGovern と Tim Frick によって書かれた。***
*Chris Adams、 Caleb Queern と Edmond de Tournadre によってレビュ。*
*Fershad Irani、 Cameron Casher と Arik Smith による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

2019年にGreenIT.frは、34億台の機器と41億人のインターネットユーザーが存在すると推定しました[711]。そのため、デジタル世界が人類の炭素足跡に与える影響は、一次エネルギー消費および温室効果ガス排出の約4%、水消費の0.2%、電気消費の5.5%を占めるかもしれません。

もう1つの重要な指標は、非生物資源（金属などの「生きていない」資源）の枯渇への寄与です。私たちが使用するすべてのデバイスは製造のために材料を必要とします。そのため、ユーザー機器の製造は環境への影響のもっとも重要な源と考えられており、ほとんどがまったくリサイクルされていない機器の使用終了がこれに続きます。これはデータセンターやネットワーク、さらにはユーザー機器の使用よりもはるかに影響が大きいです。一部のメーカーからの努力にもかかわらず、インジウム、銅、金などの必要な材料の枯渇のため、今後数

711. *https://www.greenit.fr/environmental-footprint-of-the-digital-world/*

年間で状況は悪化する可能性が高いです。

GreenIT.frの前述の研究によると、デジタルサービスの全体的な影響は年々着実に増加しており、2010年から2025年の間に2倍または3倍になる可能性があります。これを回避するためには、少なくとも軽減するためには、私たちは所有する接続デバイスの数を減らし、できるだけ長く各デバイスを保持する必要があります：購入するのではなく修理すること。これは、とくにスマートフォンなど、すぐに古くなってしまうように見えるデバイスにとって厳しいかもしれません：ウェブサイトやアプリケーションのロードに時間がかかるほど、バッテリーの持ちが悪くなります。

デジタルサービスの影響を減らし、デジタルサービスを無形で環境に優しいものとして考える方法を変えることができます。

収集されたすべてのデータを考慮すると、Web Almanacはウェブサイト全体の環境への影響を評価するのに適した場所のようです。この旅の中で、ベストプラクティスを通じてそれらをどのように減らすか、そしてこれらがどれだけ広く採用されているかも見ていきます。

このために、私たちは次のことを区別します。

- 節度：必要なときにのみ何かを実装すること。これはデジタル全体（本当に接続されたおむつが必要ですか？）、機能（ホームページにソーシャルメディアのフィードは役立ちますか？）、またはコンテンツ（装飾的な画像、ビデオなど）である可能性があります。あなたのウェブサイト上のすべてが有用で、使用されている、使いやすい（そして再利用可能）かどうか自問してください。
- 効率：節度を考慮した後、ウェブサイト上に残るもののサイズや影響をどのように減らすか。ウェブサイトの場合、これは主に縮小、圧縮、キャッシングなどの技術的最適化を通じて行われます。

始めるためのオンライン活動。

- あなたはエコ責任あるインターネットユーザーですか？[712]
- あなたのインターネット閲覧の影響は何ですか？[713]
- ウェブページを構成するさまざまな要素の重量比較[714]

この旅に役立つリソースは以下のものがあります:

- ベストプラクティスのリポジトリ: 115 bonnes pratiques[715], Handbook of

712. *https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/Partie3/Activites/Capsule_Partie3_4_AgirUtilisateur/story.html*
713. *https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/Partie3/Activites/Capsule_Partie3_3_Mesurer/story.html*
714. *https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/Partie3/Activites/Capsule_Partie3_2_Mesurer/indexEn.html*
715. *https://github.com/cnumr/best-practices/*

sustainable digital services[716].

- 書籍とウェブサイト: Sustainable Web Design[717].
- オンラインコース: INR - Sustainable IT[718], Environmental impact of digital services[719], Principles of Sustainable Software Engineering[720].

## 制限と仮定

すべてのベストプラクティスを網羅するわけではなく、利用可能な指標はそれらすべてをカバーすることができません。指標は、特定のウェブサイトに不必要な機能があるか、または一部の画像が純粋に装飾的であるかどうかを教えてくれません。このような考慮事項はこの章の範囲を超えていますが、まだできることはたくさんあります。そして、Lighthouseがさらに多くの種類の監査を提供するにつれて、新しい指標が利用可能になると期待できます。

ここでの環境指標は炭素排出のみですが、水消費、土地利用、非生物資源消費などの他の指標も考慮すべきです。これはLCA（ライフサイクルアセスメント）[721]の正確なポイントです。しかし、そのような操作には専門知識、多くの情報、時間が必要です。現在、一部の構造は、より少ない指標と情報を組み合わせて妥協を図っており、LCI（ライフサイクルインベントリ）を使用しています。これにより、環境への影響評価をより手頃な価格でアクセスしやすく（たとえばCI/CDまたはモニタリングでの繰り返し可能な）行うことができますが、あなたが行う必要がある仮定を制御下に置くことができます。

ページで収集された指標のみを使用しますが、デジタルサービスの環境への影響を評価するためには、ユーザージャーニー全体の指標を収集する方が正確かもしれません。たとえば、電子商取引サイトでは、ユーザーが記事を購入して支払いをすることを考慮する方がよいでしょう。

## 交差点環境問題

持続可能性は、1987年のブルントランド委員会による初期の定義以来、大きく進化しました。現在では、その中核となる環境に焦点を当てつつ、交差する社会的およびガバナンスの問題（”ESG”の”S”と”G”）を取り入れています。より責任ある持続可能なインターネットは、これを反映すべきです。

言い換えれば、デジタルの持続可能性は排出量だけに焦点を当てることはできません。気候

716. *https://gr491.isit-europe.org/en/*
717. *https://sustainablewebdesign.org/*
718. *https://www.isit-academy.org/*
719. *https://learninglab.inria.fr/en/mooc-impacts-environnementaux-du-numerique/*
720. *https://docs.microsoft.com/en-us/learn/modules/sustainable-software-engineering-overview/*
721. *https://learninglab.gitlabpages.inria.fr/mooc-impacts-num/mooc-impacts-num-ressources/en/Partie3/FichesConcept/FC3.3.1-ACVservicesnumeriques-MoocImpactNum.html?lang=en*

変動は大きな要因ですが、それを不公平な解決策を正当化するため[722]や不平等を助長するため[723]に使用することはできません。[PDF]それは気候 *正義* に基づいていなければなりません。

この目的のために、デジタル製品やサービスを設計する際には、以下の交差する問題を念頭に置いてください:

- **アクセシビリティ:** コンテンツへの障壁を取り除くことで、あなたのウェブサイトはより使いやすく、アクセスしやすくなります。これは、障害を持つユーザーを含むユーザーがタスクを達成するために回り道をしなくて済むため、環境への影響も改善します。
- **プライバシー:** 侵入的でないウェブサイトは、ユーザーにデータのコントロールを与え、共有したいものを選択させることで、より良いものです。プライバシーに焦点を当てたウェブサイトは、追跡、保存、維持するデータが少ないため、通常環境にも優しいです。
- **誤情報/偽情報:** 人々は質問に答えるためにインターネットを利用します。誤情報（非故意的）と偽情報（故意的）を含むコンテンツは、ユーザーが効率的にこれを行う能力を損ないます。
- **注意経済:** 欺瞞的なパターン[724]を避けることで、ユーザーを集中させ、無意味なブラウジングや最初の目的からの逸脱を減らします。
- **セキュリティ:** 持続可能性を目指すことは外部リソースが少なく、機能が少ないなど、ウェブサイトの攻撃対象を減らすことで、セキュリティにも役立ちます。

これらはすべて、デジタル持続可能性の原則に沿った企業デジタル責任[725]へのより広範な組織的アプローチの一部です。

## ウェブの環境への影響を理解する

インターネットはこれまでに存在した最大で、もっともエネルギー集約的な機械です。インターネットを作成し維持するためには、膨大な物質入力が必要です。1台のサーバーは製造中に1トン以上のCO2を引き起こすことがあります。ラップトップは製造に300kgのCO2を引き起こし、1,200kgの原材料の採掘につながります。持続可能な採掘というものは存在しません。

インターネットの大部分のエネルギーと廃棄物はデバイス自体に組み込まれていますが、インターネットの運用に必要なエネルギーも無視できません。データは本質的に無料で、私た

722. *https://qz.com/845206/renewable-energy-human-rights-violations/*
723. *https://www.iisd.org/system/files/publications/green-conflict-minerals.pdf*
724. *https://blog.mozilla.org/en/internet-culture/mozilla-explains/deceptive-design-patterns/*
725. *https://www.mightybytes.com/blog/what-is-corporate-digital-responsibility/*

ちが望むだけ保存できると常にマーケティングされていますが、データの保存と処理には実際の、急速に増加するエネルギー要求があります。たとえば2015年、アイルランドのデータセンターは電力の5%を消費していましたが、2021年には14%に増加し、アイルランドの農村地域のすべての家庭や建物の需要を上回りました。

ウェブの持続可能性を高めるために、デバイスのより良い管理に焦点を当て、ウェブサイトやアプリと対話するために使用されるデバイスに可能な限りストレスをかけないようにすることで、設計および開発を進めることができます。私たち自身のデバイスに関しては、デバイスの寿命とエネルギー消費に焦点を当てる必要があります。コンピューターの稼働寿命が長ければ長いほど、製造中に引き起こされた損害を償却できます。この考えの頂点は、オープンソースになり、Linuxのようなオペレーティングシステムを使用してデバイスの寿命を延ばすことです。オープンソースは、再利用と共有に焦点を当てることで、元々のデジタル持続可能性の哲学です。それでも、持続可能性のベストプラクティスの実装を防ぐべきではありません。

設計および開発プロセス中に消費されるエネルギーが少なければ少ないほど良いです。コードやコンテンツを再利用できるなら、それは素晴らしいアイデアです。最小限のワット数を使用してください。ラップトップはデスクトップよりもはるかにエネルギー効率が良いです。たとえば、大画面はラップトップと同じくらいのエネルギーを消費する可能性があるため、避けるべきです。エネルギー消費を減らすことは良いことです。

人気があり、需要の高いウェブサイトやアプリの場合、エネルギーと廃棄物の影響の最大98%がユーザーのスマートフォンやラップトップで発生します。小さな節約が大きな違いを生むことがあります。ダニー・ヴァン・クーテンは、200万のウェブサイトで使用されるWordPressのMailchimpプラグインを開発しました。彼はコードを20KB削減し、それが月間59,000kgのCO2の削減につながったと推定しています[726]。

## ウェブサイトの環境への影響を評価する

Ecograder、Website Carbon、Ecoping、CO2.jsなど、いくつかの利用可能なツールですでに共有されている方法論を採用しました[727]。これにより、データ転送量（たとえば、ページの重さ）に基づいて温室効果ガス排出量を推定します。

コミュニティはこの話題について合意に達するのにまだ苦労しています[728]。ここで利用可能な指標を考慮すると、これが最良の妥協案と思われます。しかし、すべての人がこれに同意するわけではなく、この方法論は今後数ヶ月から数年の間に進化するべきであり、おそらく進化するでしょう。

726. *https://raidboxes.io/en/blog/wordpress/wordpress-plugin-co2/*
727. *https://sustainablewebdesign.org/calculating-digital-emissions/*
728. *https://marmelab.com/blog/2022/04/05/greenframe-compare.html*

そこで、ページの重さの概要から始め、次に炭素排出量の計算に進みます。

## ページの重さ

ページの重さは、ウェブページにアクセスするために転送されるデータ量を表します（HTTPリクエストに基づいてのみ）。前述のように、ここでは温室効果ガス排出量を計算するための代理指標として使用されます。

この指標はできるだけ低く保つことが推奨されます。始めるときは1 MBが最大ですが、究極の閾値は500 kBであるべきです[729]。

これについての詳細は、Page Weightの章を参照してください。

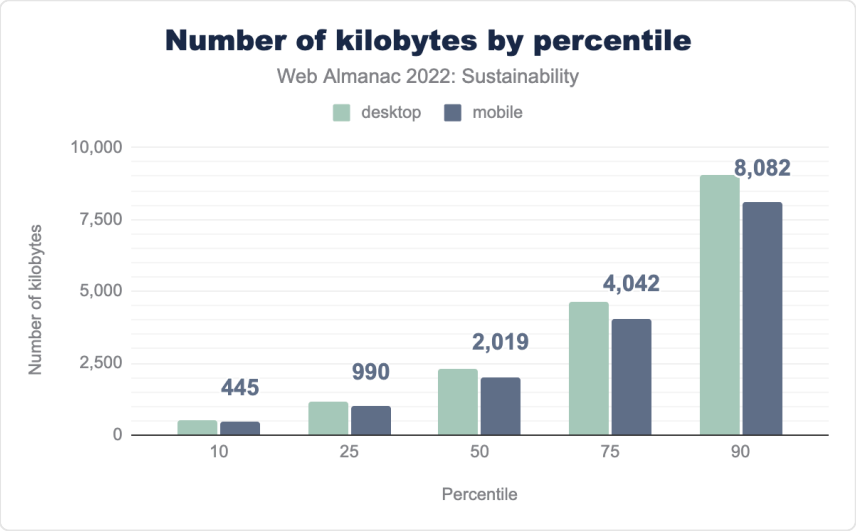



*図20.1. パーセンタイル別のキロバイト数*

モバイルとデスクトップのページ重量を比較すると、その差が小さいことがわかりますが、これは意外なことです。メディアは画面のサイズに応じて適切なサイズと形式で提供されるべきですが、ここではそうではないかもしれません。

90パーセンタイルでは、デスクトップページは9 MBを超え、モバイルページは8 MBを超えています。推奨される閾値500 kBからはかけ離れています。この閾値以下のページを見つけるには、10パーセンタイルに到達する必要があります。寛大な気持ちで1 MBを目指す場合、これは25パーセンタイルの周辺で見つかるかもしれません。まだまだ長い道のりがあり

729. *https://infrequently.org/2021/03/the-performance-inequality-gap/*

ます...

## 炭素排出量

*注:「炭素排出量」という概念は単純化されています。ここでは温室効果ガス排出量を考慮しており、単に炭素排出量のみではありません。*

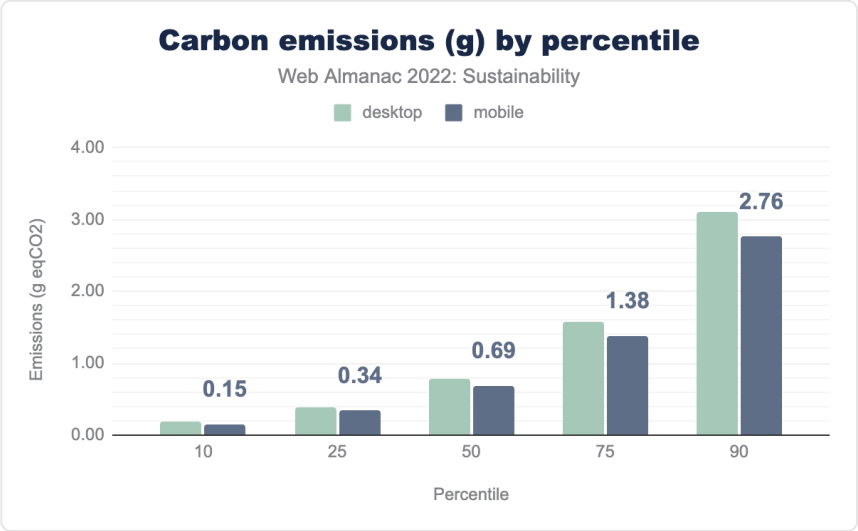



*図20.2. パーセンタイル別の炭素排出量（g）*

ウェブサイトの炭素排出量はモバイルとデスクトップで非常に近いです。10パーセンタイルでは非常に低く（約0.15 g eqCO2、これは熱車で1メートル未満に相当[730]）。90パーセンタイルでは最大2.76 g eqCO2に達し（熱車で少し14メートル以上に相当）。

これは多くないように思えますが、各ウェブサイトは毎月何千もの訪問者を獲得し、場合によってはさらに多くの訪問者を獲得することを念頭に置く必要があります（時にはそれ以上）。グラフで見るのは1ページを1回訪問したときの排出量です。すべてのウェブサイトの毎月の環境への影響は積み上がります。

さらに追加のグラフ: コンテンツタイプ別のパーセンタイルごとの排出量。

730. *https://datagir.ademe.fr/apps/mon-impact-transport/*

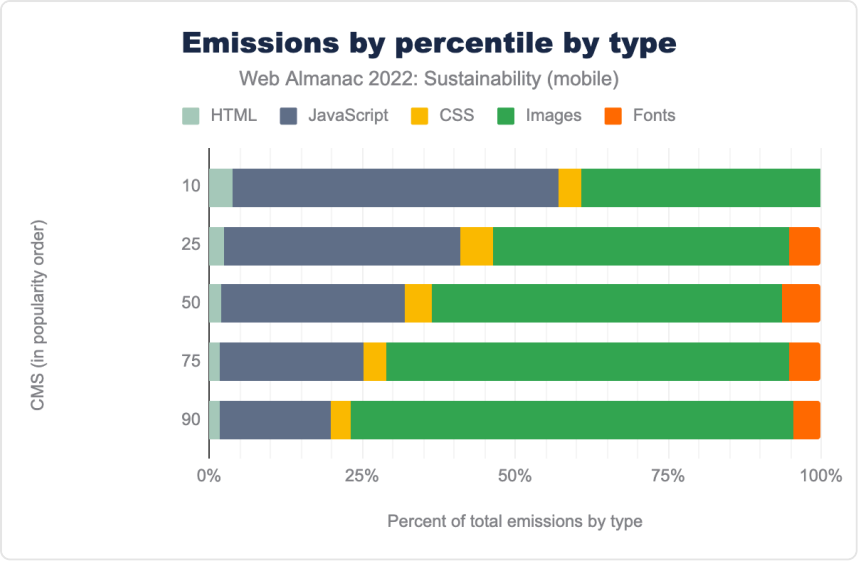



*図20.3. タイプ別の総排出量のパーセンタイルごとの割合（モバイル）*

画像とJavaScriptは影響が大きいようですが、上位パーセンタイルに行くにつれて画像の影響がさらに大きくなります。ただし、炭素排出量の計算にはデータ転送のみを考慮していることに注意してください。通常、JavaScriptの処理は画像よりも影響が大きいです。JavaScriptファイルをダウンロードしたあとでも、それらを処理する必要があり、時にはページの再読み込みや他のリソースの取得を行うことがあります。それにもかかわらず、このグラフはこれらの影響を減らす必要性を強調しています。これは画像に関しては後の章で見るように、かなり簡単かもしれません。JavaScriptの場合はもっと難しいかもしれませんが、最小化、圧縮、またはそれへの依存を減らすなどの技術的な最適化がいくつかあります。これについても後で詳しく説明します。

## リクエスト数

ページを読み込むためにファイルが必要な場合、リクエストが発行されます。このため、リクエストはネットワークとサーバーに与えるページの影響を表すのに役立ち、これが環境への影響を計算するためにときどき使用される理由です。リクエストを分析することで、可能な最適化を見つけることができます。これについては、さまざまなタイプのアセットと外部リクエストを議論する際に考慮します。

リクエストの数は最小限に抑えるべきです。25未満という上限を設けることはかなり良いスタートです。しかし、トラッカーなどがしばしばその目標を達成するのを難しくしています。

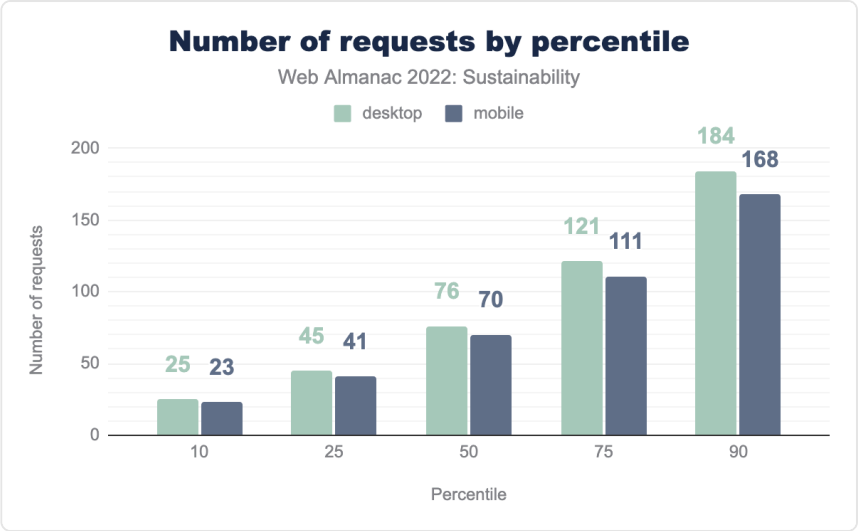



*図20.4. パーセンタイル別のリクエスト数*

モバイルとデスクトップのリクエスト数を比較すると、またしてもわずかな違いが見られますが、そうあるべきではありません。25 HTTPリクエストの閾値以下のページを見つけるには、再び10パーセンタイルに到達する必要があります。

それでは、どのコンテンツタイプが原因でしょうか？

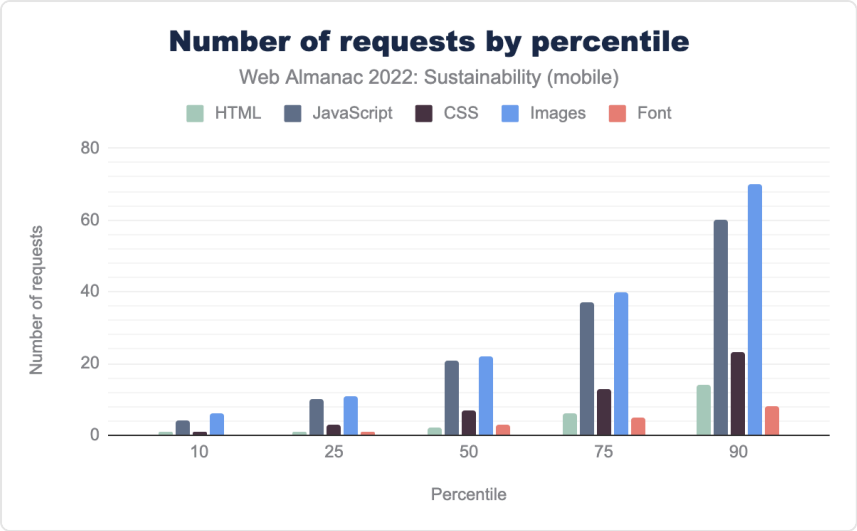



*図20.5. モバイルでのタイプ別パーセンタイルごとのリクエスト数*

通常通り、画像が主な原因ですが、JavaScriptもそれに迫っています。

モバイルとデスクトップのバージョンでほぼ同数のHTTPリクエストがありますが、そうあるべきではありません。ページの重さと同様に、モバイルページはできるだけ軽く保つべきです。これは、古くなったデバイス、不安定な接続、高価なモバイルデータを考慮しています。多くの人がまだこのような最適でない条件でウェブを使用しているため、モバイルウェブはこれを考慮して、すべての人にアクセスしやすいように最善を尽くすべきです。

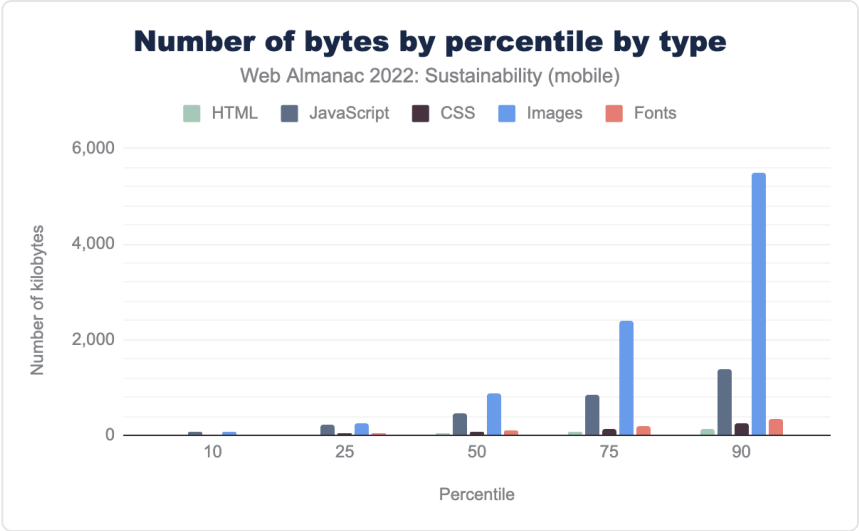



*図20.6. モバイルでのタイプ別パーセンタイルごとのバイト数*

画像とJavaScriptのHTTPリクエスト数はほぼ同じですが、全体の重さは画像の方がはるかに高いです。一般的に、画像よりもJavaScriptを処理する方が重いため、これはまだ悪いニュースです。再び、結果はモバイルとデスクトップで非常に近いですが、モバイルで軽い体験を提供することが理にかなっているように思えます。

## より持続可能なホスティング

*注: ここでは（他の場所でも）「グリーンホスティング」という言及が見られます。これはある種の省略形で、実際に完全にグリーンである、またはカーボンニュートラルなどのホスティングは存在しません。ここでは、より持続可能なホスティングの使用方法に焦点を当てます。*

この章の大部分では、ネットワーク、コンピュート、ストレージなどのリソース量の変化がデジタルサービスの環境への影響にどのように影響するかに焦点を当てています。これを持続可能性のためのレバーとしての*消費*と考えるかもしれません。しかし、他にもレバーは存在します。ゼロまで効率化することはできませんが、同じ種類のサーバーで実行される同じコードが、よりグリーンなエネルギーで実行される場合、それはそうでない場合よりも環境への影響が低くなります。このレバーを*強度*と考えることができます。

ここにはいくつかの良いニュースがあります。世界中で私たちが依存している電力グリッドは、再生可能エネルギーと貯蔵のコストが下がることによって時間とともによりグリーンになっています。2022年には、私たちの電気の38%がクリーンなソースから得られています

（エンバー気候の例[731]およびこのチャート[732]）。

しかし、すべてのグリッドや、プロバイダーが運用するすべての地域が同じくらいグリーンであるわけではありません。アマゾンウェブサービスの顧客カーボンフットプリントツール[733]は、一方の地域でサービスを実行することと他方で実行することの間に、炭素排出量に計測可能な違いがあることを示しています。また、オープンソースのクラウドカーボンフットプリント[734]も、増え続ける数のプロバイダーについてそれを示しています。他の場所では、グリーンウェブ財団[735]も、その地域のグリッドが化石燃料でどれだけ動力を得ているかの見積もりのために、任意のドメインを調べるAPIを提供しています。

しかし、再生可能エネルギーを使用するだけでは、真に持続可能なホスティングを提供するには不十分です。PUE（電力使用効率）、WUE（水使用効率）、機器の取り扱い方法なども確認する必要があります。これをさらに調査するには、Wholegrain Digitalの記事[736]やヨーロッパデータセンターコードオブコンダクト[737]を確認できます。一般的に、カーボンニュートラルであると主張する企業には注意してください（フランスのADEME機関が述べているように[738]）、とくにほとんどの企業がスコープ3の排出を含んでいないためです。また、上記のように、あなたの炭素排出を補償するだけでは不十分で、それを減らすべきです。

### HTTPアーカイブにリストされているサイトの何割がグリーンホスティングを利用していますか？

技術企業の数が増えており、彼らのインフラを動力するために購入するすべての電力をグリーンにするための措置を講じています。MicrosoftやSalesforceなどの企業はすでに、そのサーバーファームが年間で使用するだけのグリーンエネルギーを購入しています。多くの他の企業も同様です。私たちはグリーンウェブ財団のデータセット[739]を使用して、どれだけの組織が「グリーンホスト」として類似の措置を講じているか[740]、そして彼らが毎年使用するすべてのエネルギーをグリーンエネルギーで動力を供給している証拠を共有している場所を見ました。

731. https://ember-climate.org/insights/research/global-electricity-review-2022/
732. https://public.flourish.studio/story/1176231/?utm_source=showcase&utm_campaign=story/1176231
733. https://aws.amazon.com/aws-cost-management/aws-customer-carbon-footprint-tool/
734. https://www.cloudcarbonfootprint.org/
735. https://www.thegreenwebfoundation.org/
736. https://www.wholegraindigital.com/blog/choose-a-green-web-host/
737. https://e3p.jrc.ec.europa.eu/communities/data-centres-code-conduct
738. https://presse.ademe.fr/2022/02/lademe-publie-un-avis-dexperts-sur-lutilisation-de-largument-de-neutralite-carbone-dans-les-communications.html
739. https://www.thegreenwebfoundation.org/green-web-datasets/
740. https://www.thegreenwebfoundation.org/what-we-accept-as-evidence-of-green-power/

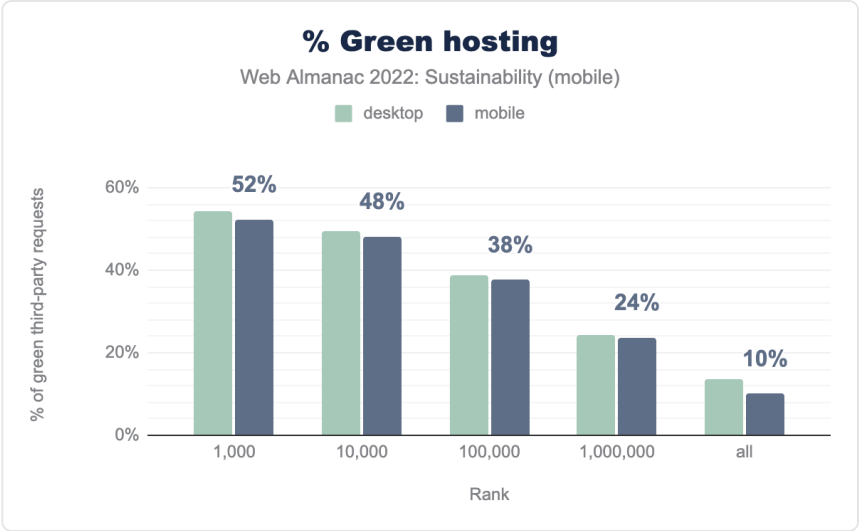



*図20.7. グリーンホスティングの割合*

全体として、測定されたウェブサイトのわずか10パーセントがグリーンホスティングに依存しています。これはウェブサイトがグリーンホスティングを選択することと、ホスティング会社が持続可能性を目指すことの両方で、多くのことが行われるべきであり、行われるべきであることを強調しています。

*注: グリーンドメインのこれらの数字は、グリーンウェブ財団と直接共有された情報、または公開された情報に基づいています。彼らの検索サービスのAPIレスポンスでリンクされています。サイトがグリーンであると思われる場合に「グリーンではない」と表示される理由については、彼らの説明ページ[741]を参照してください。*

## ウェブサイトの環境への影響を減らす

ベストプラクティスは、測定がなければ機能しませんし、その逆もまた同様です。ウェブサイトの環境への影響についてより良い表現を得た今、この影響をどのように軽減するか見ていきましょう。

### ムダを避ける

ウェブサイトの影響を減らすもっとも明白な方法の1つは、不必要なものをすべて避けることです。

741. https://www.thegreenwebfoundation.org/support/why-does-my-website-show-up-as-grey-in-the-green-web-checker/

- コンテンツとコードのムダを減らす：多くのウェブサイトやアプリには不必要なコンテンツや機能があります。このページやその機能が本当に必要かどうかを問う声になりましょう。陳腐なストック画像はほとんどのページに何も加えませんが、重さだけが加わります。画像を削除することは、ページの重さを減らす最良の方法の1つです。使用されていないコードも削除すべきです。

- 処理を減らす。バイト単位で見ると、JavaScriptはHTML、CSS、画像、テキストよりもはるかに大きな影響を与えます。これは、ユーザーのデバイス上でエネルギー集約的な処理が発生するためです。できるだけエネルギー効率の良い実装を選びましょう。多くのウェブページはJavaScriptさえ必要としません。常に最軽量のコーディングオプションを選びましょう。

- 最軽量の通信オプションを選ぶ。 テキストは圧倒的に環境に優しいコミュニケーション方法です[742]。ビデオは圧倒的にエネルギー集約的で持続不可能です。そのため、ユーザーのニーズに応じて必要な場合にのみ使用すべきです。この場合、ビデオはできるだけ効率的に統合すべきです。

- 長寿命のデザインをする。古いマシンや古いオペレーティングシステムを使用している人々が、あなたのウェブサイト/アプリを引き続き使用できるようにデザインしてください。デバイスをできるだけ長く保持することを支援するように設計してください。デジタルの観点から見ると、環境にとってこれ以上のことはありません。

### 使用されていないアセットの読み込み

表示されるページ、とくに見えるページの部分に必要なアセットのみを読み込むべきです。これは、遅延読み込み、クリティカルCSS、*インタラクション時のインポート*や*可視性時のインポート*といったパターンを通じて行うことができます。また、クライアントデバイスに適したサイズで画像を読み込むことも含まれます。ここでは主に過剰なフォントと使用されていないコードに焦点を当てます。

#### フォント

持続可能性のためには、システムフォントを使用することが推奨されます[743]。カスタムフォントを使用する必要がある場合は、ムダを避けるためにいくつかのことを考慮する必要があります。フォントを読み込むことは、必要のない多くの文字や記号を読み込むことを意味する場合があります。たとえば、すべてのウェブサイトがキリル文字を必要とするわけではありませんが、一部のフォントはそれをネイティブに含んでいます。これを確認するには、

742. *https://www.google.com/url?q=https://text.npr.org/&sa=D&source=docs&ust=1662467318246688&usg=AOvVaw2K1v83mXXmEePMRoG6edxq*
743. *https://www.smashingmagazine.com/2015/11/using-system-ui-fonts-practical-guide/*

wakamaifondue[744]などのツールを使用できます。フォントファイルのサイズを減らすためには、WOFF2形式を目指し、可変フォントを使用する[745]べきです。また、サブセットを使用する[746]か、subfont[747]などのツールを使用できます。Google Fonts APIは、これらすべてのためにいくつかの賢いオプションを提供します[748]。Google Fontsに関しては、GDPRも念頭に置くべきです[749]。

このトピックについての詳細は、Fonts章を参照してください。また、web.dev[750]で関連するドキュメントを見つけることができます。

### 使用されていないCSS

使用されていないCSSは、とくにCSSフレームワーク（Bootstrapなど）を使用している場合に見られます。その際、ビルドフェーズで使用されていないCSSを削除することを念頭に置くべきです。Chrome Dev Toolsは、これを確認するためのCoverageツールを提供しています[751]。注意が必要です。多くのウェブサイトでは、初回訪問時にすべてのCSSとJavaScriptが読み込まれ、それらをキャッシュしてウェブサイトのさらなる訪問と探索のために使用します。これは必ずしも悪いことではありませんが、使用されていないコードはとくに考慮すべき欠点の1つです。なぜなら、それはさらなるコード処理を遅くする可能性があるからです。

744. *https://wakamaifondue.com/*
745. *https://the-sustainable.dev/reduce-the-weight-of-your-typography-with-variable-fonts/*
746. *https://everythingfonts.com/subsetter*
747. *https://github.com/Munter/subfont*
748. *https://web.dev/api-for-fast-beautiful-web-fonts/*
749. *https://rewis.io/urteile/urteil/lhm-20-01-2022-3-o-1749320/*
750. *https://web.dev/reduce-webfont-size/*
751. *https://developer.chrome.com/docs/devtools/coverage*

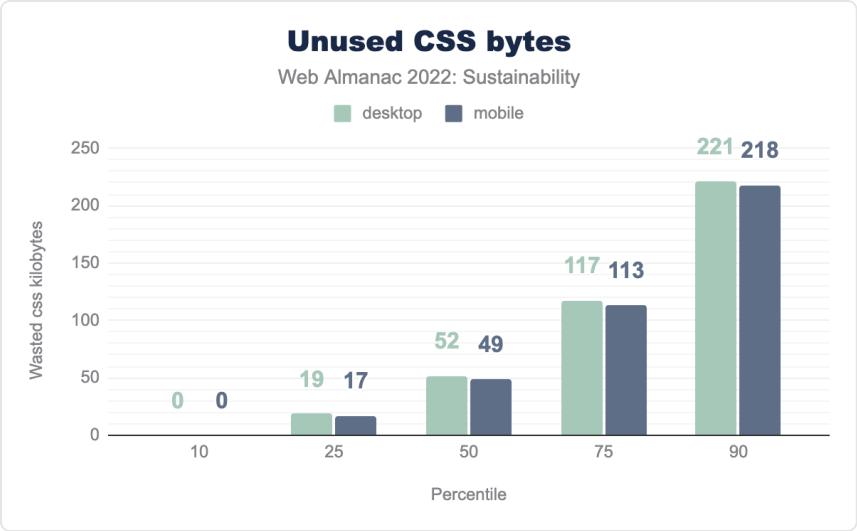



*図20.8. 使用されていないCSSのバイト数*

良いニュースは、10パーセンタイルのウェブサイトは不要なCSSを読み込まないことです。残念ながら、このグラフでは、90パーセンタイルで200 kB以上にまで増加しています。これが初期のキャッシュ理由であるかどうかは確認が必要です。持続可能性の観点から見ると、200 kBのCSSは大きな問題です。

## 使用されていない**JavaScript**

依存関係を追加したり、jQueryのようなライブラリを使用したりすると、使用されていないJavaScriptの量がすぐに増えることがあります。Chrome Dev ToolsのCoverageツール[752]は、これをチェックするのに良い方法です。CSSの場合と同様に、これは時にすべての必要なものをキャッシュする戦略の一部です。これは、使用されていないJavaScriptがより長い処理をもたらす傾向があるという事実によってバランスを取るべきです。可能であれば、必要な機能のみを持つより小さな代替品[753]を探し、すべてのツールボックスを読み込むことを望むのではなく、いつか役立つことを期待してください。かつて、jQueryはほとんどのウェブサイトで見られるオールインワンソリューションでした。今日では、多くのことがモダンなJavaScriptで処理できます[754]。NPMの依存関係とそれがあなたのバンドルをどのように大きくするかをチェックしてください[755]。

752. *https://developer.chrome.com/docs/devtools/coverage*
753. *http://microjs.com*
754. *https://youmightnotneedjquery.com/*
755. *https://bundlephobia.com/*

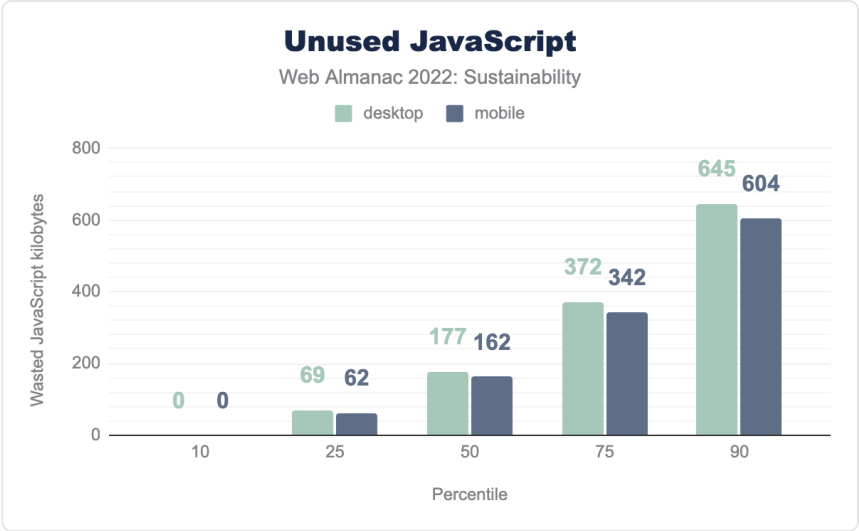



*図20.9. 使用されていないJavaScriptのバイト数*

再び、10パーセンタイルでは使用されていないJavaScriptがなく、素晴らしいです。しかし、これはCSSよりも上位パーセンタイルでさらに悪化し、90パーセンタイルで600 kB以上に達しています。これは、目指すべき理想的なページの総重量よりもすでに多いです。

## 持続可能なUX

ウェブサイトに対しては、開発プロセス中ではなく、設計とプロトタイピングの初期段階で持続可能な選択肢と最適化を行うことが可能です。効率的なコンテンツを最初から優先するユーザーエクスペリエンスを設計することが可能であり、ユーザーを持続可能性の実践における積極的な参加者として関与させる体験を作ることもできます。いくつかの信念とは対照的に、これらはすべて、美しく、地球中心のウェブ体験を作り上げることを犠牲にすることなく達成できます。

特定のユーザーエクスペリエンスタスクに関連する排出を定量化することは難しいですが、一部の研究では、消費者デバイスの使用が製品の全体的なデジタルフットプリントの最大52%を占めると推定しています。そのため、UXを持続可能性のために最適化することは、製品の環境への影響を大幅に減らすことにつながります[756]。

*756. https://www.mightybytes.com/blog/where-do-digital-emissions-come-from/*

### ステークホルダーのための設計

もっとも持続可能な製品は、非人間を含むすべてのステークホルダーが誰であるかを明確に把握しているものです。このようにして、製品は設計プロセス中に人間中心および地球中心のアプローチ[757]を取ります。

ステークホルダーマッピング[758]などの合理化された実践に従事することは、ステークホルダーのエコシステムとそのニーズを特定し、関与する全員のための包括的な体験を編成する道を設定するのに役立ちます。この研究を使用して、すべての関与するステークホルダー（人間および非人間）のニーズを無視することによる意図しない結果を防ぐ製品を設計するための機会をマッピングできます。これは、ユーザーの目標と一致するビジネスモデル[759]に地球中心の革新を組み込むための交差するタッチポイントを活用することによってさらに進めることができます。

### ユーザージャーニーの最適化

ユーザーが最小のステップで目標を達成するのを優先する戦略的なユーザージャーニーを作成することは、サイトの炭素に優しいウェブ体験を作成する方法の1つです。ユーザーが製品をナビゲートする時間が短ければ短いほど、その訪問中に使用されるエネルギー、データ、およびリソースが少なくなります。これを行う戦略には、ユーザーのエンゲージメントと方向性を促進するのに役立つ画像、ビデオ、視覚的資産の使用に注意を払うことが含まれます。

これには、「より少ないほどよい」というアプローチも含まれ、必要なコンテンツのみをユーザーに表示するムダのないデザイン実践に従事し、同じ価値を提供する資産の選択を強調します。これらはすべて、すでに増加している注意経済[760]を避けながら、ユーザーが必要なものをより早く取得するのに役立ちます。プロトタイピングやその他の方法を通じてユーザーフィードバックをテストし続け、ユーザーにとって最適な体験を作成していることを確認するための潜在的な痛みのポイントを特定します。

### 持続可能な行動を促進する

製品の機能に選択アーキテクチャを組み込むことで、ユーザーにその製品の環境タッチポイントに関連して持続可能な選択を促すことの人気が高まっています。この実践の例には、チェックアウト時にユーザーにより持続可能な包装オプションを提供する、もっとも炭素に優しい製品オプションを表示する、報酬システムやこれらの選択を視覚化して奨励するダッシュボードを構築するなどがあります。

757. https://planetcentricdesign.com/
758. https://www.mightybytes.com/blog/stakeholder-mapping/
759. https://www.mightybytes.com/blog/how-to-design-an-impact-business-model/
760. https://econreview.berkeley.edu/paying-attention-the-attention-economy/

この意思決定を支援し、この種の選択を提供することは、ユーザーがウェブサイトとより持続可能な方法で対話するのを助けるだけでなく、ユーザーの対話を最適化するための参入障壁を取り除くのにも役立ちます。最近では、アクセシビリティ機能、言語選択、デバイス最適化、または低エネルギーカラーを利用しながら適切なコントラストを促進する非常に人気のあるダークモードなどがあります。

これらの種類のオプションは、ユーザーの潜在的な痛みのポイントを最小限に抑えながら、時間、エネルギーを節約し、ユーザーのフラストレーションを防ぐカスタム体験を可能にするのに役立ちます。選択の力は通知の有効化と頻度など、より人気のあるオプトアウト機能にさらに深く浸透し、利用された場合にはリソースを節約し、ユーザーが訪問ごとの体験と影響をカスタマイズすることを可能にします。

### 循環性と製品寿命終了のための設計

デジタル製品やサービスの全ライフサイクルを分析し理解することで、廃棄物を減らし、時間とともに環境への影響を改善する機会が明らかになります。明確で測定可能な成功指標を定義し追跡することが、このプロセスを導くのに役立ちます。

- **循環性:** モジュラーで簡単に交換または更新可能なコンポーネントの設計と、継続的な改善に焦点を当てることで、技術的負債を減らし[761]、デジタル製品またはサービスの寿命を延ばすことができます。これにより、時間とリソースの使用も削減されます。
- **製品寿命終了:** デジタル製品やサービスに明確な廃止計画を作成することで、時代遅れまたは使用されていないデータを保存および提供するために必要なエネルギーが削減されます。また、良好なデータ廃棄の実践は、ユーザーの「忘れられる権利」を尊重する新たなデータプライバシー法とも一致します。

## コンテンツの最適化

不要なコンテンツや機能が削除されたことを確認してウェブサイトを最適化したとしましょう。これは節度の部分（通常は難しい部分）です。次に、持続可能な限り効率的に保つために、残されたすべてのものに焦点を当てることができます。

このセクションでは、画像、ビデオ、アニメーションに焦点を当てます。これらに関する詳細は、Media章で取り上げます。

761. *https://www.mightybytes.com/blog/technical-debt-agile-and-sustainability/*

## 画像の最適化

画像はリクエストとページの重さの大きな部分を占めています。これを軽減するためにできることを見てみましょう。すでに言及されているように、必要のない画像はすでに削除されているはずです。

可能な技術最適化から期待できる相対的な利点を詳しく見るために、HTTP Archiveの投稿がそれらを比較しています[762]。HTML（時にはCSS）にますます簡単に依存できるようになっているので、それらをすべて実装するべきです。

### フォーマット（WebP/AVIF）

WebPはすでに広くサポートされており[763]、画像にとって最高のフォーマットの1つです。その圧縮は印象的で、転送および処理されるデータが少なくなります。さらに、広範なサポートを享受しています。AVIFはさらに優れているはずですが、ブラウザーからの広範な採用が進むまで待つ[764]のが賢明かもしれません。その間は、画像にWebPフォーマットを使用してください。アイコンは最適化されたSVG[765]であるべきで、追加のリクエストを避けるためにHTMLに直接含めることもできます。

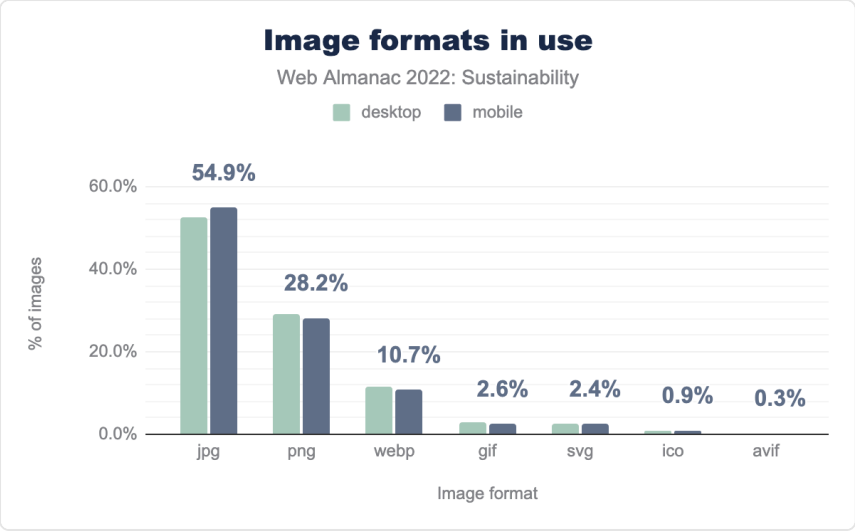



*図20.10. 使用中の画像フォーマット*

762. https://discuss.httparchive.org/t/state-of-the-web-top-image-optimization-strategies/1367
763. https://caniuse.com/webp
764. https://caniuse.com/avif
765. https://jakearchibald.github.io/svgomg/

現在、ウェブサイトのわずか10%がWebPを使用しており、これは昨年[766]よりも改善されていますが、理想からはほど遠い状態です。これは巨大な機会であり、画像の全体的な重さを減らすのに役立ちます。AVIFはさらに後れを取っており、わずかに0%を超える程度ですが、今後数年でこの数字が増えることを期待できます。

### 応答性、サイズ、品質

ユーザーがさまざまなデバイス（主にスマートフォンですが、ゲームコンソール、スマートウォッチ、タブレットなども含まれます）でウェブを閲覧する割合が増えているため、それぞれに適したサイズと重さの画像を配信することを目指すべきです。これはレスポンシブデザインの主要なトピックの1つであり、開発者はこれを自動化するための多くのツールを持っています。

また、人間の目はこの上の違いを検出できないため、品質を85%以上にする必要はほとんどありません。品質を85%に下げることで、画像のサイズを減らすことができます。

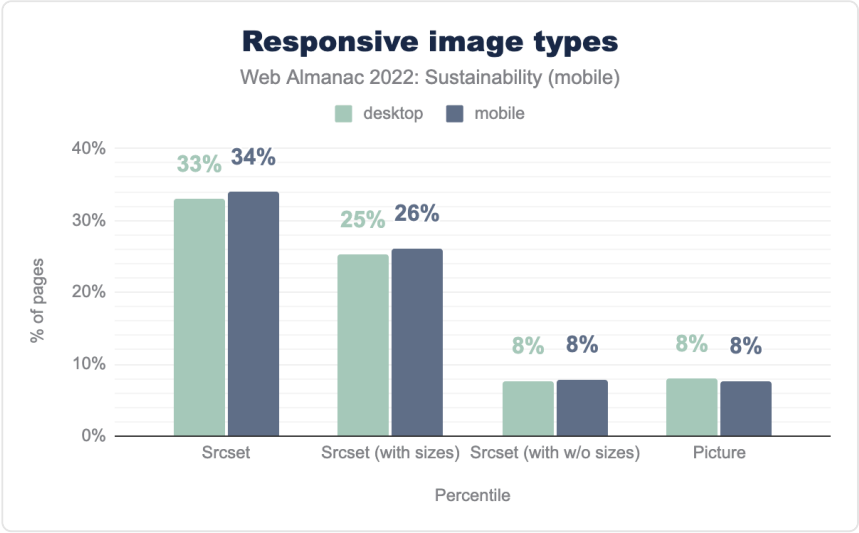



*図20.11. レスポンシブ画像タイプ*

ウェブサイトの約34%が`srcset`属性を使用しており、これはレスポンシブ画像を統合するための素晴らしい方法です。`<picture>`要素も素晴らしく機能し、すでにウェブサイトの7%で見られます。楽観的に考えると、レスポンシブ画像が毎年地盤を固めていると焦点を当てることができますが、それでもウェブサイトの多数派で使用されているわけではありません。しかし、レスポンシブデザインはかなりの時間存在しており、より広く普及している

766. *https://almanac.httparchive.org/ja/2021/media#フォーマットの採用*

べきです。

### 遅延読み込み

最初の読み込みを高速化する簡単な方法は、画像を段階的に読み込むことです：必要なときに必要なもののみを読み込みます。これは遅延読み込みを通じて行われ、ほとんどのブラウザが現在これをネイティブでサポートしています[767]。すべてのユーザーがページを最後までスクロールするわけではないので、現在のユーザーによって見られることがないかもしれない画像の読み込みを避けるべきです。そのため、これは持続可能性とユーザーの両方にとって迅速な勝利です。

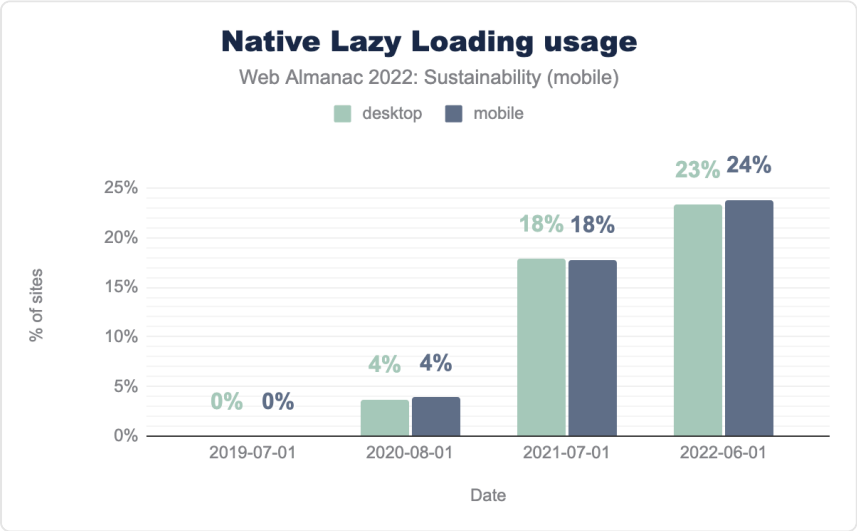



*図20.12. ネイティブの遅延読み込みの使用状況*

このグラフから、ネイティブの遅延読み込みが実装されて以来、より広く採用されていることがわかります。約4分の1のウェブサイトがこれを使用しています。一部はまだJavaScriptライブラリを使用してこの動作を実装しており、このグラフには表示されません。ネイティブの遅延読み込みに切り替えることは、リクエストをわずかに減らし、一部のJavaScript処理を避けるための素晴らしい機会です。

iframeに関するクイックノート：遅延読み込みはiframeにもネイティブで適用可能ですが、持続可能性の理由から、iframeを完全に避けることを検討すべきです。ほとんどの場合、埋め込みビデオやインタラクティブマップを含めたい場合など、ファサード[768]が適切なパターンです。ページの重量とリクエストを増加させることが多く、しばしばアクセシビリティの

767. https://caniuse.com/loading-lazy-attr
768. https://web.dev/third-party-facades/

問題を引き起こすため、ページに直接外部コンテンツを含めるのは良くありません。

### ビデオ

ビデオは、ウェブサイトに含めることができるもっとも影響力のあるリソースの1つです[769]。これについての詳細は、Media章で取り上げます。サードパーティのビデオを統合するには、ファサードを使用する必要があります[770]。さらに、賢く設定する必要があります[771]。たとえば、プリロードと自動再生を避けるべきです。また、ビデオのサイズを迅速に削減する方法を学ぶことができます[772]。

### プリロード

ビデオ（またはオーディオファイル）を自動的にプリロードすることは、すべてのユーザーにとって有用でない可能性のあるデータを取得することを含みます。このようなコンテンツを含むページに多くの訪問者がいる場合、迅速に蓄積する可能性があります。したがって、プリロードは避けるべきであり、ユーザーの操作に基づいてのみ行うべきです。

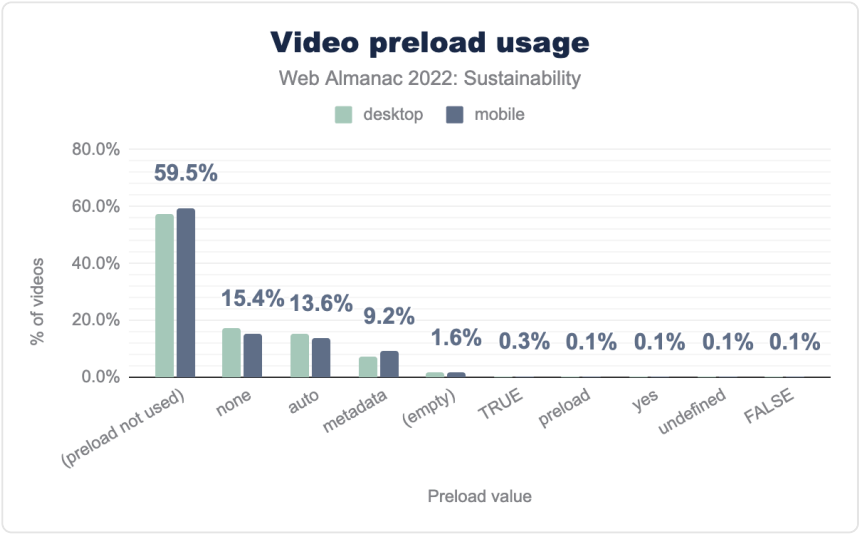



*図20.13. ビデオプリロードの使用状況*

このグラフを見るとき、プリロード属性には3つの可能な値があることを心に留めておく必要があります。「none」、「auto」、「metadata」（デフォルト）。値なしでプリロード

769. https://theshiftproject.org/en/article/unsustainable-use-online-video/
770. https://web.dev/third-party-facades/
771. https://www.smashingmagazine.com/2021/02/optimizing-video-size-quality/
772. https://theshiftproject.org/en/guide-reduce-weight-video-5-minutes/

属性を使用することや、誤った値で使用することは、「metadata」値を使用するのと同じかもしれません。それでもメタデータを取得するためにビデオの最大3%を読み込むことが関与し、それによってかなりの影響を与える可能性があります。持続可能性のためには、「none」が最善の方法です。しかし、これはブラウザに対するヒントに過ぎないことを心に留めておく必要があります。最終的には、ブラウザがビデオのプリロードをどのように処理するかが、あなたが考えていたことと合致しないかもしれません。

この件に関しては、Steve Soudersの記事（2013年）[773]とweb.devの別の記事（2017年）[774]をチェックすべきです。ブラウザやデバイスを設定してデータを節約できるかもしれませんが、ビデオのプリロードは、ブラウザがデフォルトでより持続可能に処理すべきものです。

### 自動再生

プリロードに関して行ったほとんどの考慮事項は、自動再生にも適用されます。これに加えて、関心のないユーザーにデータを読み込み、コンテンツを表示することにより、アクセシビリティの問題を引き起こす可能性があります。一部のユーザーにとって、無許可の動く画像や音声は邪魔であり、ブラウジング体験を妨げる可能性があります。

また、この属性は`preload`設定を上書きする可能性があります。自動再生には明らかに読み込みが必要です。

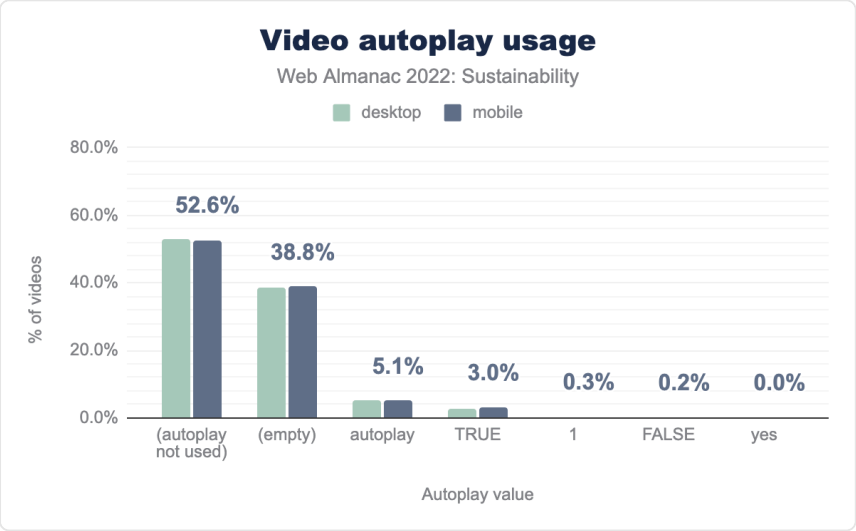



*図20.14. ビデオ自動再生の使用状況*

773. https://www.stevesouders.com/blog/2013/04/12/html5-video-preload/
774. https://web.dev/fast-playback-with-preload/

ウェブサイトの半数以上が自動再生を使用していないのは良いことです。しかし、これはブール属性なので、空の値（または間違った値）を持つことでも自動再生がトリガーされます。上記の理由から、持続可能性とアクセシビリティのためにこれを避けるべきです。

### アニメーション

アクセシビリティのために、ユーザーがある程度のコントロールを持っていない限り、動く部分や点滅する部分は避けるべきです。持続可能性に関しては、アニメーションはコストがかかります。これはバッテリーの放電速度とCPU消費を増加させる傾向があり、結果としてスマートフォンの自立性を低下させる可能性があります。また、いくつかのコードを取得して実行することも関与し、レンダリングを遅延させる可能性があります。

（悪名高い）カルーセルのケースは以下のページで文書化されています：

- なぜそれらを使用すべきではないか[775]
- 代わりに何ができるか[776]

アニメーションを使用する必要がある場合は、GIFを避ける[777]か、少なくとも最適化されたビデオに変換するべきです。なぜなら、アニメーションGIFは非常に重くなる可能性があるからです。

### ファビコンとエラーページ

デフォルトでは、ブラウザはウェブサイトに到着するとファビコンを探します。存在しない場合、ほとんどのサーバーは404エラーとそのウェブサイトの404ページのHTMLを返します。したがって、以下の点を考慮することをオススメします。

- ファビコンを忘れずにキャッシュしてください！
- 404ページのHTMLをできるだけ軽く最適化するか、さらに良い方法として、サーバーを設定して404ページのHTMLではなくテキストのみを送信するようにしてください。

これに関する詳細は、Matt Hobbsの記事[778]を参照してください。

775. *https://shouldiuseacarousel.com/*
776. *https://www.smashingmagazine.com/2022/04/designing-better-carousel-ux/*
777. *https://web.dev/efficient-animated-content/*
778. *https://nooshu.com/blog/2020/08/25/you-should-be-testing-your-404-pages-web-performance/*

## 外部コンテンツの最適化

ウェブ開発の素晴らしい点の1つは、フレームワークやライブラリだけでなくコンテンツにも簡単に依存できることです。しかし、実装が簡単だからといって、それが役立つか、影響が少ないわけではありません。追加したい各外部要素について、それがユーザーに本当に必要かどうかを考えてみてください。そうである場合は、できるだけ効率的に統合してください。また、各コンテンツにはコストがかかることを念頭に置いてください。リクエストや追加のコードだけでなく、時には脆弱性や少なくとも攻撃面の増大を伴うこともあります。

### サードパーティ

サードパーティのリクエストは、すべてのリクエストの45%を占めており、94%のモバイルウェブサイトで少なくとも1つの識別可能なサードパーティのリソースがあります。これは、サードパーティのコードがしばしばウェブページ上で複雑な機能を提供するために使用されるため、驚くべきことではありません。また、ウェブサイトにクロスプラットフォームのコンテンツを含めるための迅速な修正としても機能します。


*図20.15. モバイルページのサードパーティリクエストのうち、グリーンホスティングから提供されている割合。*

ウェブ上のリクエストの大部分を占めるサードパーティのリクエストが、大部分がグリーンホスティングプロバイダーから提供されていることを知ると安心です。

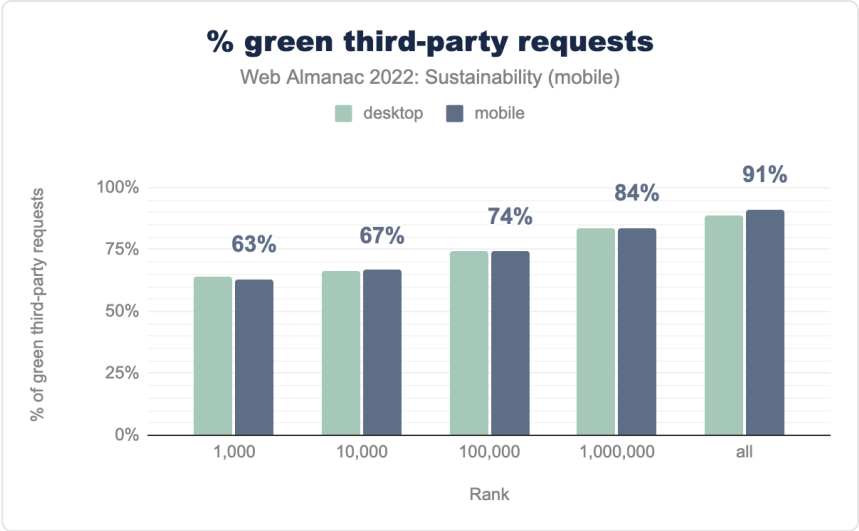



*図20.16. グリーンなサードパーティリクエストの割合*

上のグラフは、グリーンホスティングプロバイダーから提供されるサードパーティリクエストの割合を示しています。ここで興味深いのは、グリーンウェブサイトホスティングとは逆の傾向が見られることです。これは予想外かもしれませんが、もっともリクエストされたトップ5のサードパーティはすべてGoogleのエンティティ（フォント、アナリティクス、アカウント、タグマネージャー、広告）です。これらのエンティティに関連するURLは、ほぼすべてがグリーンホスティングから提供されているとリストされています。

### サードパーティリクエストをより持続可能にする

見てきたように、ほとんどのサードパーティリクエストはグリーンホスティングから提供されています。しかし、とくにランクが高いサイトでは改善の余地がまだあります。ウェブサイトで使用されているサードパーティサービスの持続可能性に興味がある場合、Are my third parties green?[779]はオンラインテストツール、ディレクトリ、およびAPIで、始めるのに役立ちます。*透明性のために、このツールは章の著者の一人によって作成されたものであることを明記しておきます。*

ホスティングを超えて、サードパーティのデータ転送の影響も考慮すべきです。サードパーティサービスのプロバイダーは、他のウェブサイトにコンテンツを統合することを比較的簡単にしますが、それが常にデータ転送量を減らすように最適化されているわけではありません。たとえば、2022年のアルマナックのサードパーティ章では次のように明らかにされています。

779. https://aremythirdpartiesgreen.com/

> *モバイルデバイスでもっとも人気のあるサードパーティはGoogleフォントで、すべてのウェブサイトの62.6%で使用されています。提供されるCSSは最小化されていません。データによると、Googleフォントを使用している平均ページは、それを最小化することで13.3KBを節約できます。*

フォントの場合、自己ホスティングとサブセッティングは、組み合わせるとこのムダを減らすのに役立つ2つの技術です。しかし、ほとんどのサードパーティはスクリプトの形で提供されます。これらはネットワークを介してデータを転送する際のコストだけでなく、エンドユーザーのデバイスでの処理能力も利用します。これらについては、「ジャストインタイム」で読み込むことで影響を減らすことができます。

このパターンは*インタラクション時のインポート*として知られており、ページが最初に読み込まれるときにインタラクティブなコンテンツの代わりに静的なファサードを使用します。その後、ユーザーが要素と対話する直前にコンテンツがリクエストされ、読み込まれます。これにより、初期に転送されるデータが少なくなり、ページを表示する際の処理も減少します。とくにスクリプトが決して要求されない場合はそうです。

## 技術最適化の実装

これまでに、ウェブサイトのコンテンツの持続可能性について多くを見てきました。これで、他のすべての技術最適化に取り組むことができます。ここでも多くの作業が必要であり、ほとんどは自動化されるべきです。再び、これはWeb Almanacの他の章といくつか交差するかもしれませんが、持続可能性についての章を提供し、ウェブサイトをより持続可能にする方法について説明することが目的です。

ウェブパフォーマンスの専門家は技術最適化の分野で多くの作業を行っているので、彼らから学ぶべきことがたくさんあります。ただし、彼らのベストプラクティスが必ずしもウェブサイトをより持続可能にするわけではないことを念頭に置いてください。しかし、物事を軽くシンプルにすることは、持続可能性とパフォーマンスのために素晴らしいことです。アクセシビリティにも言及すると良いでしょう。

### JavaScript

JavaScriptについては、ウェブの成長にどのように役立ってきたか（そして時にはそれがウェブをどのように遅くするか）について多くのことが語られています。ここでは、実装が簡単で持続可能性に大きな利益をもたらすいくつかの簡単な勝利に焦点を当てましょう。これについてもっと学びたい場合は、JavaScript章をチェックしてください。

### 最小化

JavaScriptを最小化することは、ブラウザにとって不要な文字を削除し、ファイルを軽くすることを含みます。

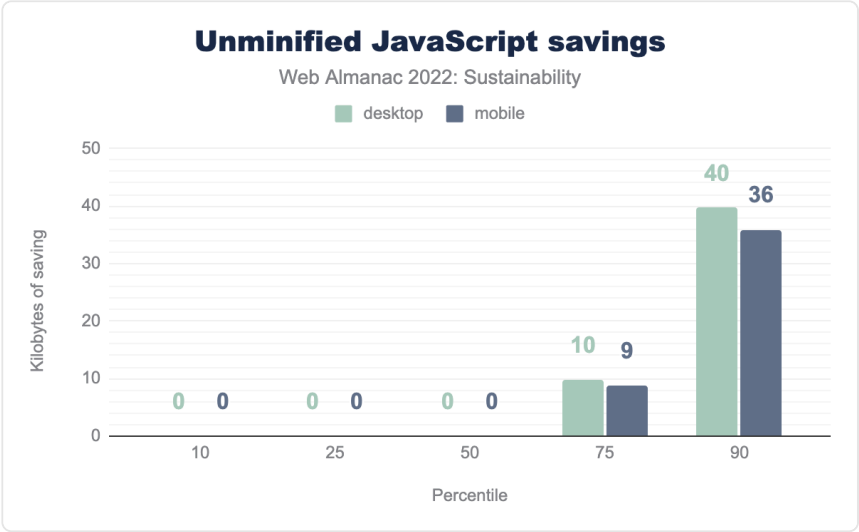



*図20.17. 最小化されていないJavaScriptの節約*

このグラフから、ほとんどのウェブサイトがJavaScriptを効率的に最小化しており、最小化からの利益はそれほど大きくないことがわかります。しかし、実装が簡単で常に有益であるため、なぜ実行しないのでしょうか？

### HTML内でできるだけ少なく含める

インラインコードは悪い習慣であり、持続可能性にとってさらに悪いです。HTMLを重くして読み込みや処理を困難にすることは望ましくありません。JavaScriptをインラインで書くことは、時に最適化（およびメンテナンス）を困難にすることもあります。

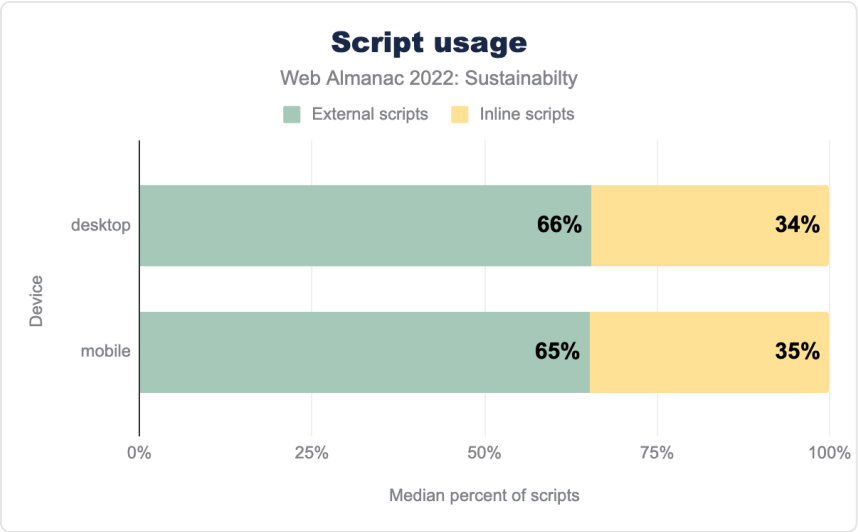



*図20.18. スクリプトの使用*

ウェブサイトの約3分の1がJavaScriptをインラインで使用しています。これはCMSでよく見られることです。

## CSS

CSSは、とくにウェブサイト上の画像の数を制限したり、前述の章で述べたアニメーションを作成したりする場合に、持続可能性のための素晴らしいレバーになり得ます。効率的なCSSの書き方に関するドキュメントは見つけることができますが、ここではすべての場所で実装されるべき標準的な最適化に焦点を当てます。これについてもっと学びたい場合は、CSS章を参照してください。

### 最小化

CSSと同様に、JavaScriptを最小化することは、ブラウザにとって不要な文字を削除し、ファイルを軽くすることを含みます。

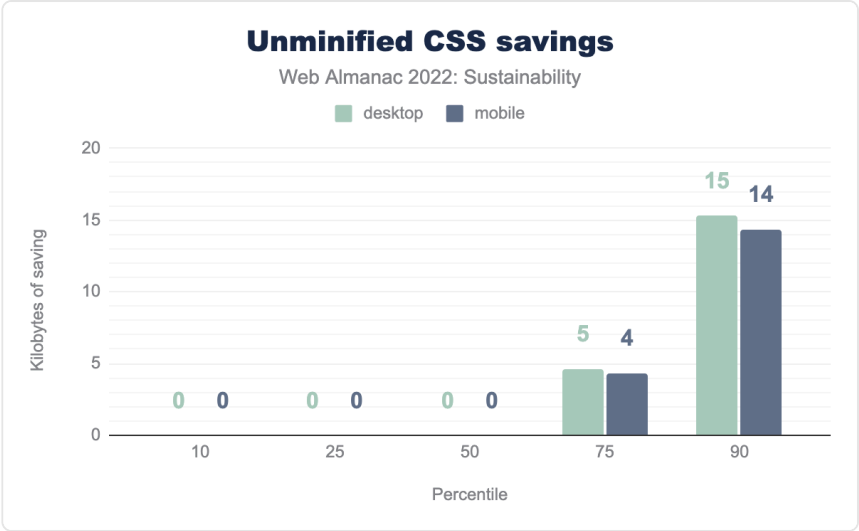



*図20.19. 最小化されていないCSSの節約*

ほとんどのウェブサイトで最小化されていないCSSは存在せず、潜在的な利益は非常に軽いようです。しかし、CSSを最小化することは依然として有益であり、すべてのウェブサイトで実装されるべきです。

**HTML内に可能な限り少なく含める**

JavaScriptと同様に、CSSをインラインで書くことはHTMLファイルのサイズとウェブサイトのパフォーマンスにとって悪影響を及ぼす可能性があります。これは、CMSを使用して構築されたウェブサイトやクリティカルCSSメソッド[780]に依存しているウェブサイトでよく見られます。

780. *https://web.dev/extract-critical-css/*

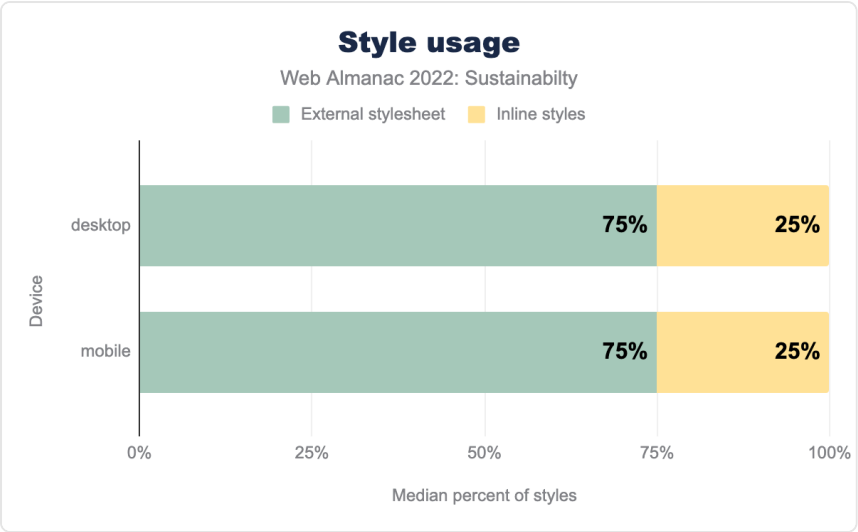



*図20.20. スタイルの使用*

このグラフから、ウェブサイトの約四分の一が依然としてCSSをインラインで使用していることがわかります。持続可能性の理由から、これは避けるべきです。

## CDN

CDNの実装は、ウェブサイトをより持続可能にするのに役立ちます。これにより、ユーザーに可能な限り近い場所でアセットを提供し、時には自動的に最適化を支援します。

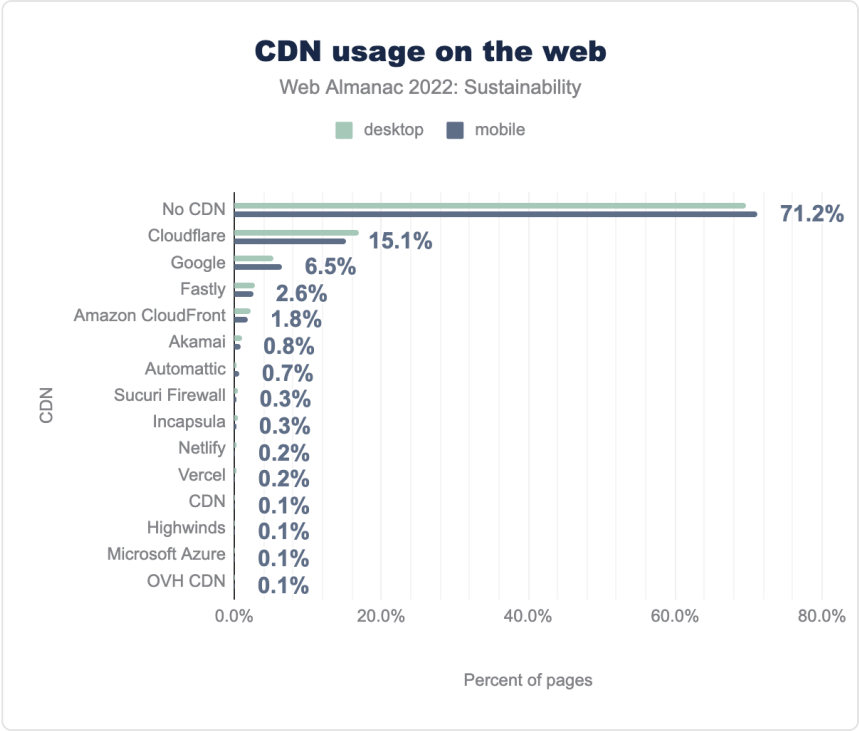



*図20.21. ウェブ上のCDNの使用*

これらの明らかな利点にもかかわらず、70%以上のウェブサイトがまだCDNを使用していません。

### テキスト圧縮

ウェブサイトのテキストアセットを圧縮する[781]ことは、いくつかの（簡単な）サーバー側の設定を必要とするかもしれません。HTML、JavaScript、CSSなどのテキストファイルは、BrotliまたはGzip形式で圧縮され、容易に軽量化されます。

781. https://web.dev/uses-text-compression/

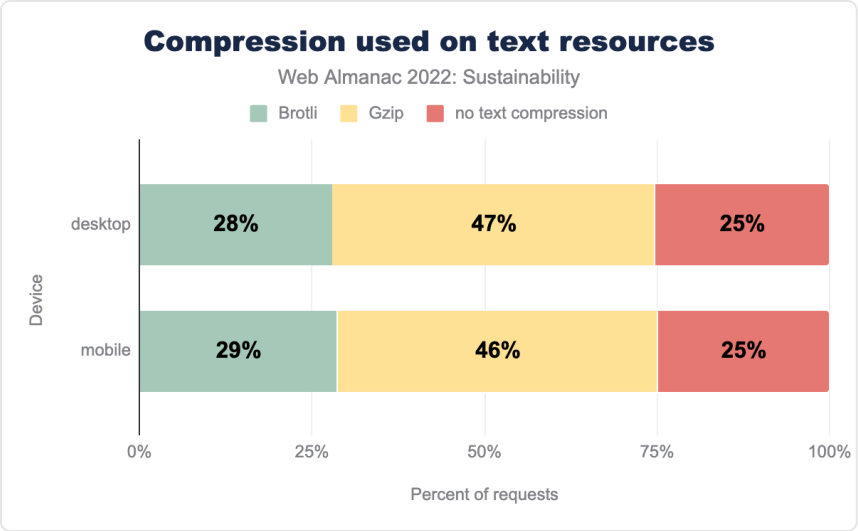



*図20.22. テキストリソースで使用される圧縮*

それにもかかわらず、ウェブサイトの4分の1はまだテキスト圧縮を実装していません。Gzipは全面的にサポートされているので、安心して使用してください。

### キャッシング

キャッシング[782]はブラウザのキラー機能の1つですが、常に完璧に実装するのは簡単ではありません。キャッシングは、ブラウザがすべてのリソースを毎回要求するのを防ぐため、持続可能性に非常に良いです。

782. https://web.dev/uses-long-cache-ttl/

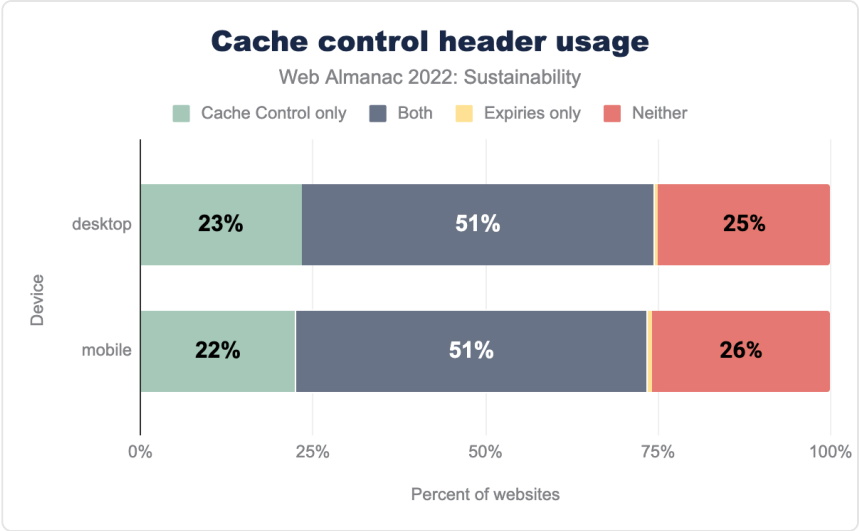



*図20.23. キャッシュコントロールヘッダーの使用状況*

このページからわかることは、ウェブサイトの四分の一以上がまったくキャッシングを使用していないということです。これは持続可能性とパフォーマンス、そして明らかな理由からユーザーにとって大きな損失です。

## SEOと持続可能性

アクセシビリティと同様に、検索エンジン最適化の取り組みに関連する排出量を特定するのは困難です。しかし、SEOは重要な持続可能性の影響を持っています。

- キーワードリサーチはSEOの基盤であり、著者が作成するコンテンツを特定のターゲットユーザーのニーズと一致させるのに役立ちます。
- 構造化データは検索エンジンがページコンテンツをよりよく理解するのを助け、結果としてより関連性の高い情報を提供できます。
- 検索最適化されたコンテンツは、通常、そのフォーマットによって素早くスキャンが可能で、明確に書かれているため、理解しやすくなります。
- コンテンツのパフォーマンスを時間とともに分析する（直帰率、スクロールの深さなど）ことで、著者は公開されたコンテンツを改善し、ユーザーのニーズにより適切に対応できます（検索結果も改善します）。分析に使用されるツールによっては、プライバシーに関する問題が発生する可能性があります。

これらの努力は、ユーザーが自分のニーズに関連する情報を探すために費やす時間を減らすのに役立ちます。これにより、エネルギーの使用が減少します。

逆に、検索エンジンは、チュートリアルやガイドなどの長文コンテンツをしばしば報酬として与えますが、これはリスト記事やその他の短い形式のコンテンツよりも多くの帯域幅（およびエネルギー）を使用する可能性があります。多くの持続可能性関連の概念と同様に、有用で魅力的なコンテンツを作成し、パフォーマンスと効率を最適化する間の適切なバランスを見つけることが鍵です。

数年前、Googleは単一の検索が60Wの電球を17秒間点灯させるのに必要なエネルギーを記録しました。2022年には、検索エンジン最適化の具体的な環境への影響を定量化するためのさらなる研究が必要ですが、良いSEO作業の持続可能性への影響は明確です。最適化されたコンテンツはエンドユーザーのエネルギー消費を減らします（イライラを減らすことを言うまでもありません）。

## 持続可能なデータおよびコンテンツ管理

上記で述べたように、構造化データは検索エンジンがウェブページをよりよく理解し、より関連性の高い結果を生成するのを助けます。しかし、データとの集団的な関係はSEOを超えて持続可能性の影響を持っています。

- 使用されていない重複した古いまたは、不完全なデータや管理が悪いコンテンツはサーバーのスペースを消費し、ユーザーにエラーを引き起こし、ホストおよび維持にエネルギーを必要とします。
- 定期的なコンテンツ監査[783]と明確なコンテンツガバナンス計画[784]は、コンテンツのパフォーマンスを測定し、時間とともに古くなったりパフォーマンスが低下したりするコンテンツを剪定するのに役立ちます。
- サードパーティサービスはウェブページに小さなコードスニペットを挿入するだけかもしれませんが、収集されたデータは非常にリソース集約的です。たとえば、1つのデジタル広告が最大323トンのCO2e[785]を生産することがあります。
- マーケティングオートメーション、メールマーケティング、CRMシステムなどのデータツールメーカーは、しばしば製品管理の努力をデータ収集に焦点を当てていますが、それを最適化するわけではありません。実際、これらのプラットフォームのいくつかはデータ使用に対して課金し、そのビジネスモデルは持続可能性の原則と相反することがあります。

783. *https://www.mightybytes.com/blog/how-to-run-a-content-audit/*
784. *https://www.mightybytes.com/blog/content-governance/*
785. *https://www.businessinsider.com/making-net-zero-possible-the-hidden-impact-of-digital-ads-2022-7*

- 同様に、多くの組織には明確なデータ廃棄ポリシーがなく、効果的なデータ管理についてチームをトレーニングしていません。これは持続可能性の問題だけでなく、プライバシーおよびセキュリティの問題でもあります。

これらは、管理が悪いコンテンツとデータに関連する長い持続可能性の問題リストからのいくつかの例です。組織は定期的にコンテンツとデータ管理の慣行を監査し、効率を向上させリソース使用を減らす[786]ための改善を図るべきです。

## 人気のフレームワーク、プラットフォーム、CMS

オンラインプラットフォームやCMSツールは、ウェブ上で出版やビジネスを行いたい人々の参入障壁を下げるのに役立ちます。同様に、開発フレームワークやサイトジェネレーターは、ウェブのために構築する人々がプロジェクトを迅速に開始し、一般的な開発問題を解決するデフォルトやソリューションを活用できるようにします。

以下のチャートは、もっとも人気のある5つのeコマースプラットフォーム、CMSツール、サイトジェネレーターツールの中央ページ重量を示しています。

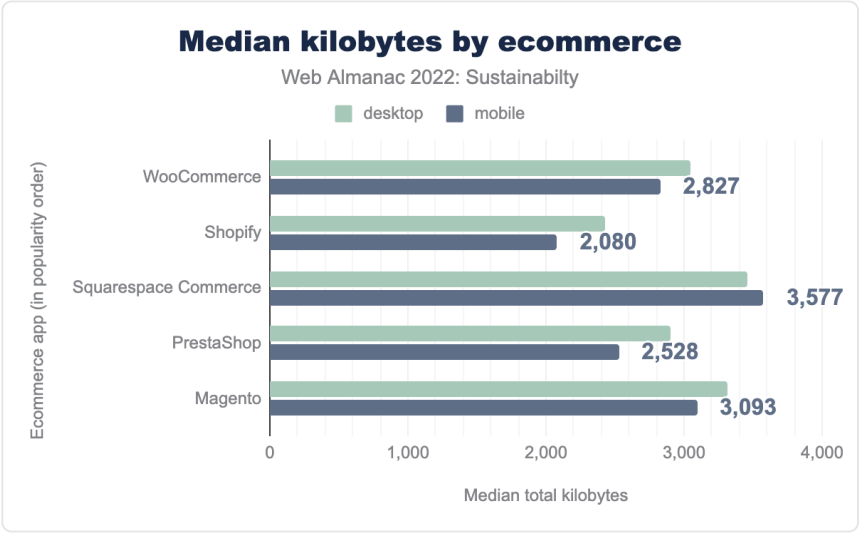



*図20.24. eコマース別の中央値キロバイト*

786. https://www.mightybytes.com/blog/design-a-sustainable-data-strategy/

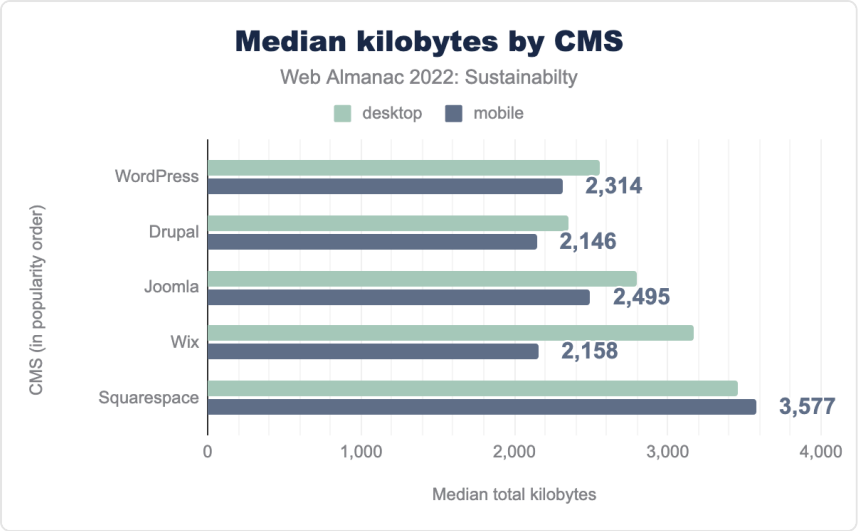



*図20.25. CMS別の中央値キロバイト*

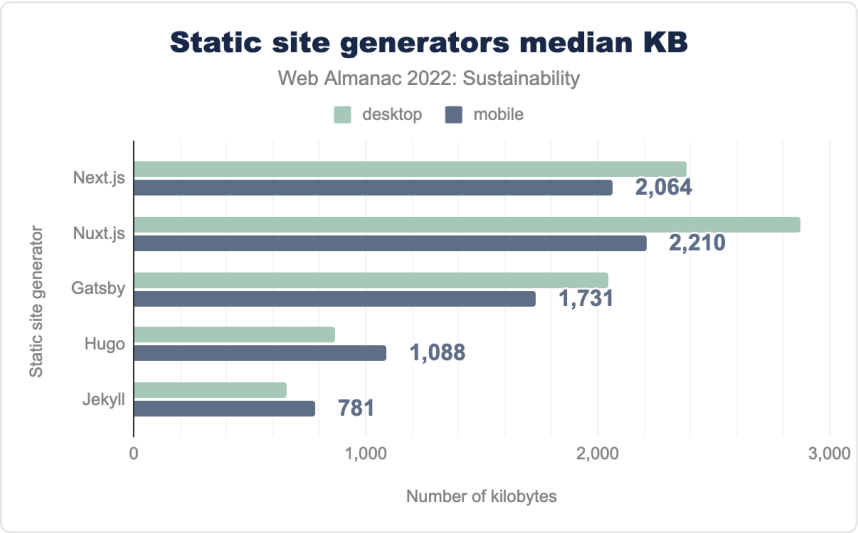



*図20.26. 静的サイトジェネレーターの中央値KB*

とくに興味深いのは、リストされたプラットフォーム/ツールの中で、全体の中央値（2,019 KB）よりもモバイルページの重量が少ないのは3つだけであることです。これらはすべて静的サイトジェネレーターカテゴリにあり、とくにHugoとJekyllの場合は、これらのツールが使用されているウェブサイトの種類、主にブログやテキストコンテンツ、JavaScriptへの依存度が少ないことに起因する可能性が高いです。また、SSGはパフォーマンスを念頭に置い

て使用されることが多いため、商品理由のみでCMSを使用している平均的なウェブサイトよりもさらに最適化される可能性が高いことも指摘しておきます。

3つのセグメントを横断して見ると、デスクトップとモバイルのページサイズの間に大きな差があることがわかります。詳しく見ると、これはいくつかのプラットフォームがモバイルデバイス用に画像の最適化を行っているためのようです。これを強調するために、デスクトップとモバイルのサイズに大きな違いを示すWixなど、他の人気プラットフォームと比較したCMSカテゴリを見てみましょう。

| CMS | デバイス | HTML | JavaScript | CSS | Image | Fonts |
|---|---|---|---|---|---|---|
| WordPress | デスクトップ | 40 | 521 | 117 | 1,202 | 166 |
| WordPress | モバイル | 37 | 481 | 115 | 1,100 | 137 |
| Drupal | デスクトップ | 23 | 416 | 68 | 1,279 | 114 |
| Drupal | モバイル | 23 | 406 | 66 | 1,158 | 92 |
| Joomla | デスクトップ | 26 | 452 | 86 | 1,690 | 104 |
| Joomla | モバイル | 22 | 401 | 83 | 1,504 | 82 |
| Wix | デスクトップ | 123 | 1,318 | 86 | 647 | 197 |
| Wix | モバイル | 118 | 1,215 | 9 | 290 | 148 |
| Squarespace | デスクトップ | 27 | 997 | 89 | 1,623 | 214 |
| Squarespace | モバイル | 27 | 990 | 89 | 1,790 | 202 |

*図20.27. CMS、デバイス、およびリソースタイプ別の中央値キロバイト*

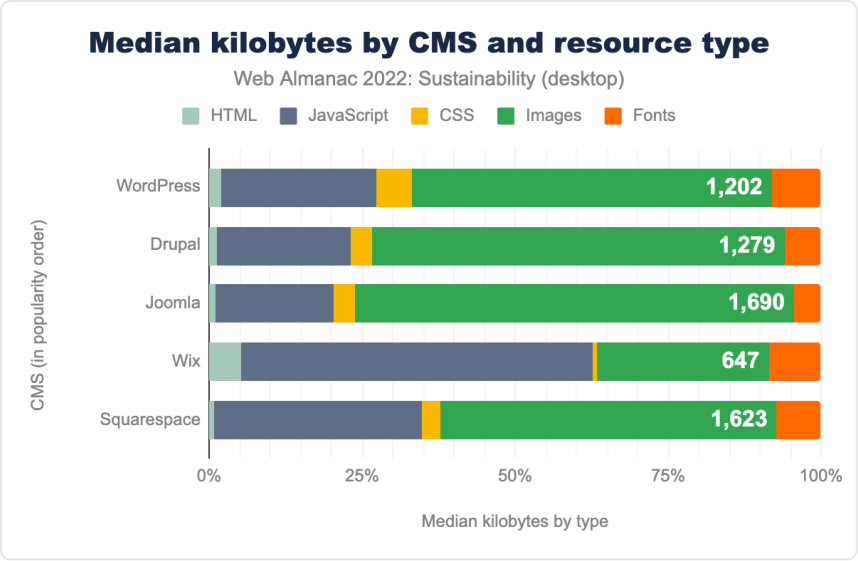



*図20.28. デスクトップでのCMSおよびリソースタイプ別の中央値キロバイト*

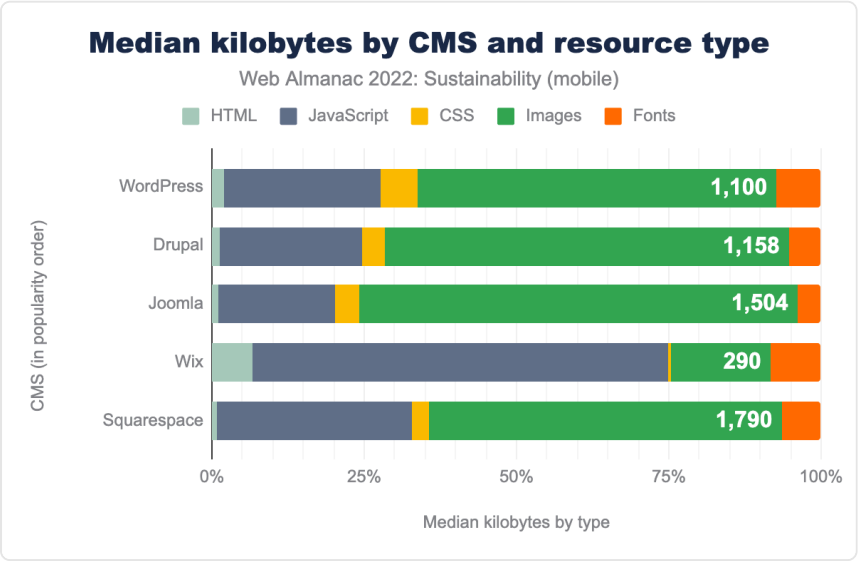



*図20.29. モバイルでのCMSおよびリソースタイプ別の中央値キロバイト*

上の表およびグラフは、Wixがモバイルイメージの最適化により積極的に取り組んでいることを示しています。Next.jsやNuxt.jsのようなフレームワークを使用しているサイトジェネレーターのセグメントでも同様のパターンが見られます。

この見かけ上小さな洞察は、プラットフォームおよびフレームワークがより持続可能なウェブサイトを提供するために果たす重要な役割を捉えています。適切なデフォルトを適用することにより、プラットフォーム開発者およびフレームワークの著者は、開発者がツールを活用して、**green by default**のサイトを作成できるように支援できます。

## 結論

これは持続可能性に関する最初の **Web Almanac** の章であり、世界中で発生している干ばつ、熱波、その他の気候イベントという象徴的な年に行われます。確かに何か問題があり、見過ごすことがどんどん難しくなっています。ウェブはこれに一役買っており、その環境への影響を理解することが優先事項です。**HTTP Archive** からのアクセス可能なデータを鑑みると、この章はメトリクスを集め、持続可能性に関してウェブの状態を振り返る絶好の機会です。

利用可能なメトリクスに基づくと、一部のベストプラクティスがすでに採用され、徐々に広がっているのが見られます。しかし、まだ多くのことが行われる必要があります。これらのアクションのいくつかは実装が容易ですが、非常に有益であることが証明されるかもしれません。また、ベストプラクティスと実際のデバイスでの措置は、持続可能性の継続的な改善に不可欠です。

持続可能性についての認識を高め、ベストプラクティスを発見し、実装することは、皆に委ねられています。これについての言葉を広め、議論を生み出すことが不可欠です。アクセシビリティと同様に、これは私たち全員に関係しており、ウェブをより持続可能にするために必要なすべてを持っています。

ここで見たほとんどのことは、開発者がウェブサイトをより持続可能にするためにできることを示しています。しかし、さらに進む必要があります。デザインを通じて節度を求める必要があります。プロジェクトマネージャーは持続可能性を優先事項とし、これが後で常に対処できるわけではないことを確認する必要があります。企業は、ビジネスモデルをより持続可能なものにすることを考える必要があります。

この章を通じてウェブの持続可能性についてより意識を高め、現在の持続可能なウェブサイトの状況を理解し、このトピックを扱い、言葉を広めるためのいくつかのツールと手がかりを提供したいと考えています。

### 行動できること

ウェブサイトの持続可能性は考慮されるべきです。今日に至るまで、まだ多くのことが行われる必要があります。必要であれば、上で推奨されたいくつかのリソースから始めて、この件についての認識を高め、言葉を広めるべきです。

既存のウェブサイトで始めるには、次のことができます。

- 画像を最適化する（WebP + 遅延読み込み + レスポンシブ性 + キャッシュ + 品質）し、これが自動的に行われるようにする
- ビデオの実装を避ける。必要な場合、自動再生やプリロードはしない
- より持続可能なホスティングを探す

その後、以下を行うべきです。

- あなたのサードパーティーを整理する。
- 簡単な技術的最適化から始めて、CSSとJavaScriptを最適化し、それを自動化する。
- デザインを見直して、ページをよりシンプルにする（視覚的コンテンツを減らす、アニメーションを減らすなど）し、ユーザージャーニーを合理化する

ウェブサイトをより持続可能にすることは、継続的な改善の一部です。すべてを一度に行うことはできませんし、すべきではありません。ベストプラクティスと測定に依存して、正しい方向に進んでいることを確認してください。既存のウェブサイトで作業をしている場合でも、ゼロから新しいものを作成している場合でも、チーム内の全員を巻き込むか、少なくともこのトピックを認識していることが重要です。

あなたのユーザーの中には、あなたのウェブサイトがより持続可能であること、およびそれをどのように達成したかを知りたいと思う人もいます。そして、それはすべての人にとって利益になります。

*特別な感謝を込めてTom Greenwood*[787]*、Hannah Smith*[788]*、Eugenia Zigisova*[789]*、Rick Viscomi*[790]*およびこの章を可能にした他の素晴らしい人々全員に。*

787. *https://www.wholegraindigital.com/team/tom-greenwood/*
788. *https://twitter.com/hanopcan*
789. *https://twitter.com/jevgeniazi*
790. *https://twitter.com/rick_viscomi*

## 著者


### Laurent Devernay

@ldevernay ldevernay https://ldevernay.github.io/

ローラン・デヴェルネは、Greenspector[791]のデジタル節約エキスパートです。彼のブログは、個人的なもの[792]や会社のためのもの[793]がありますが、ほぼ常にウェブの持続可能性についてです。これにより、彼は熱心な支持者であるとも、一途なマニアであるとも言えます。

### Gerry McGovern

gerrymcgovernireland

ジェリーは8冊の本を出版しています。最新作であるWorld Wide Waste[794]は、デジタルが環境に与える影響を検証しています。彼は、人々にとって本当に重要なことを特定するのに役立つ研究方法であるTop Tasks[795]を開発しました。

### Tim Frick

@timfrick timfrick https://www.mightybytes.com/

Tim Frick[796]は1998年にデジタルエージェンシーMightybytes[797]を設立し、非営利団体、ソーシャルエンタープライズ、目的を持った企業が問題を解決し、その影響を増幅し、測定可能なビジネス成果を達成するのを支援しています。Mightybytesは、善のためにビジネスを使う認定Bコーポレーション[798]です。認定Bコーポレーションは、社会的および環境的パフォーマンス、透明性、説明責任のもっとも厳しい検証基準を満たしています。Timは4冊の著書を持ち、その中には*Designing for Sustainability: A Guide to Building Greener Digital Products and Services*[799]が含まれます。熟練のパブリックスピーカーとして、彼は定期的に会議で発表し、持続可能なデザイン、影響の測定、デジタル経済での問題解決に関するワークショップを提供しています。また、Climate Ride[800]、B Local Illinois[801]、Alliance for the Great Lakes[802]など、いくつかの非営利団体の理事を務めてきました。


791. https://greenspector.com/en/home/
792. https://ldevernay.github.io/
793. https://greenspector.com/en/blog-2/
794. https://gerrymcgovern.com/books/world-wide-waste/
795. https://gerrymcgovern.com/books/top-tasks-a-how-to-guide/
796. https://www.mightybytes.com/teammember/tim-frick/
797. https://www.mightybytes.com/
798. https://www.mightybytes.com/b-corporation/
799. https://www.oreilly.com/library/view/designing-for-sustainability/9781491935767/
800. https://www.climateride.org/
801. https://www.illinoisbcorps.org/
802. https://greatlakes.org/

# 部 IV 章 21
# ページの重さ


*Jamie Indigo と Dave Smart によって書かれた。*
*Chris Steele によってレビュ。*
*Danielle Rohe による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

過去10年間でモバイルページの重量が594%増加したことを示します。この期間に、影響を緩和するためのパフォーマンス向上技術が登場しました。最近のパフォーマンス測定方法であるCore Web Vitalsは、ページの重さを要因として除外しています。

Googleが推進するCore Web Vitalsイニシアチブは、上の折り返しのコンテンツがどれだけ早く見え、使えるかにパフォーマンスの認識をシフトしている一方で、大きなネットワークペイロードは依然として長いロード時間と相関しています。

多くのウェブサイトの構築者は、高速のデスクトップ接続の利点を享受しており、限られた、しばしば高価なモバイルネットワークアクセスを経験していません。

国際電気通信連合のグローバル・コネクティビティ・レポート[803]によると、世界の世帯の

803. https://www.itu.int/itu-d/reports/statistics/2022/05/30/gcr-chapter-2/

66%がインターネットアクセスを持っています。低所得国では、高所得国の91%に比べて22%しかアクセスがありません。途上国の農村地域では、4Gが有意義な接続の最低限であるにもかかわらず、3Gしか利用できないことが多いです。

世界の低所得国と中所得国の半数以上で、住民は月平均収入の2%以上を1GBのモバイルブロードバンドデータ[804]に支払っています。

ページの重さは依然として重要です。不運なタイミングで弱いネットワーク接続を経験している場合でも、インターネットアクセスがメガバイト単位で課金される市場に住んでいる場合でも、ページの重さが増加すると情報の利用可能性が低下します。

## ページの重さとは何か？

ページの重さとは、ウェブページのバイトサイズを指します。もはや1994年ではないため、ウェブページはブラウザのアドレスバーで表示されるURLのHTMLだけではまれです。むしろ、ブラウザで表示されレンダリングされたウェブページは、特定の要素とアセットを使用します。

これが、ページの重さがページの作成に使用されるすべてのアセットを含む理由です。これには以下が含まれます。

- ページ自体を構成するHTML
- ページに埋め込まれた画像やその他のメディア（ビデオ、オーディオなど）
- ページのスタイリングに使用されるカスケーディングスタイルシート（CSS）
- インタラクティビティを提供するJavaScript
- 上記の1つ以上を含むサードパーティリソース

各リソースはページのバイトと、ページの転送、処理、レンダリングに関わる計算リソースに追加されます。スクリプトのようなリソースは、ダウンロード、解析、コンパイル、および実行が必要なため、CPU使用量の形で追加のオーバーヘッドがあります。

ページの重さが膨れ上がるにつれて、影響を軽減するための勇敢な努力や方法が提案されています。それでも、リソース割り当てとページの重さの複雑な関係のからくりは、ほとんどのユーザーには見えません。

リソースのページの重さの影響をさらに詳しく調査しましょう：ストレージ、転送、およびレンダリング。

804. *https://digital-world.itu.int/ministerial-roundtable-cutting-the-cost-can-affordable-access-accelerate-digital-transformation/*

## ストレージ

最終的にウェブページを構成するすべてのリソースはどこかに保存する必要があります。ウェブサイトの場合、それはしばしば複数の場所を意味し、それぞれが自身のコストとオーバーヘッドを持っています。

ウェブサーバー上のストレージは、ディスクストレージ上では比較的低コストで、比較的スケーラブルです。もしウェブサーバーがメモリから提供している場合は少し高価かもしれません。これらのリソースは、中間キャッシュやCDNで複製される場合もあります。

それがソースですが、最終的にはリソースはクライアント側にも何らかの形で保存する必要があります。そこでは、とくにモバイルデバイスの場合、ストレージがより限定されている可能性があります。巨大で膨れ上がったファイルを提供することは、ユーザーのキャッシュを満たし、他の有用なリソースを押し出す可能性があります。

解像度がプリントメディアに適している未最適化の画像や、複数のメガバイトに及ぶ巨大なビデオファイルが依然として定期的に見られます。

これの多くは、メディアの適切なフォーマットとコーデックを選び、サイズと品質の両方に注意を払うことで軽減できます。Squoosh[805]のようなサービスは、可能な限り小さなサイズで画像を最大限に活用するのに役立ち、画像CDN[806]の専門家がこれの多くを自動化できます。

メディアは定期的にもっとも重い要素ですが、節約できるのはメディアだけではありません。テキストリソースも圧縮して最小化できます。

リソースをダイエットにすることは、これまでになく簡単になりました！

## 転送

はじめてウェブページを訪問するとき、そのページが要求するすべてのリソースは、サーバーからあなたのデバイスへインターネットを介して転送される必要があります。

これは超高速で高月間使用量のブロードバンド接続を介して行われることがありますが、遅くて高価でキャップされたモバイル接続や、衛星を介して行われることもあります。

ページの重さが大きいほど、これにかかる時間は長くなります。また、データプランが低いユーザーにとってはより高価になることもあります。

`preconnect`、`preload`、Priority Hints[807]などロードされるものの順序を管理し、知覚さ

805. https://squoosh.app/
806. https://web.dev/image-cdns/
807. https://web.dev/articles/fetch-priority

れるロード時間を助ける最適化がありますが、最終的にはリソースはまだ転送され受信される必要があり、最良の最適化はより小さなリソースを提供することです。

### レンダリング

リソースの取得は、ウェブサイトを画面上に描画し、ユーザーに表示するための最初のステップに過ぎません。それを行うために、ウェブブラウザはページをレンダリングするためにすべての関連リソースを使用する必要があります。

ここでページの重さは重要な役割を果たします。まず、転送が遅い場合（ファイルが大きいため）、ブラウザがこれらをレンダリングするために作業を開始するまでの時間が長くなります。

大きなファイルは、ネットワーク上で受信された後も影響を及ぼします。大きなファイルは読み取り、処理、およびレンダリングのためにより多くの処理能力とメモリを必要とします。これにより、コンテンツの一部、またはすべてがユーザーに表示されるまでの遅延が長くなることにつながります。遅延が長ければ長いほど、ユーザーがページを放棄してより応答性の高いサイトから情報を求める可能性が高くなります。

ここでのJavaScriptはとくに懸念されます。ダウンロードする必要があるだけでなく、解析して実行する必要もあります。

巨大なファイルは、ユーザーが最初にダウンロードするのを待つ場合でも、継続的なパフォーマンスのペナルティを持つことがあります。これらはブラウザに利用可能なすべてのクライアントサイドリソースを消費し、パフォーマンスを遅くするか、ブラウザを完全にクラッシュさせる可能性があります。

最新の高性能スマートフォン、ラップトップ、タブレットでは、これらの大きなファイルを処理するパワーがあり、顕著なパフォーマンスの問題なく処理できるかもしれませんが、古いまたは低パワーのデバイスはうまく処理できない可能性があります。これらは、遅くておそらく高価なモバイル接続を持つデバイスでもあり、もっとも単純で信頼性の高いアクセスを必要とする人々にとって「二重のペナルティ」になる可能性があります。

## 私たちは何を配信しているのか？

1995年にHTML 2.0が導入される前、ページの重さは予測可能で管理しやすかったです。唯一のアセットはHTMLでした。RFC1866[808]は `<img>` タグを導入しました。画像がウェブページに含まれるようになると、ページの重さは劇的に増加しました。

808. https://www.rfc-editor.org/rfc/rfc1866

HTML仕様のさらなるバージョンでは、ページの重さを増加させる可能性のあるさらに多くの機能が追加されました。たとえば外部CSSを使ってページ間で一貫したスタイリングを可能にしました。

1996年にはJavaScriptがはじめて登場し、2005年にはXHRが登場し、2006年にはjQueryのようなライブラリが誕生し、Angular、React、Vueなどのフレームワークに続き待ち構えていたリバイアサンであるJavaScriptを完全に解き放ちました。

ページの重さの増加は、機能を維持しながらパフォーマンスを向上させることを目的としたファイルタイプの増加をもたらしました。資産タイプ別にそれらを検討することで、トレードオフが強調されます。

### 画像

画像は、パフォーマンス向上技術とアセットのバイトサイズのバランスをとる代表例です。これらの静的ファイルは、ウェブページを構築し、レンダリングするためのリソースとして機能します。ウェブの視覚的な性質が高まるにつれて、このメディアタイプはもっとも普及しているアセットとしての地位を保持するでしょう。

PNG、JPEG、GIFなどの古い形式は、より広い「歴史的な」ブラウザサポートの遺産を楽しんでいます。パフォーマンスに焦点を当てたファイルタイプWebP[809]は顕著なブラウザサポートを獲得し、現在全世界のユーザーの97%[810]が利用可能です。

ウェブ上の画像の使用とその影響についての調査と考察については、メディア章を参照してください。

## JavaScript

JavaScriptはウェブ上でもっとも人気のあるクライアントサイドスクリプト言語です。98%のウェブサイトがインタラクティブなオンラインコンテンツを作成するために使用しており、他のソースもそれが非常に高いと同意しています[811]。適度に使用された場合は素晴らしいですが、JavaScriptの魅力は、パフォーマンス、検索エンジン最適化、ユーザーエクスペリエンスの問題にもつながる可能性があります。

インターネットでもっとも好まれる問題を引き起こす魔法のアイテムについての詳細な洞察については、JavaScript章を参照してください。

809. https://developers.google.com/speed/webp
810. https://caniuse.com/webp
811. https://w3techs.com/technologies/details/cp-javascript

### サードパーティサービス

ページの重さは、元のホストのアセットに限定されません。ページによって要求されるサードパーティリソースが、分析、チャットボット、フォーム、埋め込み、分析、A/Bテストツール、データ収集の形で重さに追加されます。

サードパーティ章によると、モバイルデバイスのウェブサイトの94%が少なくとも1つのサードパーティリソースを使用しています！それぞれがページ重さのバイトサイズに貢献しています。

### その他のアセット

ウェブの基本的な構成要素は25年以上にわたって比較的変わらずに残っています。ウェブ体験の豊かさが増すにつれて、フォントとビデオの使用も増えています。

これらの増加は他のファイル重量の増加と同様に、とくに100パーセンタイルに顕著です。


*図21.1. モバイルページでの最大フォント使用量。*

2021年には、モバイルサイトの100パーセンタイルがフォントファイルとして20,452キロバイトを使用しました。2022年には、これらの外れ値は110メガバイトに膨らみました。この540%の成長は、デスクトップの前年比で見られませんでした。デスクトップは2021年に66,257キロバイト、2022年に68,285キロバイトでした。

しかし、100パーセンタイルはおもしろい調査対象ではありますが、常にウェブの最悪の部分を示します。90パーセンタイルでは、モバイルフォントの重さはそれほど極端ではありませんでしたが、それでも大きく401キロバイトでした。

ウェブのタイポグラフィ的な性質に関するさらなる洞察は、フォント章で見つけることができます。

## 数字によるページの重さ

これで、ページの重さを考慮する際に何に主に興味があるかがわかりました。詳細について詳しく見ていきましょう。

## リクエストの量

リクエストされたバイト数の合計だけでなく、ページを作成するために行われたリクエストの数もページのパフォーマンスに影響を与えることがあります。したがって、これもページの重さの一部として考慮します。

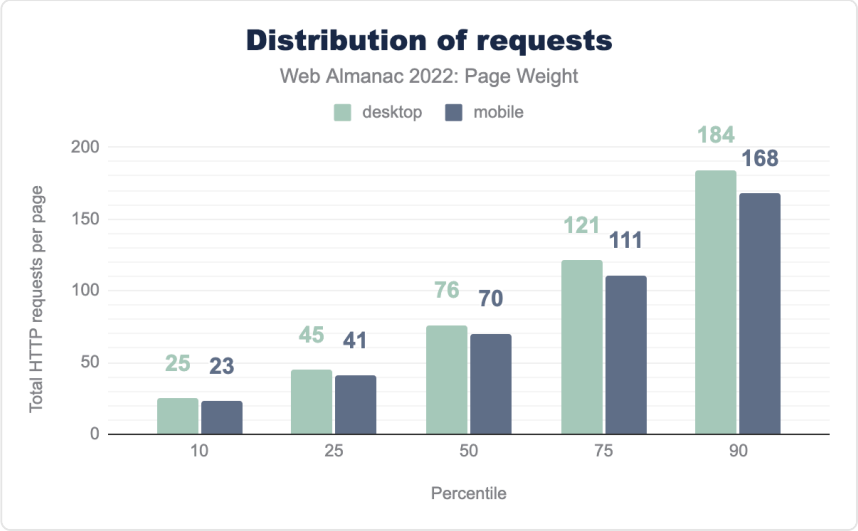



*図21.2. リクエストの分布。*

中央値（50パーセンタイル）のページでは、デスクトップページのロードには76のリクエストがあり、モバイルでは70のリクエストがあります。すべてのパーセンタイルでデスクトップとモバイルの差はほとんどありません。

昨年の中央値のデスクトップリクエストは74でしたので、昨年との大きな違いはありません。

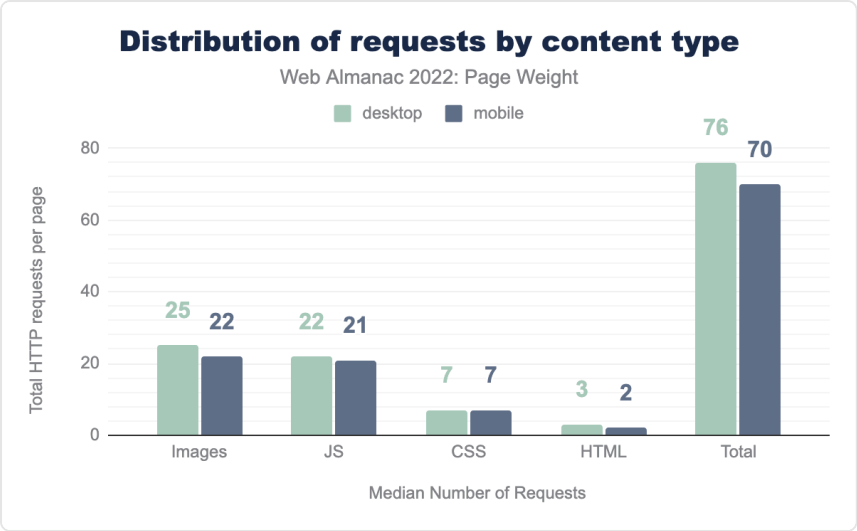



*図21.3. コンテンツタイプ別の中央値のリクエスト数。*

タイプ別にリクエストを分析すると、画像がもっとも多いリソースリクエストで、中央値のページではデスクトップページのロード時に25枚の画像、モバイルでは22枚の画像をリクエストします。これは昨年のデスクトップ23枚、モバイル25枚とほぼ同じです。

JavaScriptはリクエスト数が次に多く、デスクトップで22リクエスト、モバイルで21リクエストです。これも2021年のデスクトップ21、モバイル20と非常に近い数値です。

一般的に、デスクトップとモバイルの違いはほとんどなく、モバイルで画像がわずかに少ないことがありますが、これは初期ビューポートが小さいために遅延ロードが発火しないことが原因かもしれません。


*図21.4. パーセンタイル別のページ重さの分布。*

50パーセンタイルでは、デスクトップページは2MBを超え、モバイルページはそれ未満です。90パーセンタイルでは、デスクトップは約9.0MBに、モバイルは約8.0MBに成長しています。

全体的なページ重さは、デスクトップとモバイルのユーザーエージェントで提供されるものを見ると非常に近いですが、より高いパーセンタイル（大きなページ）ではギャップがわずかに広がっています。モバイルデバイスはローカルリソースが少なく、ネットワーク機能が制限されている傾向があるため、これは懸念されます。

**678 MB**

*図21.5. 最大のデスクトップページの重さ*

100パーセンタイルでは、検出された最大のページで、デスクトップユーザーは678MBのページに直面し、モバイルユーザーは390MBでした。

これらの大きなサイズを構成するものをもう少し詳しく掘り下げてみましょう。

## リクエストバイト


*図21.6. 時間経過による中央値のページ重量。*

時間経過とともに中央値のページ重量を見ると、傾向は一貫してがっかりさせるものであり、中央値の重量は時間とともに増加しています。

**594%**

*図21.7. 10年間でのモバイルページ重量の増加*

2012年6月から2022年6月の10年間で、デスクトップページロードの中央値のページ重量は221%、または1.6 MB増加し、モバイルページロードでは594%、または1.7 MB増加しました。

年間比較（2022年6月対2021年6月）では、デスクトップは2,121 KBから2,315 KBに、モバイルは1,912 KBから2,020 KBに増加しました。

## コンテンツタイプとファイル形式

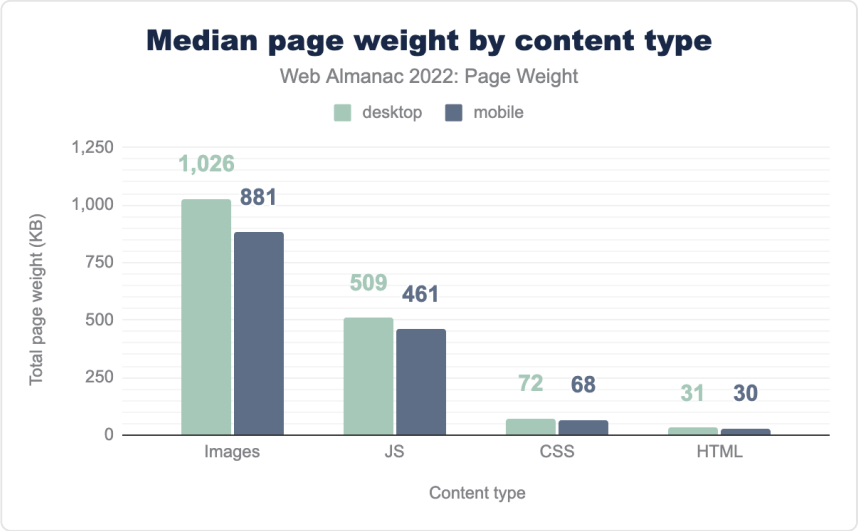



*図21.8. コンテンツタイプ別の中央値ページ重量。*

ページの重さを構成するもっとも一般的なリソースコンテンツタイプの中央値の重量を見ると、画像が最大の貢献者であり、デスクトップページでは1,026 KB、モバイルでは811 KBです。次に大きな貢献者は、デスクトップとモバイルの両方のページロードでJavaScriptです。

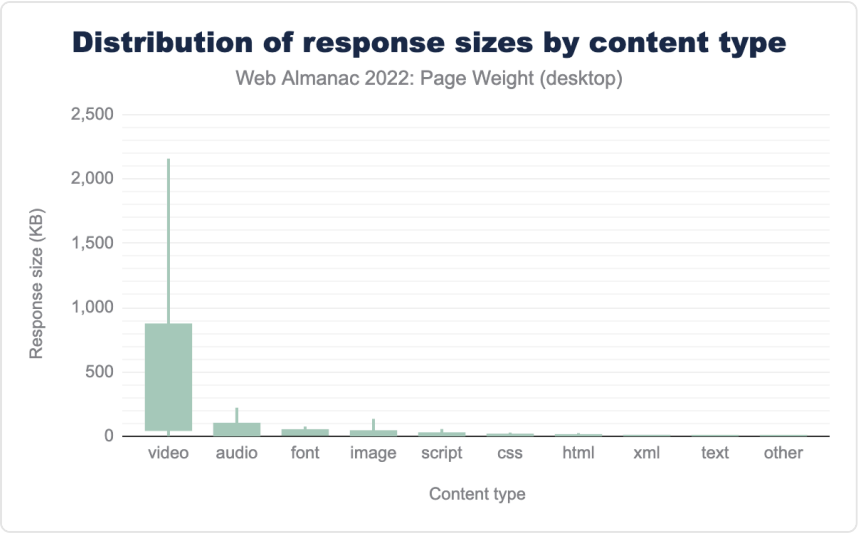



*図21.9. コンテンツタイプ別の応答サイズの分布。*

インターネット全体でページ重量にもっとも大きく貢献しているのは画像ですが、リクエストごとの純粋なサイズでもっとも大きな貢献者はビデオ、オーディオ、フォントです。90パーセンタイルで、ビデオリクエストは2,158 KBで、他のすべての90パーセンタイルタイプを合わせたものの4倍の重さがあります。

画像と同様に、より現代的なフォーマットや、より良いエンコーディング、サイズ設定、圧縮を活用することで、この重さを減らす機会があります。ただし、ビデオはその性質上重くなりがちであり、許容できる品質とサイズの間のバランスが必要です。詳細はメディア章のビデオセクションを参照してください。

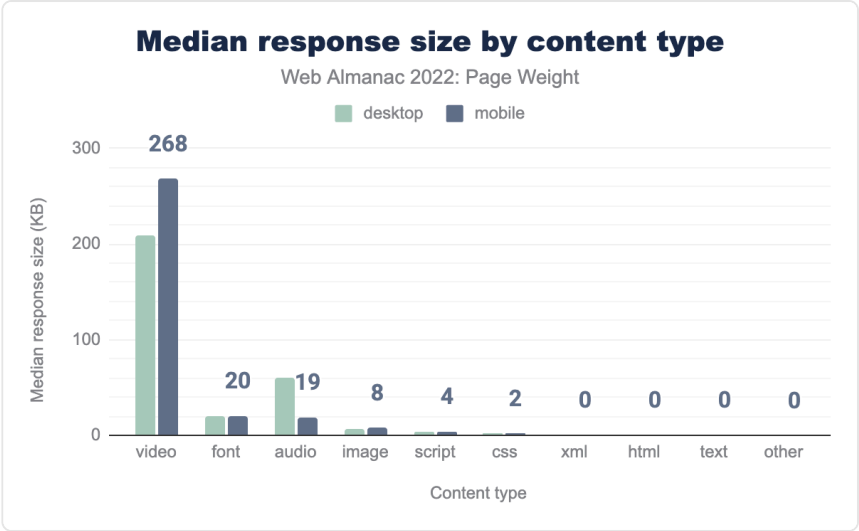



*図21.10. コンテンツタイプ別の中央値応答サイズ。*

各コンテンツタイプの中央値応答サイズを見ると、モバイルページロードでビデオコンテンツがデスクトップよりも大きい268 KBであることが意外です。デスクトップでは208 KBです。フォントの中央値重量が画像よりも高く、モバイルで20 KB対8 KBと2倍以上になっていることも驚きです。

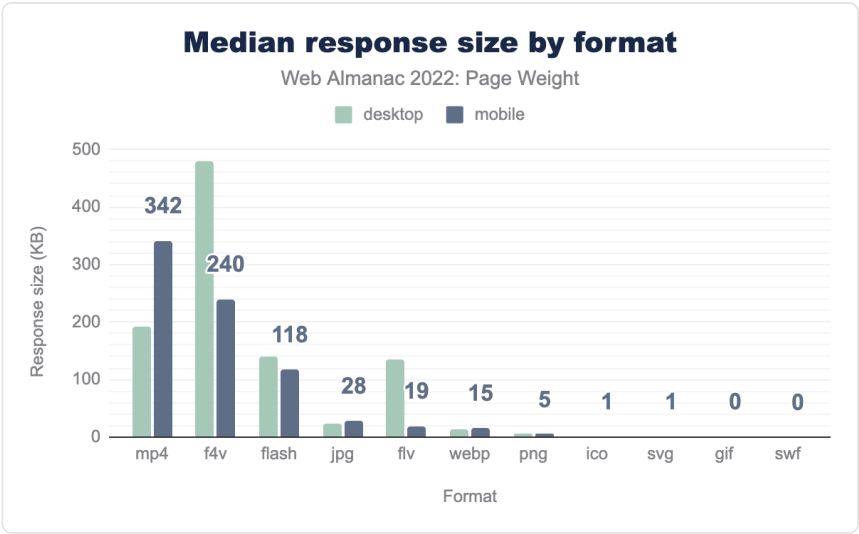



*図21.11. フォーマット別の中央値応答サイズ。*

ファイル形式に焦点を当てると、f4v、フラッシュ、flvがページにかなりの重さを加えていることが残念です。フラッシュプレイヤープラグインは2021年に廃止され、Chromeのような主要なブラウザから削除されました[812]ので、これらのバイトはおそらく完全にムダです。

### 画像バイト

Web Almanacの開始以来、画像はページ重量の最大の割合を占めていますので、どのフォーマットが使用されているかを見る価値があります。

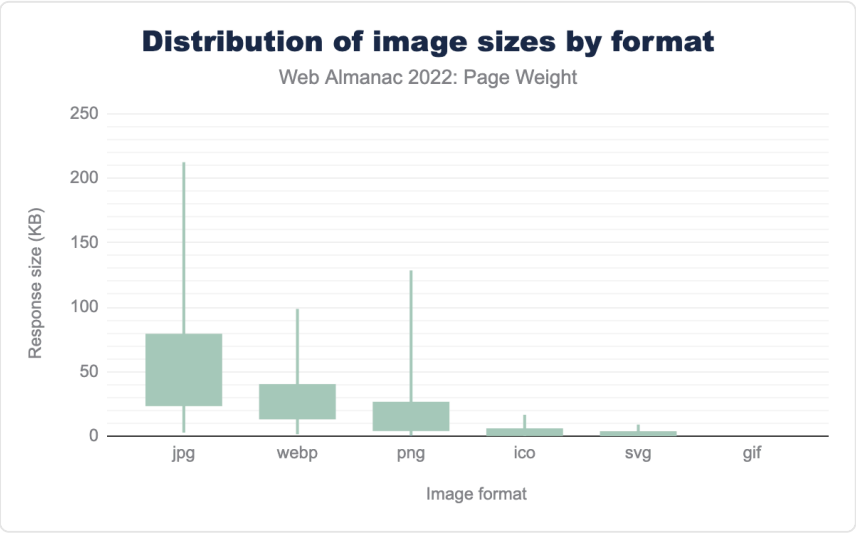



*図21.12. フォーマット別の画像サイズの分布。*

フォーマット別の画像サイズの分布は、JPG、WebP、PNGファイル形式が2021年の状態を維持し、画像重量の主要なソースであることを示しています。

812. https://support.google.com/chrome/answer/6258784


*図21.13. 画像サイズの分布。*

2022年のデスクトップの画像の中央値重量は1,026キロバイトで、2021年からわずか44キロバイトの増加でした。モバイルは881キロバイトでほとんど変わりませんでした。

年次の一貫性は、100パーセンタイルの極端な部分によってのみ崩れます。最大のデスクトップページには672MBの画像が含まれており、2021年の186MBという重い最大値から大幅に増加しました。同様に、モバイルの100パーセンタイルは959%増の385MBを見ました。

### ビデオバイト

モバイルウェブのメディア章によると、モバイルページの5%に`video`要素が含まれています。この情報は、ビデオファイルがセットにグループ化されている全体的なページ重量の他のファイルタイプの100パーセンタイルと一致しています。ビデオ体験を提供するページは、それに応じて重量が増加します。

ウェブ上のビデオの51.5%を占めるMP4は、最大の応答サイズの可能性も表しています。50パーセンタイルで、mp4の応答サイズは534キロバイトになります。

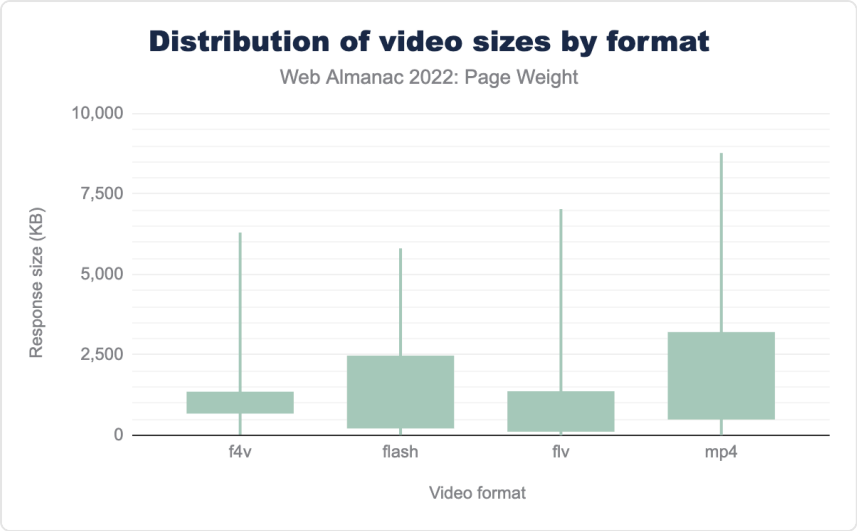



*図21.14. フォーマット別のビデオサイズの分布。*

## バイト節約技術の採用率

では、送信しているこれらのバイトについて何ができるでしょうか？明らかに、ただ送信を停止することもできますが、理由があって送信されているはずです！したがって、内容を保持しつつ、ただキロバイトをパイプに詰め込むよりも効率的な方法で送信する方法を見てみましょう。

### ビデオとその他の埋め込みのためのファサード

ビデオやその他のインタラクティブな埋め込みは、ページの全体的な重量を大幅に増加させる可能性があります。ビデオはその性質上バイト数が多いことがありますが、ソーシャルメディアの埋め込みなど他のコンテンツも、たとえばツイートを埋め込むために多くのJavaScriptやその他のデータを取り込むことがあります。

良いデザインパターンは、ファサードの使用、つまり遅延ローディングの形態です。これは基本的に要素のグラフィカル表現を表示し、必要になるまで完全なものをロードしないというものです。たとえば、YouTubeビデオの場合、初期ロードはビデオのポスター画像だけであることがあります。これは人気のあるlite-youtube-embed[813]ライブラリによって採用されており、クリック時に実際の完全なYouTube埋め込みに変わります。また、従来の画像の遅

813. *https://github.com/paulirish/lite-youtube-embed*

延ロードのように振る舞い、ビューポート内または近くに変更することもできます。

このアプローチには欠点がありますが、web.devのサードパーティファサードに関する記事[814]で詳述されているように、ワイヤー上で送信されるデータの観点からユーザーにとっての利点は明らかです。彼らはビデオを見たい、またはライブチャットアプリと対話したい場合にのみ、そのコストを支払う必要があります。

実際には、ここでの採用は追跡が難しいです。Lighthouseは限られたサードパーティリソースの使用を見て、これらがリクエストされている場合、ファサードが利用可能かもしれないと指摘します。

サイトがファサードをうまく使用している場合、これらのリソースはリクエストされず、したがってLighthouseではテストできません。

ファサードパターンはサードパーティリソースに限定される必要はなく、これらは追加のルックアップや接続のデメリットがありますが、大きな自己ホストリソースにもうまく機能する可能性があります。

分析によると、Lighthouseがファサードが有益である可能性があることを検出できたサイトがいくつかあります。

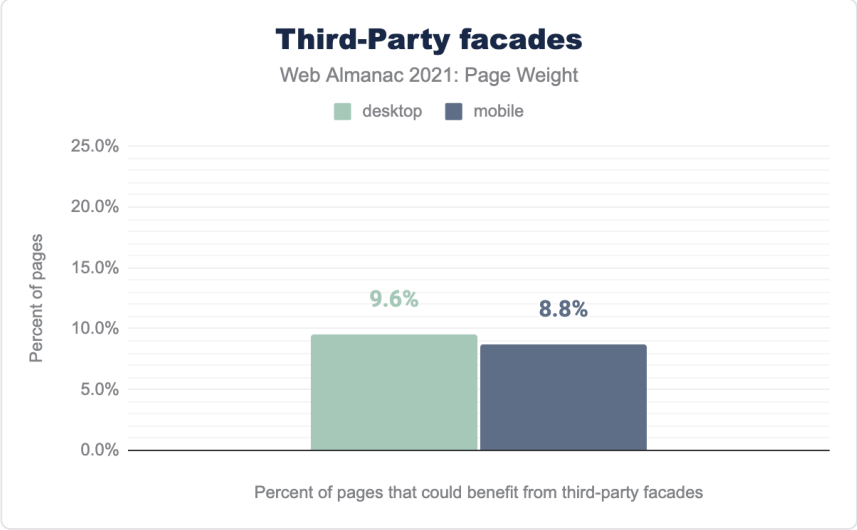



*図21.15. サードパーティのファサード。*

テストされたデスクトップページの9.6%がファサードの実装から恩恵を受ける可能性があり、モバイル訪問ではやや良好な8.8%です。

814. https://web.dev/third-party-facades/

## 圧縮

クライアントにリソースを提供する前に圧縮することでネットワークを介して送信する必要があるバイトを節約でき、理論上、そして通常は実際にも、より速いロードが可能になります。

HTML、CSS、JavaScript、JSON、SVGのようなテキスト非メディアファイルや、ttf、icoファイルに対して、HTTP圧縮は強力なツールで、gzipやBrotli圧縮を使用してファイルサイズを小さくできます。画像やビデオのようなメディアはすでに圧縮されているため、恩恵はほとんどありません。

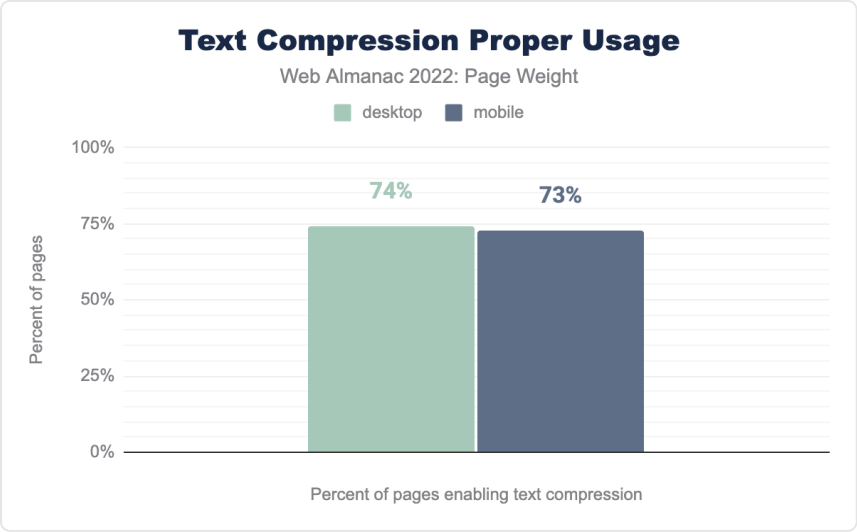



*図21.16. テキスト圧縮の適切な使用。*

デスクトップのページロードの74%、モバイルのページロードの73%で圧縮が検出されました。

モバイルの使用率がやや低いのは、モバイルが遅くて高価なネットワーク接続を持つ可能性が高いため、失望する結果です。

最終的に圧縮はページの重さの全体的な影響を減らすわけではありません。なぜなら、これらのリソースは使用前にクライアントで解凍されなければならないからです。

また、圧縮は完全に無料のプロセスではありません。静的リソースに対してキャッシュ可能であるとしても、サーバーには圧縮のための処理オーバーヘッドがあり、クライアント側にはこれらのリソースを使用する前に解凍するコストがあります。

これはトレードオフについてであり、しばしばネットワークが最悪のボトルネックとなります。圧縮技術は驚くほど効率的であり、通常、その正味の利益はそれだけの価値があります。

## 縮小化

縮小化[815]は、不要な文字（空白、コードコメントなど）を削除することで、テキストベースのリソースの全体的なサイズを減らすのに役立ちます。これらはブラウザがこれらのリソースをどのように解釈し使用するかには関係ありません。

CSSとJavaScriptは縮小化のための素晴らしい候補です。私たちはLighthouseのテストを使用して両方を調べました。

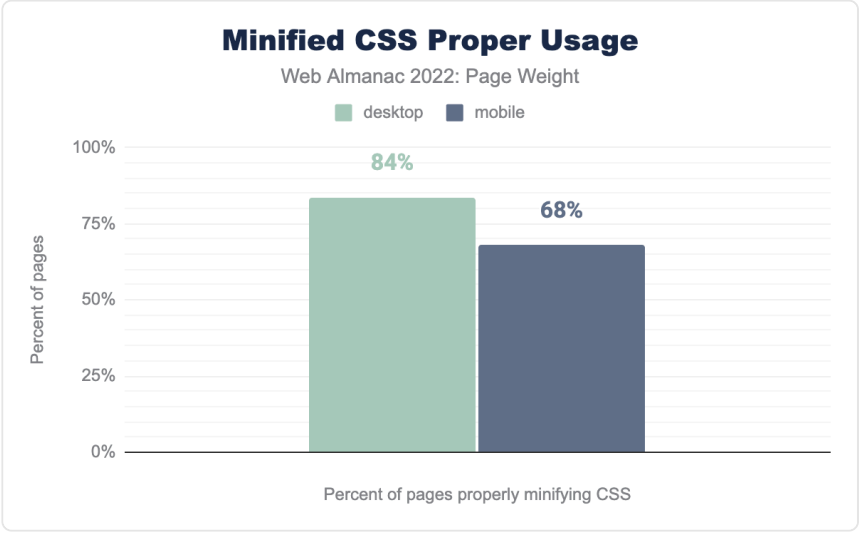



*図21.17. CSSの適切な縮小化。*

デスクトップページロードの84%が提供されたCSSを正しく縮小化し、モバイルページロードの68%がより少ない割合でした。

815. *https://en.wikipedia.org/wiki/Minification_(programming)*

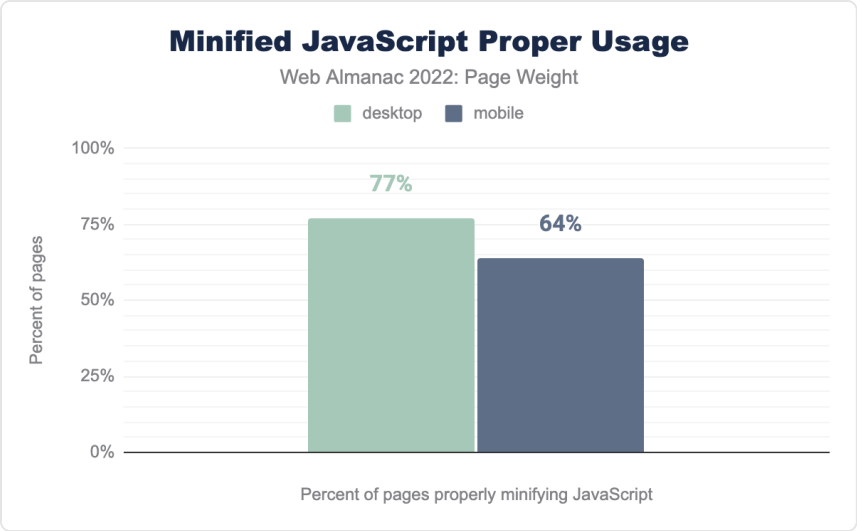



*図21.18. JavaScriptの適切な縮小化。*

デスクトップページロードの77%が提供されたJavaScriptリソースを正しく縮小化し、モバイルページロードの64%がより少ない割合でした。

CSSとJavaScriptの縮小化はありがたいことに人気があり、多くのサイトが正しく行っていますが、まだ改善の余地があります。

圧縮と同様に、モバイルユーザーのための縮小化がデスクトップよりも遅れているのは残念です。圧縮と同様に、モバイルデバイスではバイトを節約することがとくに役立ちます。

圧縮とは異なり、クライアント側に縮小化のオーバーヘッドはありません。リソースを使用するために「非縮小化」する必要はありません。サーバー側には提供時にリアルタイムで縮小化が行われる場合にオーバーヘッドが発生する可能性がありますが、CSSとJavaScriptは静的ファイルである可能性が高く、縮小化はビルド時またはリソースを公開する前に行うべきで、それ以上のオーバーヘッドはありません。

## キャッシュとCDN

キャッシュを使用すると、リソースを指定された有効期限まで再利用できます。キャッシュはブラウザやサーバーで使用されます。

CDNは人気のある例です。これら相互接続されたサーバーは地理的に分散されており、ユーザーにもっとも近いネットワーク位置からキャッシュされたコンテンツを送信することで、遅延を減らします。CDNはページの重さを減らすわけではなく、リクエスターとリソース間

の距離を短縮することで遅延を減らします。

そのため、この章ではこの点については調査していませんが、CDN章ではこれについて詳しく取り上げており、昨年のキャッシング[816]章ではそれについてさらに詳しく説明しています。

## 結論

重たいページの重さは、ユーザーの待ち時間を長くします。ウェブページの増大するコストは、データアクセスのコスト、技術要件を満たすためのデバイスのコスト、および時間のコストで支払われます。

画像とJavaScriptは依然としてバイトサイズの最大の貢献者である一方で、2022年はモバイル上でのバイト重視のビデオの増加や、画像のカウンターパートよりも中央値のフォントバイトサイズが高いといった驚くべき増加が明らかになりました。

ほとんどのウェブと比べて幸いにも極端であるとはいえ、100パーセンタイルで見られる荒唐無稽な外れ値は、新しいファイルタイプや機能がデジタル体験に導入されると、制御されていない膨張の可能性を示しています。

バイト節約技術はいくつかの圧力を軽減していますが、モバイルユーザーにとってより影響が大きいにもかかわらず、デスクトップでの採用率が高いままです。このまま放置されると、デジタルの不平等の格差はさらに拡大するでしょう。

### 著者

#### Jamie Indigo

@Jammer_Volts fellowhuman1101 https://not-a-robot.com

Jamie Indigoはロボットではありませんが、ボットの言葉を話します。技術的なSEOとしてDeepCrawl[817]で、彼らは検索エンジンがウェブをどのようにクロール、レンダリング、そしてインデックス化するかを研究しています。彼らは野生のJavaScriptを手懐けることと、レンダリング戦略の最適化が大好きです。仕事をしていないとき、Jamieはホラー映画やグラフィックノベル、ダンジョンズ＆ドラゴンズが好きです。

816. *https://almanac.httparchive.org/ja/2021/cdn*
817. *https://www.deepcrawl.com*

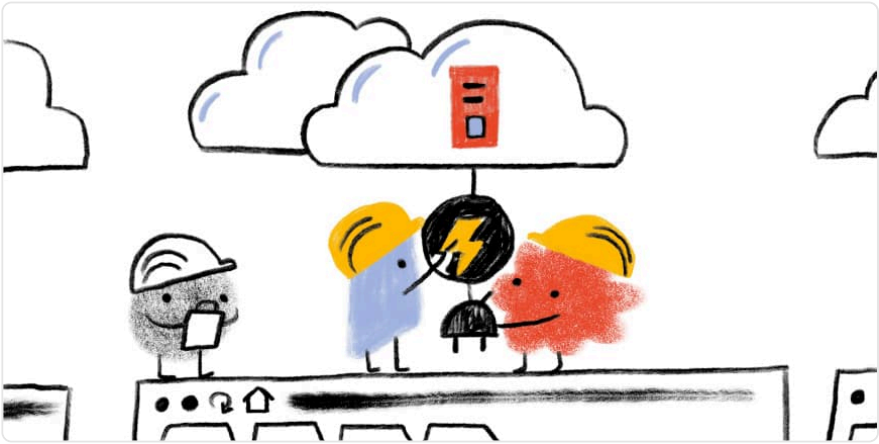

## Dave Smart

@davewsmart dwsmart https://tamethebots.com


Dave Smartは、Tame the Bots[818]で開発者および技術的な検索エンジンコンサルタントとして活動しています。彼は現代のウェブでツールを構築し、実験することが大好きで、しばしばライブギグの最前線で見かけることがあります。

*818. https://tamethebots.com*

# 部 IV 章 22
# CDN



***Haren Bhandari と Joe Viggiano によって書かれた。***
*Yutaka Oka によってレビュ。*
*Haren Bhandari と Joe Viggiano による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

この章では、CDNの現状に関する洞察を提供します。CDNは、JavaScriptライブラリ、フォント、その他のコンテンツなどの静的およびサードパーティのコンテンツの配信を容易にすることで、小規模なサイトを含め、世界中のユーザーにコンテンツを提供する上でますます重要な役割を果たしています。この章で議論するCDNの重要な側面のもう1つは、CDNがTLSやHTTPバージョンなどの新しい標準の採用において果たす役割です。

私たちは、CDNが将来においてもコンテンツの配信だけでなく、コンテンツのセキュリティにおいても重要な役割を果たし続けると考えています。ユーザーには、パフォーマンスとセキュリティの観点からCDNを見ることをオススメします。

# CDNとは何か？

*Content Delivery Network*（CDN）は、地理的に分散されたプロキシサーバーのネットワークです。CDNの目的は、ウェブコンテンツの高可用性とパフォーマンスを提供することです。これは、コンテンツをエンドユーザーに近い場所に配布することと、コンテンツを最適に配信するための先進技術をサポートすることによって達成されます。

ビデオや画像などのウェブコンテンツの爆発的な増加により、CDNは多くのウェブサイトにおいてスムーズなユーザーエクスペリエンスを提供するための重要な部分となっています。COVID-19以降、実店舗のビジネスがオンラインに移行し、ウェブ会議、オンラインゲーム、ビデオストリーミングが増加したため、CDNの必要性はさらに高まっています。

初期の頃、CDNは単なるプロキシサーバーのネットワークであり、以下の機能を果たしていました。

1. コンテンツ（HTML、画像、スタイルシート、JavaScript、ビデオなど）をキャッシュする
2. エンドユーザーがコンテンツにアクセスするためのネットワークホップを減らす
3. TCP接続の終了をウェブプロパティをホスティングしているデータセンターからオフロードする

これらは主に、ウェブ所有者がページのロード時間を改善し、これらのウェブプロパティをホスティングしているインフラからのトラフィックをオフロードするのに役立ちました。

時が経つにつれて、CDNプロバイダーによって提供されるサービスはキャッシングや帯域幅/接続のオフロードを超えて進化しました。その分散性と広範な分散ネットワーク容量により、CDNは大規模なDistributed Denial-of-Service (DDoS)[819]攻撃を非常に効率的に処理することが証明されました。近年、エッジコンピューティングは別のサービスとして人気を博しています。多くのCDNベンダーは、ウェブ所有者がエッジで単純なコードを実行できるようにするエッジでのコンピュートサービスを提供しています。

CDNベンダーによって提供されるその他のサービスには以下が含まれます。

- クラウドホスト型Web Application Firewalls (WAF)[820]
- ボット管理ソリューション
- クリーンパイプソリューション（データセンターのデータ洗浄）
- 画像およびビデオ管理ソリューション

819. *https://en.wikipedia.org/wiki/Denial-of-service_attack#Distributed_attack*
820. *https://ja.wikipedia.org/wiki/Web_Application_Firewall*

エンドユーザーに近い場所でウェブアプリケーションのロジックとワークフローを進めることは、ウェブ所有者にとって利点があります。これにより、HTTP/HTTPSリクエストが取るラウンドトリップと帯域幅が削減されます。また、起点の瞬間的なスケーラビリティ要件も処理します。これによる副作用として、インターネットサービスプロバイダー（ISP）もスケーラビリティの管理から恩恵を受けるため、そのインフラ容量が向上します。

## 注意事項と免責事項

あらゆる観察研究と同様に、測定できる範囲と影響には限界があります。Web AlmanacのCDN使用に関する統計は、使用されている適用技術により焦点を当てており、特定のCDNベンダーのパフォーマンスや効果を測定することを目的としていません。これにより、私たちはどのCDNベンダーにも偏らないことを保証しますが、これはより一般化された結果であることも意味します。

こちらが私たちのテスト方法論の限界です。

- **シミュレートされたネットワーク遅延：** トラフィックを合成的に形成する専用ネットワーク接続を使用します。
- **単一地理的位置：** テストは単一のデータセンターから実行され、多くのCDNベンダーの地理的分散をテストすることはできません。
- **キャッシュの効果：** 各CDNは独自の技術を使用し、多くの場合、セキュリティ上の理由からキャッシュのパフォーマンスを公開しません。
- **ローカリゼーションと国際化：** 地理的分布と同様に、言語と地域固有のドメインの影響もこれらのテストでは不透明です。
- **CDNの検出：** これは主にDNS解決とHTTPヘッダーを通じて行われます。ほとんどのCDNはDNS CNAMEを使用してユーザーを最適なデータセンターにマッピングします。ただし、一部のCDNはAnycast IPを使用するか、デリゲートされたドメインから直接のA+AAAA応答を使用し、DNSチェーンを隠します。その他の場合では、ウェブサイトが複数のCDNを使用してベンダー間でバランスを取ることが、私たちのクローラーの単一リクエストパスからは隠されています。

これらすべてが私たちの測定に影響を与えます。

もっとも重要なことは、これらの結果は特定の機能（たとえばTLSv1.3、HTTP/2）のサイトごとのサポートを反映していますが、実際のトラフィックの使用を反映していないことです。YouTubeは`www.example.com`よりも人気がありますが、どちらも私たちのデータセットでは同等に表示されます。

これを念頭に置いて、CDNの文脈で意図的に測定されなかったいくつかの統計をここに示します。

1. 最初のバイトまでの時間（TTFB）
2. CDNラウンドトリップタイム
3. Core Web Vitals
4. 「キャッシュヒット」対「キャッシュミス」のパフォーマンス

これらの一部はHTTPアーカイブのデータセットで測定可能であり、他はCrUXデータセットを使用して測定可能ですが、方法論の限界と一部のサイトが複数のCDNを使用しているため、測定が困難であり、誤って帰属される可能性があります。これらの理由から、この章ではこれらの統計を測定しないことにしました。

モバイルとデスクトップの間に顕著な違いは見られなかったため、繰り返しを避けるため、この章で提供されるデータはとくに注記がない限り主にモバイルの使用に関するものです。

## CDNの採用

ウェブページは以下の主要なコンポーネントで構成されています。

1. ベースHTMLページ（たとえば、`www.example.com/index.html`は、しばしば`www.example.com`のようなよりフレンドリーな名前で提供されます）。
2. 主ドメイン（`www.example.com`）およびサブドメイン（たとえば、`images.example.com`、`assets.example.com`）にある画像、CSS、フォント、JavaScriptファイルなどの組み込みファーストパーティコンテンツ。
3. サードパーティドメインから提供されるサードパーティコンテンツ（たとえば、Googleアナリティクス、広告）。

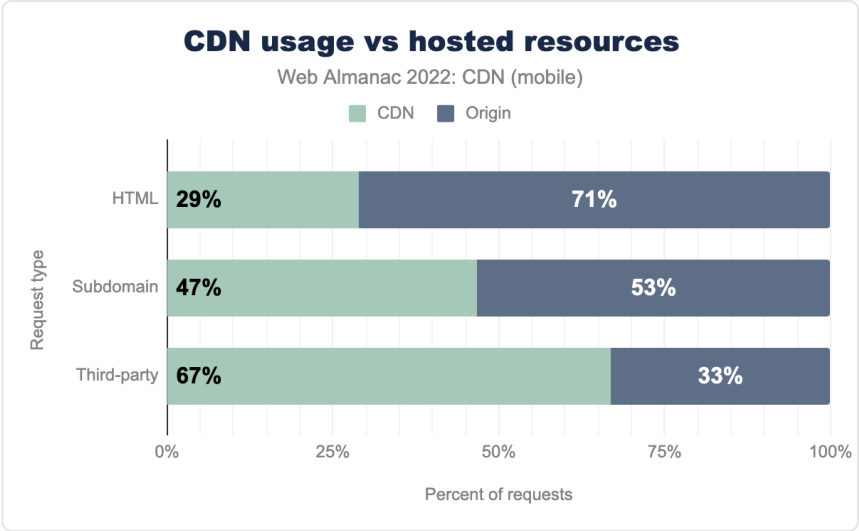



*図22.1. モバイルでのCDNの使用とホストされたリソース。*

上のグラフは、デスクトップでもほぼ同じ数字が見られたため、モバイルの各種リクエストのCDN対ホストリソースの分割を示しています。CDNはしばしば画像、スタイルシート、JavaScript、フォントなどの静的コンテンツの配信に利用されます。この種のコンテンツは頻繁に変更されないため、CDNのプロキシサーバーでキャッシュするのに適しています。とくにサードパーティコンテンツについては、この種のリソースでCDNがより頻繁に使用されています。

CDNは静的でないコンテンツの配信にもパフォーマンスを向上させることができます。CDNはルートを最適化し、もっとも効率的な輸送手段を使用することが多いですが、HTMLの配信の使用は他の2つのタイプに比べてかなり遅れています。

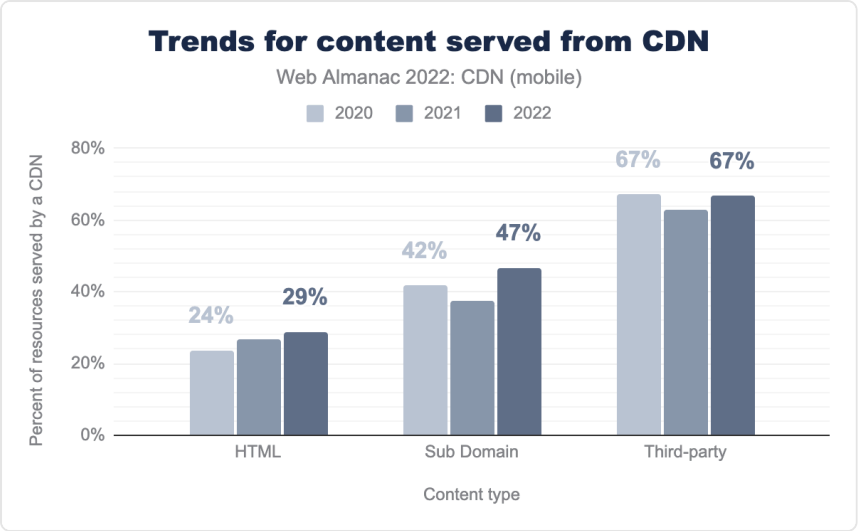



*図22.2. モバイルでのCDNから提供されるコンテンツのトレンド*

2021年[821]と比較して、CDNの使用が着実に増加していることがわかりました。サブドメインから提供されるコンテンツのCDN使用に大きな増加があり、HTMLの増加は小さく、サードパーティのCDN使用は比較的一定でした。

この増加の背後にある潜在的な理由は次のとおりです。

- パンデミック後、多くのビジネスがその大部分をオンラインに移行しました。これによりサーバーに多くの負担がかかり、静的コンテンツをCDNを通じて提供する方が、キャッシングと高速配信によってオフロードする方がはるかに効率的であることがわかりました。
- この増加は2021年には見られませんでした。多くのビジネスが問題の最適な解決策を見つけ出そうとしていたからです。実際、サブドメインとサードパーティタイプのCDN使用に減少が見られました。
- サイトは自社ドメインの代わりにサードパーティドメインを通じてサードパーティコンテンツを提供することに依存しました。この期間にサードパーティドメインから提供されるコンテンツの量が3%増加したという事実がこの仮説を支持しています。

ベースHTMLページに関しては、伝統的なパターンはベースHTMLをオリジンから提供することで、このパターンはほとんどのベースページが引き続きオリジンから提供されるため続

821. *https://almanac.httparchive.org/ja/2021/cdn*

いています。しかし、CDNから提供されるベースページが4%増加しました。ベースHTMLページがCDNから提供される傾向は明らかに上昇しています。

この背後にある可能性のある理由は次のとおりです。

- CDNは、顧客体験を向上させ、ユーザーを引き付ける上で重要なベースHTMLページのロード時間を改善できます。
- CDNプロバイダーによる分散DNSの使用は、よりシンプルで速いです。
- ベースHTMLページを含むほとんどのコンテンツをCDNを通じてプッシュする場合、災害復旧の計画が容易になります。CDNは通常、プライマリサイトが不安定または利用不可能になった場合に自動的に代替サイトに切り替えるフェイルオーバー機能を提供します。

サイトの人気に基づいて、さまざまなタイプのコンテンツでCDNの採用を観察しましたが、以下ではサイトの人気に基づいた異なる視点からこのデータを見ていきます。

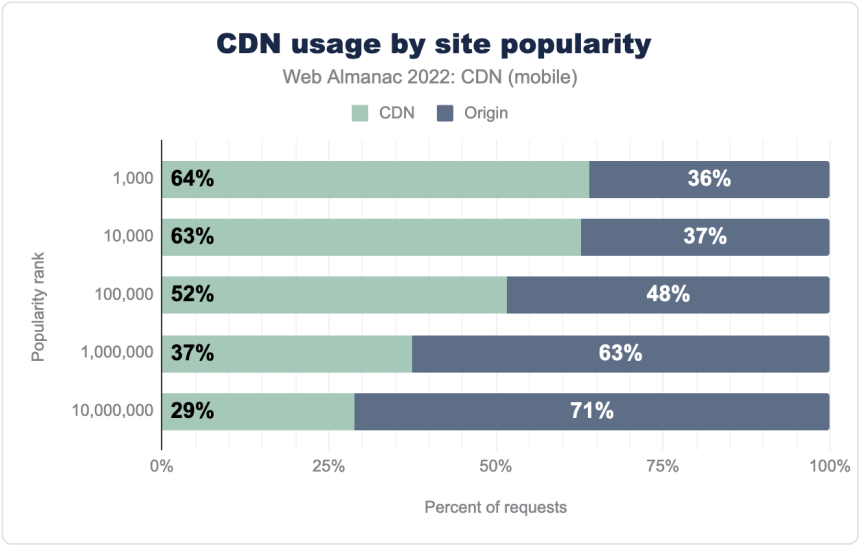



*図22.3. モバイルでのサイトの人気によるCDNの使用。*

GoogleのChrome UX Reportからのデータに基づいてウェブサイトの人気によるCDNの使用を見ると、トップ1,000-10,000がCDNのもっとも高い使用率を示しています。ランクの高いサイトでは、所有者の企業がパフォーマンスとその他の利点のためにCDNに投資していることが理解できますが、トップ1,000,000のサイトでも、CDNを通じて配信されるコンテンツ

の量が2021年と比較して[822]約7%増加しています。人気の低いサイトでのCDNの使用の増加は、CDNの無料または手頃なオプションの増加と、多くのホスティングソリューションがサービスにCDNを組み込んでいるためと考えられます。

## 主要なCDNプロバイダー

CDNプロバイダーは大きく2つのセグメントに分類できます。

1. 汎用CDN（Akamai、Cloudflare、CloudFront、Fastlyなど）
2. 特定目的用CDN（Netlify、WordPressなど）

汎用CDNは大衆市場の要求に対応しています。その提供内容には以下が含まれます。

- ウェブサイトの配信
- モバイルアプリAPIの配信
- ビデオストリーミング
- エッジコンピューティングサービス
- ウェブセキュリティの提供

これはより多くの業界に訴求し、データに反映されています。

822. *https://almanac.httparchive.org/ja/2021/cdn#fig-3*

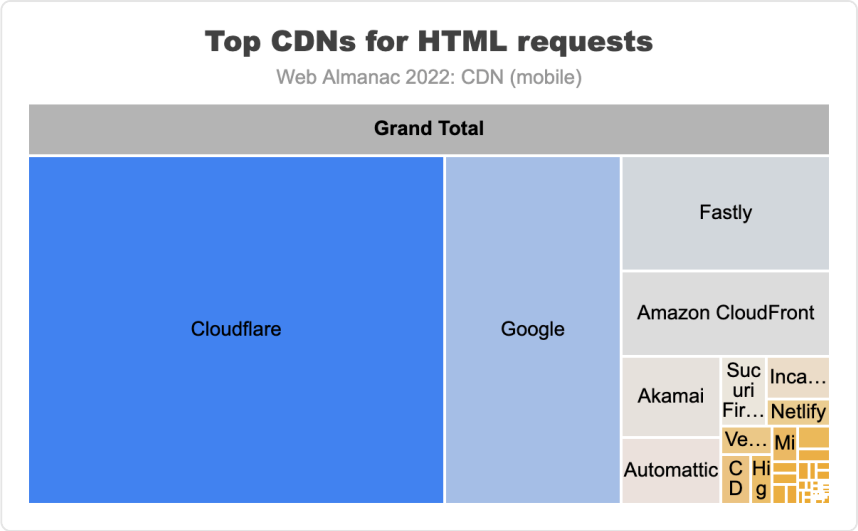



*図22.4. モバイルでのHTMLリクエストに対する主要CDN。*

上記の図は、ベースHTMLリクエストに対する主要なCDNプロバイダーを示しています。このカテゴリーでトップのベンダーはCloudflare、Google、Fastly、Amazon CloudFront、Akamai、そしてAtomatticです。

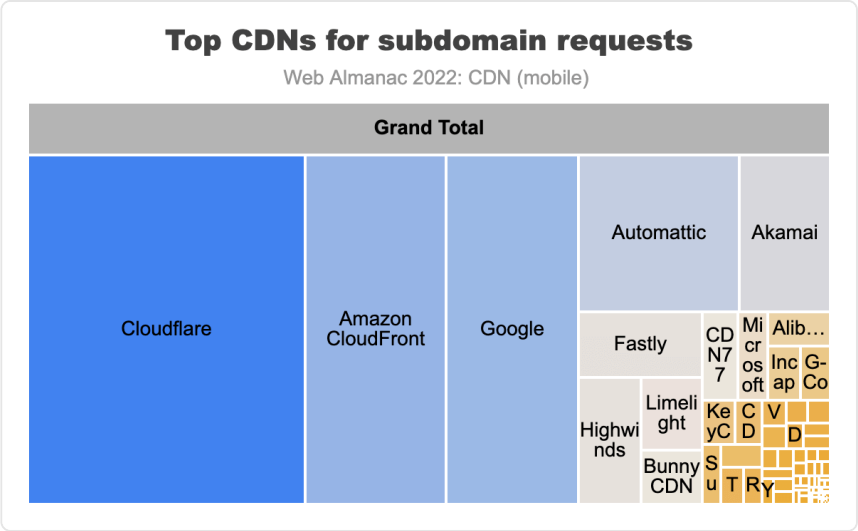



*図22.5. モバイルでのサブドメインリクエストに対する主要CDN。*

サブドメインリクエストについては、AmazonやGoogleなどのプロバイダーの使用が増えています。これは多くのユーザーが提供されるクラウドサービスでコンテンツをホストしており、クラウドサービスとともにCDNオファリングを利用しているためです。これにより、ユーザーはアプリケーションをスケールし、アプリケーションのパフォーマンスを向上させることができます。

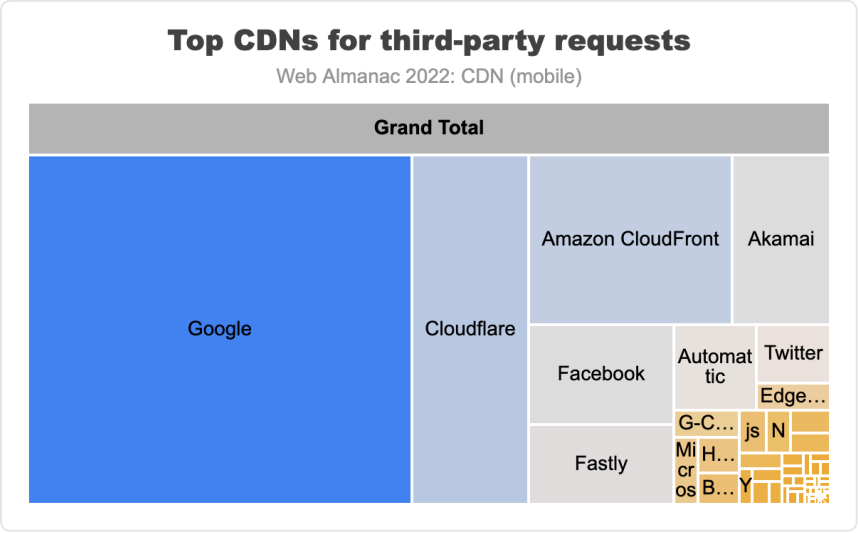



*図22.6. モバイルでのサードパーティリクエストに対する主要CDN。*

サードパーティドメインを見ると、トップのCDNプロバイダーで異なる傾向が見られます。Googleが汎用CDNプロバイダーの前にリストのトップに立っています。リストにはFacebookも目立っています。これは、多くのサードパーティドメインの所有者が他の業界よりもCDNを必要としているという事実に支えられています。GoogleやFacebookのような大手サードパーティプロバイダーにとっては、特定のコンテンツ配信ワークフローに最適化された特定目的用CDNを構築する必要があります。特定目的用CDNは、特定の市場セグメントの正確な要件に応えるために設計されています。

たとえば、広告を配信するために特別に構築されたCDNは、以下に最適化されます。

- 高い入出力（I/O）操作
- ロングテール[823]コンテンツの効果的な管理
- サービスを必要とするビジネスに地理的に近い

823. https://ja.wikipedia.org/wiki/%E3%83%AD%E3%83%B3%E3%82%B0%E3%83%86%E3%83%BC%E3%83%AB

これは、特定目的用CDNが汎用CDNソリューションとは対照的に、特定の市場セグメントの正確な要件を満たすことを意味します。汎用ソリューションはより広範な要件を満たすことができますが、特定の業界や市場に最適化されているわけではありません。

## TLSの使用

CDNがリクエストレスポンスワークフローに設定されると、エンドユーザーのTLS接続はCDNで終了します。その後、CDNは独立した2つ目のTLS接続を設定し、この接続はCDNからオリジンホストへと続きます。CDNワークフローのこの中断により、CDNはエンドユーザーのTLSパラメーターを定義できます。CDNはインターネットプロトコルの自動更新も提供する傾向があります。これにより、Webオーナーはオリジンを変更することなくこれらの利点を受け取ることができます。

### TLS採用の影響

下記のグラフは、CDNから提供されるコンテンツとオリジンから提供されるコンテンツとを比較した際、最新バージョンのTLSの採用率がCDNの方がはるかに高いことを示しています。

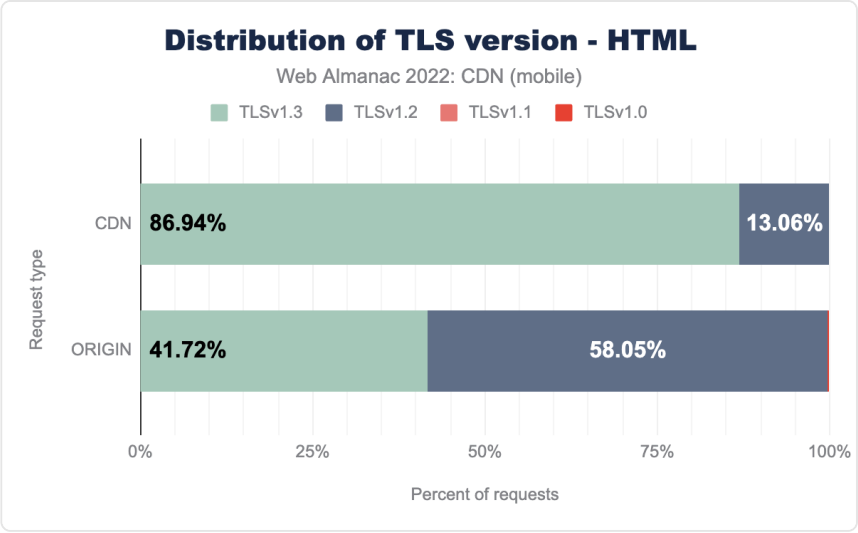



*図22.7. HTML（モバイル）のTLSバージョンの分布。*

2021年のデータ[824]と比較すると、モバイルHTMLコンテンツにおけるTLS v1.3の採用は5%増加していますが、オリジンから提供されるコンテンツの場合はTLS v1.3の採用が10%増加しています。

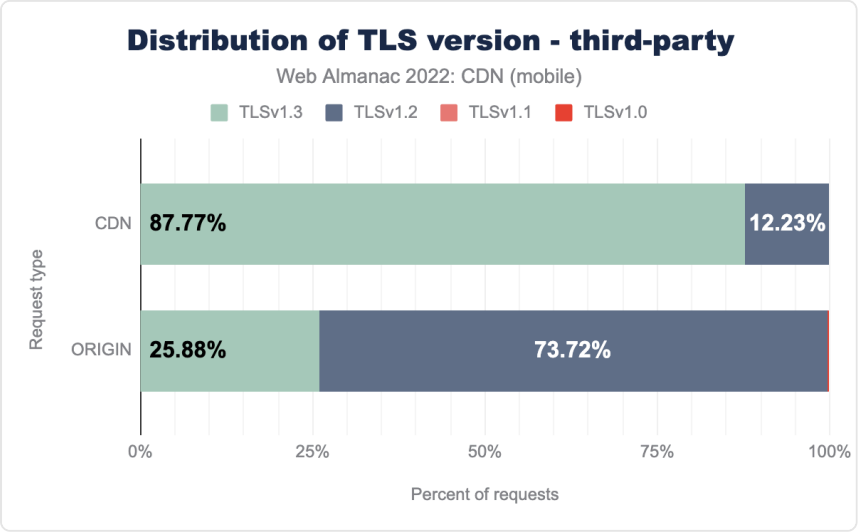



*図22.8. サードパーティリクエスト（モバイル）のTLSバージョンの分布。*

現在のセキュリティ環境では、コンテンツが最新のTLSバージョンを介して配信されることが重要です。上記のデータから、CDNを介してTLS v1.3への移行がオリジンに比べてはるかに迅速であったことがわかります。これはCDNを使用してコンテンツを配信する際の付加的なセキュリティ利点を示しています。

## TLSのパフォーマンスへの影響

一般的な論理では、HTTPSリクエストレスポンスが通過するホップの数が少ないほど、往復の時間は速くなるとされています。

824. *https://almanac.httparchive.org/ja/2021/cdn#TLS採用の影響*

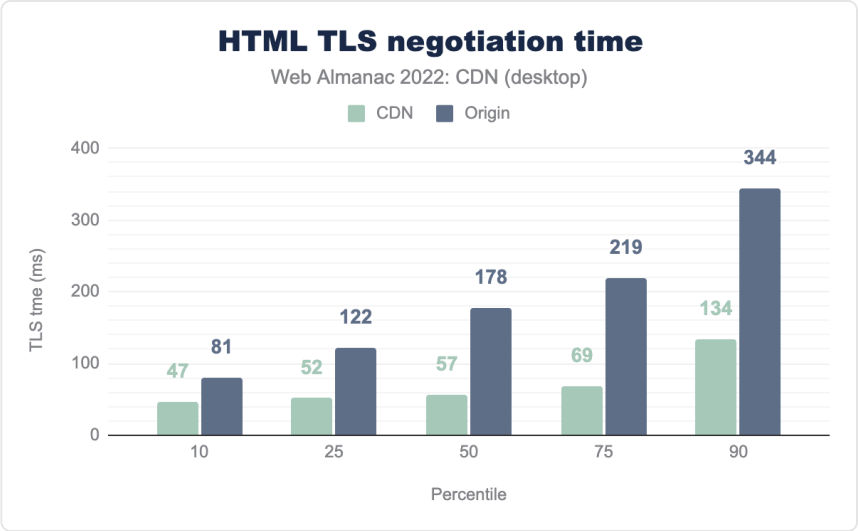



*図22.9. HTML TLSネゴシエーション - CDN対オリジン（デスクトップ）*

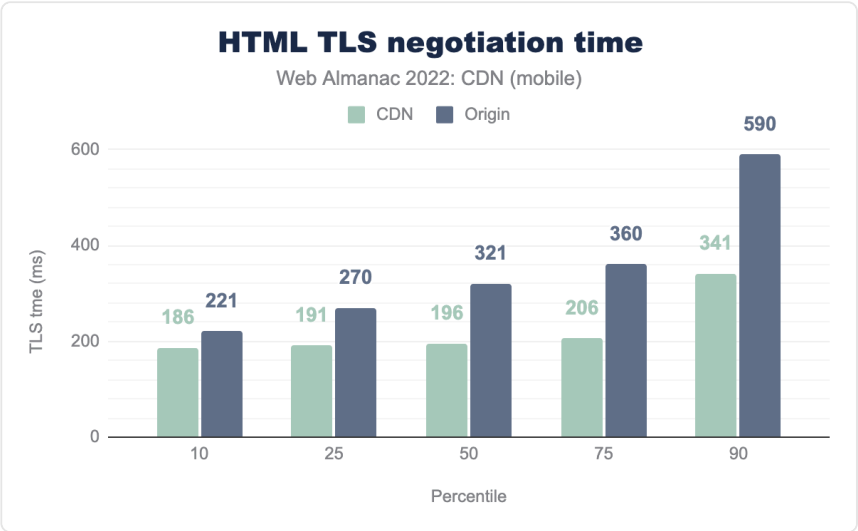



*図22.10. HTML TLSネゴシエーション - CDN対オリジン（モバイル）*

上記の図からわかるように、TLSネゴシエーション時間はCDNを利用する場合に一般にはるかに優れています。これは、デスクトップとモバイルのデータを比較した場合、さらに顕著で、*Round Trip Time* (RTT)が使用されるモバイルクローラーの結果、TLSネゴシエーション時間がはるかに長くなります。

CDNはTLS接続時間を大幅に短縮するのに役立っています。これは、エンドユーザーに近いことと、TLSネゴシエーションを最適化する新しいTLSプロトコルを採用しているためです。CDNはすべてのパーセンタイルでオリジンに対して優位性を持っています。P10およびP25では、CDNはオリジンよりもTLS設定時間で約1.5倍速いです。中央値を超えると、CDNはモバイルで約2倍、デスクトップで約3倍速くなります。90番目のパーセンタイルのユーザーは、直接オリジン接続の50番目のパーセンタイルのユーザーよりもパフォーマンスが良いでしょう。

これは、すべてのサイトが現在TLSを使用していることを考慮するととくに重要です。この層での最適なパフォーマンスは、TLS接続に続く他のステップにとって不可欠です。この点で、CDNは直接オリジン接続に比べてより多くのユーザーを低いパーセンタイルブラケットに移動させることができます。

## HTTP/2+ の採用

HTTP/2の仕様は2015年にはじめて導入され、同年の終わりまでにほとんどの主要ブラウザが採用しました。HTTPアプリケーション層プロトコルは1997年のHTTP 1.1以来更新されておらず、それ以降、Webトラフィックの傾向、コンテンツタイプ、コンテンツサイズ、ウェブサイトデザイン、プラットフォーム、モバイルアプリなどが大きく進化しました。したがって、現代のWebトラフィックの要件を満たすことができるプロトコルが必要であり、そのプロトコルはHTTP/2として実現されました。

HTTP/2のハイプと遅延の削減などの機能の約束にもかかわらず、採用は新しいアプリケーションプロトコルをサポートするためにサーバー側の更新に依存していました。CDNは、Webオーナーのために新しい実装の課題を橋渡しする手助けをすることができ、これはさらに新しいHTTP/3プロトコルにも当てはまります。HTTP接続はCDNレベルで終了し、これによりWebオーナーは自身のインフラストラクチャをアップグレードする必要なく、自分のウェブサイトやサブドメインをHTTP/2およびHTTP/3で配信する能力を持つようになります。新しいTLSバージョンの採用においても同様の利点が見られました。

CDNは、ホスト名を一元化し、ホスティングインフラストラクチャに最小限の変更を加えてトラフィックを関連するエンドポイントにルーティングするための層を提供することで、このギャップを埋めるプロキシとして機能します。キュー内のコンテンツの優先順位付けやサーバープッシュなどの機能はCDN側で管理でき、いくつかのCDNはこれらの機能をウェブサイトオーナーからの入力なしに自動的に運用するソリューションを提供しており、HTTP/2の採用を促進しています。

*HTTP/3の動作方式により（詳細はHTTP章を参照）、HTTP/3は初回接続にはほとんど使用されません。これは、多くのHTTP/2接続が実際にはリピート訪問者のためにHTTP/3である可能性があるため、"HTTP/2+"を測定しているからです（サーバーがHTTP/2なしでHTTP/3を実装することはないと仮定しています）。*

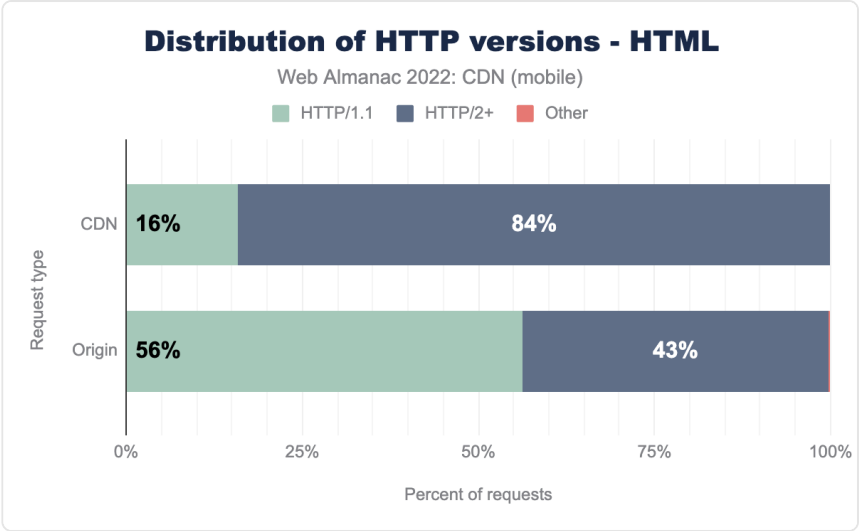



*図22.11. HTML（モバイル）のHTTPバージョンの分布。*

上のグラフには、CDNを使用しているドメインと使用していないドメインとの間に明確な対照があり、CDNを使用しているドメインのHTTP/2+の採用率が高くなっています。

2021年には、オリジンから提供されるコンテンツの約40%がHTTP/2を採用していました[825]が、同時期にCDNから提供されるコンテンツの81%がHTTP/2を通じて提供されました。オリジンのこの数値は3ポイント増加しましたが、CDNの場合は6ポイント増加し、すでに存在していたかなりの差がさらに広がりました。これは、CDNがウェブサイトオーナーが初期段階から新しいプロトコルを利用できるようにする方法を示しています。

825. *https://almanac.httparchive.org/ja/2021/cdn#HTTP/2+（HTTP/2以上）の採用*

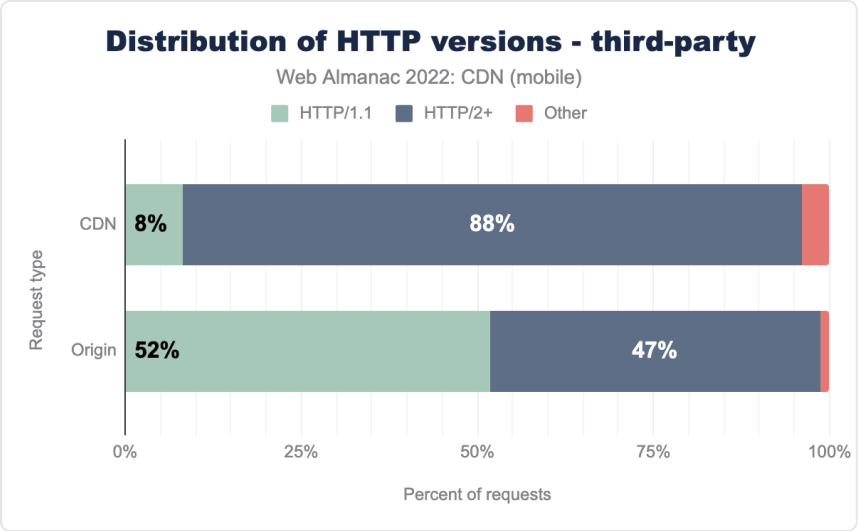



*図22.12. サードパーティリクエスト（モバイル）のHTTPバージョンの分布。*

サードパーティドメインは新しいプロトコルのサポートにさらに迅速に対応していることが以前の研究で見られました。2022年には、サードパーティドメインのHTTP/1.1のシェアがさらに減少しましたが、今年使用されたプロトコルの大部分を特定できなかったため、さらなる調査が必要です。

サードパーティドメインはすべてのネットワーク条件で一貫したパフォーマンスを必要としており、ここでHTTP/2+が価値を加えます。2022年6月には、インターネットエンジニアリングタスクフォース（IETF）がHTTP/3 RFC[826]を公開し、WebをTCPからUDPに移行しました。多くのCDNプロバイダーはHTTP/3のサポートを公式のRFC公開前に迅速に採用しており、時間が経つにつれてウェブオーナーがHTTP/3を採用することが見られるでしょう。とくにモバイルネットワークトラフィックがインターネットトラフィックの大部分を占める場合には、2023年にさらなる洞察をお届けします。

## Brotliの採用

インターネット経由で配信されるコンテンツは、ペイロードサイズを削減するために圧縮を使用します。ペイロードが小さいほど、サーバーからエンドユーザーへのコンテンツの配信が速くなります。これにより、ウェブサイトの読み込みが速くなり、エンドユーザーの体験が向上します。画像については、JPEG、WEBP、AVIFなどの画像ファイルフォーマットによって圧縮が処理されます。これについての詳細はメディア章を参照してください。

826. *https://www.theregister.com/2022/06/07/http3_rfc_9114_published/*

HTML、JavaScript、スタイルシートなどのテキストWebアセットについては、従来*Gzip*ファイルフォーマットによって圧縮が行われていました。Gzipは1992年から存在しており、テキストアセットのペイロードを小さくするのに役立っていましたが、新しいテキストアセット圧縮は、より新しい*Brotli*圧縮フォーマットを使用することでGzipよりも優れた結果をもたらすことができます。

TLSやHTTP/2の採用と同様に、BrotliはWebプラットフォーム全体で段階的に採用されてきました。この記事を書いている時点で、Brotliは世界中のWebブラウザの96%以上でサポートされています[827]。しかし、すべてのウェブサイトがテキストアセットをBrotliフォーマットで圧縮しているわけではありません。これは、サポートが不足していることと、Brotliフォーマットでテキストアセットを圧縮するのに必要な時間がGzip圧縮と比較して長くなることが理由です。また、古いプラットフォームでBrotliフォーマットをサポートしていない場合にGzip圧縮アセットを提供できるように、ホスティングインフラストラクチャには後方互換性が必要であり、これが複雑さを増す原因となります。

この影響は、CDNを使用しているウェブサイトと使用していないウェブサイトを比較するときに観察されます。

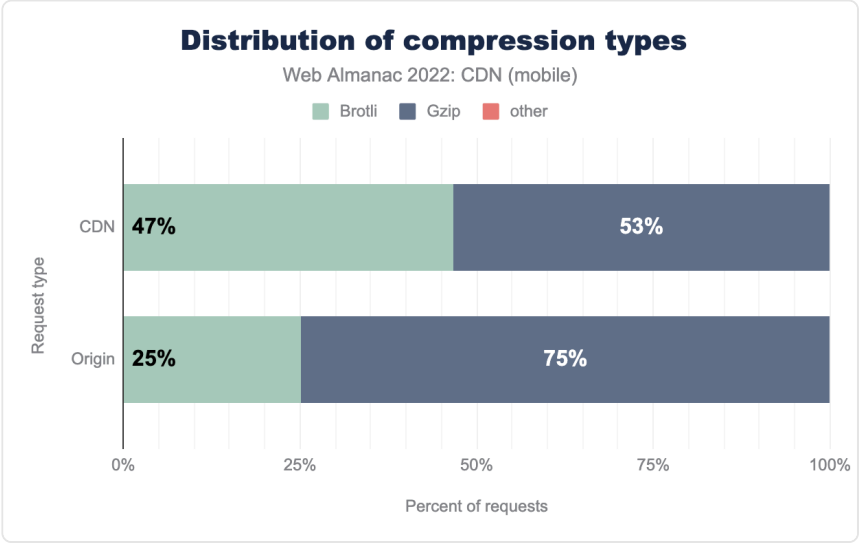



*図22.13. 圧縮タイプの分布（モバイル）。*

CDNとオリジンはともに、前年と比較して[828]Brotliの採用が増加しています。CDNでのBrotliの採用は5ポイント増加し、オリジンはほぼ4ポイント増加しました。この傾向が2023年に続くか、または飽和点に達したかどうかを確認することができるでしょう。

827. *https://caniuse.com/brotli*
828. *https://almanac.httparchive.org/ja/2021/cdn#Brotli採用*

# クライアントヒントの採用

*Client Hints* は、Webサーバーがクライアントから積極的にデータをリクエストできるようにし、HTTPヘッダーの一部として送信されます。クライアントは、デバイス、ユーザーエージェントの設定、ネットワーク情報などを提供することがあります。提供された情報に基づいて、サーバーは要求するクライアントに応答するための最適なリソースを決定できます。クライアントヒントは最初にGoogle Chromeブラウザで導入され、他のChromiumベースのブラウザもこれを採用していますが、他の人気ブラウザではクライアントヒントのサポートが限定的であったり、まったくなかったりします。

CDN、オリジンサーバー、クライアントブラウザがすべてクライアントヒントを適切に利用するためには、サポートが必要です。このフローの一部として、CDNはクライアントに `Accept-CH` HTTPヘッダーを提示して、クライアントが後続のリクエストに含めるべきクライアントヒントをリクエストできます。我々は、CDNがこのヘッダーをリクエスト内に提供したクライアントのレスポンスを測定し、テストの一環として記録されたすべてのCDNリクエストでそれを測定しました。

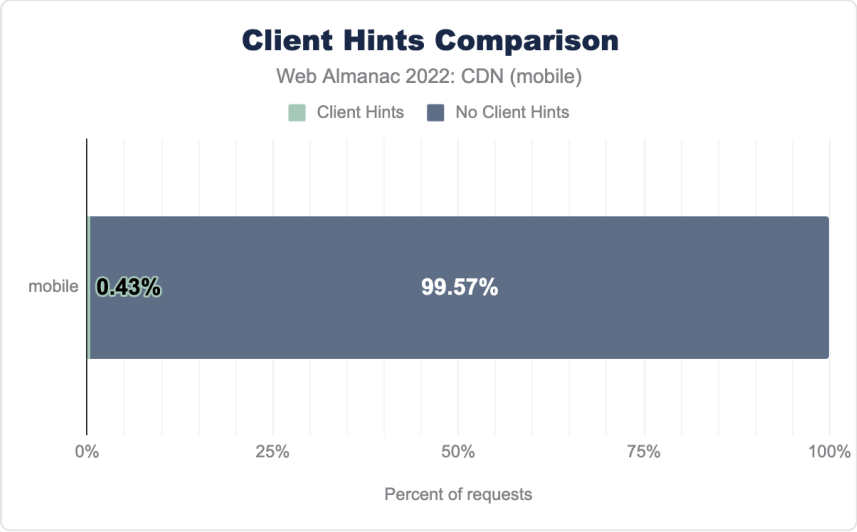



*図22.14. クライアントヒントの比較（モバイル）。*

デスクトップとモバイルの両方のクライアントで、クライアントヒントの使用率は1%未満で、クライアントヒントの採用はまだ初期段階にあることを示しています。

# 画像フォーマットの採用

CDNは伝統的に、画像などの静的コンテンツをキャッシュ、圧縮、配信するために使用されてきました。それ以来、多くのCDNは、さまざまな使用例に最適化するために、画像をフォーマットとサイズの両方で動的に変更する能力を追加しています。

これは、ユーザーエージェントやクライアントヒントに基づいて自動的に、クエリ文字列に追加パラメーターを送信することで実行される場合があります。これにより、エッジでの計算が画像を解釈し、応答に応じて画像を変更します。これによりサイト運営者は1つの高解像度画像をアップロードし、サムネイルなど、低品質または低解像度の画像が必要な場合に、画像を動的に変更できます。

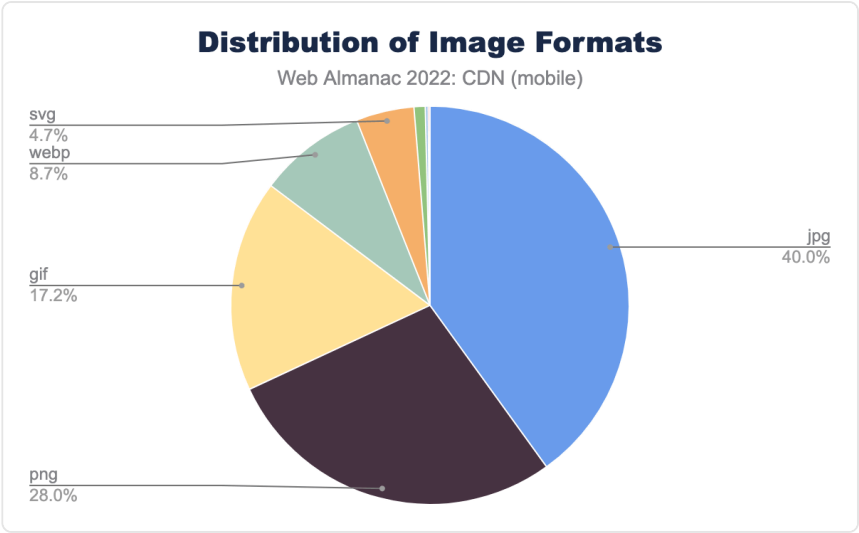



*図22.15. 画像フォーマットの分布（モバイル）。*

デスクトップとモバイルの両方で、主要な画像フォーマットはJPG（JPEG）とPNGでした。JPGはサイズを最適化するためのロスリ圧縮ファイルフォーマットを提供します。PNGまたはポータブルグラフィックスフォーマットはロスレス圧縮をサポートし、画像が圧縮されてもデータが失われることはありませんが、画像全体のサイズはJPGよりも大きくなります。JPGとPNGの詳細については、Adobeのウェブサイト[829]を訪れてください。

829. https://www.adobe.com/creativecloud/file-types/image/comparison/jpeg-vs-png.html

# 結論

過去数年にわたる我々の継続的な研究から、CDNは、ウェブサイトオーナーがオリジンから世界中のどのユーザーにもコンテンツを確実に配信するために重要な役割を果たしているだけでなく、新しいセキュリティおよびWeb標準の採用にも大きな役割を果たしていることがわかります。

一般的に、CDNの使用率は全体的に増加しています。TLS 1.3のような新しいウェブセキュリティ標準の採用においても、CDNはその採用を大いに促進しており、CDNからのトラフィックのほとんどがTLS 1.3を使用していることが確認されています。

HTTP/2やBrotli圧縮など、新しいWeb標準やWeb技術の採用に関しても、CDNがリードしていることが見られました。CDNを利用して提供されるウェブサイトの大部分がこれらの新しい標準を採用していることが確認されました。エンドユーザーの観点から見ると、これらは非常にポジティブな進展であり、彼らは安全にサイトを利用しながら、最適なユーザー体験を得ることができるでしょう。

今年からはクライアントヒントや画像フォーマットの採用など、新しい指標にも注目しています。来年のデータ収集が進むにつれ、より多くの洞察を得ることができるでしょう。

CDNについて外部から得られる洞察には限界があり、その内部で動作する「秘密のソース」を知ることは困難です。しかし、我々はドメインをクロールし、CDNを使用しているサイトとそうでないサイトを比較しました。その結果、CDNがネットワーク層からアプリケーション層に至るまで、新しいWebプロトコルの採用を促進していることがわかりました。

CDNのこの役割は非常に価値があり、今後も続くでしょう。CDNプロバイダーはインターネットエンジニアリングタスクフォース[830]の重要な一部であり、インターネットの未来を形作る手助けをしています。彼らは今後も、インターネット対応産業がスムーズかつ確実に、そして迅速に運営できるように、重要な役割を果たし続けるでしょう。

来年のデータ収集が進むことを楽しみにしており、読者の皆様に有用な洞察を提供できることを期待しています。

830. *https://www.ietf.org/*

## 著者


### Haren Bhandari

harendra

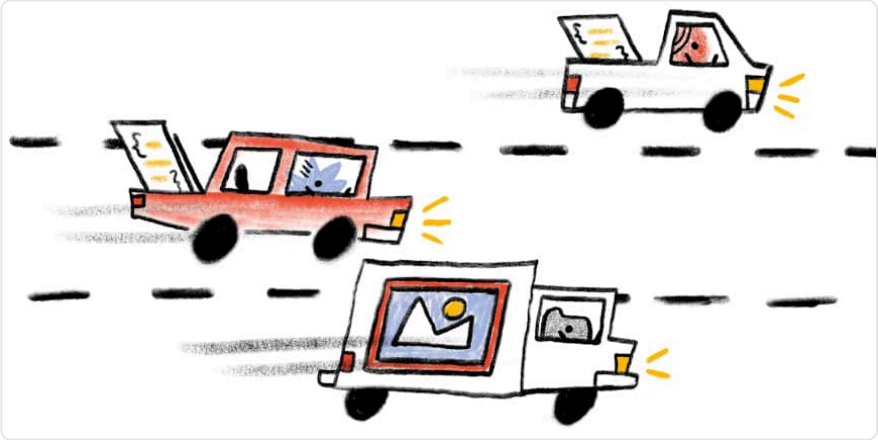
HarenはAmazon Web Servicesのソリューションアーキテクトです。Amazon Web Servicesに参加する前、HarenはAkamai Technologiesで働いており、CDNに関する深い経験を持っています。

### Joe Viggiano

joeviggiano

JoeはAmazon Web Servicesのメディア＆エンターテイメントソリューションアーキテクトで、顧客が大規模にメディアコンテンツを配信するのを支援しています。

# 部 IV 章 23
# HTTP



***Vaspol Ruamviboonsuk によって書かれた。***
*Barry Pollard、 Robin Marx と Lucas Pardue によってレビュ。*
*Vaspol Ruamviboonsuk による分析。*
*Barry Pollard 編集。*
*Sakae Kotaro によって翻訳された。*

## 序章

HTTPはウェブエコシステムの基盤であり、データ交換の基礎を提供し、さまざまなタイプのインターネットサービスを可能にします。とくに過去数年間で、HTTP/2や最近ではHTTP/3の導入により、いくつかの進化を遂げています。HTTP/2は、HTTP/1.1の同時に処理できるリクエストの数が限られているといった欠点を解決しようとしました。一見すると、HTTP/3はHTTP/2と似ておりセマンティクスは同じですが、内部的にはTCPの代わりにQUICをトランスポートプロトコルとして利用する点で、先行するバージョンと根本的に異なります。

HTTP/2がHTTP/3の基礎を提供するため、HTTP/2 Push、優先順位付け、多重化などのHTTP/2の主要機能を分析し、それらがどれだけまだ使用されているかを理解します。また、これらの機能のさまざまな展開経験からの事例研究も提示します。たとえば、HTTP/2 Pushでは、ウェブサーバーがクライアントの要求前にリソースのレスポンスを先取りして送信できます。

プッシュと優先順位付けは理論的にはエンドユーザーにとって有益ですが、使用するのが難しい場合があります。新しい技術について議論し、パフォーマンスが低いHTTP/2の機能の代替として使用できる可能性があります。たとえば、103 Early Hintsレスポンスは、先取りリソースフェッチの同じパフォーマンス目標を達成するHTTP/2サーバープッシュの代替手段を提供します。

最後に、HTTP/3がHTTP/2をどのように改善しているか、また2021年からのいくつかの増加を観察するHTTP/3の現在のサポートについて分析しながら議論します。この章で提供されるデータポイントが、将来のトレンドや新しい技術に関する洞察を提供し、開発者がユーザー体験を向上させるために実験できる新しい技術への指針となることを願っています。

## HTTPの進化

HTTPはウェブの通信を支えるもっとも重要なインターネットプロトコルの1つです。最初の3バージョン（0.9、1.0、1.1）ではテキストベースのプロトコルとして始まりました。その拡張性により、HTTP/1.1は2015年まで15年間現行のHTTPバージョンでした。

HTTP/2は、テキストベースのプロトコルからバイナリベースのものへと進化したHTTPの大きなマイルストーンでした。HTTP/1.1がシリアルなリクエストとレスポンスの交換のみをサポートするのに対し、HTTP/2は並行処理をサポートします。クライアントとサーバーは、リクエストとレスポンスをバイナリフレームのストリームとして表現します。ストリームには一意の識別子があり、フレームを多重化してインターリーブできます。

HTTPの最新バージョンはHTTP/3で、最近2022年6月にIETFによって標準化されました[831]。HTTP/3はHTTP/2と同じ機能を実装していますが、インターネット上で輸送される方法が重要な違いです。HTTP/3はQUIC上に構築されており、QUICはUDPベースのプロトコルで、HTTP/2が損失の多いネットワークで直面するパフォーマンスの問題のいくつかを軽減します。

## HTTP/2の採用

年々導入されてきたさまざまなHTTPバージョンをふまえ、HTTPバージョンの採用状況の現状から説明します。2022年の **Web Almanac** データセットのすべてのリソースを取り、それぞれのリソースがどのHTTPバージョンで提供されたかを特定することによって、HTTPバージョンの使用状況を測定します。

しかし、設定上、クライアントがHTTP/3サポートを発見する必要があり、通常は `alt-svc` HTTPレスポンスヘッダーを介して行われるため、HTTP/3で提供されたリソースを正確にカ

831. *https://www.rfc-editor.org/rfc/rfc9114.html*

ウントすることは容易ではありません。クライアントが`alt-svc`ヘッダーを受け取る時点で、すでにHTTP/1.1またはHTTP/2のプロトコル交渉を完了しています。この時点の後に、クライアントは後続のリクエストやページの読み込みでプロトコルをHTTP/3にアップグレードできます。そのため、私たちのデータは完全なHTTP/3ページの読み込みを決して捉えることができません。

`alt-svc` HTTPヘッダー機構を介してHTTP/3の発見が遅れることにより、通常のブラウジングユーザーに対してHTTP/3で提供されたリソースが過少報告される可能性があります。したがって、HTTP/2とHTTP/3で提供されたリソースをHTTP/2+としてまとめています。

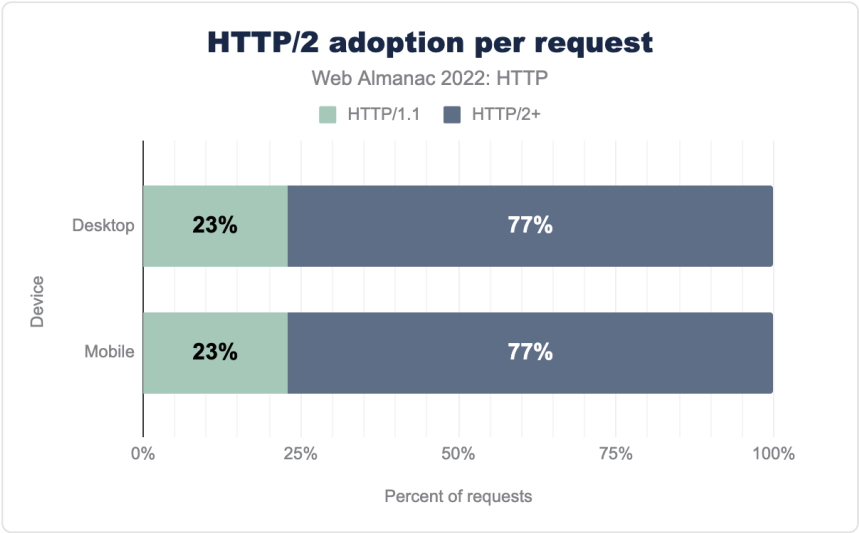



*図23.1. リクエストにおけるHTTP/2以上の採用率。*

まず、現状を理解するために、HTTP/2+の普及率を測定します。2022年6月のデータでは、私たちのリクエストの約77%がHTTP/2+を使用していることが観察されました。これは、2021年7月のHTTP/2+採用率から5%増加しています[832]。当時、リクエストの73%がHTTP/2+でした。

832. *https://almanac.httparchive.org/ja/2021/http#リクエストによる採用*

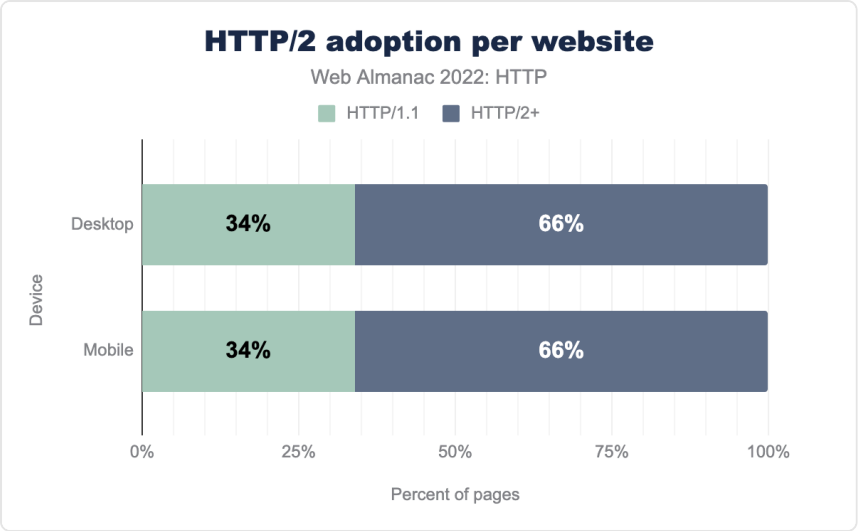



*図23.2. ウェブサイトにおけるHTTP/2以上の採用率。*

HTTP/2+の普及率の増加を理解するために、まず、ウェブサイトのランディングページがHTTP/2+で提供されているかどうかを確認することで、ウェブサイト単位でのHTTP/2+の採用を分析します。データセットの約66%のウェブサイトが、デスクトップとモバイルの設定でHTTP/2+で提供されていましたが、2021年のデータセットでは約60%のウェブサイトがそうでした[833]。この増加は、ウェブサイトが最新のHTTPバージョンへ移行していることを示すポジティブな傾向です。

833. *https://almanac.httparchive.org/ja/2021/http#HTTP/2の採用*

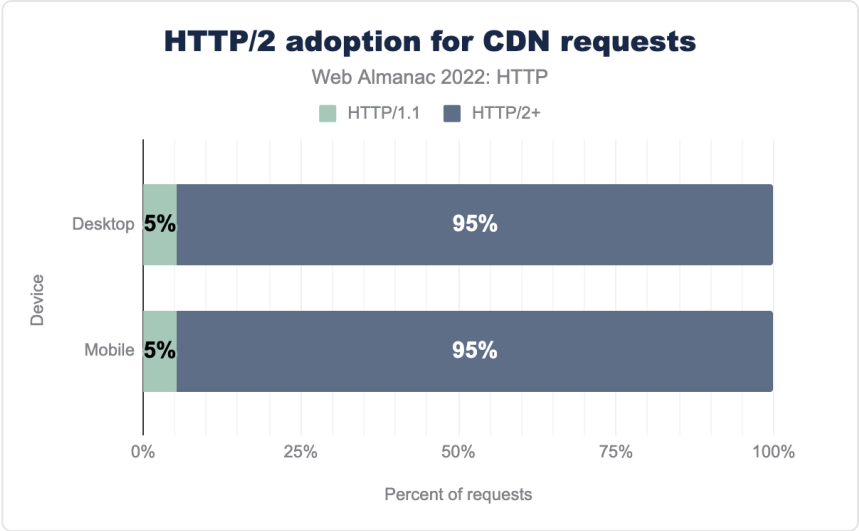



*図23.3. CDNから提供されるリクエストにおけるHTTP/2以上の採用率。*

HTTP/2+の採用を促進する別の要因はCDNから提供されるリソースです。2021年の分析[834]での観察と同様に、CDNから提供されたほとんどのリソースがHTTP/2+でした。上記の図は、CDNから提供されたリクエストの95%がHTTP/2+で提供されたことを示しています。

## HTTP/2とHTTP/3の利点

次に、HTTP/2が導入した機能がどのように採用されているかに焦点を当てます。主に、単一のネットワーク接続上での*多重化*リクエスト、リソースの*優先順位付け*、および*HTTP/2 Push*の3つの顕著な概念に焦点を当てます。

### 複数のリクエストを単一の接続で多重化

HTTP/2の重要な特徴の1つに、単一のTCP接続で複数のリクエストを多重化することがあります。これは、TCP接続上で同時に1つのリクエストしか許可されず、ほとんどの場合、ホスト名に対して6つの平行TCP接続のみが許可されるという、以前のHTTPバージョンの制限を大幅に改善します。HTTP/2では、リソース配信に使用される接続の論理表現であるストリームの概念が導入されました。それにより、複数のHTTP/2ストリームを単一のTCP接続に多重化でき、接続ごとの同時実行の制限を解除します。

834. *https://almanac.httparchive.org/ja/2021/http#CDNによる採用*

単一のTCP接続にリクエストを多重化することの含意として、ページロード中に必要な接続数の削減があります。2021年の私たちの調査結果[835]と同様に、HTTP/2が有効なページでは、HTTP/2が有効でないページよりも接続数が少ないことが確認できます。

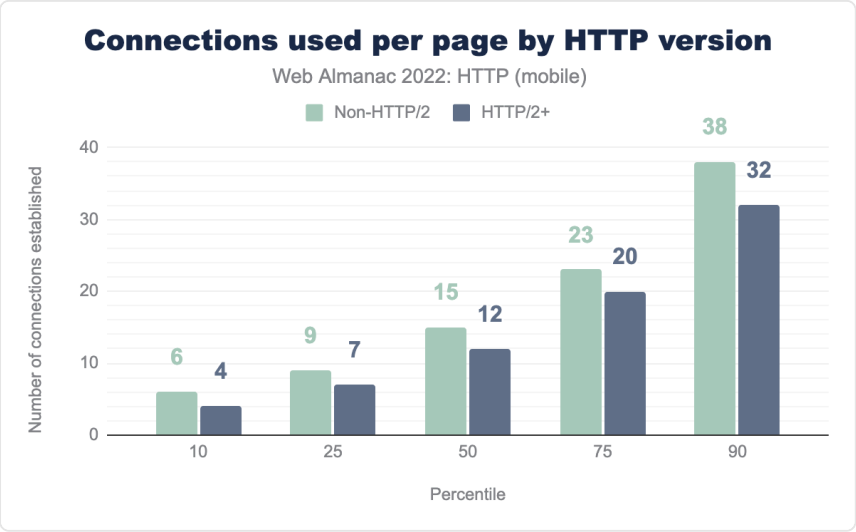



*図23.4. HTTPバージョンごとのページ毎の使用される接続数。*

上の図は、HTTP/2が有効な場合、中央値のモバイルページがページロード中に12の接続を確立していることを示しています。これに対し、HTTP/2が有効でない中央値のページでは15の接続が確立され、追加で3つの接続が生じるというオーバーヘッドがあります。しかし、より高いパーセンタイルではオーバーヘッドはさらに悪化します。HTTP/2が有効な90パーセンタイルのページは32の接続があり、HTTP/2が有効でない90パーセンタイルのページは38の接続があり、6つの追加接続のオーバーヘッドがあります。これらの傾向は、デスクトップ版とモバイル版のウェブサイト間で同様です。

年を追うごとにHTTP/2の採用が増加していることを考えると、全体的にTCP接続数が徐々に減少していることは驚くべきことではありません。HTTP Archiveからの縦断的な視点[836]によれば、確立された接続の中央値は2017年の22接続から2022年の13接続に9接続減少しました。

## リソース優先順位付け

HTTP/2では、クライアントは同じ接続上で複数のリクエストを多重化[837]できます。これによ

835. *https://almanac.httparchive.org/ja/2021/http#接続台数*
836. *https://httparchive.org/reports/state-of-the-web#tcp*
837. *https://stackoverflow.com/questions/36517829/what-does-multiplexing-mean-in-http-2/36519379#36519379*

り、多数のリソースが同時に処理中の場合、レンダリングをブロックするリソースの配信に悪影響を及ぼす可能性があり、これがユーザー体験を損なうことがあります。元の標準[838]では、HTTP/2はこの問題に対処しようと、クライアントがページロード中に構築する優先度ツリーを提案し、ウェブサーバーがより重要なレスポンスの送信を優先するために使用できます。しかし、このアプローチは使用が難しく、多くのウェブサーバーやCDNは正しく実装されていないか無視されました[839]。このため、HTTP/2の後の反復[840]では、異なる方式が使用されることが提案されました。

HTTP/2の優先順位付けの課題により、新しい優先順位付けスキームが必要とされました。HTTPのための拡張可能な優先順位付けスキーム[841]はHTTP/3とは別に開発され、2022年6月に標準化されました。このスキームでは、クライアントは`priority` HTTPヘッダーや`PRIORITY_UPDATE`フレームを通じて、2つのパラメーターで明確に優先順位を割り当てることができます。最初のパラメーターである`urgency`は、要求されたリソースの優先度をサーバーに伝えます。2つ目のパラメーターである`incremental`は、リソースがクライアントで部分的に使用できるかどうかをサーバーに伝えます（たとえば、画像の一部が到着するにつれて部分的に表示される場合）。このスキームをHTTPヘッダーと`PRIORITY_UPDATE`フレームとして定義することで、両方の形式が将来の拡張性を提供するように設計されているため、拡張可能です。執筆時点で、このスキームはSafari、Firefox、ChromeでHTTP/3に対して展開されています。

リソースの優先順位のほとんどはブラウザ自身によって決定されますが、開発者は今では新しい優先度ヒント[842]を使用して特定のリソースの優先度を調整することもできます。優先度ヒントはHTMLの`fetchpriority`属性を通じて指定できます。たとえば、ウェブサイトがヒーローイメージを優先したい場合、画像タグに`fetchpriority`を追加できます。

```
<img src="hero.png" fetchpriority="high">
```

優先度ヒントはユーザー体験を向上させるのに非常に効果的です。たとえば、Etsyは実験を行い[843]、特定の画像に優先度ヒントを含めた製品リスティングページでLargest Contentful Paint (LCP)が4%向上することを観察しました。


*図23.5. 優先度ヒントを使用しているモバイルページ。*

838. *https://www.rfc-editor.org/rfc/rfc7540#page-25*
839. *https://github.com/andydavies/http2-prioritization-issues*
840. *https://www.rfc-editor.org/rfc/rfc9113.html#section-5.3*
841. *https://httpwg.org/specs/rfc9218.html*
842. *https://web.dev/articles/fetch-priority*
843. *https://www.etsy.com/codeascraft/priority-hints-what-your-browser-doesnt-know-yet*

優先度ヒントは2022年4月末にChrome 101の一部として最近リリースされましたが、2022年8月時点で約1%のデスクトップウェブページと1.2%のモバイルウェブページがすでに優先度ヒントを採用しているため、採用率が非常に有望です。比較的容易にユーザー体験を向上させる可能性があるため、実験には良い機能かもしれません。

## HTTP/2 Push

HTTP/2 Pushにより、ウェブサーバーはクライアントからリクエストが送信される前に、あらかじめレスポンスを送信できます。たとえば、ウェブサイト提供者はページロード中に使用されるリソースを主要なHTMLと共にエンドユーザーにプッシュできます。

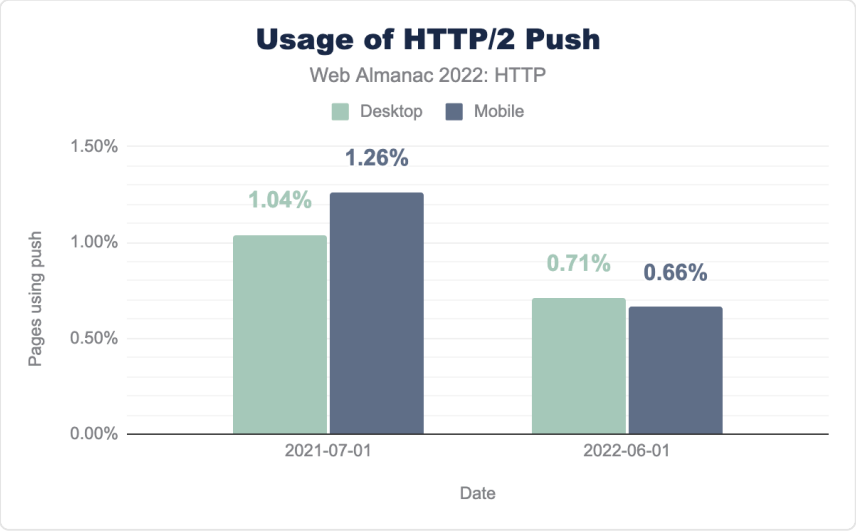



*図23.6. HTTP/2 Pushの使用状況。*

2021年の上記の図に示されているように、ウェブサイトでプッシュを使用している割合はモバイルで非常に低く、1.26%でした。しかし、今年の分析では、モバイルウェブサイトでプッシュを使用しているウェブサイトの数が0.66%に減少しています。これは2020年以降はじめてのプッシュ使用の減少を示しています。

プッシュを使用するウェブサイトの減少は、効果的に使用するのが難しい[844]ためである可能性が高いです。たとえば、ウェブサイトはプッシュされたリソースがクライアントのキャッシュにすでに存在するかどうかを正確に知ることができません。それがクライアントのキャッシュにある場合、そのリソースをプッシュするために使用される帯域幅はムダになります。

844. *https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/*

この困難さにより、ChromeはHTTP/2 Pushを非推奨[845]とすることを決定し、Chromeバージョン106[846]から始まりました。プッシュは公式にはまだHTTP/3標準の一部ですが、HTTPアーカイブクローラーが使用するChromeはHTTP3接続に対してプッシュを実装しておらず、それが使用減少の一因となり、サイトがそのバージョンに移行しプッシュ機能を失ったと考えられます。

## HTTP/2 Pushの代替手段

HTTP/2 Pushの使用に伴う課題やChromeからの非推奨化を考慮すると、開発者は代替手段について考えるかもしれません。

### プリロード

開発者は、ページロードの後半で使用されるリソースを事前にリクエストするための代替手段としてプリロードを使用できます。これは、HTMLの `<head>` セクションに `<link rel="preload">` を含めることで有効になります。たとえば。

```
<link rel="preload" href="/css/style.css" as="style">
```

または、 `Link` HTTPヘッダーとしても使用できます。

```
Link: </css/style.css>; rel="preload"; as="style"
```

どちらのオプションも、ウェブサーバーが追加のURLや重要なリソースを含めることを可能にします。クライアントは、ページロード中に通常発見される前に、提供されたURLに対してリクエストを発行できます。

リソースを積極的にプッシュするほど速くはありませんが、ブラウザがそれらのリソースをフェッチするかどうかを選択できるため、たとえばキャッシュにコピーがすでにある場合はそれをフェッチしないこともあり、より安全です。

845. https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/ho4qP49oAwAJ
846. https://developer.chrome.com/blog/removing-push/


*図23.7. `<link rel="preload">` を使用しているページ。*

私たちのデータセットには、`<link rel="preload">` タグを含むページが大量にあり、デスクトップとモバイルの両方で約25%です。

### 103 Early Hints

2017年に103 Early Hintsステータスコードが提案され[847]、Chromeは今年それをサポートしました[848]。

Early Hintsは、要求されたオブジェクトの最終レスポンスの前に暫定的なHTTPレスポンスを送信するために使用できます。ウェブサーバーがレスポンスを準備するのに時間がかかること、とくに動的にレンダリングされるメインのHTMLドキュメントの場合、この事実を利用してパフォーマンスを向上させることができます。

Early Hintsの1つの使用例は、事前にフェッチするための `Link: rel="preload"` を送信することや、他のドメインに事前に接続するための `Link: rel="preconnect"` を送信することです。他のヘッダーも概念的に伝達できますが、これはどのブラウザでもサポートされていません。

Early Hintsはプッシュよりも優れた代替手段である可能性があります。クライアントはリソースの取得方法をより大きくコントロールできますが、メインドキュメントのHTMLにプリロードやプリコネクトを追加するだけの改善を許可します。さらに、Early Hintsはプッシュでは不可能だった第三者リソースにも使用できる可能性がありますが、これもまだどのブラウザでもサポートされていません[849]。


*図23.8. 103 Early Hintsを使用しているデスクトップページ。*

Early Hintsの採用がユーザー体験を向上させることを示す研究があります。たとえば、Shopifyはラボ研究でLCPが20-30%改善されたことを観察しました[850]。私たちのデータセット

847. https://www.rfc-editor.org/rfc/rfc8297
848. https://developer.chrome.com/blog/early-hints
849. https://developer.chrome.com/blog/early-hints#current-limitations
850. https://blog.cloudflare.com/early-hints-performance/

では、初期段階にもかかわらず、約1.6%のウェブサイトがEarly Hintsを採用しており、そのほとんど（1.5%）はShopifyのプラットフォーム上で実行されているウェブサイトからのものです。

ページレスポンスに`<link rel="preload">`を含むウェブサイトが25%あるため、これらのリンクノードをEarly Hintsに変換する潜在的な可能性があります。

# HTTP/3

このセクションの残りでは、HTTPの未来であり、2022年6月に標準化された[851]HTTP/3に焦点を当てます。とくに、HTTP/3がその前身よりもどのように改善されたか、そして現在どれだけのサポートがあるかを探ります。HTTP/3についての詳細な説明については、この章のレビューにも協力してくれたRobin Marx[852]が書いた素晴らしい一連の投稿[853]を参照してください。

## HTTP/3の利点

HTTP/2はその前身に比べてさまざまな改善を導入しましたが、プロトコルにはさらなる課題と機会が残されています。たとえば、単一のTCP接続上でのリクエストの多重化は各リクエストのヘッド・オブ・ライン・ブロッキング問題を軽減しましたが、この方法でのリクエストの配信はまだ非効率的である可能性があります[854]。これは、失われたTCPパケットが、その再送信が到着するまで正しく受信された後続のTCPパケットの処理を阻止するためです。これは、後続のTCPパケットが別のHTTPストリーム用であっても、TCPはHTTPプロトコルの上層で行われている多重化を認識できないため、すべてのストリームを止めます。

HTTP/3はHTTP/2の短所を改善することを目指しており、それを実現するためにUDPベースのトランスポートプロトコルであるQUIC上に構築されています。QUICはUDP上でストリームごとの信頼性の高いパケット配信を実装することにより、TCPのヘッド・オブ・ライン・ブロッキングを解決します。

## HTTP/3のサポート

HTTP/3がサポートされていることを広告するために、ウェブサーバーはHTTPレスポンスヘッダーの`alt-svc`に依存しています。`alt-svc`ヘッダーの値には、サーバーがサポートするプロトコルのリストが含まれます。

851. *https://www.rfc-editor.org/rfc/rfc9114.html*
852. *https://twitter.com/programmingart*
853. *https://www.smashingmagazine.com/2021/08/http3-core-concepts-part1/*
854. *https://calendar.perfplanet.com/2020/head-of-line-blocking-in-quic-and-http-3-the-details/*

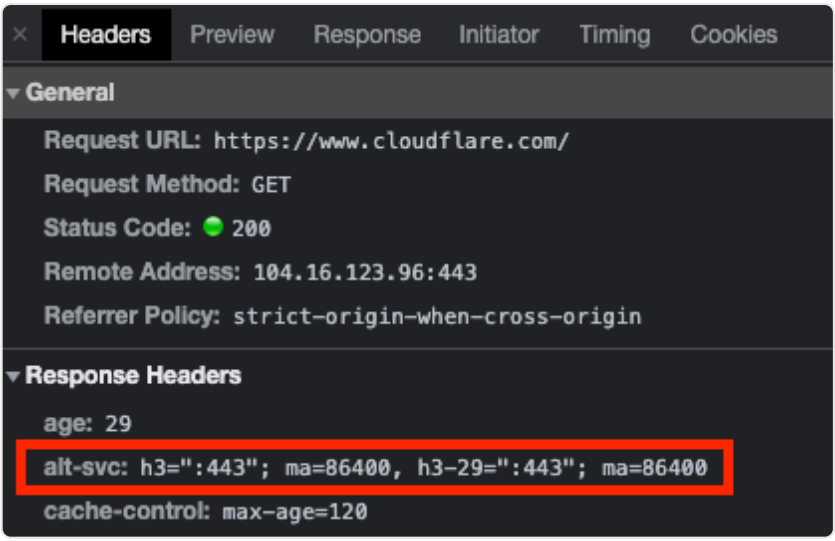



*図23.9. `alt-svc` レスポンスヘッダーの例。*

たとえば、2022年9月にhttps://www.cloudflare.comのレスポンスでの `alt-svc` 値は、以下のスクリーンショットに示されるように、 `h3=":443"; ma=86400, h3-29=":443"; ma=86400` です。 `h3` と `h3-29` は、CloudflareがUDPポート443上でHTTP/3とHTTP/3のIETFドラフト29をサポートしていることを示しています。HTTP/3の発見をDNSルックアップの一部として高速化する提案もあります。詳細についてはこのCloudflareの投稿[855]を参照してください。

HTTP/3の採用を分析するために、リソースがHTTP/3で提供されたか、そのレスポンスヘッダーに `alt-svc` ヘッダーが含まれていて、 `h3` または `h3-29` が広告されたプロトコルの1つとして含まれているかを識別します。これにより、HTTP/3が使用可能であるかを理解し、クローラーがまだ `alt-svc` ヘッダーを見ていないという上記の制限を無視します。

855. *https://blog.cloudflare.com/speeding-up-https-and-http-3-negotiation-with-dns/*

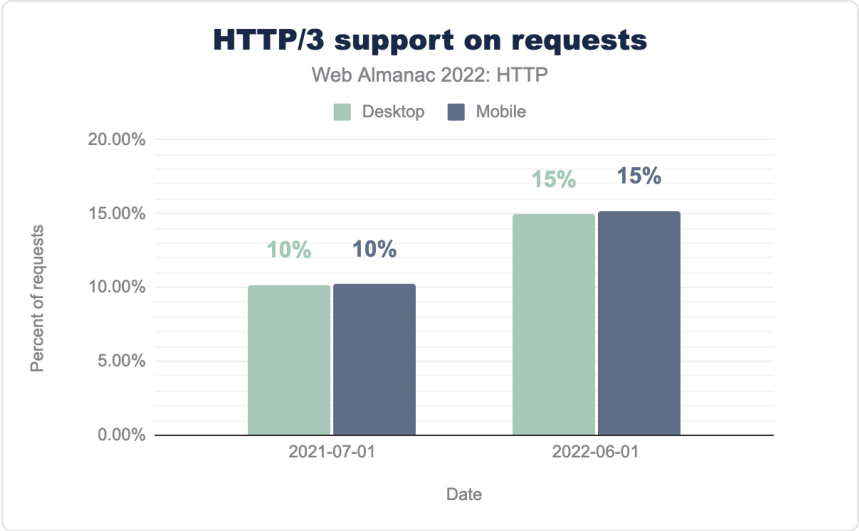



*図23.10. リクエストにおけるHTTP/3のサポート。*

上の図は、昨年のWeb Almanac以来、HTTP/3サポートがあるリクエストの割合が10%から15%に5ポイント増加したことを示しています。デスクトップとモバイルのリクエストで同様の増加が観察されました。

HTTP/2+の採用と同様に、HTTP/3のサポートのほとんどはCDNから発生しています。将来的にさらに多くのCDNがHTTP/3をサポートし始めると、サポートが増えると予想されます。

## 結論

今年はHTTPプロトコルにとって画期的な年であり、とくにHTTP/3が標準化されたことで注目されました。私たちはHTTP/2の高い利用率を継続して観察しており、ウェブサーバーからの強力な今後のHTTP/3サポートを見ています。

さらに、HTTP/2の重要な課題に対処する技術のエコシステムでの強い成長を見てきました。103 Early Hintsはサーバープッシュのより安全な代替手段を提供し、クライアントサポートはChromeが現在サポートしていることで大きく前進しました。新しいHTTP優先順位付けの標準が最終化され、主要なブラウザといくつかのウェブサーバーはすでにHTTP/3でそれをサポートしています。さらに、優先度ヒントがウェブプラットフォームに追加され、クライアントがHTTP/2およびHTTP/3の両方で優先順位をさらに洗練させることを可能にし、初期の実験は有望なユーザー体験の改善をもたらしました。

これからのHTTPプロトコルとウェブエコシステムにとって興奮する時期です。これらの新しい技術がどのように採用され、ユーザー体験にどのような影響を与えるかを見るのが楽しみです。

## 著者

### Vaspol Ruamviboonsuk

@paivaspol paivaspol vaspol-ruamviboonsuk-7898b824

Vaspol RuamviboonsukはMicrosoftのソフトウェアエンジニアです。彼はミシガン大学でウェブページの読み込み速度を向上させるシステムに関する研究を行い、博士号を取得しました。彼とLinkedIn[856]で繋がることができます。

856. *https://www.linkedin.com/in/vaspol-ruamviboonsuk-7898b824/*

# 付属資料 A
# 方法論

## 概要

Web Almanacは、HTTP Archive[857]によって組織されたプロジェクトです。HTTP Archiveは、2010年にSteve Soudersによってウェブがどのように構築されているかを追跡する使命で始まりました。月に数百万のウェブページの構成を評価し、そのテラバイトのメタデータをBigQuery[858]で分析用に提供しています。

The Web Almanacの使命は、Webの現状に関する公共知識の年間リポジトリになることです。我々の目標は、HTTP ArchiveのデータウェアハウスをWebコミュニティにとってさらに

857. *https://httparchive.org*
858. *https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data*

アクセスしやすくすることであり、専門家がコンテキストに沿った洞察を提供することでこれを実現します。

2022年版のWeb Almanacは、コンテンツ、エクスペリエンス、パブリッシング、ディストリビューションの4つの部分に分かれています。それぞれの部分では、複数の章がさまざまな角度からその全体的なテーマを探ります。たとえば、第2部では、パフォーマンス、セキュリティ、アクセシビリティの章などでユーザーエクスペリエンスを異なる視点から探求しています。

## データセットについて

HTTP Archiveのデータセットは、毎月新しいデータで継続的に更新されています。2022年版のWeb Almanacでは、とくに章で記載がない限り、すべての指標は2022年6月のクロールから取得されました。これらの結果は、BigQuery上で `2022_06_01` というプレフィックスが付いたテーブルで公開クエリが可能[859]です。

Web Almanacで提示されているすべての指標は、BigQuery上のデータセットを使用して公開再現可能です。すべての章で使用されたクエリは、GitHubリポジトリ[860]で閲覧できます。

*これらのクエリの一部は非常に大きく、自分で実行する場合、高額[861]になる可能性がありますのでご注意ください。費用を抑える方法については、Tim Kadlec氏の投稿Using BigQuery Without Breaking the Bank[862]を参照してください。*

たとえば、デスクトップおよびモバイルページごとのJavaScriptの中央値のバイト数を理解するには、bytes_2022.sql[863]を参照してください。

```
#standardSQL
# Sum of JS request bytes per page (2022)
SELECT
  percentile,
  _TABLE_SUFFIX AS client,
  APPROX_QUANTILES(bytesJs / 1024, 1000)[OFFSET(percentile *
10)] AS js_kilobytes
```

859. https://github.com/HTTPArchive/httparchive.org/blob/main/docs/gettingstarted_bigquery.md
860. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2022
861. https://cloud.google.com/bigquery/pricing
862. https://timkadlec.com/remembers/2019-12-10-using-bigquery-without-breaking-the-bank/
863. https://github.com/HTTPArchive/almanac.httparchive.org/blob/main/sql/2022/javascript/bytes_2022.sql

```
FROM
  `httparchive.summary_pages.2022_06_01_*`,
  UNNEST([10, 25, 50, 75, 90, 100]) AS percentile
GROUP BY
  percentile,
  client
ORDER BY
  client,
  percentile
```

Results for each metric are publicly viewable in chapter-specific spreadsheets, for example JavaScript results[864]. Links to the raw results and queries are available at the bottom of each chapter. Metric-specific results and queries are also linked directly from each figure.

## Websites

データセットには8,360,179のウェブサイトが含まれています。そのうち、7,905,956はモバイルウェブサイトで、5,428,235はデスクトップウェブサイトです。ほとんどのウェブサイトは、モバイルおよびデスクトップの両方のサブセットに含まれています。

HTTP Archiveは、Chrome UX ReportからウェブサイトのURLを取得しています。Chrome UX Reportは、Googleによる公開データセットで、Chromeユーザーが積極的に訪問している何百万ものウェブサイトにおけるユーザーエクスペリエンスを集約しています。これにより、最新の実際のWeb使用状況を反映したウェブサイトのリストを入手できます。Chrome UX Reportのデータセットにはフォームファクターの次元が含まれており、これを使用してデスクトップまたはモバイルユーザーがアクセスしたすべてのウェブサイトを取得しています。

Web Almanacで使用された2022年6月のHTTP Archiveクロールは、もっとも最近利用可能なChrome UX Reportリリースをウェブサイトのリストとして使用しました。202204のデータセットは2022年5月3日にリリースされ、4月にChromeユーザーが訪問したウェブサイトをキャプチャしています。

リソースの制約により、HTTP Archiveは以前、Chrome UX Reportの各ウェブサイトから1ページしかテストできず、ホームページのみが含まれていました。ホームページは必ずしもウ

864. https://docs.google.com/spreadsheets/d/1vOeFUyfEtWRen3Xj57ZsWav40n5tlcJoV0HmAxmNE_I/edit?usp=sharing

ェブサイト全体を代表しているわけではないため、結果にバイアスがかかる可能性があることに注意してください。今年、Web Almanacプロジェクトが開始された後にセカンダリページ[865]を導入し、一部の章ではこの新しいデータを使用しています。しかし、ほとんどの章ではホームページのみを使用しました。今後の分析では、この新しいデータセットがさらに活用されることを期待しています。

HTTP Archiveは、ラボテストツールとも見なされており、データセンターからウェブサイトをテストし、実際のユーザーエクスペリエンスからのデータを収集していません。すべてのページは、キャッシュが空の状態で、ログアウトした状態でテストされており、実際のユーザーがアクセスする方法を反映していない場合があります。

## メトリクス

HTTP Archiveは、Webがどのように構築されているかについて何千ものメトリクスを収集しています。これには、ページごとのバイト数、ページがHTTPSで読み込まれたかどうか、個々のリクエストおよびレスポンスヘッダーなどの基本的なメトリクスが含まれます。これらのメトリクスの大部分は、各ウェブサイトのテストランナーとして機能するWebPageTestによって提供されます。

他のテストツールは、ページに関するより高度なメトリクスを提供するために使用されます。たとえば、Lighthouseは、アクセシビリティやSEOなどの分野でページの品質を分析するための監査を実行するために使用されます。これらのツールの詳細については、下のツールセクションで説明しています。

ラボデータセットの固有の制約を回避するために、Web Almanacは、とくにウェブパフォーマンスの分野でのユーザーエクスペリエンスに関するメトリクスのためにChrome UX Reportも使用しています。

一部のメトリクスは完全に把握できません。たとえば、ウェブサイトを構築するために使用されたツールを検出する能力は必ずしも持っていません。create-react-appを使用してウェブサイトが構築されている場合、そのサイトがReactフレームワークを使用していることはわかりますが、特定のビルドツールが使用されているかどうかは必ずしもわかりません。これらのツールがウェブサイトのコードに検出可能な指紋を残さない限り、それらの使用を測定することはできません。

他のメトリクスは測定が不可能ではないにしても、挑戦的であるか、信頼性が低いことがあります。たとえば、ウェブデザインの側面は本質的に視覚的であり、ページに煩わしいモーダルダイアログがあるかどうかなど、定量化が困難です。

865. *https://discuss.httparchive.org/t/improving-the-http-archive-pipeline-and-dataset-by-10x/2372*

# ツール

Web Almanacは、次のオープンソースツールの助けを借りて実現されています。

## WebPageTest

WebPageTest[866]は、著名なウェブパフォーマンステストツールであり、HTTP Archiveの基盤です。私たちは、プライベートインスタンス[867]のWebPageTestを使用しており、各ウェブページをテストするためのプライベートテストエージェント（実際のブラウザ）を使用しています。デスクトップおよびモバイルウェブサイトは、異なる構成でテストされています。

| 設定 | デスクトップ | モバイル |
| --- | --- | --- |
| デバイス | Linux VM | エミュレートされたMoto G4 |
| ユーザーエージェント | Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.61 Safari/537.36 PTST/220609.133020 | Mozilla/5.0 (Linux; Android 8.1.0; Moto G (4)) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/102.0.5005.115 Mobile Safari/537.36 PTST/220609.133020 |
| ロケーション | Google Cloud Locations, USA | Google Cloud Locations, USA |
| 接続 | Cable (5/1 Mbps 28ms RTT) | 4G (9 Mbps 170ms RTT) |
| ビューポート | 1376 x 768px | 512 x 360px |

デスクトップウェブサイトは、Linux VM上のデスクトップChrome環境内で実行されます。ネットワーク速度はケーブル接続に相当します。

モバイルウェブサイトは、エミュレートされたMoto G4デバイス上のモバイルChrome環境内で実行され、ネットワーク速度は4G接続に相当します。

866. https://www.webpagetest.org/
867. https://docs.webpagetest.org/private-instances/

テストエージェントは、米国に拠点を置くさまざまなGoogle Cloud Platformロケーション[868]から実行されます。

HTTP ArchiveのWebPageTestのプライベートインスタンスは、最新のパブリックバージョンと同期されており、カスタムメトリクス[869]が追加されています。これらは、各ウェブサイトでテストの終了時に評価されるJavaScriptのスニペットです。

各テストの結果は、ウェブページに関するメタデータを含むJSON形式のアーカイブファイルであるHARファイル[870]として利用可能です。

## Lighthouse

Lighthouse[871]は、Googleによって開発された自動ウェブサイト品質保証ツールです。ウェブページを監査し、最適化されていない画像やアクセシビリティのないコンテンツなど、ユーザーエクスペリエンスに反するパターンが含まれていないことを確認します。

HTTP Archiveは、すべてのページで最新バージョンのLighthouseを実行しています。今年ははじめて、Lighthouseテストがモバイルとデスクトップの両方のページで行われています。2022年6月のクロール時点で、HTTP Archiveは9.6.2[872]バージョンのLighthouseを使用しました。

Lighthouseは、WebPageTest内から独自のテストとして実行されますが、独自の設定プロファイルを持っています。

| 設定 | デスクトップ | モバイル |
|---|---|---|
| CPUの速度低下 | *N/A* | 1x/4x |
| ダウンロードのスループット | 1.6 Mbps | 1.6 Mbps |
| アップロードのスループット | 0.768 Mbps | 0.768 Mbps |
| RTT | 150 ms | 150 ms |

LighthouseおよびHTTP Archiveで利用可能な監査に関する詳細は、Lighthouse開発者ドキュメント[873]を参照してください。

868. *https://cloud.google.com/compute/docs/regions-zones/#locations*
869. *https://github.com/HTTPArchive/custom-metrics*
870. *https://en.wikipedia.org/wiki/HAR_(file_format)*
871. *https://developer.chrome.com/docs/lighthouse/overview/*
872. *https://github.com/GoogleChrome/lighthouse/releases/tag/v9.6.2*
873. *https://developer.chrome.com/docs/lighthouse/overview/*

## Wappalyzer

Wappalyzer[874]は、ウェブページで使用されている技術を検出するツールです。JavaScriptフレームワーク、CMSプラットフォーム、さらには暗号通貨マイナーまで、98のカテゴリ[875]にわたる技術がテストされています。サポートされている技術は3,805を超えており（昨年の2,600から増加しています）。

HTTP Archiveは、すべてのウェブページで最新バージョンのWappalyzerを実行しています。2022年7月の時点で、Web Almanacは6.10.26バージョン[876]のWappalyzerを使用しました。

Wappalyzerは、WordPress、Bootstrap、jQueryなどの開発者ツールの人気を分析する多くの章で利用されています。たとえば、CMS章では、Wappalyzerが検出したCMS[877]カテゴリの技術に大きく依存しています。

Wappalyzerを含むすべての検出ツールには制限があります。その結果の正確性は、検出メカニズムの精度に依存します。Web Almanacは、Wappalyzerが使用されるすべての章において、特定の理由で分析が正確でない可能性がある場合、その旨を注記します。

## Chrome UX Report

Chrome UX Report[878]は、実際のChromeユーザーのエクスペリエンスに関する公開データセットです。エクスペリエンスは、ウェブサイトのオリジンごとにグループ化されます。たとえば、`https://www.example.com`です。このデータセットには、ペイント、ロード、インタラクション、レイアウト安定性などのUX指標の分布が含まれています。月ごとにグループ化されるだけでなく、国レベルの地理、フォームファクター（デスクトップ、電話、タブレット）、有効な接続タイプ（4G、3Gなど）などの次元で分割することもできます。

Chrome UX Reportのデータセットには、相対的なウェブサイトランキングデータ[879]も含まれています。これらは*ランクマグニチュード*と呼ばれ、1位や116位のような細かな順位ではなく、上位1k、上位10k、最大で上位10Mまでのランクバケットにウェブサイトがグループ化されます。各ウェブサイトは、すべてのページを合わせた対象となる[880]ページビューの数に基づいてランク付けされます。今年のWeb Almanacでは、この新しいデータを広範に使用し、サイトの人気によってWebの構築方法にどのような違いがあるかを探る手段としています。

Web Almanacのメトリクスで、Chrome UX Reportからの実際のユーザーエクスペリエンス

874. *https://www.wappalyzer.com/*
875. *https://www.wappalyzer.com/technologies*
876. *https://github.com/AliasIO/Wappalyzer/releases/tag/v6.10.26*
877. *https://www.wappalyzer.com/categories/cms*
878. *https://developer.chrome.com/docs/crux/*
879. *https://developer.chrome.com/blog/crux-rank-magnitude/*
880. *https://developer.chrome.com/docs/crux/methodology/#eligibility*

データを参照しているものには、2022年6月のデータセット（202206）が使用されています。

データセットの詳細については、Using the Chrome UX Report on BigQuery[881]ガイドをweb.dev[882]でご覧ください。

## Blink Features

Blink Features[883]は、特定のWebプラットフォーム機能が使用されていると検出された際にChromeがフラグを立てる指標です。

私たちは、Blink Featuresを使用して機能の採用状況を別の視点から捉えています。このデータは、ページに実装されている機能と実際に使用されている機能を区別するのにとくに有用です。たとえば、CSS章のGridレイアウトに関するセクションでは、Blink Featuresデータを使用して、実際のページレイアウトの一部がGridで構築されているかどうかを測定しています。これに対して、多くのページが未使用のGridスタイルをスタイルシートに含んでいることもあります。どちらの統計もそれぞれに興味深く、Webがどのように構築されているかについて何かを教えてくれます。

Blink Featuresは、WebPageTestの通常のテストの一環として報告されています。

## Third Party Web

Third Party Web[884]は、HTTP ArchiveおよびLighthouseデータを使用して、Web上のサードパーティリソースの影響を特定し分析するPatrick Hulce（2019年のサードパーティ章の著者）による研究プロジェクトです。

ドメインは、少なくとも50の異なるページに表示される場合、サードパーティプロバイダーと見なされます。このプロジェクトでは、プロバイダーを広告、分析、ソーシャルなどのカテゴリごとにグループ化しています。

Web Almanacのいくつかの章では、このデータセットのドメインとカテゴリを使用して、サードパーティの影響を理解しています。

## Rework CSS

Rework CSS[885]は、JavaScriptベースのCSSパーサーです。スタイルシート全体を取り込み、各

881. *https://web.dev/chrome-ux-report-bigquery*
882. *https://web.dev/*
883. *https://chromium.googlesource.com/chromium/src/+/HEAD/docs/use_counter_wiki.md*
884. *https://www.thirdpartyweb.today/*
885. *https://github.com/reworkcss/css*

スタイルルール、セレクター、ディレクティブ、値を区別するJSON形式のオブジェクトを生成します。

この特別なツールは、CSS章の多くの指標の精度を大幅に向上させました。各ページの全外部スタイルシートおよびインラインスタイルブロックのCSSが解析され、クエリされ、分析が可能になりました。BigQuery上のHTTP Archiveデータセットとの統合方法については、このスレッド[886]を参照してください。

### Rework Utils

今年のCSS章では、2020年のCSS章で導入された多くの指標を再訪します。その章はLea Verouが主導しました。Leaは、Rework CSSの出力から洞察をより簡単に抽出するためにRework Utils[887]を書きました。CSS章で見るほとんどの統計は、これらのスクリプトによって引き続き提供されています。

### Parsel

Parsel[888]は、CSSセレクターパーサーおよび特異性計算機で、元々は2020年のCSS章のリーダーであるLea Verouによって書かれ、別のライブラリとしてオープンソース化されました。これは、セレクターや特異性に関連するすべてのCSS指標で広く使用されています。

## 分析プロセス

Web Almanacは、ウェブコミュニティの100人以上の寄稿者との調整を伴い、計画および実行に約1年かかりました。このセクションでは、Web Almanacで選ばれた章がどのような理由で選ばれ、その指標がどのようにクエリされ、どのように解釈されたかを説明します。

### 計画

2022年のWeb Almanacは、2022年3月に寄稿者募集の呼びかけ[889]でスタートしました。プロジェクトは前年の全26章から始まり、コミュニティから提案された追加のトピックが今年の新しい2章となりました：相互運用性と持続可能性。

私たちは創刊年の方法論で述べた通り：

> *Web Almanacの将来の版の1つの明確な目標は、著者やピアレビュアー*

886. https://discuss.httparchive.org/t/analyzing-stylesheets-with-a-js-based-parser/1683
887. https://github.com/LeaVerou/rework-utils
888. https://projects.verou.me/parsel/
889. https://twitter.com/HTTPArchive/status/1508506002383069192

*として代表されていない多様な声をさらに奨励することです。*

そのために、今年も著者選定プロセス[890]を続けています：

- 以前の著者には、新しい視点を提供するため、再び執筆することをとくに奨励しませんでした。
- 2022年の著者を推薦する全員に対して、外見や考え方が同じ人々を推薦しないようにとくに意識するよう求めました。
- プロジェクトリーダーはすべての著者の推薦をレビューし、新しい視点をもたらし、コミュニティ内の代表されていないグループの声を増幅する著者を選出する努力をしました。

私たちは、Web Almanacがさらに多様で包括的なプロジェクトとなるよう、このプロセスを将来的に反復し、すべての背景を持つ寄稿者からの意見を取り入れることを望んでいます。

## 分析

2022年の4月と5月に、データアナリストは著者およびピアレビュアーと協力して、各章でクエリを実行する必要がある指標のリストを作成しました。場合によっては、分析能力のギャップを埋めるためにカスタムメトリクス[891]が作成されました。

2022年6月を通じて、HTTP Archiveのデータパイプラインは数百万のウェブサイトをクロールし、Web Almanacで使用されるメタデータを収集しました。これらの結果は後処理され、BigQuery[892]に保存されました。

4年目となる今年、以前のアナリストによって書かれたクエリを更新して再利用することができました。それでも、ゼロから書かれる必要がある多くの新しい指標がありました。GitHub上のオープンソースクエリリポジトリ[893]で、年度および章ごとのすべてのクエリを閲覧できます。

## 解釈

著者はアナリストと協力して結果を正しく解釈し、適切な結論を導き出しました。著者がそれぞれの章を書く際には、これらの統計を利用してウェブの状態を枠組みとして支えました。ピアレビュアーは著者と協力して分析の技術的正確さを保証しました。

結果を読者により容易に理解してもらうために、ウェブ開発者およびアナリストは章に埋め

890. *https://github.com/HTTPArchive/almanac.httparchive.org/discussions/2165*
891. *https://github.com/HTTPArchive/custom-metrics*
892. *https://console.cloud.google.com/bigquery?p=httparchive&d=almanac&page=dataset*
893. *https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2022*

込むデータビジュアリゼーションを作成しました。いくつかのビジュアライゼーションは、ポイントをより明確にするために簡略化されています。たとえば、完全な分布を示すのではなく、いくつかのパーセンタイルのみが表示されます。とくに記載がない限り、すべての分布はパーセンタイルを使用して要約され、とくに中央値（50パーセンタイル）が使用され、平均は使用されません。

最後に、編集者が章を見直して単純な文法的な誤りを修正し、読書体験全体の一貫性を保証しました。

## 将来に向けて

2022年版のWeb Almanacは、ウェブコミュニティの内省と前向きな変化への取り組みとして、継続することを希望する年次の伝統の第4回目です。この地点に到達するまで、多くの献身的な寄稿者のおかげで壮大な努力がなされました。私たちは、将来の版をさらに効率的にするために、可能な限りこの作業を活用することを望んでいます。

2023年版のWeb Almanacへの寄与に興味がある方は、関心フォーム[894]に記入してください。一緒にウェブの状態を追跡しましょう！

894. *https://forms.gle/zmk6wXfDrmkkKzXo8*

# 付属資料 B
# 貢献者






Web Almanacは、ウェブ・コミュニティの努力によって実現しています。2022 Web Almanac々は、企画、調査、執筆、制作の段階で116人が数え切れないほどの時間をボランティアで費やしてきました。

### Aaron Gustafson
@AaronGustafson
aarongustafson
https://www.aaron-gustafson.com
レビュワー

### Abel Mathew
@DesignrKnight
DesignrKnight
http://designrknight.com/
編集者

### Adriana Jara
@tropicadri
tropicadri
レビュワー

### Akshay Ranganath
@rakshay
akshay-ranganath
akshayranganath
https://akshayranganath.github.io/
分析者 と 著作者

### Alex Denning
@AlexDenning
alexdenning
https://getellipsis.com/
レビュワー

### Alex N. Jose
@4x13
alexnj
alexnj
https://alexnj.com
レビュワー

### Allen ONeill
@DataBytesAI
DataBytzAI
allenoneill
https://webdataworks.io/
著作者

### Alon Kochba
@alonkochba
alonkochba
alonkochba
レビュワー

### Andrea Volpini
@cyberandy
cyberandy
https://wordlift.io/blog/en/entity/andrea-volpini
著作者

### Arik Smith
4upz
分析者

### Barry Pollard
@tunetheweb
tunetheweb
tunetheweb
https://www.tunetheweb.com
分析者、 開発者、 編集者、 リーダー と レビュワー

### Belem Zhang

@ibelem
ibelem
翻訳者

### Ben Smith

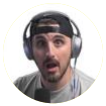
binji
https://binji.github.io
レビュワー

### Beth Pan

@beth_panx
beth-panx
分析者 と レビュワー

### Bram Stein

@bram_stein
bramstein
http://www.bramstein.com/
分析者 と 著作者



### Brian Clark

@_clarkio
clarkio
https://www.clarkio.com
著作者

### Brian Kardell

@briankardell
bkardell
https://bkardell.com
著作者 と レビュワー

### Caleb Queern

@httpsecheaders
cqueern
レビュワー

### Cameron Casher

camcash17
分析者

### Carlos Castro

@mxcarloscastro
carloscastromx
https://carloscm.me/en/
翻訳者

### Chris Adams

mrchrisadams
https://chrisadams.me.uk
レビュワー

### Chris Lilley

@svgeesus
svgeesus
https://svgees.us
レビュワー

### Chris Steele

CSteele-gh
レビュワー

### Christian Liebel

@christianliebel
christianliebel
https://christianliebel.com
レビュワー

### Cindy Krum

@suzzicks
Suzzicks
https://mobilemoxie.com/
著作者

### Colin Eberhardt

@ColinEberhardt
ColinEberhardt
https://blog.scottlogic.com/ceberhardt/
著作者

### Colt Sliva

@signorcolt
csliva
分析者

### Dan Knauss

dknauss
https://newlocalmedia.com
編集者 と レビュワー

### Danielle Rohe

@d4ni_s
drohe
https://www.digital-danielle.com
分析者

### Dave Smart

@davewsmart
dwsmart
https://tamethebots.com
著作者 と レビュワー

### David Fox

@theobto
foxdavidj
https://www.lookzook.com
リーダー と レビュワー

### Derek Perkins

derekperkins
http://derekperkins.com
分析者

### Diego Gonzalez

@diekus
diekus
https://diek.us
著作者

### Edmond de Tournadre

Djohn12
レビュワー

### Eric A. Meyer

@meyerweb
meyerweb
http://meyerweb.com/
レビュワー

### Eric Portis

@etportis
eeeps
https://ericportis.com
分析者 と 著作者

### Estelle Weyl

@estellevw
estelle
http://standardista.com
レビュワー

### Eugenia Zigisova

@jevgeniazi
imeugenia
https://github.com/imeugenia/speaking/blob/main/README.md
著作者

### Fershad Irani

@fershad
fershad
https://www.fershad.com
分析者

### Gerry McGovern

gerrymcgovernireland
著作者

### Gertjan Franken

GJFR
分析者

### Giulia Laco

@webmatter_it
webmatter-it
https://www.webmatter.it/
翻訳者

### Haren Bhandari

harendra
分析者 と 著作者

### Hemanth HM

@gnumanth
hemanth
http://h3manth.com
レビュワー

### Houssein Djirdeh

@hdjirdeh
housseindjirdeh
https://houssein.me
レビュワー

### Ingvar Stepanyan

@RReverser
RReverser
https://rreverser.com/
レビュワー

### Iskander Sanchez-Rola

iskander-sanchez-rola
https://iskander-sanchez-rola.com/
レビュワー

### Itamar Blauer

@ItamarBlauer
itamarblauer
https://www.itamarblauer.com/
著作者

### JR Oakes

@jroakes
jroakes
分析者

### Jamie Indigo

@Jammer_Volts
fellowhuman1101
https://not-a-robot.com
著作者

### Jamie Macdonald

JamieWhitMac
分析者

### Jasmine Drudge-Willson

JasmineDW
編集者

### Jens Oliver Meiert

@j9t
j9t
https://meiert.com/en/
著作者 と レビュワー

### Jeremy Wagner

@malchata
malchata
https://jlwagner.net/
著作者

### Joe Viggiano

joeviggiano
分析者 と 著作者

### John Murch

@johnmurch
johnmurch
http://www.johnmurch.com
レビュワー

### Jonathan Wold

@sirjonathan
sirjonathan
https://jonathanwold.com
著作者

### Jono Alderson

@jonoalderson
jonoalderson
https://www.jonoalderson.com
レビュワー

### José Solé

@jmsoleb
jmsole
https://www.jmsole.cl/
レビュワー

### Kai Hollberg

@schweinepriestr
Schweinepriester
レビュワー

### Kanmi Obasa

@kanmiobasa
konfirmed
https://www.knfrmd.com
分析者 と レビュワー

### Kevin Farrugia

@imkevdev
kevinfarrugia
https://imkev.dev
分析者 と レビュワー

### Kirsty Simmonds

@keinegurke_
dereknahman
https://kirsty.codes
編集者

### Kushal Das

@kushaldas
kushaldas
https://kushaldas.in
レビュワー

### Laurent Devernay

@ldevernay
ldevernay
https://ldevernay.github.io/
著作者

### Laurie Voss

seldo
http://seldo.com
分析者 と 著作者

### Liran Tal

@liran_tal
lirantal
https://twitter.com/liran_tal
著作者

### Lucas Pardue

@SimmerVigor
LPardue
https://lucaspardue.com
レビュワー

### Max Ostapenko

@themax_o
max-ostapenko
https://maxostapenko.com
分析者

### Maxim Salnikov

@webmaxru
webmaxru
https://medium.com/@webmaxru
レビュワー

### Melissa Ada

@mel_melificent
mel-ada
mel-ada
著作者

### Michael Lewittes

MichaelLewittes
編集者

### Michael Solati

@MichaelSolati
MichaelSolati
https://michaelsolati.com
著作者

### Michelle O'Connor

デザイナー

### Minko Gechev

@mgechev
mgechev
https://blog.mgechev.com/
レビュワー

### Mobeen Ali

@mobeenali97
mobeenali97
https://siffar.com
レビュワー

### Mordy Oberstein

mordy-oberstein
著作者

### Nicolas Hoizey

@nhoizey
nhoizey
https://nicolas-hoizey.com/
レビュワー

### Nishu Goel

@TheNishuGoel
NishuGoel
https://unravelweb.dev/
分析者 と レビュワー

### Nurullah Demir

@nrllah
nrllh
https://www.internet-sicherheit.de/team/demir-nurullah.html
著作者

### Pankaj Parkar

@pankajparkar
pankajparkar
https://pankajparkar.dev
レビュワー

### Patrick Meenan

@patmeenan
pmeenan
https://www.webpagetest.org/
レビュワー

### Patrick Stox

@patrickstox
patrickstox
https://patrickstox.com
レビュワー

### Paul Calvano

@paulcalvano
paulcalvano
https://paulcalvano.com
リーダー

### Philip Jägenstedt

foolip
https://foolip.org/
レビュワー

### Pilar Mera

@DecreceFeliz
decrecementofeliz
http://pi-comunicacion.com
翻訳者

### Prathamesh Rasam

25prathamesh
prathameshrasam
分析者 と レビュワー

### Rachel Andrew

@rachelandrew
rachelandrew
https://rachelandrew.co.uk
著作者

### Rick Viscomi

@rick_viscomi
rviscomi
https://rviscomi.dev/
分析者、 著作者、 編集者 と リーダー

### Rob Teitelman

@teitelmanrob
SeoRobt
https://www.paulteitelman.com/
レビュワー

### Robin Marx

@programmingart
rmarx
http://internetonmars.org
レビュワー

### Roel Nieskens

@PixelAmbacht
RoelN
http://pixelambacht.nl
レビュワー

### Sakae Kotaro

@beltway7
ksakae1216
https://ksakae1216.com/
翻訳者

### Salma Alam-Naylor

@whitep4nth3r
whitep4nth3r
https://whitep4nth3r.com/
著作者

### Saptak Sengupta

@Saptak013
SaptakS
https://saptaks.website
著作者

### Scott Davis

@scottdavis99
scottdavis99
http://thirstyhead.com
著作者

### Shaina Hantsis

shantsis
デザイナー、 編集者 と レビュワー

### Sia Karamalegos

@TheGreenGreek
siakaramalegos
karamalegos
https://sia.codes
分析者 と リーダー

### Simon Pieters

@zcorpan
zcorpan
レビュワー

### Siwin Lo

siwinlo
編集者

### Sophie Brannon

@SophieBrannon
SophieBrannon
著作者

### Thibaud Colas

@thibaud_colas
thibaudcolas
https://thib.me/
分析者 と 著作者

### Thomas Steiner

@tomayac
tomayac
https://blog.tomayac.com/
レビュワー

### Tim Frick

@timfrick
timfrick
https://www.mightybytes.com/
著作者

### Tom Van Goethem

@tomvangoethem
tomvangoethem
https://tom.vg/
著作者

### Tushar Pol

TusharPol
レビュワー

### Vaspol Ruamviboonsuk

@paivaspol
paivaspol
vaspol-ruamviboonsuk-7898b824
分析者 と 著作者

### Victor Le Pochat

@VictorLePochat
VictorLeP
victor-le-pochat
https://lepoch.at
分析者

### Vik Vanderlinden

vikvanderlinden
分析者

### William Sandres

@hakacode
HakaCode
https://hakacode.github.io
翻訳者

### Xavier Jouvenot

@10xLearner
Xav83
xavier-jouvenot-98787794
https://10xlearner.com/
翻訳者

### Yana Dimova

ydimova
分析者

### Yoav Weiss

@yoavweiss
yoavweiss
https://blog.yoav.ws
レビュワー

### Yutaka Oka

ytkoka
レビュワー

### Zhiwei Li

Levix
翻訳者

### Zongchao Bai

luckybai
翻訳者