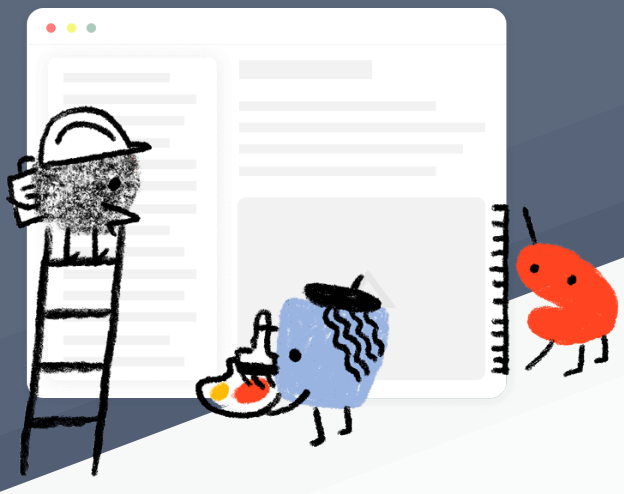
2020
Web Almanac

HTTP Archiveの年次報告書
ウェブの状態レポート



目次

導入

序章 iii

部 I. ページコンテンツ

章 1: CSS 1
章 2: JavaScript 83
章 3: マークアップ 113
章 4: フォント 143
章 5: メディア 165
章 6: サードパーティ 191

部 II. ユーザー体験

章 7: SEO 207
章 8: アクセシビリティ 251
章 9: パフォーマンス 279
章 10: プライバシー 305
章 11: セキュリティ 317
章 12: モバイルウェブ 357
章 13: 能力 381
章 14: PWA 401

部 III. コンテンツ公開

章 15: CMS 419
章 16: Eコマース 441
章 17: Jamstack 469

部 IV. コンテンツ配信

章 18: Page Weight 483
章 19: 圧縮 493

章 20: キャッシング 505
章 21: リソースのヒント 545
章 22: HTTP/2 563

付属資料

方法論 589
貢献者 601

序章

2020年は、多くの人にとって忘れたい年となりました。私たちのようにグローバル化したコミュニティが、COVID-19のパンデミックや人種差別に対する抗議活動のような広範囲に及ぶ出来事の影響を受けるのは稀なことです。多くの人々が肉体的にも精神的にも疲弊している中で、誰もが貢献したいと思い、そのための時間とエネルギーを持てるとは思えないからです。私たちは、コミュニティの関心がまだあることを願いながら、慎重に作業を進めました。

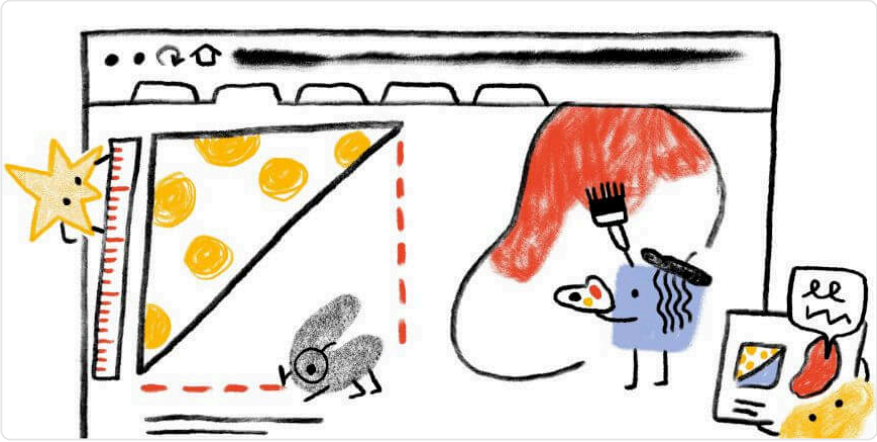
今回のWeb Almanacの目的は、2020年を忘れることではなく、追悼することです。良くも悪くも、これは私たちの歴史の一章です。今年は外部からのプレッシャーが大きかったにもかかわらず、ウェブコミュニティから100人以上のコントリビューターがサインアップし、2020年とウェブの状況を記憶することに特化したプロジェクトのために、数え切れないほどの時間をボランティアで提供してくれました。驚くべきことに、今年の版では、3つの新しい章を追加し、1つの章を失うだけで、実際に範囲を拡大することができました。

プロジェクトで最も楽しいことは何かとコントリビューターに尋ねると、ほとんどの場合、「人」という答えが返ってきます。私たちはチームとして協力し、お互いに支え合いながら、わずか5ヵ月で600ページの本に相当するものを作ることができました。世界の問題を解決したわけではありませんが、人々が協力し合うことで何が可能になるかを示したのです。

私たちのウェブへの愛の労働の集大成である2020年版Web Almanacをどうぞお楽しみください。また、チームに参加したいと思われる方は、reach outをぜひご利用ください。

— *Rick Viscomi, Web Almanacの編集長*

部 I 章 1
CSS



Lea Verou、 Chris Lilley と Rachel Andrew によって書かれた。
Estelle Weyl、 Elika Etemad aka fantasai、 Jens Oliver Meiert、 Miriam Suzanne、 Catalin Rosu と Andy Bell によってレビュ。
Rick Viscomi、 Lea Verou と Pokidov N. Dmitry による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

CSS（Cascading Stylesheets）とは、ウェブページやその他のメディアをレイアウト、フォーマット、ペイントするために使用される言語です。Webサイトを構築するための3つの主要言語の1つで、他の2つは構造を指定するために使用されるHTMLと、動作を指定するために使用されるJavaScriptです。

昨年のWeb Almanac[1]では、2019年のテクノロジーの状態を評価するために、HTTP Archiveのコーパス上で41個のSQLクエリを通して測定されたさまざまなCSSのメトリクス[2]を調べました。今年は与えられたCSS機能を使用しているページの数だけでなく、それをどのように使用しているかを測定するために、さらに深く掘り下げてみました。

1. *https://almanac.httparchive.org/ja/2019/*
2. *https://almanac.httparchive.org/ja/2019/css*

全体的に見て私たちが観察したのは、CSSの採用に関しては、ウェブは2つの異なる歯車の中にあるということでした。私たちのブログやTwitterでは、最新のものを話題にしがちですが、10年以上前のコードを使用しているサイトはまだ何百万もあります。過ぎ去った時代のベンダープレフィックス、独自のIEフィルタ、レイアウト用のフロートのようなものは、それらのすべてのclearfixの栄光の中で使われています。しかし、私たちは、多くの新機能が印象的に採用されていることも観察しました。しかし一般的には、何かがどれだけクールに認識されるかと、それがどれだけ実際に使われるかの間には逆相関があります。

同様に私たちのカンファレンスでの講演では、頭が爆発したり、Twitterのフィードが「CSSは*あれ*ができるのか！」で埋め尽くされたりするような、複雑で凝ったユースケースに焦点を当てることがよくあります。しかし、実際のところ、ほとんどのCSSの使用法は非常に単純なものです。CSS変数はほとんど定数として使われるし、他の変数を参照することはほとんどなく、`calc()`はほとんどが2つの用語で使われるし、グラデーションはほとんどが2つのストップを持つなどです。

ウェブはもう10代の若者ではなく、30歳になった今、そのように振る舞っています。それは、複雑さよりも安定性や読みやすさを優先する傾向がありますが、たまにある罪悪感はさておき。

方法論

HTTP Archive[3]は毎月数百万ページ[4]をクロールし、WebPageTest[5]のプライベートインスタンスを通して実行し、各ページのキー情報を保存しています。（これについての詳細はmethodologyを参照してください）。

今年は、どのメトリクスを勉強するか、コミュニティを巻き込むことにしました。私たちはまず、メトリクスを提案して投票するアプリ[6]から始めました。最終的には非常に多くの興味深いメトリクスがあったので、ほぼすべてのメトリクスを含めることにしましたが、フォントメトリクスだけを除外しました。

この章のデータは、121個のSQLクエリを使用し、SQL内の3000行のJavaScript関数を含む10000行以上のSQLを作成しました。これは、Web Almanacの歴史の中で最大の章となっています。

この規模の分析を可能にするため、多くの技術的な作業が行われました。昨年と同様に、私たちはすべてのCSSコードをCSSパーサー[7]に通し、コーパス内のすべてのスタイルシートの抽象構文木[8](AST)を保存しました。今年は、このAST上で動作するヘルパーのライブラリ[9]と

3. *https://httparchive.org/*
4. *https://httparchive.org/reports/state-of-the-web#numUrls*
5. *https://webpagetest.org/*
6. *https://projects.verou.me/mavoice/?repo=leaverou/css-almanac&labels=proposed%20stat*
7. *https://github.com/reworkcss/css*
8. *https://ja.wikipedia.org/wiki/%E6%8A%BD%E8%B1%A1%E6%A7%8B%E6%96%87%E6%9C%A8*
9. *https://github.com/leaverou/rework-utils*

セレクタパーサー[10]も開発しましたが、これらも別々のオープンソースプロジェクトとしてリリースされました。ほとんどのメトリクスでは、単一のASTからデータを収集するためにJavaScript[11]を使用し、コーパス全体のデータを集約するためにSQL[12]を使用しました。あなた自身のCSSが私たちのメトリクスに対してどのように機能しているか興味がありますか？私たちはonline playground[13]を作成し、自分のサイトで試してみることができます。

特定の指標については、CSS ASTを見るだけでは十分ではありませんでした。私たちは、ソースマップを介して提供されたSCSS[14]を見たいと考えていました。それは、開発者がCSSから何を必要としているかを示しているのに対し、CSSを研究することで開発者が現在使用しているものを示すことができるからです。そのためには、クローラーが特定のページを訪問したときにクローラーで実行されるカスタムメトリック—JavaScriptコードを使用しなければなりませんでした。適切なSCSSパーサーを使用するとクロールが遅くなりすぎるので、正規表現[15](*ああ、恐ろしい！*)に頼らざるを得ませんでした。粗野なアプローチにもかかわらず、私たちは多くの洞察を得ることができました！

カスタムメトリクスはカスタムプロパティの分析の一部にも使われていました。スタイルシートだけでもカスタムプロパティの使用状況に関する多くの情報を得ることができますが、カスタムプロパティは継承されているため、DOMツリーを見てコンテキストを確認できなければ依存関係グラフを構築することはできません。またDOMノードの計算されたスタイルを見ることで、各プロパティがどのような種類の要素に適用されているのか、どの要素がregistered[16]であるのかといった情報を得ることができます。

我々はデスクトップとモバイルの両方のモードでページをクロールしていますが、多くのデータでは同様の結果が得られるため、特に断りのない限り、この章で提示されている統計はモバイルページのセットを参照しています。

使用方法

ページ重量のシェアではJavaScriptがCSSをはるかに上回っていますが、CSSのサイズは年々確実に大きくなっておりデスクトップページの中央値では62KBのCSSコードが読み込まれ、10ページに1ページが240KB以上のCSSコードを読み込んでいます。モバイルページでは、すべてのパーセンタイルでCSSコードの使用量がわずかに減少していますが、その差は4〜7KBにとどまっています。これは間違いなく過去数年よりも多いですが、[JavaScriptの中央値444KBと1.2MBのトップ10%]には及ばないですね(./javascript#how-much-javascript-do-we-use)

10. *https://projects.verou.me/parsel*
11. *https://github.com/LeaVerou/css-almanac/tree/master/js*
12. *https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2020/css*
13. *https://projects.verou.me/css-almanac/playground*
14. *https://sass-lang.com/*
15. *https://github.com/LeaVerou/css-almanac/blob/master/runtime/sass.js*
16. *https://developer.mozilla.org/docs/Web/API/CSS/RegisterProperty*

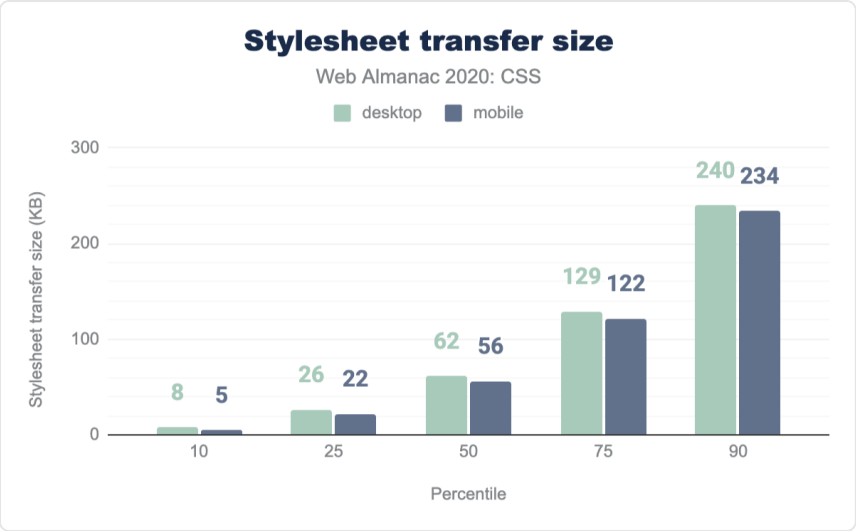


図1.1. 1ページあたりのスタイルシート転送サイズの分布。

これらのCSSの多くはプリプロセッサや他のビルドツールで生成されていると考えるのが妥当でしょうが、ソースマップが含まれているのは約15%に過ぎませんでした。これがソースマップの採用を意味しているのか、それともビルドツールの使用状況を意味しているのかは不明です。これらのうち、圧倒的多数（45%）は他のCSSファイルからのもので、ミニ化、autoprefixer[17]、PostCSS[18]など、CSSファイルを操作するビルドプロセスを利用していることを示しています。Sass[19]は、Less[20]よりもはるかに人気が高く（ソースマップを持つスタイルシートの34%対21%）、SCSSの方が人気がありました（.scssの33%対.sassの1%）。

これらすべてのキロバイトのコードは、通常、複数のファイルと`<style>`要素に分散されています。実際、中央値のページには3つの`<style>`要素と6つのリモートスタイルシートがあり、そのうちの10%は14個の`<style>`要素と20個以上のリモートCSSファイルを持っています。これはデスクトップで最適ではありませんが、生のダウンロード速度よりも往復のレイテンシが重要なモバイルでは、パフォーマンスを本当に殺してしまいます。

1,379

図1.2. ページによって読み込まれるスタイルシートの数が最も多い。

17. *https://autoprefixer.github.io/*
18. *https://postcss.org/*
19. *https://sass-lang.com/*
20. *https://lesscss.org/*

驚くべきことに、1ページあたりのスタイルシートの最大数は、`<style>`要素が26,777個、リモートのものが1,379個という驚異的な数なのです！　そのページを読み込むのは絶対に避けたいですね！

図1.3. 1ページあたりのスタイルシート数の分布。

サイズのもう1つの指標は、ルールの数です。中央値のページには、合計448のルールと5,454の宣言が含まれています。興味深いことに、ページの10%はわずかな量のCSSを含んでおり、13のルールよりも少ないのです。モバイルの方がスタイルシートがわずかに小さいにもかかわらず、ルール数がわずかに多く、全体的にルールが小さいことを示しています（メディアクエリで起こりがちなことですが）。

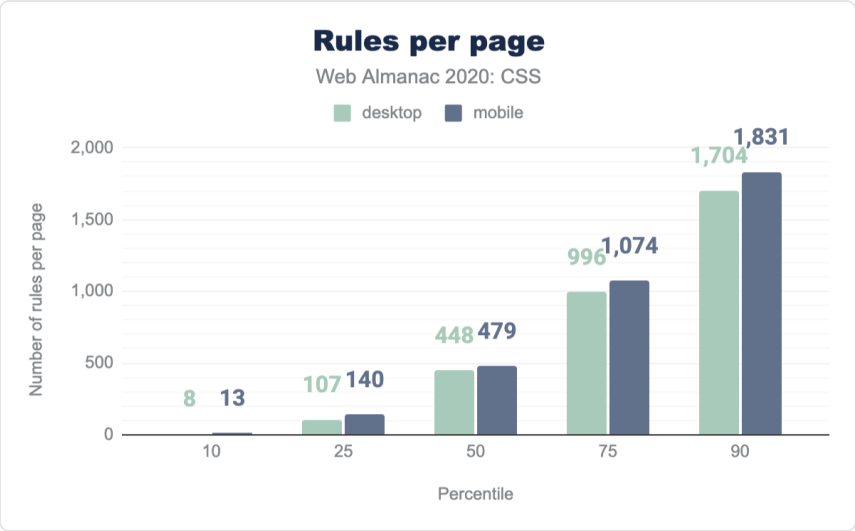


図1.4. 1ページあたりのスタイルルールの総数の分布。

セレクターとカスケード

CSSには、クラスやID、スタイルの重複を避けるために重要なカスケードの使用など、ページにスタイルを適用する方法がたくさんあります。では、開発者はどのようにしてページにスタイルを適用しているのでしょうか？

クラス名

最近の開発者は何にクラス名を使っているのでしょうか？　この質問に答えるために、最も人気のあるクラス名を調べてみました。そのリストは、Font Awesome[21]クラスが占めており、198個中192個が `fa` または `fa-*` でした。最初に調査してわかったことは、Font Awesomeは非常に人気があり、ほぼ3分の1のWebサイトで使用されているということだけでした。

しかし、一度 `fa-*` クラスと `wp-*` クラス（これも非常に人気のあるソフトウェアであるWordPress[22]に由来する）を分解してみると、より意味のある結果が得られました。これらを省略すると、状態関連のクラスが最も人気があるようで、 `.active` が半数近くのウェブサイトで出現し、 `.selected` と `.disabled` がその直後に続いています。

21. https://fontawesome.com/
22. https://wordpress.com/

上位クラスのうち、プレゼンテーション的なものはわずか数個だけで、それらのほとんどはアライメントに関連したもの（古いBootstrap[23]の`pull-right`や`pull-left`、`alignright`, `alignleft`など）か`clearfix`で、フロートがより近代的なGridやFlexboxモジュールによってレイアウト方法として置き換えられているにもかかわらず、いまだにウェブサイトの22%で発生しています。

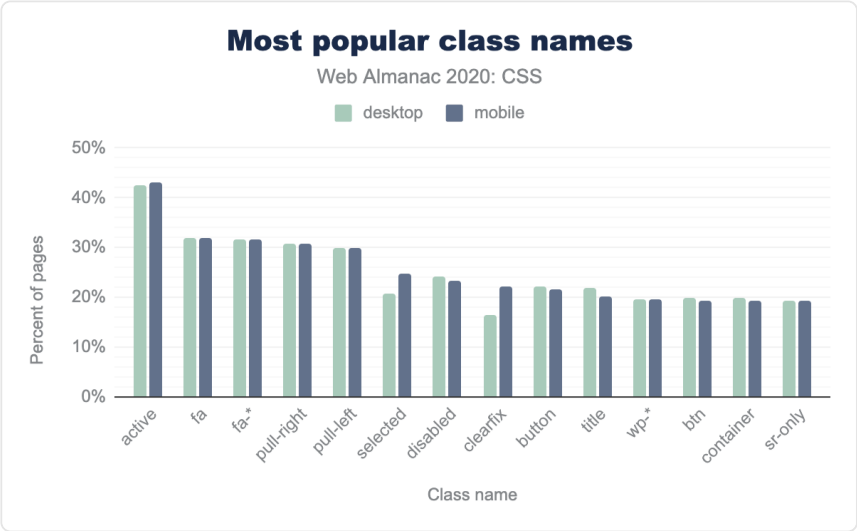


図1.5. ページ数の割合で最も人気のあるクラス名です。

ID

最近ではIDの特異度が非常に高いため、一部の業界では推奨されていませんが、ほとんどのウェブサイトでは、ほとんど使用されていません。セレクターで複数のIDを使用しているページは半数以下（最大特異度は(1,x,y)以下）で、ほぼ全てのページでIDを含まない特異度の中央値は(0,x,y)となっています。特異度の計算の詳細とこの(a,b,c)表記法については、selectors specification[24]を参照してください。

しかし、これらのIDは何に使われているのでしょうか？　最も人気のあるIDは構造的なものであることがわかりました: `#content`, `#footer`, `#header`, `#main`ですが、対応するHTML要素[25]が存在するにもかかわらず、セレクターとして使用できます。

23. https://getbootstrap.com/
24. https://www.w3.org/TR/selectors/#specificity-rules
25. https://developer.mozilla.org/docs/Learn/HTML/Introduction_to_HTML/Document_and_website_structure#html_layout_elements_in_more_detail

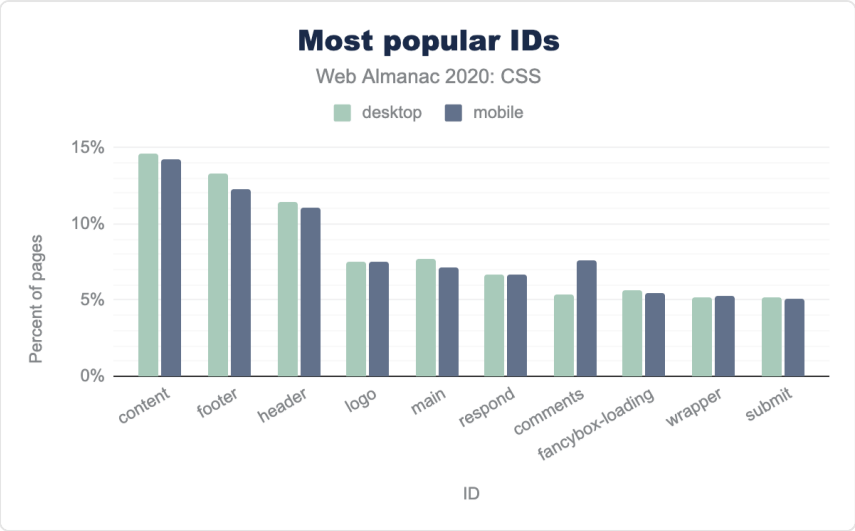


図1.6. ページの割合で最も人気のあるIDです。

IDは、意図的に特異度を下げたり、上げたりするために使うこともできます。IDセレクタを属性セレクタとして書くという特異性ハック[26]（特異性を減らすために`#foo`の代わりに`[id="foo"]`を使う）は驚くほど珍しく、少なくとも一度は0.3%のページでしか使われていませんでした。もうひとつのID関連の特異性ハックは、`.foo`の代わりに`:not(#nonexistent) .foo`のような否定＋子孫セレクターを使って特異性を高めるというものですが、これも非常に稀で、0.1%のページでしか使われていませんでした。

`!important`

代わりに、古くて粗雑な`!important`は、そのよく知られた欠点[27]にもかかわらず、まだかなり使われています。中央値のページでは、宣言のほぼ2%、つまり50分の1で`!important`が使用されています。

2,138

図1.7. すべての宣言で`!important`を使用するモバイルページ！

一部の開発者は文字通りそれを十分に得ることができません。すべての宣言で`!important`

26. https://csswizardry.com/2014/07/hacks-for-dealing-with-specificity/
27. https://www.impressivewebs.com/everything-you-need-to-know-about-the-important-css-declaration/#post-475:~:text=Drawbacks,-to

を使用する2304のデスクトップページと2138のモバイルページが見つかりました！

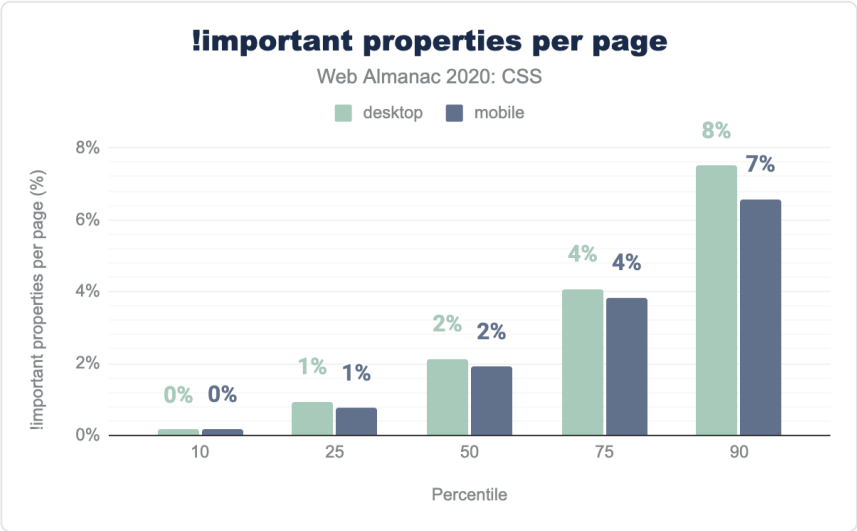


図1.8. ページごとの `!important` プロパティの割合の分布。

開発者がオーバーライドすることに熱心なのは何ですか？　プロパティごとの内訳を調べたところ、80%近くのページが `display` プロパティで `!important` を使用していることがわかりました。 `display : none !important` を適用してヘルパークラスのコンテンツを非表示にし、 `display` を使用してレイアウトモードを定義する既存のCSSをオーバーライドするのが一般的な戦略です。 これは、後から考えると、CSSの欠陥であったものの副作用です。3つの直交する特性を1つに組み合わせました。内部レイアウトモード、フロー動作、および可視性ステータスはすべて、 `display` プロパティによって制御されます。これらの値を個別の `display` キーワードに分けて、カスタムプロパティを介して個別に調整できるようにする努力がありますが、当分の間ブラウザのサポートは事実上存在しません[28]。

28. *https://caniuse.com/mdn-css_properties_display_multi-keyword_values*

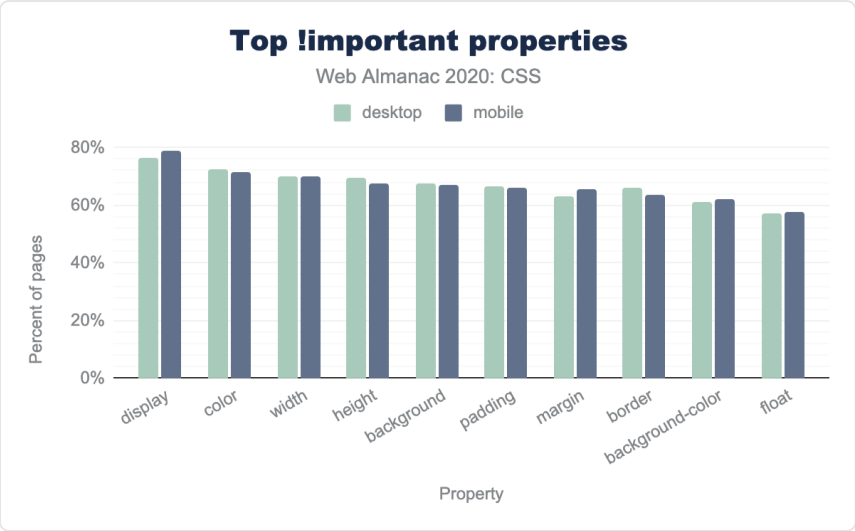


図1.9. ページの割合で上位の `!important` プロパティを表示します。

特異性とクラス

`id` と `!important` をほとんど使わないようにすることに加えて、セレクターのすべての選択基準を単一のクラス名に詰め込むことで特異性を完全に回避する傾向があり、その結果すべてのルールが同じ特異性を持つことを強制し、カスケードをより単純なラストワンウィンシステムに変える。BEMは、唯一のものではありませんが、このタイプの一般的な方法論です。すべてのルールでBEMスタイルの方法論に従うことは稀なので、どのくらいの数のウェブサイトがBEMスタイルの方法論を独占的に使用しているのかを評価することは困難ですが（BEMのウェブサイト[29]でさえ、多くのセレクターで複数のクラスを使用しています）、ページの約10%の中央値が(0,1,0)の特異度を持っており、これはほとんどがBEMスタイルの方法論に従っていることを示しているかもしれません。BEMの反対側では、開発者はしばしば重複クラス[30]を使って特異性を高め、セレクターを別のセレクターよりも先に進めるようにしています（例えば、 `.foo` の代わりに `.foo.foo.foo` とするなど）。この種の特異性ハックは実際にはBEMよりも人気があり、モバイルサイトの14%（デスクトップの9%）に存在しています。これは、ほとんどの開発者がカスケードを完全に排除することを望んでいるわけではなく、単にカスケードをよりコントロールしたいだけだということを示しているのかもしれません。

29. https://en.bem.info/
30. https://csswizardry.com/2014/07/hacks-for-dealing-with-specificity/#safely-increasing-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	0,2,0	0,1,2
50	0,2,0	0,2,0
75	0,2,0	0,2,0
90	0,3,0	0,3,0

図1.10. 1ページあたりの特異度中央値の分布。

セレクター属性

最も人気のある属性セレクターは`type`属性で、45%のページで使用されており、異なるタイプの入力をスタイルする可能性があります。

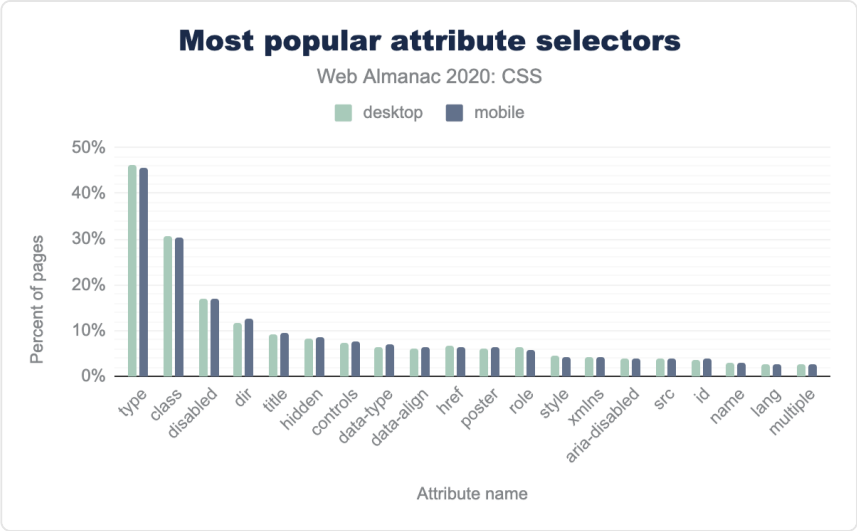


図1.11. ページの割合で最も人気のあるセレクター属性。

擬似クラスと擬似要素

ウェブプラットフォームが長く確立された後に何かを変更するときには、常に多くの慣性があります。例えば10年以上前に変更があったにもかかわらず、疑似要素が疑似クラスとは別

の構文を持つことに、ウェブはまだほとんど追いついていません。レガシーな理由で疑似クラスの構文でも利用できるすべての疑似要素は、疑似クラスの構文の方がはるかに普及しています（2.5倍から5倍！）。

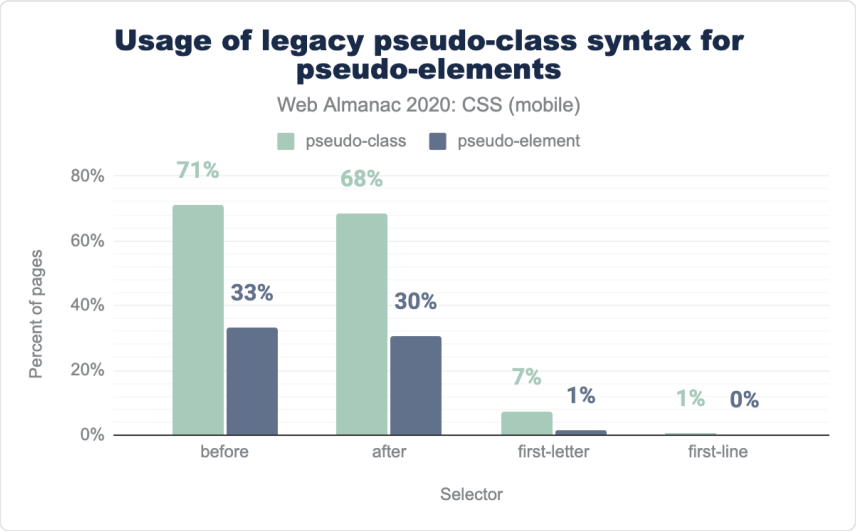


図1.12. レガシーの `:pseudo-class` 構文を `::pseudo-elements` のモバイルページの割合として使用します。

圧倒的に最も人気のある擬似クラスはユーザーアクションのもので、`:hover`, `:focus`, `:active` がリストのトップにあり、すべてのページの3分の2以上で使用されていて、開発者が宣言的なUIインタラクションを指定できる利便性を気に入っていることがわかります。

`:root` は、その機能が正当化されるよりもはるかに人気があるようで、3分の1のページで使用されています。HTMLコンテンツでは、`<html>` 要素を選択するだけなのに、なぜ開発者は `html` を使わなかったのでしょうか？　考えられる答えは、`:root` 擬似クラスへカスタムプロパティを定義することに関連した一般的な慣習にあるかもしれませんこれも非常によく使われています。もう1つの答えは特異性にあるかもしれません: `:root` は擬似クラスであるため、`html` よりも高い特異性を持っています。(0, 1, 0) 対 (0, 0, 1) です。例えば、`:root.foo` の特異度は (0, 2, 0) であるのに対し、`.foo` の特異度は (0, 1, 0) です。これは、カスケードレースの中でセレクターが他のセレクターよりもわずかに上を行くのに必要なことが多く、`!important` のようなスレッジハンマーを避けるために必要なことです。この仮説を検証するために、私たちはまた、まさにこれを測定しました：子孫セレクターの先頭に `:root` を使うページがどれくらいあるか？　結果は我々の仮説を検証しました：29%のページがそのように `:root` を使用しています。さらにデスクトップページの14%とモバイルページの19%が、子孫セレクターの開始時に `html` を使用しており、おそらくセレクター

の特異性をさらに小さくするために使用していると思われます。これらの特異性ハックの人気は、開発者が `!important` を介してそれらに与えられているものよりも特異性を微調整するためのより細かい制御を必要とすることを強く示しています。ありがたいことに、これは `:where()` でまもなく実現します。すでに 全面的に実装されています[31]（今のところChromeではフラグの後ろに隠れていますが）。

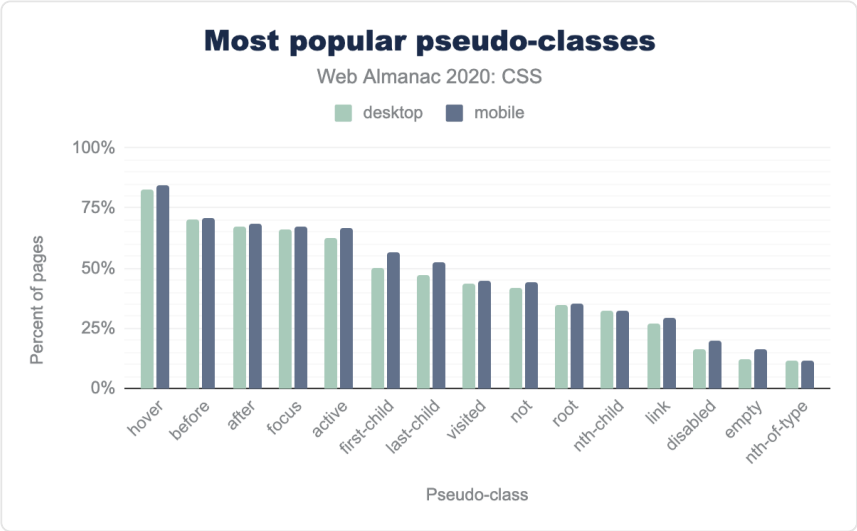


図1.13. ページ数に占める割合としては、最も人気のある疑似クラス。

疑似要素に関しては通常の疑似要素である `::before` と `::after` に続いて、人気のある疑似要素のほとんどは、フォームコントロールやその他の組み込みUIをスタイリングするためのブラウザ拡張機能であり、組み込みUIのスタイリングをより細かく制御したいという開発者のニーズを強く反映しています。フォーカスリング、プレースホルダー、検索入力、スピナー、選択、スクロールバー、メディアコントロールのスタイリングは特に人気がありました。

31. *https://caniuse.com/mdn-css_selectors_where*

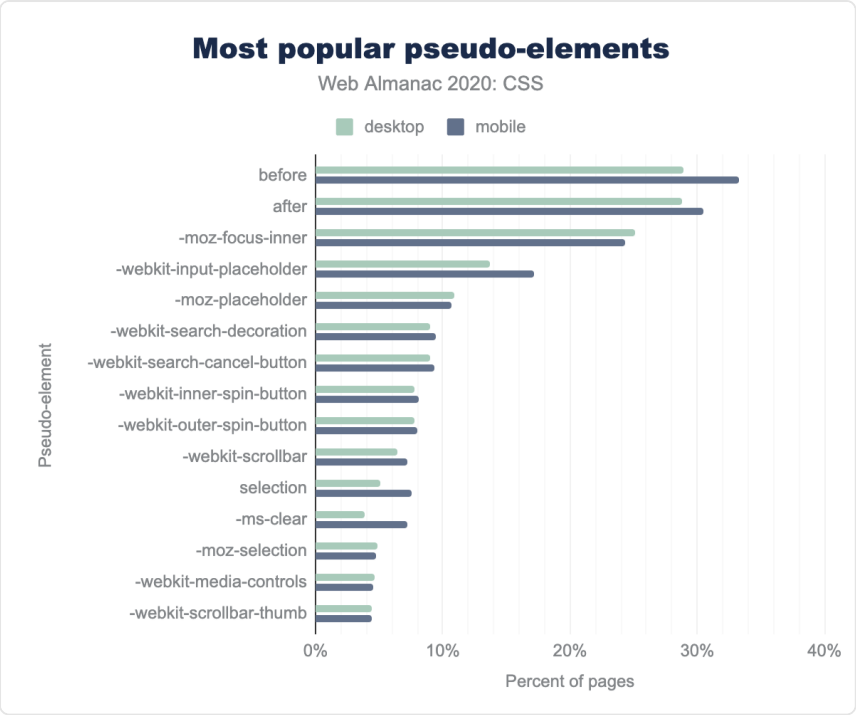


図1.14. ページ数に占める割合としては、最も人気のある疑似要素です。

値と単位

CSSには値や単位を指定する方法が多数用意されており、長さの設定や計算、あるいはグローバルキーワードに基づいて指定できます。

長さ

地味な `px` 単位は、長年にわたって多くの否定的な報道を受けてきました。最初は古いInternet Explorerのズーム機能との相性が悪かったことが原因でしたが、最近ではビューポートサイズ、要素フォントサイズ、ルートフォントサイズなどの別のデザイン要素に基づいてスケーリングする、ほとんどのタスクに対応する優れた単位が登場し暗黙のデザイン関係を明示的にすることでメンテナンスの手間を減らすことができました。ピクセルの主なセールスポイントであった `px` -デザイナーが完全に制御できる1つのデバイスピクセルに対応するという点も、現在の高ピクセル密度スクリーンではピクセルはもはやデバイスピクセルではないため、今ではなくなっています。このような状況にもかかわらず、CSSピクセルは今

でもウェブのデザインをほぼ全面的に牽引しています。

72.58%

図1.15. px 単位を使用している <length> 値の割合。

px 単位は全体的に最も人気のある長さの単位であり、全スタイルシートの長さの72.58%が px を使用しています。さらにパーセンテージを除外すると（パーセンテージは実際には単位ではないので）、 px のシェアはさらに増えて84.14%となっています。

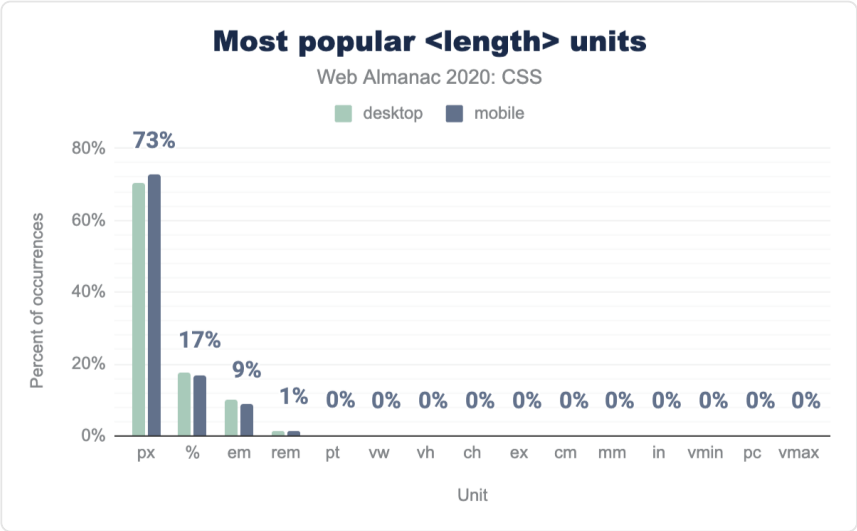


図1.16. 最も人気のある <length> の単位を出現率で表したものです。

この px はプロパティ間でどのように分布しているのでしょうか？　プロパティによって違いはありますか？　ほとんど間違いありません。例えば、 font-size や line-height 、 text-indent などのフォント関連のメトリクスに比べて、 px の使用率はボーダーの方がはるかに高い（80-90%）と予想されます。しかし、これらの場合でも px の使用率は他の単位をはるかに上回っています。実際、 px よりも他のユニット（ *他の* ユニット）が使われているプロパティは、 vertical-align (55% em)、 mask-position (50% em)、 padding-in-line-start (62% em)、 margin-block-start と margin-block-end (65% em)、そして新しい gap (62% rem) だけです。

これらのコンテンツの多くは古いもので、相対単位を使ってデザインの適応性を高めたり、時間を節約したりすることに作者がもっと意識を向けるようになる前に書かれたものだと簡

単に論じることができます。しかし、これは`grid-gap`(62%`px`)のような最近のプロパティを見れば簡単に反論できます。

プロパティ	px	<number>	em	%	rem	pt
font-size	70%	2%	17%	6%	4%	2%
line-height	54%	31%	13%	3%		
border	71%	27%	2%			
border-radius	65%	21%	3%	10%		
text-indent	32%	51%	8%	9%		
vertical-align	29%	12%	55%	4%		
grid-gap	63%	11%	9%	1%	16%	
mask-position			50%	50%		
padding-inline-start	33%	5%	62%			
gap	21%	16%	1%		62%	
margin-block-end	4%	31%	65%			
margin-inline-start	38%	46%	14%		1%	

図1.17. プロパティ別の単位使用量。

同様に、多くのユースケースで`rem`と`em`の優位性が大いに宣伝されており、ブラウザでの普遍的なサポートが 何年にもわたって[32]されているにもかかわらず、ウェブはまだほとんど追いついていないのが現状です。野生では`ch`（'0'グリフの幅）と`ex`（使用中のフォントのx-height）の使用も見られましたが、非常に小さいものでした（全フォント関係単位の0.37%と0.19%に過ぎませんでした）。

32. https://caniuse.com/rem

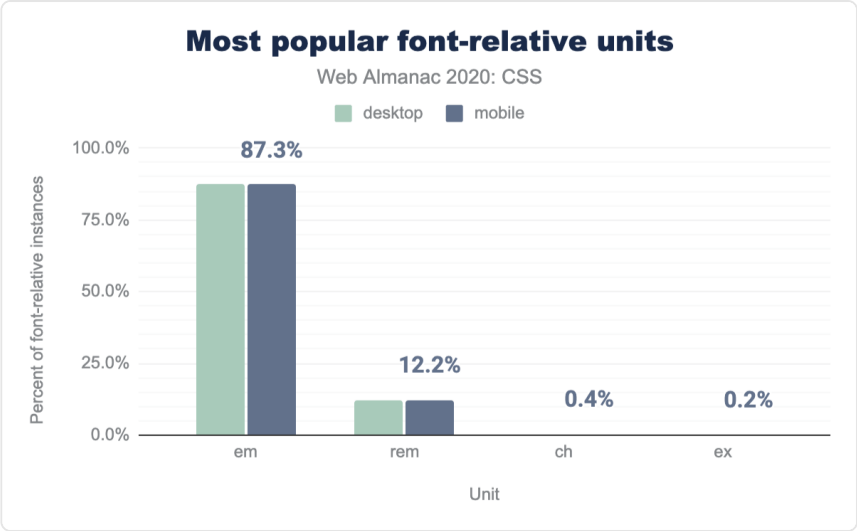


図1.18. フォント相対単位の相対シェア

つまり、`0px`や`0em`などの代わりに`0`と書くことができます。開発者（あるいはCSSのミニファイヤー？）はこれを広く利用しています。すべての`0`値のうち、89%は単位がありませんでした。

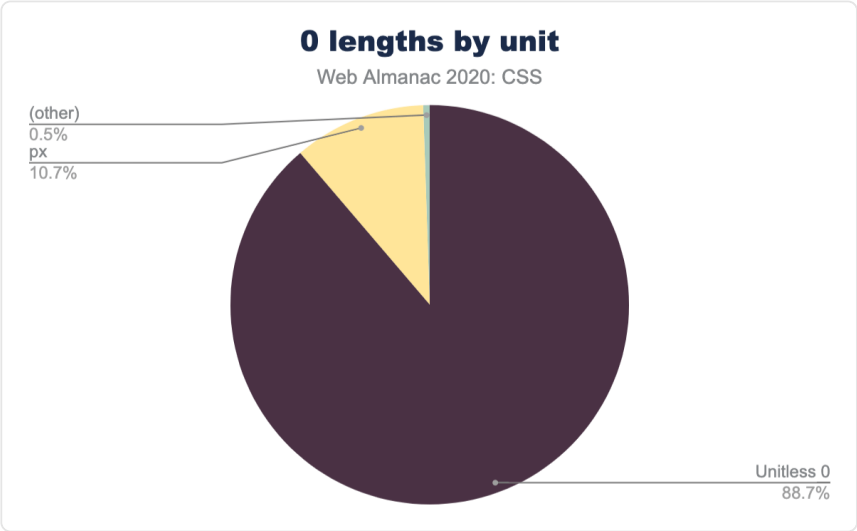


図1.19. モバイルページでの出現率としての単位ごとの長さ0の相対的な人気度。

計算

CSSで異なる単位間の計算を行うために`calc()`関数が導入されたとき、それは革命でした。以前はプリプロセッサだけがこのような計算に対応していましたが、結果は静的な値に限定され、しばしば必要とされる動的なコンテキストを欠いていたため、信頼性がありませんでした。

今日では、`calc()`はすべてのブラウザでサポートされている[33]ということで、すでに9年が経過しているので60%のページで一度は使われており、広く採用されているのは驚くに値しません。どちらかと言えば、これよりももっと高い採用率を期待していました。

`calc()`は主に長さのために使われ、その96%は`<length>`値を受け付けるプロパティに集中しており、そのうちの60%（全体の58%）は`width`プロパティに使用されています！

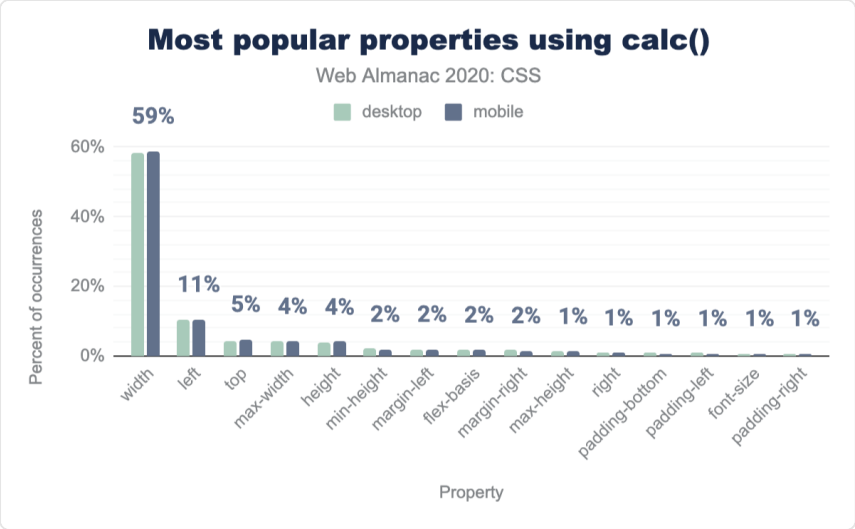


図1.20. `calc()`を使用するプロパティの相対的な人気度を、出現率の割合で示します。

これは、`calc()`で最もよく使われる単位が`px`（`calc()`の51%）と`%`（`calc()`の42%）であり、`calc()`の64%が減算を含んでいることからも明らかなように、ほとんどがパーセントからピクセルを引くために使われているようです。興味深いことに、`calc()`で最も人気のある長さの単位は全体的に最も人気のある長さの単位とは異なります（例えば、`em`よりも`rem`の方が人気があり、次いでビューポート単位が続きます）。

33. *https://caniuse.com/calc*

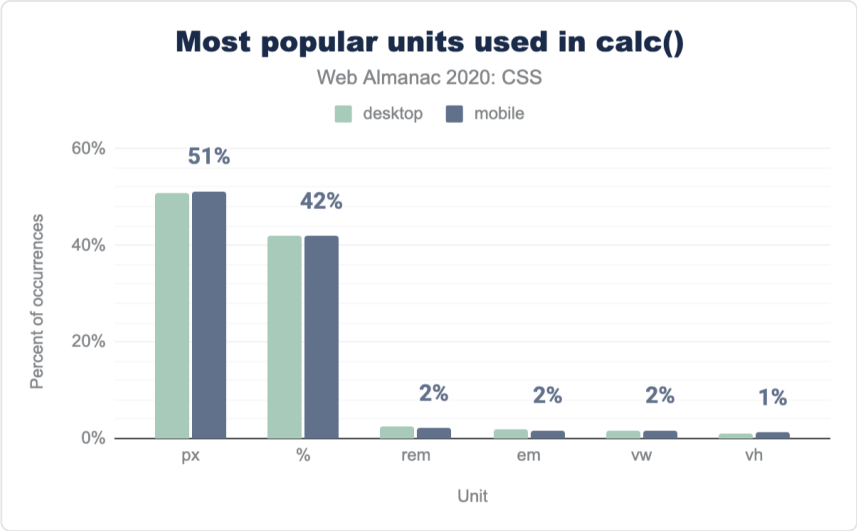


図1.21. `calc()` を使用する単位の相対的な人気度を、出現率の割合で示します。

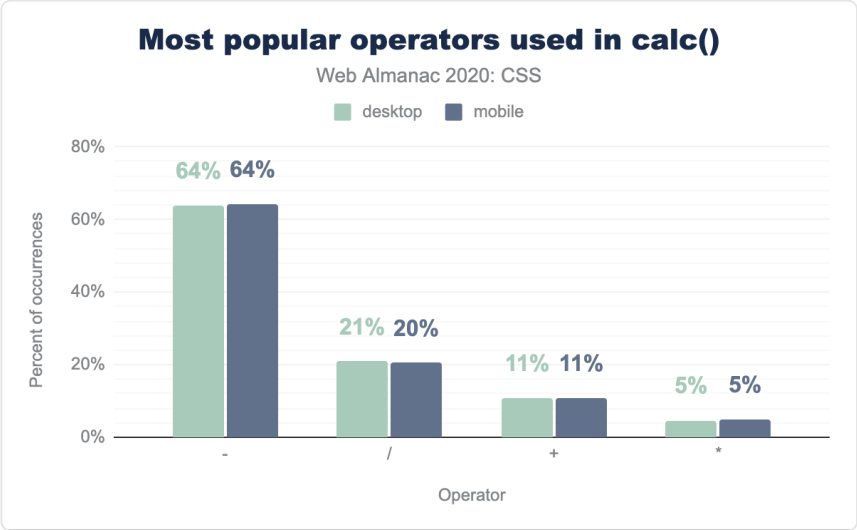


図1.22. `calc()` を使用する演算子の相対的な人気度を、出現回数の割合で示します。

ほとんどの計算は非常に単純で、最大2つの異なる単位を含む計算の99.5%、最大2つの演算子を含む計算の88.5%、1セット以下の括弧を含む計算の99.4%（4つの計算のうち3つは括弧が全く含まれていません）となっています。

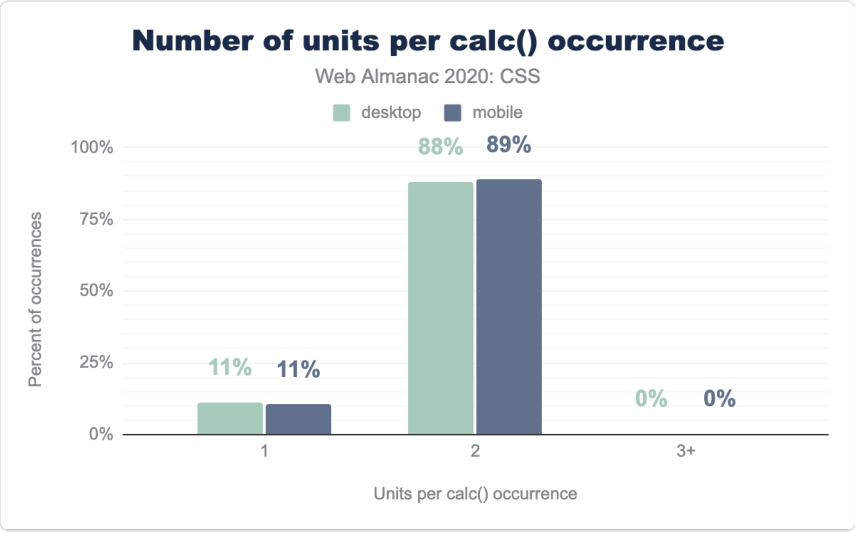


図1.23. `calc()` の発生ごとの単位数の分布。

グローバルキーワードと `all`

それは `inherit` で、継承可能なプロパティを継承された値にリセットしたり、継承不可能なプロパティに対して親の値を再利用したりすることを可能にします。前者の方が後者よりもはるかに一般的で、 `inherit` の81.37%が継承可能なプロパティで使用されています。残りのほとんどは、背景、境界線、寸法を継承するためのものです。後者は、適切なレイアウトモードでは `width` と `height` を強制的に継承する必要がほとんどないため、レイアウトの難しさを示している可能性が高いです。

`inherit` キーワードは、リンクのアフォーダンスとして色以外のものを使おうとしている場合に、デフォルトのリンク色を親のテキスト色にリセットするのに特に便利です。したがって、 `color` が `inherit` で使用される最も一般的なプロパティであることは驚くことではありません。すべての `inherit` の使用量のほぼ3分の1は `color` プロパティにあります。75%のページでは、少なくとも一度は `color: inherit` を使用しています。

プロパティの*初期値*はCSS1の頃から存在していた[34]概念ですが、それを明示的に参照するための専用のキーワード `initial` ができたのは17年後[35]で、そのキーワードがブラウザのユニバーサルサポート[36]になるまでにはさらに2年かかりました。したがって、 `inherit` よりもはるかに使用されていないのは当然のことです。古い継承が85%のページで見られるのに対

34. *https://www.w3.org/TR/CSS1/#cascading-order*
35. *https://www.w3.org/TR/2013/WD-css3-cascade-20130103/#initial-keyword*
36. *https://caniuse.com/css-initial-value*

し、`initial`は51%のページで見られます。さらに、`initial`が実際に何をするのかについては多くの混乱があり、`display`は`initial`とともに最もよく使われるプロパティのリストのトップにあり、`display: initial`はページの10%に表示されています。おそらく開発者は、これによって`display`がuser agent stylesheet[37]の値にリセットされ、`display: none`のオンオフを切り替えるために利用されていると考えたのだろう。しかし、`display`の初期値は`inline`であるので、`display: initial`は単に`display: inline`を書くための別の方法であり、コンテキストに依存した魔法のような特性を持たない。

代わりに、`display: revert`は、これらの開発者が期待していたことを実際に行い、`display`を与えられた要素のUA値にリセットすることになるだろう。しかし、`revert`はもっと新しいもので、定義されたのは2015年[38]であり、今年ユニバーサルブラウザのサポートを得ただけ[39]であるため、あまり使われていないことがわかります：ページの0.14%にしか表示されず、使用量の半分はWordPressのTwentyTwentyテーマの最近のバージョン[40]にある`line-height: revert;`です。

最後のグローバルキーワード`unset`は、本質的に`initial`と`inherit`のハイブリッドです。継承されたプロパティでは`inherit`になり、それ以外のプロパティでは`initial`になります。同様に、`initial`と同様に、`unset`は2013年に定義[41]され、2015年にはブラウザのフルサポート[42]されました。`unset`の方が実用性が高いにもかかわらず、`initial`が51%のページで使用されているのに対し、`unset`は43%のページでしか使用されていません。さらに、`max-width`と`min-width`以外のすべてのプロパティにおいて、`initial`の使用率が`unset`の使用率を上回っています。

37. *https://developer.mozilla.org/docs/Web/CSS/Cascade#user-agent_stylesheets*
38. *https://www.w3.org/TR/2015/WD-css-cascade-4-20150908/#valdef-all-revert*
39. *https://caniuse.com/css-revert-value*
40. *https://github.com/WordPress/WordPress/commit/303180b392c530b8e2c8b3c27532d591b915caeb*
41. *https://www.w3.org/TR/2013/WD-css-cascade-3-20130730/#inherit-initial*
42. *https://caniuse.com/css-unset-value*

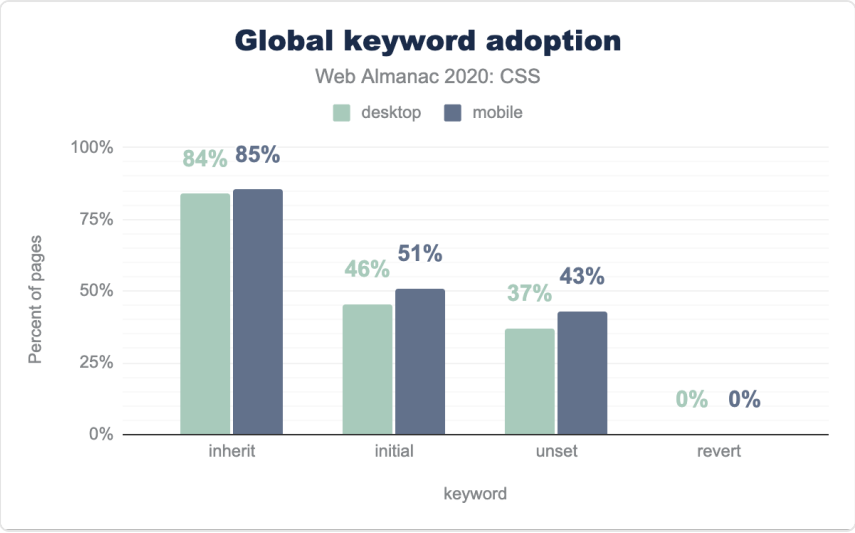


図1.24. グローバルキーワードの採用率がページ数に占める割合。

`all` プロパティは2013年に導入[43] され、2016年にはほぼユニバーサルに近い形でサポートされ(Edgeを除く)、今年初めにはユニバーサルに対応[44] しました。これは、CSSのほぼすべてのプロパティ（カスタムプロパティ、`direction`、`unicode-bidi` を除く）の短縮形であり、4つのグローバルキーワード[45]（`initial`、`inherit`、`unset`、`revert`）のみを値として受け入れます。これは、どのようなリセットをしたいかに応じて `all: unset` や `all: revert` のように、CSSのリセットを一本化することを想定していました。しかし、まだ採用率は非常に低く、`all` が見つかったのは477ページ（全ページの0.01%）で、`revert` キーワードと一緒に使われただけでした。

カラー

古いジョークは最高だと言われますが、それは色にも当てはまります。オリジナルの暗号的な `#rrggbb` 16進法は、2020年のCSSで色を指定するための最も一般的な方法であり続けています。全色の半分はこの方法で書かれています。次に人気のある形式は、やや短めの `#rgb` 3桁の16進数形式で、26%となっています。これはより短いとはいえ、表現できる色数はかなり少なく、1670万個のsRGB値のうちわずか4096個です。

43. *https://www.w3.org/TR/2013/WD-css3-cascade-20130103/#all-shorthand*
44. *https://caniuse.com/css-all*
45. *https://drafts.csswg.org/css-cascade-4/#defaulting-keywords*

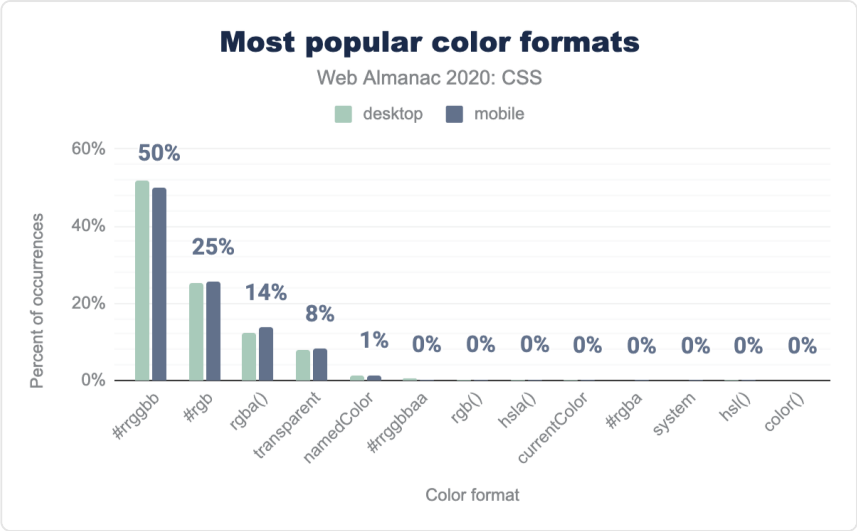


図1.25. カラーフォーマットの相対的な普及率。

同様に、機能的に指定されたsRGBカラーの99.89%は、カンマなしの新しい形式`rgb(127 255 84)`ではなく、コンマ付きのレガシー形式`rgb(127, 255, 84)`を使用しています。なぜなら、すべての最新のブラウザが新しい構文を受け入れているにもかかわらず、変更しても開発者にとってのメリットはゼロだからです。

では、なぜ人々はこれらの試行錯誤されたフォーマットから迷走するのでしょうか？アルファの透過性を表現するためです。これは`rgba()`を見れば明らかで、`rgb()`の40倍の使用率（全色の13.82%対0.34%）と`hsl()`の30倍の使用率（全色の0.25%対0.01%）である`hsla()`を見ればわかります。

HSLはわかりやすく、修正しやすい[46]はずです。しかし、これらの数字は、実際にはスタイルシートでのHSLの使用がRGBよりもはるかに少ないことを示しています。

46. https://drafts.csswg.org/css-color-4/#the-hsl-notation

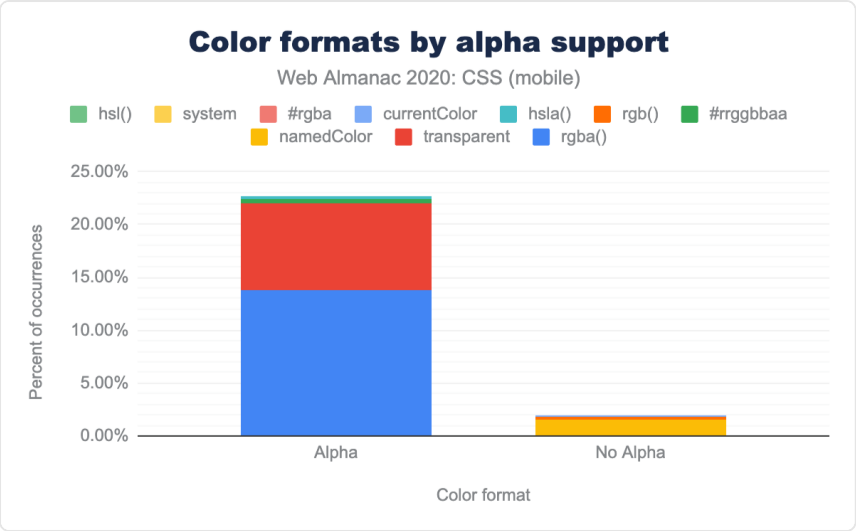


図1.26. カラーフォーマットの相対的な人気度をアルファサポート別にグループ化し、モバイルページでの出現率で示したもの（`#rrggbb`および`#rgb`を除く）。

名前付きカラーについてはどうでしょうか？　透明な`transparent`というキーワードは、`rgb(0 0 0 0 / 0)`という言い方もできますが、最も人気のあるキーワードで、sRGBの値全体の8.25%を占めています（名前付きカラーの使用量全体の66%）。これらの中で最も人気があったのは、`white`, `black`, `red`, `gray`, `blue`のようなわかりやすい名前でした。非日常的な名前の中では`whitesmoke`が最もよく使われていました（確かに、whitesmokeはイメージできますよね）が、`gainsboro`, `lightCoral`, `burlywood`などはあまり使われていませんでした。その理由は理解できますが、実際に何を意味するのかは調べてみる必要があります。

また、派手な色の名前をつけたいのであれば、CSSCustom propertiesで自分の色を定義してみてはいかがでしょうか。`--intensePurple`と`--corporateBlue`は、あなたが必要とするものなら何でも意味を持ちます。これはおそらく、カスタムプロパティの50%が色に使われている理由を説明していると思います。

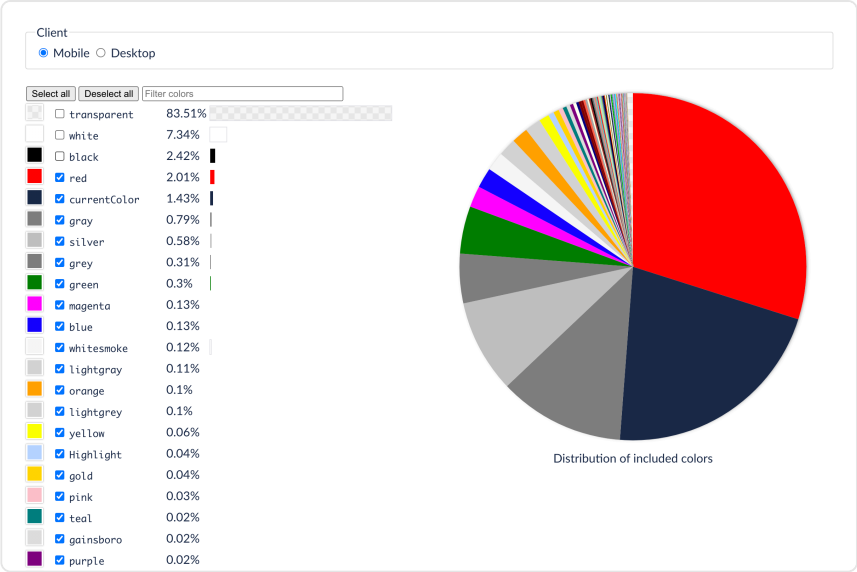


図1.27. このインタラクティブなアプリ[47]を使って、カラーキーワードの使用データをインタラクティブに探ってみましょう！

47. https://codepen.io/leaverou/pen/GRjjJwJ

	キーワード	デスクトップ	モバイル
	transparent	84.04%	83.51%
	white	6.82%	7.34%
	black	2.32%	2.42%
	red	2.03%	2.01%
	currentColor	1.43%	1.43%
	gray	0.75%	0.79%
	silver	0.66%	0.58%
	grey	0.35%	0.31%
	green	0.36%	0.30%
	magenta	0.00%	0.13%
	blue	0.16%	0.13%
	whitesmoke	0.17%	0.12%
	lightgray	0.06%	0.11%
	orange	0.12%	0.10%
	lightgrey	0.04%	0.10%
	yellow	0.08%	0.06%
	Highlight	0.01%	0.04%
	gold	0.04%	0.04%
	pink	0.03%	0.03%
	teal	0.03%	0.02%

図1.28. カラーキーワードの相対的な人気度を出現率で表す。

そして最後に、`Canvas`や`ThreeDDarkShadow`のような、かつては非推奨だったが、今では部分的に非推奨になっているシステムカラー: これらはひどいアイデアでJavaやWindows 95のようなものの典型的なユーザーインターフェースをエミュレートするために導入されましたが、すでにWindows 98には追いつけず、すぐに廃れてしまいました。いくつかのサイトでは、これらのシステムカラーを使ってあなたの指紋を採取しようとしていますが、この抜け穴は私たちが話している間に閉鎖しようとしています[48]。これらを使用する正当な理由はほとんどなく、ほとんどのウェブサイト(99.99%)は使用していないので、私たちは問題ありません。

驚くべきことに、かなり有用な値`currentColor`は、全sRGB色の0.14%（名前のついた色の1.62%）にとどまりました。

これまで議論してきたすべての色に共通しているのは、ウェブの標準色空間であるsRGB（そしてハイビジョンテレビの場合は、そこから来ている）ということです。なぜそれがそんなに悪いのでしょうか？　それは、限られた色の範囲しか表示できないからです。携帯電話、テレビ、そしておそらくあなたのラップトップは、ディスプレイ技術の進歩により、はるかに鮮やかな色を表示することができます。以前は高給取りのプロの写真家やグラフィックデザイナーのために予約されていた広い色域を持つディスプレイは、今では誰もが利用できるようになりました。ネイティブアプリはデジタル映画やストリーミングTVサービスと同様にこの機能を使用していますが、最近までウェブはこの機能を利用していませんでした。

そして、我々はまだ見落としている。2016年にSafariで実装された[49]にもかかわらず、ウェブページでのdisplay-p3カラーの使用は、ごくわずかです。私たちがウェブをクロールしたところ、モバイルでは29ページ、デスクトップでは36ページしか使用されていませんでした。（そして、その半分以上は構文エラーやミス、未実装の`color-mod()`関数を使おうとしたものでした）。私たちはその理由を知りたいと思いました。

相性だよね？　壊れてほしくないですよね？　私たちが調べたスタイルシートでは、フォールバックがしっかりと使われていました:ドキュメントの順番、カスケード、`@supports`、`color-gamut`メディアクエリなど、良いものはすべて揃っていました。スタイルシートの中には、デザイナーが望む色が表示され、display-p3で表現され、フォールバックのsRGB色も表示されていました。希望する色と予備色の間の目に見える差（ΔE2000[50]と呼ばれる計算）を計算してみましたが、これは一般的には非常に控えめなものでした。ちょっとした微調整。慎重な探索。実際、display-p3で指定した色は、sRGBが管理できる色の範囲（色域）内に収まっていることが37.6%もありました。今のところは、実際に利益を得るためというよりも、慎重に実験しているだけのようだが、今後の展開に期待したいところだ。

48. https://github.com/w3c/csswg-drafts/issues/5710
49. https://webkit.org/blog/6682/improving-color-on-the-web/
50. https://zschuessler.github.io/DeltaE/learn/

sRGB	display-p3	ΔE2000	色域
rgba(255,205,63,1)	color(display 1 0.80 0.25 / 1)	3.880	false
rgba(120,0,255,1)	color(display 0.47 0 1 / 1)	1.933	false
rgba(121,127,132,1)	color(display 0.48 0.50 0.52 / 1)	0.391	true
rgba(200,200,200,1)	color(display 0.78 0.78 0.78 / 1)	0.274	true
rgba(97,97,99,1)	color(display 0.39 0.39 0.39 / 1)	1.474	true
rgba(0,0,0,1)	color(display 0 0 0 / 1)	0.000	true
rgba(255,255,255,1)	color(display 1 1 1 / 1)	0.015	false
rgba(84,64,135,1)	color(display 0.33 0.25 0.53 / 1)	1.326	true
rgba(131,103,201,1)	color(display 0.51 0.40 0.78 / 1)	1.348	true
rgba(68,185,208,1)	color(display 0.27 0.75 0.82 / 1)	5.591	false
rgb(255,0,72)	color(display 1 0 0.2823 / 1)	3.529	false
rgba(255,205,63,1)	color(display 1 0.80 0.25 / 1)	3.880	false
rgba(241,174,50,1)	color(display 0.95 0.68 0.17 / 1)	4.701	false
rgba(245,181,40,1)	color(display 0.96 0.71 0.16 / 1)	4.218	false
rgb(147, 83, 255)	color(display 0.58 0.33 1 / 1)	2.143	false
rgba(75,3,161,1)	color(display 0.29 0.01 0.63 / 1)	1.321	false
rgba(255,0,0,0.85)	color(display 1 0 0 / 0.85)	7.115	false
rgba(84,64,135,1)	color(display 0.33 0.25 0.53 / 1)	1.326	true
rgba(131,103,201,1)	color(display 0.51 0.40 0.78 / 1)	1.348	true
rgba(68,185,208,1)	color(display 0.27 0.75 0.82 / 1)	5.591	false
#6d3bff	color(display .427 .231 1)	1.584	false
#03d658	color(display .012 .839 .345)	4.958	false
#ff3900	color(display 1 .224 0)	7.140	false
#7cf8b3	color(display .486 .973 .702)	4.284	true
#f8f8f8	color(display .973 .973 .973)	0.028	true
#e3f5fd	color(display .875 .945 .976)	1.918	true
#e74832	color(display .905882353 .282352941 .196078431 / 1)	3.681	true

図1.29. この表は、フォールバックのsRGB色を表示した後、ディスプレイp3色を表示したものです。1の色差(ΔE2000)はかろうじて見えますが、5ははっきりと区別できます。これは要約表です(表全体を参照[51])。

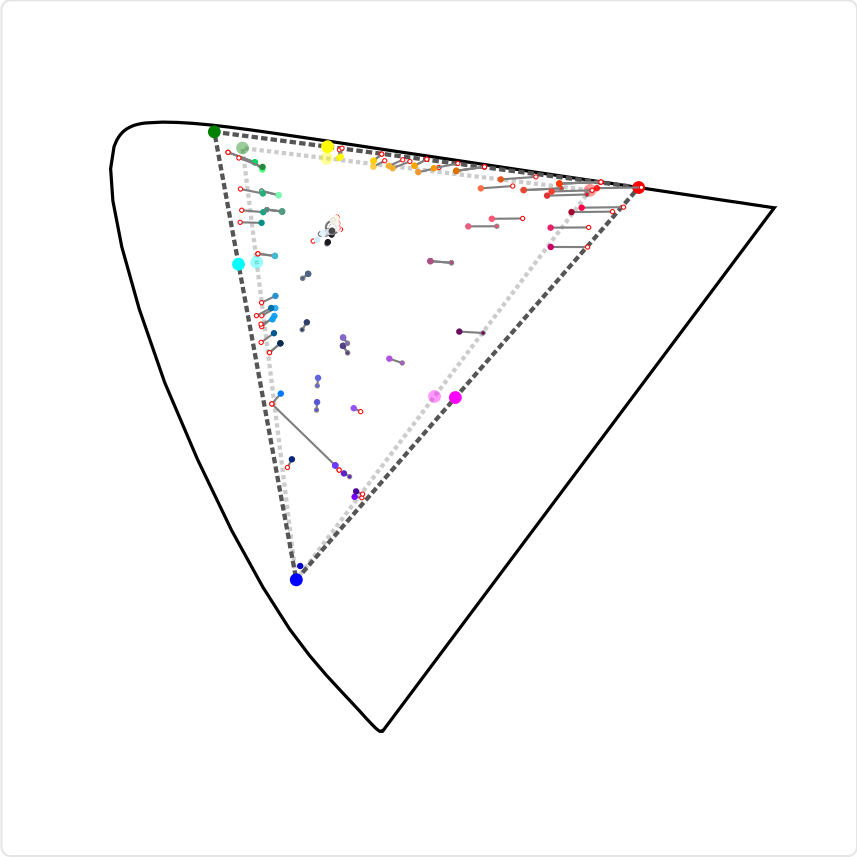


図1.30. 指定されたdisplay-p3色のUV色度とそのフォールバック。

sRGBとdisplay-p3では紫がかった色が似ていますが、これはおそらくどちらの色空間も同じ青の原色を使っているからでしょう。赤、オレンジ、黄色、緑の色は、sRGBの色域境界付近（ほぼ飽和状態）にあり、display-p3の色域境界付近の類似点にマッピングされています。

ウェブが未だにsRGBの地に囚われている理由は2つあるようです。1つ目はツールの不足、良いカラーピッカーの不足、もっと鮮やかな色があることへの理解不足です。しかし、私たちが考える大きな理由は、現在までにsRGBを実装しているブラウザはSafariだけだということです。これは急速に変化していて、ChromeもFirefoxも現在実装していますが、そのサポ

51. https://docs.google.com/spreadsheets/d/1sMWXWjMujqfAREYxNbG_t1fOJKYCA6ASLwtz4pBQVTw/#gid=264429000

ートが出荷されるまではおそらくdisplay-p3を使用するのは、閲覧者の17%しか見ない[52]ので、あまりにも少ない利益のために努力しすぎているのではないでしょうか。ほとんどの人はフォールバックを見るでしょう。だから現在の使用法は、大きな違いというよりは、色の鮮やかさの微妙なシフトです。

display-p3カラー（他にもオプションはあるが、野生で見つけたのはこれだけ）の使い方が今後1、2年でどのように変化していくのかを見てみるのも面白いだろう。

なぜなら、*wide color gamut* (WCG)は始まりに過ぎないからです。テレビや映画業界では、すでにP3を超えて、さらに広い色域へと移行しています*Rec.2020*。*ハイダイナミックレンジ* (HDR)は、特にゲームやストリーミングテレビ、映画などの家庭ではすでに登場しています。ウェブ上では、まだまだ追いつけない部分があります。

グラデーション

ミニマリズムとフラットデザインが流行しているにもかかわらず、CSSのグラデーションは75%のページで使用されています。予想通り、ほぼすべてのグラデーションが背景で使用されています。74.45%のページでは背景にグラデーションを指定していますが、その他のプロパティで指定しているのはわずか7%です。

線形グラデーションは放射状グラデーションの5倍の人気があり、ほぼ73%のページに表示されているのに対し、放射状グラデーションは15%です。この人気の差は驚異的で、これまで必要とされていなかった `-ms-linear-gradient()` でさえも、 `radial-gradient()` よりも人気があるのです（Internet Explorer 10は `-ms-` 接頭辞を付けても付けなくてもグラデーションをサポートしていました）。新しくサポートされた[53]の `conic-gradient()` はさらに利用率が低く、デスクトップページでは652ページ(0.01%)、モバイルページでは848ページ(0.01%)にしか表示されていません。

すべてのタイプの繰り返しグラデーションの利用率もかなり低く、 `repeating-linear-gradient()` が登場するページはわずか3%に過ぎず、他のタイプのグラデーションの利用率はさらに低くなっています（ `repeating-conic-gradient()` は21ページでしか利用されていません！）。

接頭辞付きグラデーションも、2013年以降、グラデーションに接頭辞が必要とされなくなったにもかかわらず、いまだに非常によく使われています。注目すべきは、-webkit-gradient()が2011年以降必要とされなくなった[54]にもかかわらず、すべてのウェブサイトの半数でいまだに使用されていることです。また、 `-webkit-linear-gradient()` はグラデーション関数の中で2番目によく使われており、57%のウェブサイトで使われています。

52. *https://gs.statcounter.com/browser-market-share*
53. *https://caniuse.com/css-conic-gradients*
54. *https://caniuse.com/css-gradients*

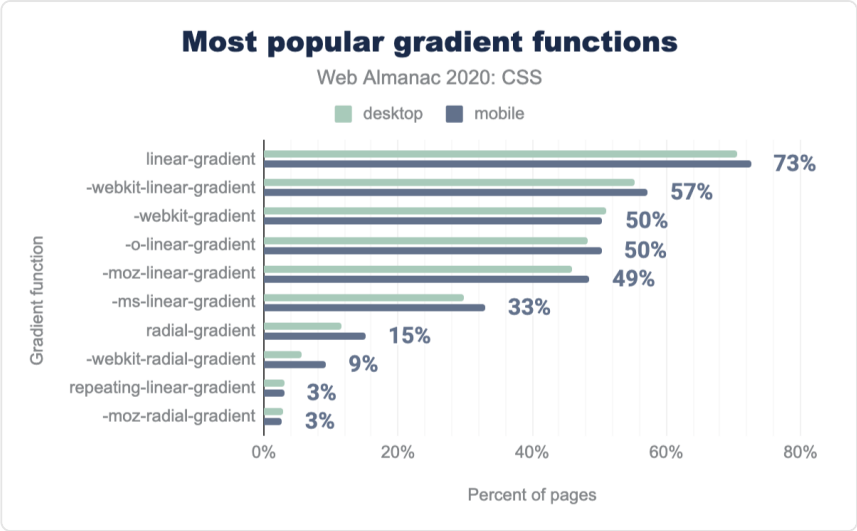


図1.31. ページのパーセンテージとして最も人気のあるグラデーション機能です。

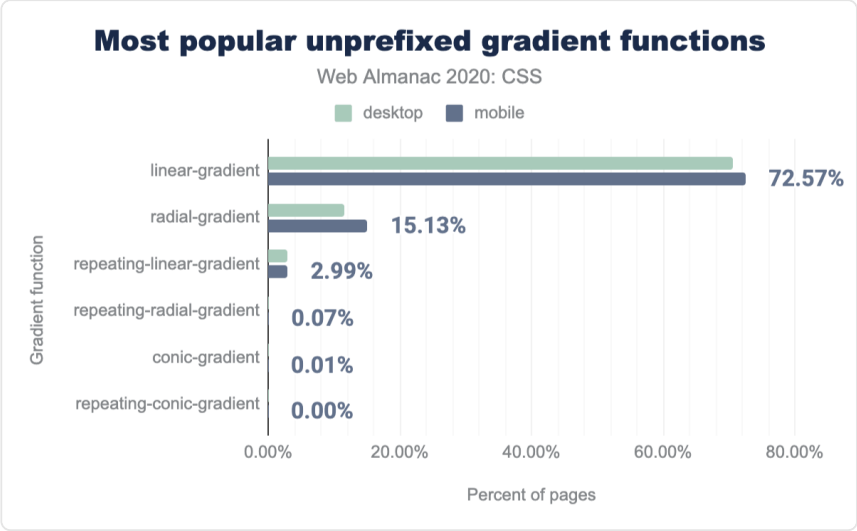


図1.32. 最も人気のあるグラデーションは、ベンダーの接頭辞を省略して、ページのパーセンテージとして機能します。

異なる色を同じ位置に配置したカラーストップ（ハードストップ）を使用してストライプやその他のパターンを作成することは、2010年にLea Verouによって初めて普及した[55]テクニッ

55. https://lea.verou.me/2010/12/checkered-stripes-other-background-patterns-with-css3-gradients/

クで、現在ではブレンドモードを備えた本当にクールなもの[56]など、多くの興味深いバリエーションがあります。ハックのように見えるかもしれませんが、ハードストップはページの50%に見られ、画像エディターや外部のSVGに頼らずにCSS内で軽量なグラフィックを作成したいという開発者の強いニーズがあることを示しています。

補間ヒント（または、この技術を普及させたAdobeが「中間点」と呼ぶ）は、2015年以降、ほぼ万国共通のブラウザのサポート[57]にもかかわらず、ページの22%にしか見られません。これは残念なことです。これがないと、カラーストップは滑らかな曲線ではなく、色空間の直線で結ばれてしまうからです。この使用率の低さは、それらが何をするのか、あるいはどのように使うのかについての誤解を反映していると思われます。これとは対照的に、CSSトランジションやアニメーションと比較すると、イージング関数（キーフレームをジャラジャラした直線ではなく曲線で接続するなど、ほとんど同じことをする）の方がはるかに一般的に使用されています（トランジションの80%）。”Midpoints”というのはあまりわかりやすい記述ではありませんし、”interpolation hint”というのは、ブラウザが簡単な算術をするのを手伝っているように聞こえます。

ほとんどのグラデーションの使用法は単純で、データセット全体の75%以上のグラデーションが2色のストップを使用しているだけです。実際、半分以下のページでは、3色以上のカラーストップを使ったグラデーションが1つでも含まれています。

一番色が止まっているグラデーションはこちら[58]で646色！　可愛いですね。これはほぼ確実に生成されたもので、結果として得られるCSSコードは8KBなので、1pxの高さのPNGであればフットプリントが小さくても問題ないでしょう（下の画像は1.1KBです）。

図1.33. 最も色が止まっているグラデーション、646。

レイアウト

CSSには現在、レイアウトオプションが多数用意されており、テーブルをレイアウトに使用しなければならなかったテーブルとは一線を画しています。Flexbox、Grid、Multiple-columnレイアウトはほとんどのブラウザでサポートされています。

56. *https://bennettfeely.com/gradients/*
57. *https://caniuse.com/mdn-css_types_image_gradient_linear-gradient_interpolation_hints*
58. *https://dabblet.com/gist/4d1637d78c71ef2d8d37952fc6e90ff5*

Flexboxとグリッドの採用

2019年版[59]では、モバイルとデスクトップをまたいだページの41%がFlexbox[60]のプロパティを含むと報告されていました。2020年には、この数字はモバイルで63%、デスクトップで65%にまで拡大しています。Flexboxが実行可能なツールとなる前に開発されたレガシーサイトの数がまだ存在していることから、このレイアウト方法は広く採用されていると言えるでしょう。

グリッドレイアウト[61]を見てみると、グリッドレイアウトを利用しているサイトの割合は、モバイルでは4%、デスクトップでは5%にまで伸びています。昨年から倍増していますが、フレックスレイアウトにはまだ大きく遅れをとっています。

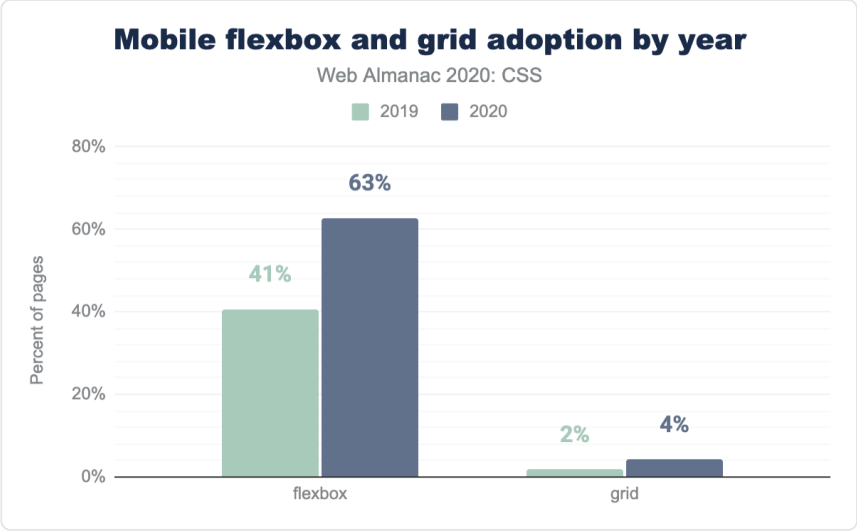


図1.34. 年ごとのFlexboxとグリッドの採用率をモバイルページの割合で表示しています。

59. https://almanac.httparchive.org/ja/2019/css#flexbox
60. https://developer.mozilla.org/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
61. https://developer.mozilla.org/docs/Web/CSS/CSS_Grid_Layout

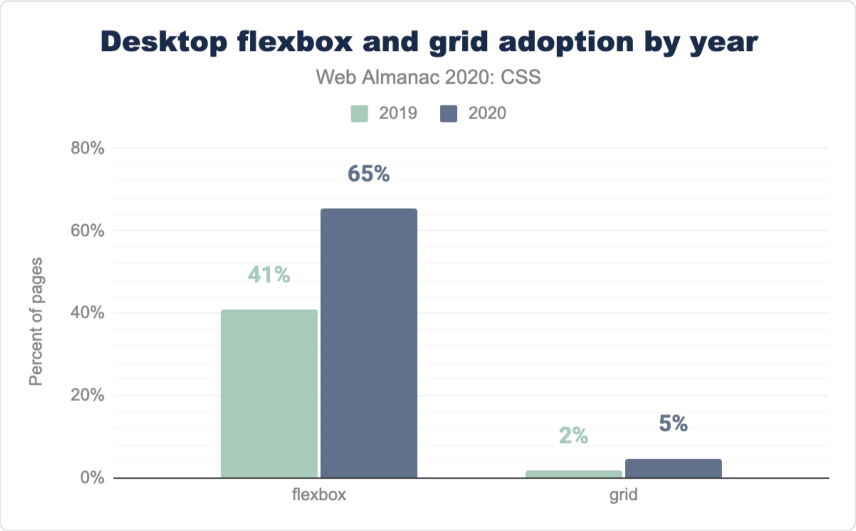


図1.35. 年ごとのflexboxとグリッドの採用率をデスクトップページの割合で示す。

この章の他のほとんどのメトリクスとは異なり、これは実際に測定されたグリッドの使用状況であり、スタイルシートで指定されていて使用されない可能性のあるグリッド関連のプロパティや値だけではないことに注意してください。一見するとこれがより正確に見えるかもしれませんが、心に留めておくべきことは、HTTP Archiveはホームページをクロールしているため、グリッドが内部ページに多く出現することが多いため、このデータはより低く偏っているかもしれないということです。

スタイルシートで`display: grid`と`display: flex`を指定しているページがどれだけあるか？　この指標ではグリッドレイアウトの採用率が著しく高く、30%のページが`display: grid`を一度は使用しています。しかし、Flexboxの採用率にはそれほど大きな影響はなく、68%のページが`display: flex`を指定しています。これはFlexboxの採用率が非常に高いように聞こえますが、80%のページでテーブル表示モードを使用しているCSSテーブルの方がはるかに人気があることは注目に値します。この使用法のいくつかは、ある種のclearfix[62]が`display: table`を使用していることに起因するもので、実際のレイアウトではありません。

62. https://css-tricks.com/snippets/css/clear-fix/

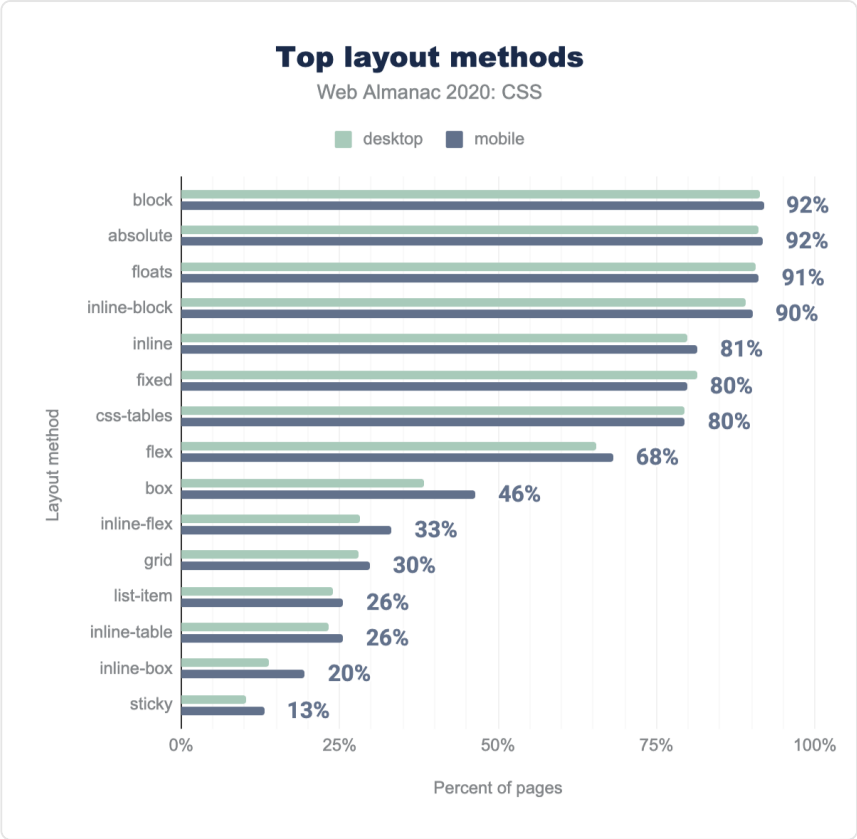


図1.36. レイアウトモードと、それらが表示されるページの割合。このデータは `display`, `position`, `float` プロパティの値を組み合わせたものです。

グリッドレイアウトよりも前のブラウザではFlexboxが使用可能であったことを考えると、グリッドシステムを設定するためにFlexboxを使用している可能性があります。グリッドとしてFlexboxを使うためには、Flexboxの固有の柔軟性の一部を無効にする必要があります。そのためには、`flex-grow` プロパティを `0` に設定し、パーセンテージを使ってフレックスアイテムのサイズを設定します。この情報を使って、デスクトップとモバイルの両方で19%のサイトがこのグリッドのような方法でFlexboxを使用していることを報告することができました。

GridではなくFlexboxを選択した理由は、GridレイアウトがInternet Explorerではサポートされていなかった[63]ことを考えると、ブラウザのサポートがよく挙げられます。さらに、著者の中にはまだグリッドレイアウトを学んでいなかったり、Flexboxベースのグリッドシステ

63. https://caniuse.com/css-grid

ムを使っているフレームワークを使っている人もいるでしょう。Bootstrap[64]フレームワークは現在Flexboxベースのグリッドを使用しており、他のいくつかの一般的なフレームワークの選択肢と共通しています。

異なるグリッドレイアウト技術の使用法

グリッド・レイアウト仕様では、CSSでレイアウトを記述し定義するための多くの方法を提供しています。最も基本的な使い方は、あるグリッド線から別のグリッド線へ[65]のように項目をレイアウトすることです。名前付き行[66]や`grid-template-areas`の使用はどうでしょうか？

名前付きの行については、トラックリストの中に角括弧があるかどうかをチェックしました。角括弧の内側に配置されている名前または名前。

```
.wrapper {
  display: grid;
  grid-template-columns: [main-start] 1fr [content-start] 1fr
[content-end] 1fr [main-end];
}
```

その結果、モバイルではグリッドを利用しているページの0.23%が名前付きラインを持っていたのに対し、デスクトップでは0.27%という結果が出ました。

グリッドテンプレートエリア[67]の機能は、グリッドアイテムに名前を付け、`grid-template-areas`プロパティの値としてグリッド上に配置することを可能にするもので、少し良い結果が得られました。グリッドを使用しているサイトのうち、モバイルでは19%、デスクトップでは20%がこの方法を使用していました。

これらの結果から、グリッドレイアウトの使用率は、プロダクションウェブサイトではまだ比較的低いだけでなく、その使用方法も比較的シンプルであることがわかります。著者は、行や領域に名前を付けられるような方法よりも、シンプルなラインベースの配置を選択しています。そうすることは何も悪いことではありませんが、グリッドレイアウトの採用が遅れているのは、作者がまだこの機能の威力を理解していないことが一因なのではないでしょうか。もしグリッドレイアウトがブラウザのサポートが不十分で本質的にFlexboxとみなされているとしたら、それは確かに説得力のある選択ではないでしょう。

64. https://getbootstrap.com/docs/4.5/layout/grid/
65. https://www.smashingmagazine.com/2020/01/understanding-css-grid-lines/
66. https://developer.mozilla.org/docs/Web/CSS/CSS_Grid_Layout/Layout_using_Named_Grid_Lines
67. https://developer.mozilla.org/docs/Web/CSS/CSS_Grid_Layout/Grid_Template_Areas

複数カラムレイアウト

マルチカラムレイアウト[68]、または*multicol*とは、新聞のようにコンテンツを列に並べることができる仕様のことです。印刷用のCSSではよく使われていますが、Web上では読者がコンテンツを読むために上下にスクロールしなければならないような状況が発生するリスクがあるため、あまり有用ではありません。しかし、データによると、デスクトップでは15.33%、モバイルでは14.95%と、グリッドレイアウトよりもmulticolを使用しているページがかなり多くなっています。基本的なmulticolプロパティは十分にサポートされていますが、より複雑な使い方や断片化[69]での改行制御はざっくばらんなサポート[70]となっています。このようなことを考えると、これだけの使用法があるというのはかなり驚きでした。

ボックスのサイジング

ページ上のボックスの大きさを知っておくと便利ですが、標準のCSSボックスモデル[71]では、コンテンツボックスのサイズにpaddingとborderを追加しているため、ボックスに与えたサイズはページ上でレンダリングされるボックスよりも小さくなってしまいます。履歴を変更することはできませんが、`box-sizing`プロパティで指定したサイズを`border-box`に適用するように切り替えることができるので、設定したサイズがレンダリングされるサイズになります。どのくらいのサイトが`box-sizing`プロパティを使っているのでしょうか？ほとんどのサイトが使っています。box-sizing'プロパティは、デスクトップCSSの83.79%、モバイルでは86.39%のサイトで利用されています。

68. *https://developer.mozilla.org/docs/Web/CSS/CSS_Columns/Basic_Concepts_of_Multicol*
69. *https://www.smashingmagazine.com/2019/02/css-fragmentation/*
70. *https://caniuse.com/multicolumn*
71. *https://developer.mozilla.org/docs/Learn/CSS/Building_blocks/The_box_model#what_is_the_css_box_model*

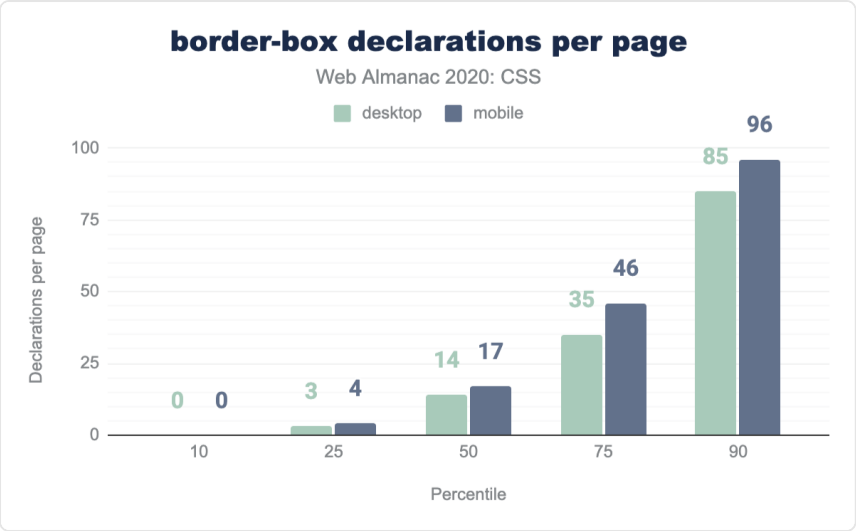


図1.37. 1ページあたりの`border-box`宣言の数の分布。

デスクトップページの中央値には14件の`box-sizing`宣言があります。モバイルページには17個あります。おそらく、スタイルシート内のすべての要素に対してグローバルなルールとしてではなく、コンポーネントシステムがコンポーネントごとに宣言を挿入しているためでしょう。

トランジションとアニメーション

トランジションとアニメーションは全体的に非常に人気があり、`transition`プロパティは全ページの81%で、`animation`プロパティはモバイルページの73%、デスクトップページの70%で使用されている。モバイルページでは、バッテリーの節約[72]を優先しているはずなのに、モバイルページでの使用率が低くないのはやや驚きです。一方で、JSアニメーションよりもCSSアニメーションの方がはるかにバッテリー効率が良く、特に変形と不透明度をアニメーションさせるだけのものが大半を占めています（次項参照）。

最も一般的に指定されている遷移プロパティは`all`で、41%のページで使用されています。`all`は初期値なので、実際には明示的に指定する必要がないので、これは少し不可解です。次いで、フェードイン/フェードアウトのトランジションが最も一般的なタイプで、クロールされたページの3分の1以上で使用されており、`transform`プロパティのトランジション（スピン、スケール、動きのトランジションが多い）が続いている。驚くべきことに、`height`のトランジションは`max-height`のトランジションよりもはるかに人気があります

72. *https://css-tricks.com/how-web-content-can-affect-power-usage/*

が、後者は開始または終了の高さが不明な場合の回避策として一般的に教えられています（自動）。また、`scale`プロパティはFirefox以外ではサポートされていないにもかかわらず、かなりの使用率があることも驚きでした(2%)。最先端のCSSを意図的に使っているのか、タイプミスなのか、それとも変形をアニメーションさせる方法についての誤解なのか。

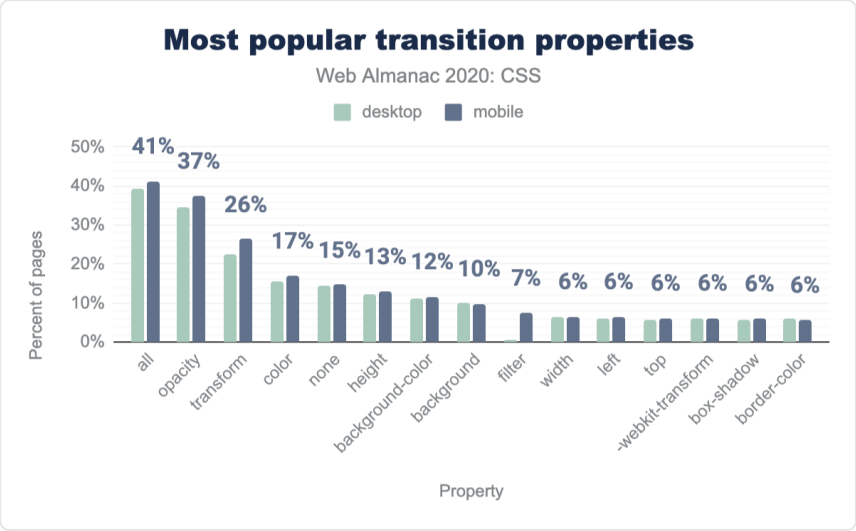


図1.38. ページの割合としてトランジションプロパティを採用しています。

嬉しいことに、これらのトランジションのほとんどはかなり短く、トランジションの持続時間の中央値はわずか300msで、90%のWebサイトの持続時間の中央値は半秒以下であることがわかりました。トランジションが長いとUIが重く感じてしまいますが、短いトランジションは邪魔にならず変化を伝えることができるので、これは一般的に良い方法です。

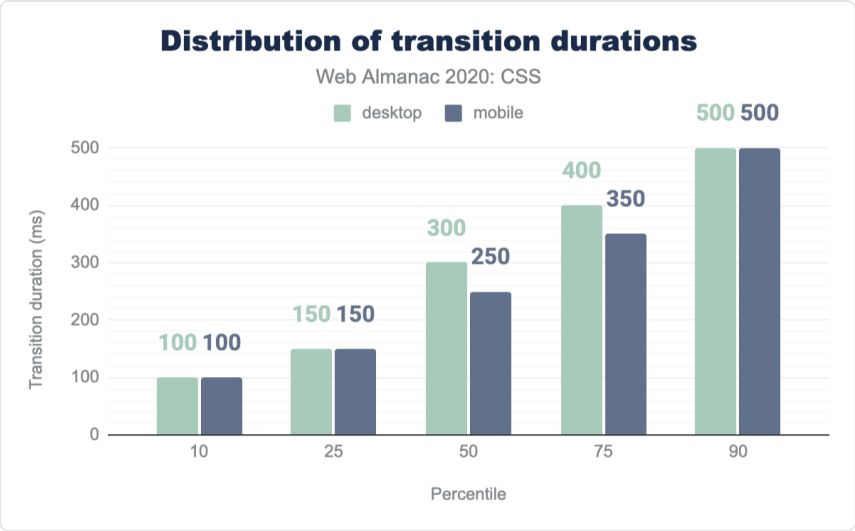


図1.39. 移行期間の分布。

仕様書の作者はそれを正しく理解している！ `Ease`は最もよく使われるタイミング関数であるが、それはデフォルトであるため、実際には省略することも可能です。おそらくデフォルトを明示的に指定するのは自己文書化された冗長性を好むからなのか、あるいは、おそらくもっとありそうなことだが、デフォルトであることを知らないからなのか、どちらかだろう。直線的にアニメーションを進行させるという欠点があるにもかかわらず、`linear`は19.1%と、2番目に多く使われているタイミング関数です。また、組み込みのイージング関数がすべての遷移の87%以上に対応していることも興味深い。

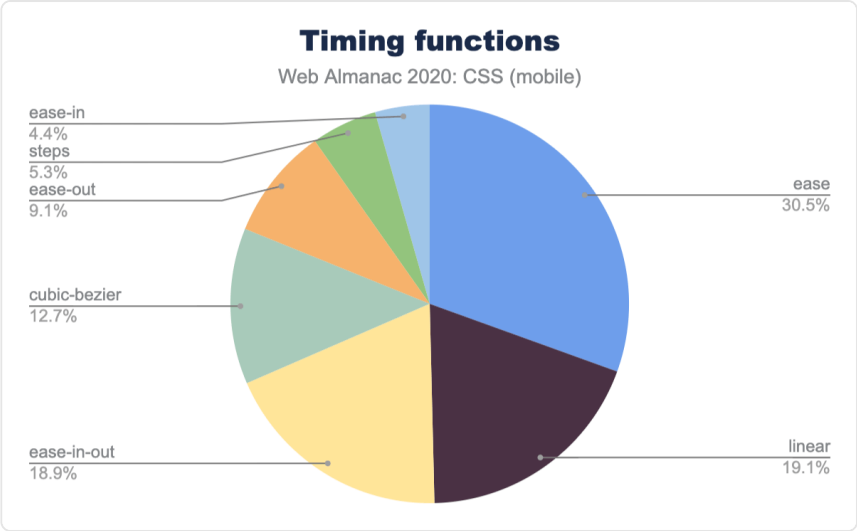


図1.40. タイミング機能の相対的な人気度は、モバイルページでの出現率として示されています。

アニメーションを採用した主な要因は、Font Awesomeであると考えられています。アニメーション名には様々な種類がありますが、その多くは基本的なカテゴリーに分類されているようで、5つに1つは「スピン」のようなものが含まれています。滑らかな永久回転を求めるのであれば、「直線的な」トランジションやアニメーションの割合が高いのもそのためかもしれません。

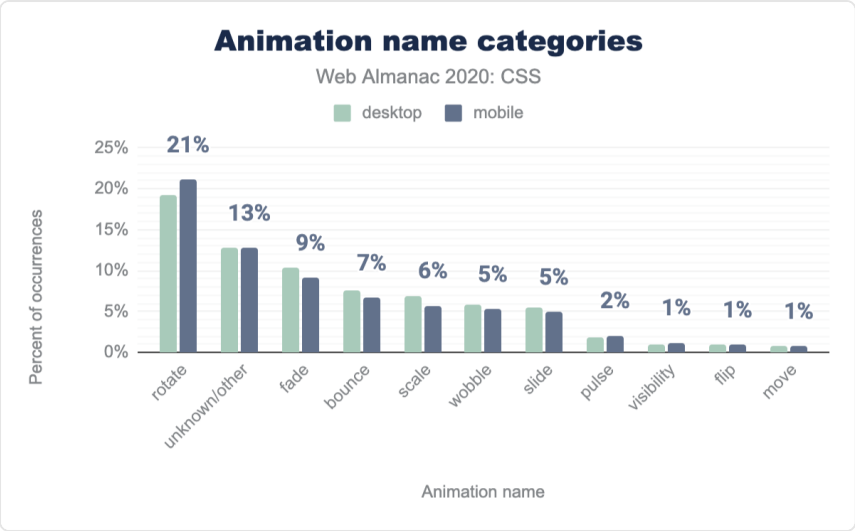


図1.41. 使用されているアニメ名のカテゴリの相対的な人気度を、出現率として表示しています。

視覚効果

CSSはまた、デザイナーに少量のコードでアクセスできるブラウザに組み込まれた高度なデザイン技術へのアクセスを可能にする、非常に多様な視覚効果を提供しています。

ブレンドモード

昨年は、8%のページでブレンドモードを使用していました。今年は採用率が大幅に上昇し、13%のページで要素にブレンドモードを使用しており(`mix-blend-mode`)、2%のページで背景にブレンドモードを使用しています(`background-blend-mode`)。

フィルター

フィルターの採用率は依然として高く、`filter`プロパティは79.43%のページに登場しています。最初これは非常に刺激的でしたが、その多くは同じプロパティ名を共有していた古いIE DX filters(`-ms-filter`)である可能性が高いです。Blinkが認識する有効なCSSフィルターのみを考慮した場合、使用率はモバイルで22%、デスクトップで20%にまで低下し、`blur()`が最も人気のあるフィルタタイプで、4%のページに表示されています。

これは、半透明の背景のコントラストを改善したり、多くのネイティブUIでおなじみのエレガントな“すりガラス”効果[73]を作り出すのに非常に便利です。 `フィルター` ほど人気があるわけではありませんが、ページの6%で `backdrop-filter` が見つかりました。

`filter()` 関数を使うと、特定の画像にのみフィルタを適用することができ、背景に非常に便利です。残念ながら、これは現在のところSafariでのみサポートされている[74]。 `filter()` の使用法は見つからなかった。

マスク

10年前、Safariで `-webkit-mask-image` でマスクを取得して盛り上がっていました。みんなとその犬が使っていました。最終的には 仕様[75] と、WebKitプロトタイプに近いモデルの接頭辞なしプロパティのセットを手に入れることができ、マスクが標準化され、すべてのブラウザで一貫した構文を持つようになるのは時間の問題だと思われました。10年後の今、接頭辞なしの構文はChromeやSafariではまだサポートされておらず、世界の5%以下のユーザーのブラウザで利用できることを意味します[76]。したがって、 `webkit-mask-image` が標準のものよりもまだ人気があり、22%のページで見つかっているのも不思議ではありません。しかし、サポートが非常に悪いにもかかわらず、 `mask-image` は19%のページで見つかりました。他のほとんどのマスキングプロパティでも同様のパターンが見られ、接頭辞なしのバージョンは `webkit-` とほぼ同じくらいの数のページに表示されています。全体的に見ると、マスクは誇大広告から外れているにもかかわらず、ウェブの4分の1近くでまだ見つかっており、実装者の関心が低いにもかかわらず、ユースケースがまだ存在していることを示しています（ヒント、ヒント！）。

73. *https://css-tricks.com/backdrop-filter-effect-with-css/*
74. *https://caniuse.com/css-filter-function*
75. *https://www.w3.org/TR/css-masking-1/*
76. *https://caniuse.com/css-masks*

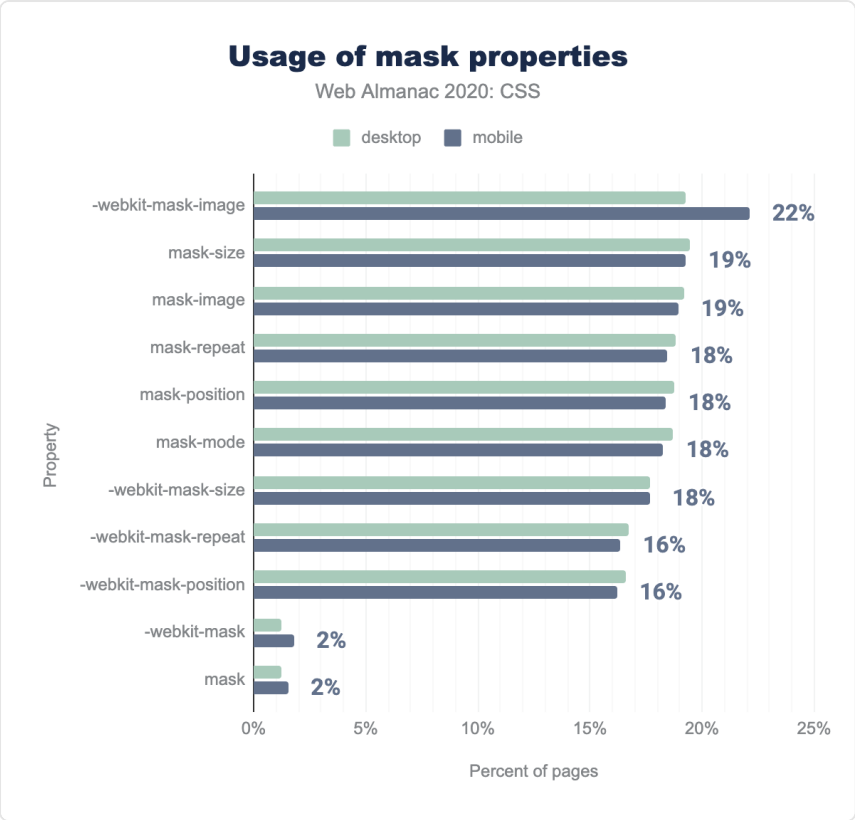


図1.42. マスクのプロパティの相対的な人気度を出現率で表す。

クリッピングパス

マスクが人気になったのと同じ頃、似たような、しかしよりシンプルなプロパティ（元々はSVGのもの）が登場し始めました。それは`clip-path`です。しかし、マスクとは異なり、このプロパティには明るい運命がありました。このプロパティはかなり早く標準化され、比較的早くボード全体でサポートされるようになりましたが、最後に残ったのは2016年にプレフィックスを削除したSafariでした。今日では、接頭辞なしのページの19%、接頭辞`-webkit-`のページの13%で見られるようになりました。

レスポンシブデザイン

FlexboxやGridのような柔軟で応答性の高い新しいレイアウト手法が組み込まれていること

で、ウェブを閲覧する多くの異なる画面サイズやデバイスに対応できるサイトを作ることが幾分容易になりました。これらのレイアウト方法は、通常、メディアクエリを使用することでさらに強化されています。データによると、デスクトップサイトの80%、モバイルサイトの83%が、`min-width`のようなレスポンシブデザインに関連したメディアクエリを使用していることがわかります。

人々はどのメディア機能を利用しているのか？

ご想像の通り、最も一般的に使用されているメディア機能は、レスポンシブウェブデザインの初期の頃から使用されているビューポートサイズ機能です。`max-width`をチェックしているサイトの割合は、デスクトップとモバイルの両方で78%です。モバイルサイトの75%、デスクトップサイトの73%で`min-width`機能をチェックしています。

画面が縦か横かに基づいてレイアウトを差別化できる`orientation`メディア機能は、全サイトの33%のサイトで利用されています。

統計の中で、いくつかの新しいメディア機能が出てきています。`prefers-reduced-motion`メディア機能は、ユーザーがモーションの縮小を要求したかどうかをチェックする方法を提供します。この機能は、ユーザーが制御するオペレーティングシステムの設定で明示的にオンにすることもできますし、バッテリー残量が減っているなどの理由で暗黙的にオンにすることもできます。24%のサイトがこの機能をチェックしています。

その他の良いニュースとしては、Media Queries Level 4[77]仕様の新機能が登場し始めています。モバイルでは、5%のサイトがユーザーが持っているポインターの種類をチェックしています。`coarse`ポインターはタッチスクリーンを使用していることを示し、`fine`ポインターはポインティングデバイスを示しています。ユーザーがどのようにサイトを操作しているのかを理解することは、画面サイズを見るよりも役立つことが多いですが、それ以上に役立つこともあります。人は、キーボードとマウスを使用して小さな画面のデバイスを使用しているかもしれませんし、タッチスクリーンを使用して高解像度の大画面デバイスを使用しているかもしれませんし、より大きなヒットエリアの恩恵を受けているかもしれません。

77. https://www.w3.org/TR/mediaqueries-4/

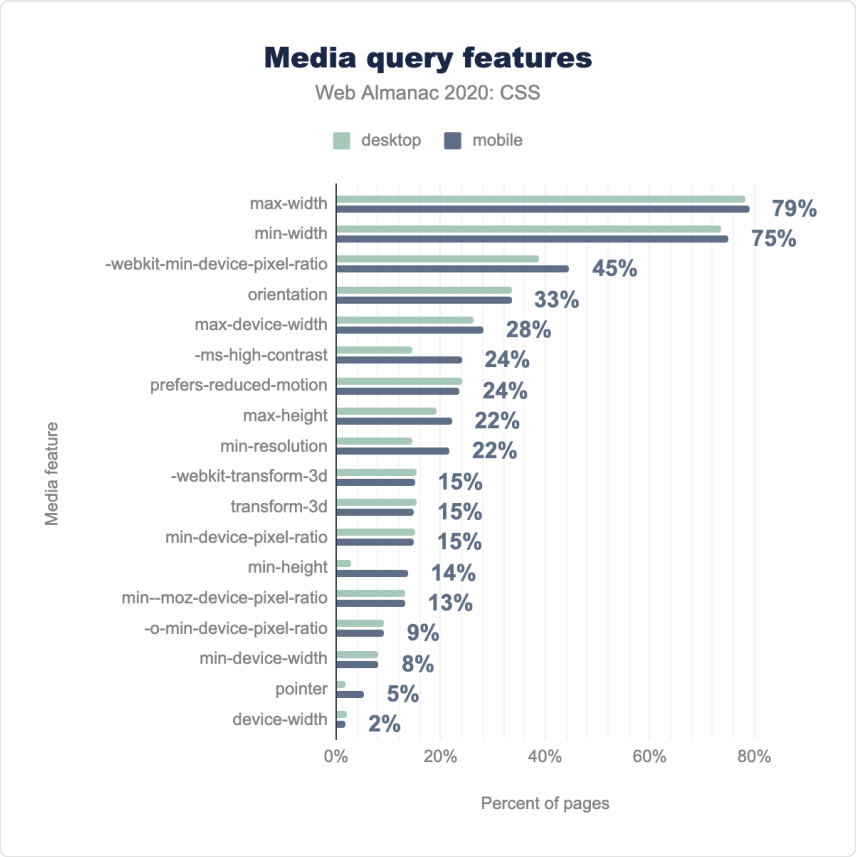


図1.43. 最も人気のあるメディアクエリの特徴は、ページのパーセンテージです。

一般的なブレークポイント

デスクトップとモバイルデバイスで使用されている最も一般的なブレークポイントは768pxの`min-width`です。54%のサイトがこのブレークポイントを使用しており、次いで`max-width`の767pxが50%と密接に続いています。Bootstrap framework[78]は768pxの最小幅を「Medium」のサイズとして使用しているので、これが多くの使用量の源になっているのかもしれません。他にも、1200px(40%)と992px(37%)という2つのよく使われる`min-width`値もBootstrapにはあります。

78. https://getbootstrap.com/docs/4.1/layout/overview/

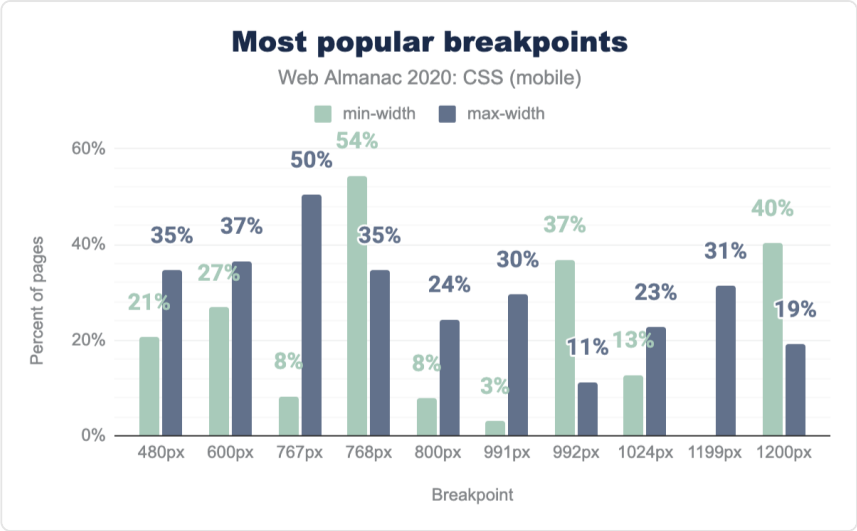


図1.44. モバイルページに占める割合として、`min-width`と`max-width`による最も人気のあるブレークポイントを示します。

ピクセルはブレークポイントに使用される単位です。`em`はリストの中ではかなり下の方にありますが、ピクセルでブレークポイントを設定するのが一般的な選択のようです。これには多くの理由があると思われます。レガシー: レスポンシブデザインに関する初期の記事はすべてピクセルを使用しており、多くの人がレスポンシブデザインを作成する際に特定のデバイスをターゲットにすることを今でも考えています。サイジング。これはウェブデザインについての新しい考え方で、おそらくレイアウトのための本質的なサイジング方法とともに、まだ十分に活用されていないものです。

メディアクエリ内で使用されるプロパティ

モバイルデバイスでは79%、デスクトップでは77%のメディアクエリが`display`プロパティを変更するために使用されています。おそらく、人々はFlexやグリッドフォーマットのコンテキストへ切り替える前にテストをしていることを示しています。繰り返しになりますが、これはリンクされたフレームワーク、例えばBootstrap responsive utilities[79]のようなものかもしれません。78%の作者がメディアクエリの内部で`width`プロパティを変更しており、`margin`、`padding`、および`font-size`はすべて変更されたプロパティの上位にランクされています。

79. https://getbootstrap.com/docs/4.1/utilities/display/

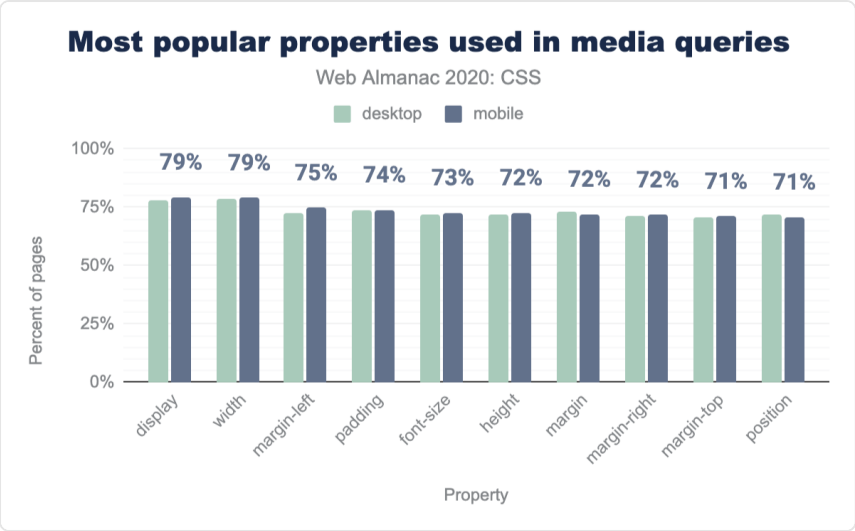


図1.45. メディアクエリで使用されている最も人気のあるプロパティをページのパーセンテージで表示しています。

カスタムプロパティ

昨年、カスタムプロパティを採用しているウェブサイトはわずか5%でした。今年は採用率が急上昇しています。昨年のクエリ（カスタムプロパティを設定する宣言のみをカウントしていた）を使用すると、モバイルでは4倍（19.29%）、デスクトップでは3倍（14.47%）になりました。しかし、`var()`を介してカスタムプロパティを参照している値を見ると、さらに良い結果が得られます。モバイルページの27%、デスクトップページの22%が、少なくとも一度は`var()`関数を使用していました。

一見すると、これは印象的な採用率ですが、WordPressが主な推進力となっているように見えます。

ネーミング

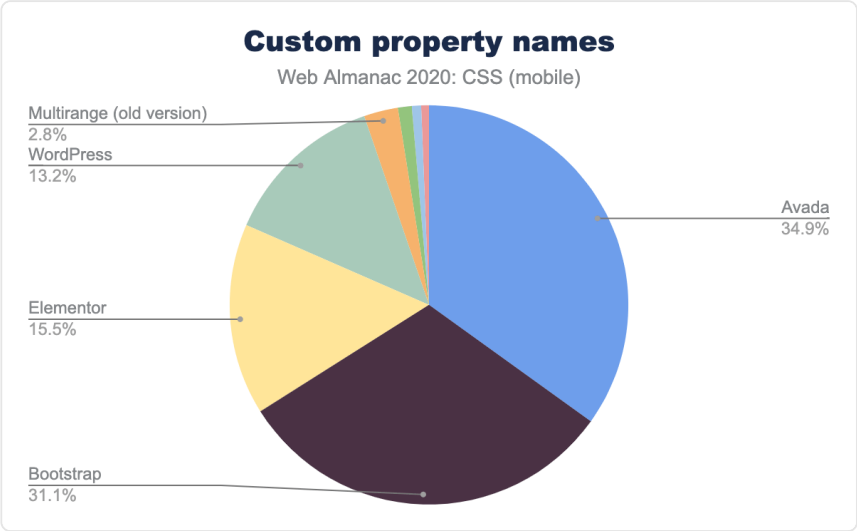


図1.46. ソフトウェアエンティティごとのカスタムプロパティ名の相対的な人気度を、モバイルページでの出現率として示しています。

上位1,000件のプロパティ名のうち、個人のウェブ開発者によって作られた「カスタム」は13件にも満たない。その大半は、WordPress、Elementor、Avadaなどの人気のあるソフトウェアに関連したものです。これを判断するために、どのカスタムプロパティがどのソフトウェアに登場するのか（GitHubでの検索）だけでなく、どのプロパティが似たような頻度でグループに登場するのかを考慮しました。これは、カスタムプロパティがウェブサイト上で使われる主な方法がそのソフトウェアの使用によるものだということを必ずしも意味するものではありません（人々は今でもコピー＆ペーストをしています！）が、開発者が定義したカスタムプロパティ間にはあまり有機的な共通点がないことを示しています。トップ1000のリストに有機的に入っていると思われるカスタムプロパティの名前は、`--height`、`--primary-color`、そして`--caption-color`だけです。

タイプ別利用法

カスタムプロパティの最大の使用法は、色の命名と、全体的に一貫した色を維持することであるようです。デスクトップページの約5人に1人、モバイルページの約6人に1人が`background-color`でカスタムプロパティを使用しており、`var()`参照を含む上位11のプロパティは、カラープロパティか、色を含む略語のいずれかです。長さは2番目に大きく、`width`と`height`が`var()`と一緒に使われているのはモバイルページの7%です（興味深いことに、デスクトップページでは3%程度しか使われていない）。このことは、最も人気の

ある値の型でも確認されており、カスタムプロパティ宣言の52%をカラー値が占めています。寸法（数値＋単位（長さ、角度、時間など））は、2番目に人気のあるタイプで、単位のない数値（12%）よりも高くなっています。なぜなら、数値は`calc()`と乗算で次元に変換できますが、次元での除算はまだサポートされていないため、次元を数値に変換することはできません。

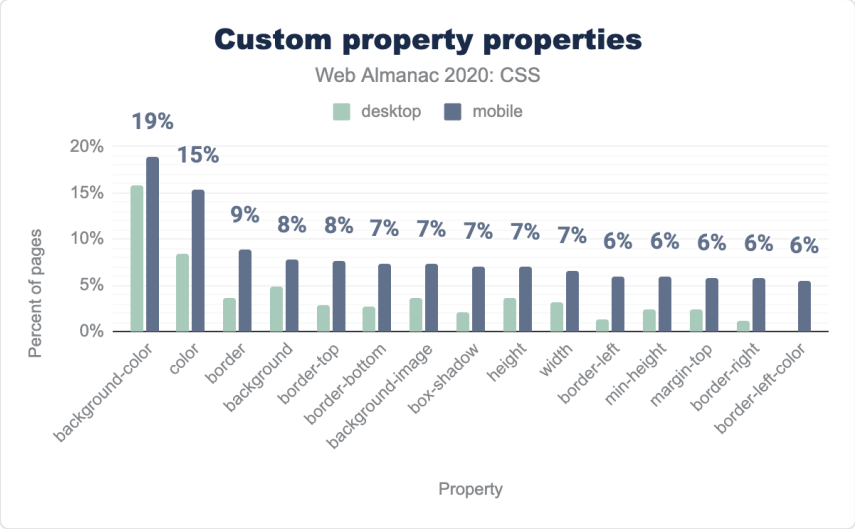


図1.47. カスタムプロパティで使用されているプロパティ名をページのパーセンテージとして最もよく使用されています。

プリプロセッサでは、色変数を操作して、異なる色合いなどの色のバリエーションを生成することがよくあります。しかし、CSSでは色変更関数[80]は未実装の草案に過ぎません。今のところ、変数から新しい色を生成する唯一の方法は、個々のコンポーネントに変数を使用し、`rgba()`や`hsla()`のようなカラー関数にプラグインすることです。しかしモバイルページの4%未満、デスクトップページの0.6%未満がこれを行っており、カラー変数の使用率が高いのは主に色全体を保持するためであり動的に生成されるのではなく、そのバリエーションを別の変数にしていることを示しています。

80. https://drafts.csswg.org/css-color-5/

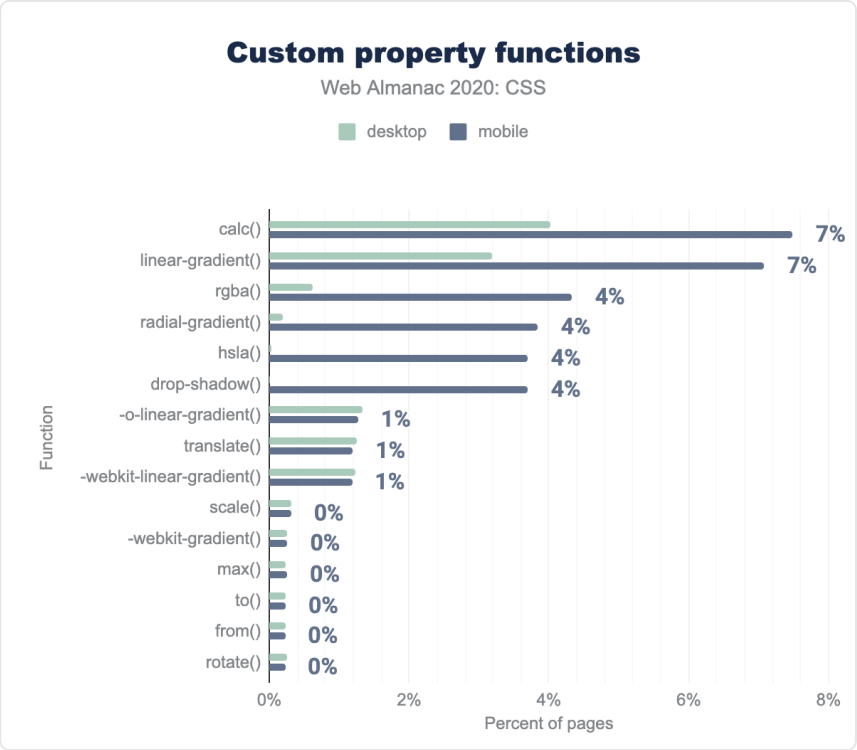


図1.48. カスタムプロパティで使用される最も一般的な関数名をページのパーセンテージで表示します。

複雑さ

次に、カスタムプロパティの使い方がどれだけ複雑かを見てみました。ソフトウェアエンジニアリングにおけるコードの複雑さを評価する1つの方法として、依存関係グラフの形状があります。まず、各カスタムプロパティの *depth* を調べました。例えば `#fff` のようなリテラル値に設定されたカスタムプロパティの深さは0ですが、var()を介してそれを参照するプロパティの深さは1などとなります。例えば、以下のようになります。

```
: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 */
}
```

調査した3つのカスタムプロパティのうち2つ（67%）は深さが0で、30%は深さが1でした（モバイルではやや少なかった）。深さが2のプロパティは1.8%未満で、深さが3以上のプロパティはほとんどなく（0.7%）、かなり基本的な使い方であることを示しています。このような基本的な使い方の利点は、間違いを犯しにくいということです。

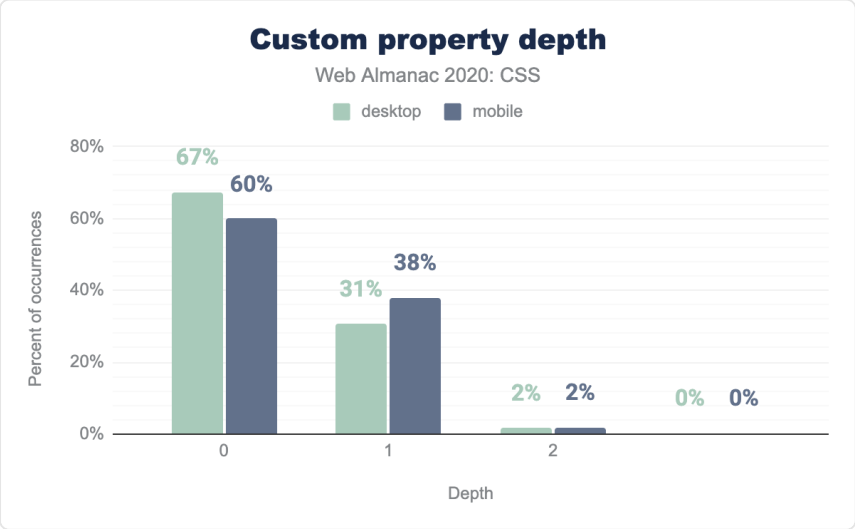


図1.49. カスタムプロパティの深さの分布を出現率で表したもの。

カスタム・プロパティが宣言されているセレクターを調べてみると、ほとんどのカスタム・プロパティの使用法はかなり基本的なものであることがわかります。3つのカスタムプロパティ宣言のうち2つはルート要素上で宣言されており、基本的にグローバル定数として使用されていることがわかります。多くの一般的なポリフィルでは、このようにグローバルであることが要求されているため、そのようなポリフィルを使用している開発者は選択の余地がなかったのかもしれません。

CSSとJS

ここ数年、CSSのクラスやスタイルを単純に設定したり、オフにしたりするだけではなく、CSSとJavaScriptの相互作用が大きくなってきています。では、Houdiniのような技術やCSS-in-JSのようなテクニックをどれだけ使っているのでしょうか？

Houdini

Houdini[81]という言葉を聞いたことがあるかもしれません。HoudiniはCSSエンジンの一部を公開する低レベルAPIのセットで、ブラウザのレンダリングエンジンのスタイリングやレイアウトプロセスにフックすることでCSSを拡張する力を開発者に与えます。いくつかのHoudiniの仕様がブラウザで出荷されている[82]ので、実際に使用されているかどうかを確認する時が来たと考えました。短い答え: いいえ。そして今、長い答えのために。。。

まず、Properties & Values API[83]を見てみました。これは開発者がカスタムプロパティを登録して、型や初期値を与え、継承を防ぐことができるというものです。主なユースケースの1つはカスタムプロパティをアニメーションさせることができることなので、カスタムプロパティがアニメーションされる頻度も調べてみました。

最先端の技術によくあるように、特にすべてのブラウザでサポートされていない場合は、野生での採用は非常に低くなっています。登録されているカスタムプロパティがあるのはデスクトップページが32ページ、モバイルページが20ページのみでしたが、これには登録されているがクロール時には適用されていないカスタムプロパティは含まれていません。アニメーションでカスタムプロパティを使用しているのは、325のモバイルページと330のデスクトップページ（0.00%）のみで、そのほとんど（74%）はVueコンポーネント[84]によって駆動されているようです。これは、クロール時にアニメーションがアクティブになっていなかったため、スタイルを登録する必要のない計算されたスタイルが存在しなかったためと考えられます。

Paint API[85]は、より広く実装されたHoudiniの仕様で、開発者はカスタムグラデーションやパターンを実装するなど、`<image>`の値を返すカスタムCSS関数を作成することができます。12ページだけが`paint()`を使用していることがわかりました。各ワークレット名（`hexagon`, `ruler`, `lozenge`, `image-cross`, `grid`, `dashed-line`, `ripple`）はそれぞれ1ページに1つしか表示されませんでした。

Typed OM[86]は、別のHoudini仕様で、古典的なCSS OMの文字列の代わりに構造化された値へのアクセスを可能にします。他のHoudini仕様に比べてかなり高い採用率を持っているようですが、全体的にはまだ低いです。デスクトップページでは9,864件（0.18%）、モバイルページ6,391件（0.1%）で使用されています。これは低いように見えるかもしれませんが、視点を変えれば、これらの数字は`<input type="date">`の採用と同じようなものです! この章のほとんどの統計とは異なり、これらの数字は実際の利用状況を反映しており、ウェブサイトの資産に含まれているだけではないことに注意してください。

81. *https://developer.mozilla.org/docs/Web/Houdini*
82. *https://ishoudinireadyyet.com/*
83. *https://developer.mozilla.org/docs/Web/API/CSS/RegisterProperty*
84. *https://quasar.dev/vue-components/expansion-item*
85. *https://developer.mozilla.org/docs/Web/API/CSS_Painting_API*
86. *https://github.com/w3c/css-houdini-drafts/blob/master/css-typed-om/README.md*

CSS-in-JS

CSS-in-JSについては、誰もが愛犬と一緒に使っていると思われるほど、多くの議論（または議論）が行われています。

2%

図1.50. 任意のCSS-in-JS方式を使用しているサイトの割合。

しかし、各種CSS-in-JSライブラリの利用状況を見てみると、いずれかのCSS-in-JS方式を利用しているWebサイトは2%程度で、Styled Components[87]が半分近くを占めていることがわかりました。

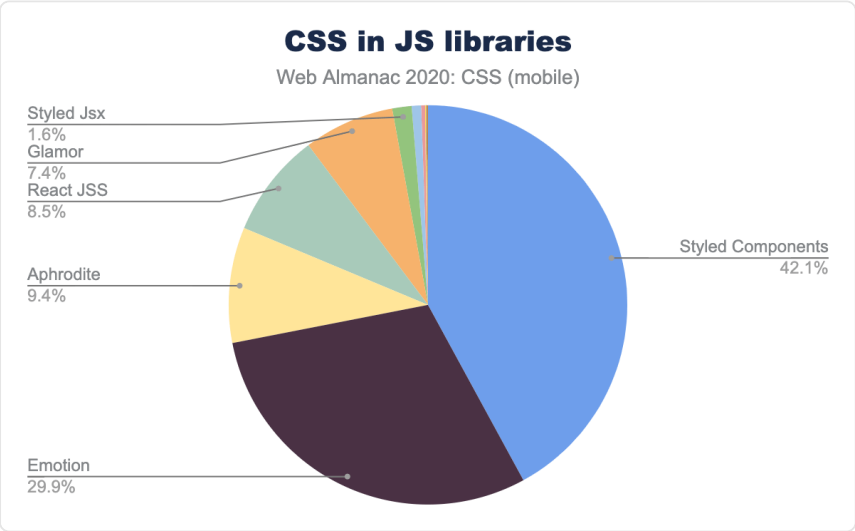


図1.51. モバイルページでのCSS-in-JSライブラリの相対的な普及率。

国際化

英語は多くの言語と同様に、横書きで書かれ、文字は左から右に並べられています。しかし、一部の言語（アラビア語やヘブライ語など）では、ほとんどが右から左に書かれていて、上から下に向かって縦書きで書かれている場合もあります。他の言語からの引用は言う

87. https://styled-components.com/

までもありません。そのため、物事は非常に複雑になることがあります。HTMLとCSSの両方には、これを処理する方法があります。

方向

テキストが水平線で表示されている場合、ほとんどの筆記システムでは左から右へ文字が表示されます。ウルドゥー語、アラビア語、ヘブライ語は右から左に文字を表示しますが、数字は左から右に書かれています。括弧、引用符、句読点などの一部の文字は、左から右または右から左の文脈で使用され、方向性は中立であると言われています。異なる言語のテキスト文字列が互いに入れ子になっている場合、状況はより複雑になります。例えば、英語のテキストにヘブライ語の短い引用符が含まれていて、その中に英語の単語が含まれている場合などです。Unicodeの双方向性アルゴリズムは、方向が混在するテキストの段落をどのようにレイアウトするかを定義していますが、段落の基本方向を知る必要があります。

双方向性をサポートするために、HTMLでは `dir` 属性と `<bdo>` 要素、CSSではdirection[88]と `unicode-bidi` の両方のプロパティを使って方向を明示的にサポートしています。HTMLとCSSの両方のメソッドの使い方を見てみました。

モバイルページのうち、 `<html>` 要素に `dir` 属性を設定しているのはわずか12.14%（デスクトップでは同様の10.76%）です。世界のほとんどのライティングシステムは `ltr` であり、デフォルトの `dir` の値は `ltr` です。 `<html>` に `dir` を設定したページのうち、91%が `ltr` に設定し、8.5%が `rtl` に設定し、0.32%が `auto` に設定した（明示的な方向は未知の値で、主に未知の内容で埋められるテンプレートに有用です）。さらに少数の2.63%は `dir` を `<html>` よりも `<body>` に設定しています。これは良いことで、 `<html>` に設定すると `<title>` のように `<head>` の中のコンテンツもカバーできるからです。

なぜCSSのスタイリングではなく、HTML属性を使って方向性を設定するのか？　理由の1つには、懸念事項の分離があります。これは、推奨されている方法[89]でもあります。“マークアップが使える方向性を管理するためにCSSやUnicodeのコントロールコードを使うのは避ける”。結局のところ、スタイルシートは読み込まれないかもしれませんし、テキストはまだ読み込まれる必要があります。

論理的特性と物理的特性

CSSを学ぶ際に最初に教えられるプロパティの多くは、 `width` , `height` , `margin-left` , `padding-bottom` , `right` など、特定の物理的な方向性に基づいています。しかし、コンテンツを異なる指向性特性を持つ複数の言語で表示する必要がある場合、これらの物理的な方向はしばしば言語に依存します。方向性は2次元的な特性である。例えば、縦書きで

88. *https://www.w3.org/TR/css-writing-modes-3/#direction*
89. *https://www.w3.org/International/tutorials/bidi-xhtml/index.en*

コンテンツを表示する場合（伝統的な中国語のような）、`height`は`width`になる必要があるかもしれません。

過去には、これらの問題に対する唯一の解決策は、異なる記述システム用のオーバーライドを持つ別個のスタイルシートでした。しかし、最近になって、CSSは*物理的な*プロパティと値を取得しました。これは *物理的な*プロパティと同じように動作しますが、コンテキストの方向性に敏感です。例えば、`width`の代わりに`inline-size`と書き、`left`の代わりに`inset-inline`プロパティを使用できます。論理的な *property* の他に、論理的な *keywords* もあります。例えば、`float: left`の代わりに`float: inline-start`と書くこともできます。

これらのプロパティはかなりよくサポートされている[90]（いくつかの例外を除いて）ですが、ユーザーエージェントスタイルシートの外ではあまり使われていません。論理プロパティは0.6%以上のページで使用されていませんでした。ほとんどの場合、余白とパディングを指定するために使用されていた。`text-align`のための論理キーワードは2.25%のページで使われていたが、それ以外のキーワードには全く出くわしていなかった。これはブラウザのサポートによるところが大きい。`text-align: start`と`end`は かなり良いブラウザサポート[91]であるのに対し、`clear`と`float`の論理キーワードはFirefoxでしかサポートされていません。

ブラウザサポート

ウェブプラットフォームの長年の問題は、新しい機能をどのように導入してプラットフォームを拡張するかということです。CSSでは、変化を導入するより良い方法として、ベンダープレフィックスからフィーチャークエリへの移行を見てきました。

ベンダープレフィックス

91.05%

図1.52. いずれかのベンダーの接頭辞付き機能を使用しているモバイルページの割合。

プレフィックスは開発者に実験的な機能を紹介するための失敗した方法として認識されており、ブラウザはほとんどの場合プレフィックスの使用を止め、代わりにフラグを選択していますが、それでも91%のページでは少なくとも1つのプレフィックス機能が使用されています。

90. https://caniuse.com/css-logical-props
91. https://caniuse.com/mdn-css_properties_text-align_flow_relative_values_start_and_end

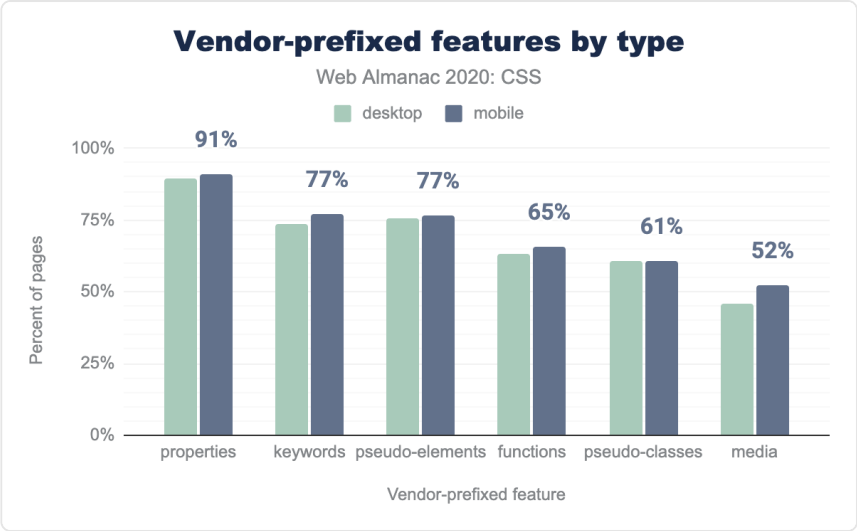


図1.53. タイプ別で最も人気のあるベンダープリフィックス機能をページ数に占める割合で表示しています。

使用されているプレフィックス機能の84%がプロパティで、モバイルページの90.76%、デスクトップページの89.66%で使用されています。これは、2009年から2014年頃のCSS3時代の名残である可能性が高いです。これは、最も人気のある接頭辞付きのものからも明らかですが、2014年以降、接頭辞を必要としていないものはありませんでした。

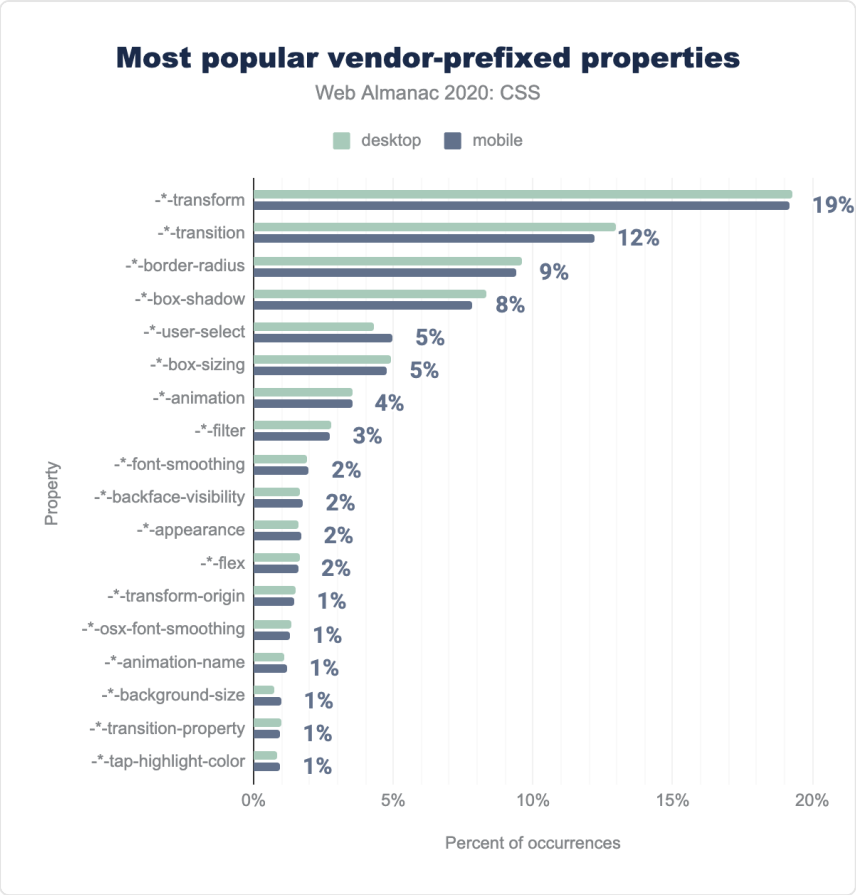


図1.54. ベンダープレフィックスを使用して最もよく使用されるプロパティの相対的な人気度。

これらの接頭辞の中には、`-moz-border-radius`のようなものがあり、2011年以降は役に立たなくなっています。さらに驚くべきことに、これまで存在しなかった他の接頭辞付きプロパティは、今でも適度に一般的で、全ページの約9%が`-o-border-radius`を含んでいます！このような接頭辞付きプロパティは、2011年には存在しませんでした。

`webkit-`が最も人気のある接頭辞で、接頭辞付きプロパティの半数がこの接頭辞を使用していることは驚くことではありません。

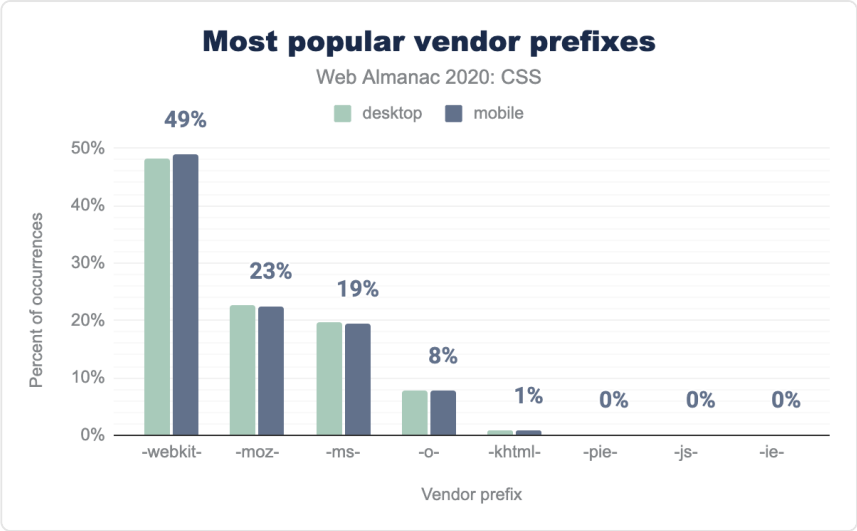


図1.55. ベンダープレフィックスの相対的な人気度、出現率としての。

接頭辞付き擬似クラスはプロパティほど一般的ではなく、10%以上のページで使われているものはありません。接頭辞付き擬似クラス全体の3分の2近くは、プレースホルダのスタイリングのためのものです。対照的に、標準の `:placeholder-shown` 擬似クラスはほとんど使用されておらず、0.34%以下のページで使用されています。

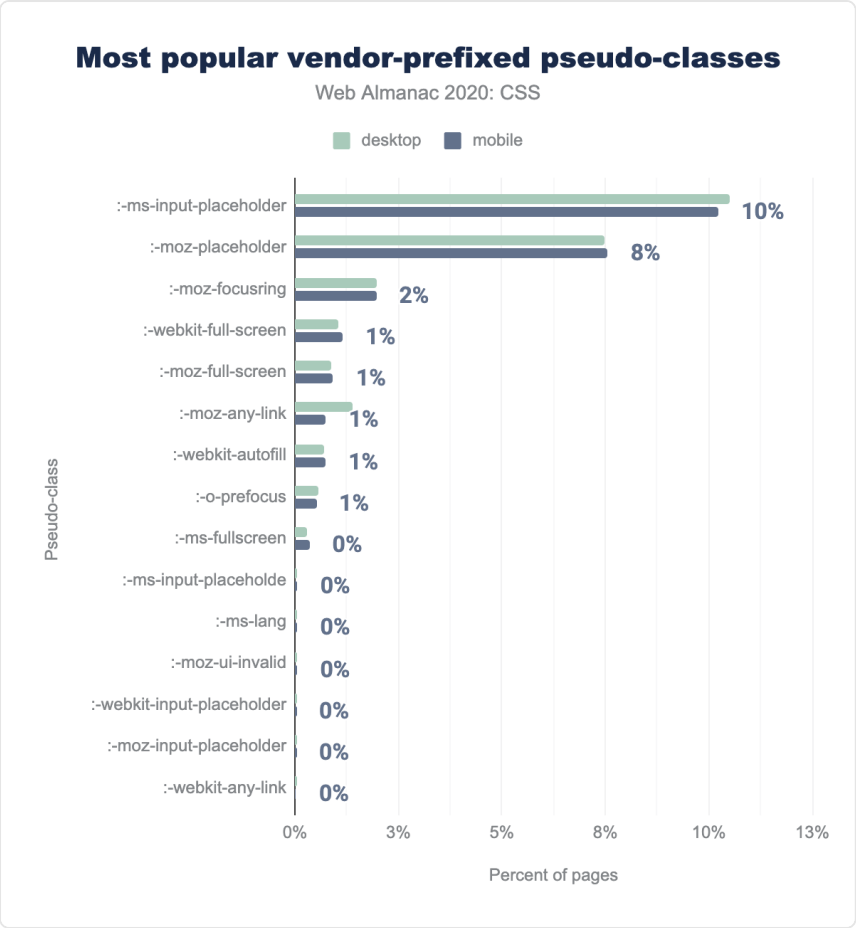


図1.56. ページに占める割合として、最も人気のあるベンダープリフィックスの疑似クラス。

最も一般的な接頭辞付き疑似要素は`::-moz-focus-inner`で、Firefoxの内側のフォーカスリングを無効にするために使われます。これは接頭辞付き疑似要素のほぼ4分の1を占めており、標準的な代替手段はありません。標準バージョンの`::placeholder`は後れを取っており、使用されているページはわずか4%に過ぎません。

接頭辞付き擬似要素の残りの半分は、主にネイティブ要素（検索入力、ビデオ＆オーディオコントロール、スピナーボタン、スライダー、メーター、プログレスバー）のスクロールバーとシャドウDOMのスタイリングに費やされています。これは、標準に準拠したCSSはまだ不十分であるものの、標準に準拠した組み込みUIコントロールのカスタマイズに対する開発者の強いニーズを示しています。

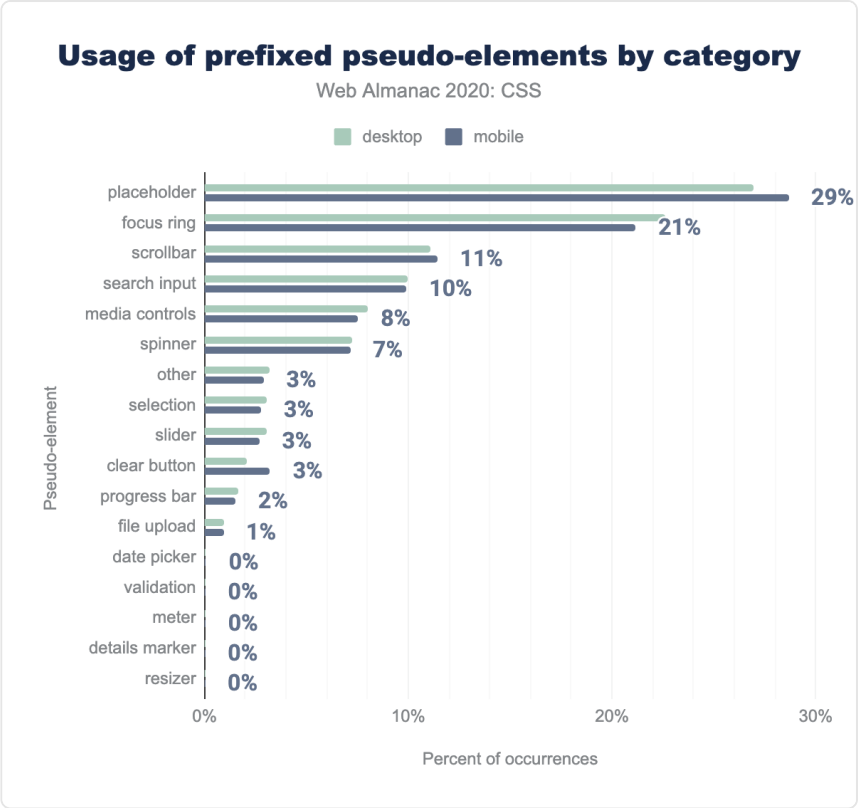


図1.57. カテゴリ別の接頭辞付き疑似要素の使用法。

ChromeやSafariの方がずっとプレフィックスを好むようになっているのは周知の事実ですが、特に疑似要素についてはその傾向が顕著です。

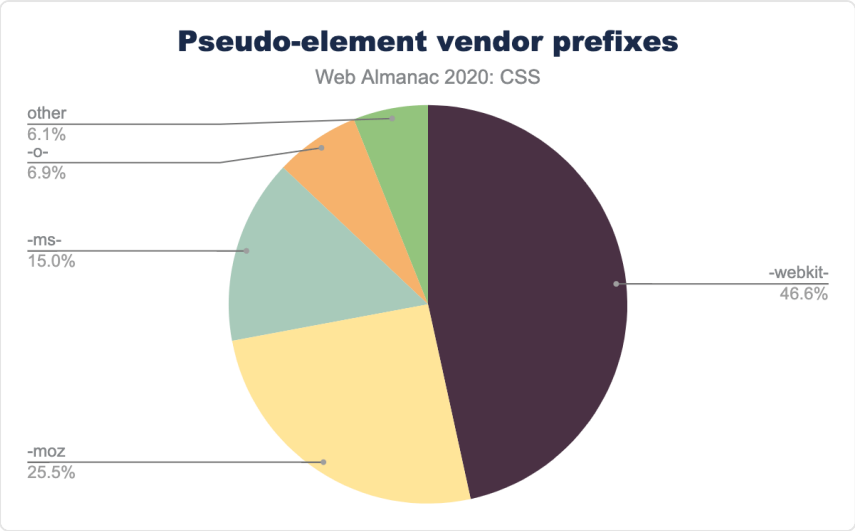


図1.58. 疑似要素ベンダープレフィックスの相対的な人気度を、モバイルページでの出現率として示しています。

接頭辞付き関数のほぼすべての使用法(98%)はグラデーションを指定するためのものであるが、2014年以降は必要とされていない[92]。これらの中で最もよく使われているのは `-webkit-linear-gradient()` で、調査したページの4分の1以上で使われています。残りの2%未満は主にcalcのためのもので、2013年以降は接頭辞が不要となっています[93]。

98.22%

図1.59. モバイルページでのベンダープレフィックス関数の全出現率におけるグラデーション関数の割合

接頭辞付きメディア機能の使用率は全体的に低く、最も人気のあるものは `-webkit-min-pixel-ratio` で、「Retina」ディスプレイを検出するために13%のページで使用されています。これに対応する標準メディア機能である `resolution` がついにこれを上回り、モバイルページの22%とデスクトップページの15%で使用されています。

全体では、 `*-min-pixel-ratio` はデスクトップではプレフィックス付きメディア機能の4分の3を占め、モバイルでは約半分を占めています。この違いの理由はモバイルでの利用率の低下ではなく、もう1つのプレフィックスメディア機能である `-*-ハイコントラスト` の人

92. https://caniuse.com/css-gradients
93. https://caniuse.com/calc

気がモバイルでははるかに高く、プレフィックスメディア機能の残りの半分近くを占めていますが、デスクトップでは18%しか占めていません。対応する標準メディア機能であるforced-colors[94]は、ChromeとFirefoxではまだ実験的であり、フラグの後ろに隠れているため、我々の分析では全く表示されていません。

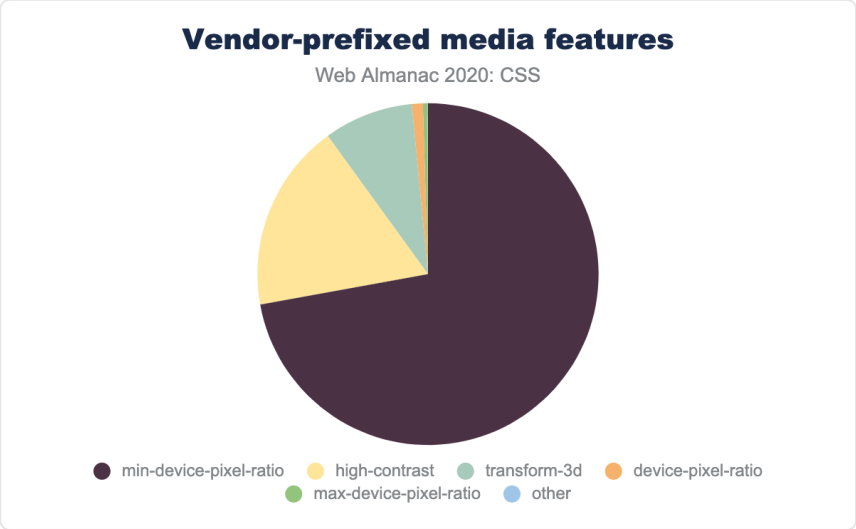


図1.60. ベンダーが接頭辞を付けたメディア機能の相対的な人気度を、モバイルページでの出現率として示しています。

フィーチャークエリ

フィーチャークエリ（@supports）は、ここ数年着実にトラクションを得ており、ページの39%で使用されており、昨年の30%から顕著に増加しています。

しかし、それらは何のために使用されているのでしょうか？我々は、最も人気のあるクエリを見てみました。結果は意外なものでした。グリッド関連のクエリがトップになると思っていたのですが、圧倒的に人気のある機能クエリは`position: sticky`のものでした。これはフィーチャークエリ全体の半分を占め、約4分の1のページで使用されています。対照的に、グリッド関連のクエリは全クエリの2%に過ぎず、開発者はグリッドのブラウザサポートが十分に快適であると感じているため、グリッドの機能強化としてだけ使用する必要はないことを示しています。

さらに不思議なのは、`position: sticky`自体が機能クエリほど使われておらず、デスクトップページの10%とモバイルページの13%にしか表示されていないことです。つまり、

94. https://developer.mozilla.org/docs/Web/CSS/@media/forced-colors

`position: sticky` を使わずに検出しているページが50万以上もあるということです。なぜでしょうか？

最後に、`max()` がすでに検出された機能のトップ10に入っており、デスクトップページの0.6%とモバイルページの0.7%で検出されていることは心強いことでした。`max()` (および `min()`、`clamp()`)[95] が 今年はボード全体でしかサポートされていなかったことを考えると、これは非常に印象的な採用であり、開発者がどれだけこの機能を必要としていたかを浮き彫りにしています。

少数ですが注目に値するページ数（約3000または0.05%）は、奇妙なことに、`@supports` よりもかなり前に存在していた `display : block` や `padding : 0px` などのCSS2構文で `@supports` を使用、実装されました。これが何を達成することを意図していたのかは不明です。おそらく、`@supports` を実装していない古いブラウザからCSSルールを保護するために使用されたのでしょうか？

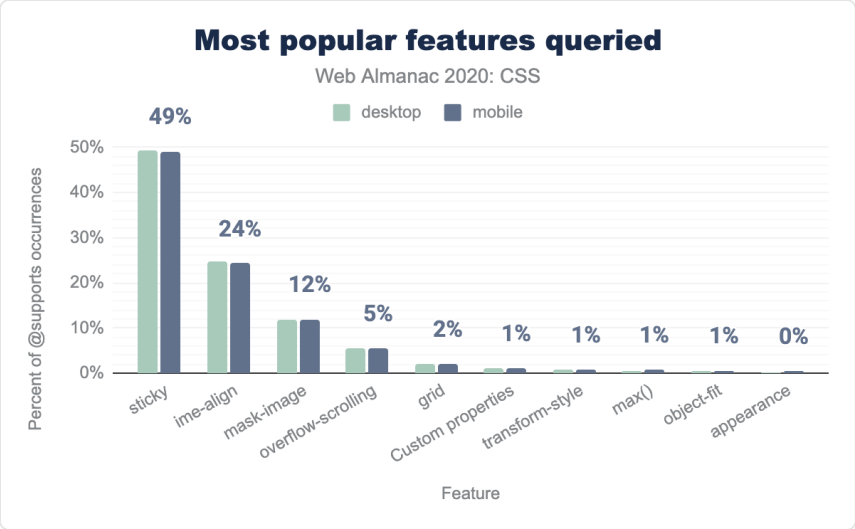


図1.61. サポート機能の相対的な人気度を、出現率として問い合わせました。

メタ

これまでCSS開発者が何を使用してきたかを見てきましたが、このセクションでは、CSS開発者がどのように使用しているかについて詳しく見ていきたいと思います。

95. https://caniuse.com/mdn-css_types_max

宣言の繰り返し

スタイルシートがどれだけ効率的で保守性の高いものであるかを知るために、大まかな要因の1つは宣言の繰り返し、つまり一意な(異なる)宣言と総宣言数の比率です。この要因は大まかなもので、宣言を正規化することは些細なことではないからです (`border: none`, `border.0`, `border-width: 0`, `border-width: 0`)。また、繰り返しにはメディア・クエリ（最も有用ですが測定が難しい）、スタイルシート、Almanacの全体的なメトリクスのようなデータセット・レベルなどのレベルがあるからです。

宣言の繰り返しに注目してみたところ、モバイルでは中央値のウェブページで合計5,454個の宣言を使用しており、そのうち2,398個が一意の宣言であることがわかりました。中央値の比率（この2つの値ではなく、データセットに基づいています）は45.43%でした。これが意味するのは、中央値のページでは、各宣言がおよそ2回使用されているということです。

パーセンタイル	ユニーク / 全体
10	30.97%
50	45.43%
90	63.67%

図1.62. モバイルページのリピート率の分布。

これらの比率はその後、我々が乏しい過去のデータから知っているものよりも優れています。2017年、Jens Oliver Meiert220の人気ウェブサイトをサンプリング[96]して、以下のような平均値を出しました。6,121件の宣言があり、そのうち1,698件がユニークで、ユニーク/全体の比率は28%(中央値34%)でした。このトピックについてはさらに調査が必要かもしれませんが、これまでのところわかっていることはほとんどありません、宣言の繰り返しは目に見えるものであり、人気のある大規模なサイトでは改善されているか問題となっている可能性があります。

短縮形と通常の記載

短縮形の中には、他のものよりも成功しているものもあります。時には、その短縮形は十分に使いやすく、その構文も覚えやすいので特定の値を独立して上書きしたい場合には、意図的に通常の記載だけを使うことになることもあります。また、構文があまりにも混乱しているために、ほとんど使われていない短縮形もあります。

96. https://meiert.com/en/blog/70-percent-css-repetition/

通常の記載の前の短縮形

短縮形の中には、他のものよりも成功しているものもあります。時には、その短縮形は十分に使いやすく、その構文も覚えやすいので特定の値を独立して上書きしたい場合には、意図的に通常の木しあだけを使うことになることもあります。また、構文があまりにも混乱しているために、ほとんど使われていない短縮形もあります。短縮形を使って、同じルールの中でいくつかの通常の記載を使ってそれをオーバーライドするのは、さまざまな理由から良い戦略と言えます。

第一に、それは良い守備的コーディングです。短縮形は、明示的に指定されていない場合、すべての通常の記載を初期値にリセットします。これにより、カスケードを通して不正な値が入ってくるのを防ぎます。

第二に、短文がスマートなデフォルト値を持っている場合に、値の繰り返しを避けることは保守性に優れています。例えば、`margin: 1em 1em 0 1em` の代わりに、次のように書くことができます。

```
margin: 1em;
margin-bottom: 0;
```

同様にリスト値のプロパティについても、すべてのリスト値で値が同じである場合には、通常の記載を使用することで繰り返しを減らすことができます。

```
background: url("one.png"), url("two.png"), url("three.png");
background-repeat: no-repeat;
```

第三に、短縮形の構文の一部があまりにも奇妙な場合、通常の記載は可読性を向上させるのに役立ちます。

```
/* Instead of: */
background: url("one.svg") center / 50% 50% content-box border-box;

/* This is more readable: */
background: url("one.svg") center;
background-size: 50% 50%;
background-origin: content-box;
```

```
background-clip: border-box;
```

では、どのくらいの頻度でこのようなことが起こるのでしょうか？　非常に頻繁です。88%のページで少なくとも一度はこの戦略を使用しています。これは、`background`の中の`background-size`のスラッシュ構文が、私たちが考え出した中で最も読みやすく、記憶に残りやすい構文ではなかった可能性があることを示しています。他のロングハンドでは、このような頻度に近いものはありません。残りの60%はロングテールで、他の多くのプロパティに均等に広がっています。

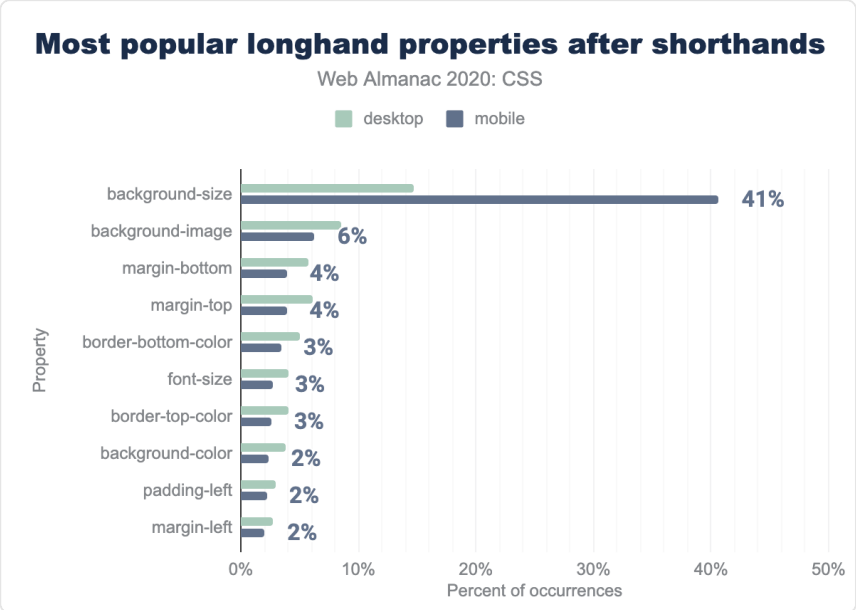


図1.63. 同じルールの短縮形の後に来る最も人気のある通常の記載。

font

`font`はかなり人気がありますが（80%のページで4,900万回使われています）、他のほとんどの通常の記載（`font-variant`と`font-stretch`を除く）に比べれば、ほとんど使われていません。これは、ほとんどの開発者がこの通常の記載を快適に使っていることを示しています (この通常の記載は非常に多くのウェブサイトで使われているので)。開発者はしばしば、子孫ルールで特定のタイポグラフィを上書きする必要がありますが、 それがなぜ通常の記載がこれほど多く使われているのかを説明しているのでしょう。

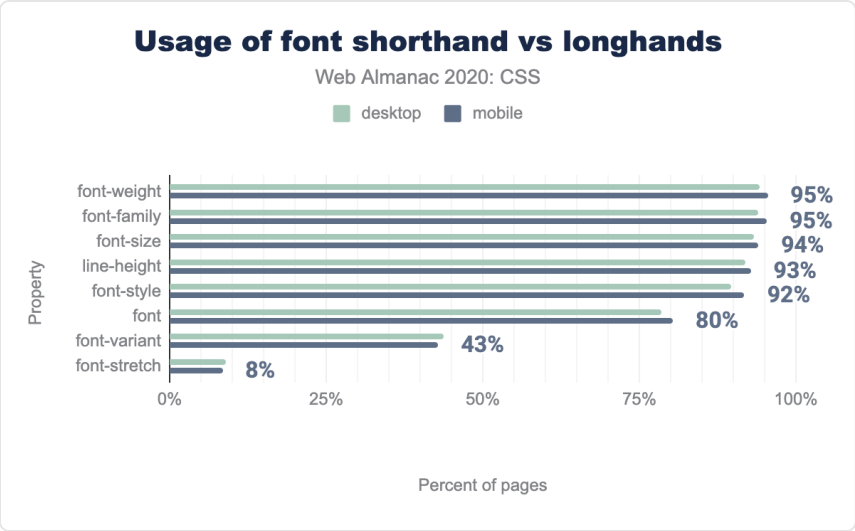


図1.64. `font` の短縮形とロングハンドのプロパティを採用しました。

background

最も古い通常の記載の一つである `background` もまた、非常によく使われており、92%のページで10億回使用されています。しかし、これは開発者がすべての構文を使いこなせているわけではありません: `background` のほぼすべて（90%以上）は非常にシンプルなもので、1つか2つの値を持つものです。それ以上のものについては、ロングハンドの方がわかりやすいと考えられています。

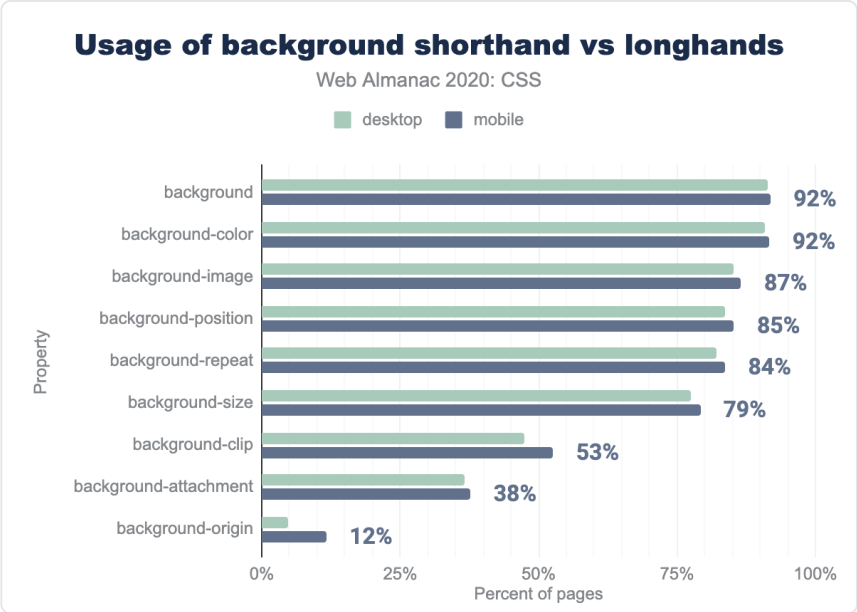


図1.65. `background` の短縮形とその通常の記載の用法比較。

Marginsとpaddings

`margin` と `padding` の両方の短縮形、およびそれらの通常の記載は、CSSプロパティの中で最もよく使用されているものの1つです。Paddingは短縮記号として指定される可能性がかなり高くなっています（`padding` は150万回使用されているのに対し、それぞれの短縮記号は300〜400万回使用されています）が、marginはあまり差がありません（`margin` は110万回使用されているのに対しそれぞれの短縮記号は500〜800万回使用されています）。多くのCSS開発者がこれらの短縮形の値の時計回りの順序と、2つまたは3つの値の繰り返しルールについて最初は混乱していたことから、これらの短縮形の使用のほとんどは単純なもの（1つの値）になると予想していましたが、1,2,3,4つの値のすべての範囲を見ることができました。明らかに1や2の値の方が一般的ですが、3や4の値は全く珍しくなく、`margin` の25%以上、`padding` の10%以上で使用されています。

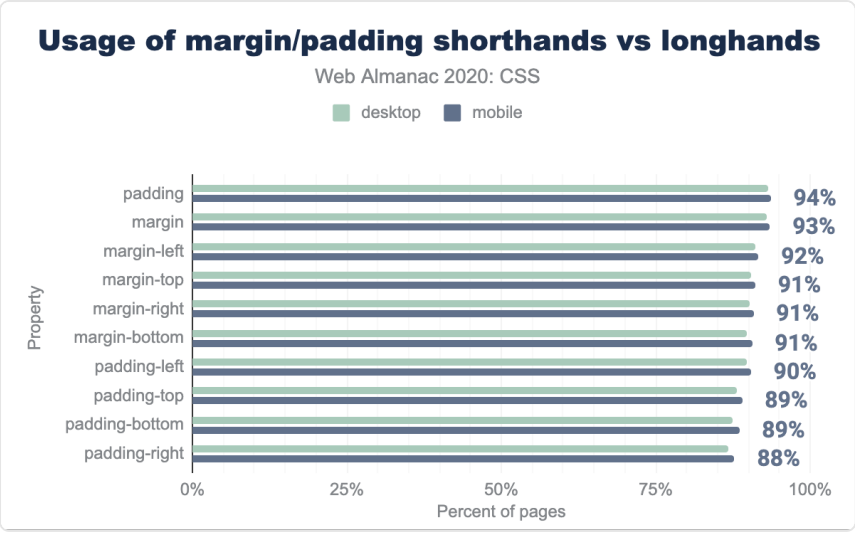


図1.66. `margin` と `padding` の短縮形とその通常の記載の用法比較。

Flex

ほぼすべての `flex`, `flex-*` プロパティは非常によく使われており、30〜60%のページで使用されています。しかし、 `flex-wrap` と `flex-direction` の両方が、その短縮形である `flex-flow` よりもはるかに多く使われています。 `flex-flow` が使われるときは、2つの値、つまり両方の通常の記載を短く設定する方法として使われます。1つまたは2つの値で `flex` を使用するための精巧な賢明なデフォルト[97]があるにもかかわらず、使用の約90%は3つの値の構文で、3つの通常の記載を明示的に設定しています。

97. *https://developer.mozilla.org/docs/Web/CSS/flex#Syntax:~:text=The%20flex%20property%20may%20be%20specified%20using%20one%2C%20two%2C%20or%20three%20value*

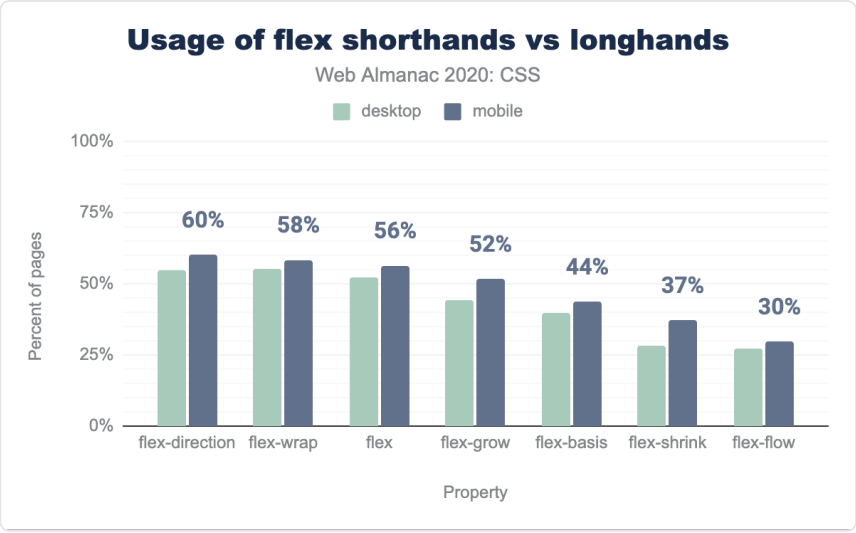


図1.67. flexの短縮形とその通常の記載の使い方比較

Grid

`grid-template` の略語として `grid-template-columns`, `grid-template-template-rows`, `grid-template-areas` があることをご存知でしょうか? また、 `grid` プロパティがあり、それらはすべてその通常の記載の一部であることを知っていましたか? 知らないのですか？　ほとんどの開発者はそうではありません。 `grid` プロパティが使われたのは 5,279 のウェブサイト(0.08%)で、 `grid-template` が使われたのは 8,215のウェブサイト(0.13%)だけでした。一方、 `grid-template-columns` は170万のウェブサイトで使用されており、その200倍以上も使用されています。

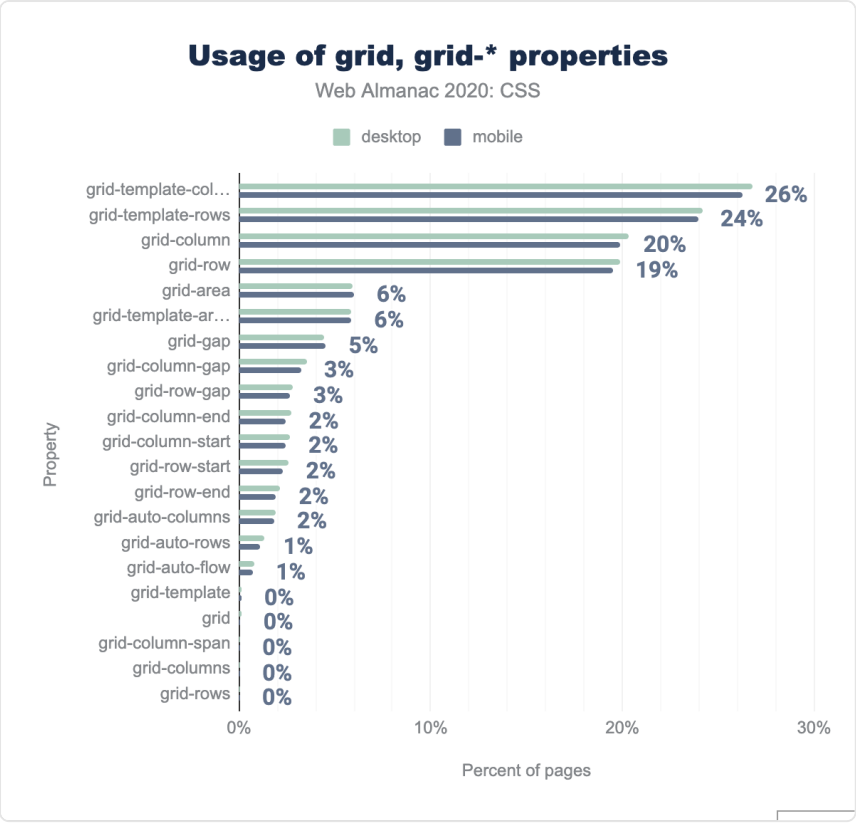


図1.68. Gridの短縮形とその通常の記載の使い方比較

CSSの間違い

どんな複雑で進化するプラットフォームでもそうですが、すべてが正しく行われるわけではありません。そこで、開発者が犯している間違いのいくつかを見てみましょう。

構文エラー

この章のほとんどのメトリクスでは、CSSパーサーであるRework[98]を使用しました。これは精度を劇的に向上させるのに役立ちますが、ブラウザに比べて構文エラーに寛容でないことも意味します。スタイルシート全体の1つの宣言に構文エラーがあったとしても、構文解析は失敗し、そのスタイルシートは解析の対象外となります。しかし、そのような構文エラー

98. https://github.com/reworkcss/css

を含むスタイルシートはどれくらいあるのだろうか？　モバイルよりもデスクトップの方がかなり多いことが判明しました。具体的には、デスクトップページで発見されたスタイルシートの10%近くに少なくとも1つの回復不可能な構文エラーが含まれていたのに対し、モバイルページでは2%しか含まれていませんでした。すべての構文エラーが実際にパースに失敗するわけではないので、これらは基本的には構文エラーの下限値であることに注意してください。例えば、セミコロンがない場合、次の宣言が値の一部として解析されるだけで（例えば、`{property: "color", value: "red background: yellow"}`など）、パーサが失敗することはありません。

存在しないプロパティ

また、既知のプロパティのリストを使用して、最も一般的な存在しないプロパティを調べました。この分析のこの部分から接頭辞付きのプロパティを除外し、接頭辞なしのプロプライエタリなプロパティを手動で除外しました (例: Internet Explorerの`behavior`、奇妙なことに今でも20万件のウェブサイトに表示されています)。残りの存在しないプロパティのうち。

- そのうち37%は接頭辞付きプロパティの変形した形式でした(例: `webkit-transition`や`-transition`)。
- 43%は接頭辞のみでしか存在しないプロパティの接頭辞なしの形式(例えば、`font-smoothing`は384Kのウェブサイトに登場しました)で、おそらく互換性のために、標準であるという誤った仮定の下で、あるいは標準になることを願って含まれていたのでしょう。
- 人気のあるライブラリへの道を見つけたタイプミス。この分析の結果、`white-wpace`というプロパティが234,027のウェブサイトに存在していることがわかりました。これは、同じタイプミスが有機的に発生したとは思えないほどの数です。そして驚いたことに、Facebookウィジェットが原因であることが判明しました(https://x.com/rick_viscomi/status/1326739379533000704)。修正はすでに行われています。
- そして、もう一つの奇妙なことがあります。プロパティ`font-rendering`が2,575ページに登場しています。しかし、プリフィックスの有無にかかわらず、このようなプロパティが存在する証拠を見つけることができません。非標準の`-webkit-font-smoothing`がありますが、これは300万のウェブサイト、つまりページの約49%に登場しており、非常に人気がありますが、`font-rendering`はスペルミスというには十分に近くありません。`テキストレンダリング`は約10万のWebサイトで使われているので、2.5万人の開発者が皆、`font-smoothing`と`text-rendering`の混成語を間違えて作ってしまったと考えられます。

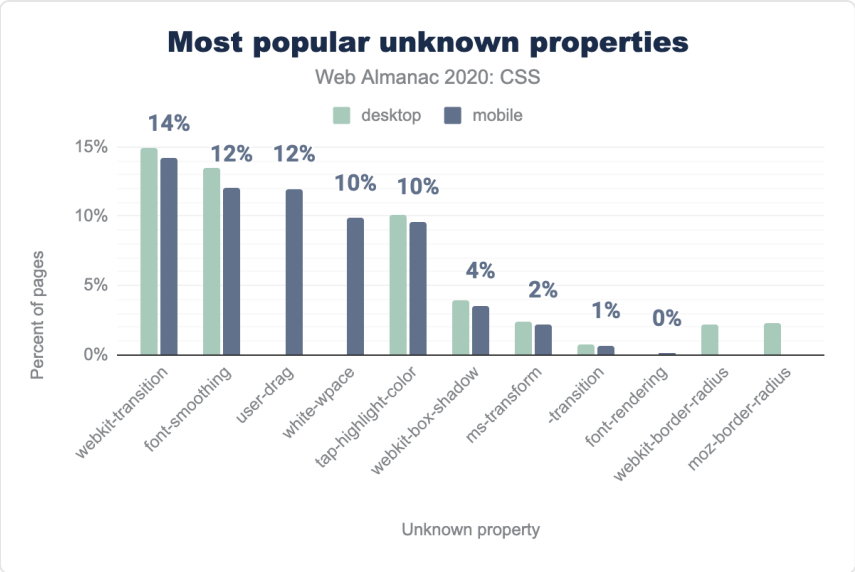


図1.69. 最も人気のある未知の物件。

短縮形の前の通常の記載

短縮形の後に通常の記載を使うのは、デフォルトを使ったり、いくつかのプロパティを上書きしたりするのに便利な方法です。これは特にリスト値を持つプロパティで便利です。反対に、短縮形の前に通常の記載を使うことは、常に間違いです。例えば、これを見てみましょう。

```
background-color: rebeccapurple; /* longhand */
background: linear-gradient(white, transparent); /* shorthand */
```

これは`white`から`rebeccapurple`へのグラデーションではなく、`white`から`transparent`へのグラデーションを生成します。`rebeccapurple`の背景色は、この後に続く`background`短縮形によって上書きされ、すべての通常の記載が初期値にリセットされます。

開発者がこのような間違いを犯す主な理由は2つあります。それは、短縮形がどのように動作し、どの短縮形によってどの通常の記載がリセットされるのかについての誤解、あるいは宣言を移動させた際に残った残骸です。

では、この間違いはどのくらい一般的なのでしょうか？　確かに、トップ600万のウェブサ

イトでは、それほど一般的ではありませんよね？　違う！　結局のところ、それは非常に一般的であり、ウェブサイトの54%で少なくとも一度は発生しています！

この種の混乱は、他の速記法よりも `background` 速記法の方がはるかに多いようです: これらの間違いの半分以上(55%)は、 `background` の前に `background-*` の通常の記載を置くことに起因しています。この場合、これは実際には全くの間違いではなく、優れたプログレッシブな機能強化なのかもしれません。リニアグラデーションなどの機能をサポートしていないブラウザは、以前に定義された通常の記載値、この場合は背景色をレンダリングします。短縮形を理解しているブラウザは、暗黙的にまたは明示的に通常の記載に上書きします。

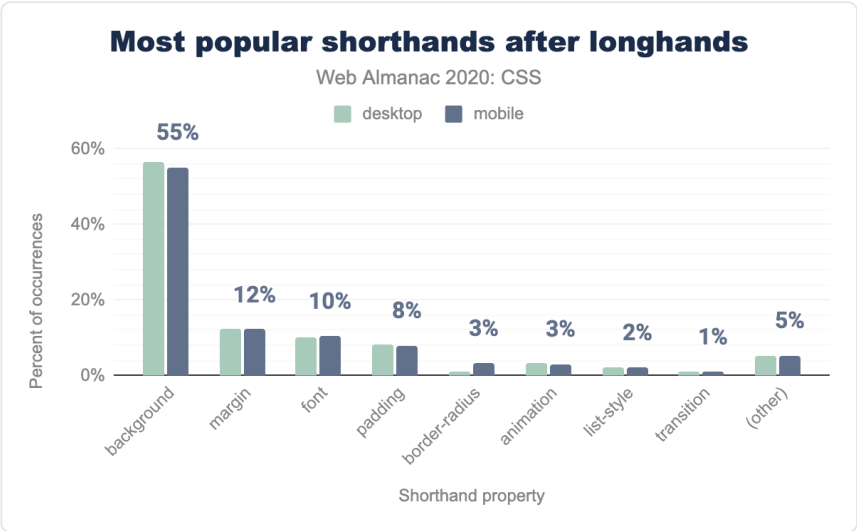


図1.70. 通常の記載の次に人気のある短縮形。

Sass

CSSコードを分析することで、CSS開発者が何をしているかがわかりますが、プリプロセッサのコードを見ることで、CSS開発者が何をしたくてもできないことを少し知ることができます。Sassは2つの構文で構成されています。よりミニマルなSassと、CSSに近いSCSSです。前者は人気が落ちてきていて、今ではあまり使われていないので、後者だけを見てみました。ソースマップ付きのCSSファイルを使って、SCSSのスタイルシートを抽出して解析してみました。ソースマップの分析に基づいて、SCSSは最もポピュラーな前処理構文であるため、SCSSに注目することにしました。

開発者が色修正機能を必要としていることは以前からわかっており、CSS Color 5[99]で取り組んでいます。しかし、SCSSの関数呼び出しを解析することで、色修正関数がどれだけ必要かを証明するためのデータが得られ、またどのような種類の色修正が最も一般的に必要とされているかを知ることができます。

全体的に、Sassの関数呼び出しの3分の1以上は、色を変更したり色の成分を抽出したりするために行われています。私たちが見つけたほとんどの色の変更は、かなり単純なものでした。半分は色を暗くするためのものでした。実際、`darken()`は全体的に最も人気のあるSass関数コールで、一般的な`if()`よりも多く使われていました。明るいコアカラーを定義し、`darken()`を使って暗い色のバリエーションを作るのが一般的な戦略のようです。反対に明るくすることはあまり一般的ではなく、関数呼び出しのうち`lighten()`が使われているのは5%に過ぎませんが、それでも全体では6番目に人気のある関数でした。アルファチャンネルを変更する関数は全体の約4%で、色を混ぜる関数は全体の3.5%となっている。色相、彩度、赤/緑/青のチャンネルを調整する関数や、より複雑な`adjust-color()`のような他のタイプの色の変更はほとんど使われていません。

99. *https://drafts.csswg.org/css-color-5/*

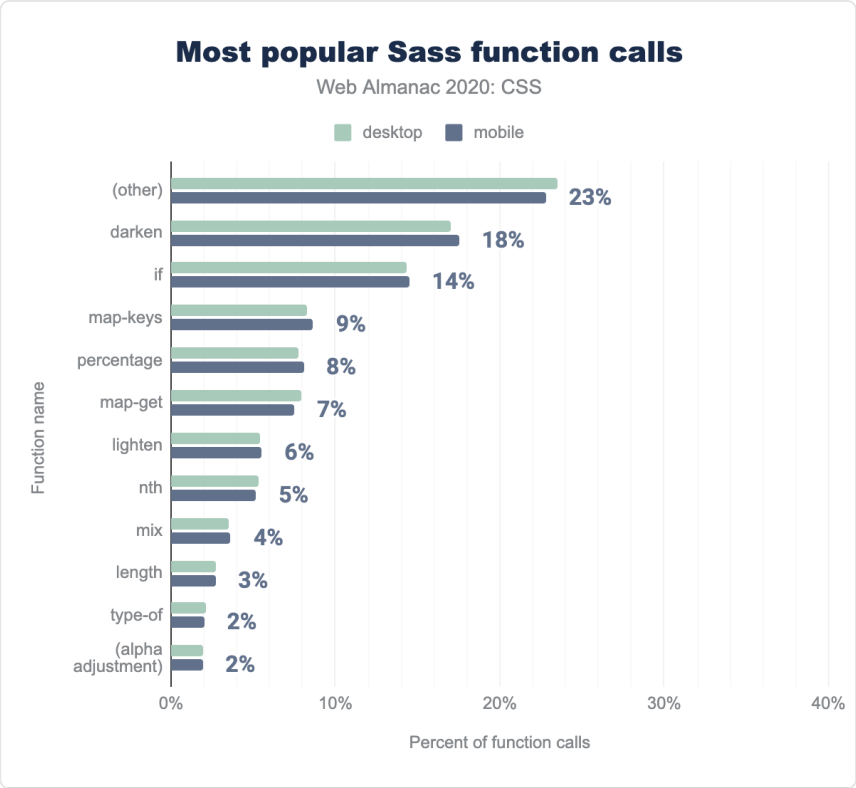


図1.71. 最も人気のあるSassの関数呼び出し。

カスタム関数の定義はHoudiniで何年も議論されてきた[100]のですが、Sassスタイルシートを研究していると、その必要性がどれだけ一般的なものかというデータが得られます。非常に一般的であることがわかりました。調査したSCSSスタイルシートの少なくとも半分はカスタム関数を含んでおり、中央値のSCSSシートは1つではなく2つのカスタム関数を含んでいます。

また、CSS WGでは限定的な条件式の導入について最近[101]議論[102]が行われており、Sassではこれがどのくらいの頻度で必要とされているかについてのデータを提供してくれています。SCSSのシートのほぼ3分の2が少なくとも1つの `@if` ブロックを含んでおり、すべての制御フロー文のほぼ3分の2を占めています。また、値内の条件式のための `if()` 関数もあり、これは全体で2番目によく使われる関数です(14%)。

100. https://github.com/w3c/css-houdini-drafts/issues/857
101. https://github.com/w3c/csswg-drafts/issues/5009
102. https://github.com/w3c/csswg-drafts/issues/5624

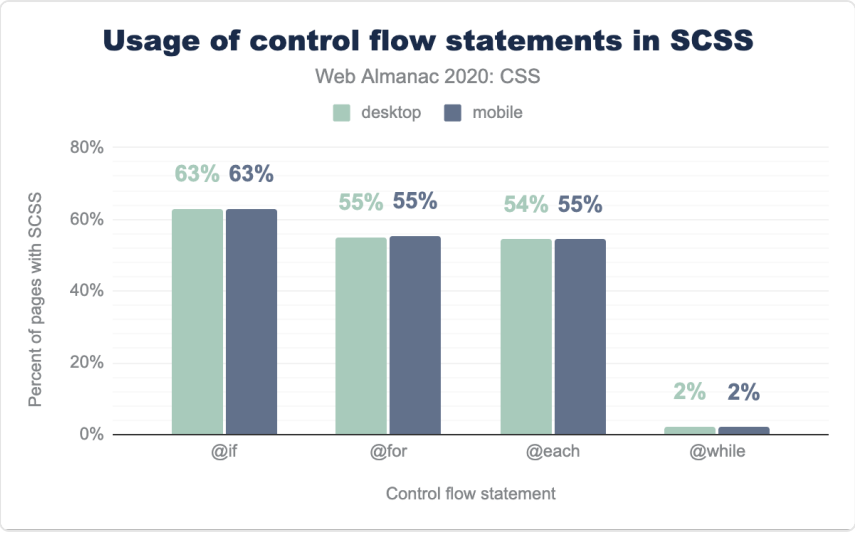


図1.72. SCSSでの制御フロー文の使用法

これは、Sassや他のプリプロセッサで`&`を使ってできることと同様に、他のルールの中にルールを入れ子にできます。SCSSシートでネスティングはどのくらい一般的に使われているのでしょうか？　非常によく使われています。SCSSシートの大部分は、少なくとも1つの明示的に入れ子にしたセレクタを使用しており、擬似クラス(例:`&:hover`)とクラス(例:`&.active`)がそのうちの4分の3を占めています。また、子孫が想定され、`&`文字が不要な暗黙の入れ子は考慮されていません。

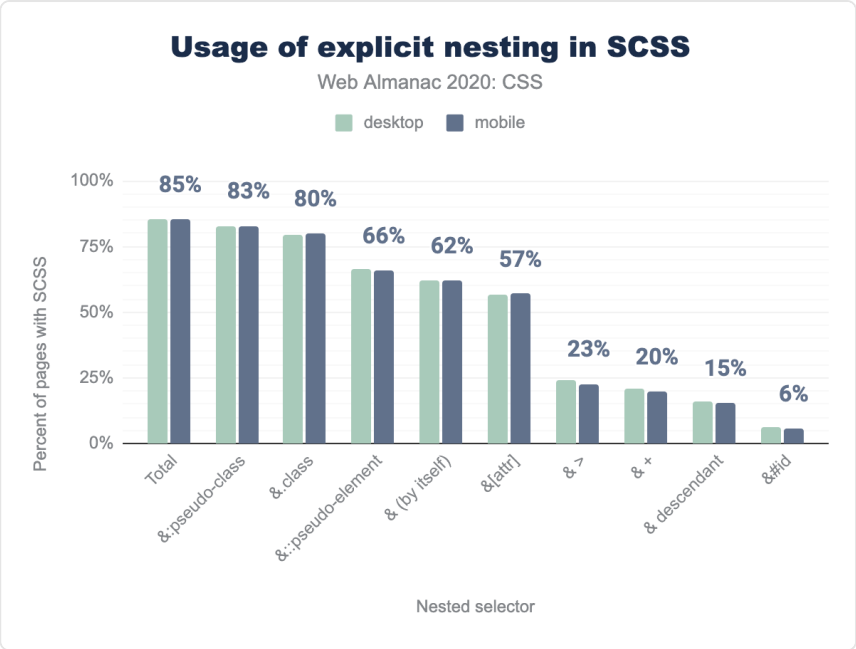


図1.73. usage-of-explicit-nesting-in-scss.

結論

ふー！　それはたくさんのデータでした！ 私たちが行ったように、あなたが興味を持ってくれたことを願っています。私たちは、あなたが私たちと同じように興味を持ってくれたことを願っていますし、おそらくそれらのいくつかについてあなた自身の洞察を形成できました。

その結果、WordPress、Bootstrap、Font Awesomeなどの人気のあるライブラリが新機能を採用する主な要因となっている一方で、個人の開発者は保守的な傾向があるということがわかりました。

もう1つの観察は、ウェブには新しいコードよりも古いコードの方が多いということです。実際のウェブは、20年前に書かれた可能性のあるコードから最新のブラウザでしか動作しない最先端の技術まで、非常に広範囲にわたっています。しかし、この研究が示してくれたのは相互運用性に優れているにもかかわらず、しばしば誤解され十分に活用されていない強力な機能があるということです。

また、開発者がCSSを使いたくても使えない方法のいくつかを示し、開発者が何を混乱させ

ているのかについての洞察を得ることができました。このデータの一部はCSS WGに持ち帰られ、CSSの進化を促進するために役立てられる予定です。

私たちは、この分析がウェブサイトの開発方法にさらなる影響を与える可能性があることに興奮しており、これらのメトリクスが時間の経過とともにどのように発展していくかを見るのを楽しみにしています。

著者

Lea Verou

@leaverou LeaVerou https://lea.verou.me/

Lea HCIとウェブプログラミングの講師[103]、ウェブプログラミングをより簡単にする方法[104]をMIT[105]で研究しています。彼女はベストセラーの技術系著者[106]であり、経験豊富な講演者[107]でもあります。彼女はオープンなウェブ標準に情熱を注いでおり、CSSワーキンググループ[108]の長年のメンバーでもあります。Leaは、人気のあるオープンソースプロジェクトやウェブアプリケーション[109]、Prism[110]、Awesomplete[111]などを立ち上げています。彼女は@leaverouでツイートし、lea.verou.me[112]でブログを書いています。

Chris Lilley

@svgeesus svgeesus https://svgees.us

Chris LilleyはWorld Wide Web Consortium(W3C)のテクニカルディレクターです。「SVGの父」と呼ばれ、PNGの共著、CSS2の共同編集者、`@font-face`を開発したグループの議長、WOFFの共同開発者でもあります。元テクニカルアーキテクチャグループ。ChrisはまだWeb上でColor Managementを使おうとしています、ため息。現在はCSSレベル3/4/5 (いや、本当に)、Web Audio、そしてWOFF2に取り組んでいます。

103. https://designftw.mit.edu
104. https://mavo.io
105. https://mit.edu
106. https://www.amazon.com/CSS-Secrets-Lea-Verou/dp/1449372635?tag=leaverou-20
107. https://lea.verou.me/speaking
108. https://www.w3.org/Style/CSS/members.en.php3
109. https://github.com/leaverou
110. https://prismjs.com
111. https://github.com/leaverou/awesomplete
112. https://lea.verou.me

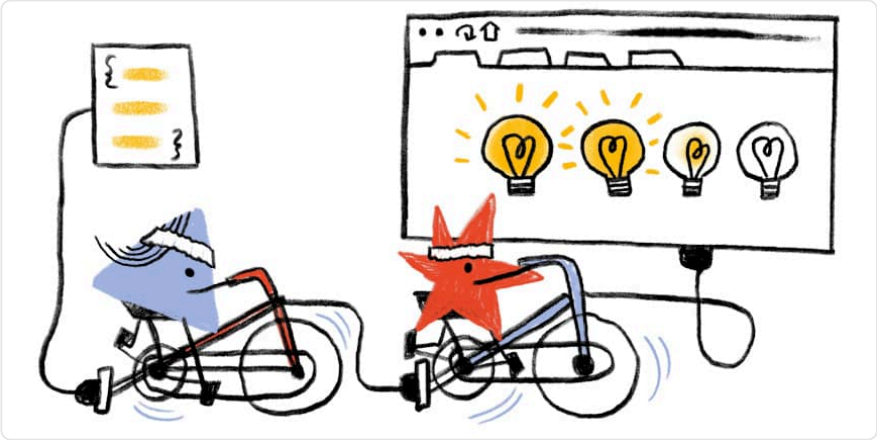
Rachel Andrew

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

私はウェブ開発者、ライター、スピーカーです。Perch CMS[113]とNotist[114]の共同設立者。CSSワーキンググループ[115]のメンバー。Smashing Magazine[116]の編集長です。

113. https://grabaperch.com
114. https://noti.st
115. https://www.w3.org/wiki/CSSWG
116. https://www.smashingmagazine.com/

部 I 章 2
JavaScript



Tim Kadlec によって書かれた。
Sawood Alam と Artem Denysov によってレビュ。
Rick Viscomi と Paul Calvano による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

JavaScriptは、CSSとHTMLと並んでウェブの3つの礎となる最後の礎として、その謙虚な起源から長い道のりを歩んできました。今日では、JavaScriptは技術スタックの幅広い範囲に浸透し始めています。JavaScriptはもはやクライアントサイドに限定されたものではなく、ビルドツールやサーバーサイドのスクリプティングのための選択肢として、ますます人気が高まっています。エッジコンピューティングソリューションのおかげで、JavaScriptはCDNレイヤーにも浸透してきています。

開発者はJavaScriptが大好きです。マークアップの章によると、`script`要素は、現在使用されている6番目に人気のあるHTML要素です（数え切れないほどの要素の中で、`p`や`i`などの要素よりも上位にあります）。ウェブの構成要素であるHTMLの約14倍、CSSの約6倍のバイト数を費やしています。

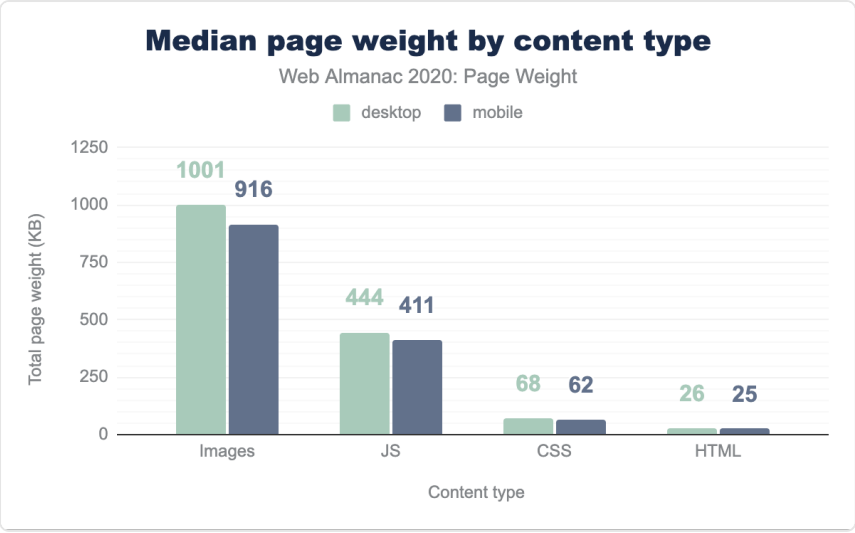


図2.1. コンテンツタイプごとのページ重さの中央値。

しかし、無料のものはありません。特にJavaScriptコードにはすべてコストがかかります。私たちがどのくらいのスクリプトを使っているのか、どのように使っているのか、そしてその結果はどうなるのか、詳しく見ていきましょう。

どのくらいのJavaScriptを使っているのか？

前に、`script`タグはHTML要素の中で6番目に多く使われている要素であることを述べました。実際にどのくらいのJavaScriptが使われているのか、もう少し掘り下げて見てみましょう。

中央値のサイト（50パーセンタイル）では、デスクトップ端末で読み込んだときに444KBのJavaScriptを送信し、モバイル端末ではわずかに少ない（411KBの）JavaScriptを送信しています。

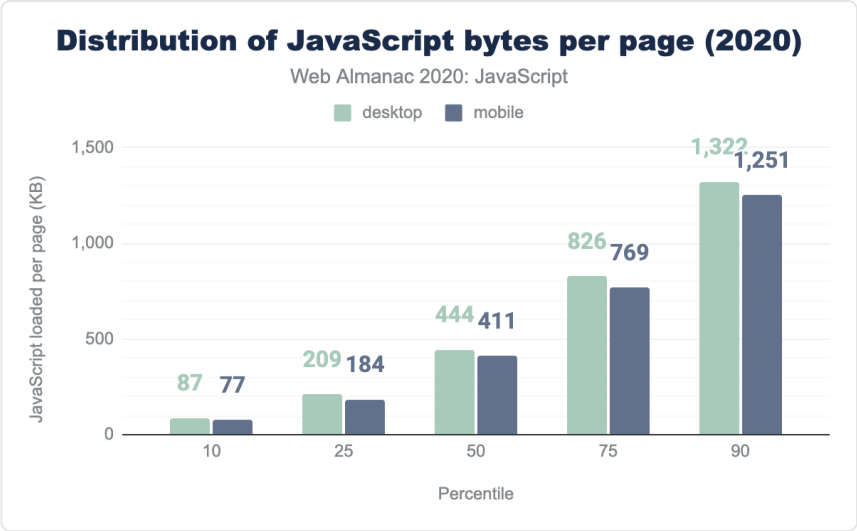


図2.2. 1ページあたりに読み込まれたJavaScriptのキロバイト数の分布。

ここに大きなギャップがないのは少し残念です。使用中のデバイスが電話かデスクトップ（またはその間のどこか）かどうかに基づいて、ネットワークや処理能力についてあまりにも多くの仮定をするのは危険ですが、HTTP Archive mobile tests は、Moto G4と3Gネットワークをエミュレートすることによって行われていることは注目に値します。言い換えれば、より少ないコードを渡すことであまり理想的でない状況に適応するために行われている作業があった場合、これらのテストはそれを示しているはずです。

また、傾向としては、JavaScriptの使用量を減らすのではなく、より多くのJavaScriptを使用することに賛成しているようです。昨年の結果[117]と比較すると、中央値ではデスクトップでテストされたJavaScriptが13.4%増加し、モバイルに送信されたJavaScriptの量が14.4%増加しています。

クライアント	2019	2020	変化
デスクトップ	391	444	13.4%
モバイル	359	411	14.4%

図2.3. 1 ページあたりの JavaScript キロバイト数の中央値の前年比変化。

少なくともこの重量の一部は不要なもののようです。任意のページロードでどのくらいのJavaScriptが未使用になっているかの内訳を見ると、中央値のページでは152KBの未使用

117. https://almanac.httparchive.org/ja/2019/javascript#how-much-javascript-do-we-use

JavaScriptが提供されていることがわかります。その数は75パーセンタイルで334KB、90パーセンタイルで567KBに跳ね上がっています。

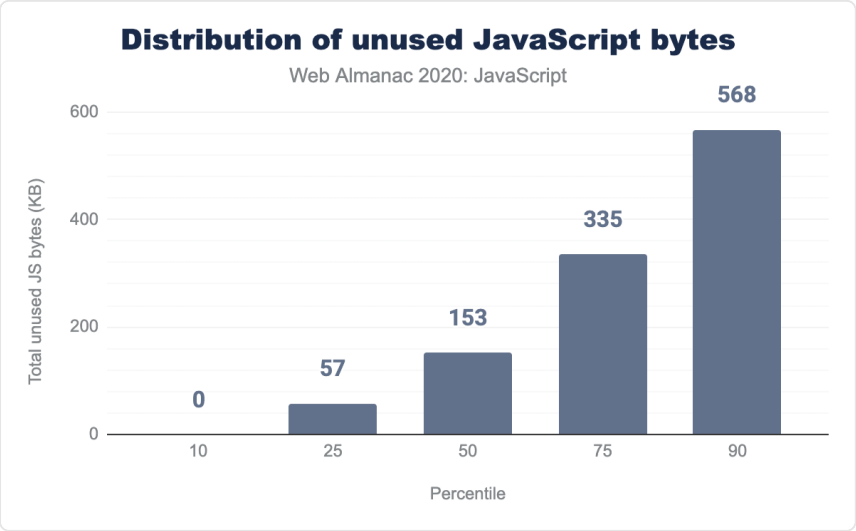


図2.4. モバイルページあたりのJavaScriptの無駄なバイト数の分布。

しかし、各ページで使用されているJavaScriptの全体のパーセンテージとして見ると、どれだけ無駄なものを送っているのかが少しわかりやすくなります。

37.22%

図2.5. 中央のモバイルページのJavaScriptバイトのうち、未使用のものの割合。

その153KBは、モバイルデバイスに送信するスクリプトの総サイズの約37%に相当します。ここには間違いなく改善の余地があります。

`module` と `nomodule`

送信するコードの量を減らす可能性のあるメカニズムの1つとして、`module` / `nomodule` パターンを利用することがあります。このパターンでは、モダンブラウザ向けのバンドルとレガシーブラウザ向けのバンドルの2つのセットを作成します。モダンブラウザ向けのバンドルは `type=module`、レガシーブラウザ向けのバンドルは `type=nomodule` となります。

このアプローチにより、サポートしているブラウザに最適化された最新の構文を持つ小さな

バンドルを作成できますが、そうでないブラウザには条件付きで読み込まれたポリフィルと異なる構文を提供します。

`module` や `nomodule` のサポートは広がっているが、まだ比較的新しいものです。その結果、採用率はまだ少し低いです。レガシーブラウザをサポートするために `type=module` のスクリプトを少なくとも1つ使っているのはモバイルページの3.6%に過ぎず、`type=nomodule` のスクリプトを少なくとも1つ使っているのはモバイルページの0.7%に過ぎません。

リクエスト数

私たちがどれだけのJavaScriptを使用しているかを見るもう1つの方法は、各ページでどれだけのJavaScriptリクエストが行われているかを調べることです。HTTP/1.1ではリクエスト数を減らすことがパフォーマンスを維持するための最優先事項でしたが、HTTP/2ではその逆です。JavaScriptを小さくて個別のファイル[118]に分解することが[通常はパフォーマンスを向上させる]のです../2019/http#impact-of-http2)。

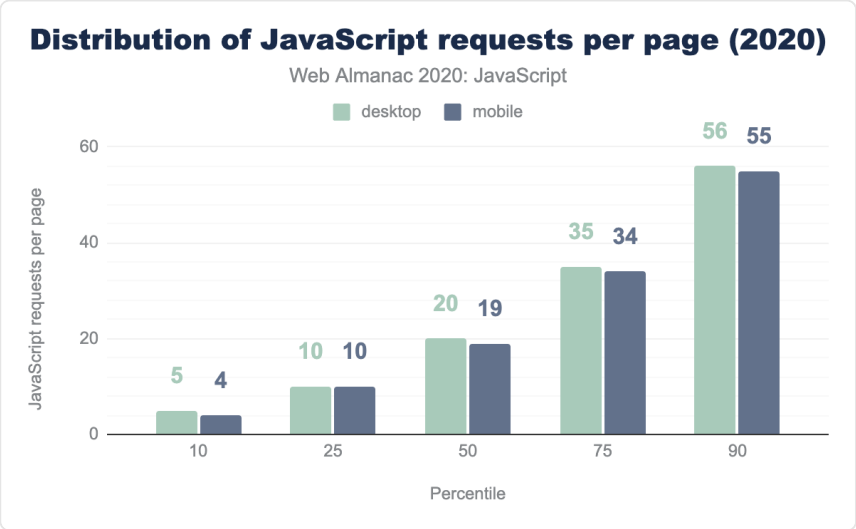


図2.6. 1ページあたりのJavaScriptリクエストの分布。

中央値では、ページは20回のJavaScriptリクエストを行います。これは中央値で19回のJavaScriptリクエストを行っていた昨年よりもわずかに増加しています。

118. https://web.dev/granular-chunking-nextjs/

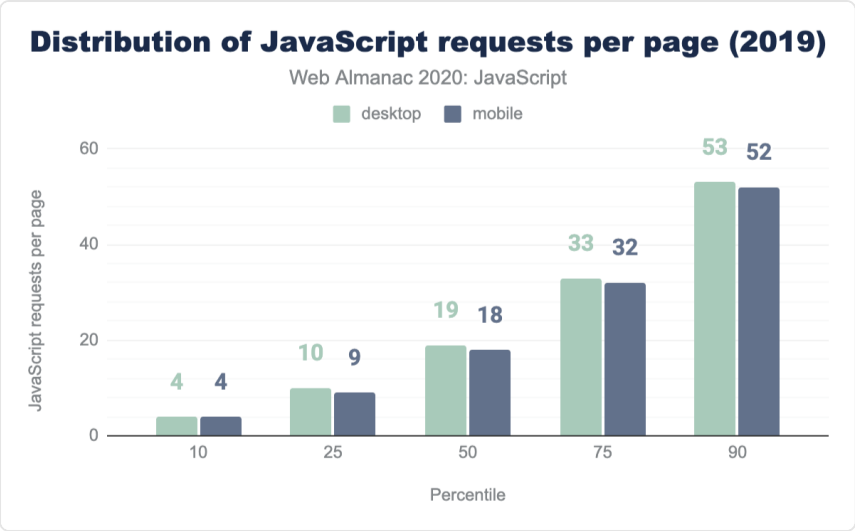


図2.7. 2019年の1ページあたりのJavaScriptリクエストの配信。

それはどこから来るのか？

ページで使用されるJavaScriptの増加の一因となっていると思われる傾向として、クライアント側のA/Bテストや分析から広告配信、パーソナライゼーションの処理まで、あらゆることを支援するためページに追加されるサードパーティ製のスクリプトの量が増え続けていることが挙げられます。

サードパーティのスクリプトがどれくらいあるのか、少し掘り下げてみましょう。

中央値までは、サードパーティのスクリプトとほぼ同じ数のファーストパーティのスクリプトを提供しています。中央値では、1ページあたり9個のファーストパーティのスクリプトがあるのに対し、サードパーティのスクリプトは1ページあたり10個となっています。サイトが提供しているスクリプトの数が多いほど、その大半がサードパーティ製のスクリプトである可能性が高くなります。

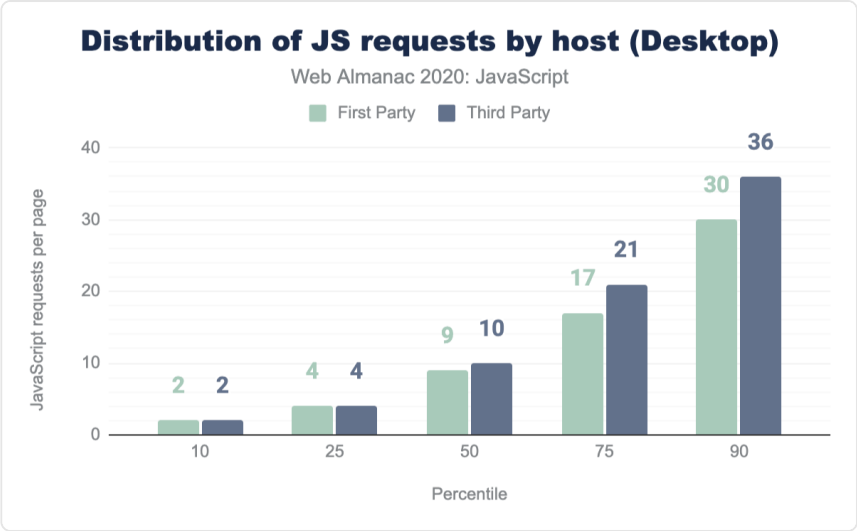


図2.8. デスクトップ用のホスト別のJavaScriptリクエスト数の分布。

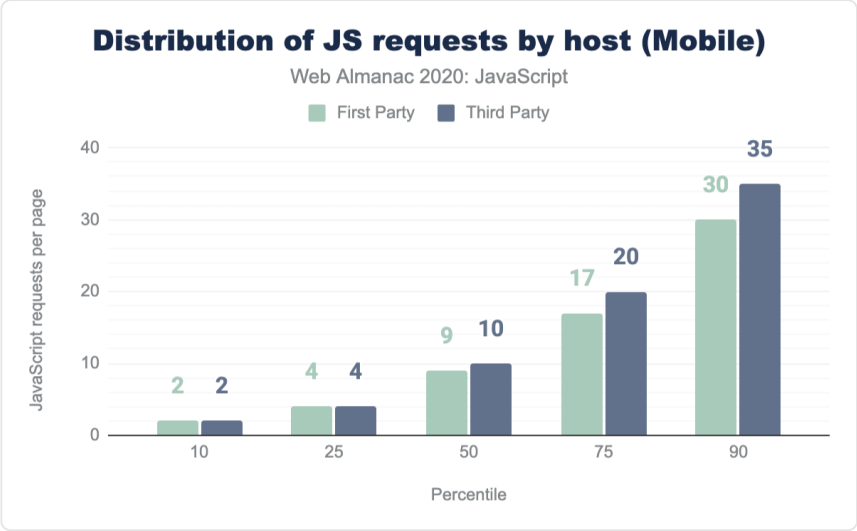


図2.9. モバイル向けのホスト別のJavaScriptリクエスト数の分布。

JavaScriptリクエストの量は中央値では同じようなものですが、実際のスクリプトのサイズは、サードパーティのソースの方が少し重くなっています（ダジャレを意図したものです）。中央値のサイトでは、サードパーティからデスクトップデバイスに267KBのJavaScriptを送信していますが、ファーストパーティからは147KBです。モバイルでも状況は非常によく似ており、中央のサイトがサードパーティのスクリプトを255KB送っているの

に対し、ファーストパーティのスクリプトは134KBです。

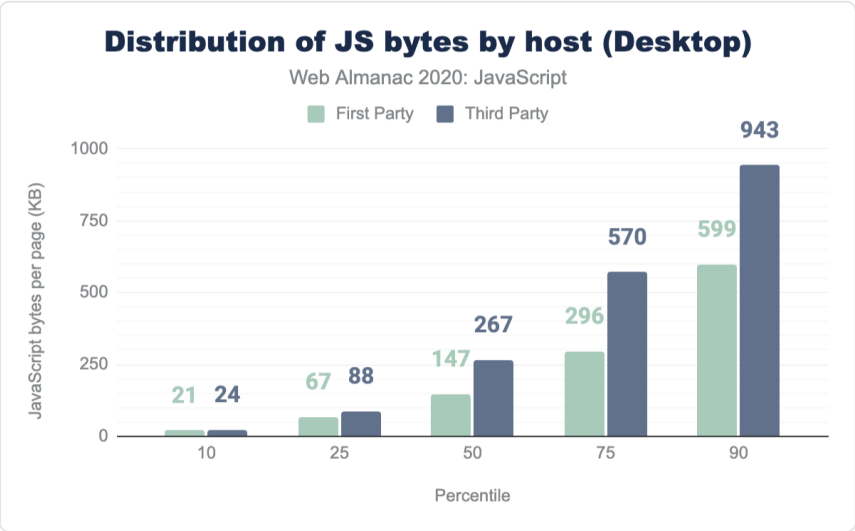


図2.10. デスクトップ用のホスト別のJavaScriptバイト数の分布。

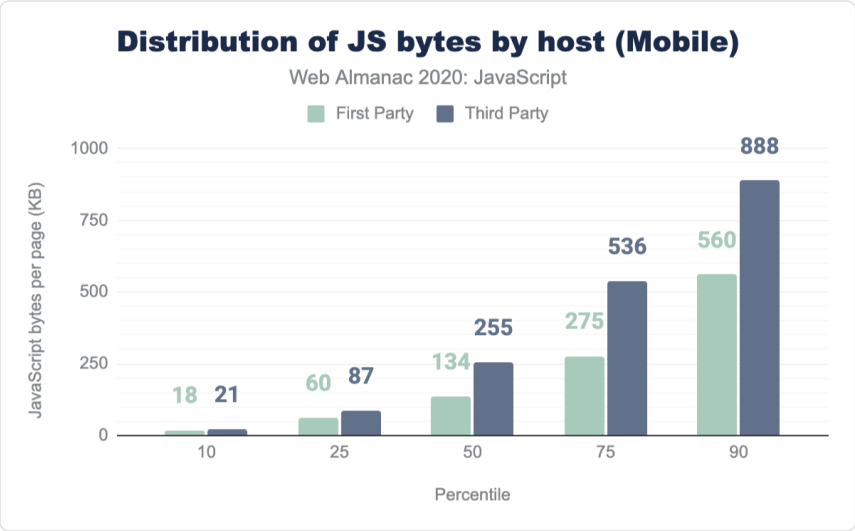


図2.11. モバイル向けのホスト別のJavaScriptバイト数の分布。

どうやって**JavaScript**を読み込むのか？

JavaScriptの読み込み方法は、全体的な体験に大きな影響を与えます。

デフォルトでは、JavaScriptは*parser-blocking*になっています。言い換えれば、ブラウザが`script`要素を発見したとき、スクリプトがダウンロードされ、解析され、実行されるまでHTMLの解析を一時停止しなければなりません。これは重要なボトルネックであり、レンダリングに時間がかかるページの一般的な原因となっています。

非同期（`async`属性を使用）にスクリプトを読み込むことで、JavaScriptの読み込みにかかるコストの一部を相殺できます。どちらの属性も外部スクリプトでのみ利用可能で、インラインスクリプトには適用できません。

モバイルでは外部スクリプトが、検出されたスクリプト要素全体の59.0%を占めています。

余談ですが、先ほどページに読み込まれるJavaScriptの量について話したとき、その合計にはこれらのインラインスクリプトのサイズは含まれていませんでした。つまり、数字が示す以上に多くのスクリプトが読み込まれているということです。

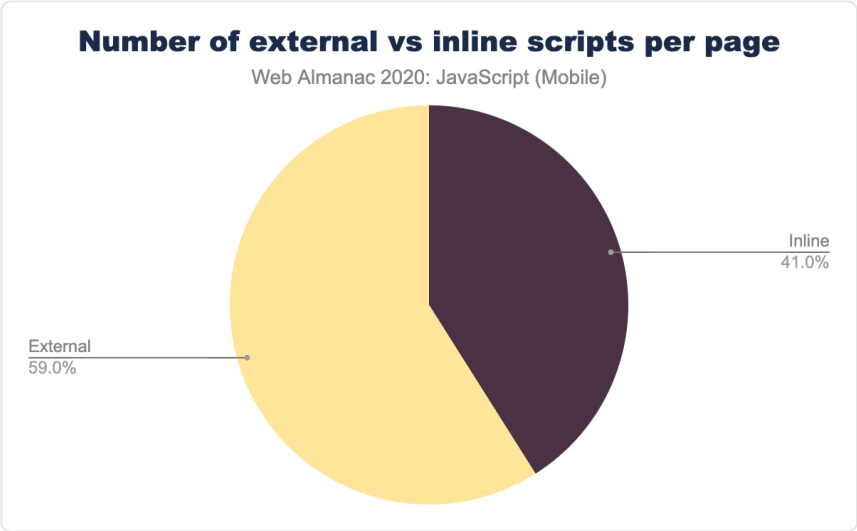


図2.12. モバイルページごとの外部スクリプトとインラインスクリプトの数の分布。

これらの外部スクリプトのうち、`async`属性でロードされたものは12.2%に過ぎず、`defer`属性でロードされたものは6.0%に過ぎない。

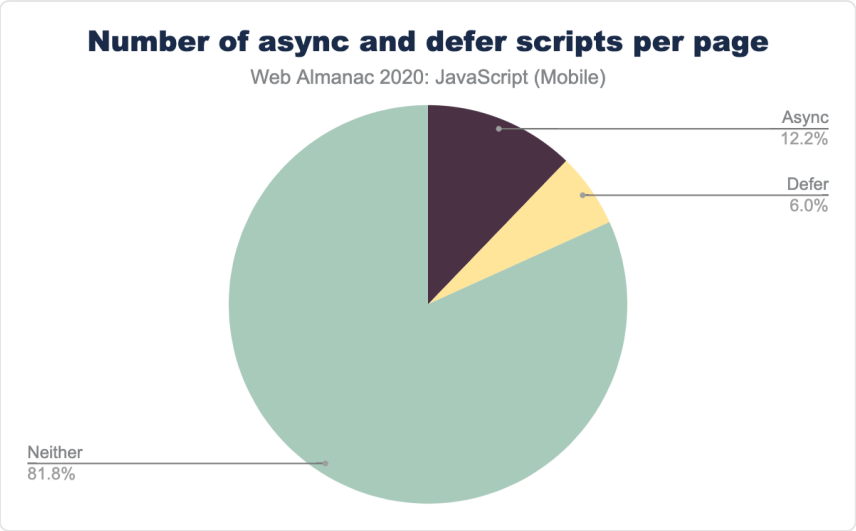


図2.13. モバイルページあたりの `async` および `defer` スクリプトの数の分布。

`defer` が最高のロードパフォーマンスを提供してくれることを考えると（スクリプトのダウンロードが他の作業と並行して行われ、ページを表示できるようになるまで実行が待たされることを保証することで）、その割合がもう少し高くなることを期待したいところです。実際には、6.0%というのは少し膨らんでいます。

IE8とIE9のサポートが一般的だった頃は、 `async` と `defer` 属性の両方を使うのが比較的一般的でした。両方の属性を使えば、両方をサポートしているブラウザは `async` を使うことになります。IE8とIE9は `async` をサポートしていないので、 `defer` 属性を使うようになります。

今日では、大多数のサイトではパターンは不要でありパターンが配置された状態で読み込まれたスクリプトはページが読み込まれるまで延期するのではなく、実行する必要があるときにHTMLパーサーを中断します。このパターンは今でも驚くほど頻繁に使用されており、モバイルページの11.4%がこのパターンを実装したスクリプトを少なくとも1つ提供しています。言い換えれば、 `defer` を使用している6%のスクリプトのうち、少なくとも一部は `defer` 属性の恩恵を十分に受けていないということです。

しかし、ここには心強い話があります。

Harry RobertsがTwitterでアンチパターンについてつぶやいていた[119]ということで、野生でどれくらいの頻度で発生しているのかを確認するように促されました。Rick Viscomiが調べて

119. https://x.com/csswizardry/status/1331721659498319873

みた[120]で、「stats.wp.com」が最も多くの違反者を出していたことが判明しました。Automatticの@Kraftさんから返信があり、パターンは現在削除済です[121]。

ウェブのオープン性についての素晴らしい点の1つは、1つの観察がいかに意味のある変化につながるかということであり、それはまさにここで起こったことです。

リソースのヒント

JavaScriptの読み込みにかかるネットワークコストの一部を相殺するためのもう1つのツールとしてのリソースヒント、具体的には`prefetch`と`preload`があります。

ヒント`prefetch`は、次のページのナビゲーションでリソースが使用されることを開発者に示すものです。

`preload`ヒントは、現在のページでリソースが利用されることを示し、ブラウザはそのリソースをより高い優先度ですぐにダウンロードすべきであることを示します。

全体では、モバイルページの16.7%が、JavaScriptをより積極的に読み込むための2つのリソースヒントのうち少なくとも1つを使用していることがわかります。

そのうち、ほぼすべての利用は`preload`に由来しています。モバイルページの16.6%がJavaScriptを読み込むために少なくとも1つの`preload`ヒントを使用しているのに対し、モバイルページでは少なくとも1つの`prefetch`ヒントを使用しているのは0.4%に過ぎません。

特に`preload`の場合、ヒントを多用しすぎて効果が低下するリスクがあるので、ヒントを使用しているページを見て、そのページがどれだけの数のヒントを使用しているかを確認する価値があります。

120. https://x.com/rick_viscomi/status/1331735748060524551
121. https://x.com/Kraft/status/1336772912414601224

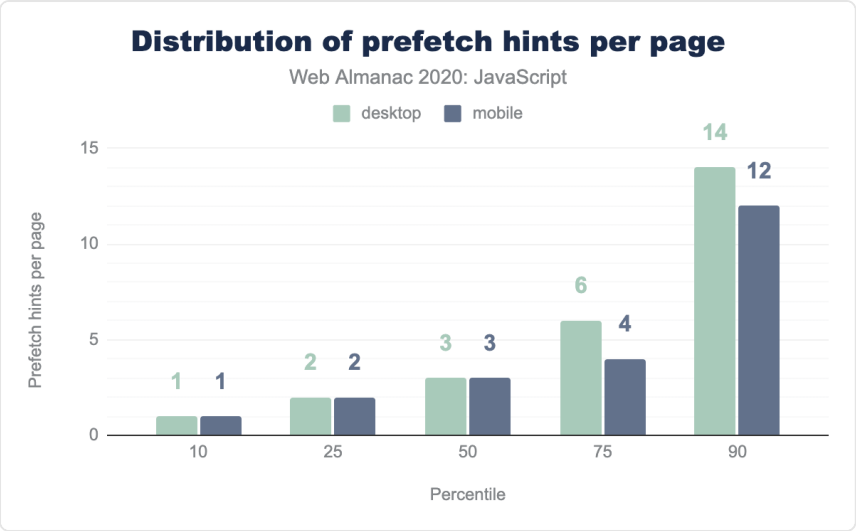


図2.14. 1ページあたりの`prefetch`ヒントの数の分布。

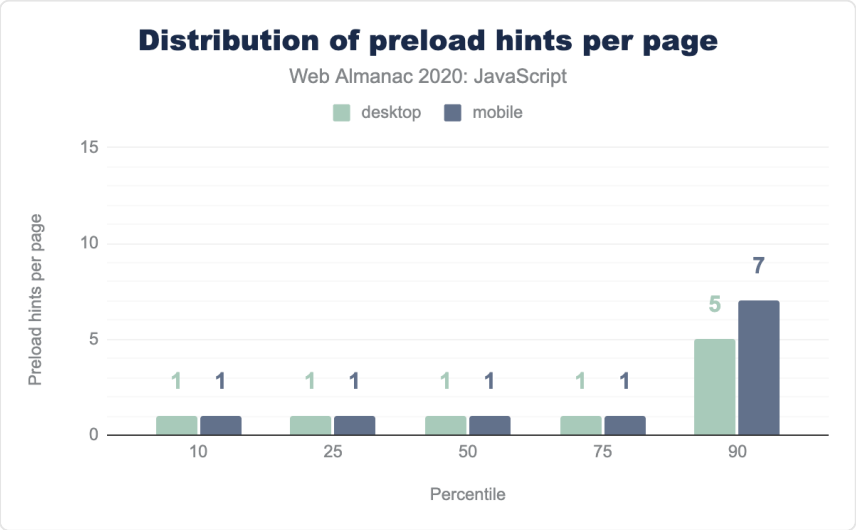


図2.15. 任意の`preload`ヒントを持つページあたりの`preload`ヒントの数の分布。

中央値では、`prefetch`ヒントを使ってJavaScriptを読み込むページで3つのヒントが使われていますが、`preload`ヒントを使っているページでは1つしか使われていません。90パーセンタイル台では12個の`prefetch`ヒントが使用され、90パーセンタイル台では7個の`preload`ヒントが使用されています。リソースヒントの詳細については、今年の Resource Hints の章を参照してください。

どのようにJavaScriptを提供するのか？

ウェブ上のあらゆるテキストベースのリソースと同様に、最小化と圧縮によってかなりのバイト数を節約できます。これらの最適化はどちらも新しいものではなく、かなり前から行われていたものなので、適用されていないケースが多いと予想されます。

Lighthouseの監査では、未定義のJavaScriptをチェックし、その結果に基づいてスコア（0.00が最悪、1.00が最高）を提供しています。

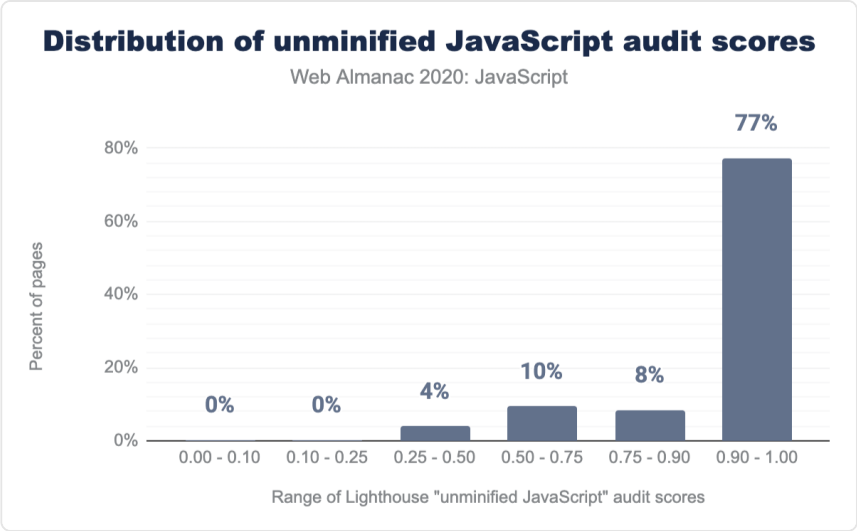


図2.16. 未定義のJavaScript Lighthouseの監査スコアをモバイルページごとに配布しています。

上のチャートを見ると、ほとんどのページが0.90以上のスコアを取得している（77%）ことがわかりますが、これは未完のスクリプトがほとんどないことを意味します。

全体では、記録されたJavaScriptリクエストのうち、4.5%しか未処理のものはありません。

興味深いことに、サードパーティのリクエストを少し取り上げましたが、これはサードパーティのスクリプトがファーストパーティのスクリプトよりもうまくいっている分野の1つです。平均的なモバイルページの統一されてないJavaScriptバイトの82%はファーストパーティのコードから来ています。

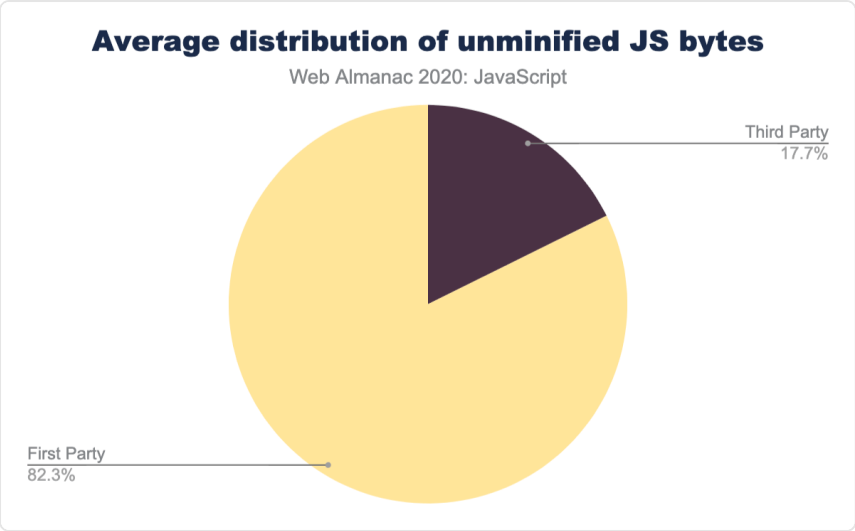


図2.17. ホスト別の最小化されていないJavaScriptのバイト数の平均分布。

圧縮

縮小化はファイルサイズの削減に役立つ優れた方法ですが、圧縮はさらに効果的であり、それゆえより重要です。

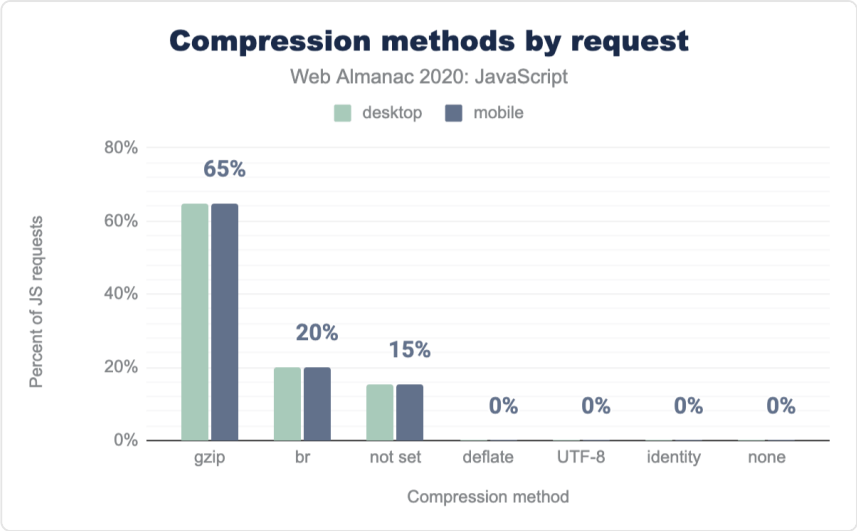


図2.18. 圧縮方法別のJavaScriptリクエストの割合の分布。

すべてのJavaScriptリクエストの85%には、何らかのレベルのネットワーク圧縮が適用されています。Gzipがその大部分を占めており、スクリプトの65%にGzip圧縮が適用されているのに対し、Brotli(br)は20%です。Brotli（Gzipよりも効果が高い）の割合は、ブラウザのサポートに比べれば低いですが、正しい方向に向かっており、昨年より5%もポイント増加しています。

繰り返しになりますが、これはサードパーティ製スクリプトがファーストパーティ製スクリプトよりも実際にうまくいっている分野のようです。圧縮方法をファーストパーティスクリプトとサードパーティスクリプトで分けてみると、サードパーティスクリプトの24%がBrotliを適用しているのに対し、ファーストパーティスクリプトでは15%しか適用されていません。

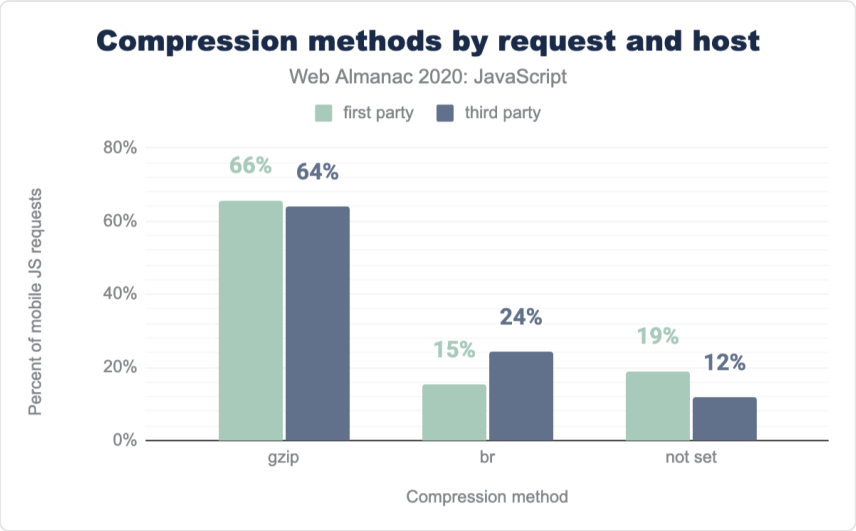


図2.19. モバイルJavaScriptリクエストの圧縮方法とホスト別の割合の分布。

サードパーティ製のスクリプトは、圧縮を全く行わずに提供される可能性も低くなっています。サードパーティ製のスクリプトの12%がGzipもBrotliも適用されていないのに対し、ファーストパーティ製のスクリプトは19%です。

圧縮が適用されていないスクリプトをよく見てみる価値はあります。圧縮は、作業しなければならない内容が多いほど、節約の面で効率的になります。つまりファイルが極小であれば、ファイルを圧縮するためのコストが、ファイルサイズの極小化に勝るとも劣らないことがあります。

90.25%

図2.20. 5KB未満のサードパーティ製JavaScriptの非圧縮リクエストの割合。

ありがたいことに、特にサードパーティスクリプトでは、非圧縮スクリプトの90%が5KB未満のサイズであることが、まさにそれを示しています。一方、非圧縮のファーストパーティスクリプトの49%は5KB未満で、非圧縮のファーストパーティスクリプトの37%は10KBを超えています。このように、小さな非圧縮ファーストパーティ スクリプトが多く見られますが、ある程度圧縮することで恩恵を受けることができるものはまだかなりの数があります。

何を使うのか？

サイトやアプリケーションにJavaScriptを使用する機会が増えるにつれ、開発者の生産性やコードの保守性を向上させるためのオープンソースのライブラリやフレームワークへの需要も増えてきました。これらのツールを使用していないサイトは、今日のウェブでは少数派であることは間違いありません。

ウェブを構築するために使用するツールとそのトレードオフは何かを批判的に考えることが重要なので、現在使用されているものをよく見ることは理にかなっています。

ライブラリ

HTTP ArchiveはWappalyzer を使用して、特定のページで使用されている技術を検出します。WappalazyerはJavaScriptライブラリ（jQueryのような開発を容易にするためのスニペットやヘルパー関数の集合体と考えてください）とJavaScriptフレームワーク（Reactのようなテンプレートや構造を提供する足場となる可能性が高いものです）の両方を追跡します。

使用されている人気のあるライブラリは昨年とほとんど変わっておらず、jQueryが引き続き圧倒的に使用されており、上位21のライブラリのうち1つだけが脱落しています（lazy.jsはDataTablesに置き換わっています）。実際、上位のライブラリの割合も昨年とほとんど変わっていません。

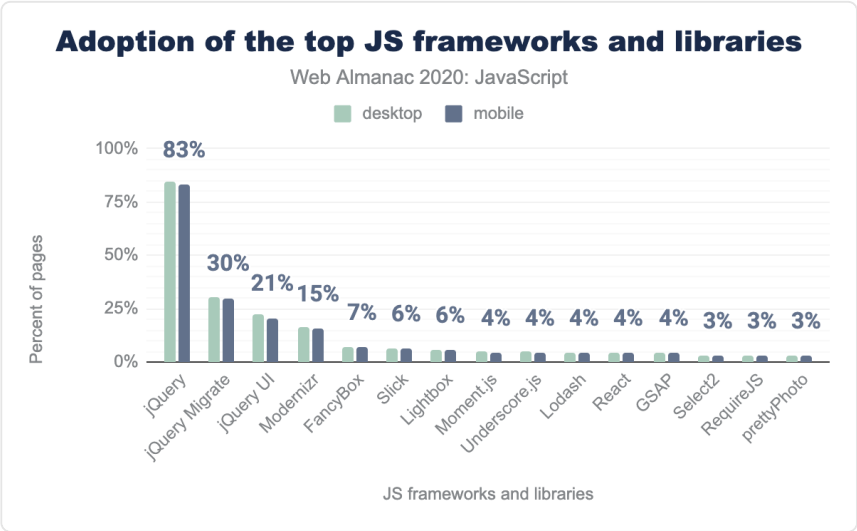


図2.21. ページに占めるトップJavaScriptフレームワークとライブラリの採用率。

昨年、Houssein posited some reasons why jQuery's dominance continues[122]を発表しました。

> *3割以上のサイトで利用されているWordPressには、デフォルトでjQueryが含まれています。 アプリケーションの規模によってはjQueryから新しいクライアントサイドライブラリへの切り替えに時間がかかる場合があり、多くのサイトでは新しいクライアントサイドライブラリに加えてjQueryで構成されている場合があります。*

どちらも非常に健全な推測であり、どちらのフロントでも状況はあまり変わっていないようです。

実際、上位10のライブラリのうち、jQuery UI、jQuery Migrate、FancyBox、Lightbox、Slickの6つがjQueryであるか、使用するためにjQueryを必要としていることを考えると、jQueryの優位性はさらに裏付けられています。

フレームワーク

フレームワークを見てみると、昨年注目された主要なフレームワークでも、採用率という点であまり劇的な変化は見られません。Vue.jsは大幅に増加し、AMPは少し成長しましたが、ほとんどのフレームワークは1年前とほぼ同じ位置にあります。

昨年指摘された問題が今も適用されていることは注目に値し[123]、ここでも結果に影響を与えます。これらのツールの人気に大きな変化があった可能性はありますが、現在のデータ収集方法ではそれが見られないだけです。

それが何を意味するのか

ツール自体の人気よりも面白いのは、作ったものにインパクトがあることです。

まず、1つのツール対1つのツールの使い方を考えても、実際には1つのライブラリやフレームワークだけを使って制作することはほとんどありません。分析されたページのうち、わずか21%のページが1つのライブラリまたはフレームワークのみを報告しています。フレームワークが2つ、3つというのはかなり一般的で、ロングテールはあっという間に長くなってしまいます。

制作現場でよく見かける組み合わせを見てみると、そのほとんどが予想通りの組み合わせになっています。jQueryの優位性を知っていれば、人気のある組み合わせのほとんどは、jQueryとjQuery関連のプラグインをいくらでも含んでいるのは当然のことです。

122. *https://almanac.httparchive.org/ja/2019/javascript#open-source-libraries-and-frameworks*
123. *https://github.com/AliasIO/wappalyzer/issues/2450*

組み合わせ	ページ数	(%)
jQuery	1,312,601	20.7%
jQuery, jQuery Migrate	658,628	10.4%
jQuery, jQuery UI	289,074	4.6%
Modernizr, jQuery	155,082	2.4%
jQuery, jQuery Migrate, jQuery UI	140,466	2.2%
Modernizr, jQuery, jQuery Migrate	85,296	1.3%
FancyBox, jQuery	84,392	1.3%
Slick, jQuery	72,591	1.1%
GSAP, Lodash, React, RequireJS, Zepto	61,935	1.0%
Modernizr, jQuery, jQuery UI	61,152	1.0%
Lightbox, jQuery	60,395	1.0%
Modernizr, jQuery, jQuery Migrate, jQuery UI	53,924	0.8%
Slick, jQuery, jQuery Migrate	51,686	0.8%
Lightbox, jQuery, jQuery Migrate	50,557	0.8%
FancyBox, jQuery, jQuery UI	44,193	0.7%
Modernizr, YUI	42,489	0.7%
React, jQuery	37,753	0.6%
Moment.js, jQuery	32,793	0.5%
FancyBox, jQuery, jQuery Migrate	31,259	0.5%
MooTools, jQuery, jQuery Migrate	28,795	0.5%

図2.22. モバイルページでよく使われるライブラリやフレームワークの組み合わせ。

また、React、Vue、Angularなどの「モダン」なフレームワークがjQueryとペアリングされているのも、サードパーティによる移行や組み込みなどの結果として、かなりの量を目にすることがあります。

組み合わせ	jQueryなし	jQueryあり
GSAP, Lodash, React, RequireJS, Zepto	1.0%	
React, jQuery		0.6%
React	0.4%	
React, jQuery, jQuery Migrate		0.4%
Vue.js, jQuery		0.3%
Vue.js	0.2%	
AngularJS, jQuery		0.2%
GSAP, Hammer.js, Lodash, React, RequireJS, Zepto	0.2%	
合計	**1.7%**	**1.4%**

図2.23. React、Angular、VueのjQueryの有無で最も人気のある組み合わせです。

さらに重要なことは、これらのツールはすべて、より多くのコードとより多くの処理時間を意味します。

使用されているフレームワークを具体的に見てみると、それらを使用しているページのJavaScriptのバイト数の中央値は、*what*が使用されているかによって大きく異なることがわかります。

以下のグラフは、最も一般的に検出されたフレームワークのトップ35のいずれかが検出されたページのバイト数の中央値をクライアント別に示しています。

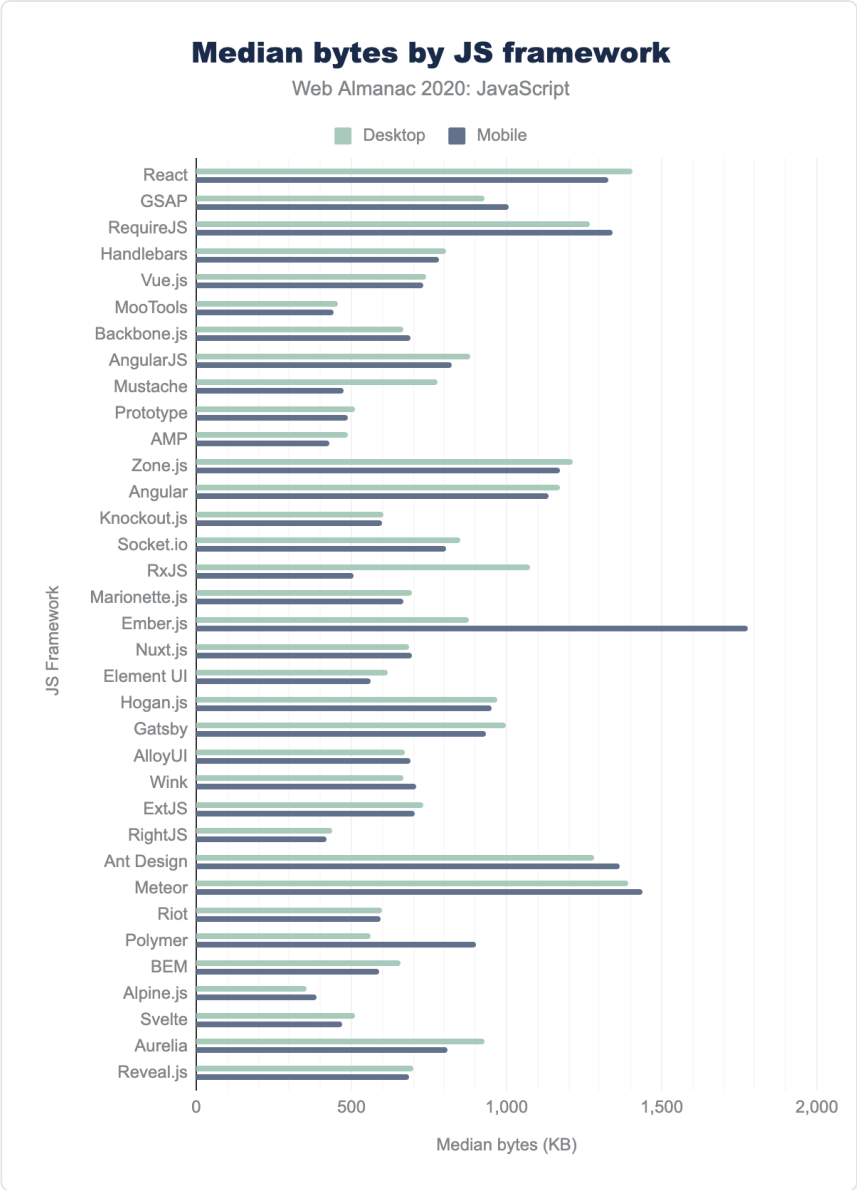


図2.24. JavaScriptフレームワークごとの1ページあたりのJavaScriptのキロバイト数の中央値。

ReactやAngular、Emberのようなフレームワークは、クライアントに関係なく多くのコードを提供する傾向があります。一方、Alpine.jsやSvelteのようなミニマリストフレームワーク

は、非常に有望な結果を示しています。デフォルトは非常に重要で、パフォーマンスの高いデフォルトから始めることで、SvelteもAlpineも（今のところサンプルサイズはかなり小さいですが）軽いページを作ることに成功しているようです。

これらのツールが検出されたページのメインスレッドの時間を見ると、非常に似たような画像が得られます。

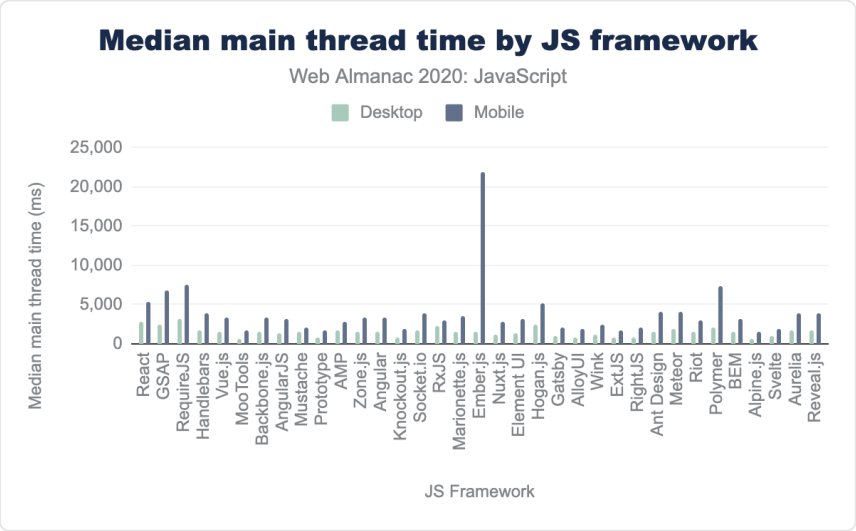


図2.25. JavaScriptフレームワークによる1ページあたりのメインスレッド時間の中央値。

Emberのモバイルメインスレッドの時間が飛び出して、どれくらいの時間がかかるかでグラフが歪んでしまいます。（これについてもう少し時間をかけて調べてみましたが、Ember自体に根本的な問題があるというよりも、このフレームワークを非効率的に使用しているある特定のプラットフォームに大きく影響されている[124]ようです）。それを引き出すことで、絵が少し理解しやすくなります。

124. *https://timkadlec.com/remembers/2021-01-26-what-about-ember/*

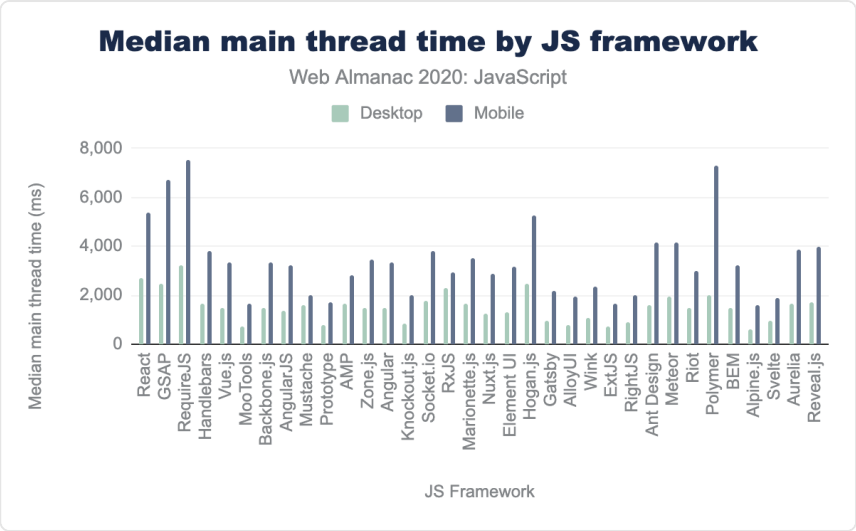


図2.26. Ember.jsを除くJavaScriptフレームワーク別の1ページあたりのメインスレッド時間の中央値。

ReactやGSAP、RequireJSなどのツールは、デスクトップやモバイルのページビューに関わらず、ブラウザのメインスレッドに多くの時間を費やす傾向があります。アルパインやスベルテのようなツールは、全体的にコードが少なくなる傾向はありますが、同じツールでもメインスレッドへの影響が少なくなる傾向はあります。

フレームワークがデスクトップとモバイルに提供する経験のギャップもまた、掘り下げる価値があります。モバイルトラフィックはますます支配的になってきており、私たちのツールがモバイルページビューに対して可能な限りのパフォーマンスを発揮することが重要です。フレームワークのデスクトップとモバイルのパフォーマンスのギャップが大きければ大きいほど、危険信号は大きくなります。

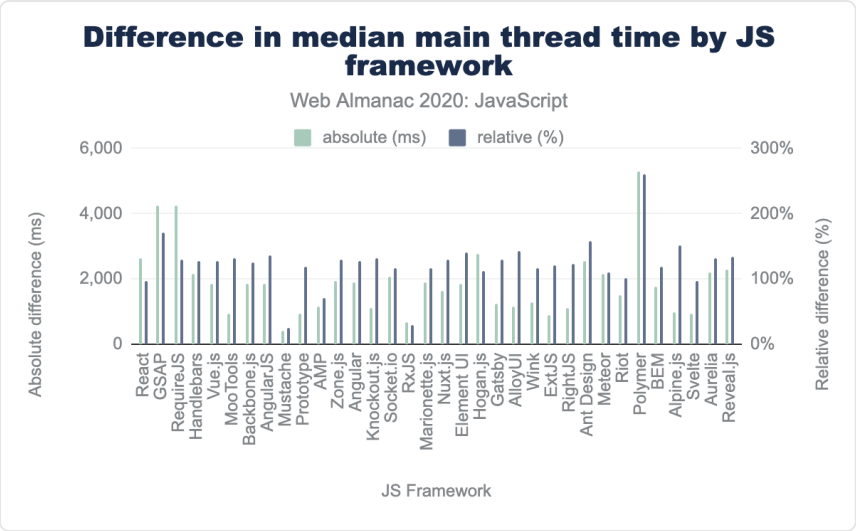


図2.27. デスクトップとモバイルの1ページあたりのメインスレッド時間の中央値の差 Ember.jsを除くJavaScriptフレームワーク別。

さすがにエミュレートされたMoto G4の処理能力が低いため、使用しているすべてのツールにギャップがあります。EmberとPolymerは特に忌まわしい例として飛び出ているように見えますが、RxJSやMustacheのようなツールはデスクトップとモバイルの間でわずかな違いしかありません。

何の影響があるの？

私たちは今JavaScriptをどのくらい使っているのか、どこから来ているのか、何のために使っているのか、かなり良い画像を持っています。それだけでも十分に興味深いのですが、本当のキッカケは「だから何？」このスクリプトが実際にページの体験にどのような影響を与えるのか？

まず最初に考えるべきことは、ダウンロードされたJavaScriptがどうなるかということです。ダウンロードはJavaScriptの旅の最初の部分に過ぎません。ブラウザはスクリプトをすべて解析してコンパイルし、最終的に実行しなければなりません。ブラウザはそのコストの一部を他のスレッドにオフロードする方法を常に模索していますが、その作業の多くは依然としてメインスレッドで行われており、ブラウザはレイアウトやペイント関連の作業を行うことができずユーザーのインタラクションに応答することもできません。

思い起こせば、モバイルデバイスに提供されるものとデスクトップデバイスに提供されるものの間には、わずか30KBの差しかありませんでした。考え方にもよりますが、デスクトッ

プブラウザとモバイルブラウザに送信されるコードの量にわずかな差があることにあまり動揺しないということは許されるかもしれません。

最大の問題はそのコードがすべてローエンドからミドルエンドのデバイスへ提供されたときに生じるもので、ほとんどの開発者が持っている可能性のあるデバイスではなく、世界中の大多数の人が目にするであろうデバイスのようなものです。デスクトップとモバイルの間のこの比較的小さな差は、処理時間の面で見るとはるかに劇的です。

デスクトップサイトの中央値は、JavaScriptのすべての作業を行うブラウザのメインスレッドに891msを費やしています。しかし、モバイルサイトの中央値は1,897ミリ秒と、デスクトップの2倍以上の時間を費やしています。ロングテールのサイトの方がよっぽどヤバい。90パーセンタイルで、モバイルサイトではJavaScriptを扱うメインスレッドの時間が8,921ミリ秒であるのに対し、デスクトップサイトでは3,838ミリ秒であるという驚異的な結果が出ています。

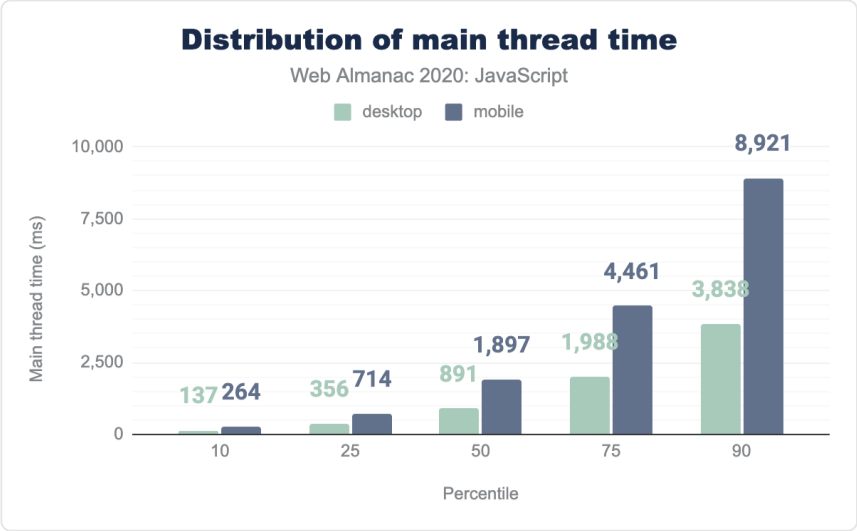


図2.28. メインスレッドの時間配分。

JavaScriptの利用とLighthouseのスコアリングの相関関係

これがどのようにユーザー体験に影響を与えるかを見る方法の1つとして、先に確認したJavaScriptのメトリクスのいくつかと、さまざまなメトリクスやカテゴリのLighthouseのスコアを関連付けることを試してみることがあります。

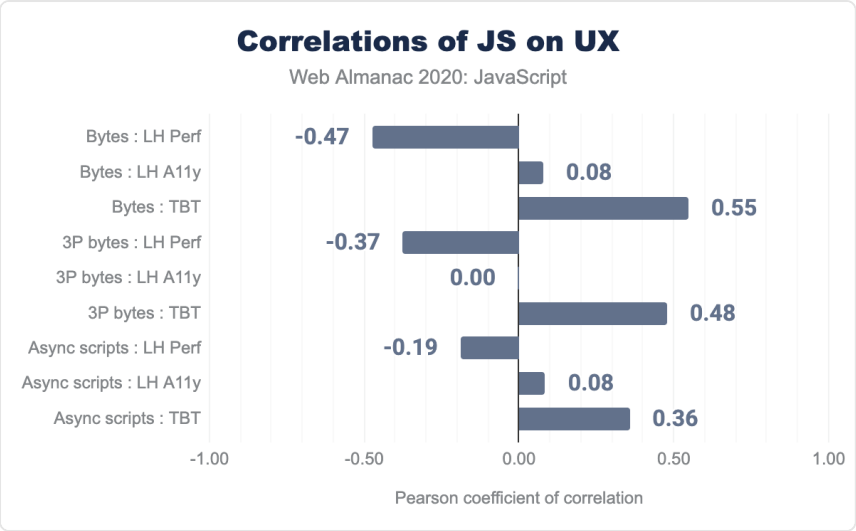


図2.29. ユーザー体験の様々な側面におけるJavaScriptの相関関係。

上のグラフはピアソン相関係数[125]を使用しています。正確には何を意味するのか、長くてちょっと複雑な定義がありますが、要は2つの異なる数字の間の相関の強さを探しているということです。係数が1.00だとすると、正の相関関係があることになります。0.00の相関関係は、2つの数字の間に何の関係もないことを示しています。0.00以下のものは、負の相関関係、つまり、ある数字が上がるともう1つの数字が下がることを示しています。

まず、JavaScriptのメトリクスとLighthouseのアクセシビリティ（チャートでは「LH A11y」）のスコアの間には、測定可能な相関関係はあまりないようです。これは、特にWebAimの年次調査[126]など、他の場所で発見されていることとは全く対照的です。

これについて最も可能性の高い説明は、Lighthouseのアクセシビリティテストが、WebAIMのようなアクセシビリティを主眼とした他のツールで利用可能なものほど包括的ではないということです（まだです！）。

強い相関関係が見られるのは、JavaScriptのバイト数（「バイト数」）とLighthouseの総合的なパフォーマンス（「LH Perf」）スコアと総ブロッキング時間（「TBT」）の間です。

JavaScriptのバイト数とLighthouseのパフォーマンススコアの相関関係は-0.47です。つまり、JSのバイト数が増えるとLighthouseのパフォーマンススコアは低下します。全体のバイト数は、サードパーティのバイト数（「3Pバイト」）よりも強い相関関係があります。

総ブロッキング時間とJavaScriptのバイト数の関係はさらに重要です (全体のバイト数では

125. https://ja.wikipedia.org/wiki/%E7%9B%B8%E9%96%A2%E4%BF%82%E6%95%B0
126. https://webaim.org/projects/million/#frameworks

0.55、サードパーティのバイト数では0.48)。これは、ブラウザがページ内でJavaScriptを実行させるためにブラウザが行わなければならないすべての作業について知っていることを考えれば、それほど驚くべきことではありません。

セキュリティの脆弱性

Lighthouseが実行するもう1つの有益な監査は、サードパーティのライブラリに既知のセキュリティ脆弱性がないかどうかをチェックすることです。これは、あるページでどのライブラリやフレームワークが使われているか、またそれぞれのバージョンを検出することで行います。次に、Snyk's open-source vulnerability database[127]をチェックして、特定されたツールにどのような脆弱性が発見されているかを確認します。

83.50%

図2.30. モバイルページの割合は、少なくとも1つの脆弱性のあるJavaScriptライブラリを含んでいます。

監査によると、モバイルページの83.5%が、少なくとも1つの既知のセキュリティ脆弱性を持つJavaScriptのライブラリやフレームワークを使用しています。

これがjQuery効果と呼ばれるものです。jQueryが83%ものページで使用されていることを覚えていますか？　いくつかの古いバージョンのjQueryには既知の脆弱性が含まれており、この監査でチェックされる脆弱性の大部分を占めています。

約500万ほどのモバイルページのうち、81%に脆弱性のあるバージョンのjQueryが含まれており、2番目によく見られる脆弱性のあるライブラリjQuery UIの15.6%を大きく引き離しています。

127. https://snyk.io/vuln?type=npm

ライブラリ	脆弱なページ
jQuery	80.86%
jQuery UI	15.61%
Bootstrap	13.19%
Lodash	4.90%
Moment.js	2.61%
Handlebars	1.38%
AngularJS	1.26%
Mustache	0.77%
Dojo	0.58%
jQuery Mobile	0.53%

図2.31. Lighthouseによると、脆弱性のあるモバイルページの数が最も多いことに貢献しているライブラリトップ10。

言い換えれば、時代遅れで脆弱性のあるバージョンのjQueryから人々を移行させることができれば、既知の脆弱性を持つサイトの数は激減するでしょう（少なくとも新しいフレームワークで脆弱性を見つけ始めるまでは）。

発見された脆弱性の大部分は、「中程度」の深刻度カテゴリに分類されます。

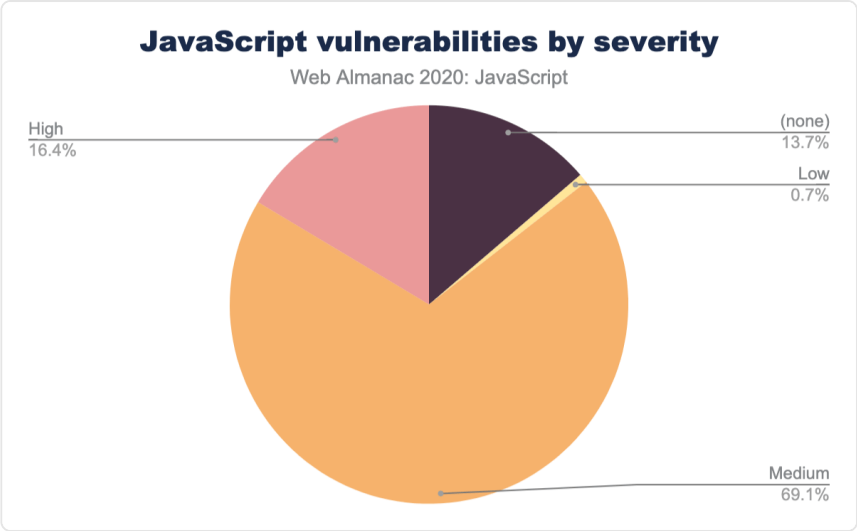


図2.32. JavaScriptの脆弱性を持つモバイルページの割合を深刻度別に分布。

結論

JavaScriptの人気は着実に上昇しており、それはポジティブなことがたくさんあります。数年前には想像もできなかったようなことが、JavaScriptのおかげで今日のウェブ上でできるようになったことを考えると、信じられないことです。

しかし、私たちも慎重に行動しなければならないことは明らかです。JavaScriptの使用量は毎年一貫して増加しています（株式市場がこれほど予測可能であれば私たちは皆、信じられないほど裕福になっているでしょう）が、それにはトレードオフが伴います。JavaScriptの増加は処理時間の増加と関連しており、総ブロッキング時間のような主要な指標に悪影響を及ぼします。また、これらのライブラリを更新せずに放置しておくと、既知のセキュリティ上の脆弱性を利用してユーザーを危険にさらすことになります。

ページに追加するスクリプトのコストを慎重に検討しツールに批判的な目を向けて、より多くの質問をすることがアクセスしやすく、パフォーマンスが高く、安全なウェブを構築するための最善の策です。

著者

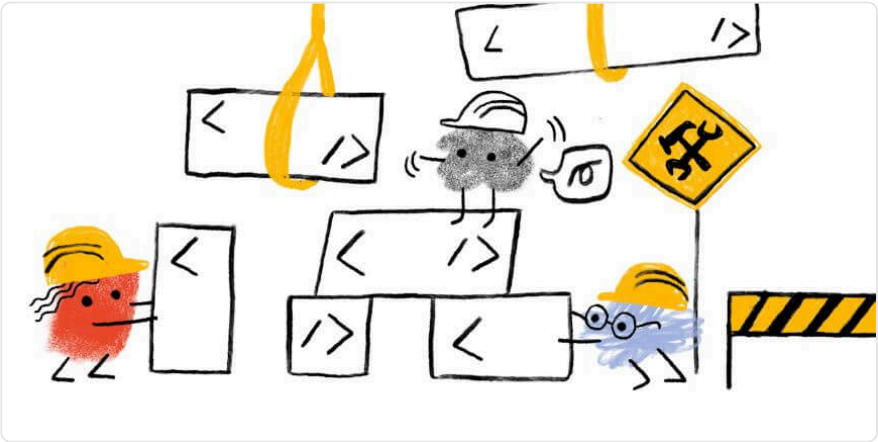
Tim Kadlec

@tkadlec　tkadlec　https://timkadlec.com/

Timは、誰もが使えるWebを構築することに焦点を当てたWebパフォーマンスコンサルタントであり、トレーナーです。著書に『High Performance Images』（O'Reilly, 2016）、『Implementing Responsive Design』（New Riders, 2012）があります。彼はtimkadlec.com[128]でウェブ全般について書いています。@tkadlecで、彼の考えを簡潔にまとめてTwitterで紹介しています。

128. *https://timkadlec.com/*

部 I 章 3
マークアップ



Jens Oliver Meiert、 Catalin Rosu と Ian Devlin によって書かれた。
Simon Pieters、 Manuel Matuzović と Brian Kardell によってレビュ。
Tony McCreath による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

ウェブはHTMLで構築されています。HTMLがなければウェブページも、ウェブサイトも、ウェブアプリケーションもありません。何もないのです。平文のドキュメントやXMLツリーは、このような特別な挑戦を楽しんでいる平行世界にあるかもしれません。この世界では、HTMLがユーザーフレンドリーなウェブの基礎となっています。ウェブの基盤となっている標準は数多くありますが、HTMLはもっとも重要な標準のひとつであることは間違いありません。

HTMLをどのように使うのかでは、どれだけ素晴らしい基盤があるのか。2019年版マークアップの章[129]の序章で、著者のBrian Kardell[130]は、長い間本当は分かっていなかったことを示唆しています。小さなサンプルはいくつかありました。たとえばIan Hicksonの研究[131]（モダン

129. https://almanac.httparchive.org/ja/2019/markup#introduction
130. https://almanac.httparchive.org/ja/2019/contributors#bkardell
131. https://web.archive.org/web/20060203035414/http://code.google.com/webstats/index.html

HTMLの親玉の1つ）などがありましたが、昨年まで開発者である私たち、著者である私たちが、HTMLをどのように利用しているのかについて主要な洞察を欠いていました。2019年には、Catalin Rosuの研究[132]（本章の共著者の1人）と2019年版Web Almanacの両方があり、実際のHTMLについて再び良い見解を得ることができました。

昨年は580万ページを分析し、そのうち440万ページをデスクトップで、530万ページをモバイルでテストしました。今年は2020年にユーザーが訪れるWebサイトに関する最新データを用いて750万ページを分析し、そのうち560万ページをデスクトップで、630万ページをモバイルでテストしました。昨年との比較も行っていますが、新たな洞察のために追加の指標を分析したのと同様に、私たち自身の個性や視点を章全体に付与しました。

このマークアップの章では、同じマークアップ言語であるSVGやMathMLではなく、ほぼHTMLのみを取り上げています。特に断りのない限り、この章で紹介する統計は、モバイルページのセットを指します。また、Web Almanacの全章のデータは公開されており、利用可能です。結果[133]を見て、あなたの意見[134]をコミュニティに伝えてください！

全般

このセクションではドキュメントタイプやドキュメントのサイズ、さらにはコメントやスクリプトの使用など、HTMLのより高度な側面を取り上げています。”生きたHTML“は、まさに生きているのです

Doctypes

96.82%

図3.1. doctypeを持つページの割合。

96.82%のページが*doctype*を宣言しています。HTML文書がdoctypeを宣言することは、Ian Hickson氏が2009年に説明したように[135]、「ブラウザのquirksモードのトリガーを避けるため」という歴史的な理由から有用です。もっとも人気のある値は何ですか？

132. *https://www.advancedwebranking.com/html/*
133. *https://docs.google.com/spreadsheets/d/1Ta7amoUeaL4pILhWzH-SCzMX9PsZeb1x_mwrX2C4eY8/*
134. *https://discuss.httparchive.org/t/2039*
135. *https://lists.w3.org/Archives/Public/public-html-comments/2009Jul/0020.html*

Doctype	ページ	ページ(%)
HTML ("HTML5")	5,441,815	85.73%
XHTML 1.0 Transitional	382,322	6.02%
XHTML 1.0 Strict	107,351	1.69%
HTML 4.01 Transitional	54,379	0.86%
HTML 4.01 Transitional (quirky[136])	38,504	0.61%

図3.2. もっとも人気のある5つのdoctypes。

XHTML1.0以降、数がかなり減り、いくつかの標準的なもの、いくつかの難解なもの、さらにはインチキなdoctypeを含むロングテールに入ることはすでにおわかりでしょう。

この結果から、2つの点が注目されます。

1. 生きたHTML[137]（通称「HTML5」）の発表から約10年、生きたHTMLは明らかに当たり前のものになりました。
2. 生きたHTML以前のウェブは、XHTML1.0のような、次によく使われるdoctypにまだ見られます。XHTMLです。それらのドキュメントは、おそらくMIMEタイプが `text/html` のHTMLとして配信されていますが、これらの古いdoctypesはまだ死んでいません。

ドキュメントサイズ

ページのドキュメントサイズとは、ネットワーク上で転送されたHTMLのバイト数のことで、圧縮が有効な場合はそれも含めて計算されます。630万のドキュメントのセットの両極端では

- 1,110個のドキュメントは空です（0バイト）。
- ドキュメントの平均サイズは49.17KBです（ほとんどの場合、圧縮されています[138]）。
- 最大のドキュメントは61.19 *MB* で、Web Almanacの中でも独自の分析と章立てが必要なほどです。

では、この状況は一般的にどうなのでしょうか？　ドキュメントの中央値は24.65KBで、こ

136. https://hsivonen.fi/doctype/#xml
137. https://blog.whatwg.org/html-is-the-new-html5
138. https://w3techs.com/technologies/details/ce-gzipcompression

れは驚き[139]がない状態です。

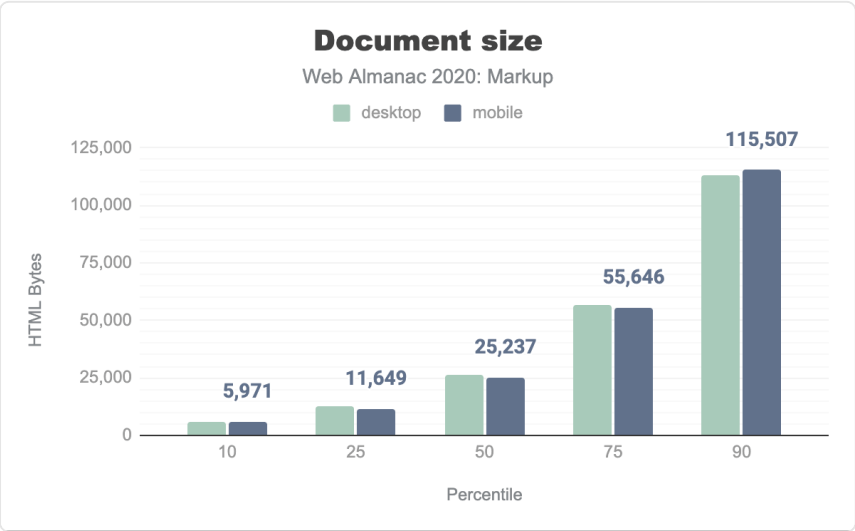


図3.3. ネットワーク上で転送されたHTMLのバイト数（有効な場合は圧縮を含む）。

ドキュメント言語

`html` 開始タグの `lang` 属性に2,863の異なる値を確認しました（Ethnologueによる7,117の話されている言語[140]と比較してみてください）。アクセシビリティの章によると、ほとんどすべての値が有効なようです。

全ドキュメントの22.36%が `lang` 属性を指定していません。一般的にはそうすべきだ[141]という意見は多いです、ソフトウェアが最終的に言語を自動的に検出する[142]という事実を無視してドキュメントの言語はプロトコルレベルで[143]指定することもできますが、これは私たちがチェックしなかったことです。

ここでは、サンプルの中でもっとも人気のある（正規化された）10の言語を紹介します。HTTP Archiveは米国のデータセンターから英語の言語設定でクロールしているため、ページが書かれている言語を見ると、英語に偏っていることに注意してください。とはいえ、分析したサイトのコンテキストを示すために、 `lang` 属性を提示します。

139. *https://httparchive.org/reports/page-weight*
140. *https://www.ethnologue.com/guides/how-many-languages*
141. *https://www.w3.org/TR/i18n-html-tech-lang/#overall*
142. *https://meiert.com/en/blog/lang/*
143. *https://developer.mozilla.org/docs/Web/HTTP/Headers/Content-Language*

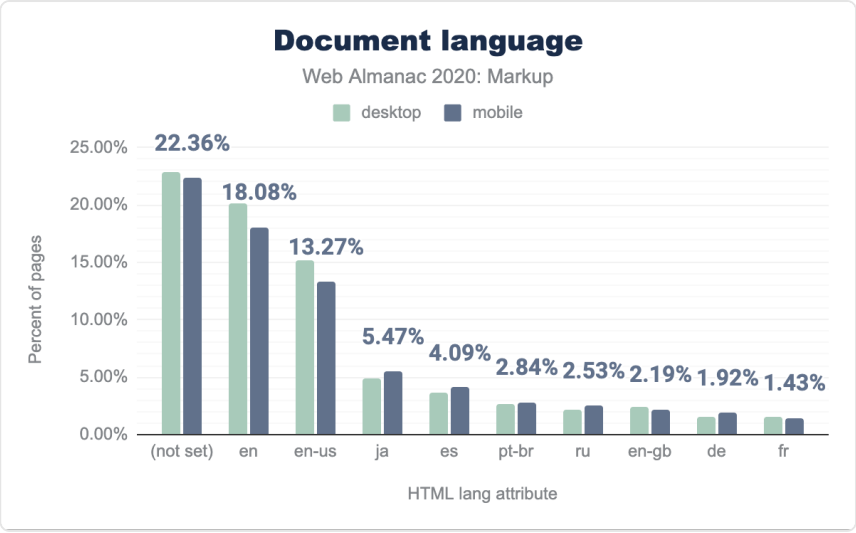


図3.4. HTMLの上位の `lang` 属性です。

コメント

コードにコメントを付けることは一般的に良い習慣であり、HTMLコメントはユーザーエージェントによってレンダリングされることなく、HTML文書にメモを付けるためのものです。

```
<!-- これは、HTMLのコメントです。 -->
```

多くのページでは制作のためにコメントが削除されているでしょうが90パーセンタイルのホームページでは、モバイルでは約73個、デスクトップでは約79個のコメントが使用されており、10パーセンタイルでは約2個のコメントが使用されていることがわかりました。 中央値のページでは、16個（モバイル）または17個（デスクトップ）のコメントが使用されています。

約89%のページには少なくとも1つのHTMLコメントが含まれており、約46%のページには条件付きのコメントが含まれています。

条件付きコメント

```
<!--[if IE 8]>
  <p>これは、Internet Explorer 8でのみ表示されます。</p>
<![endif]-->
```

上記は、非標準のHTML条件付きコメントです。過去にはブラウザの違いに取り組むためそれらが役立つことを証明されていましたが、MicrosoftがInternet Explorer 10でdropped conditional comments[144])を採用したことで、しばらくの間、歴史に名を残しました。

それでも上記のパーセンタイルの両極端では、90パーセンタイルで約6個、10パーセンタイルでは約1個の条件付きコメントを使用しているウェブページがあることをわかりました。ほとんどのページでは、html5shiv[145]、selectivizr[146]、respond.js[147]などのヘルパーのために条件付きコメントを使用しています。これらのページはきちんとしていて、まだアクティブなページですが、私たちの結論はこれらの多くが時代遅れのCMSテーマを使用していたということです。

本番環境では、HTMLコメントは通常、ビルドツールによって取り除かれます。以上のカウントとパーセンテージ、そして一般的なコメントの使用状況を考慮すると、多くのページがHTMLミニファイアーを介さずに提供されていると考えられます。

スクリプトの使用

以下の上位の要素のセクションで示すように、`script`要素は、6番目によく使われるHTML要素です。この章では、データセットの数百万ページの中で、`script`要素がどのように使われているかに興味を持ちました。

全体では、約2%のページにスクリプトがまったく含まれておらず、`type="application/ld+json"`属性を持つ構造化データスクリプトも含まれていません。最近では、1つのページに少なくとも1つの分析ソリューション用のスクリプトが含まれていることがかなり一般的になっていることを考えると、これは注目すべきことだと思います。

その反対に、約97%のページには、インラインまたは外部のスクリプトが少なくとも1つ含まれているという数字が出ています。

144. *https://docs.microsoft.com/ja-jp/previous-versions/windows/internet-explorer/ie-developer/compatibility/hh801214(v=vs.85)*
145. *https://github.com/aFarkas/html5shiv*
146. *http://selectivizr.com/*
147. *https://github.com/scottjehl/Respond*

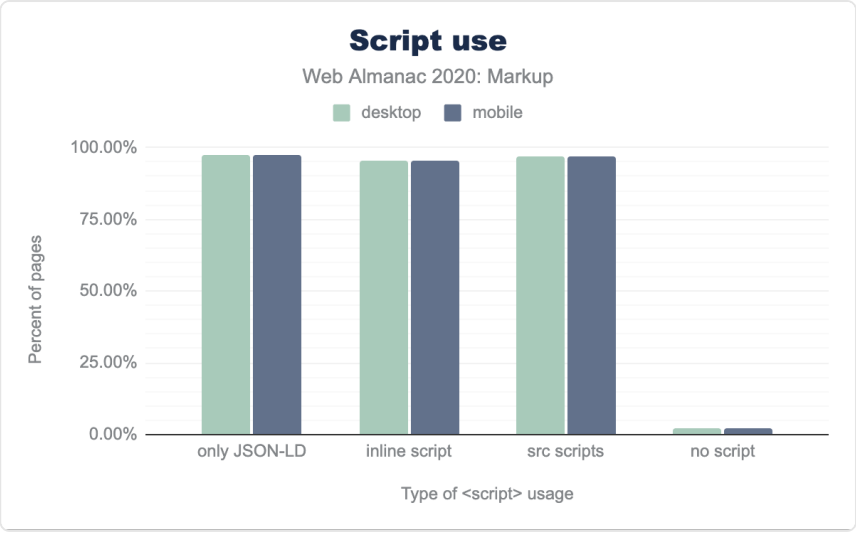


図3.5. `script` 要素の使い方。

ブラウザでスクリプトがサポートされていなかったり、オフになっている場合、`noscript` 要素は、ページ内にHTMLセクションを追加するのに役立ちます。上記のスクリプト数を考えると、`noscript` 要素についても気になりました。

分析の結果、約49%のページが `noscript` 要素を使用していることがわかりました。同時に、約16%の `noscript` 要素が、”googletagmanager.com “を参照する `src` 値を持つ `iframe` を含んでいます。

これは、実際の `noscript` 要素の総数が、ページの `body` 開始タグの後に `noscript` スニペットを追加するようにユーザーに強制するGoogleタグマネージャーなどの一般的なスクリプトの影響を受ける可能性があるという理論を裏付けているようです。

スクリプトの種類

`script` 要素で使用される `type` 属性値は何ですか？

- `text/javascript` : 60.03%
- `application/ld+json` : 1.68%
- `application/json` : 0.41%
- `text/template` : 0.41%

- `text/html` : 0.27%

`type="module"` を使用してJavaScriptモジュールスクリプト[148]を読み込む場合、現在、0.13%の `script` 要素がこの属性と値の組み合わせを指定していることがわかりました。また、 `nomodule` は、テストした全ページの0.95%で使用されています。(1つの指標は要素に関するもので、もう1つの指標はページに関するものであることに注意してください。)

全スクリプトの36.38%は、 `type` の値がまったく設定されていません。

要素

このセクションでは、要素に焦点を当てています。どのような要素が使用されているのか、どのくらいの頻度で使用されているのか、どのような要素が特定のページに表示される可能性があるのか、カスタム要素、廃止された要素、独自の要素についてはどのような状況なのか、などです。 *divitis* はまだありますか？　はい。

要素の多様性

では、実際にHTMLがどのように使われているのかを見てみましょう。作者はさまざまな要素を使っているのか、それとも比較的少ない要素しか使っていないのか。

ウェブページの中央値では、30種類の要素が587回使用されていることがわかります。

148. *https://jakearchibald.com/2017/es-modules-in-browsers/*

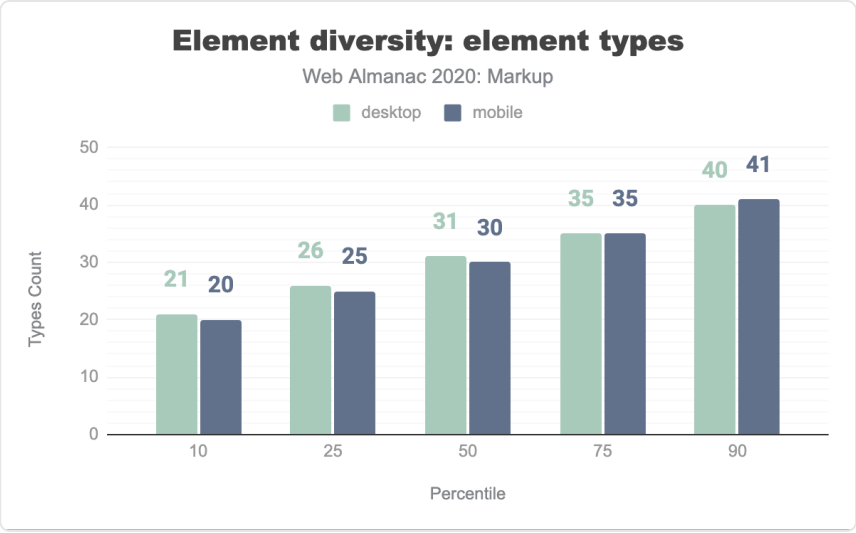


図3.6. ページごとの要素タイプ数の分布。

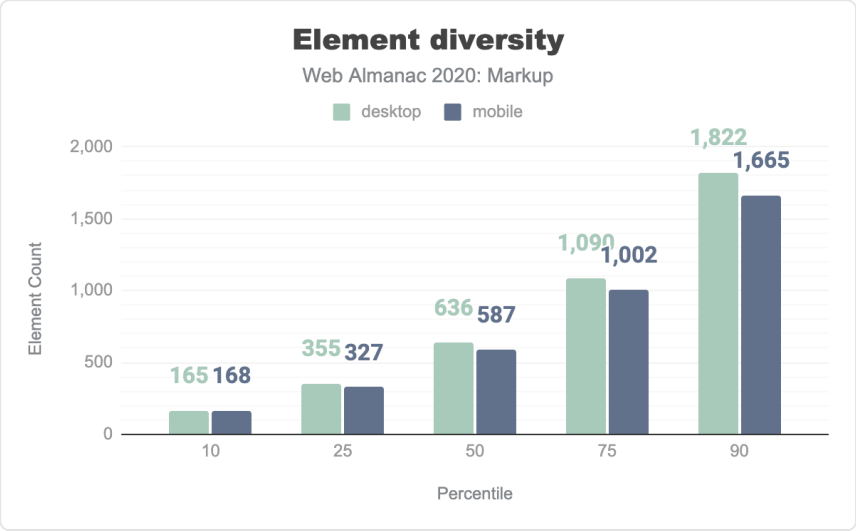


図3.7. 1ページあたりの総要素数のパーセンタイルごとの分布。

生きたHTML[149]が現在112個の要素を持っていることを考えると、90パーセンタイルが41個以上の要素を使用していないということは、ほとんどの文書でHTMLがほとんど使い尽くされていないことを示唆しているかもしれません。しかしHTMLが提供する意味的な豊かさ

149. https://html.spec.whatwg.org/multipage/

は、すべての文書がそのすべてを必要とすることを意味しないので、これがHTMLとその使用にとって実際に何を意味するかを解釈するのは難しいです。HTMLの要素は、利用可能かどうかではなく、目的（セマンティクス）に応じて使用されるべきです。

これらの要素はどのように配分されていますか？

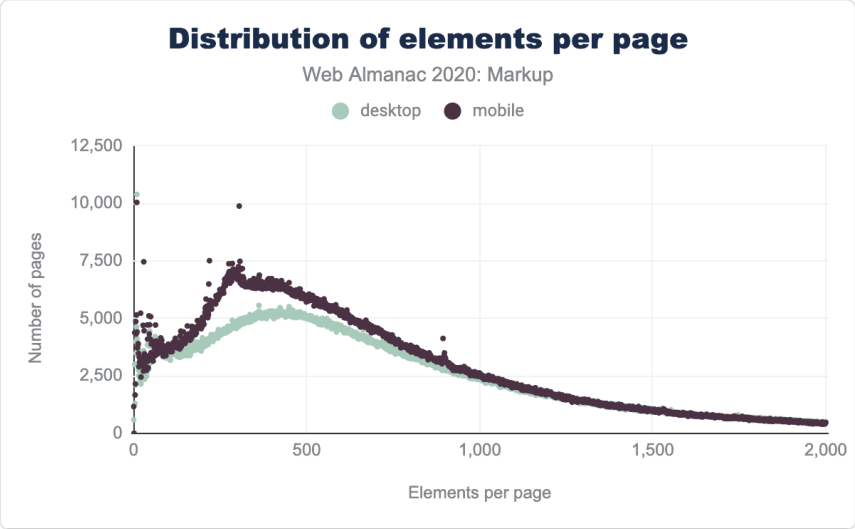


図3.8. 1ページあたりの総要素数の分布。

2019年と比べて[150]そんなに変わっていない！

トップ要素

2019年、Web AlmanacのMarkupの章では、2005年のIan Hicksonの仕事[151]を参考に、もっとも頻繁に使用されている要素を紹介しました。私たちはこれを参考にして、もう一度そのデータを見てみました。

150. *https://almanac.httparchive.org/ja/2019/markup#fig-3*
151. *https://web.archive.org/web/20060203031713/http://code.google.com/webstats/2005-12/elements.html*

2005	2019	2020
title	div	div
a	a	a
img	span	span
meta	li	li
br	img	img
table	script	script
td	p	p
tr	option	link
		i
		option

図3.9. 2005年、2019年、2020年のもっとも人気のある要素です。

トップ7には何の変化もありませんでしたが、`option`要素は少し人気がなくなり、8位から10位に下がり、`link`要素と`i`要素が人気を失ってしまいました。これらの要素が使用されるようになったのは、おそらくresource hintsの使用が増加したことや、Font Awesome[152]のようなアイコンソリューションが使用されるようになったからでしょう。

`details`と`summary`

もうひとつ気になっていたのは、とくに2020年に`details`と`summary`要素が導入されてから、その使用状況です。これらは使われていますか？　それらは著者にとって魅力的で、人気もあるのでしょうか？　結果的には、テストした全ページのうち0.39%しか使用されていませんでした。ただし必要な場面で、正しい方法で使用されているかどうかを判断するのは難しいので、「人気がある」という表現は間違っています。

ここでは、`details`要素の中で`summary`を使用する簡単な例を紹介します。

```
<details>
```

152. https://fontawesome.com/

```
    <summary>ステータス 動作確認済み</summary>
    <p>速さ：12m/s</p>
    <p>方向：北</p>
  </details>
```

しばらく前に、Steve Faulkner氏が、この2つの要素がいかに不適切に使われているかを指摘[153]していました。上の例からわかるように、`details`要素ごとに、`details`の最初の子[154]としてのみ使用可能な`summary`要素が必要になります。

そこで、`details`要素と`summary`要素の数を調べてみたところ、引き続き誤用されていることがわかりました。また、`summary`要素の数は、モバイルとデスクトップの両方で多く、モバイルでは`details`要素1つに対して`summary`要素が1.11、デスクトップでは1.19となっています。

	発生状況	
要素	モバイル(0.39%)	デスクトップ(0.22%)
summary	62,992	43,936
details	56,60	36,743

図3.10. `details`と`summary`要素の採用。

要素使用の確率

要素のポピュラリティについてもう一度見てみましょう。あるページのDOMで特定の要素が見つかる確率はどのくらいでしょうか？　確かに、`html`、`head`、`body`はすべてのHTMLページに存在し（これらのタグはすべてオプション[155]であるにもかかわらず）、一般的な要素となっていますが、他にどのような要素がよく見られるのでしょうか？

153. https://x.com/stevefaulkner/status/806474286592561152
154. https://developer.mozilla.org/docs/Web/HTML/Element/summary#usage_notes
155. https://meiert.com/en/blog/optional-html/

要素	確率
title	99.34%
meta	99.00%
div	98.42%
a	98.32%
link	97.79%
script	97.73%
img	95.83%
span	93.98%
p	88.71%
ul	87.68%

図3.11. Web Almanac 2020のサンプルのページで、ある要素が見つかる確率が高いこと。

標準要素とは、HTMLの仕様に含まれている、あるいは含まれていた要素のことです。では、どの要素が珍しいのでしょうか？　私たちのサンプルでは、次のようなものがあります。

要素	確率
dir	0.0082%
rp	0.0087%
basefont	0.0092%

図3.12. サンプルのページから特定の要素が見つかる確率が低いこと。

私たちは、どのような要素が廃れてしまったのかを知るために、これらの要素を含めています。しかし、`dir`と`basefont`はXHTML1.0(2000)で最後に指定され、もはやHTMLの一部ではありませんが、`rp`の稀な使用（これは1998年の早い段階で[156]、still part of HTML[157]に言及されており）は、Rubyマークアップ[158]があまり普及していないことを示唆しているのかも

156. https://www.w3.org/TR/1998/WD-ruby-19981221/#a2-4
157. https://html.spec.whatwg.org/multipage/text-level-semantics.html#the-rp-element
158. https://www.w3.org/TR/ruby/

しれない。

カスタム要素

Web Almanac2019年版では、いくつかの非標準的な要素を取り上げてカスタム要素[159]を扱いました。今年は、カスタム要素を詳しく見てみることに価値があると考えました。どのようにしてこれらを決定したのでしょうか？　大まかには、その定義[160]を見て、特にハイフンの使用を確認しています。ここでは、サンプル内の全URLの1%以上に使用されている要素に焦点を当ててみましょう。

要素	ページ	ページ(%)
`ym-measure`	141,156	2.22%
`wix-image`	76,969	1.21%
`rs-module-wrap`	71,272	1.12%
`rs-module`	71,271	1.12%
`rs-slide`	70,970	1.12%
`rs-slides`	70,993	1.12%
`rs-sbg-px`	70,414	1.11%
`rs-sbg-wrap`	70,414	1.11%
`rs-sbg`	70,413	1.11%
`rs-progress`	70,651	1.11%
`rs-mask-wrap`	63,871	1.01%
`rs-loop-wrap`	63,870	1.01%
`rs-layer-wrap`	63,849	1.01%
`wix-iframe`	63,590	1%

図3.13. 人気の高い14種類のカスタム要素

これらの要素は3つのソースから来ています。昨年も紹介した分析ソリューションのYandex

159. https://almanac.httparchive.org/ja/2019/markup#custom-elements
160. https://html.spec.whatwg.org/multipage/custom-elements.html#custom-elements-core-concepts

Metrica[161]（`ym-`）、Slider Revolution[162]（`rs-`）は、WordPressのスライダーで、サンプルのトップ付近にもっと多くの要素があります。そして、ウェブサイトビルダーのWix[163]（`wix-`）があります。

また、AMPマークアップ[164]の`amp-img`（11,700ページ）、`amp-analytics`（10,256ページ）、`amp-auto-ads`（7,621ページ）などの`amp-`要素を持つグループや、Angular[165]の`app-`要素である`app-root`(16,314)、`app-footer`(6,745)、`app-header`(5,274) なども含まれています。

廃止された要素

HTMLの使用については、廃止された要素（`applet`、`bgsound`、`blink`、`center`、`font`、`frame`、`isindex`、`marquee`、`spacer`などの要素）の使用を含め、さらに多くの疑問があります。

630万ページのモバイルデータセットでは、約90万ページ（14.01%）にこれらの要素が1つ以上含まれています。ここでは、10,000回以上使用された上位9つの要素を紹介します。

要素	ページ	ページ(%)
`center`	458,402	7.22%
`font`	430,987	6.79%
`marquee`	67,781	1.07%
`nobr`	31,138	0.49%
`big`	27,578	0.43%
`frame`	19,363	0.31%
`frameset`	19,163	0.30%
`strike`	17,438	0.27%
`noframes`	15,016	0.24%

図3.14. 10,000回以上使用されている廃止された要素。

161. https://metrica.yandex.com/about
162. https://www.sliderrevolution.com/
163. https://www.wix.com/
164. https://amp.dev/
165. https://angular.io/

また、 `spacer` でさえも1,584回使用されており、5,000ページごとに存在しています。Googleが自社のホームページ[166]で22年前から[167]、 `center` 要素を使用していることは知っていますが、なぜこれほど多くの模倣者がいるのでしょうか。

`isindex`

気になっていたら: このデータセットに含まれる `isindex` 要素の総数は1です。ちょうど1つのページが `isindex` 要素を使用していました。 `isindex` はHTML4.01とXHTML1.0[168] までは仕様の一部でしたが、2006年になってようやく 仕様化[169]され（ブラウザでの実装方法に合わせて）、2016年には廃止[170]されました。

独自要素と作り込み要素

その中には標準的なHTML（SVG、MathML）要素でも、カスタム要素でも、廃止された要素でもない独自の要素も含まれていました。私たちが確認したトップ10は以下の通りです。

要素	ページ(%)
`noindex`	0.89%
`jdiv`	0.85%
`mediaelementwrapper`	0.49%
`ymaps`	0.26%
`yatag`	0.20%
`ss`	0.11%
`include`	0.08%
`olark`	0.07%
`h7`	0.06%
`limespot`	0.05%

図3.15. 質問可能な遺産の要素

166. https://www.google.com/
167. https://web.archive.org/web/19981202230410/https://www.google.com/
168. https://meiert.com/en/indices/html-elements/
169. https://lists.w3.org/Archives/Public/public-whatwg-archive/2006Feb/0111.html
170. https://github.com/whatwg/html/pull/1095

これらの要素の出所は、不明なものもあれば追跡可能なものもあり、混在しているようです。もっともポピュラーなものである `noindex` は、Yandexがページのインデックス作成を禁止するために推奨している[171]ことに起因すると思われます。 `jdiv` は昨年の Web Almanac[172]で指摘された、JivoChatのものです。 `mediaelementwrapper` はMediaElementというメディアプレイヤーから来ています。 `yymaps` と `yatag` もYandexのものです。また、 `ss` 要素は、eBayの元eコマース製品であるProStoresからのものである可能性があり、 `olark` は、Olarkチャットソフトウェアからのものである可能性があります。 `h7` は間違いのようです。 `limespot` は、おそらくeコマース用のパーソナライゼーションプログラムであるLimespotに関連していると思われます。これらの要素はいずれもウェブ標準の一部ではありません。

見出し

見出し[173]は、sectioning[174]やaccessibility[175]で重要な役割を果たす、特別なカテゴリーの要素です。

見出し	発生状況	1ページあたりの平均
h1	10,524,810	1.66
h2	37,312,338	5.88
h3	44,135,313	6.96
h4	20,473,598	3.23
h5	8,594,500	1.36
h6	3,527,470	0.56

図3.16. 標準的な見出し要素の使用頻度と平均値

標準的な `<h1>` から `<h6>` までの要素しか見られないと思っていたかもしれませんが、実際にはもっと多くのレベルを使用しているサイトもあります。

171. https://yandex.com/support/webmaster/adding-site/indexing-prohibition.html
172. https://almanac.httparchive.org/ja/2019/markup#products-and-libraries-and-their-custom-markup
173. https://html.spec.whatwg.org/multipage/dom.html#heading-content
174. https://html.spec.whatwg.org/multipage/dom.html#sectioning-content-2
175. https://www.w3.org/WAI/tutorials/page-structure/headings/

見出し	発生状況	1ページあたりの平均
h7	30,073	0.005
h8	9,266	0.0015

図3.17. 非標準的な見出し要素の使用頻度と平均値。

最後の2つは、もちろんHTMLの一部ではありませんので、使用しないでください。

属性

このセクションでは、ドキュメントで属性がどのように使用されているかに焦点を当て、`data-*`の使用パターンを探ります。その結果、`class`がすべての属性の女王であることがわかりました。

トップの属性

もっとも人気のある要素のセクションと同様に、このセクションではウェブ上でもっとも人気のある属性について掘り下げます。`href`属性がウェブそのものにとってどれほど重要か、また`alt`属性が情報をアクセシブルにするため、どれほど重要かを考えると、これらはもっとも人気のある属性でしょうか？

属性	発生状況	パーセンテージ
`class`	2,998,695,114	34.23%
`href`	928,704,735	10.60%
`style`	523,148,251	5.97%
`id`	452,110,137	5.16%
`src`	341,604,471	3.90%
`type`	282,298,754	3.22%
`title`	231,960,356	2.65%
`alt`	172,668,703	1.97%
`rel`	171,802,460	1.96%
`value`	140,666,779	1.61%

図3.18. 使用頻度の高い上位10属性。

もっとも人気のある属性は`class`で、データセットに30億回近く出現し、使用されている全属性の34%を占めています。

トップ10には、意外にも`input`要素の値を指定する`value`属性がランクインしています。私たちの主観では、`value`属性がそれほど頻繁に使われている印象を受けなかったからです。

ページの属性

すべてのドキュメントに見られる属性はあるのでしょうか？　ありませんが、ほとんど:

要素	ページ(%)
href	99.21%
src	99.18%
content	98.88%
name	98.61%
type	98.55%
class	98.24%
rel	97.98%
id	97.46%
style	95.95%
alt	90.75%

図3.19. ページ別トップ10の属性

これらの結果は、私たちが答えられない問題を提起しています。たとえば、`type`は他の要素でも使用されていますが、なぜこのように絶大な人気があるのでしょうか。とくに、CSSやJavaScriptがデフォルトとされている中で、スタイルシートやスクリプトに対して指定する必要がないのは普通であることを考えると。また`alt`は実際にはどうなのでしょうか。9.25%のページには画像が含まれていないのでしょうか、それとも単にアクセスできないだけなのでしょうか。

`data-*`属性

HTML仕様によると、`data-*`属性は、”カスタムデータ、ステート、アノテーション、および同様のものをページまたはアプリケーションに対してプライベートに保存することを目的としており、これ以上適切な属性または要素がない場合に使用されます。”とあります。どのように使われていますか？　人気のあるものは何ですか？　何かおもしろいものはありますか？

もっとも人気のある2つの属性は、その後に続く各属性（使用率1%以上）よりも2倍近く人気があるため、際立っています:

属性	発生状況	パーセンテージ
data-src	26,734,560	3.30%
data-id	26,596,769	3.28%
data-toggle	12,198,883	1.50%
data-slick-index	11,775,250	1.45%
data-element_type	11,263,176	1.39%
data-type	11,130,662	1.37%
data-requiremodule	8,303,675	1.02%
data-requirecontext	8,302,335	1.02%

図3.20. もっともポピュラーな`data-*`属性です。

`data-type`、`data-id`、`data-src`のような属性は、複数の汎用的な用途がありますが、`data-src`はJavaScriptによる遅延画像読み込みで多く使用されています（例：Bootstrap 4）。Bootstrap[176]でも、`data-toggle`の存在が説明されており、トグルボタンの状態スタイリングフックとして使用されています。`data-slick-index`はSlick carousel plugin[177]が、`data-element_type`はElementor's WordPress website builder[178]が持っています。つまり、`data-requiremodule`と`data-requirecontext`は、どちらもRequireJS[179]の一部なのです。

興味深いことに、画像に対するネイティブ遅延ローディングの使用率は、`data-src`の場合と同様です。3.86%のページ[180]が`<img>`要素に`loading="lazy"`を使用しています。2月の時点ではこの数字は0.8%[181]程度だったので非常に、急速に増加しているようです。これらは、クロスブラウザソリューション[182]のために併用されている可能性があります。

その他

これまで、HTMLの一般的な使い方や、上位の要素や属性の採用について説明してきました。このセクションでは、ビューポート、ファビコン、ボタン、入力、リンクなどの特殊なケースを確認しています。ここで注意しなければならないのは、あまりにも多くのリンクが

176. https://getbootstrap.com/
177. https://kenwheeler.github.io/slick/
178. https://elementor.com/
179. https://requirejs.org/
180. https://docs.google.com/spreadsheets/d/1ram47FshAjzvbQVJbAQPgxZN7PPOPCKIK67VJZCo92c/edit#gid=2109061092
181. https://x.com/zcorpan/status/1237016679667970050
182. https://addyosmani.com/blog/lazy-loading/

いまだに「http」のURLを指していることです。

`viewport`の仕様

viewport[183]メタ要素は、モバイルブラウザでのレイアウトを制御するために使用されます。数年前まではWebページを構築する際に「viewport meta要素を忘れないように」というモットーのようなものがありましたが、やがてこれが一般的になり、「ズームやスケーリングが無効になっていないか確認する」というモットーに変わりました。

ユーザーは、テキストを最大500%[184]まで拡大・縮小できる必要があります。これが、Lighthouse[185]やaxe[186]のような人気のあるツールの監査が、`meta name="viewport"`要素内で`user-scalable="no"`が使用され、`maximum-scale`属性値が`5`未満の場合に失敗する理由です。

データを見てみると結果をよりよく理解するためにスペースを削除し、すべてを小文字に変換し、`content`属性のコンマ値で、ソートすることでデータを正規化しました。

コンテンツ属性値	ページ	ページ (%)
`initial-scale=1,width=device-width`	2,728,491	42.98%
blank	688,293	10,84%
`initial-scale=1,maximum-scale=1,width=device-width`	373,136	5.88%
`initial-scale=1,maximum-scale=1,user-scalable=no,width=device-width`	352,972	5.56%
`initial-scale=1,maximum-scale=1,user-scalable=0,width=device-width`	249,662	3.93%
`width=device-width`	231,668	3.65%

図3.21. `viewport`の仕様、およびその欠如。

その結果、分析したページのほぼ半分が、典型的なviewport`content`値を使用していることがわかりました。しかし約10%のモバイルページでは、viewport meta要素の適切な`content`値が完全に欠落しており、残りのページでは、`maximum-scale`、`minimum-scale`、`user-scalable=no`、または`user-scalable=0`の不適切な組み合わせが使用さ

183. https://developer.mozilla.org/docs/Web/HTML/Viewport_meta_tag
184. https://dequeuniversity.com/rules/axe/4.0/meta-viewport-large
185. https://developers.google.com/web/tools/lighthouse
186. https://www.deque.com/axe/

れています。

しばらく前から、モバイルブラウザEdgeでは、viewport meta要素を採用しているWebページで定義されているズーム設定に関わらず、最低でも500%[187]までWebページを拡大することができるようになりました。

ファビコン

ファビコンを取り巻く状況は非常に興味深いものがあります。あるブラウザは ドメインのルートを見る[188]に戻り、いくつかの画像フォーマットを受け入れ、さらに数十種類のサイズを促進します（いくつかのツールはそのうちの45種類を生成すると報告されています。realfavicongenerator.net[189]はすべてのケースを処理するように要求されると 37 を返します）。この記事を書いている時点では、この状況を改善するために、HTML仕様のopen issue[190]があります。

テストを作成する際には、画像の有無はチェックせず、マークアップのみを見ていました。つまり以下の内容を確認する際には、ファビコンが使用されているか、または使用頻度よりもファビコンはどのように参照されているかが重要であることに注意してください。

ファビコンの形式	ページ	ページ(%)
ICO	2,245,646	35.38%
PNG	1,966,530	30.98%
No favicon defined	1,643,136	25.88%
JPG	319,935	5.04%
No extension specified (no format identifiable)	37,011	0.58%
GIF	34,559	0.54%
WebP	10,605	0.17%
…		
SVG	5,328	0.08%

図3.22. 一般的なファビコンのフォーマット。

ここにはいくつかのサプライズがあります:

187. https://blogs.windows.com/windows-insider/2017/01/12/announcing-windows-10-insider-preview-build-15007-pc-mobile/
188. https://realfavicongenerator.net/faq#why_icons_in_root
189. https://realfavicongenerator.net/
190. https://github.com/whatwg/html/issues/4758

- 他のフォーマットにも対応していますが、ウェブ上のファビコンのフォーマットとしてはICOが主流です。
- JPGは比較的人気のあるファビコン形式ですが、多くのファビコンサイズでは最良の結果が得られない（または重量が比較的大きくなる）場合があります。
- WebPはSVGの2倍の人気があります。しかし、SVG favicon support[191]の改善により、この状況は変わるかもしれません。

ボタンと入力タイプ

最近、ボタンに関する議論[192]が盛んに行われており、ボタンがどれほど頻繁に誤用されているかが明らかになっています。私たちは、いくつかのネイティブなHTMLボタンについての調査結果を発表するために、これを調べました。

60.56%

図3.23. ボタン要素のあるページの割合

ボタンの種類	発生状況	ページ (%)
`<button type="button">`	15,926,061	36.41%
`<button>` without type	11,838,110	32.43%
`<button type="submit">`	4,842,946	28.55%
`<input type="submit" value="…">`	4,000,844	31.82%
`<input type="button" value="…">`	1,087,182	4.07%
`<input type="image" src="…">`	322,855	2.69%
`<button type="reset">`	41,735	0.49%

図3.24. ボタンタイプの採用。

分析の結果、約60%のページにボタン要素が含まれており、そのうち半数以上（32.43%）のページに、`type`属性の指定に失敗したボタンが少なくとも1つ存在していることがわか

191. https://caniuse.com/link-icon-svg
192. https://adrianroselli.com/2016/01/links-buttons-submits-and-divs-oh-hell.html

りました。`button`要素のdefault type[193]が`submit`であることに注意してください。したがって、これらの32%のページにおけるボタンのデフォルトの動作は、現在のフォームデータを送信することです。このような予期せぬ動作を避けるためには、`type`属性を指定するのがベストです。

パーセンタイル	1ページあたりのボタン数
10	0
25	0
50	1
75	5
90	13

図3.25. 1ページあたりのボタン数の分布

10パーセンタイルと25パーセンタイルのページにはボタンがまったく含まれていませんが、90パーセンタイルのページには13個のネイティブな`button`要素が含まれています。つまり、10%のページが13個以上のボタンを含んでいることになります。

リンク対象

アンカー要素[194]、つまり`a`要素は、ウェブリソースをリンクするものです。本節では、それぞれのリンク先に含まれるプロトコルの採用状況を分析します。

193. https://dev.w3.org/html5/spec-LC/the-button-element.html
194. https://developer.mozilla.org/docs/Web/HTML/Element/a

プロトコル	発生状況	ページ(%)
`https`	5,756,444	90.69%
`http`	4,089,769	64.43%
`mailto`	1,691,220	26.64%
`javascript`	1,583,814	24.95%
`tel`	1,335,919	21.05%
`whatsapp`	34,643	0.55%
`viber`	25,951	0.41%
`skype`	22,378	0.35%
`sms`	17,304	0.27%
`intent`	12,807	0.20%

図3.26. リンクターゲットプロトコルの採用

また、`https`や`http`がもっとも多く、次いでメールの作成や電話、メッセージの送信を容易にする「良心的」なリンクが続いていることがわかります。`javascript`は、JavaScriptがネイティブで緩やかに機能を低下するオプションが用意されているにもかかわらず、いまだに人気のあるリンクターゲットとして際立っています。

リンクは新規ウィンドウで表示

71.35%

図3.27. `target="_blank"`のリンクに`noopener`や`noreferrer`の属性を持たないページの割合。

`target="_blank"`を使用することは、以前からセキュリティ上の脆弱性[195]であることが知られていました。しかし、71.35%のページが`target="_blank"`を使ったリンクを含んでおり、`noopener`や`noreferrer`を使っていません。

195. https://mathiasbynens.github.io/rel-noopener/

エレメント	ページ
`<a target="_blank" rel="noopener noreferrer">`	13.63%
`<a target="_blank" rel="noopener">`	14.14%
`<a target="_blank" rel="noreferrer">`	0.56%

図3.28. 空白の関係。

経験則として、また、使いやすさの観点から[196]、そもそも`target="_blank"`を使用しないことをオススメします。

最新のSafariとFirefoxでは、`a`要素に`target="_blank"`を設定すると、`rel="noopener"`を設定するのと同じ`rel`の動作が暗黙的に提供されます。これはすでにChromium[197]でも実装されており、Chrome 88にも搭載される予定です。

結論

この章では、いくつかの見解に触れてきましたが2020年のマークアップのあり方を振り返る意味で、いくつか目についた点をご紹介します。

3.97%

図3.29. 変わったdoctypeを持つページの割合。

癖のあるモードに着地するページが減りました。2016年には、その数は約7.4%[198]でした。2019年の終わりには、4.85%[199]と観測されました。そして今は、約3.97%となっています。この傾向は、本章のレビューに登場したSimon Pietersの言葉を借りれば、明確で励みになると思われます。

開発の全体像を把握するための歴史的なデータはありませんが、「無意味な」`div`、`span`、`i`のマークアップは、1990年代から2000年代前半に見られた`table`のマークアップをほぼ置換しています。意味的により適切な代替手段がないのに、常に`div`や`span`要素が使用されていることに疑問を感じるかもしれませんが、これらの要素は`table`マークアップよりも好ましいものです。表形式のデータ以外のすべてに使用されます。

196. https://www.nngroup.com/articles/new-browser-windows-and-tabs/
197. https://chromium-review.googlesource.com/c/chromium/src/+/1630010
198. https://discuss.httparchive.org/t/how-many-and-which-pages-are-in-quirks-mode/777
199. https://x.com/zcorpan/status/1205242913908838400

ページあたりの要素とページあたりの要素タイプはほぼ変わらず、2019年と比較して、私たちのHTMLの書き方に大きな変化はないことを示しています。このような変化は、顕在化するまでに時間を要する可能性があります。

`g:plusone`（モバイルサンプルでは17,607ページで使用）や`fb:like`（11,335ページ）のような独自の製品固有の要素は、昨年は[もっとも人気のある要素の1つ]（../2019/markup#products-and-libraries-and-their-custom-markup）だったのですが、ほとんどなくなっています。しかし、Slider RevolutionやAMP、Angularなどのカスタム要素が多く観測されます。また、`ym-measure`、`jdiv`、`ymaps`などの要素もまだ多く見られます。ここで私たちが想像するのはゆっくりと変化するプラクティスの海の下で作者が廃止されたマークアップを捨て、新しいソリューションを採用することでHTMLが非常によく開発され、維持されているということです。

さて、2019年Web Almanacマークアップの章[200]では、このテーマに関する前回の大規模な調査から14年分の追記があったため、それからの1年間ではあまりカバーできないと思われるかもしれません。しかし、今年のデータで観察されたのは、HTMLの海の底や岸辺付近で多くの動きがあったということです。私たちは、生きたHTMLをほぼ完全に採用しようとしています。GoogleやFacebookのウィジェットのような流行のページはすぐに削除されています。AngularやAMP（「コンポーネントフレームワーク」と呼ばれています）の人気が著しく低下しReactやVueのようなソリューションが求められているように、私たちはフレームワークを採用するのも、避けるのも早いです。

それでも、HTMLが与えてくれる選択肢を使い切った形跡はありません。あるページで使われている30種類の要素の中央値は、HTMLが提供する要素の約4分の1であり、HTMLがかなり一方的に使われていることを示唆しています。これは、`div`や`span`のような要素の絶大な人気によって裏付けられており、これらの2つの要素が表す可能性のある需要を満たすカスタム要素はありません。残念ながら、サンプルの各文書を検証することはできませんでした。しかし、注意が必要な逸話として、W3Cでテストされた文書の79%[201]には検証エラーがあることを知りました。これだけ見ても、私たちはまだまだHTMLの技術を習得するには程遠いようです。

だからこそ、私たちは訴えたいことがあるのです。HTMLに注意を払ってください。HTMLに注目してください。HTMLに投資することは重要であり、価値があります。HTMLはドキュメント言語であり、プログラミング言語のような魅力はありませんが、それでもウェブはHTMLの上に構築されています。HTMLの使用量を減らし、本当に必要なものを学びましょう。もっと適切なHTMLを使いましょう、何が使えるのか、何のために存在するのかを学びましょう。そして、あなたのHTMLを検証しましょう[202]。誰でも無効なHTMLを書くことができますが（次に会う人にHTML文書を書いてもらい、その出力を検証してみてください）、プロの開発者は有効なHTMLを作成することが期待できます。正しくて有効なHTMLを書く

200. *https://almanac.httparchive.org/ja/2019/markup*
201. *https://github.com/HTTPArchive/almanac.httparchive.org/issues/899#issuecomment-717856201*
202. *https://validator.w3.org/docs/why.html*

ことは、誇りを持って行うべき技術です。

次回のWeb Almanacの章では、HTMLを書く技術を詳しく見て、できれば私たちがどのように改善してきたかを見ていく準備をしましょう。

あとは、あなたにお任せします。あなたの観察力はどうですか？何があなたの目に留まりましたか？何が悪くなって、何が良くなったと思いますか？Leave a comment[203] であなたの意見を聞かせてください。

著者

Jens Oliver Meiert

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

Jens Oliver Meiert はウェブ開発者であり、著者(*CSS Optimization Basics*[204], *The Web Development Glossary*[205])で、Jimdo[206]でエンジニアリングマネージャーを務めています。彼は、HTMLとCSSの最適化を専門とするウェブ開発のエキスパートです。Jensは技術標準に貢献し、特に自身のウェブサイトmeiert.com[207]では、仕事や研究について定期的に執筆しています。

Catalin Rosu

@catalinred catalinred https://catalin.red/

Catalin Rosuは、Caphyon[208]のフロントエンド開発者で、現在はWattspeed[209]に携わっています。ウェブ標準への情熱と、優れたUXとUIへの鋭い目を持ち、ツイート[210]したり、自身のウェブサイト[211]で執筆したりしています。

203. *https://discuss.httparchive.org/t/2039*
204. *https://leanpub.com/css-optimization-basics*
205. *https://leanpub.com/web-development-glossary*
206. *https://www.jimdo.com/*
207. *https://meiert.com/en/*
208. *https://www.caphyon.com/*
209. *https://www.wattspeed.com/*
210. *https://x.com/catalinred*
211. *https://catalin.red/*

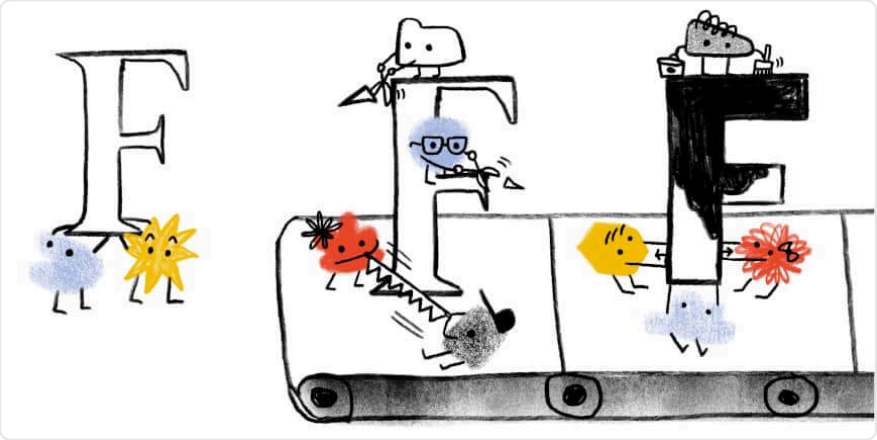
Ian Devlin

@iandevlin　iandevlin　https://iandevlin.com

Ian Devlinは、優れたセマンティックHTMLやアクセシビリティを提唱するウェブ開発者です。HTML5 マルチメディア[212]についての本を書いたこともありますし、自身のウェブサイト[213]でウェブやその他のことについて散発的に書いています。現在は、ドイツのreal.digital[214]でシニアフロントエンドエンジニアとして働いています。

212. *https://www.peachpit.com/store/html5-multimedia-develop-and-design-9780321793935*
213. *https://iandevlin.com/*
214. *https://www.real-digital.de/*

部 I 章 4
フォント



Raph Levien と Jason Pamental によって書かれた。
Roel Nieskens、 Chris Lilley、 Dave Crossland、 Rod Sheeter と Mandy Michael によってレビュ。
Abby Tsai による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

ほとんどのウェブサイトでは、テキストが中心となっています。タイポグラフィとは、そのテキストを視覚的で魅力的かつ効果的に表現する技術です。優れたタイポグラフィを実現するには、適切なフォントを選択する必要がありますが、デザイナーは膨大な種類のウェブフォントから選択できます。他のリソースと同様にパフォーマンスや互換性の問題がありますが、適切に使用すれば、その価値は十分にあります。この章では、ウェブフォントがどのように使用されているか、とくにどのように最適化されているかをデータに基づいて説明します。

Webフォントはどこで使われていますか？

ウェブフォントの使用率は、時間の経過とともに着実に増加しており（2011年末にはゼロ

に近かった）、デスクトップ向けのウェブページでは82%、モバイルでは80%がウェブフォントを使用しています。

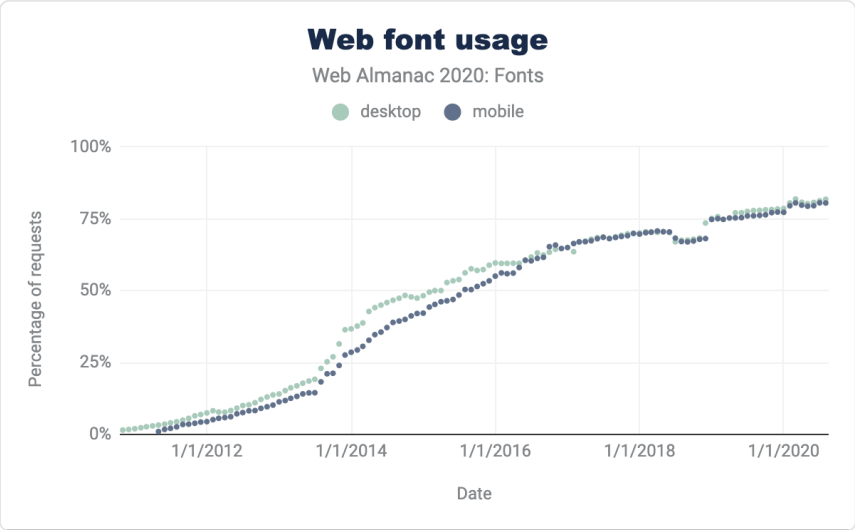


図4.1. Webフォントの使用状況を時系列で表示しています。

ウェブフォントの使用率は、少数の例外を除き、世界中でほぼ一貫しています。以下のグラフはウェブページあたりのウェブフォントのキロバイト数の中央値に基づいておりこれはフォントの多さ、大きさ、またはその両方を示す指標となります。

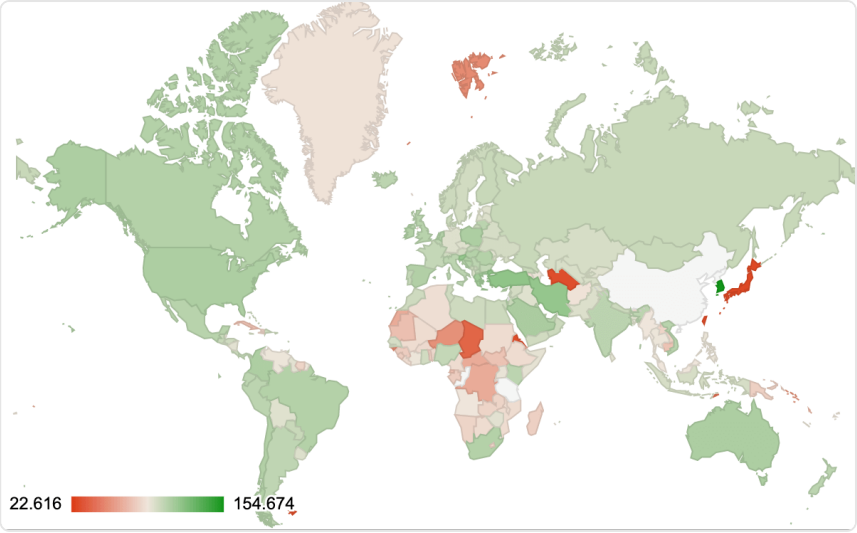


図4.2. 国別のWebフォント使用状況（デスクトップ）。

もっとも多くのフォントを使用している国は韓国ですが、これは韓国のインターネットの速度が常に高速で、遅延が少ないことや韓国語（ハングル）のフォントがラテン語よりも一桁以上大きいことを考えると驚くことではありません。日本や中国語圏では、ウェブフォントの使用量はかなり少なくなっています。これは、中国語や日本語のフォントが非常に大きいためです（フォントサイズの中央値は、ラテン語の中央値の1000倍以上にもなります）。つまり、日本ではWebフォントの利用率は非常に低く、中国では利用率はゼロに近い。しかし、後述するprogressive font enhancement[215]の最近の動向を見ると、数年以内に両国でウェブフォントが使えるようになるかもしれません。中国ではGoogle Fontsが確実に利用できないという報告もあり、それも採用を妨げる要因になっているかもしれません。

215. https://www.w3.org/TR/2020/NOTE-PFE-evaluation-20201015/

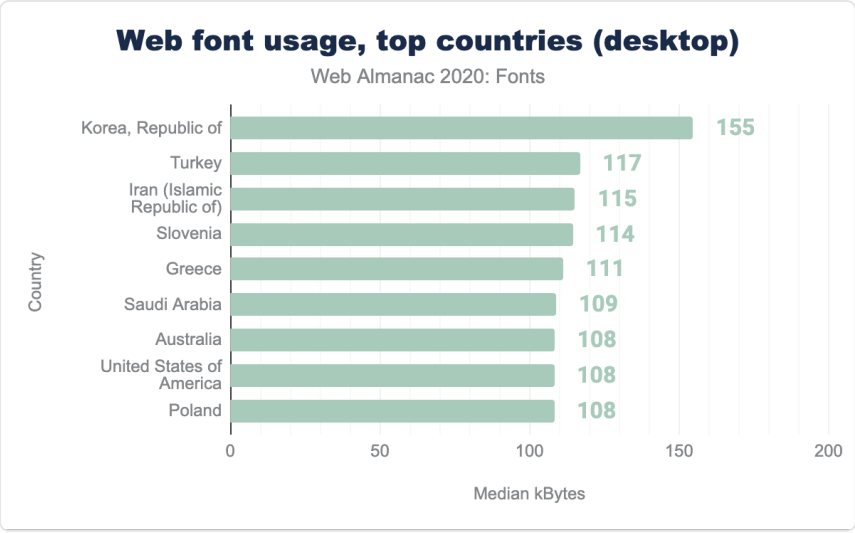


図4.3. ウェブフォント使用率、上位国（デスクトップ）。

HTTP Archiveのディスカッションフォーラムには、web font usage by country[216]という興味深いスレッドがあり、本章で使用したクエリにも影響を与えています。アジアの言語用に制作された書体の数が多いことを考えると、それらのフォントをより効率的に提供する技術が利用可能になれば、アジア地域での使用率は上昇すると考えられます。

サービスで提供する

70.3%

図4.4. フォントホスティングサービスの中でのGoogle Fontsの人気。

Google Fonts[217]が依然として圧倒的に人気のあるプラットフォームであることについて驚くことではないと思われますが、実際に使用割合は2019年から5%近く減少し、約70%となっています。Adobe Fonts[218]（旧Typekit）も同様に3%ほど下がっていますが、Bootstrapの使用率[219]は約3%から6%以上に伸びています（複数のプロバイダーからの集計）。なお、Bootstrapの最大のプロバイダー（BootstrapCDN[220]）はFont Awesome[221]のアイコンフォント

216. https://discuss.httparchive.org/t/how-does-web-font-usage-vary-by-country/1649
217. https://fonts.google.com/
218. https://fonts.adobe.com/
219. https://getbootstrap.com/
220. https://www.bootstrapcdn.com/
221. https://fontawesome.com/

も提供しているのでBootstrap自体ではなく、アイコンフォントファイルを参照している古いバージョンがこのソースデータの上昇の背景にあるのかもしれません。

もう1つの驚きは、Shopify[222]で提供されるフォントの増加です。2019年には約1.1%だったのが、2020年には約4%になっており、このプラットフォームでホストされているサイトのWebフォントの使用率が大幅に上昇していることが明らかになっています。そのサービスが自社のCDNでホスティングするフォントをより多く提供しているためなのか、そのプラットフォームの利用が増えているためなのか、あるいはその両方なのかは不明です。しかしShopifyとBootstrapの使用量の増加は、Google Fonts以外では最大の増加量であり、非常に注目すべきデータです。

すべてのサービスが同じサービスではない

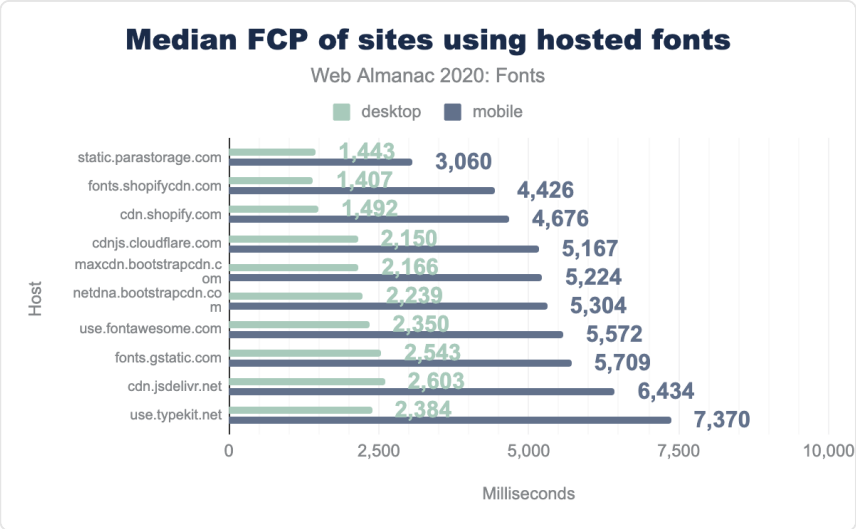


図4.5. ホストされたフォントを使用しているサイトのFCPの中央値。

興味深いのは、さまざまなフリー/オープンソースおよび商用サービスを使用しているサイトの速度の違いです。コンテンツの初回ペイント（FCP）と最大のコンテンツフルペイント（LCP）の時間を見ると、Google Fontsを使用しているサイトはほぼ中央に位置していますが、一般的には中央値よりも少し遅くなっています。データセットの中でもっとも高速なサイトはShopifyとWix（`parastorage.com`からアセットを提供）であり、これらのサイトは少数の高度に最適化されたファイルに焦点を当てていると推測されます。一方、Googleは、（言語によって）サイズが大きく異なるWebフォントをグローバルに提供しており、その結果、中央値の時間が若干遅くなっていると思われます。

222. https://www.shopify.com/

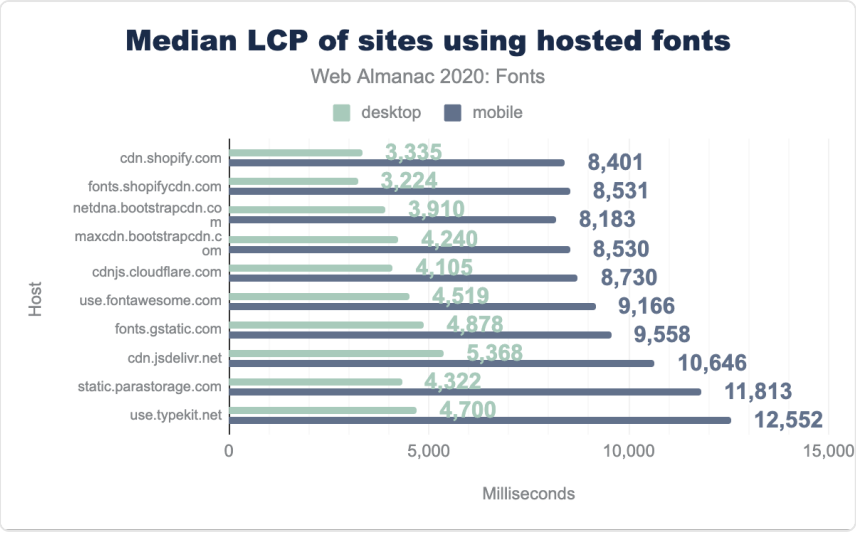


図4.6. ホストされたフォントを使用しているサイトのLCPの中央値。

Adobe（`use.typekit.net`）やMonotype（`fast.fonts.com`）などの商用サービスを見ていると、デスクトップではGoogle Fontsと同じかわずかに速い傾向があるが、モバイルでは明らかに遅いという点が興味深いです。これまでの常識では、これらのサービスで使用されているトラッキングスクリプトが大幅に速度を低下させていると考えられていましたが、現在では以前ほど問題になっていないようです。私たちが測定しているのはサイトのパフォーマンスであり、必ずしもフォントホストのパフォーマンスではないことは事実ですが、これらのトラッキングスクリプトはクライアントでのフォント読み込みに影響を与えるため、これらの見解を含めることは適切だと思われます。

セルフホスティングが必ずしも良いとは限らない

私たちがこのウェブサイトで発見したように[223]、ウェブサイトと同じドメインでフォントをセルフホスティングすると速くなることがありますが、データが示すように、必ずしもそうではありません。

223. *https://www.tunetheweb.com/blog/should-you-self-host-google-fonts/*

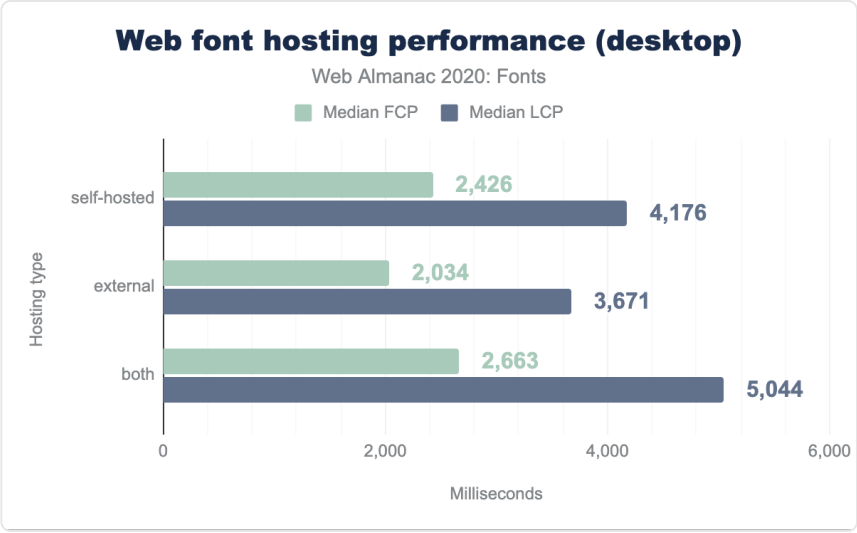


図4.7. Webフォントのホスティング性能、デスクトップ

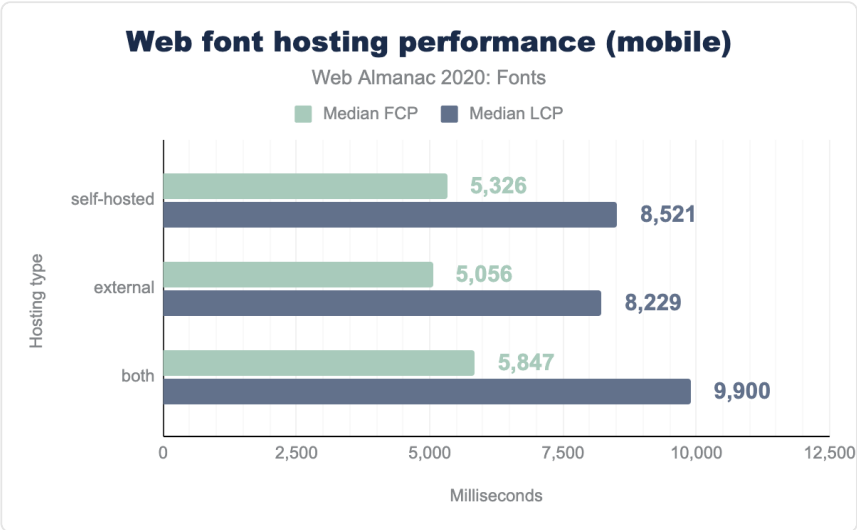


図4.8. Webフォントのホスティング性能、モバイル

上記のデータからホスティング戦略との因果関係を推測するのは健全でないでしょう、なぜなら関係を混同する他の変数があるからです。しかし、それはさておきセルフホスティングフォントを追加しても、必ずしもパフォーマンスが向上するとは限らないことがわかりました。ホスティングされたフォントソリューションは、多くの最適化（サブセット化、OpenType機能の削除、可能な限り小さいフォントフォーマットの確保など）を実行します

が、セルフホスティングの場合は必ずしも再現されないことがあります。

ローカルが必ずしも良いとは限らない

サイトのサーバーでフォントを自前でホストする方法の他に、システムにインストールされているフォントを、クライアントでも `font-face` 宣言で `local` を使用して使用する方法があります。 `local` の使用は、バイト数を節約できるため、議論の的[224]となっていますが、ローカルにインストールされたフォントのバージョンが古い場合には悪い結果になることもあります。2020年11月[225]現在、Google Fontsはモバイルプラットフォーム上のRobotoに対してのみ `local` を使用するように移行しており、それ以外の場合は常にネットワーク経由でフォントを取得しています。

ファーストペイントへの挑戦

Webフォントを統合する際のパフォーマンス上の最大の懸念点は、最初に読めるテキストが表示されるまでの時間が遅れることです。このような問題を軽減するために、2つの最適化技術があります。 `font-display` とリソースヒントです。

`font-display` の設定は、Webフォントの読み込みを待つ間の動作を制御し、一般的にはパフォーマンスと視覚的な豊かさのトレードオフとなります。 もっともポピュラーなのは `swap` で、約10%のWebページで使用されています。これは、Webフォントがすぐ読み込まれない場合に予備フォントを使用して表示し、読み込まれたときにWebフォントを入れ替えるというものです。その他の設定としては、テキストの表示を一切行わない（Flash効果の可能性を最小限に抑える） `block` 、 `swap` と似ていますが、適度な時間でフォントが読み込まれない場合に使用する `fallback` 、すぐに諦めて予備フォントを使用する `optional` などがあります。これを使用しているのは、パフォーマンスを重視するウェブページの1%に過ぎません。

224. *https://bramstein.com/writing/web-font-anti-patterns-local-fonts.html*
225. *https://x.com/googlefonts/status/1328761547041148929?s=19*

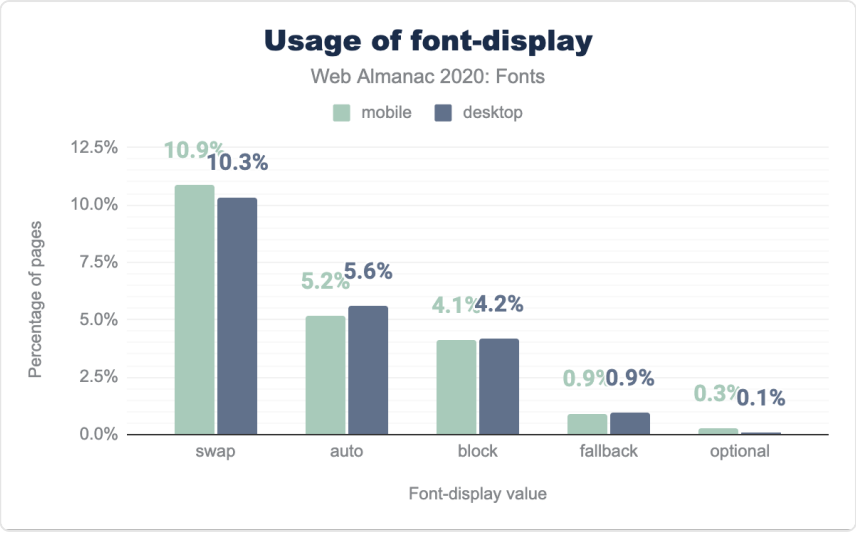


図4.9. font-displayの使い方。

これらの設定がコンテンツの初回ペイントと最大のコンテンツフルペイントに与える影響を分析できます。驚くことではありませんが、`optional`の設定は最大のコンテンツフルペイントに大きな影響を与えます。また、コンテンツの初回ペイントにも影響がありますが、これは因果関係よりも相関関係の方が強いかもしれません。というのも、`block`を除くすべてのモードは、”極端に小さいブロック期間”の後に *何らかの*テキストを表示するからです。

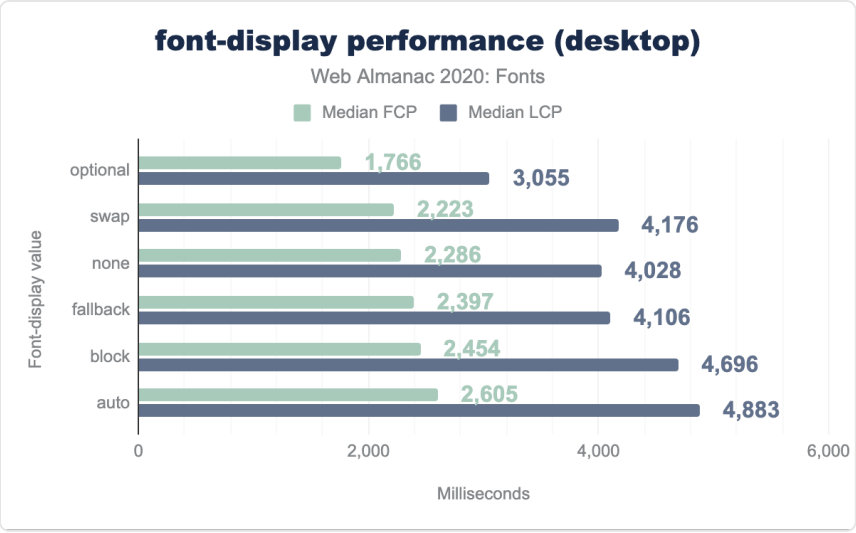


図4.10. デスクトップでの `font-display` のパフォーマンスを改善しました。

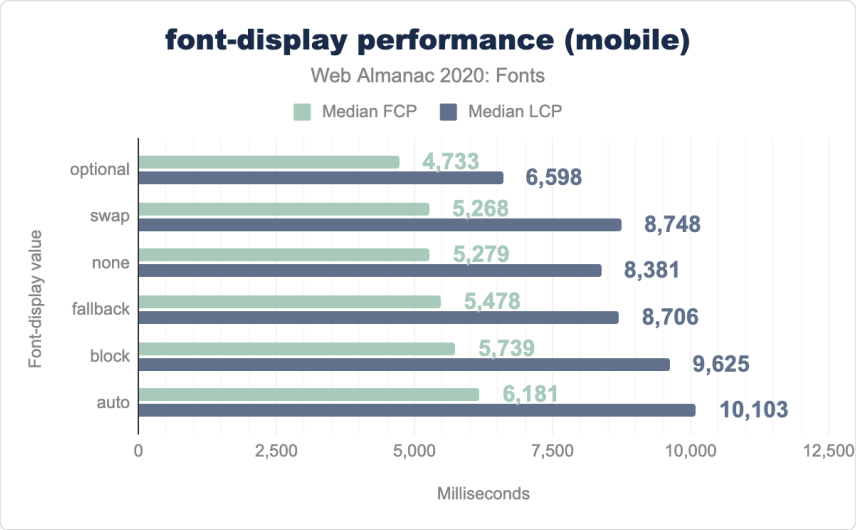


図4.11. モバイルでの `font-display` のパフォーマンスを向上させました。

このデータからは、他にも2つの興味深い推論があります。`block` の設定は、とくにモバイルではFCPに大きな影響を与えると予想されますが、実際にはその効果はそれほど大きくありません。これは、フォントアセットの待ち時間がWebページ全体のパフォーマンスを制限する要因になることはほとんどないことを示唆していますが、画像などのリソースが多くないページでは大きな要因になることは間違いありません。`auto` の設定（指定しなかった場

合の設定）は、ブラウザ次第です。デフォルトではほとんどの場合にブロックされます[226]ので、`block`とよく似ています。

最後に、`fallback`を使用する正当な理由のひとつとして、`swap`と比較して最大コンテンツの描画時間を向上させることが挙げられます（デザイナーの視覚的意図を尊重する可能性が高いです）。そのためか、この設定は人気がなく、約1%のページでしか使われていません。

Google Fontsの推奨統合コードに`swap`が追加されました。現在使用していない場合は、これを追加することで、とくに低速回線のユーザーのパフォーマンスを向上させることができるかもしれません。

リソースのヒント

`font-display`はフォントの読み込みが遅いときにページの表示を速くできますが、リソースヒントはWebフォントアセットの読み込みをカスケードの早い段階に移動させることができます。

通常、Webフォントの取得は2段階のプロセスで行われます。第一段階はCSSの読み込みで、CSSには実際のフォントバイナリへの参照が（`@font-face`セクションに）含まれています。

これは、ホスティングされたフォントソリューションにとくに関係しています。フォントが、必要であることがわかってはじめて、そのサーバーへの接続を開始できます。これはさらに、サーバーへのDNSクエリと、実際の接続の開始（最近では通常HTTPSの暗号ハンドシェイクを伴う）に分かれます。

226. *https://nooshu.github.io/blog/2020/02/23/improving-perceived-performance-with-the-css-font-display-property/*

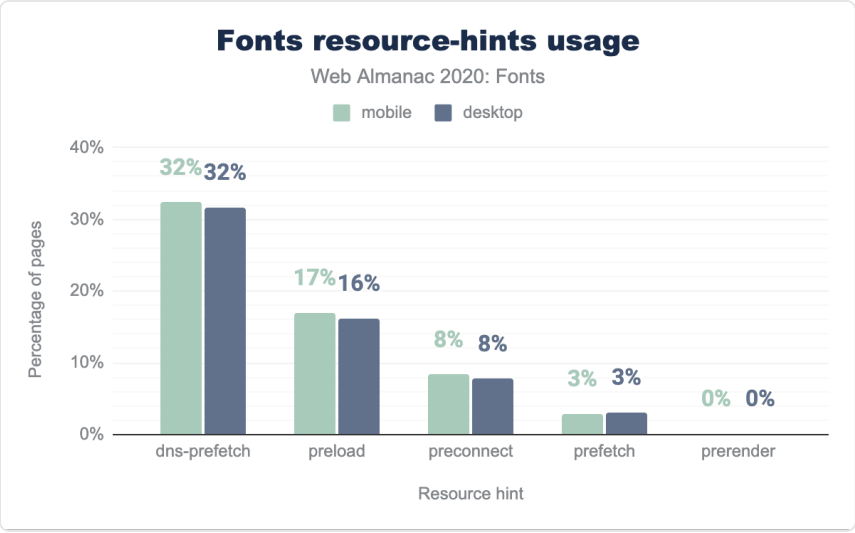


図4.12. フォントに関するリソース・ヒントの利用

HTMLにリソースヒント要素[227]を追加すると、2つ目の接続が早く始まります。さまざまなリソースヒントの設定は、実際のフォントリソースのURLを取得するまでの距離を制御します。もっとも一般的なのは（ウェブページの約32%）`dns-prefetch`ですが、ほとんどの場合、より良い選択肢があります。

次に、これらのリソースヒントがページのパフォーマンスに影響を与えるかどうかを見てみましょう。

227. https://www.w3.org/TR/resource-hints/#resource-hints

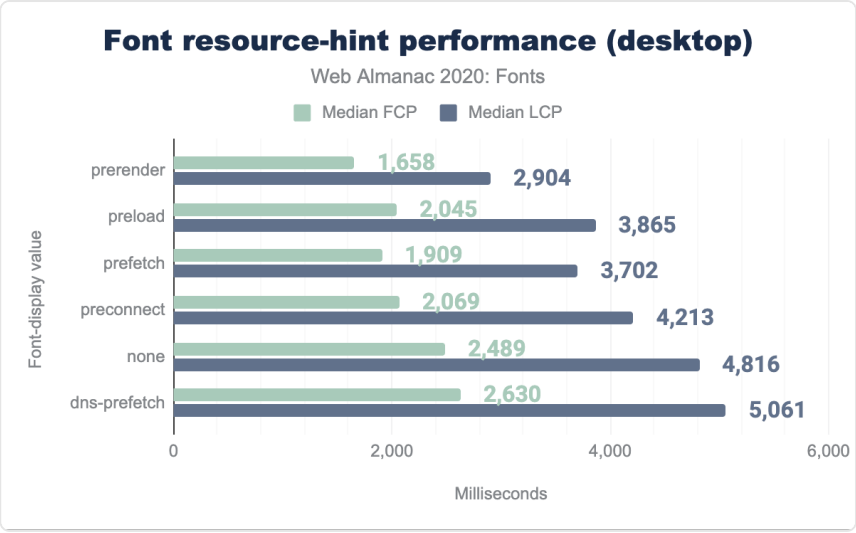


図4.13. リソース・ヒント パフォーマンス、デスクトップ

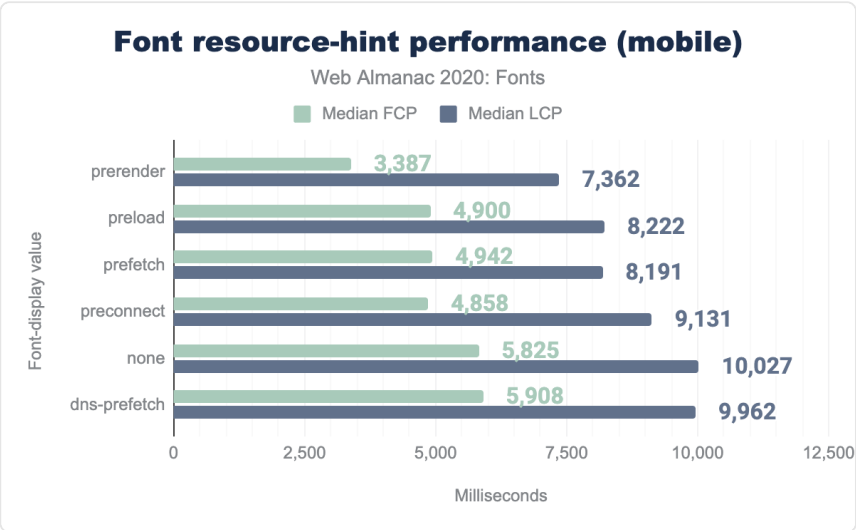


図4.14. リソース・ヒント・パフォーマンス、モバイル

このデータを分析すると、`dns-prefetch`設定は、もっとも一般的な設定であるにもかかわらず、あまりパフォーマンスを改善していないことがわかります。おそらく、人気のあるWebフォントサーバのDNSは、いずれにせよキャッシュされていると思われます。他の設定では、使いやすさ、柔軟性、パフォーマンスの向上の点で、`preconnect`がもっとも優れています。2020年3月現在、Google Fontsは、HTMLソースのCSSリンクの直前にこの行を追加

することを推奨している。

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

`preconnect` の使用率は昨年の2%から8%へと大幅に増加していますが、まだまだ潜在的なパフォーマンスは残っています。この行を追加することは、Google Fontsを使用しているWebページにとって唯一の最適化となるかもしれません。

パイプラインをさらに進めてフォントアセットのプリロードやプリレンダリングを行いたいと思うかもしれませんが、レンダリングエンジンの性能に合わせてフォントを微調整したり、後述する `unicode-range` の最適化など他の最適化と衝突する可能性があります。リソースをプリロードするためにはどのリソースをロードするかを *正確* に知る必要があり、タスクに最適なリソースは、HTMLオーサリング時には容易に入手できない情報に依存する可能性があります。

ホーム・オン・ザ・（Unicode）・レンジ

フォントはますますたくさんの言語へ対応するようになっています。他のフォントでは、筆記体系（とくに日中韓）が必要とするために大量のグリフを持つことがあります。いずれの理由であれ、ファイルサイズは大きくなります。ウェブページが実際には多言語辞書ではなく、フォントの機能のほんの一部しか使っていない場合は残念なことです。

古い方法としては、HTML作成者が明示的にフォントサブセットを指定する方法があります。しかしこれにはコンテンツに関するより深い知識が必要で、コンテンツがフォントでサポートされている文字を使用していても選択されたサブセットではサポートされていない場合、「身代金要求」効果を発生させる危険性があります。フォールバックの仕組みについては、Marcin Wichary氏の素晴らしいエッセイ When fonts fall[228] をご覧ください。

この問題を解決するには、`unicode-range` で示される静的なサブセットの方が良い方法です。フォントはサブセットにスライスされ、それぞれ個別の `@font-face` ルールで、そのスライスのUnicodeカバレッジを `unicode-range` 記述子で示します。ブラウザは、レンダリングパイプラインの一部としてコンテンツを分析し、そのコンテンツのレンダリングに必要なスライスのみをダウンロードします。

アルファベット言語の場合、これは一般的にうまく機能しますが、異なるサブセット内のキャラクター間のカーニングが、悪くなることがあります。アラビア語、ウルドゥー語、多くのインド系言語など、グリフシェーピングに依存する言語では、静的サブセットを使用するとテキストレンダリングがよくおかしくなります。また日中韓では連続したUnicode範囲に

228. *https://www.figma.com/blog/when-fonts-fall/*

基づく静的サブセットは、特定のページで使用される文字がさまざまなサブセットに、ほぼランダムに散らばっているためほとんどメリットがありません。このような問題があるため、静的サブセットを正しくかつ効率的に使用するには、慎重な分析と実装が必要となります。

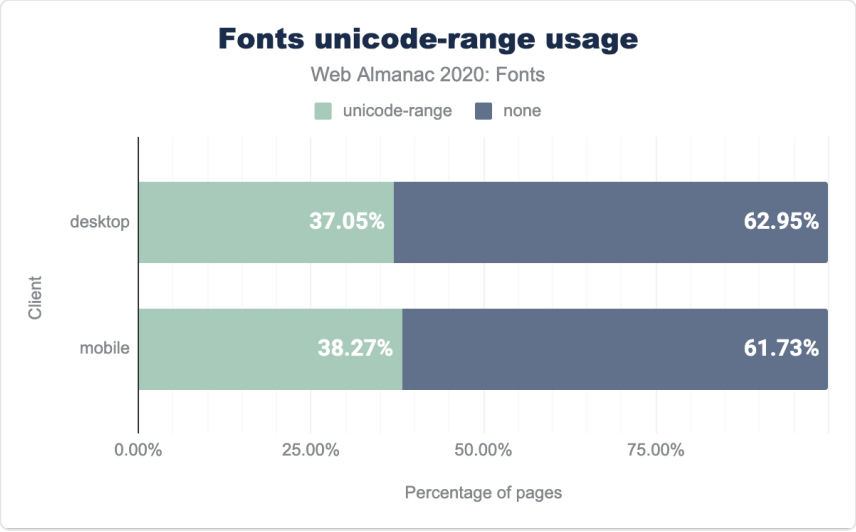


図4.15. `unicode-range` の使い方。

テキストレイアウトがUnicodeをグリフにマッピングする方法には多くの複雑さがあるため、`unicode-range` を正しく適用するのは難しいが、Google Fontsはこれを自動的かつ透過的に行う。しかし、Google Fontsはこれを自動的かつ透過的に行う。いずれにせよ、現在の使用率はデスクトップで37%、モバイルで38%となっている。

フォーマットとMIMEタイプ

WOFF2はもっとも優れた圧縮フォーマットで、現在ではInternet Explorerのバージョン11以前を除くほぼすべてのブラウザで サポートされています[229]。WOFF2ソースのみで `@font-face` ルールを使用してウェブフォントを提供することは、 *ほぼ* 可能です。このフォーマットは、提供されるフォントの約75%を占めています。

229. https://caniuse.com/woff2

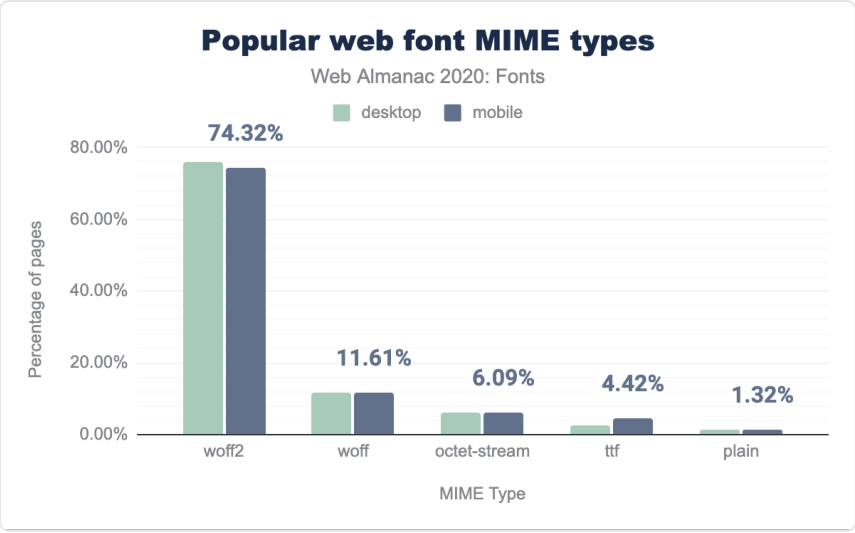


図4.16. ポピュラーなウェブフォントのMIMEタイプ。

WOFFは、より古く効率の悪い圧縮メカニズムですがほぼ全世界でサポートされており、提供されているフォントの11.6%を占めています。ほとんどすべての場合（Internet Explorer 9-11は例外）、フォントをWOFFとして提供することはパフォーマンスを犠牲にすることであり、セルフホスティングのリスクを示しています。ホストサービスを利用すると、最適なフォーマットが選択され、関連するすべての最適化が保証されます。

世界のブラウザシェアの約1.5%を占めるInternet Explorerの古いバージョン（6〜8）は、EOTフォーマットしかサポートしていません。これらはMIMEフォーマットのトップ5には含まれていませんが、最大限の互換性を確保するために必要です。

OTFやTTFファイルなどの非圧縮フォントは圧縮フォントに比べて2〜3倍の大きさですが、それでも提供されるフォントの約5%を占めておりモバイルでの利用が、多いことが分かります。もしこれらのフォントを提供しているのであれば、最適化が可能であることを示すレッドフラッグとなるはずです。

人気のあるフォント

アイコンフォントは、もっとも人気のあるウェブフォントのトップ10のうち半分を占めており、残りはクリーンで堅牢なサンセリフ書体のデザインです（このランキングではRoboto Slabが19位、Playfair Displayが26位に入っており他のスタイルのデビューとなっていますがセリフ書体のデザインは分布の末尾によく現れています）。

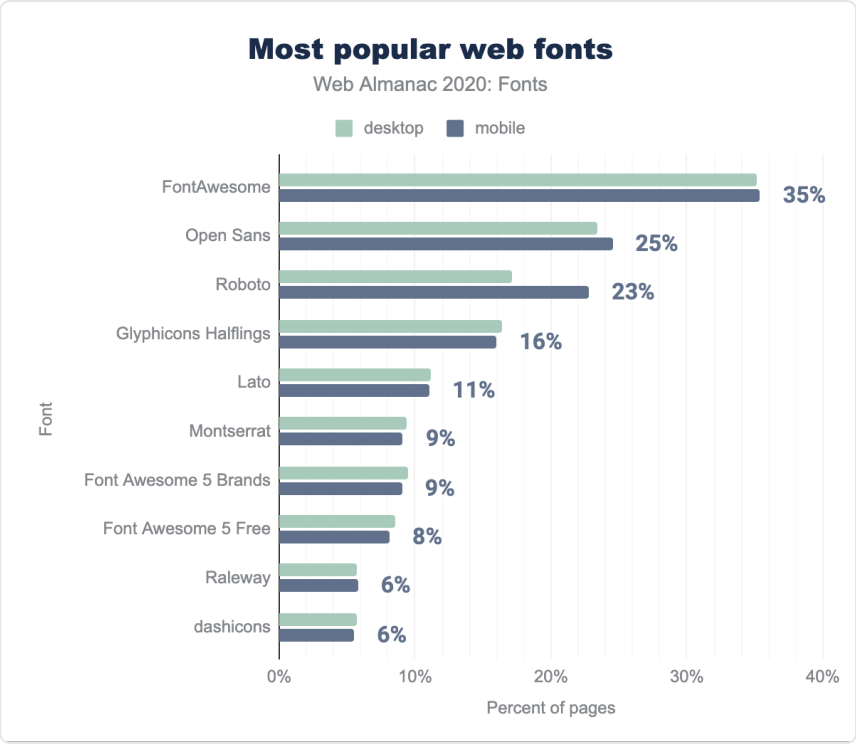


図4.17. 人気のある書体です。

注意点としてはもっとも人気のあるフォントを決定する際、測定方法によって異なる結果が、得られる可能性があります。上のグラフは指定されたフォントを参照する`@font-face`ルールを含むページの数をカウントしたものです。これは、複数のスタイルを一度だけカウントするものであり、シングルスタイルのフォントに有利に働くことは間違いありません。

カラーフォント

カラーフォントは、何らかの形でほとんどのモダンブラウザでサポートされていますが使用率はまだ皆無に近い状態です（合計で755ページ、その大部分はChromeではサポートされていないSVGフォーマットです）。問題の1つは、フォーマットの多様性で、実際には4つのフォーマットが広く使用されています。これらには、ビットマップ形式とベクター形式があります。2つのビットマップフォーマットは技術的には非常によく似ていますが、SBIX（元々はApple独自のフォーマット）はFirefoxではサポートされておらず、CBDT/CBLCはSafariではサポートされていません。

COLRベクトルフォーマットは、すべての主要なモダンブラウザでサポートされていますが、かなり最近になってサポートされました。4つ目のフォーマットは、OpenType（SVGフォントと混同しないように）にSVGを埋め込むものですが、Chromeではサポートされていません。OpenTypeのSVGの欠点は、現代のウェブデザインでますます重要になっている、フォントのバリエーションがサポートされていないことです。とくにグラデーションやクリッピングのサポートについては、COLRの将来のバージョンに向けて開発が進められているため、COLRフォーマットが主流になると思われます。ベクターフォーマットは通常、画像よりもはるかに小さく、また、より大きなサイズにきれいに拡大できます。

ウェブでのカラーフォントのサポートが不十分な理由のひとつは、カラーをフォントファイル自体に焼き付ける必要があることです。同じ書体を3つの異なる色の組み合わせで使用する場合、ほぼ同じファイルを3回ダウンロードしなければならず、色を変更するにはフォントエディターを使用しなければなりません。

CSSには、フォントのカラーパレットを上書きしたり、置き換えたりする機能がありますが[230]、これはまだブラウザに実装されておらず、カラーウェブフォントの導入のしやすさを阻んでいるのが現状です。

カラーフォントの用途は、おそらく絵文字がほとんどだと思いますが、その機能は汎用的でカラーフォントには多くのデザインの可能性があります。カラーWebフォントはまだ普及していませんが、その技術は、ファイルフォーマットの互換性があまり問題にならないシステム絵文字の配信に多用されています。

ブラウザのサポートは非常に断片的であるため、カラーフォントはまだcaniuse.comでは追跡されていませんが、そのための課題が公開されています[231]。

カラーフォントについては、例を含めて多くの情報がcolorfonts.wtf[232]にあります。

可変フォント

11.00%

図4.18. モバイルでの可変フォントの使用

可変フォントは、確かに今年の最大の話題の1つです。デスクトップでは10.54%、モバイルでは11.00%が可変フォントを使用しています。これは、昨年の平均1.8%からの増加で、大きな成長要因となっています。デザインの自由度が高くとくに同じページで同じフォントを

230. https://drafts.csswg.org/css-fonts-4/#font-palette-values
231. https://github.com/Fyrd/caniuse/issues/1018
232. https://www.colorfonts.wtf/

複数使用している場合、2つのフォントサイズが、小さくなる可能性があるからです。

この増加の最大の要因はGoogle Fontsがページで使用されているウェイトが十分で、ブラウザがサポートしている場合に、人気のあるフォントの多くを可変フォントとして提供するようなったことにあると思われます。使用されているCSSを変更することなく、またWeb制作者の手を煩わせることなくパフォーマンスを向上させることができる可変フォントの「スワップイン」機能は、この技術の実行可能性を顕著に示している。

バリアブルフォントフォーマットを簡単に説明すると、1つのフォントファイルが複数の役割を果たすということです。ウェイトや幅さらにはイタリックなど、それぞれに個別のフォントファイルを用意するのではなく、それらをすべて1つの効率の良いファイルに収めることができます。このファイルは、CSS（またはCSSをサポートする他のアプリケーション）を介して、指定された軸の値の組み合わせでフォントをレンダリングできます。標準化された、つまり「登録された」軸がいくつかあるほか、フォントデザイナーが独自の軸を定義してユーザーに公開することもできます。

ウェイト（`wght`）はレギュラーやボールドライトといった伝統的な概念に対応し、幅（`wdth`）はコンデンスやエクステンデッドといったスタイルに対応し、スラント（`slnt`）はフォントの斜めの角度を意味し、イタリック（`ital`）は通常フォントを斜めにし、特定のグリフを代替スタイルに置き換えます。そして、オプティカルサイズ（`opsz`）は、ウェブでは比較的新しいものを意味しますが、実際には数百年前の金属活字の制作で一般的だった技術の復活です。歴史的にはオプティカルサイジングとは、物理的に小さいサイズのフォントで読みやすさを高めるためにストロークのコントラスト（太い線と細い線）を減らして文字の間隔を広げ、逆にかなり大きなサイズで表示されたフォントでは、コントラストを増やして文字の間隔を狭めることを意味します。デジタルタイプでこの機能を有効にすると、1つのフォントを非常に小さいサイズで使用した場合と大きいサイズで使用した場合とで、見た目や動作を大きく変えることができます。variablefonts.io[233]では、この機能について詳しく説明し、多くの例を見ることができます。

233. *https://variablefonts.io*

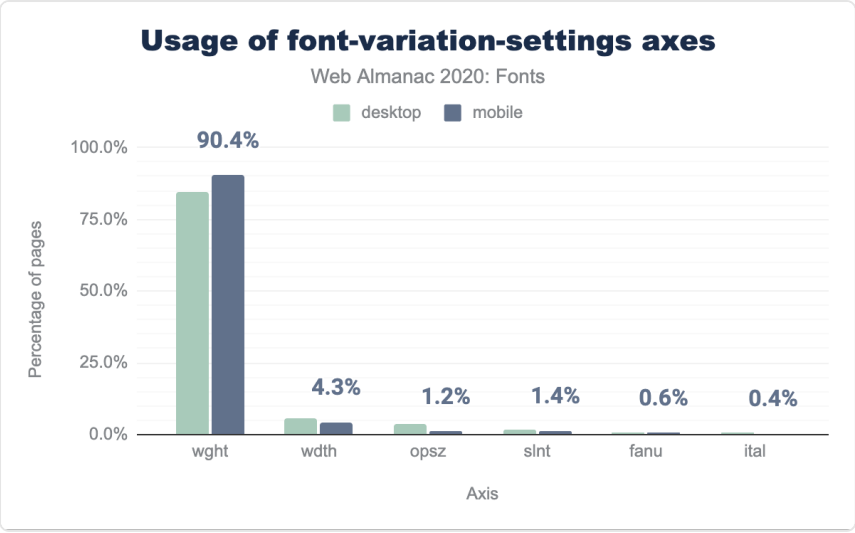


図4.19. font-variation-settingsの軸の使い方。

もっともよく使われている軸は`wght`（重さを制御する）で、デスクトップで84.7%、モバイルでは90.4%となっています。しかし、`wdth`（幅）は可変フォントの使用率の約5%を占めている。2020年、Google Fontsは幅軸と重さ軸の両方を持つ2軸フォントの提供を開始した。

この2つの軸については、可変フォントをサポートするすべてのブラウザで完全にサポートされているため、低レベルの`font-variation-settings`構文ではなく、`font-weight`および`font-stretch`を使用するのが望ましい方法であることを覚えておいてください。重さを`font-weight: [number]`で重みを設定し、`font-stretch.html`で幅を設定します。これにより、可変フォントの読み込みに失敗した場合でも、エンドユーザーに適切なレンダリングを提供できます。また、カスケードによるスタイルの通常の継承を変更することもありません。

オプティカルサイズ（`opsz`）機能は、可変フォントの使用量の約2%に使用されています。意図した表示サイズに合わせてフォントの外観を調整することで微妙ではありますが、視覚的な洗練度が向上するため、この機能には注目が集まっています。また光学的サイズの定義方法について、現在のクロスブラウザやクロスプラットフォームの不確実性が解消されれば、使用量は増加すると思われます。オプティカルサイズ機能の魅力的な点は、`auto`の設定で、自動的に変化することです。つまり、開発者は`opsz`機能を持つフォントを使うだけで、洗練された恩恵を受けることができます。

可変フォントの使用には多くの利点があります。軸が増えるごとにファイルサイズは大きくなりますが、一般的にはある書体の2つまたは3つ以上のウェイトが使用されている場合には

可変バージョンの方が、総ファイルサイズが同等か小さくなる可能性が高いというのが転換点のようです。これは、Google Fontsで提供されているバリアブル・フォント[234]の劇的な増加によって裏付けられている。

また、可変フォントを採用して活用することで、より変化に富んだデザインを実現することもできます（利用可能なウェイトや幅の範囲をより多く使用できます）。幅軸を使用することで、とくに大きな見出しや長い言語の場合、小さな画面での行の折り返しを改善できます。代替ライトモードの採用が進んでいるため、モードを切り替える際にフォントウェイトを少しずつ調整することで、読みやすさを向上させることができます（使い方や実装方法については、variablefonts.io[235]を参照してください）。

結論

Webフォントの技術は、圧縮などの技術的な改良を重ねながらかなり成熟していますが、新しい機能も続々と登場しています。可変フォントに対するブラウザのサポートはかなり充実してきており、この機能は昨年もっとも成長した機能です。

cache partitioning[236]の出現により、CDNフォントリソースのキャッシュを複数のサイトで共有することによるパフォーマンス上のメリットが減少しているため、パフォーマンスの状況は多少変化しています。CDNを利用するよりも、サイトと同じドメインでより多くのフォント資産をホスティングする傾向は、おそらく今後も続くでしょう。それでも、Google Fontsのようなサービスは高度に最適化されており、`swap`や`preconnect`の使用などのベストプラクティスにより、追加のHTTP接続による影響の多くが緩和されます。

可変フォントの使用は非常に加速しており、とくにブラウザやデザインツールのサポートが向上すれば、この傾向は間違いなく続くでしょう。また、2021年はカラーWebフォントの年になるかもしれません。技術的には実現していますが、まだ実現していません。

最後に、W3CのWeb Font Working Groupで研究されているWebフォント技術の新しいコンセプトについて触れておきましょう。Progressive Font Enrichmentです。PFEは、本章で指摘した多くの課題に対する回答として設計されています。すなわちグリフ数の多いフォントファイル（アラビア語や日中韓のフォントなど）や、より大きな多軸フォントやカラーフォントを使用する場合、または単に遅いネットワーク接続環境を使用する場合のパフォーマンスとユーザーエクスペリエンスに対処するものです。

簡単に言うと、あるページのコンテンツをレンダリングするために、あるフォントファイルの一部だけをダウンロードすればよいというコンセプトです。その後のページ読み込みでは、各ページのレンダリングに必要なグリフのみを含むフォントファイルの「パッチ」が配

234. *https://fonts.google.com/?vfonly=true*
235. *https://variablefonts.io*
236. *https://developers.google.com/web/updates/2020/10/http-cache-partitioning*

信されます。このように、ユーザーは一度にフォントファイル全体をダウンロードする必要はありません。

プライバシーの確保や下位互換性の確保など、さまざまな詳細を詰めていく必要がありますが、初期の研究では非常に有望な結果が得られており今後2、3年のうちにこの技術が広くウェブに普及することが期待されています。この技術については、Jason Pamental氏による紹介文[237]や、W3Cのサイトにある Working Group Evaluation Report[238]の全文で詳しく知ることができます。

著者

Raph Levien

@raphlinus　raphlinus　https://levien.com

Raph Levienは、カリフォルニア大学バークレー校でフォントデザインツールに関する博士号を取得するなど、35年以上にわたってフォントに携わってきました。2010年にチームを共同設立した後、フォント技術研究者としてGoogle Fonts[239]に再参加しています。

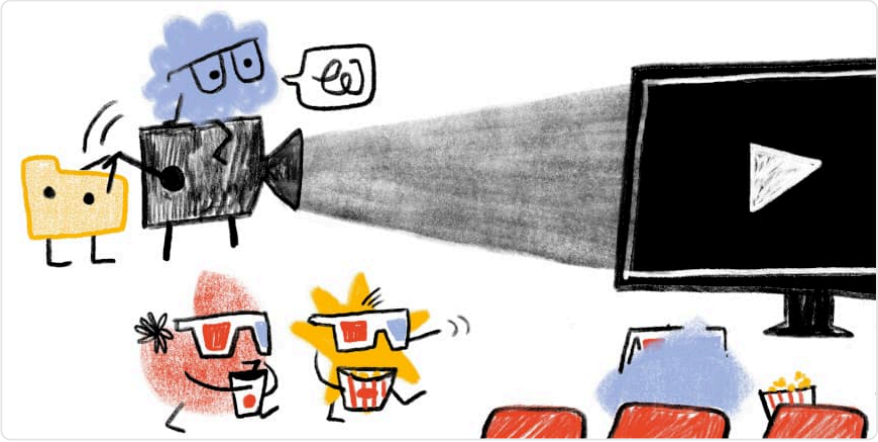
Jason Pamental

@jpamental　jpamental　https://rwt.io

デザイナー、試行錯誤する人、タイポグラファー。 レスポンシブタイポグラフィの作家は、W3Cの専門家、そしてWeb上のタイポグラフィに焦点を当てた10年以上の経験を述べました。

237. *https://rwt.io/typography-tips/progressive-font-enrichment-reinventing-web-font-performance*
238. *https://www.w3.org/TR/2020/NOTE-PFE-evaluation-20201015/*
239. *https://fonts.google.com/*

部 I 章 5
メディア



Tamas Piros、 Ben Seymour と Eric Portis によって書かれた。
Nicolas Hoizey、 Colin Bendell、 Doug Sillars と Navaneeth Krishna によってレビュ。
Stefan Matei による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

今日、私たちはビジュアルウェブの世界に生きており、メディアがウェブサイトの魂を提供しています。Webサイトでは画像や動画を使って情報を提供したり、楽しませたりする視覚的なストーリーを伝え、オーディエンスを惹きつけています。本章では、私たちがウェブ上で画像や動画をどのように使っているか（場合によっては誤用しているか）を分析します。

イメージ

“百聞は一見にしかず“と言いますが、バイト換算すると一桁も二桁も高いことが多いのです。

画像は、瞬時のコミュニケーションと、生来の感情的な反応を引き起こすことができるという、もっとも強力な組み合わせです。しかし、画像はテキストに比べてはるかに重いため、

ユーザー体験の妨げにならないよう配慮した実装が必要です。ここでは、モダンブラウザの機能がどの程度活用されているかを見てみましょう。

画像のレスポンシブHTMLマークアップ

JavaScriptを使ってメディアを埋め込むアプローチは無数にありますが、私たちはさまざまな形のHTMLマークアップが継続的に利用されていることに興味を持ちました。`<picture>`要素や、`srcset`属性、`sizes`属性など、いくつかの*responsive images*のアプローチは、2014年にはじめて紹介されて以来、支持が高まっています。

Srcset

`srcset`属性は、ユーザーエージェントが候補リストから読み込むのに最適なメディアアセットの決定を試みることを可能にします。

たとえば、以下のように。

```
<img srcset="images/example_3x.jpg 3x, images/example_2x.jpg 2x"
      src="images/example.jpg" alt="...">
```

全ページの約26.5%が`srcset`を含むようになりました。

ユーザーエージェントが選択できる画像の数は、2つの主要なパフォーマンス要因に直接影響を与えます。

1. イメージブレイクポイント[240]（パフォーマンスバジェットを満たすために）
2. キャッシングの効率化

画像候補の数が少ないほど、アセットをキャッシュする可能性が高くなります。また、CDNを使用している場合は、クライアントの最寄りのエッジノードで利用できる可能性が高くなります。しかし、メディアのサイズの違いが大きければ大きいほど、デバイスやコンテキストに適していないメディアを提供してしまう可能性が高くなります。

240. *https://cloudfour.com/thinks/responsive-images-101-part-9-image-breakpoints/*

Srcset：画像候補の数量

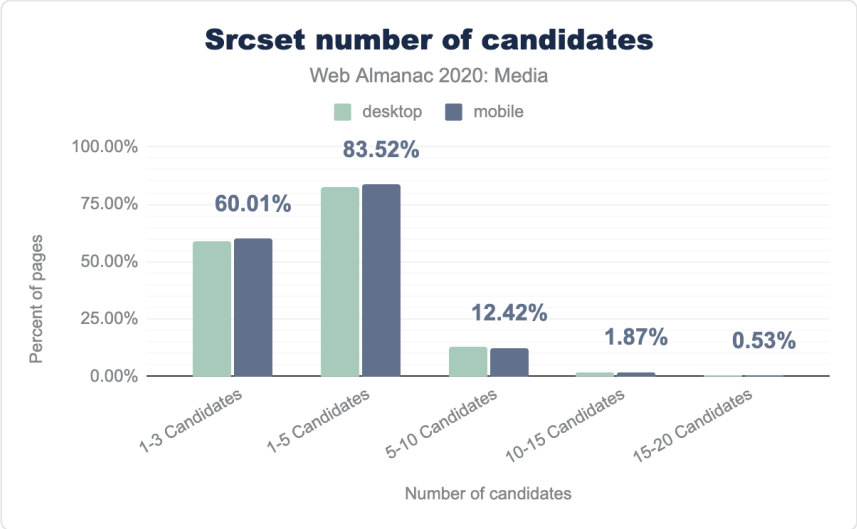


図5.1. Srcset候補者数

先に述べたキャッシュの非効率性に加えて、次元的なバリエーションの数が増えると、使用するメディアパイプラインやサービスの複雑さと、必要なメディアストレージの両方が増加します。

このデータを見る際には、いくつかのプラットフォーム（WordPress[241]など）では、多数のサイトに影響を与える自動化されたアプローチを使用していることに注意してください。

Srcset：記述子

候補リストをユーザエージェントに提供する際、候補画像に注釈を付けるための2つの仕組みがあります。それは、`x`記述子と`w`記述子です。

`x`記述子は、特定のリソースのデバイスピクセルの比率を記述します。たとえば、`2x`の記述子は、特定の画像リソースが各軸の2倍のサイズ（4倍のピクセルを含む）で、`window.devicePixelRatio`が`2`のデバイスに適していることを示します。同様に、`3x`の記述子は、9倍のピクセル数を意味し、これはもちろん、かなりのペイロードを意味します。

241. https://make.wordpress.org/core/2015/11/10/responsive-images-in-wordpress-4-4/

```
<img srcset="images/example_3x.jpg 3x, images/example_2x.jpg 2x"
      src="images/example.jpg" alt="...">
```

`w` 記述子は、候補者のピクセル幅を記述するもので、適切な画像を選択するために使用される `sizes` 属性と一緒になっています。

```
<img srcset="images/example_small.jpg 600w, images/
example_medium.jpg 1400w, images/example_large.jpg 2400w"
      sizes="100vw"
      src="images/example_fallback.jpg" alt="...">
```

どちらの方法でも、ユーザーエージェントは、最適な画像候補を評価する際に、現在のデバイスのピクセル比を数学的に考慮できます。

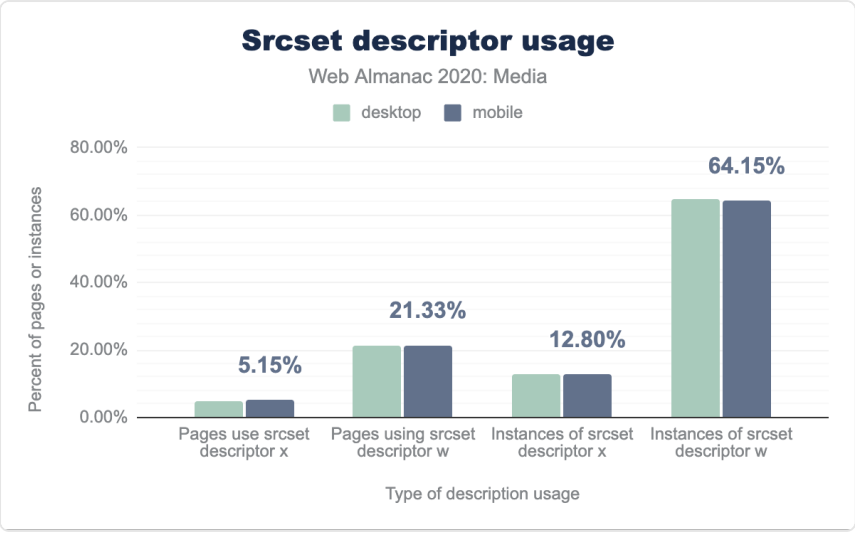


図5.2. Srcset記述子の使い方。

レスポンシブイメージの初期には、一部のブラウザは `x` 記述子しかサポートしていませんでしたが、現在は明らかに `w` 記述子がもっとも好まれています。

寸法ごとに間隔をあけて画像候補を選択するのが一般的ですが（720px、1200px、1800pxなど、あらかじめ選択された幅のセットですべての画像をレンダリングする）、より直線的なペイロードステップを与えるアプローチもあります（たとえば、50kbの差がある一連のリ

ソース）。Responsive Image Breakpoints Generator[242]のようなツールは、これを容易にするための役立ちます。

サイズ

`sizes`属性がないと、ユーザーエージェントは、画像がビューポートの幅いっぱいに配置されているという最悪のケースを想定して計算を行います。この属性があれば、ブラウザは画像の実際のレイアウトサイズについてより多くの情報を得ることができ、より適切な選択をできます。

たとえば、以下のように。

```
<img sizes="(min-width: 640px) 50vw, 100vw"
      srcset="images/example_small.jpg 600w, images/
example_medium.jpg 1400w, images/example_large.jpg 2400w"
      src="images/example_fallback.jpg" alt="...">
```

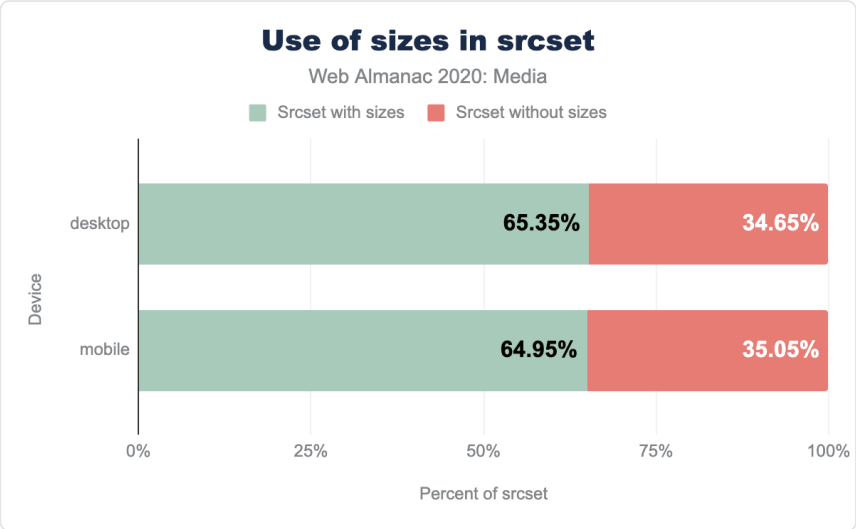


図5.3. srcset でのサイズの使用。

2020年のデータでは、`srcset`を使用しているサイトの約35%が`sizes`と組み合わせていませんでした。ブラウザは喜んで`sizes="100vw"`のデフォルトに戻りますが、この属性を

242. https://www.responsivebreakpoints.com/

オフにしておくと、技術的に正しくありません[243]。この見落としにより、もっとも適切な画像候補を決定する数学に欠陥があり、しばしば不必要に大きな画像がリクエストされてしまうケースが定期的に発生しています。

この件について多くの人に聞いたところ、`sizes`は正しく弾力的な方法で実装するのがとくに難しいそうです。これは、（CSSで管理・決定される）レイアウトと、（HTMLの）レスポンシブ・イメージ・マークアップとの間で、リソースを超えた整合性を確保する必要があるためです。

写真

`srcset`や`sizes`は、ビューポートやデバイス、レイアウトに適した寸法の画像をブラウザに提供するためのツールですが、`<picture>`要素は、より効果的な画像フォーマットの活用や「アートディレクション」の検討など、より洗練されたメディア戦略を提供します。

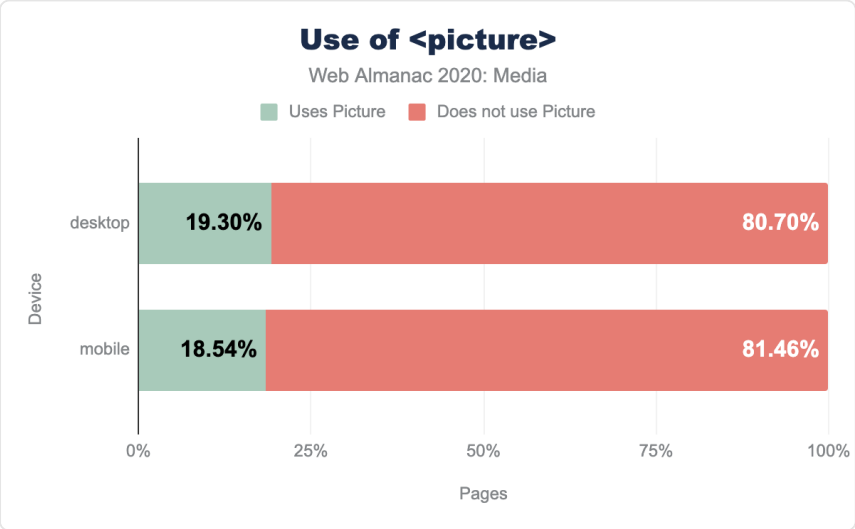


図5.4. `<picture>`の使用。

現在では、約19%のページで`<picture>`要素を使用して、少なくとも1つの画像を提供しています。

写真：フォーマット切り替え

バックエンドのロジックを使って、1つの画像URLから自動フォーマット切り替えを行うこ

243. *https://alistapart.com/blog/post/article-update-dont-rely-on-default-sizes/*

とができるサービスや画像CDNもありますが、同様の動作をマークアップだけで実現することができるのは、`<picture>`要素です。

```
<picture>
    <source type="image/webp" srcset="images/example.webp">
    <img src="images/example.jpg" alt="...">
</picture>
```

これを提供するフォーマットの数に分解します。

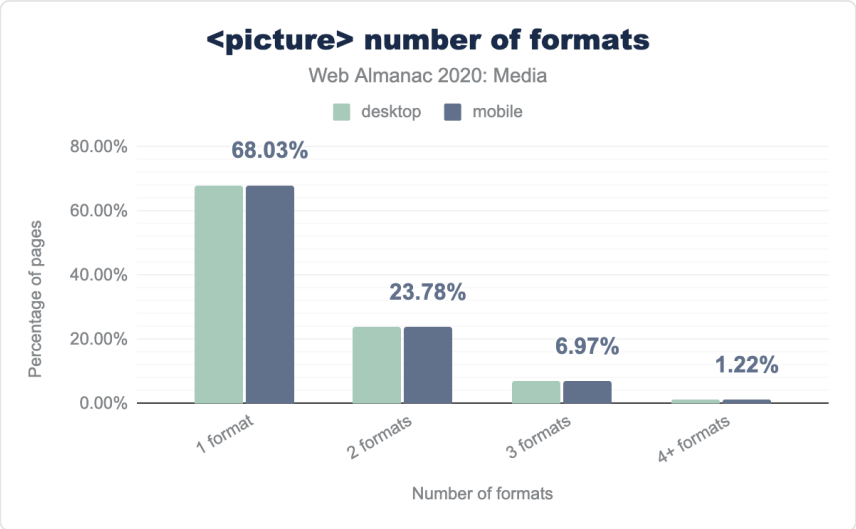


図5.5. `<picture>` フォーマットの数。

フォーマットの切り替えに`<picture>`を使用しているページのうち、約68%が、デフォルトとして機能する`<img src>`に加えて、単一のタイプのバリエーションを提供しています。

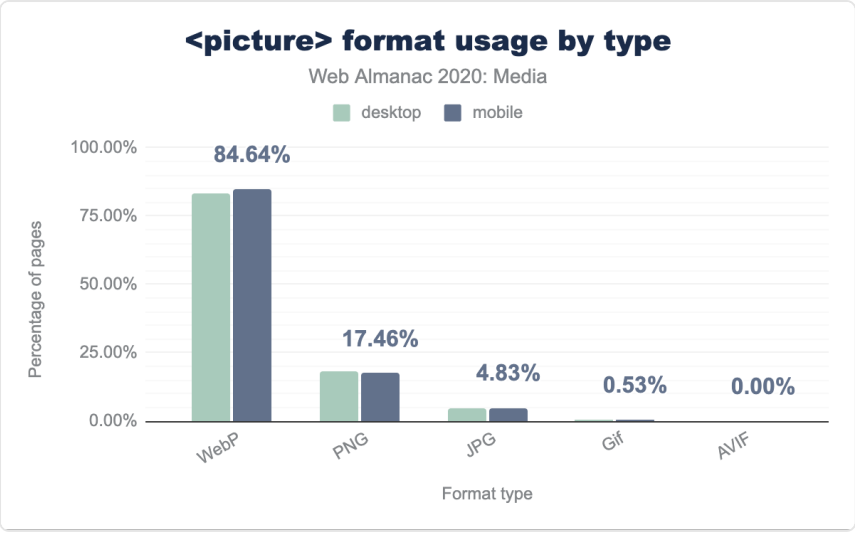


図5.6. タイプ別の写真フォーマットの使用状況

`<source>`の要素では、WebPが圧倒的に多く、次いでPNG、JPGは`<picture>`の4.83%に過ぎないことがわかります。

なお、当社のクローラーはWebPをサポートしているChromeでクロールしていますが、サポートしていない他のブラウザを使用している場合には、異なる結果が表示されます。

以下は、複数のフォーマットバリエーションを提供するために使用できるマークアップ構文の例です。

```
<picture>
  <source type="image/avif" srcset="images/example.avif">
  <source type="image/webp" srcset="images/example.webp">
  <source type="image/jp2" srcset="images/example.jp2">
  <source type="image/vnd.ms-photo"  srcset="images/example.jxr">
  <img src="images/example.jpg" alt="説明">
</picture>
```

ユーザーエージェントは、ポジティブマッチした最初のものを効果的に選択するため、ここでの順序が重要になります。

`<picture>`でフォーマットを切り替えているページのうち、83%がWebPをフォーマットのバリエーションの1つとして提供しており、ブラウザのサポートが拡大していることも関

係しています。

ブラウザ間のフォーマットサポートは、移動祝祭日のようなものです。WebPは現在、より広範にサポートされています。

- WebP: 90%のカバー率[244] (Edge, Firefox, Chrome, Opera, Android)
- JPEG 2000: 18.5%のカバー率[245] (Safari)
- JPEG XR: 1.7%のカバー率[246] (IE)
- AVIF: 25%のカバー率[247] (Chrome, Opera)

代替フォーマットのセットを構築する際、作者は圧縮性能に加えて機能も考慮しなければなりません。たとえば、画像に透明度がある場合、`img src`に指定する「最低公倍数」としてはPNGが良いでしょう。その上で、WebPやJPEG2000、AVIFなど、透明度をサポートする次世代フォーマットを含む1つ以上の`<source>`要素を使用できます。

同様に、アニメーションGIFの上に、アニメーションWebPや、ミュート、ループ、自動再生のMP4を重ねることも検討してみてください（ただし動画と画像を混在させることはマークアップのアプローチやメディア処理の必要性に影響します）。

フォーマットスイッチを導入する際には、3つの観点から検討する必要があります。

- ブラウザのフォーマットがランドスケープに対応していること
- サイトのメディアパイプライン：必要なメディアをさまざまなフォーマットで作成するためのプロセス
- どのフォーマットが提供されていて、いつ選択するかをブラウザに伝えるためのマークアップの実装

いくつかのダイナミックメディアサービスや画像CDNは、この作業を自動化し刻々と変化するブラウザフォーマットのサポート状況を追跡し、同期させるよう努力することで、この作業を大幅に簡素化できます。

注：ChromeでAVIFがサポートされたのはバージョン85（2020年8月下旬リリース）からですが、このアルマナックのデータは主にそれ以前のものです。しかし、2020年11月初旬のより最近のデータにアドホッククエリを実行すると、数万件のAVIFリクエストが表示されます。

244. https://caniuse.com/webp
245. https://caniuse.com/jpeg2000
246. https://caniuse.com/jpegxr
247. https://caniuse.com/avif

写真：メディアアートディレクション

`<picture>` 要素が提供するメディアの *アートディレクション* 機能により、これまでタイプやレイアウトをデザインする際に楽しんできた、ビューポートに依存した高度なメディア操作が可能になります。

たとえば、アスペクト比が非常に広くて短いランドスケープ指向のメディア（バナーなど）が、狭いポートレート指向のモバイルレイアウトに押し込まれた場合、どれほどうまく機能しないかを考えてみてください。メディアクエリに基づいて画像のトリミングやコンテンツを調整する機能は、十分に活用されていないと私たちは考えています。

この例では、提供されるメディアのアスペクト比を、正方形（1:1）から4:3、最終的にはビューポートの幅に応じて16:9に変更し、メディアのために利用可能なスペースを最大限に活用するように努めています。

```
<picture>
  <source media="(max-width: 780px)"
          srcset="image/example_square.jpg 1x, image/
example_square_2x.jpg 2x">
  <source media="(max-width: 1400px)"
          srcset="image/example_4_3_aspect.jpg 1x, image/
example_4_3_aspect_2x.jpg 2x">
  <source srcset="image/example_16_9_aspect.jpg 1x, image/
example_16_9_aspect_2x.jpg 2x">
  <img src="image/example_fallback.jpg" alt="...">
</picture>
```

写真：方向転換

データによると、`<picture>` を使用しているページのうち、オリエンテーションを利用しているのは1%弱に過ぎませんが、ウェブサイトのデザイナーや開発者にとっては、さらに検討すべき分野であると感じています。

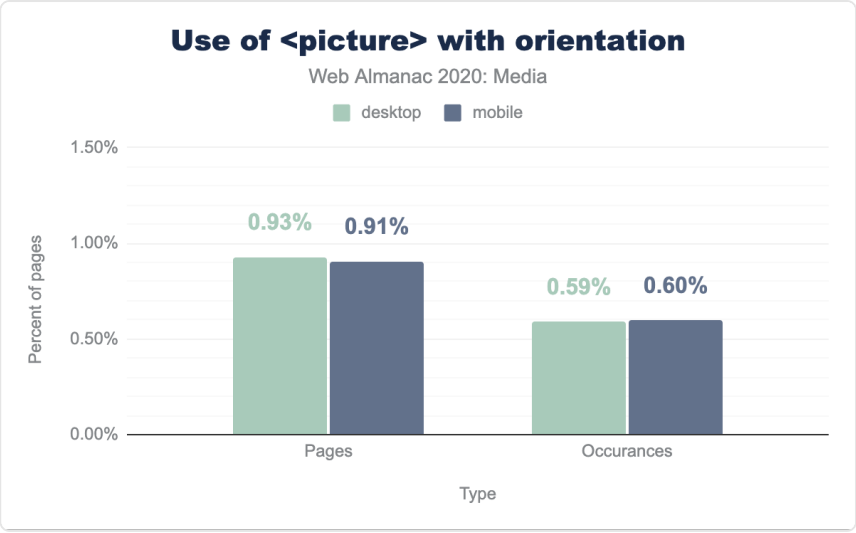


図5.7. `<picture>` オリエンテーションの使用方法。

モバイルデバイスのビューポートは小さく、狭く、手の中でポートレートモードからランドスケープモードにするのも簡単です。方向指定のメディアクエリには、あまり活用されていない興味深い可能性があります。

構文の例

```
<picture>
  <source srcset="images/example_wide.jpg"
          media="(min-width: 960px) and (orientation: landscape)">
  <source srcset="images/example_tall.jpg"
          media="(min-width: 960px) and (orientation: portrait)">
  <img src="..." alt="...">
</picture>
```

画像フォーマットの有効活用

Webページでメディアを効果的に使用するには、適切な画像フォーマットとそのフォーマットが提供する機能を使用することが重要です。

MIMEタイプとエクステンション

拡張子の分布が多く、同じ拡張子でもさまざまな綴りがあることを確認しました（例：`jpg` vs `JPG` vs `jpeg` vs `JPEG`）。また、MIMEタイプの指定が間違っている場合もあります。たとえば、`image/jpeg` は、JPEG画像のMIMEタイプとして正しく認識されています。しかし、JPEGを使用しているページの0.02%が、間違ったMIMEタイプを指定していることがわかります。さらに、`pnj` という拡張子が28,420回使用されていて（タイプミスである可能性が高い）、そのMIMEタイムが `image/jpeg` に設定されていることがわかります。

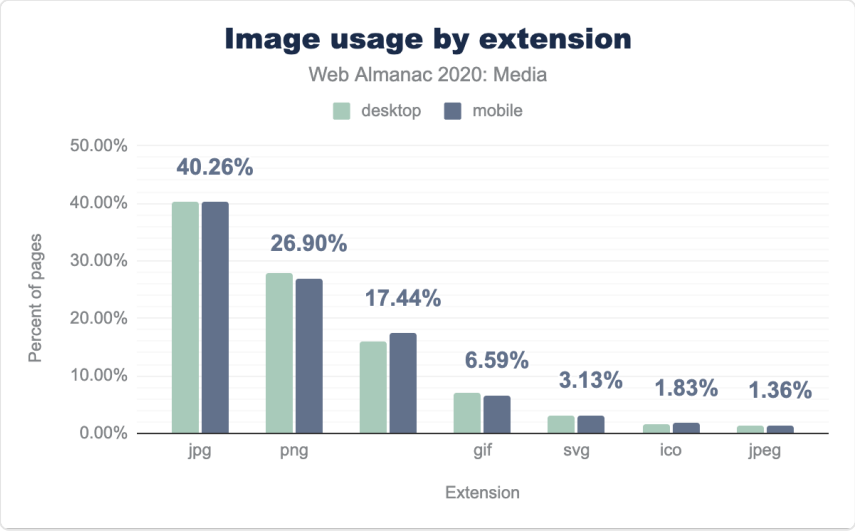


図5.8. 拡張による画像利用

たとえば、「.jpg」が「image/webp」というMIMEタイプで配信されるなど、拡張子とMIMEタイプの間にはさらに不整合が見られますが、これらのいくつかは、オンザフライでの変換や最適化機能を備えたImage CDN配信サービスに起因する自然現象であると考えられます。

プログレッシブJPEG

プログレッシブJPEG[248]はどれくらい一般的ですか？WebPageTestでは、各ページに「スコア」を付けています。これは、プログレッシブ・エンコードされたJPEGから読み込まれたすべてのJPEGバイトを合計し、プログレッシブ・エンコードされる可能性があったJPEGバイトの合計数で割ったものです。大半（57%）のページでは、JPEGバイトの25%以下しか

248. *https://www.smashingmagazine.com/2018/02/progressive-image-loading-user-perceived-performance/#back-to-basis-progressive-jpegs*

プログレッシブエンコードされていませんでした。これは、JPEGのプログレッシブ化が長年のベストプラクティスであり、MozJPEGのような最新のエンコーダーがデフォルトでプログレッシブにエンコードしているにもかかわらず、ダウンサイドのない圧縮節約のための大きなチャンスを示しています。

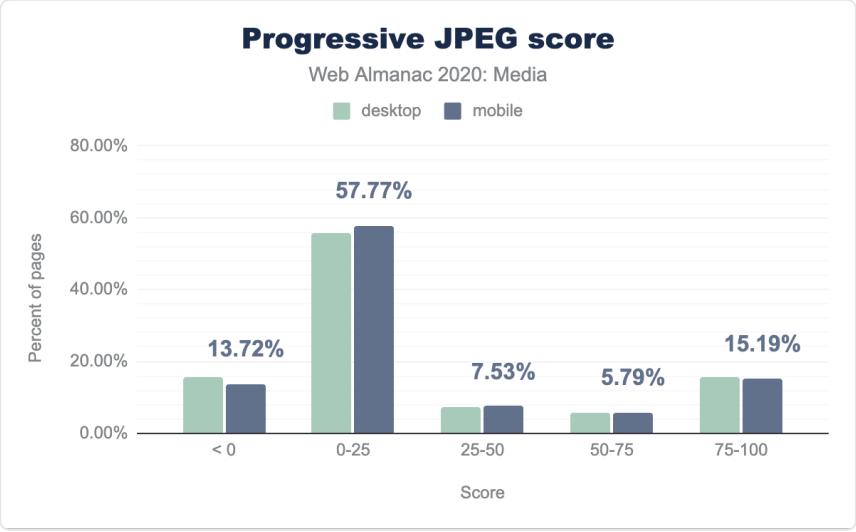


図5.9. プログレッシブJPEGスコア

マイクロブラウザ

次に、マイクロブラウザ[249]の話題に移りましょう。“link unfurlers”や”link expanders”とも呼ばれるこれらのユーザーエージェントは、ウェブページを要求し、メッセージやソーシャルメディアでリンクが共有されたときにリッチなプレビューを組み立てるために、ウェブページから断片的な情報を取得します。マイクロブラウザの*共通語*は、FacebookのOpen Graphプロトコル[250]であることから、Open Graphの`<meta>`タグにマイクロブラウザ向けの画像や動画が含まれているウェブページの割合を調べました。

249. https://24ways.org/2019/microbrowsers-are-everywhere/
250. https://ogp.me

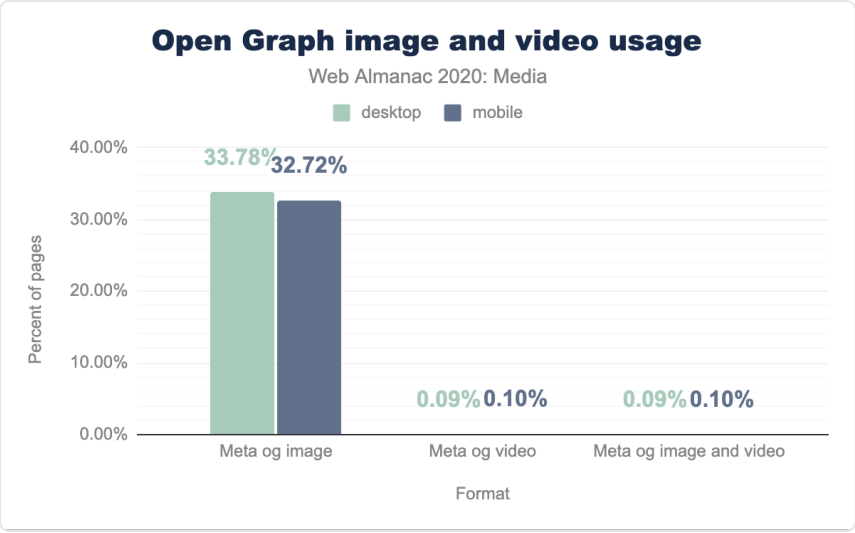


図5.10. オープングラフの画像や動画の使用状況

ウェブページの3分の1は、マイクロブラウザ用のオープングラフタグで画像を含んでいます。しかし、マイクロブラウザ専用の動画を掲載しているページは全体の0.1%程度で、動画を掲載しているページのほとんどが画像も掲載しています。

マイクロブラウザで調整されたリッチなプレビューと組み合わせた、関係性のある口コミマーケティングの力は、明らかに投資する価値があります。

動画コンテンツは制作費が高く、画像に比べてウェブ上での利用が少ないため、利用率が比較的低いことは理解できます。しかし、動画はフルブラウザでなくても、リンクプレビューの中で再生や自動再生の可能な場合が多いため、エンゲージメントを高める大きなチャンスだと考えています。

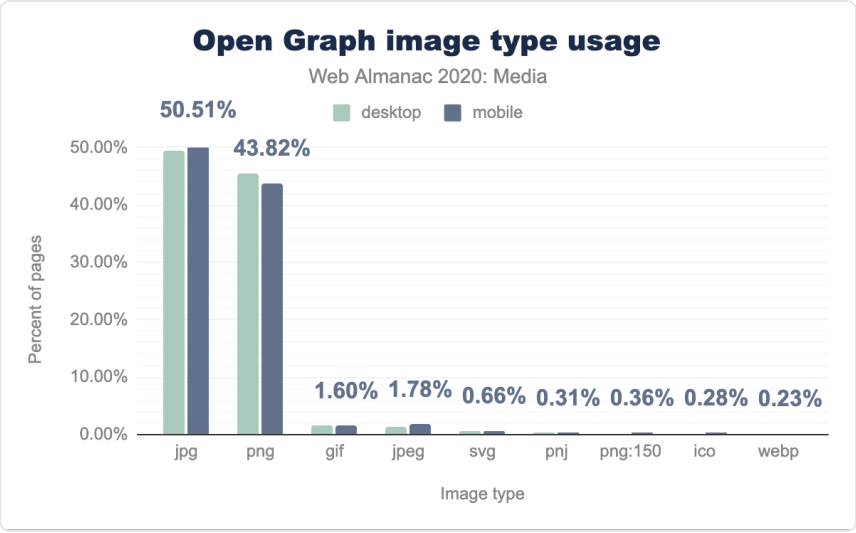


図5.11. オープングラフイメージタイプの使用

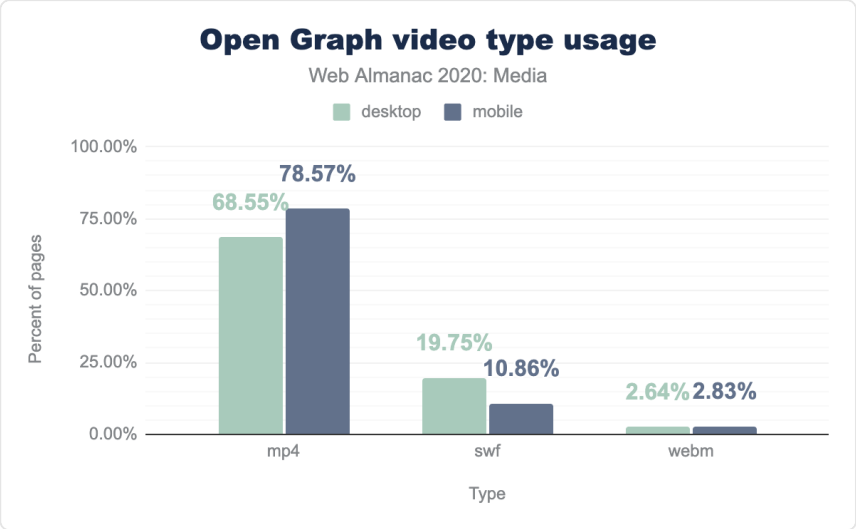


図5.12. オープングラフのビデオタイプの使用状況

Open Graphプロトコルでは、画像や動画のURLは1つしか含めることができません。`<picture>`や`srcset`で提供されるようなコンテキストに応じた柔軟性はありません。そのため、作者はマイクロブラウザに送信するフォーマットを選ぶ際に、どちらかというと保守的になりがちです。マイクロブラウザ専用画像の半分はJPEG、45%はPNG、2%弱はGIFです。WebPは、マイクロブラウザ用画像の0.2%に過ぎません。

同様に、ビデオについても、大半のリソースがもっとも一般的なフォーマットで送信されています。MP4です。2番目に人気のあるフォーマットが現在は廃止されている[251] SWFである理由は謎であり、これらがどのマイクロブラウザでも再生可能かどうかも気になるところです。

`rel=preconnect` の使い方

メディアアセットは、ローカルに保存されることもあれば、画像CDNに保存されることもあります。アセットが最適化され、変換され、エンドユーザーに配信される方法は、使用する適切な技術に大きく依存します。別のドメインからの画像を含める場合、`<link>` 要素に `rel=preconnect` 属性を使用することで、ブラウザがDNS接続を必要とする前に開始する機会を与えることができます。これは比較的安価な処理ですが、このような接続の確立にかかる追加のCPU時間が他の作業を遅らせる状況も考えられます。

8.19%

図5.13. preconnectを使ったモバイルページ

マークアップを分析すると、デスクトップでは7.83%、モバイルでは8.19%のページで採用されていることがわかります。リソースヒントの章では、DOMを分析することで少し異なる方法を使用しており、それぞれ8.15%と8.65%という似たような、しかし少し大きな数字を得ています。

`data:` のURLの使い方

データURL（以前はデータURIと呼ばれていました）を使用することは、開発者がHTMLの中にbase64エンコードされた画像を直接埋め込むことができる技術です。これにより、HTMLがDOMツリーに解析されるまでに画像が完全に読み込まれ、最初のペイントで画像を利用できることが事実上保証されます。しかし、バイナリのように無線で圧縮することができず、他の（おそらくより重要な）リソースの読み込みを妨げ、キャッシュを複雑にするため、base64の画像はアンチパターン[252]のようなものになっています。

251. https://blog.adobe.com/en/publish/2017/07/25/adobe-flash-update.html#gs.my93m2
252. https://calendar.perfplanet.com/2020/the-dangers-of-data-uris/

9.10%

図5.14. データURIを利用したモバイルページ。

画像を表示するためにデータURLを利用しているページは9%と、それほど広くは利用されていないようです。ただし、今回調査したのは、HTMLに埋め込まれたBase64エンコードされた画像の`src`のみで、CSSに埋め込まれた背景画像などのBase64エンコードされた画像は含まれていないことに注意が必要です。

SEOとアクセシビリティ

説明文を画像に関連付けることは、画像を見ることができない人やスクリーンリーダーを利用している人のアクセシビリティに役立つだけでなく、さまざまなコンピュータビジョンのアルゴリズムで画像の主題を理解するために使用されています。説明文は、ページの文脈の中で意味を持ち、説明している画像に関連するものでなければなりません。これらのトピックに関する詳しい情報は、SEOおよびアクセシビリティの章に記載されています。

`alt`テキストの使い方

画像の`alt`属性は、画像の説明を提供するために使われます。これはスクリーンリーダーによってアナウンスされ、ビジュアルブラウザでは画像がロードされないときにも表示されます。

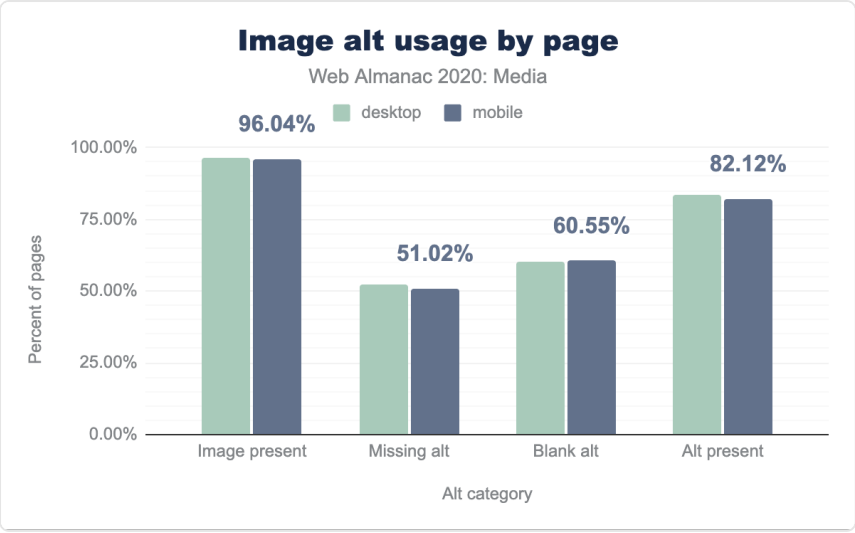


図5.15. ページごとの画像alt使用量

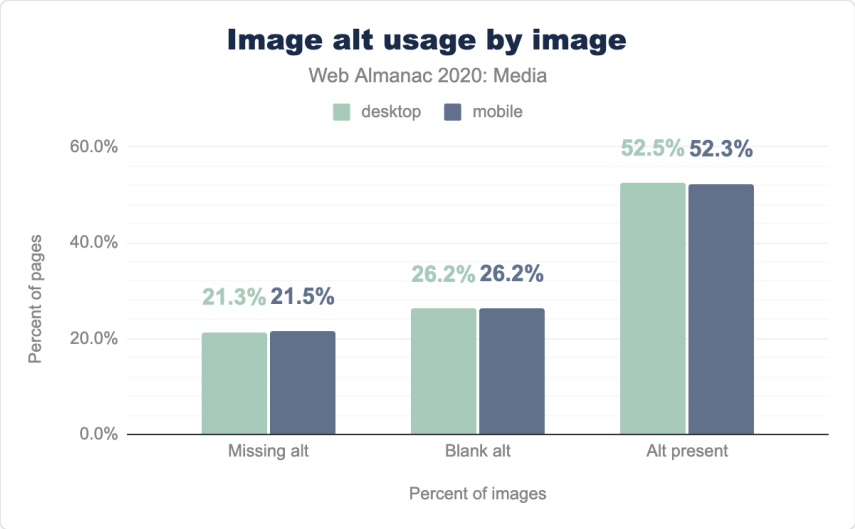


図5.16. 画像ごとのalt使用量

処理された全ページの約96%に`<img>`要素がありましたが、そのうち21%の画像には`alt`属性がありませんでした。52% の画像には`alt`属性がありましたが、そのうち26%は空白でした。簡単に言うと、ウェブ上の画像のうち、空白ではない`alt`属性を持つものは約4分の1しかなく、有用な説明文である`alt`テキストを持つものはそれよりもさらに少ないと思われます。

図と図のキャプション

HTML5では、言語にさまざまな新しいセマンティック要素が追加されました。そのような要素の1つが `<figure>` で、これはオプションで子として `<figcaption>` 要素を含むことができます。 `<figcaption>` に含まれるテキストの説明は、 `<figure>` の他のコンテンツと意味的にグループ化されます。

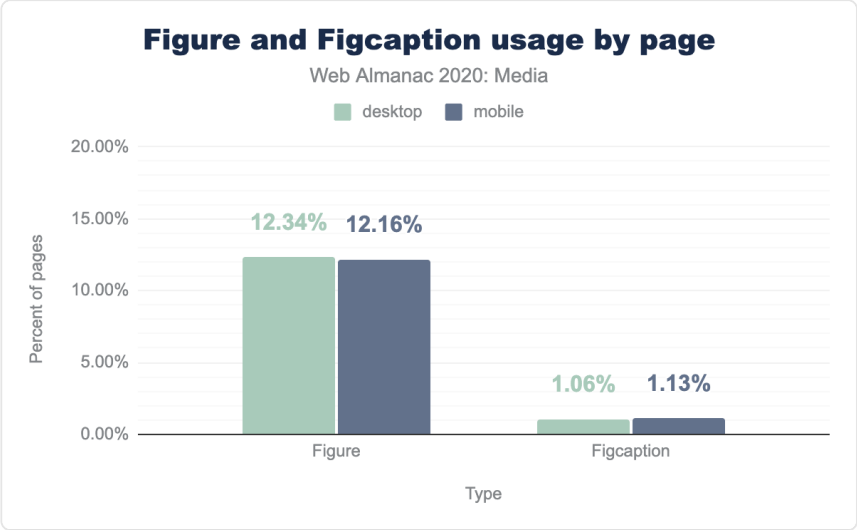


図5.17. ページごとの図と図キャプションの使用状況

デスクトップとモバイルでも、約12%のページで `<figure>` 要素が使用されていますが、約1%のページで `<figcaption>` を使用して説明文を追加していることがわかります。

動画

「百聞は一見にしかず」ならば、30fpsの動画1分は180万円の価値があるに違いない。

動画は、今日、視聴者と関わりを持つもっとも強力な方法のひとつです。しかし、サイトに動画を追加することは、簡単なことではありません。迷路のようなフォーマットやコーデックをナビゲートし、無数の実装方法を検討しなければなりません。しかし、ビデオのインパクトは、視覚的なインパクトとパフォーマンスへのインパクトの両方において、過大評価されることはありません。

`<video>` 要素

`<video>` 要素は、ウェブ上での動画配信の中核をなすもので、単独で使用されるほか、動画配信のために段階的に強化されていくJavaScriptプレイヤーと組み合わせて使用されます。

ソースの有無、総使用量

`<video>` 要素を使ってビデオリソースを埋め込むには2つの方法があります。1つのリソースURLを要素自体の `src` 属性に指定するか、または任意の数の子 `<source>` 要素を指定し、ブラウザは読み込めると思われるソースを見つけるまでそれを参照します。最初のクエリでは、サンプルページ全体でこれらのパターンがどのくらいの頻度で見られるかを調べます。

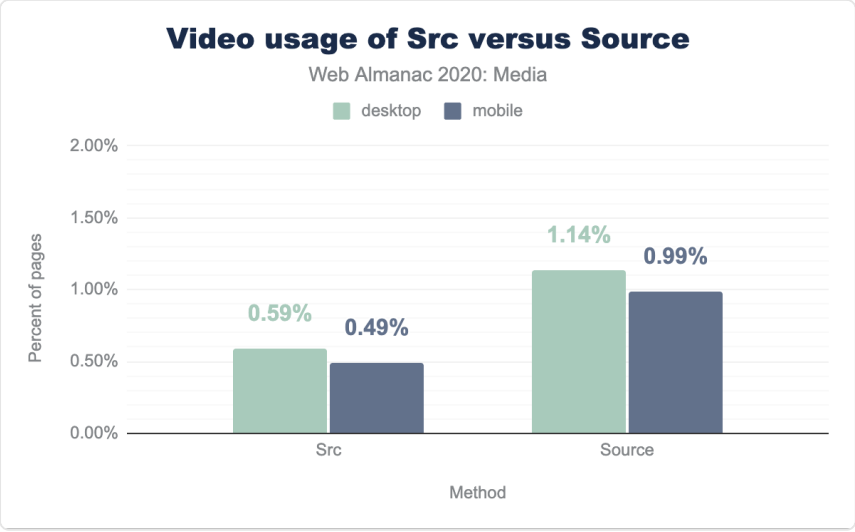


図5.18. Src対Sourceのvideoの使い方。

`<video>` の中には、 `<src>` 属性ではなく `<source>` という子を持つものが2倍あります。これは作者が、最低公約数的な単一のリソースをすべての人に送るのではなく、文脈に応じて異なるリソースを異なるエンドユーザーに送る機能を望んでいることを示しています（あるいは一部の視聴者により悪い、あるいは壊れた体験を与えることになります）。

また、興味深いことに、すべてのページにおいて、 `<video>` 要素がまったく含まれていないのは、わずか1パーセントか2パーセントであることがわかります。これは、 `<img>` よりもはるかに少ないのです。

JavaScriptプレーヤー

いくつかの一般的なプレーヤー（hls.js、video.js、Shaka Player、JW Player、Brightcove Player、Flowplayer）の存在を調べました。これらの特定のプレーヤーを使用しているページは、ネイティブの`<video>`要素を使用しているページの半分以下しかありませんでした。

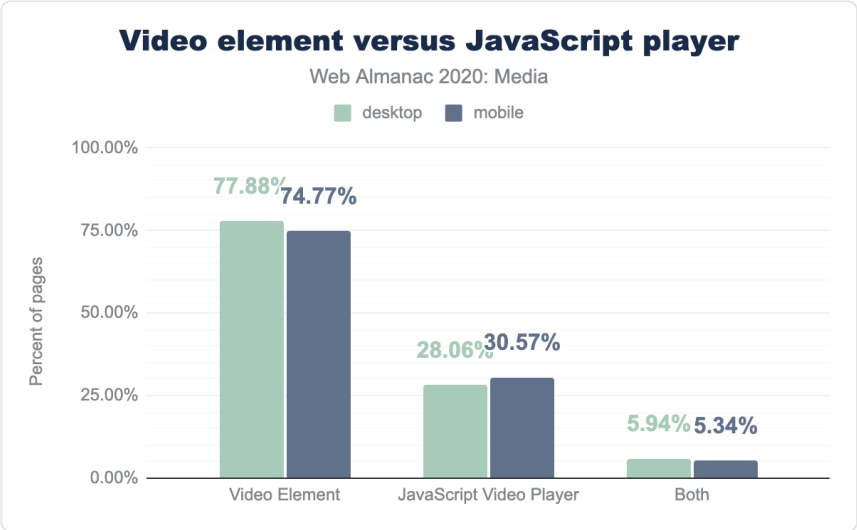


図5.19. Video要素とJavaScriptプレーヤーの比較。

video.jsなどの多くのプレイヤーが、ソース内の`<video>`要素を強化しているため、分析は少し複雑になります。検索されたプレーヤーを使用しているページのうち、`<video>`要素も含まれていたのは、わずか5〜6%でした。しかし、このパターンの証拠は、`<video>`と`<source>`要素内の`type`属性の値を見ると、実際にはよりはっきりとわかります。

タイプ別属性

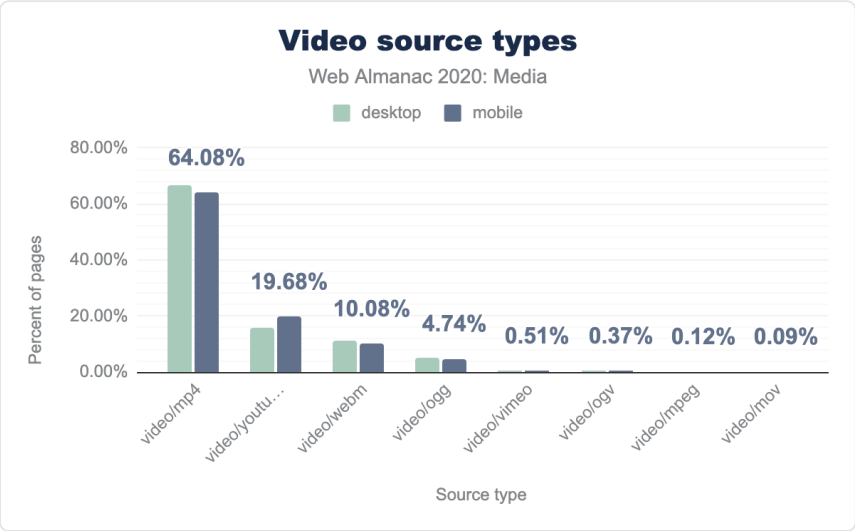


図5.20. ビデオソースの種類

当然のことながら、もっとも一般的なタイプの値は `video/mp4` です。しかし、次に多いのは、デスクトップ用の `タイプ` の15%、モバイル用クローラーに送信される `タイプ` の20%を占める `video/youtube` で、これは登録されたMIMEタイプではありません。これは、登録されたMIMEタイプではなく、（WordPressを含む）いくつかのプレイヤーがYouTubeの動画を埋め込む際に使用する特別な値です。さらに、Vimeoの埋め込みにも同様のパターンが見られます。

MP4とWebMは、一般的に使用されていると思われる唯一の2つのフォーマットです。これらのコンテナの中でどのコーデックが使われているのか、またVP8、HVEC、AV1などの次世代コーデックがどれだけ普及しているのかを知ることは興味深いことです。しかし、そのような分析は、残念ながらこの記事の範囲外である。

動画のプリロード

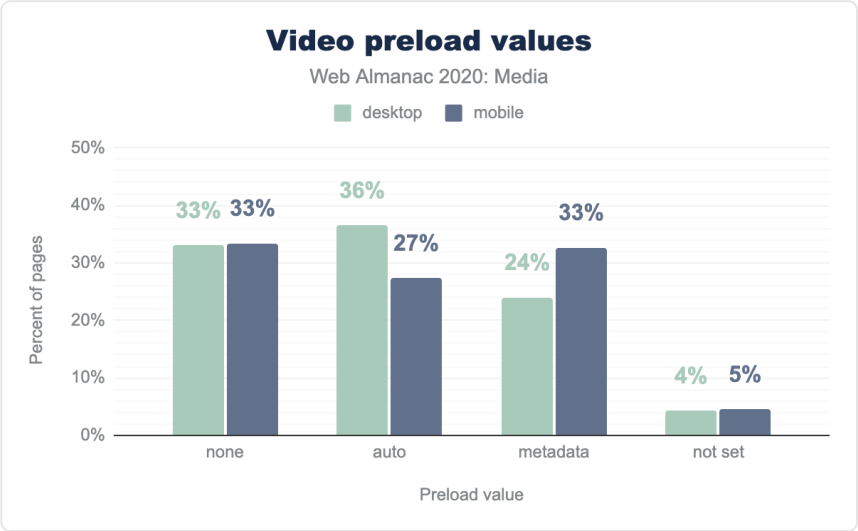


図5.21. 動画のプリロード値。

`preload`属性は、動画をダウンロードするかどうかを示すもので、3つの値を持つことができます。`none`, `metadata`, `auto`の3つの値を持つことができます（空欄のままだと`auto`の値が仮定されることに注意してください）。4.81% のページが`<video>`要素を持ち、そのうち45.37%が`preload`属性を持っていることがわかります。モバイルでの数字は若干異なり、`<video>`要素を持つページは3.59%のみで、そのうち43.38%に`preload`属性があります。

オートプレイとミュート

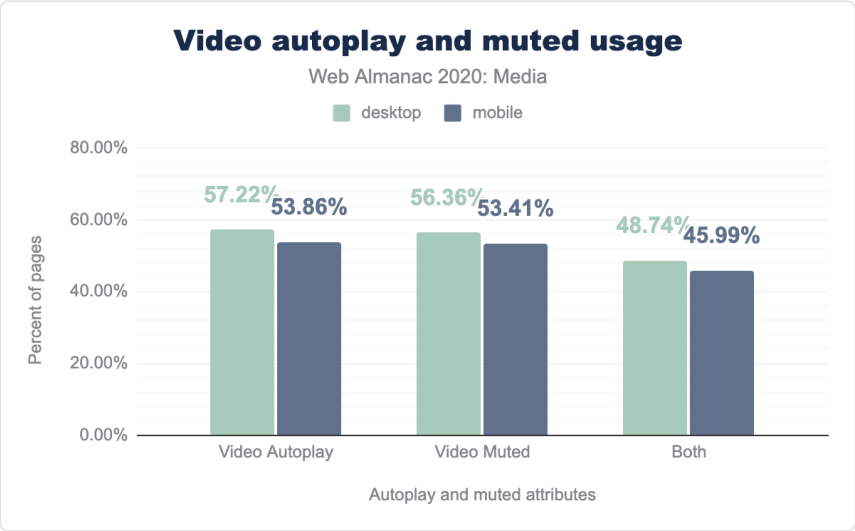


図5.22. 動画の自動再生とミュートの使用

ビデオに関する追加情報を見るとデスクトップでは、57.22% の `<video>` 要素が `autoplay` 属性を持っており、デスクトップで `<video>` 要素を持つページの56.36%が `muted` 属性を利用しており、最後に48.74%のページが `autoplay` と `muted` の両方を併用しています。モバイルの場合も同様で、53.86% が `autoplay`、53.41%が `muted`、45.99% が両方の属性を使用しています。

結論

ウェブは、視覚的なストーリーを伝えるのに最適な場所です。今回の調査では、ウェブには実に多くのメディアの要素が活用されていることがわかりました。この多様性は、今日のウェブでメディアが表現される方法の数にも表れています。メディアを表示するためのほとんどの基本的な機能は積極的に使用されていますが、最新のブラウザを使用すれば、さらに多くのことができるようになります。現在使用されている高度なメディア機能の中には素晴らしいものもありますが、時には間違った使い方や、誤った文脈で使用されることもあります。私たちは皆さんに、さらに一歩踏み込んで、ユーザーにもっと素晴らしいビジュアル体験を提供するために、現代のウェブのすべての機能と性能を利用することをオススメしたいと思います。

著者

Tamas Piros

@tpiros　tpiros　https://tamas.io

Tamas Pirosは、Cloudinary[253]のDeveloper Experience Engineerであり、Google Developer Expertであり、Full Stack Training[254]を運営するテクニカルインストラクターです。

Ben Seymour

@bseymour　bseymour　benseymour　https://benseymour.com

Ben Seymourは、Cloudinary[255]のDynamic Media & Content Specialistであり、Practical Responsive Images[256]の著者であり、Storyus[257]とHaktive[258]の共同設立者です。

Eric Portis

@etportis　@ericportis.com　eeeps　https://ericportis.com

Eric Portisは、Cloudinary[259]のWeb Platform Advocateです。

253. *https://cloudinary.com/*
254. *https://fullstacktraining.com*
255. *https://cloudinary.com/*
256. *http://responsiveimag.es/*
257. *https://storyus.life/*
258. *https://haktive.com/*
259. *https://cloudinary.com/*

部 I 章 6
サードパーティ

***Simon Hearne** によって書かれた。*
Julia Yang と Shane Exterkamp によってレビュ。
Max Ostapenko と Paul Calvano による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

サードパーティのコンテンツは、今日ではほとんどのウェブサイトの重要なコンポーネントとなっています。アナリティクス、ライブ チャット、広告、動画共有など、あらゆるものに影響を与えています。サードパーティのコンテンツは、サイト所有者の負担を軽減し、コアコンピタンスに集中できるようにすることで価値を提供します。

多くの人は、サードパーティのコンテンツはJavaScriptベースのものだと考えていますが、データによると、これはリクエストの22%にすぎません。サードパーティのコンテンツは、画像(37%)から音声(0.1%)まで、あらゆる形で提供されています。

この章では、サードパーティコンテンツの普及率と2019年以降の変化をレビューします。また、サードパーティコンテンツがページの重み（全体的なパフォーマンスへの影響を表す良いプロキシ）に与える影響、ページのライフサイクルの早い段階でロードされるスクリプト、サードパーティコンテンツがブラウザのCPU時間に与える影響、サードパーティがパフ

ォーマンスデータをどのように公開しているかについてもレビューします。

定義

データに飛び込む前に、この章で使用する用語を定義する必要があります。

“サードパーティ”

サードパーティリソースとは、主なサイトユーザーの関係の外にある存在のことです。これは、サイト所有者の管理下にはないが、サイト所有者の承認を得て存在するサイトの側面を含みます。例えば、Google Analyticsスクリプトは一般的なサードパーティリソースです。

私たちは、サードパーティのリソースをこれらとみなしています。

- *共有* および *パブリック* オリジンでホストされています
- 様々なサイトで広く利用されている
- 個人のサイトオーナーの影響を受けない

これらの目標を可能な限り正確に一致させるために、この章ではサードパーティリソースの正式な定義を以下のようにしています：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にリソースを見つけることができるドメインに由来するリソース。

これらの定義を使用して、ファーストパーティドメインから提供されたサードパーティコンテンツはファーストパーティコンテンツとしてカウントされることに注意してください。例えば、セルフホスティングのGoogle Fontsやbootstrap.cssは *ファーストパーティコンテンツ* 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ツもサードパーティのコンテンツとしてカウントされます。関連する例。サードパーティのドメインでCDNを介して提供されるファーストパーティの画像は、 *サードパーティのコンテンツ* とみなされます。

プロバイダカテゴリ

この章では、サードパーティプロバイダーをさまざまなカテゴリに分けて説明します。それぞれのカテゴリには、簡単な説明が含まれています。ドメインとカテゴリのマッピングは、サードパーティのウェブリポジトリ[260]にあります。

260. *https://github.com/patrickhulce/third-party-web/blob/8afa2d8cadddec8f0db39e7d715c07e85fb0f8ec/data/entities.json5*

- 広告 - 広告の表示と測定
- アナリティクス - サイト訪問者の行動を追跡
- CDN - 公共の共有ユーティリティやユーザーのプライベートコンテンツをホストするプロバイダー
- コンテンツ - パブリッシャーを促進し、シンジケートされたコンテンツをホストするプロバイダー
- カスタマーサクセス - サポートと顧客関係管理機能
- ホスティング - ユーザーの任意のコンテンツをホストするプロバイダー
- マーケティング - 営業、リードジェネレーション、メールマーケティング機能
- ソーシャル - ソーシャルネットワークとその連携
- タグマネージャー - 他のサードパーティの包含を管理することが唯一の役割であるプロバイダー
- ユーティリティー - サイトオーナーの開発目的を支援するコード
- ビデオ - ユーザーの任意のビデオコンテンツをホストするプロバイダー
- その他 - 未分類または不適合な活動

CDNに関する注意事項。ここでのCDNカテゴリには、パブリックCDNドメイン（例：bootstrapcdn.com、cdnjs.cloudflare.comなど）上でリソースを提供するプロバイダが含まれており、単にCDN上で提供されるリソースは含まれていません。

警告事項

- ここで提示されているデータはすべて、非インタラクティブなコールドロードに基づいています。これらの値は、ユーザーとのインタラクションの後では、かなり異なって見えるようになる可能性があります。
- このページは、Cookieが設定されていない米国のサーバーで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ていません。これは特に、一般データ保護規則[261]やその他の類似の法律の適用範囲内にある国でホストされ、主に提供されているページに影響を与えます。
- テスト対象はホームページのみです。それ以外のページは、サードパーティの要

261. 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

件が異なる場合があります。

- リクエスト量別のサードパーティドメインの約84%が特定され、分類されています。残りの16%は「その他」のカテゴリーに分類されています。

方法論についてはこちらをご覧ください。

普及率

この分析の良い出発点は、サードパーティコンテンツが今日のほとんどのウェブサイトの重要な構成要素であるという声明を確認することです。サードパーティコンテンツを利用しているウェブサイトはどれくらいあるのでしょうか？

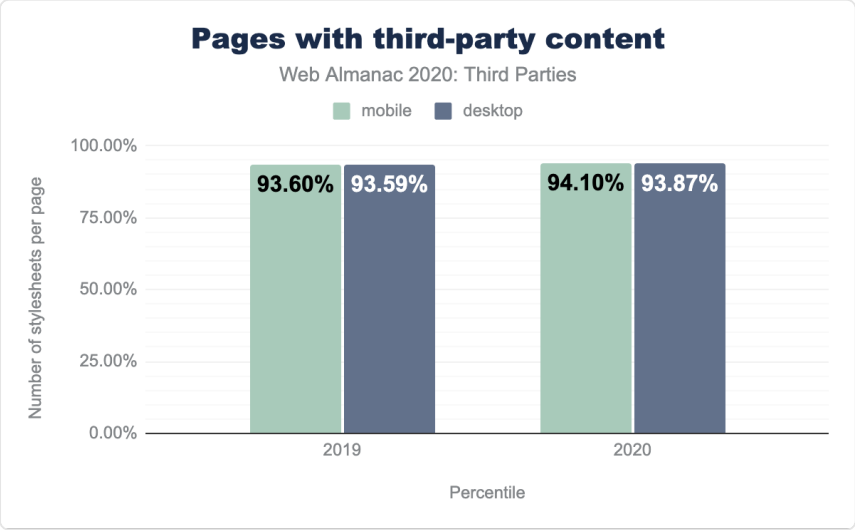


図6.1. サードパーティのコンテンツがあるページ

これらの普及率の数字は、2019年の結果[262]: デスクトップクロールのページの93.87%が少なくとも1つのサードパーティのリクエストを持っていたが、モバイルクロールのページの94.10%とわずかに高くなっていた。サードパーティのコンテンツがないページの数が少ないことを簡単に調べてみると多くはアダルトサイトであり、いくつかは政府のドメインであり、いくつかはコンテンツが少ない基本的なランディング/ホールディングページであることがわかりました。大多数のページには、少なくとも1つのサードパーティ製コンテンツが含まれていることがわかります。

262. *https://almanac.httparchive.org/ja/2019/third-parties*

下のグラフは、サードパーティ数別のページの分布を示しています。10%台のページでは2件のサードパーティからのリクエストがあるのに対し、中央値のページでは24件となっています。10%以上のページでは、100件以上のサードパーティからのリクエストがあります。

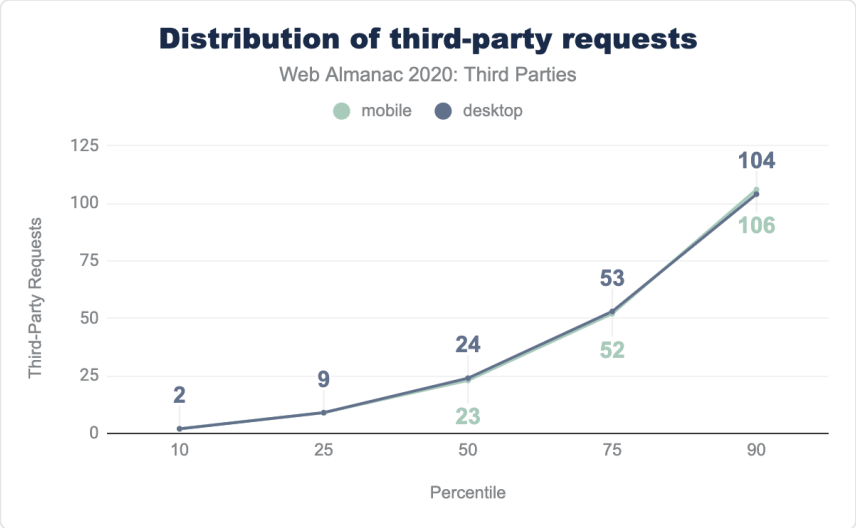


図6.2. サードパーティリクエストの配信。

コンテンツタイプ

サードパーティのリクエストをコンテンツタイプ別に分解できます。これは、サードパーティドメインから配信されたリソースのcontent-type[263]が報告されているものです。

263. https://developer.mozilla.org/docs/Web/HTTP/Headers/Content-Type

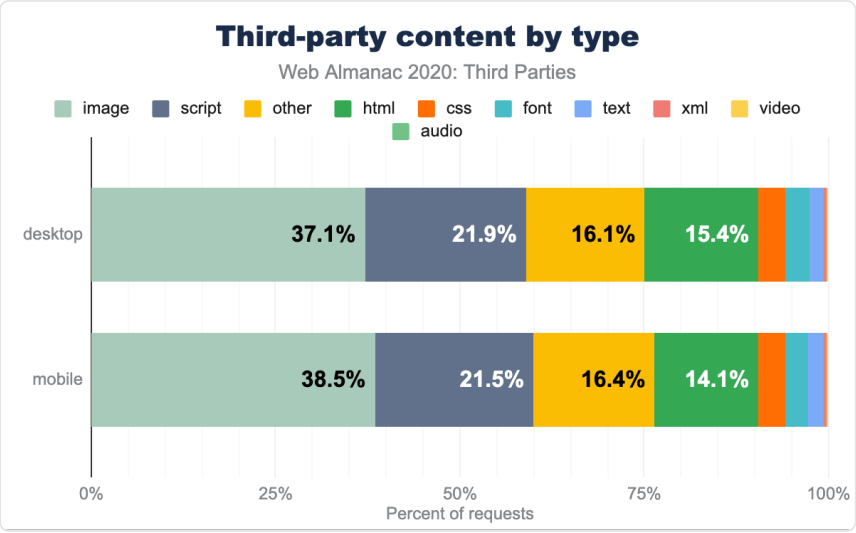


図6.3. タイプ別サードパーティコンテンツ

その結果、サードパーティコンテンツの主な貢献者は画像（38%）とJavaScript（22%）で、次に多いのは不明（16%）であることがわかりました。不明は、テキスト/プレーンなどの非カテゴライズグループのサブセットと、コンテンツタイプのヘッダーがないレスポンスです。

これはシフトを示しています2019年と比較して[264]：相対的な画像コンテンツは33%から38%に増加している一方で、JavaScriptコンテンツは32%から22%に大幅に減少しています。この減少は、cookieとデータ保護規制への順守が増え、HTTPアーカイブのテスト実行の範囲外である明示的なユーザーのオプトインの後までサードパーティの実行を減らしたことによるものと思われます。

サードパーティドメイン

サードパーティのコンテンツを提供しているドメインをさらに掘り下げてみると、Google Fontsが圧倒的に多いことがわかります。テストしたモバイルページの7.5%以上に存在してます。フォントがサードパーティコンテンツの3%程度しか占めていないのに対し、これらのコンテンツのほとんどはGoogle Fontsサービスによって配信されています。あなたのページでGoogle Fontsを使用している場合は、可能な限り最高のユーザーエクスペリエンスを確保するために、ベストプラクティス[265]に必ず従ってください。

264. https://almanac.httparchive.org/ja/2019/third-parties#%E3%83%AA%E3%82%BD%E3%83%BC%E3%82%B9%E3%81%AE%E7%A8%AE%E9%A1%9E
265. https://csswizardry.com/2020/05/the-fastest-google-fonts/

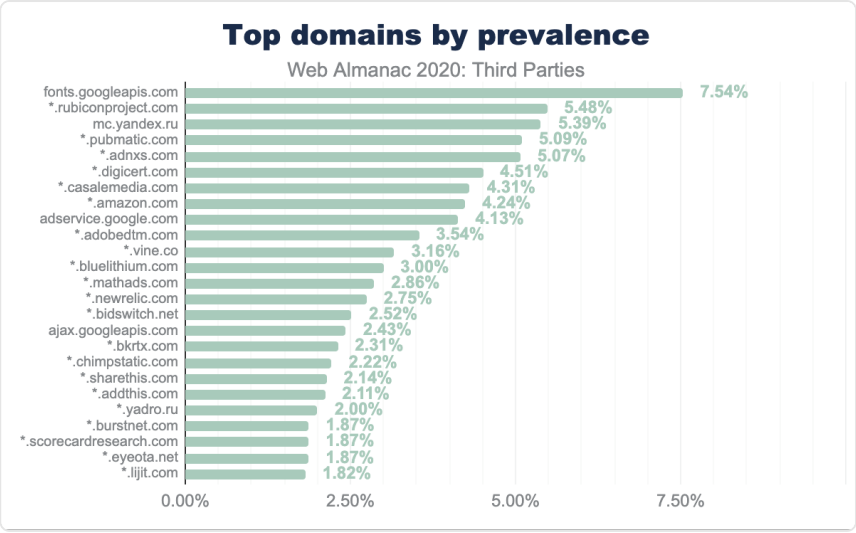


図6.4. 普及率別トップドメイン。

次の4つの最も一般的なドメインはすべて広告プロバイダです。それらはページから直接要求されるのではなく、別の広告ネットワークによって開始されたリダイレクトの複雑なチェーンを経由している可能性があります。

6番目に多いドメインは`digicert.com`です。`digicert.com`への呼び出しは、TLS証明書がOCSPのステープリングを有効にしていなかったり、中間証明書のピン留めを防ぐExtended Validation(EV)証明書を使用していたりするため、一般的にOCSPの失効チェックが行われます。この数は、すべてのページロードが事実上の初回訪問者であるため、HTTP Archiveでは誇張されています。OCSPレスポンスは一般的にキャッシュされ、実際のブラウジングでは7日間有効です。この問題についての詳細はこのブログ記事[266]を参照してください。

2.43%でさらに下位にランクインしているのは、GoogleのHosted Libraries project[267]の`ajax.googleapis.com`です。ホストされたサービスからjQueryなどのライブラリをロードするのは簡単ですが、サードパーティのドメインに接続するための追加コストがパフォーマンスにマイナスの影響を与える可能性があります。可能であれば、重要なJavaScriptとCSSはすべてルートドメインでホストするのがベストです。また、現在ではすべての主要ブラウザがページごとにキャッシュを分割[268]しているため、共有CDNリソースを使用することによるキャッシュのメリットはありません。ハリー・ロバーツ氏はHow to host your own static assets[269]で詳細なブログ記事を書いています。

266. *https://simonhearne.com/2020/drop-ev-certs/*
267. *https://developers.google.com/speed/libraries*
268. *https://developers.google.com/web/updates/2020/10/http-cache-partitioning*
269. *https://csswizardry.com/2019/05/self-host-your-static-assets/*

ページ重量の影響

サードパーティは、ブラウザによってダウンロードされたバイト数として測定されるページの重さに大きな影響を与える可能性があります。ページの重さの章で詳しく説明していますが、ここではページの重さに最も大きな影響を与えるサードパーティに焦点を当てています。

最も重いサードパーティ

最大のサードパーティを、ページ重量の影響度の中央値、つまり、それらのサードパーティがページにもたらすバイト数によって抽出できます。この結果は、サードパーティがどれだけ人気があるかを考慮しておらず、バイト数でのインパクトだけを考慮しているため、興味深いものです。

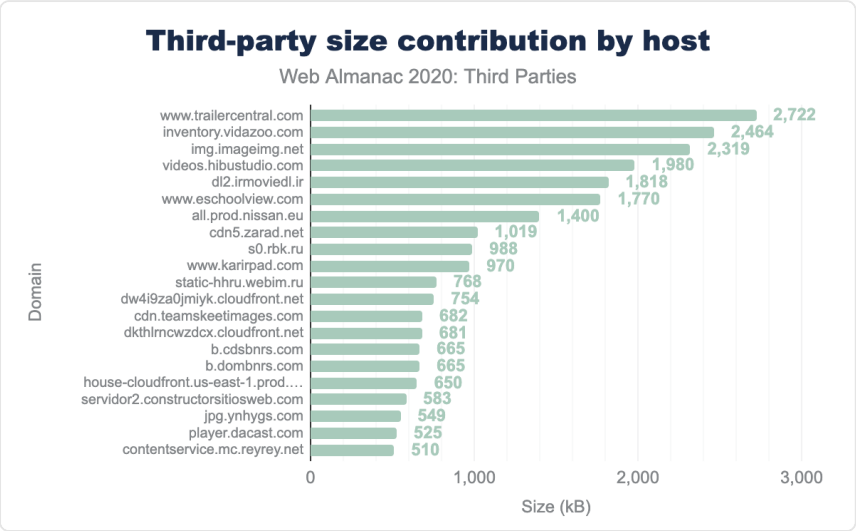


図6.5. ホストによるサードパーティサイズの貢献度。

ページ重量のトップの貢献者は、一般的に画像や動画のホスティングなどのメディア コンテンツプロバイダーです。例えば、Vidazooの場合、ページ重量の中央値は約2.5MBです。`inventory.vidazoo.com`ドメインはビデオホスティングを提供しているため、このサードパーティ製のページの中央値は2.5MBの余分なメディアコンテンツがあります。

この影響を軽減する簡単な方法は、ユーザーがページと対話するまで動画の読み込みを延期することで、動画を消費しない訪問者の影響を軽減されます。

この分析をさらに発展させて、サードパーティのカテゴリの有無による総ページサイズ（す

べてのリソースについてダウンロードされたバイト数）の分布を作成できます。このチャートでは、ほとんどのサードパーティのカテゴリの存在が総ページサイズに顕著な影響を与えていないことがわかります。これはプロットの乖離として見えるでしょう。これに対する注目すべき例外は広告（黒で表示）で、これはページサイズとの関係が非常に小さいことを示しており、広告リクエストがページに大きな重みを加えていないことを示しています。これは、これらのリクエストの多くが小さなリダイレクトであり、中央値はわずか420バイトであるためと考えられます。タグマネージャーやアナリティクスについても同様に影響が少ないことがわかります。

スペクトルのもう一方の端では、CDN、コンテンツ、ホスティングのカテゴリはすべてページの総重量と強い関係を表しています。これは、ホスティングサービスを使用しているサイトの方が、一般的にページ重量が大きいことを示しています。

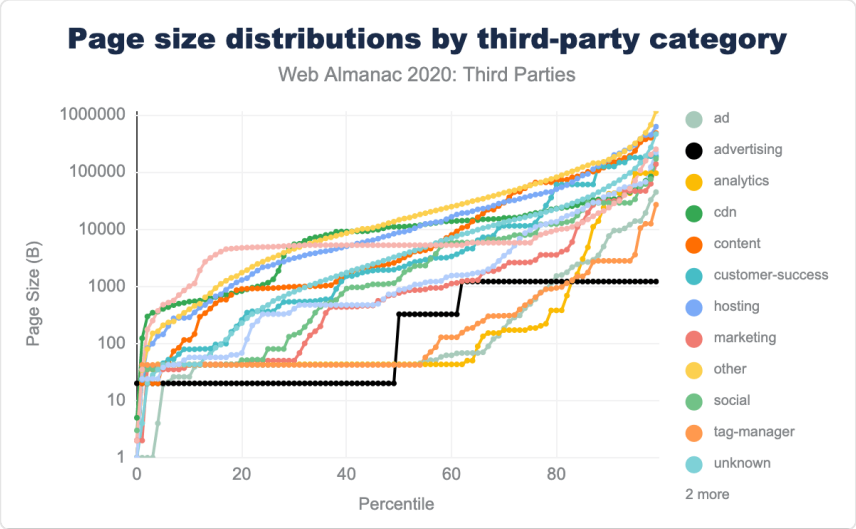


図6.6. サードパーティカテゴリ別のページサイズ分布。

キャッシュ可能性

サードパーティのレスポンスの中には、常にキャッシュされているものがあります。サードパーティが提供する画像や動画などのメディアやJavaScriptライブラリなどが良い候補となります。一方、トラッキングピクセルやアナリティクスビーコンは決してキャッシュすべきではありません。結果は、全体的にサードパーティのリクエストの3分の2が`cache-control`のような有効なキャッシュヘッダーで提供されていることを示しています。

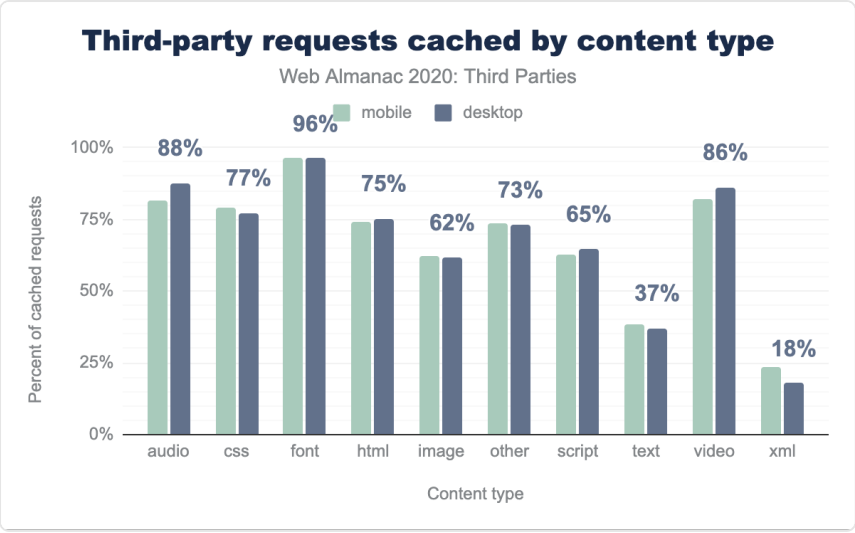


図6.7. コンテンツの種類によってキャッシュされるサードパーティのリクエスト。

レスポンスタイプ別に分解すると、次のような仮定が確認できます：xmlとテキストレスポンス（一般的にトラッキングピクセル/アナリティクスビーコンによって配信される）は、キャッシュ可能である可能性が低くなります。驚くべきことに、サードパーティによって提供される画像の3分の2以下がキャッシュ可能です。さらに調べてみると、これはキャッシュ不可能なゼロサイズGIF画像レスポンスとして返されるトラッキング「ピクセル」を使用していることが原因です。

大きなリダイレクト

多くのサードパーティは、HTTPステータスコード3XXなどのリダイレクトレスポンスを返します。これらは、バニティドメインを使用したり、リクエストヘッダーを通してドメイン間で情報を共有したりするために発生します。これは特に広告ネットワークに当てはまります。大きなリダイレクトレスポンスは設定ミスを示すもので、有効な`Location`レスポンスヘッダーにオーバーヘッドを加えた場合、レスポンスは約340Byteになるはずです。下の表は、HTTP Archiveのすべてのサードパーティ製リダイレクトのボディサイズの分布を示しています。

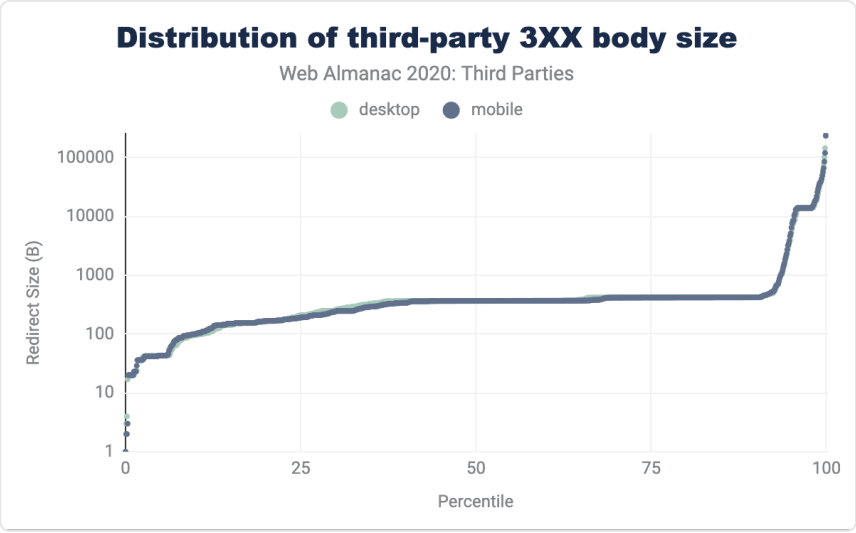


図6.8. サードパーティ3XXボディサイズの分布

結果は、3XXのレスポンスの大部分が小さいことを示しています：90パーセンタイルは420Byte、つまり3XXのレスポンスの90%は420Byte以下です。95パーセンタイルの割合は6.5KByte、99%は36KByte、99.9%は100KByteを超えています！　リダイレクトは無害に見えるかもしれませんが、単に別のレスポンスにつながるだけのレスポンスに対して、100KByteは不合理な量のバイト数です。

早期のローダー

ページの読み込みが遅いスクリプトは、総ページ読み込み時間とページの重さに影響を与えますが、ユーザー体験に影響を与えない可能性があります。しかしページの早い段階でロードされるスクリプトは、重要なファーストパーティリソースの帯域幅を共食いする可能性があり、ページのロードを妨害する可能性が高くなります。これは、パフォーマンスメトリクスやユーザー体験に悪影響を及ぼす可能性があります。

下のグラフは、デバイスの種類とサードパーティのカテゴリ別に、早くロードされるリクエストの割合を示しています。目立つ3つのカテゴリーはCDN、ホスティング、タグマネージャーで、これらはすべてドキュメントの先頭でリクエストされたJavaScriptを配信する傾向があります。広告リソースは、広告ネットワークのリクエストが一般的にページのロード後に実行される非同期スクリプトであるため、ページの早い段階でロードされる可能性が最も低くなっています。

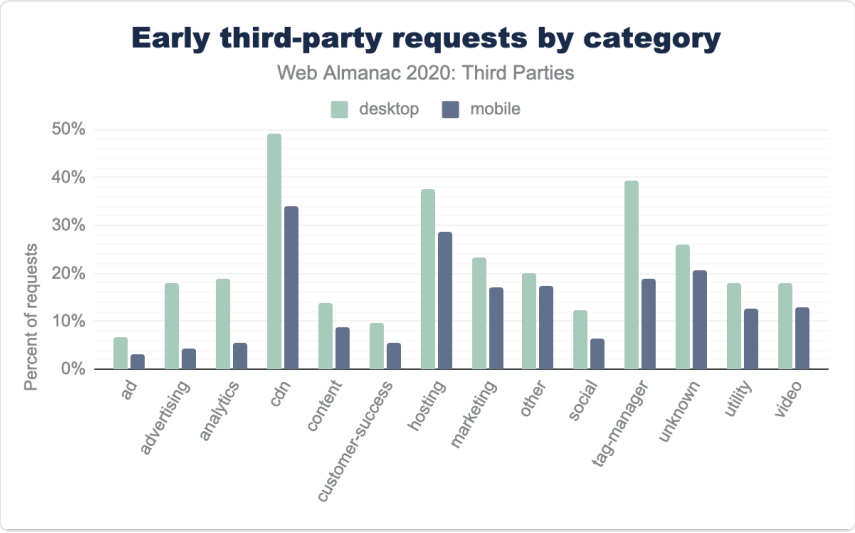


図6.9. カテゴリー別に見た初期のサードパーティのリクエスト

CPUの影響

ウェブ上のすべてのバイトが等しいわけではありません: 500KByteの画像は、500KByteの圧縮されたJavaScriptバンドルよりも、ブラウザで処理する方がはるかに簡単かもしれません。クライアント側のコードが1.8MByteに膨らみます！　サードパーティ製スクリプトのCPU時間への影響は、ネットワーク上の追加バイト数や時間よりもはるかに重要になります。

サードパーティのカテゴリの存在とページ上の総CPU時間を相関させることで、各サードパーティのカテゴリがCPU時間に与える影響を推定できます。

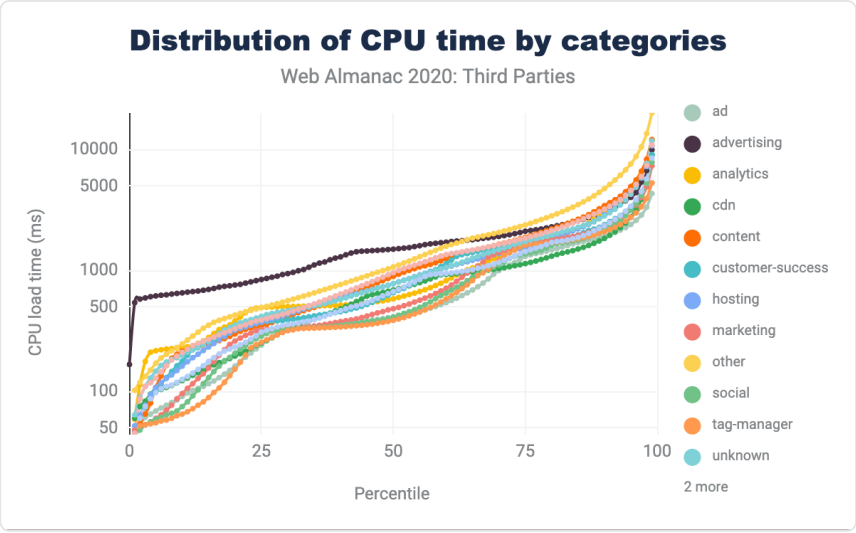


図6.10. カテゴリー別のCPU時間の分布。

このグラフは、各ページに存在するサードパーティのカテゴリ別の総ページCPU時間の確率密度関数を示しています。ページの中央値はパーセンタイル軸で50です。データによると、すべてのサードパーティのカテゴリは同様のパターンで、中央値のページのCPU時間は400〜1,000ミリ秒となっています。ここでの外れ値は広告（黒）です：広告タグが付いているページは、ページロード時のCPU使用率が高い可能性が高くなります。広告タグがあるページのCPU負荷時間の中央値は1,500msであるのに対し、広告タグがないページのCPU負荷時間は500msです。低いパーセンタイルでの高いCPU負荷時間は、最速のサイトでも広告として分類されるサードパーティの存在によって大きな影響を受けていることを示しています。

`timing-allow-origin`の普及率

Resource Timing API[270]は、ウェブサイトのオーナーがJavaScriptを介して個々のリソースのパフォーマンスを測定することを可能にします。このデータは、サードパーティコンテンツのようなクロスオリジンのリソースに対しては、デフォルトでは非常に制限されています。例えば、ウェブサイトのオーナーは、ウィジェットのリクエストのレスポンスサイズを測定することで、訪問者がFacebookにログインしているかどうかを判断できるかもしれません。しかし、ほとんどのサードパーティコンテンツの場合、`timing-allow-origin`ヘッダーを設定することは、ホスティングウェブサイトがサードパーティコンテンツのパフォーマンス

270. https://developer.mozilla.org/docs/Web/API/Resource_Timing_API/Using_the_Resource_Timing_API

とサイズを追跡できるようにするための透明性の行為です。

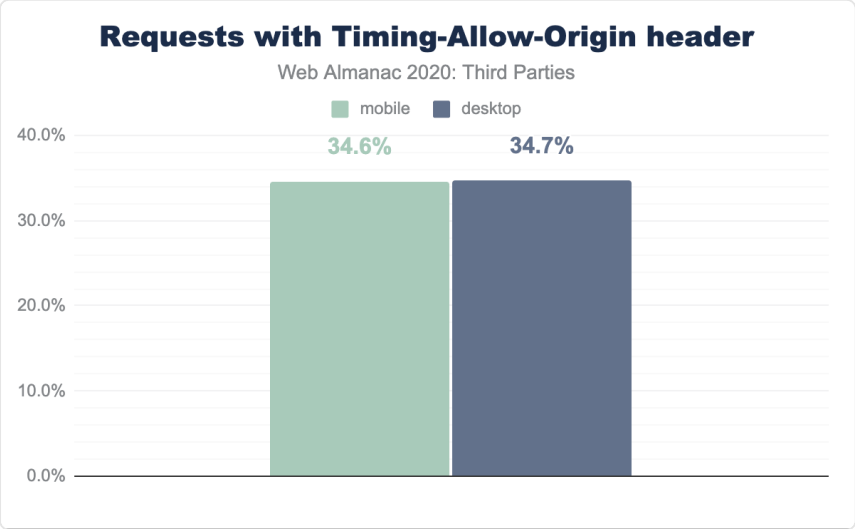


図6.11. `timing-allow-origin` *ヘッダを持つリクエスト。*

HTTP Archiveの結果は、サードパーティのレスポンスの3分の1だけが、ホスティングウェブサイトに詳細なサイズとタイミングの情報を公開していることを示しています。

反響

サイトに任意のJavaScriptを追加すると、サイトの速度とセキュリティの両方にリスクをもたらすことがわかっています。サイトの所有者はサードパーティ製スクリプトの価値と、それらがもたらす可能性のある速度のペナルティのバランスを取るために勤勉でなければならず、また強力なセキュリティ態勢を維持するために、サブリソースの完全性[271]やコンテンツセキュリティポリシー[272]のような最新の機能を使用しなければなりません。これらおよびその他のブラウザのセキュリティ機能の詳細については、セキュリティの章を参照してください。

結論

2020年のデータで驚いたのは、相対的なJavaScriptリクエストの減少です。ウェブ上の

271. *https://developer.mozilla.org/docs/Web/Security/Subresource_Integrity*
272. *https://developer.mozilla.org/docs/Web/HTTP/CSP*

JavaScriptの実際の使用量がこのように大幅に減少したとは考えられませんが、ウェブサイトが同意管理を導入している可能性が高く、ほとんどの動的なサードパーティコンテンツはユーザーのオプトインによってのみ読み込まれるようになっています。このオプトインプロセスは、場合によっては、同意管理プラットフォーム（CMP）によって管理される可能性があります。サードパーティデータベースにはまだCMPのカテゴリがありませんが、これは2021年のWeb Almanacの分析に適しており、プライバシーの章の別の方法論でカバーされています。

広告依頼は、CPU時間に与える影響が大きくなるようです。広告スクリプトを使用した中央値のページは、なしのものと同じくらいのCPUを3倍消費します。しかし興味深いことに、広告スクリプトはページの重さの増加とは相関関係がありません。このことから、単にリクエスト数やサイズだけでなく、ブラウザ上のサードパーティ製スクリプトの総合的な影響を評価することがさらに重要になります。

サードパーティのコンテンツは多くのウェブサイトにとって重要ですが、各プロバイダーの影響を監査することは、ユーザー体験、ページの重さ、またはCPU使用率に大きな影響を与えないようにするために非常に重要です。サードパーティのウェイトに貢献している上位のプロバイダーには、セルフホスティングのオプションがあることが多いですが、共有資産を使用することによるキャッシングのメリットがないため、これは特に検討する価値があります。

- Google Fontsは、セルフホスティング[273]の資産を許可します。
- JavaScriptのCDNをセルフホストのアセットに置き換えることができます。
- 実験スクリプトは、セルフホスト型にできます。 Optimizely[274]

この章では、ウェブ上のサードパーティコンテンツのメリットとコストについて説明してきました。サードパーティはほぼすべてのウェブサイトに不可欠であり、その影響はサードパーティのプロバイダーによって異なることがわかりました。新しいサードパーティをページに追加する前に、サードパーティが与える影響を考えてみてください。

273. *https://www.tunetheweb.com/blog/should-you-self-host-google-fonts/*
274. *https://help.optimizely.com/Set_Up_Optimizely/Optimizely_self-hosting_for_Akamai_users*

著者

Simon Hearne

@simonhearne simonhearne https://simonhearne.com

Simonはウェブパフォーマンスアーキテクトです。彼は、より速く、よりアクセスしやすいウェブを提供することに情熱を注いでいます。@SimonHearneでツイートしたり、simonhearne.com[275]でブログを書いています。

275. *https://simonhearne.com*

部 II 章 7
SEO

Aleyda Solis、 Michael King と Jamie Indigo によって書かれた。
Nate Dame、 Catalin Rosu、 Dave Sottimano、 Dave Smart、 Dustin Montgomery、 Sawood Alam と Barry Pollard によってレビュ。
Tony McCreath と Antoine Eripret による分析。
Rick Viscomi 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）とは、ウェブサイトの技術的な構成、コンテンツの関連性、リンクの人気度などを最適化することで、ユーザーの検索ニーズを満たすために情報を見つけやすくし関連性を高めることです。その結果、Webサイトのコンテンツやビジネスに関連するユーザーの検索結果における視認性が向上し、トラフィック、コンバージョン、利益が増加します。

SEOは複雑な複合領域であるにもかかわらず、近年ではもっとも人気のあるデジタルマーケティング戦略・チャネルの1つに進化しています。

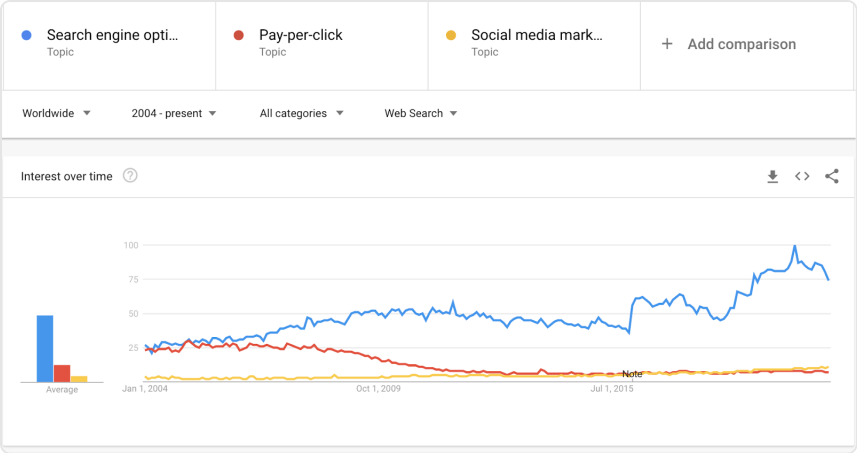


図7.1. Google TrendsによるSEOとペイパークリックやソーシャルメディアマーケティングの比較。

Web AlmanacのSEOの章の目的は、Webサイトのオーガニック検索最適化に役割を果たす主な要素や構成を特定し、評価することです。これらの要素を特定することでウェブサイトが検索エンジンにクロールされ、インデックスされ、ランキングされる能力を向上させるために我々の調査結果を活用していただきたいと考えています。本章では、2020年の状態のスナップショットと、2019年以降[276]に変更された点のまとめを提供します。

この章は、モバイルサイトのLighthouse[277]の分析に基づいていることに注意してください。モバイルとデスクトップ、および未加工のChromeUXレポート[278] モバイルとデスクトップでHTTP Archive[279]からHTML要素をレンダリングしました。 HTTP ArchiveとLighthouseの場合、サイト全体のクロールではなく、Webサイトのホームページからのみ識別されるデータに制限されます。 評価を行う際、これを考慮に入れています。 結果から結論を引き出すときは、この区別を念頭に置くことが重要です。 詳細については、方法論ページをご覧ください。

今年のオーガニック検索最適化の主な調査結果を見てみましょう。

ファンダメンタルズ

このセクションでは検索エンジンがウェブサイトを正しくクロールし、インデックスを作成し、ユーザーに最適な結果を提供するための基盤となるウェブの構成や要素について最適化

276. *https://almanac.httparchive.org/ja/2019/seo*
277. *https://developers.google.com/web/tools/lighthouse/*
278. *https://developers.google.com/web/tools/chrome-user-experience-report*
279. *https://httparchive.org/*

に関する知見を紹介します。

クローラビリティとインデクサビリティ

検索エンジンはウェブ・クローラー（スパイダーとも呼ばれる）を使用して、ページ間のリンクをたどってウェブを閲覧しながら、ウェブサイトから新規または更新されたコンテンツを発見します。クローリングとは、新しいまたは更新されたWebコンテンツ（Webページ、画像、動画など）を探すプロセスのことです。

検索クローラーは、URL間のリンクを辿るだけでなく、ウェブサイトの所有者が提供する追加ソースを利用してコンテンツを発見します。たとえば、XMLサイトマップ（ウェブサイトの所有者が検索エンジンにインデックスさせたいURLのリスト）の作成や、Googleのサーチコンソールのような検索エンジンツールを使った直接のクロールリクエストなどがあります。

検索エンジンはウェブコンテンツにアクセスすると、ウェブブラウザと同様にレンダリングを行い、インデックスを作成する必要があります。検索エンジンは識別された情報を分析してカタログ化し、ユーザーと同じように理解しようとし、最終的にインデックス（ウェブデータベース）に保存します。

ユーザーがクエリを入力すると、検索エンジンはインデックスを検索してクエリに答えるため、検索結果ページに表示する最適なコンテンツを探し、さまざまな要素を用いてどのページが他のページよりも先に表示されるかを決定します。

検索結果での表示を最適化したいと考えているウェブサイトでは、特定のクローラビリティとインデクサビリティのベストプラクティスに従うことが重要です。たとえば、`robots.txt`、robots `meta`タグ、`X-Robots-Tag` HTTPヘッダー、canonicalタグなどを正しく設定することです。これらのベストプラクティスは、検索エンジンがウェブコンテンツに容易にアクセスし、より正確にインデックスを作成するのに役立ちます。以下では、これらの設定を徹底的に分析します。

`robots.txt`

サイトのルートに置かれる`robots.txt`ファイルは検索エンジンのクローラーがどのページを操作するか、どのくらいの速さでクロールするか、そして発見されたコンテンツをどうするかをコントロールするための効果的なツールです。

Googleは、2019年に`robots.txt`をインターネットの公式規格にすることを正式に提案しました。2020年6月のドラフト[280]には、`robots.txt`ファイルの技術的要件に関する明確な

280. https://tools.ietf.org/html/draft-koster-rep-02

文書が含まれています。これにより、検索エンジンのクローラーが規格外のコンテンツにどのように対応すべきか、より詳細な情報が求められるようになりました。

`robots.txt` ファイルはプレーンテキストで、UTF-8でエンコードされ、リクエストに対して200 HTTPステータスコードで応答しなければなりません。検索エンジンのクローラーは、不正な `robots.txt` や4XX（クライアントエラー）レスポンス、5つ以上のリダイレクトは *full allow* と解釈し、すべてのコンテンツはクロールされる可能性があることを意味します。5XX（サーバーエラー）のレスポンスは、 *full disallow* と解釈され、いかなるコンテンツもクロールできないことを意味します。もし `robots.txt` が30日以上届かない場合、Googleは仕様[281]で説明されているように、最後にキャッシュされたコピーを使用します。

全体として、80.46%のモバイルページが `robots.txt` に対して2XXの応答をしました。このうち、25.09%は有効であると認識されませんでした。これは、27.84%のモバイルサイトが有効な `robots.txt` を持っていたことが判明した2019年よりも若干改善されています。

`robots.txt` の有効性をテストするためのデータソースであるLighthouseは、v6アップデートの一環として、 `robots.txt` の監査を導入しました。この導入により、リクエストが正常に解決されても、基礎ファイルがWebクローラーに必要な指示を与えることができるとは限らないことが強調されました。

レスポンスコード	*モバイル*	*デスクトップ*
2XX	*80.46%*	*79.59%*
3XX	*0.01%*	*0.01%*
4XX	*17.67%*	*18.64%*
5XX	*0.15%*	*0.12%*
6XX	*0.00%*	*0.00%*
7XX	*0.15%*	*0.12%*

図7.2. `robots.txt` のレスポンスコード。

ステータスコードが似ていることに加えて、モバイル版とデスクトップ版の `robots.txt` ファイルでは、 `Disallow` ステートメントが一貫して使用されていました。

もっとも多く見られた `User-agent` 宣言文は、ワイルドカードの `User-agent: *` 、これはモバイルの `robots.txt` リクエストの74.40%とデスクトップの73.16%に見られました。2番目に多かった宣言文は `adsbot-google` で、モバイルの `robots.txt` リクエストの5.63%とデスクトップの5.68%に現れていました。Google AdsBotは、ウェブページやアプリの広

281. *https://developers.google.com/search/reference/robots_txt#handling-http-result-codes*

告の品質をデバイス間でチェックするため、ワイルドカードの記述を無視し、具体的な名前を指定する必要があります。

もっとも頻繁に使用されたディレクティブは、検索エンジンとその有料マーケティングに関連するものでした。SEOツールのAhrefとMajesticは、どちらのデバイスでも `Disallow` ステートメントのトップ5に入っていました。

	robots.txtの使用率	
ユーザーエージェント	*モバイル*	*デスクトップ*
*	*74.40%*	*73.16%*
adsbot-google	*5.63%*	*5.68%*
mediapartners-google	*5.55%*	*3.83%*
mj12bot	*5.49%*	*5.30%*
ahrefsbot	*4.80%*	*4.66%*

図7.3. `robots.txt User-agent` のディレクティブです。

Lighthouseが提供する600万以上のサイトのデータを使って、 `robots.txt` の `Disallow` 文の使用状況を分析したところ、97.84%が完全にクロール可能で、 `Disallow` 文を使用しているのは1.05%に過ぎないことがわかりました。

また、meta robots[282] *indexability* ディレクティブに沿った `robots.txt` の `Disallow` ステートメントの使用状況を分析したところ、1.02%のサイトが、meta robots `index` ディレクティブを使用したインデックス可能なページで `Disallow` ステートメントを使用しており、meta robots `noindex` ディレクティブを使用した *noindex* ページで `robots.txt` の `Disallow` ステートメントを使用しているサイトは0.03%しかありませんでした。

これは、Googleのドキュメント[283]によると、サイトオーナーはGoogle検索からウェブページを隠す手段として `robots.txt` を使用すべきではないとしています。代わりに、サイトオーナーは、meta robotsによる `noindex` ディレクティブのような他の方法を使うべきです。

Meta robots

メタタグ `robots` とHTTPヘッダー `X-Robots-Tag` は、提案されている Robots Exclusion Protocol[284] (REP) を拡張したもので、より詳細なレベルでディレクティブを設定できます。

282. *https://developers.google.com/search/reference/robots_meta_tag*
283. *https://developers.google.com/search/docs/advanced/robots/intro*
284. *https://webmasters.googleblog.com/2019/07/rep-id.html*

REPはまだ正式なインターネット標準ではないため、ディレクティブのサポートは検索エンジンによって異なります。

メタタグは、デスクトップの27.70%、モバイルの27.96%のページで使用されており、粒度の高い実行方法として圧倒的な支持を得ていました。`X-Robots-Tag`は、デスクトップで0.27%、モバイルでは0.40%でした。

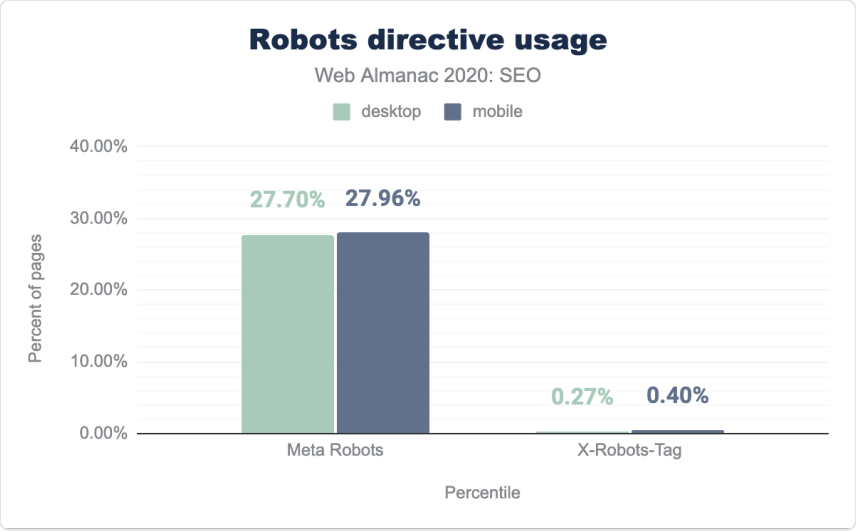


図7.4. meta robotsと`X-Robots-Tag`ディレクティブの使用。

Lighthouseのテストでmeta robotsタグの使用状況を分析したところ、クロール可能なページの0.47%が *noindex* になっていることがわかりました。これらのページの0.44%は、`noindex`ディレクティブを使用しており、`robots.txt`でそのページのクロールを禁止していませんでした。

また、robots.txt内の`Disallow`とmeta robots内の`noindex`ディレクティブの組み合わせは、わずか0.03%のページでしか見つかりませんでした。この方法は、ベルトとサスペンダーのような冗長性がありますが、ページ上の`noindex`ディレクティブを有効にするためには、ページが`robots.txt`ファイルによってブロックされていない必要があります。

興味深いことに、レンダリングによってmeta robotsタグが変更されたページは0.16%でした。JavaScriptを使ってページにmeta robotsタグを追加したり、コンテンツを変更したりすることに本質的な問題はありませんが、SEO担当者は慎重に実行する必要があります。レンダリング前、meta robotsタグに`noindex`ディレクティブを入れてページを読み込むと、検索エンジンはタグの値を変更するJavaScript[285]を実行したり、ページをインデックスできませ

285. *https://developers.google.com/search/docs/guides/javascript-seo-basics#use-meta-robots-tags-carefully*

ん。

正規化

Canonicalタグ[286]はGoogleが説明しているように、同一または非常に類似したコンテンツを掲載しているURLが多数ある場合に、そのページを代表すると考えられる優先的なcanonical URLバージョンを検索エンジンに指定するため使用されます。重要なのは以下の点です。

- canonicalタグの設定は、ページのcanonical URLを選択するために他のシグナルとともに使用されますが、それだけではありません。
- 自己参照用のcanonicalタグが使われることもありますが、必須ではありません。

昨年の章[287]では、モバイルページの48.34%がcanonicalタグを使用していることが確認されました。今年はcanonicalタグを採用しているモバイルページは53.61%に増えています。

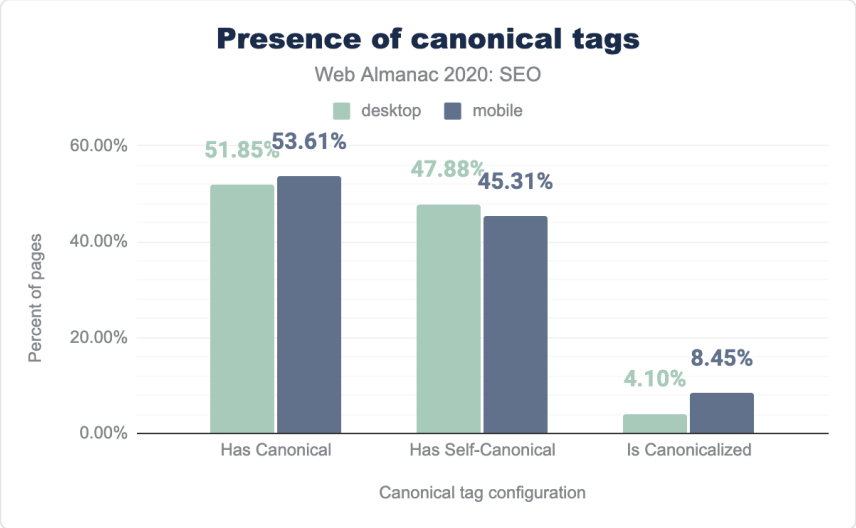


図7.5. canonicalタグの使い方。

今年のモバイルページのcanonicalタグの構成を分析したところ、45.31%が自己参照で、8.45%が正規のものとは異なるURLを指していることが検出されました。

一方、今年は51.85%のデスクトップページがcanonicalタグを採用していることが判明し、47.88%が自己参照、4.10%が別のURLを指していることがわかりました。

286. https://developers.google.com/search/docs/advanced/crawling/consolidate-duplicate-urls
287. https://almanac.httparchive.org/ja/2019/seo#canonicalization

モバイルページはデスクトップページよりも多くのcanonicalタグを含んでいる（53.61%対51.85%）だけでなく、他のURLにcanonical化しているモバイルホームページは、デスクトップホームページよりも比較的多い（8.45%対4.10%）。これは、デスクトップ版とは別のURLに正規化する必要がある一部のサイトが、独立した（または別の）モバイルウェブ版を使用しているためと考えられます。

正準化URLはHTTPヘッダーやページのHTMLの `head` を介して正準化リンクを使用する方法と、XMLサイトマップで提出する方法など、さまざまな方法で指定できます。正準化リンクの実装方法としてもっとも多く利用されているのはどれかを分析したところ、HTTPヘッダーに依存して実装しているのは、デスクトップページで1.03%、モバイルページでは0.88%に過ぎず、正準化タグはページのHTMLの `head` を介して実装されることが多いことがわかりました。

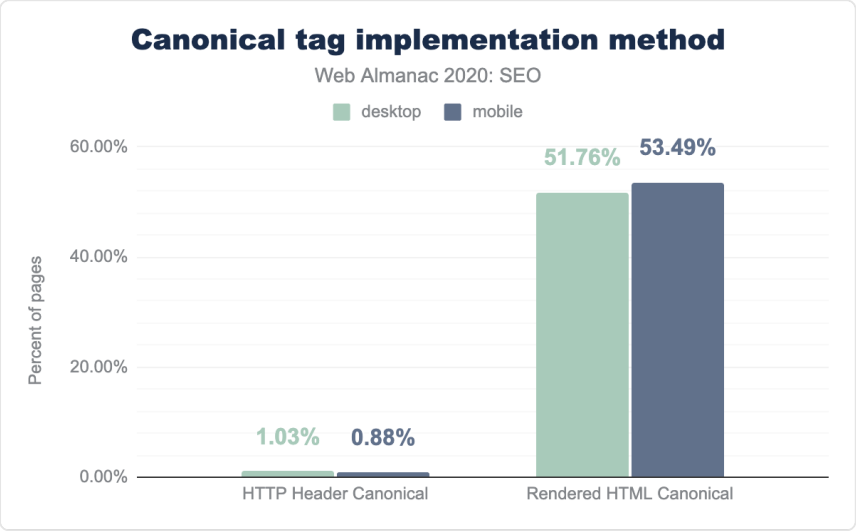


図7.6. HTTPヘッダーとHTML `head` の正規化メソッドの使用。

生のHTMLにcanonicalタグが実装されているものと、クライアントサイドのJavaScriptによるレンダリングに依存しているものを分析したところ、モバイルページの0.68%、デスクトップページの0.54%が、生のHTMLではなくレンダリングされたHTMLにcanonicalタグを含んでいることがわかりました。つまり、JavaScriptでcanonicalタグを実装しているページは、ごく少数であることがわかります。

一方、モバイルページの0.93%、デスクトップページの0.76%では、生のHTMLとレンダリングされたHTMLの両方でcanonicalタグが実装されており、同じページの生のHTMLで指定されたURLとレンダリングされたHTMLで指定されたURLの間に矛盾が生じていました。これにより、同じページのcanonical URLがどちらであるかという情報が混在して検索エンジンに

送信されるため、インデクサビリティの問題が発生します。

また、実装方法の違いによっても同様の問題が発生しており、モバイルページの0.15%とデスクトップページの0.17%で、HTTPヘッダーとHTMLの `head` で実装されたcanonicalタグの間で問題が発生しています。

コンテンツ

検索エンジンと検索エンジン最適化の主な目的は、ユーザーが必要とするコンテンツを可視化することです。検索エンジンは、ページの特徴を抽出してコンテンツの内容を判断します。このように、両者は共生しています。抽出された特徴は、関連性を示すシグナルと一致し、ランキングに反映されます。

検索エンジンが何を効果的に抽出できているかを理解するために、そのコンテンツの構成要素を分解し、モバイルとデスクトップのコンテキスト間でそれらの特徴の発生率を調べました。また、モバイルとデスクトップのコンテンツの格差についても検討しました。モバイルとデスクトップの格差がとくに重要なのは、Googleがすべての新規サイトに対して *mobile-first indexing* （MFI）は、すべての新しいサイトで2021年3月の時点でモバイルコンテキスト内に表示されない *mobile-only index* iSeriesコンテンツへ移動します。ランキングの評価は行われません。

レンダリングされたテキストコンテンツとレンダリングされていないテキストコンテンツ

ウェブの発展に伴い、シングルページアプリケーション（SPA）JavaScript技術の使用が爆発的に増加しています。このデザインパターンは実行時のJavaScript変換の実行と、ロード後のページでのユーザーのインタラクションの両方が、追加コンテンツの表示やレンダリングを引き起こす可能性があるため、検索エンジンのスパイダーにとっては困難なものとなっています。

検索エンジンはそのクロール活動を通じてページに出会いますが、ページをレンダリングする第2段階を実施することを選択する場合もあれば、しない場合もあります。その結果、ユーザーが目にするコンテンツと、検索エンジンがインデックスを作成し、ランキングの対象とするコンテンツとの間に差異が生じることがあります。

その格差のヒューリスティックな指標として、語数を評価しました。

価値観	デスクトップ	モバイル	差異
そのまま	360	312	-13.33%
レンダリング	402	348	-13.43%
差異	**11.67%**	**11.54%**	

図7.7. デスクトップとモバイルのページあたりのそのままの単語数とレンダリングされた単語数の中央値の比較。

今年は、デスクトップページの中央値が402語、モバイルページの中央値は348語であることがわかりました。一方、昨年[288]では、デスクトップページの中央値は346語、モバイルページの中央値は306語と、わずかに語数が少なかった。これはそれぞれ16.2%と13.7%の成長を示しています。

その結果、デスクトップサイトの中央値は、生のHTMLを最初にクロールしたときと比較して、レンダリング時に表示される文字数は11.67%多いことがわかりました。また、モバイルサイトの中央値は、デスクトップサイトに比べてテキストコンテンツの表示は13.33%少ないことがわかりました。モバイルサイトの中央値は、生のHTMLに比べてレンダリング時に表示される単語数は11.54%多いことがわかりました。

今回のサンプルセットでは、モバイル／デスクトップ、レンダリング／非レンダリングの組み合わせに格差が見られました。これは検索エンジンがこの分野で継続的に改善しているにもかかわらず、ウェブ上のほとんどのサイトは、コンテンツが利用可能でインデックス可能であることを確実にすることへ集中することでオーガニック検索の可視性を向上させる機会を逃していることを示唆しています。また多くのSEOツールは、上記のようなコンテキストの組み合わせではクロールせず、自動的に問題として認識するためこの点も懸念されます。

見出し

見出し要素（`H1`〜`H6`）は、ページのコンテンツの構造を視覚的に示す仕組みとして機能します。これらのHTML要素は検索ランキングにおいて以前のような重みはありませんが、ページを構造化し、 *featured snippets* やGoogleの新しいパッセージインデックス[289]に沿った他の抽出方法のように検索エンジン結果ページ（SERP）の他の要素にシグナルを送るための貴重な方法として機能しています。

288. *https://almanac.httparchive.org/ja/2019/seo#word-count*
289. *https://www.blog.google/products/search/search-on/*

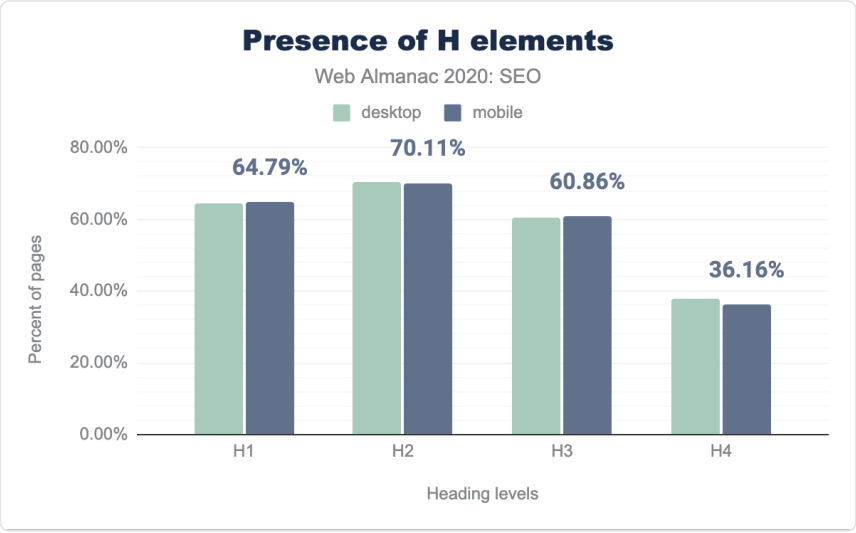


図7.8. 空の見出しを含む、見出しレベル1から4の使用。

60%以上のページが、モバイルとデスクトップの両方で `H1` 要素（空のものも含む）を使用しています。

これらの数字は、 `H2` と `H3` で60%以上を占めています。 `H4` 要素の出現率は4%以下で、ほとんどのページではこのレベルの特殊性は必要とされていないか、開発者がコンテンツの視覚的構造をサポートするために他の見出し要素を別のスタイルにしていることを示唆しています。

`H2` 要素が `H1` 要素よりも多いということは、複数の `H1` 要素を使用しているページが少ないことを示唆しています。

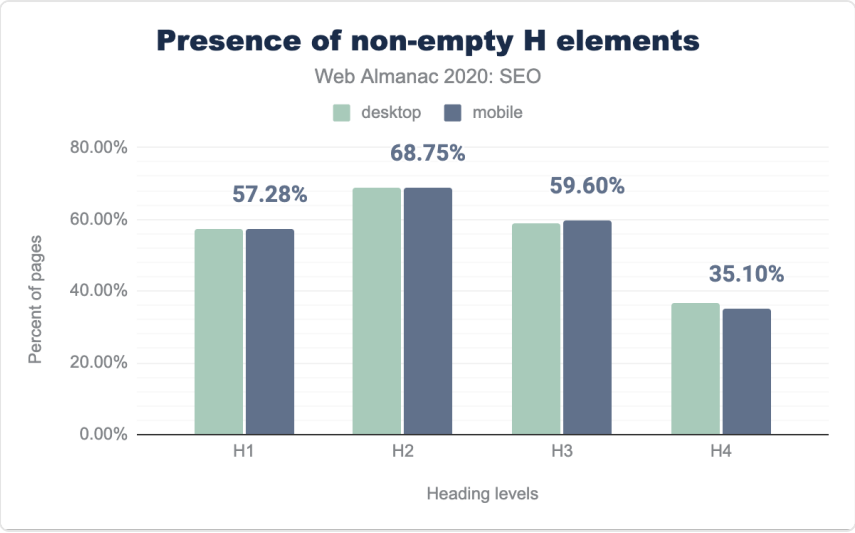


図7.9. 空の見出しを除く、見出しレベル1から4の使用法。

空ではない見出し要素の採用状況を確認したところ、テキストがない要素は、`H1`が7.55%、`H2`が1.4%、`H3`が1.5%、`H4`が1.1%でした。このような結果になった理由としては、これらの部分がページのスタイリングのために使われているか、あるいはコーディングミスの結果であることが考えられます。

見出しの使い方については、マークアップの章で詳しく説明していますが、これには非標準の`H7`や`H8`要素の誤用も含まれています。

構造化データ

過去10年間、検索エンジン、とくにGoogleは、ウェブのプレゼンテーション層になることを目指して邁進し続けてきました。これらの進歩は、非構造化コンテンツから情報を抽出する能力の向上（例：パッセージ・インデックス[290]）や、*構造化データ*という形でのセマンティック・マークアップの採用によって部分的に推進されています。検索エンジンはコンテンツ制作者や開発者に対して、検索結果のコンポーネント内でコンテンツをより見やすくするために、構造化データを実装することを推奨しています。

検索エンジンは、「文字列からモノへ」[291]という流れの中で、ウェブコンテンツ内のさまざまな人、場所、モノをマークアップするための、幅広いオブジェクトの語彙に合意しました。しかし、その語彙の一部だけが、検索結果のコンポーネントに含まれます。Googleは、

290. https://blog.google/products/search/search-on/
291. https://blog.google/products/search/introducing-knowledge-graph-things-not/

サポートしている語彙とその表示方法をsearch gallery[292]で指定し、そのサポートと実装を検証するためのツール[293]を提供しています。

検索エンジンが進化してこれらの要素が検索結果へ反映されるようになると、異なる語彙の発生率がウェブ上で変化します。

検証の一環として、さまざまな種類の構造化マークアップの発生率を調べてみました。利用可能な語彙には、RDFa[294]とschema.org[295]があり、これらにはmicroformatsとJSON-LD[296]の両方の種類があります。Googleは最近data-vocabulary[297]のサポートを終了しましたが、これは主にパンくずの実装に使用されていました。

一般的に、JSON-LDはよりポータブルで管理しやすい実装であると考えられているため、このフォーマットが好まれています。その結果、モバイルページの29.78%、デスクトップページの30.60%にJSON-LDが採用されていることがわかります。

フォーマット	モバイル	デスクトップ
JSON-LD	29.78%	30.60%
Microdata	19.55%	17.94%
RDFa	1.42%	1.63%
Microformats2	0.10%	0.10%

図7.10. 各構造化データフォーマットの使い方

このタイプのデータでは、モバイルとデスクトップの格差が続いていることがわかります。Microdataは、モバイルページの19.55%、デスクトップページの17.94%に掲載されています。RDFaは、モバイルページの1.42%、デスクトップページの1.63%に掲載されています。

レンダリングされた構造化データとレンダリングされていない構造化データ

その結果、デスクトップページの38.61%、モバイルページの39.26%が、生のHTMLにJSON-LDまたはmicroformatの構造化データを搭載しており、デスクトップページの40.09%、モバイルページの40.97%がレンダリングされたDOMに構造化データを搭載していることがわかりました。

292. https://developers.google.com/search/docs/guides/search-gallery
293. https://search.google.com/test/rich-results
294. https://www.w3.org/TR/rdfa-primer/
295. https://schema.org/
296. https://www.w3.org/TR/json-ld11/
297. https://developers.google.com/search/blog/2020/01/data-vocabulary

さらに詳しく調べてみるとデスクトップページの1.49%、モバイルページの1.77%が、検索エンジンのJavaScriptの実行能力に依存して、JavaScriptの変換によってレンダリングされたDOM内にこの種の構造化データを掲載していることがわかりました。

最後にデスクトップページの4.46%、モバイルページの4.62%が、生のHTMLに表示され、その後レンダリングされたDOM内でJavaScriptの変換によって変更された構造化データを備えていることがわかりました。構造化データの構成に適用された変更の種類によっては、検索エンジンがレンダリングする際に複雑なシグナルを生成する可能性があります。

もっとも一般的な構造化データオブジェクト

昨年に引き続き[298]、もっとも普及している構造化データオブジェクトは`WebSite`、`SearchAction`、`WebPage`、`Organization`、`ImageObject`であり、これらの使用量は増え続けています。

- `WebSite`は、デスクトップで9.37%、モバイルで10.5%の成長を遂げました。
- `SearchAction`はデスクトップとモバイルの両方で7.64%成長しました。
- `WebPage`はデスクトップで6.83%、モバイルで7.09%成長しました。
- `Organization`はデスクトップで4.75%、モバイルで4.98%成長しました。
- `ImageObject`の成長率は、デスクトップで6.39%、モバイルで6.13%でした。

`WebSite`、`SearchAction`、`Organization`はいずれも一般的にホームページに関連するもので、これはデータセットの偏りを示すものであり、ウェブ上で実装されている構造化データの大部分を反映しているわけではありません。

一方、レビューはホームページと関連づけられるべきではないにもかかわらず、データによると、モバイルでは23.9%、デスクトップでは23.7%に`AggregateRating`が使用されています。

また、動画に注釈をつけるための`VideoObject`の成長も興味深いものがあります。Googleの動画検索結果ではYouTubeの動画が圧倒的に多いのですが[299]、`VideoObject`の使用率はデスクトップで30.11%、モバイルで27.7%と伸びています。

これらのオブジェクトの増加は、構造化データの導入が進んでいることを一般的に示しています。また、Googleが検索機能の中で可視化すると、あまり使われていないオブジェクトの発生率が高まることも示唆されています。Googleは2019年に`FAQPage`、`HowTo`、`QAPage`のオブジェクトを可視化の機会として発表し、それらは前年比で大幅な成長を維持

298. *https://almanac.httparchive.org/ja/2019/seo#structured-data*
299. *https://moz.com/blog/youtube-dominates-google-video-results-in-2020*

しました。

- `FAQPage` のマークアップは、デスクトップでは3,261%、モバイルでは3,000%増加しました。
- `HowTo` マークアップは、デスクトップでは605%、モバイルでは623%増加しました。
- `QAPage` のマークアップは、デスクトップで166.7%、モバイルで192.1%増加しました。

繰り返しになりますが、これらのオブジェクトは通常、内部ページに配置されているため、このデータは彼らの実際の成長レベルを代表していない可能性があることに注意する必要があります。

構造化データの採用は、データを抽出することが豊富なユースケースにとって価値があるため、ウェブに恩恵をもたらします。検索エンジンの利用が拡大し、ウェブ検索以外のアプリケーションにも利用されるようになると、構造化データは今後も増加すると予想されます。

メタデータ

メタデータは、クリックの向こう側にあるコンテンツの価値を説明し、解説する機会です。ページタイトルは検索順位に直接影響すると考えられていますが、メタ・ディスクリプションはそうでありません。どちらの要素もその内容に基づいて、ユーザーのクリックを促したり、阻害したりできます。

これらの特徴を調べ、ページがオーガニック検索からのトラフィックを促進するためのベストプラクティスに定量的に合致しているかを確認しました。

タイトル

ページタイトルは検索エンジンの検索結果にアンカーテキストとして表示され、一般的に、ページのランキングに影響を与えるもっとも価値のあるオンページ要素の1つと考えられています。

`title` タグの使用状況を分析したところ、デスクトップおよびモバイルページの99%にタグが付いてることがわかりました。これは、モバイルページの97%が `title` タグを持っていた昨年[300]からわずかに改善されています。

中央値のページは、ページタイトルが6語となっています。このデータセットでは、モバイルとデスクトップの間で単語数に違いはありませんでした。これは、ページタイトル要素

300. *https://almanac.httparchive.org/ja/2019/seo#page-titles*

が、異なるページテンプレートタイプ間で変更されない要素であることを示唆しています。

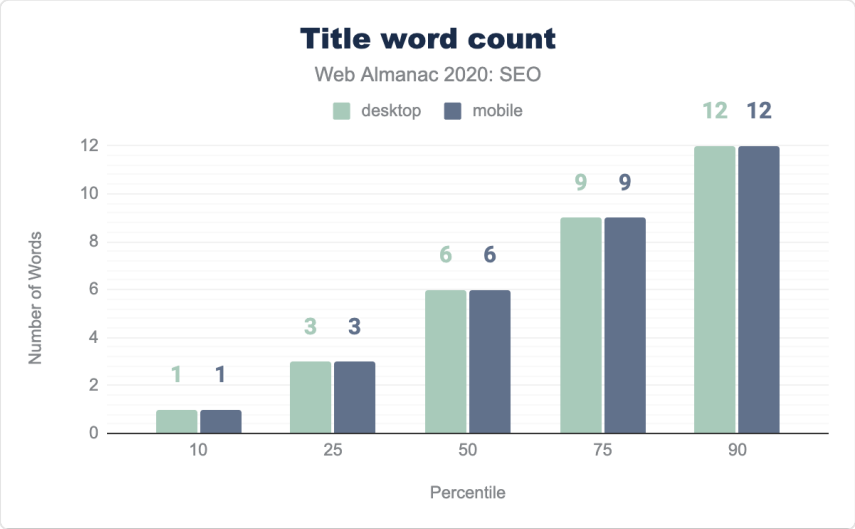


図7.11. ページタイトルごとの文字数の分布。

ページタイトルの文字数の中央値は、モバイルとデスクトップの両方で38文字となっています。興味深いことに、昨年の分析結果[301]ではデスクトップで20文字、モバイルで21文字でしたが、今回はそれよりも増えています。コンテキスト間の格差は、90パーセンタイル内で1文字の差があることを除いて、前年比でなくなっています。

301. *https://almanac.httparchive.org/ja/2019/seo#page-titles*

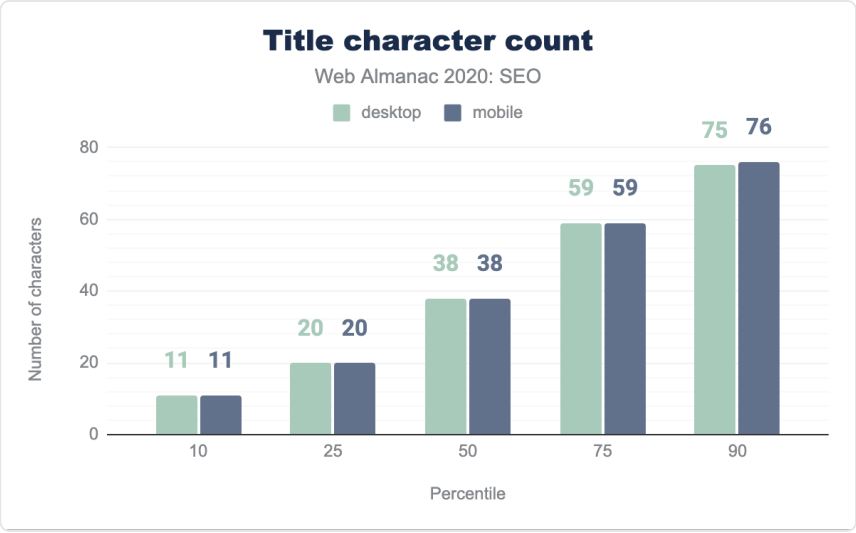


図7.12. ページタイトルごとの文字数の分布。

メタ・ディスクリプション

meta descriptionは、Webページの広告キャッチフレーズの役割を果たします。最近の調査[302]によると、このタグは70%の確率でGoogleに無視され、書き換えられているという結果が出ていますがユーザーのクリックを誘う目的で用意される要素です。

メタ・ディスクリプションの使用状況を分析したところ、デスクトップ・ページの68.62%、モバイル・ページの68.22%にメタ・ディスクリプションがあることがわかりました。これは意外と低いかもしれませんが、モバイルページの64.02%しかメタディスクリプションを持っていなかった昨年[303]と比べると、少し改善されています。

302. https://www.searchenginejournal.com/google-rewrites-meta-descriptions-over-70-of-the-time/382140/
303. https://almanac.httparchive.org/ja/2019/seo#meta-descriptions

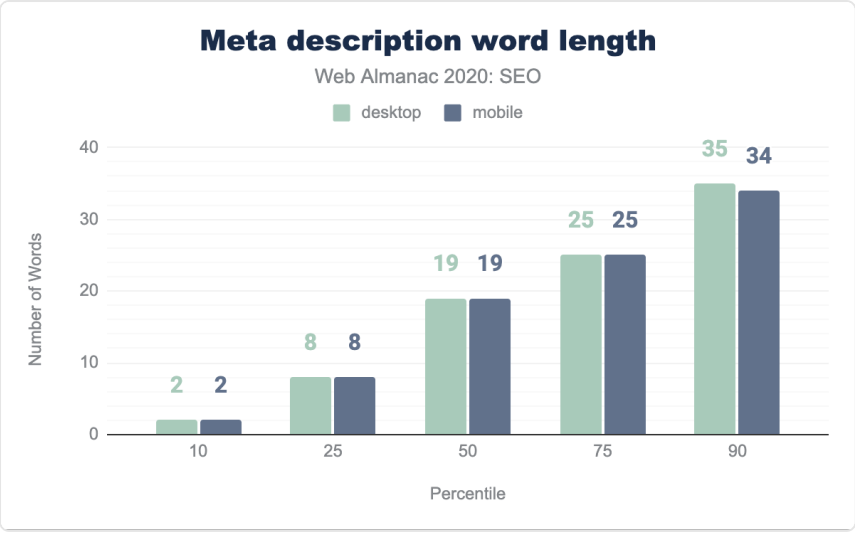


図7.13. メタ・ディスクリプションごとの単語数の分布。

メタディスクリプションの長さの中央値は19Wordです。唯一の単語数の格差は、デスクトップのコンテンツがモバイルよりも1単語多い90パーセンタイルで発生します。

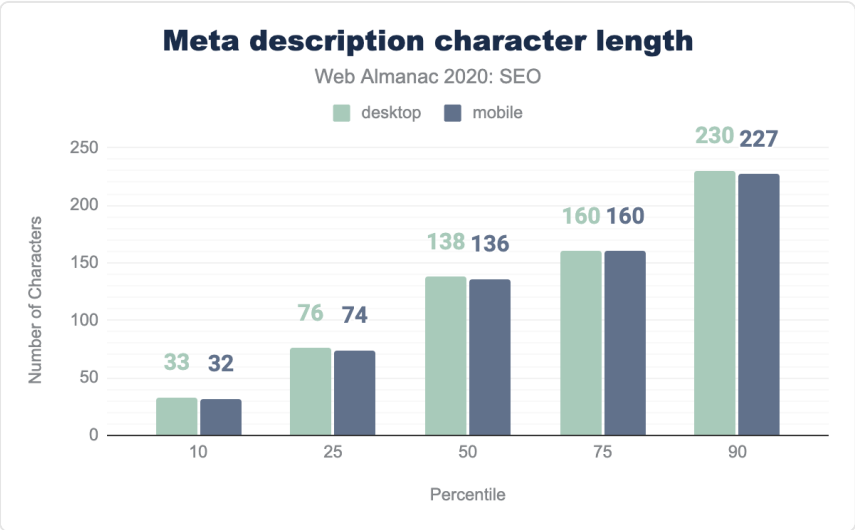


図7.14. メタ・ディスクリプション1つあたりの文字数の分布。

メタディスクリプションの文字数の中央値は、デスクトップページで138文字、モバイルペ

ージで136文字となっています。75パーセンタイルを除けば、データセット全体でモバイルとデスクトップのmeta descriptionの長さにはわずかな差があります。SEOのベストプラクティスでは、指定されたmeta descriptionを160文字以内に抑えることが推奨されていますが、Googleはスニペットで300文字以上を表示することがあります。

メタ・ディスクリプションがソーシャル・スニペットやニュース・フィード・スニペットなどの他のスニペットに影響を与え続けていること、またGoogleがメタ・ディスクリプションを継続的に書き換えており直接的なランキング要因とは考えていないことを考えると、メタ・ディスクリプションが160文字の制限を超えて成長し続けると予想するのは妥当です。

イメージ

ページ内で画像、とくに `img` タグを使用することは、コンテンツを視覚的に表現することに重点を置いていることを意味します。コンピュータビジョンに関する検索エンジンの能力は向上し続けていますが、この技術がページのランキングに使用されているという事実はありません。`alt` 属性は、検索エンジンが画像を「見る」代わりに、画像を説明するための主要な手段であり続けます。また、`alt` 属性はアクセシビリティにも対応しており、視覚障害のあるユーザーに対してページ上の要素を明確にできます。

デスクトップページの中央値には21個の `img` タグが含まれ、モバイルページの中央値には19個の `img` タグが含まれています。帯域幅の拡大やスマートフォンの普及により、ウェブは画像を多用する傾向にあります。しかし、これはパフォーマンスを犠牲にします。

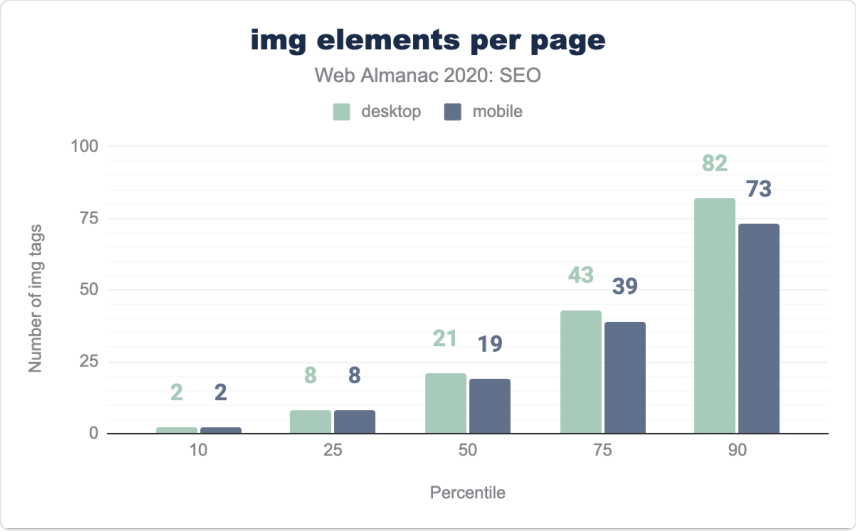


図7.15. ページあたりの `<img>` 要素数の分布。

ウェブページの中央値は、デスクトップでは2.99%の`alt`属性が欠けており、モバイルでは2.44%の`alt`属性が欠けています。`alt`属性の重要性については、アクセシビリティの章を参照してください。

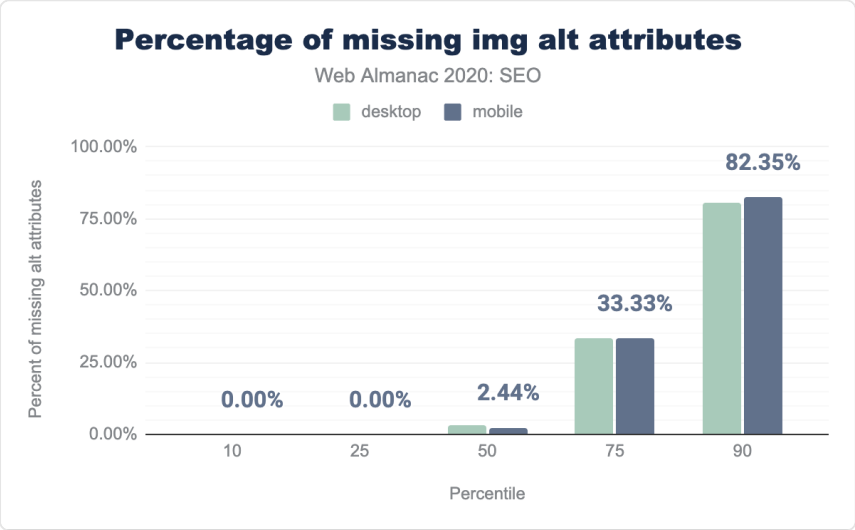


図7.16. イメージの`alt`属性を欠く`<img>`要素のページあたりの割合の分布。

その結果、中央値のページでは、画像の51.22%にしか`alt`属性が含まれていないことがわかりました。

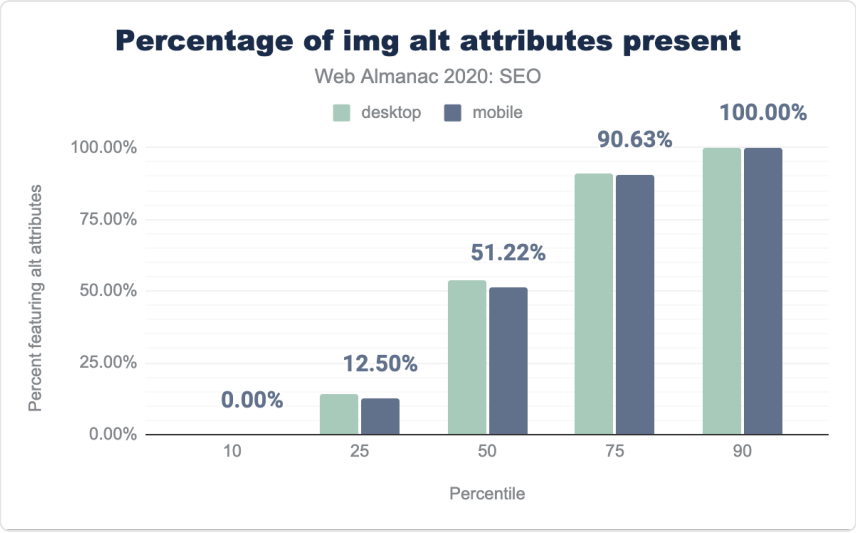


図7.17. ページごとの `alt` 属性を持つ画像の割合の分布。

ウェブページの中央値では、空白の `alt` 属性を持つ画像が、デスクトップは10.00%、モバイルでは11.11%ありました。

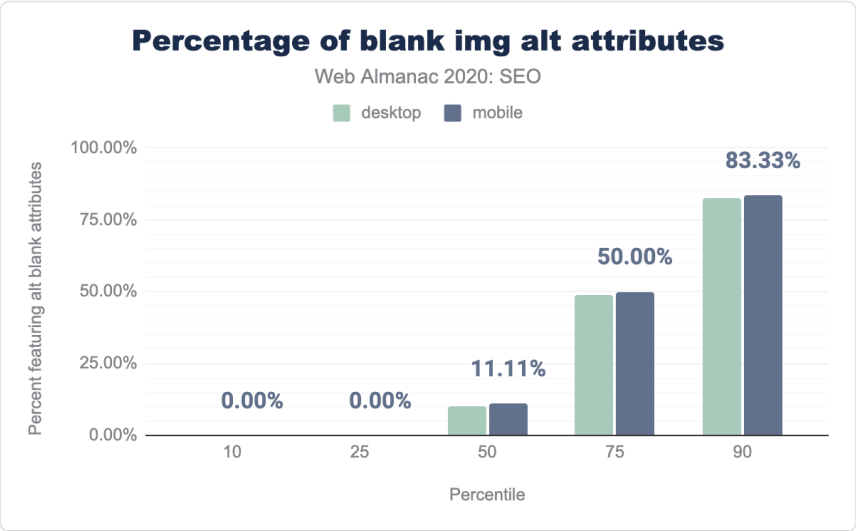


図7.18. 空白の `alt` 属性を持つイメージのページごとの割合の分布。

リンク

現代の検索エンジンはページ間のハイパーリンクを利用して、新しいコンテンツを発見してインデックスを作成したり、権威を示してランキングに反映させたりしています。リンクグラフは、検索エンジンがアルゴリズムと手動によるレビューの両方で積極的に管理しているものです。したがって、ページ全体に豊富なリンクを張ることが重要ですが、GoogleがSEOスターターガイド[304]で言及しているように賢くリンクを張ることも重要です。

発信リンク

この分析の一環として、各ページからの発信リンクを評価できます。同じドメインの内部ページや外部ページへのリンクかどうかを評価できますが、着信リンクは分析していません。

デスクトップページの中央値には76本のリンクがあり、モバイルページの中央値には67本のリンクがあります。歴史的には、Googleからの指示により、リンクは1ページあたり100個までにするよう提案されていました。この推奨事項は現代のウェブでは時代遅れであり、Googleはその後、制限なし[305]と言及していますが、私たちのデータセットの中央値のページはそれを守っています。

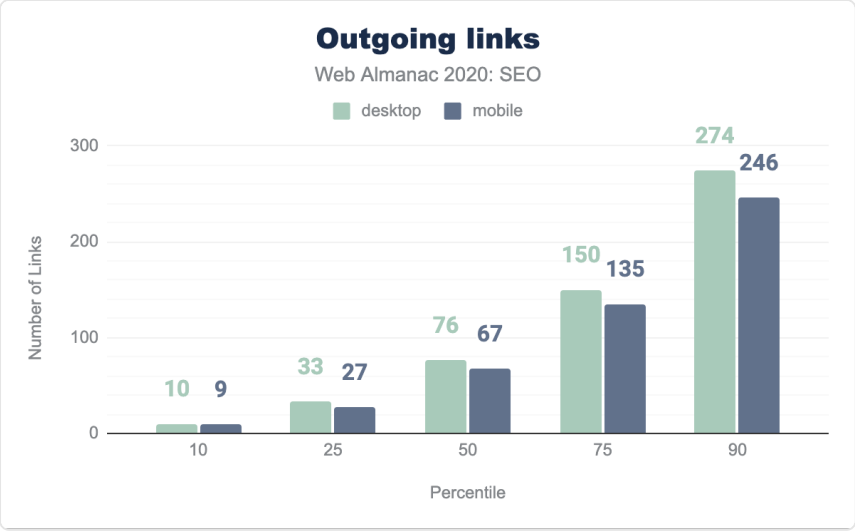


図7.19. ページごとのリンク数の分布。

中央値のページの内部リンク（同じサイト内のページへのリンク）は、デスクトップで61、

304. https://developers.google.com/search/docs/beginner/seo-starter-guide#use-links-wisely
305. https://www.seroundtable.com/google-link-unlimited-18468.html

モバイルで54となっています。これは昨年の分析[306]からそれぞれ12.8%と10%減少しています。このことから、サイトは一昨年のように、ページ内のクローラビリティとリンクエクイティの流れを改善する能力を最大限に発揮できていないと考えられます。

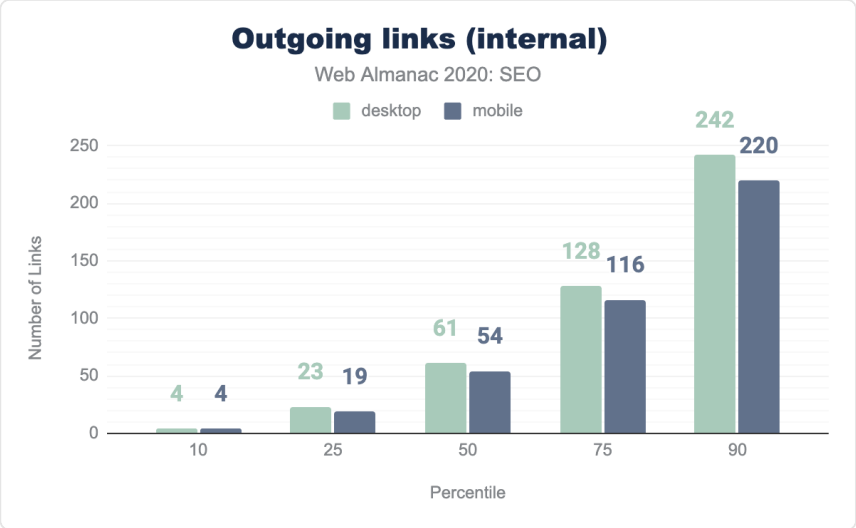


図7.20. ページごとの内部リンク数の分布。

ページの中央値は、外部サイトへのリンクがデスクトップで7回、モバイルで6回となっています。これは、1ページあたりの外部リンク数の中央値がデスクトップで10回、モバイルで8回であった昨年よりも減少しています。このような外部リンクの減少はリンクポピュラリティを渡さないために、あるいはユーザーを紹介するために、ウェブサイトが他のサイトにリンクする際により慎重になっていることを示唆していると考えられます。

306. *https://almanac.httparchive.org/ja/2019/seo#linking*

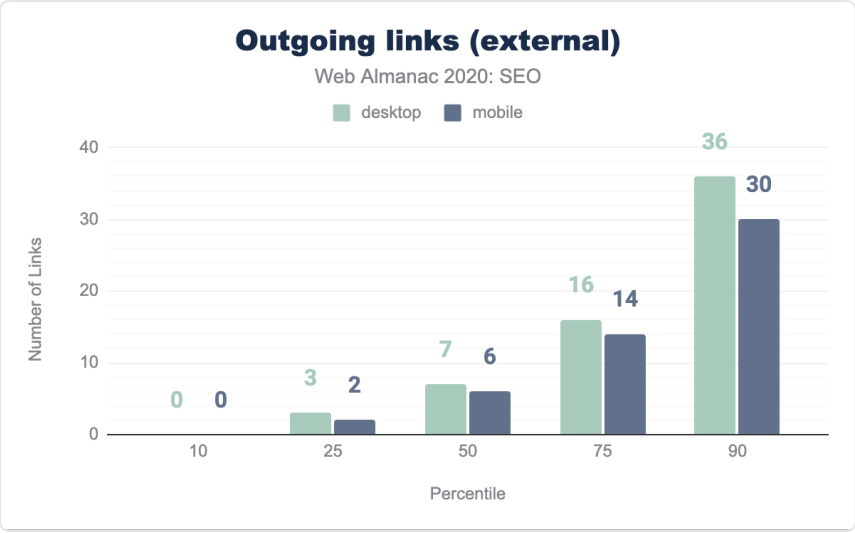


図7.21. ページごとの外部への発信リンク数の分布。

テキストリンクとイメージリンク

ウェブページの中央値では、デスクトップページの9.80%、モバイルページの9.82%で画像をアンカーテキストとして使用してリンクを張っています。これらのリンクは、キーワードに関連したアンカーテキストを実装する機会を失ったことを意味します。これが重大な問題となるのは、ページの90%に達したときです。

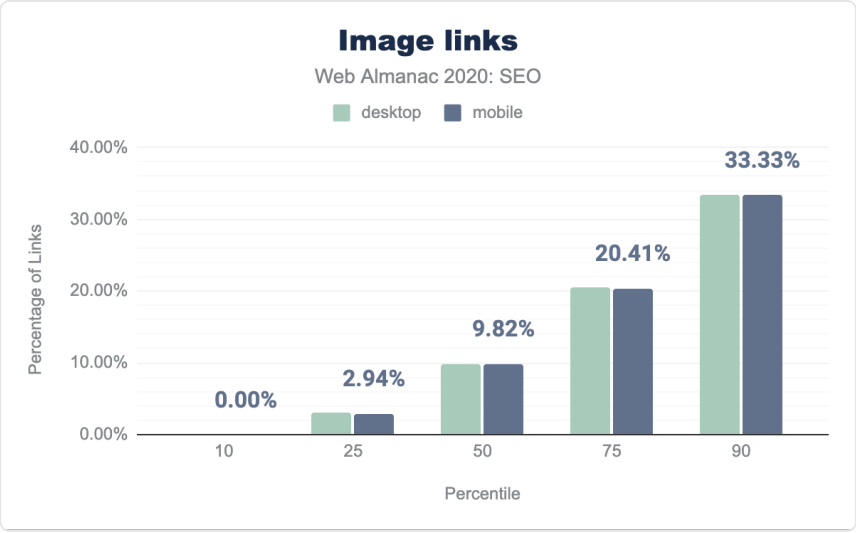


図7.22. ページごとの画像を含むリンクの割合の分布。

モバイルとデスクトップのリンク

Googleがモバイルファーストインデックスだけでなく、モバイルオンリーインデックスにも力を入れるようになると、モバイルとデスクトップのリンクに格差が生じサイトに悪影響を及ぼすことになります。このことは、中央値のウェブページのリンク数がデスクトップの68に対してモバイルでは62であることからもわかります。

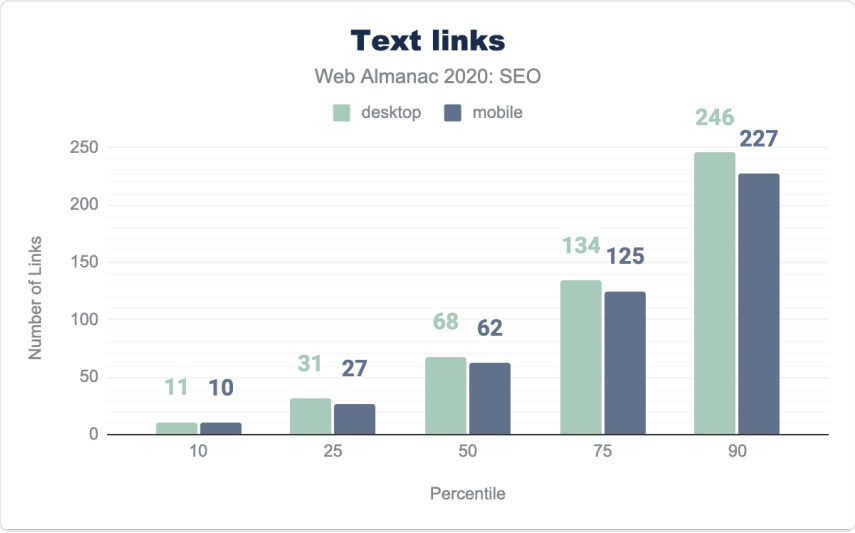


図7.23. ページごとのテキストリンク数の分布。

`rel=nofollow`、`ugc`、`sponsored`属性の使用法

2019年9月、Googleが属性を導入しました[307]これにより、パブリッシャーはリンクをスポンサー付きコンテンツまたはユーザー生成コンテンツとして分類できます。 これらの属性は、以前は2005年に導入された`rel=nofollow`へ追加されます。 新しい属性`rel=ugc`と`rel=sposnsored`は、これらのリンクが特定のWebページに表示される理由を明確化または限定することを目的としています。

ページを調査したところ、デスクトップでは28.58%、モバイルでは30.74%のページに`rel=nofollow`属性が含まれていました。しかし、`rel=ugc`と`rel=sponsored`の採用率は非常に低く、0.3%未満のページ（約20,000ページ）にしか採用されていません。これらの属性は、パブリッシャーにとって`rel=nofollow`よりも付加価値がないため、今後も採用率が低くなると予想するのが妥当でしょう。

307. *https://webmasters.googleblog.com/2019/09/evolving-nofollow-new-ways-to-identify.html*

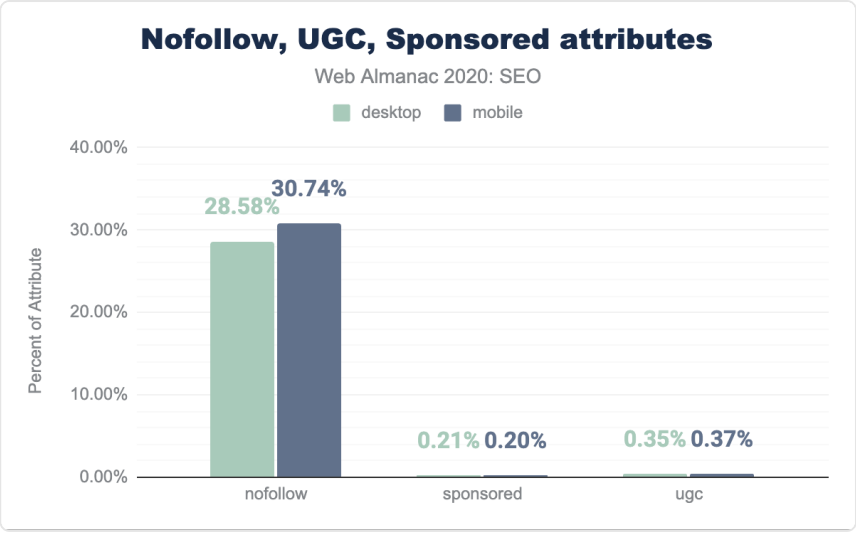


図7.24. `rel=nofollow`、`rel=ugc`、`rel=sponsored` *の属性を持つページの割合です。*

アドバンスド

このセクションではサイトのクローラビリティやインデクサビリティには直接影響しないものの、検索エンジンがランキングシグナルとして使用していることが確認されていたり、ウェブサイトが検索機能を利用する能力に影響を与えるようなウェブの構成や要素に関する最適化の機会を探ります。

モバイル対応

ウェブを閲覧、検索することにモバイルデバイスの利用が増えていることから、検索エンジンは数年前からモバイルフレンドリーをランキング要素として考慮しています[308]。

また先に述べたように、2016年から[309]Googleはモバイルファーストインデックスに移行しており、クロール、インデックス、ランキングされるコンテンツは、モバイルユーザーやスマートフォンのGooglebot[310]がアクセスできるものであることを意味しています。

さらに、2019年7月以降[311]、Googleはすべての新しいウェブサイトにモバイルファーストインデックスを使用しており、3月初めには、検索結果に表示されるページの70%がすでに移

308. https://developers.google.com/search/blog/2015/02/finding-more-mobile-friendly-search
309. https://developers.google.com/search/blog/2016/11/mobile-first-indexing
310. https://developers.google.com/search/docs/advanced/crawling/googlebot?hl=en
311. https://developers.google.com/search/blog/2019/05/mobile-first-indexing-by-default-for

行していることを発表しました[312]。現在、Googleは2021年3月に全面的にモバイルファーストインデックスに切り替えると予想されています[313]。

モバイルフレンドリーは、良好な検索体験を提供し、結果的に検索結果での順位を上げたいと考えているWebサイトにとって基本的なことです。

モバイルフレンドリーなウェブサイトを実現するには、レスポンシブウェブデザインを採用する方法、ダイナミックサービングを使用する方法、モバイルウェブ版を別途用意する方法など、さまざまな方法があります。しかしモバイル版を別に用意することは、Googleからは推奨されておらず、レスポンシブウェブデザインが推奨されています。

ビューポートのメタタグ

ブラウザのビューポートとは、ページコンテンツの可視領域のことで、使用するデバイスに応じて変化します。`<meta name="viewport">`タグ（またはviewport metaタグ）を使用すると、ブラウザにビューポートの幅とスケーリングを指定でき、異なるデバイス間で正しいサイズを表示できます。レスポンシブWebサイトでは、viewport metaタグとCSSメディアクエリを使用して、モバイルフレンドリーな表示を実現しています。

モバイルページの42.98%、デスクトップページの43.2%が、`content=initial-scale=1,width=device-width`属性のviewport metaタグを記述しています。しかしモバイルページの10.84%、デスクトップページの16.18%は、このタグをまったく含んでおらず、まだモバイルフレンドリーではないことがうかがえます。

ビューポート	モバイル	デスクトップ
initial-scale=1,width=device-width	42.98%	43.20%
not-set	10.84%	16.18%
initial-scale=1,maximum-scale=1,width=device-width	5.88%	5.72%
initial-scale=1,maximum-scale=1,user-scalable=no,width=device-width	5.56%	4.81%
initial-scale=1,maximum-scale=1,user-scalable=0,width=device-width	3.93%	3.73%

図7.25. 各ビューポートメタタグの`content`属性値を持つページの割合。

312. https://webmasters.googleblog.com/2020/03/announcing-mobile-first-indexing-for.html
313. https://webmasters.googleblog.com/2020/07/prepare-for-mobile-first-indexing-with.html

CSSメディアクエリ

メディアクエリは、レスポンシブWebデザインの基本的な役割を果たすCSS3の機能です。メディアクエリは、ブラウザやデバイスが特定のルールに合致した場合にのみスタイリングを適用する条件を指定できます。これにより、同じHTMLでもビューポートサイズに応じて異なるレイアウトを作成できます。

デスクトップページの80.29%、モバイルページの82.92%が`height`、`width`、`aspect-ratio`のいずれかのCSS機能を使用しており、高い割合でレスポンシブ機能を搭載していることがわかりました。もっともよく使われている機能を以下の表に示します。

機能	*モバイル*	*デスクトップ*
max-width	*78.98%*	*78.33%*
min-width	*75.04%*	*73.75%*
-webkit-min-device-pixel-ratio	*44.63%*	*38.78%*
orientation	*33.48%*	*33.49%*
max-device-width	*26.23%*	*28.15%*

図7.26. 各メディアクエリ機能が含まれているページの割合。

`Vary: User-Agent`

動的な配信構成（使用するデバイスに基づいて同じページの異なるHTMLを表示する構成）でモバイルフレンドリーなWebサイトを実装する場合は、検索エンジンがを検出できるように、`Vary : User-Agent` HTTPヘッダーを追加することをオススメします。 Webサイトをクロールするときのモバイルコンテンツ。これは、応答がユーザーエージェントによって異なることを通知するためです。

モバイルページの13.48%とデスクトップページの12.6%のみが、`Vary : User-Agent`ヘッダーを指定していることがわかりました。

```
<link rel="alternate" media="only screen and (max-width: 640px)">
```

モバイル版を持つデスクトップのウェブサイトは、HTMLの`head`内にこのタグを使ってリンクを張ることが推奨されます。分析したデスクトップページのうち、指定した`media`属

性値でこのタグを含んでいるのは0.64%に過ぎませんでした。

Webパフォーマンス

ウェブサイトの読み込みが速いことは、ユーザーに優れた検索体験を提供するための基本です。その重要性から、長年にわたり検索エンジンのランキング要素として考慮されてきました。Googleは当初、2010年にランキング要因としてサイトスピードを使用することを発表し[314]、その後2018年にモバイル検索[315]でも同様のことを行いました。

2020年11月に発表されたように、Core Web Vitals[316]と呼ばれる3つのパフォーマンス指標は、2021年5月に「ページ体験」シグナルの一部としてランキング要因となる予定です。Core Web Vitalsの構成は以下の通りです。

Largest Contentful Paint[317] (LCP)

- 表現：ユーザーが感じるロードエクスペリエンス
- 測定：ページロードタイムラインにおいて、ビューポート内にページ最大の画像またはテキストブロックが表示される時点
- 目標：2.5秒未満

First Input Delay[318] (FID)

- 表現：ユーザーの入力に対するレスポンスの良さ
- 計測：ユーザーが最初にページへアクセスしてから、そのアクセスに対応するイベントハンドラーの処理をブラウザが実際に開始できるまでの時間
- 目標：300ミリ秒未満

Cumulative Layout Shift[319] (CLS)

- 表現：視覚的安定性
- 計測方法：シフトしたビューポートの割合に近似した *レイアウトシフトスコア* の数の合計値
- 目標：<0.10

314. *https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html*
315. *https://webmasters.googleblog.com/2018/01/using-page-speed-in-mobile-search.html*
316. *https://webmasters.googleblog.com/2020/05/evaluating-page-experience.html*
317. ***https://web.dev/i18n/ja/lcp/***
318. ***https://web.dev/i18n/ja/fid/***
319. ***https://web.dev/i18n/ja/cls/***

デバイスごとのCore Web Vitals体験

モバイルデバイスを利用するユーザーが増えているにもかかわらず、デスクトップは引き続きユーザーにとってよりパフォーマンスの高いプラットフォームとなっています。Core Web Vitalsの評価では、デスクトップ版では33.13%が「良い」と評価されたのに対し、モバイル版では19.96%しか評価されませんでした。

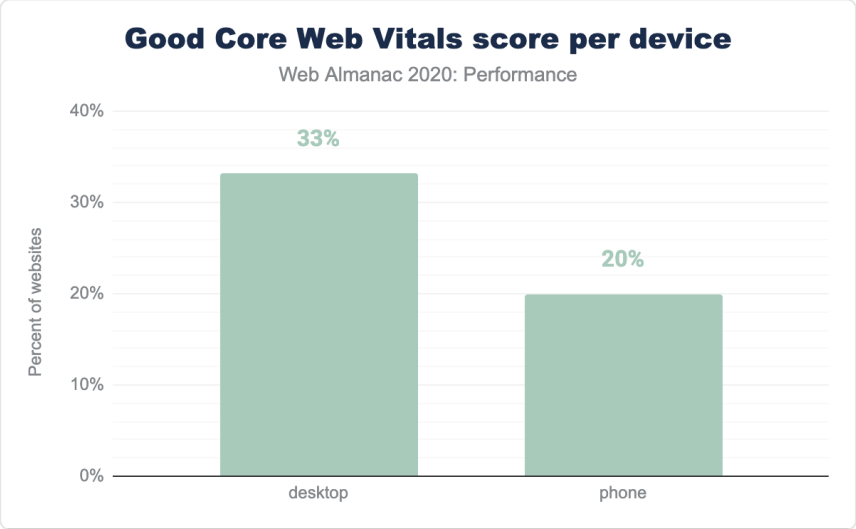


図7.27. デバイスごとのCore Web Vitals評価に合格したウェブサイトの割合。

国ごとのCore Web Vitalsの経験

ユーザーの物理的な位置は、その地域で利用可能な通信インフラ、ネットワーク帯域幅の容量、データのコストなどにより、独自の負荷条件が生じるため、パフォーマンスの認識に影響を与えます。

米国にいるユーザーは、合格点を獲得しているサイトは32%にすぎないにもかかわらず、*Good* Core WebVitalsエクスペリエンスを備えたWebサイトの絶対数が最大でした。 韓国は、*Good* Core Web Vitalエクスペリエンスの割合が52%ともっとも高かった。 各国が要求するウェブサイト全体の相対的な部分は注目に値します。 米国のユーザーは、韓国のユーザーが生成した発信元リクエストの合計の8倍を生成しました。

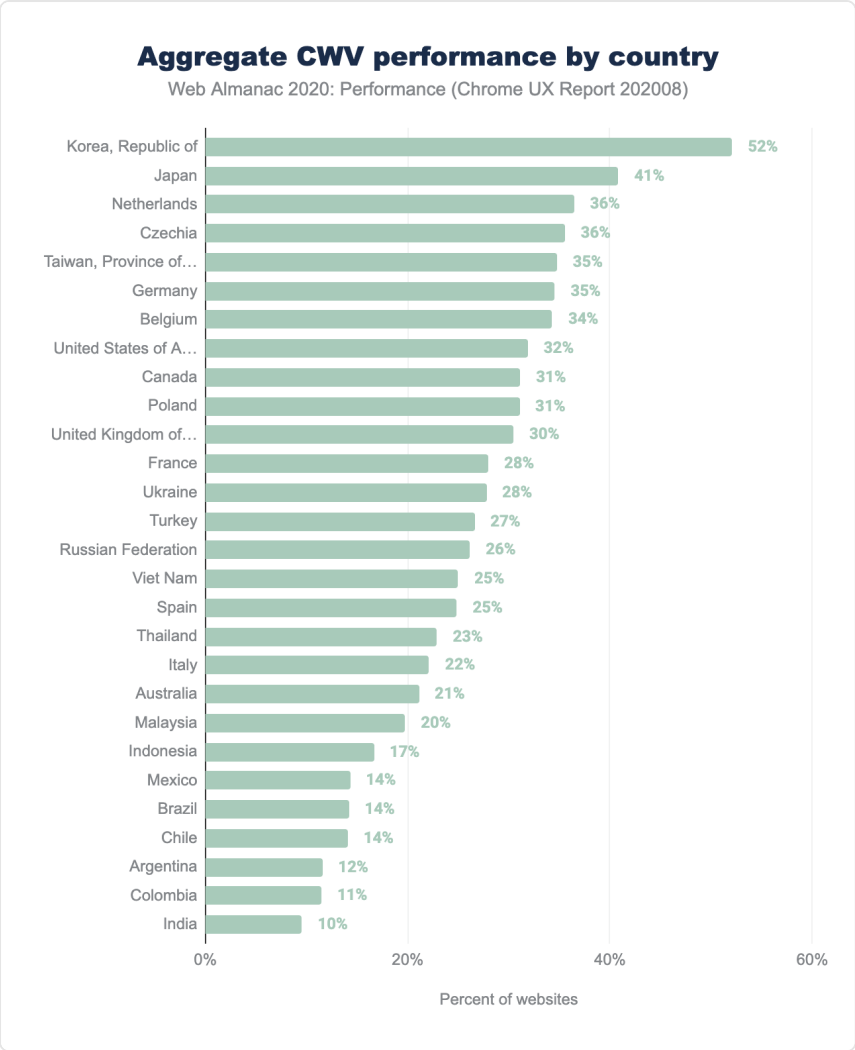


図7.28. 国別のCore Web Vitals評価に合格したウェブサイトの割合。

Core Web Vitalsのパフォーマンスについて、有効な接続タイプ別のディメンションや特定のメトリクスによる追加分析は、パフォーマンスの章でご覧いただけます。

国際化

国際化とは多言語・多国語対応のウェブサイトが、異なる言語や国のバージョンを検索エンジンに通知し、それぞれのケースでユーザーに表示する関連ページを特定しターゲティング

の問題を回避するために使用できる設定を指します。

今回分析した2つの国際的な設定は、`content-language`というメタタグと`hreflang`という属性で、これを使って各ページの言語とコンテンツを指定できます。さらに、`hreflang`アノテーションでは、各ページの代替言語や国別バージョンを指定できます。

Google[320]やYandex[321]のような検索エンジンはページの言語や国のターゲットを決定するシグナルとして`hreflang`属性を使用しており、GoogleはHTMLのlangや`content-language`メタタグを使用していませんが、後者の最後のタグはBingが使用しています。

`hreflang`

デスクトップページの8.1%、モバイルページの7.48%が`hreflang`属性を使用しており、低いと思われるかもしれませんが、これは多言語や複数の国のウェブサイトでのみ使用されているので当然です。

その結果、HTTPヘッダーで`hreflang`を実装しているのは、デスクトップページは0.09%、モバイルページでは0.07%にすぎず、ほとんどがHTMLの`head`による実装に依存していることがわかりました。

また、hreflangアノテーションのレンダリングにJavaScriptを使用しているページがあることも確認しました。デスクトップおよびモバイルページの0.12%が、生のHTMLではなくレンダリングされたものに`hreflang`を表示していました。

言語と国の値の観点から、HTMLヘッドを介した実装を分析したところ、英語（`en`）がもっともよく使われており、モバイルページの4.11%、デスクトップページの4.64%にこの値が含まれていることがわかりました。英語に次いで多く使われている値は、`x-default`（対象外の言語や国のユーザー向けに *default* や *fallback* のバージョンを定義する際に使用）で、モバイルページの2.07%、デスクトップページの2.2%がこの値を含んでいます。

3番目、4番目、5番目に人気があるのは、ドイツ語（`de`）、フランス語（`fr`）、スペイン語（`es`）で、イタリア語（`it`）、アメリカ英語（`en-us`）と続きますが下の表にあるように残りの値はHTMLの`head`で実装されています。

320. *https://support.google.com/webmasters/answer/189077?hl=en*
321. *https://yandex.com/support/webmaster/yandex-indexing/locale-pages.html*

値	モバイル	デスクトップ
en	4.11%	4.64%
x-default	2.07%	2.20%
de	1.76%	1.88%
fr	1.74%	1.87%
es	1.74%	1.84%
it	1.27%	1.33%
en-us	1.15%	1.31%
ru	1.12%	1.13%
en-gb	0.87%	0.98%
pt	0.87%	0.87%
nl	0.83%	0.94%
ja	0.73%	0.81%
pl	0.72%	0.75%
de-de	0.69%	0.78%
tr	0.69%	0.66%

図7.29. HTMLの `head` に上位の `hreflang` の値を含んでいるページの割合。

また、HTTPヘッダーで実装されている `hreflang` 言語と国の値の上位には、若干の違いが見られました。ここでも英語（`en`）がもっとも多く、フランス語（`fr`）、ドイツ語（`de`）、スペイン語（`es`）、オランダ語（`nl`）が続きました。

値	モバイル	デスクトップ
en	0.05%	0.06%
fr	0.02%	0.02%
de	0.01%	0.02%
es	0.01%	0.01%
nl	0.01%	0.01%

図7.30. HTTPヘッダーに上位の `hreflang` 値が含まれるページの割合。

`Content-Language`

`content-language` の使用状況と値を分析したところ、HTMLの `head` にmetaタグとして実装しているか、HTTPヘッダーに実装しているかにかかわらず、モバイルページの8.5%、デスクトップページの9.05%しかHTTPヘッダーで指定していないことがわかりました。また、HTMLの `head` 内の `content-language` タグで言語や国を指定しているWebサイトはさらに少なく、モバイルページの3.63%、デスクトップページの3.59%しかmetaタグを使用していませんでした。

言語と国の値の観点から見ると、 `content-language` メタタグやHTTPヘッダーで指定される値は、英語（ `en` ）と米国英語（ `en-us` ）がもっとも多くなっています。

英語（ `en` ）の場合、デスクトップページの4.34%、モバイルページの3.69%がHTTPヘッダーで英語を指定しており、デスクトップページの0.55%、モバイルページの0.48%がHTMLの `head` 内の `content-language` メタタグで指定していることがわかりました。

2番目に多い値であるEnglish for US（ `en-us` ）については、HTTPヘッダーで指定しているのはモバイルページの1.77%、デスクトップページの1.7%、HTMLで指定しているのはモバイルページの0.3%、デスクトップページの0.36%であることがわかりました。

もっとも人気のある言語と国の値の残りの部分は、以下の表で見ることができます。

値	モバイル	デスクトップ
en	3.69%	4.34%
en-us	1.77%	1.70%
de	0.50%	0.44%
es	0.34%	0.33%
fr	0.31%	0.34%
ru	0.18%	0.16%
pt-br	0.15%	0.16%
nl	0.13%	0.15%
it	0.13%	0.13%
ja	0.08%	0.10%

図7.31. HTTPヘッダーで上位の `content-language` の値を使用しているページの割合。

値	モバイル	デスクトップ
en	0.48%	0.55%
en-us	0.30%	0.36%
pt-br	0.24%	0.24%
ja	0.19%	0.26%
fr	0.18%	0.19%
tr	0.17%	0.13%
es	0.16%	0.15%
de	0.15%	0.11%
cs	0.12%	0.12%
pl	0.11%	0.09%

図7.32. HTMLメタタグで上位の `content-language` の値を使用しているページの割合。

セキュリティ

Googleは、あらゆる面でセキュリティにとくに力を入れています。検索エンジンは、不審な活動を行っているサイトやハッキングされたサイトのリストを管理しています。Search Consoleはこれらの問題を表面化させ、Chromeユーザーはこれらの問題があるサイトへアクセスする前に警告を表示します。さらに、Googleは、HTTPS[322](Hypertext Transfer Protocol Secure)で提供されるページに対して、algorithmic boost[323]を提供しています。このトピックに関するより詳細な分析については、セキュリティの章をご覧ください。

HTTPSの使用

デスクトップページの77.44%、モバイルページの73.22%がHTTPSを採用していることがわかりました。これは昨年に比べて10.38%増加しています。ここで重要なのは、ブラウザがHTTPSを積極的に推進するようになったことで、HTTPSなしでページを閲覧すると安全ではないというシグナルを発するようになったことです。また、HTTPSは現在、HTTP/2やHTTP/3（HTTP over QUICとも呼ばれる）のようなより高性能なプロトコルを活用するための要件となっています。これらのプロトコルの状況については、HTTP/2の章で詳しく説明しています。

これらのことが、前年比での採用率の上昇につながっていると思われます。

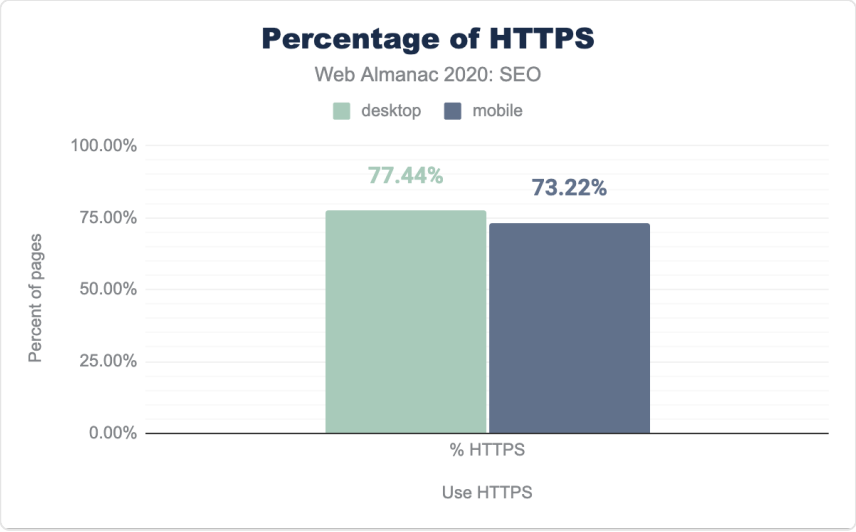


図7.33. HTTPSで提供されているページの割合。

322. https://developers.google.com/search/blog/2014/08/https-as-ranking-signal
323. https://developers.google.com/search/docs/advanced/security/https

AMP

AMP[324]（旧称：Accelerated Mobile Pages）は、とくにモバイル端末でのページの読み込みを速くするための方法として、2015年にGoogleが発表したオープンソースのHTMLフレームワークです。AMPは、既存のウェブページの代替バージョンとして実装することも、AMPフレームワークを使って一から開発することもできます。

ページにAMPバージョンが用意されていると、Googleがモバイル検索結果にAMPロゴとともに表示します。

また、AMPの使用状況はGoogle（または他の検索エンジン）のランキング要素ではありませんが、ウェブスピードはランキング要素であることにも注意が必要です。

さらに、この記事を書いている時点ではニュース関連の出版物にとって重要な機能であるGoogleのモバイル検索結果のトップストーリーズのカルーセルに掲載されるためには、AMPが必須条件となっています。ただしこれは2021年5月に変更され、AMP以外のコンテンツであっても、Googleニュースのコンテンツポリシー[325]を満たし、今年11月にGoogleが発表した優れた[326]ページエクスペリエンス[327]を提供するものであれば対象となります。

AMPベースではないページの代替バージョンとしてのAMPの使用状況を確認したところ、モバイルWebページの0.69%、デスクトップWebページの0.81%が、AMPバージョンを指す`amphtml`タグを含んでいることがわかりました。まだまだ採用率は低いですが、モバイルページの0.62%しかAMP版へのリンクを含んでいなかった昨年の調査結果[328]よりも若干改善されています。

一方、ウェブサイト開発のフレームワークとしてのAMPの利用状況を評価したところ、AMPベースのページを示す絵文字属性`<html amp>`や`<html ⚡>`を指定しているのは、モバイルページは0.18%、デスクトップページでは0.07%に過ぎませんでした。

シングルページのアプリケーション

シングル・ページ・アプリケーション（SPA）は、ユーザーの要求に応じてページ上のコンテンツが更新されても、ブラウザは単一のページ・ロードを保持し更新できます。JavaScriptフレームワーク、AJAX、WebSocketなどの複数の技術が、後続のページロードを軽量化するために使用されています。

これらのフレームワークは特別なSEO上の配慮が必要でしたが、Googleは積極的なキャッシング戦略により、クライアントサイドレンダリングによる問題を軽減するよう努めてきまし

324. https://amp.dev/
325. https://support.google.com/news/publisher-center/answer/6204050
326. https://developers.google.com/search/docs/guides/page-experience
327. https://developers.google.com/search/blog/2020/11/timing-for-page-experience
328. https://almanac.httparchive.org/ja/2019/seo#amp

た。Google Webmaster's 2019 conference[329]のビデオの中でソフトウェアエンジニアのErik Hendriksは、Googleはもはや `Cache-Control` ヘッダーに依存せず、代わりに `ETag` または `Last-Modified` ヘッダーを探してファイルのコンテンツが変更されたかどうかを確認していることを共有しました。

SPAはキャッシュを細かく制御するために、Fetch API[330]を利用する必要があります。このAPIでは、特定のキャッシュオーバーライドを設定した `Request` オブジェクトを渡すことができ、必要な `If-Modified` や `ETag` ヘッダーを設定するのに使用できます。

検索エンジンやそのWebクローラーにとって、発見できないリソースはやはり最大の関心事です。検索クローラーは、リンクされたページを見つけるために、`<a>` タグの `href` 属性を探します。これらがないと、ページは内部リンクのない孤立したものとみなされます。調査対象となったデスクトップページの5.59%、モバイルレンダリングページの6.04%に内部リンクが含まれていませんでした。これは、そのページがJavaScriptフレームワークのSPAの一部であり、内部リンクを発見するために必要な有効な `href` 属性を持つ `<a>` タグがないことを示しています。

SPAに使われている人気のJavaScriptフレームワークのリンクの発見可能性は、2020年に前年比[331]で劇的に増加しました。2019年には、Reactサイトのモバイルナビゲーションリンクの13.08%が非推奨のハッシュURLを使用していました。2020年は、テストされたReactのリンクのうち、ハッシュ化されていたのは6.12%だけでした。

同様に、Vue.jsは、前年の8.15%から3.45%に低下しました。Angularは、2019年にもっともクロールできないハッシュ化されたモバイルナビゲーションリンクの使用率が低く、モバイルページのわずか2.37%でした。2020年は、その数字が0.21%に激減しました。

結論

発見されたことと昨年の結論[332]と一致しており、ほとんどのサイトがクローラブルでインデックス可能なデスクトップページとモバイルページを持ち、SEO関連の基本的な設定を活用している。

SPAに使用されている主要なJavaScriptフレームワークのリンク発見率が、2019年に比べて飛躍的に向上したことを強調しておきます。モバイルナビゲーションのリンクをハッシュ化したURLでテストしたところ、Reactを使用したサイトからはクロールされないリンクのインスタンスが-53%、Vue.jsを使用したサイトからは-58%、AngularのSPAからは-91%減少しました。

329. *https://youtu.be/rq8sFkl0KnI*
330. *https://developer.mozilla.org/docs/Web/API/Fetch_API*
331. *https://almanac.httparchive.org/ja/2019/seo#spa-crawlability*
332. *https://almanac.httparchive.org/ja/2019/seo#conclusion*

さらに、多くの分析分野において、昨年の調査結果から若干の改善が見られたことも確認しました。

- **`robots.txt`**: 昨年は72.16%のモバイルサイトが有効な`robots.txt`を持っていましたが、今年は74.91%でした。
- **canonicalタグ**: 昨年は、モバイルページの48.34%がcanonicalタグを使用していたのに対し、今年は53.61%でした。
- **`title`タグ**: 今年は、デスクトップページの98.75%が1つを備えているのに対し、モバイルページの98.7%もそれを含んでいることがわかりました。 昨年の章では、モバイルページの97.1%に`title`タグが付いていることがわかりました。
- **`meta`記述**: 今年の調査では、デスクトップページの68.62%、モバイルページの68.22%に`meta`記述があり、モバイルページの64.02%にあった昨年よりも改善されました。
- **構造化データ**: 本来、レビューはホームページとは関係ないはずなのに、データによると`AggregateRating`はモバイルで23.9%、デスクトップでは23.7%上昇しています。
- **HTTPSの使用**: デスクトップページの77.44%、モバイルページの73.22%がHTTPSを採用している。これは、昨年より10.38%増加しています。

しかし、この1年ですべてが改善されたわけではありません。デスクトップページの中央値には61本の内部リンクが含まれているのに対し、モバイルページの中央値には54本の内部リンクが含まれています。これは昨年[333]と比べてそれぞれ12.8%と10%減少しており、サイトがページを介したクローラビリティとリンクエクイティの流れを改善する能力を最大限に発揮できていないことを示唆しています。

また、多くの重要なSEO関連の領域と構成にわたって改善の重要な機会がまだあることに注意することも重要です。 モバイルデバイスの使用が増え、Googleがモバイルファーストインデックスに移行したにもかかわらず、次のようになりました。

- モバイルページの10.84%、デスクトップページの16.18%は、`viewport`タグをまったく含んでおらず、まだモバイルフレンドリーではないことを示唆しています。
- モバイルページとデスクトップページでは、些細な違いが見られました。たとえばモバイルとデスクトップのリンクの違いは、中央値のウェブページでモバイル

333. *https://almanac.httparchive.org/ja/2019/seo#linking*

のリンクが62であるのに対し、デスクトップのリンクが68であることで示されています。

- ウェブサイトの33.13%がデスクトップで *Good* Core Web Vitalsを獲得しましたが、モバイル版の19.96%のみがCore Web Vitalsの評価に合格しました。これは、デスクトップが引き続きユーザーにとってよりパフォーマンスの高いプラットフォームであることを示しています。

これらの調査結果は、Googleが2021年3月にモバイルファーストインデックス[334]への移行を完了する際に、サイトに悪影響を及ぼす可能性があります。

また、レンダリングされたHTMLとレンダリングされていないHTMLにも格差が見られました。たとえば、モバイルページの中央値では、レンダリング時に表示される文字数が生のHTMLに比べて11.5%多く、クライアントサイドのJavaScriptに依存してコンテンツを表示していることがわかります。

検索クローラーは、リンクされたページを見つけるために、`<a href>`タグを探します。これがないと、ページは内部リンクのない孤立したものとみなされます。デスクトップページの5.59%、モバイルレンダリングページの6.04%に内部リンクがありませんでした。

これらの調査結果は、検索エンジンがウェブサイトを効果的にクロール、インデックス、ランキングする能力を継続的に進化させていることを示唆しており、もっとも重要なSEOの設定も現在ではよりよく考慮されています。

しかし、ウェブ上の多くのサイトが、重要な検索上の可視性と成長の機会を逃しているのが現状です。このことは、組織全体でのSEOの伝道とベストプラクティスの採用が依然として必要であることを示しています。

334. https://webmasters.googleblog.com/2020/07/prepare-for-mobile-first-indexing-with.html

著者

Aleyda Solis

@aleyda　aleyda　https://www.aleydasolis.com/en/

SEOコンサルタント、作家、講演者、起業家。Orainti[335]（SaaSからマーケットプレイスまで、トップレベルのウェブプロパティやブランドを扱うブティック型SEOコンサルタント）の創設者であり、Remoters.net[336]（リモートワークを促進するためのリモートジョブ、ツール、イベント、ガイドなどを提供する、無料のリモートワークハブ）の共同創設者でもあります。また、ブログ[337]、#SEOFOMO ニュースレター[338]、Crawling Mondays[339] ビデオとポッドキャストシリーズ、Twitter[340]などを通じて、SEOについての情報を発信しています。

Michael King

@IPullRank　ipullrank

アーティストであり、テクノロジストでもあるマイケル・キングは、企業向けデジタルマーケティングエージェンシーであるiPullRank[341]の創設者であり、自然言語生成アプリCopyScience[342]の創設者でもあります。独立系のヒップホップミュージシャンとしての経歴を活かし、魅力的なコンテンツを作成し、受賞歴のあるダイナミックな講演者として、世界中のテクノロジーやマーケティングに関するカンファレンスやブログで活躍しています。また、Twitter[343]では、マイクがゴミの話をしているところを見ることができます。

335. https://www.orainti.com/
336. https://remoters.net/
337. https://www.aleydasolis.com/en/blog/
338. https://www.aleydasolis.com/en/seo-tips/
339. https://www.aleydasolis.com/en/crawling-mondays-videos/
340. https://x.com/aleyda
341. https://ipullrank.com
342. https://www.copyscience.io
343. https://x.com/ipullrank

Jamie Indigo

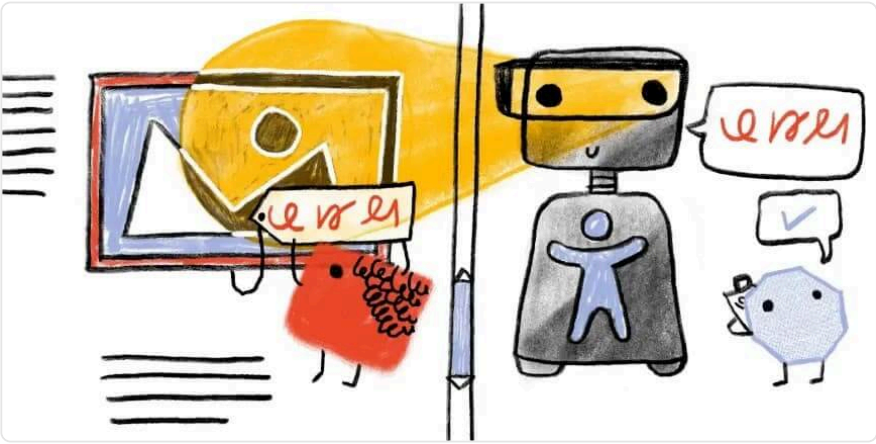
@Jammer_Volts @not-a-robot.com fellowhuman1101 jamie-indigo

https://not-a-robot.com

100%人間であり、完全にロボットではないJamie Indigoは、人間が有益な情報にアクセスし、企業が価値あるデジタル体験を提供できるよう、テクノロジーを解きほぐします。彼女はNot a Robot[344]を設立し、技術的なSEOの人間的な側面に焦点を当て、技術や検索業界における倫理やインクルージョンを含めたコンサルティングを行っています。 彼女は、Twitter[345]で公の場で学んでいるのを見つけることができます。

344. https://not-a-robot.com
345. https://x.com/Jammer_Volts

部 II 章 8
アクセシビリティ



Olu Niyi-Awosusi と Alex Tait によって書かれた。
Adrian Roselli、 Eric Bailey と David Fox によってレビュ。
David Fox による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

2020年には、これまで以上にデジタルスペースの包括化とアクセス容易化が急務となっています。現在進行中のパンデミックにより、人々が対面でサービスにアクセスすることがさらに困難になり、産業全体がオンラインに移行しているため、障害者は不釣り合いな影響を受けています。さらに、パンデミックの長期的な影響[346]により、障害者の数は増加しています。

ウェブアクセシビリティとは、機能と情報の同等性を達成し、障害者がインターフェイスのあらゆる側面に完全にアクセスできるようにすることです。デジタル製品やウェブサイトは、誰もが使えるものでなければ、完全なものではありません。特定の障害者を排除している場合、これは差別であり、罰金や訴訟の根拠となる可能性があります。

346. https://www.cdc.gov/coronavirus/2019-ncov/long-term-effects.html

Web Content Accessibility Guidelines[347]、またはWCAGとは、インターネットを利用するすべてのウェブサイトやアプリケーションで満たす必要がある国際的に認められた基準のセットです。これらは法律ではありませんが、多くの法律がその根拠としてWCAGを指摘しています[348]。

これらのガイドラインは何年にもわたって何度もリリースされており、現在の標準はWCAG 2.1であり、WCAG 2.2は現在作業草案[349]として審査されています。いくつかの地域の法律ではWCAG 2.0が要件とされていますが、Adrian Roselli[350]が彼の記事WCAG 2.1 is the Current Standard, Not WCAG 2.0 - and WCAG 2.2 is Coming[351]で取り上げているように、私たちはWCAG 2.1の基準を満たし、WCAG 2.2で来る新しい基準も考慮しなければなりません。

2020年に向けて、これまで以上に露出が増えている危険な傾向は、「アクセシビリティ・オーバーレイ」の使用です。これらのウィジェットは一歩進んだアクセシビリティの遵守を約束しているが、多くの場合、新たな障壁を導入し、障害のあるユーザーの体験をかなり困難なものにしている。デジタル関係者は、使い勝手の良いインターフェイスの設計と実装に責任を持つことが重要であり、安易な修正でこのプロセスを台無しにしようとしないことが重要です。詳細については、Lainey Feingoldの記事、Honor the ADA: Avoid Web Accessibility Quick Fix Overlays[352]を参照してください。

悲しいことに、私たちやWebAIM Million[353]のような分析を行っている他のチームは、年々これらの指標にほとんど改善が見られず場合によっては全く改善が見られないことを発見しています。すべてのLighthouse Accessibility[354]監査データのサイト全体のスコアの中央値は、2019年の73%から2020年には80%に上昇しました。この7%の上昇は、正しい方向へのシフトを表していると期待しています。しかし、これらは自動化されたチェックであり、開発者がルールエンジンを裏切る良い仕事をしていることを意味する可能性があるので慎重に楽観視しています。

当社の分析は、自動化されたメトリクスのみに基づいています。自動化されたテストは、インターフェイスに存在しうるアクセシビリティの障壁のほんの一部を捉えているに過ぎないことを覚えておくことが重要です。アクセシブルなサイトやアプリケーションを実現するためには、手動テストや障害者によるユーザビリティ・テストを含む定性分析が必要です。

私たちの最も興味深い洞察を5つのカテゴリーに分けました。

1. 読みやすさ。
2. ウェブ上のメディア。
3. ページナビゲーションのしやすさ。
4. ウェブ上の支援技術。

347. *https://www.w3.org/WAI/standards-guidelines/wcag/*
348. *https://www.w3.org/WAI/policies/*
349. *https://www.w3.org/WAI/standards-guidelines/wcag/new-in-22/*
350. *https://x.com/aardrian*
351. *https://adrianroselli.com/2020/09/wcag-2-1-is-the-current-standard-not-wcag-2-0-and-wcag-2-2-is-coming.html*
352. *https://www.lflegal.com/2020/08/quick-fix/*
353. *https://webaim.org/projects/million/*
354. *https://web.dev/i18n/ja/lighthouse-accessibility/*

5. フォームコントロールのアクセシビリティ。

私たちはウェブ上での身の引き締まるようなメトリクスと明らかなアクセシビリティの怠慢が満載のこの章が、読者にこの仕事の優先順位をつけ、より包括的で公正なインターネットへと移行していくための習慣を変えるきっかけとなることを願っています。

読みやすさ

コンテンツをできるだけシンプルで読みやすいものにすることは、ウェブアクセシビリティの重要な側面です。ページのコンテンツを読むことができないと、ユーザーはウェブサイトでのタスクを完了できません。ウェブページには色のコントラスト、ページのズームや拡大縮小、言語の識別など、読みやすくしたり、読みにくくしたりする多くの側面があります。

カラーコントラスト

ページのコントラストが高いほど、テキストベースのコンテンツを見やすくなります。コントラストの低いコンテンツを表示することが困難な人には色覚障害のある人、軽度から中等度の視力低下のある人、明るい光の中での画面のまぶしさなど、文脈上コンテンツを表示することが困難な人を含みます。

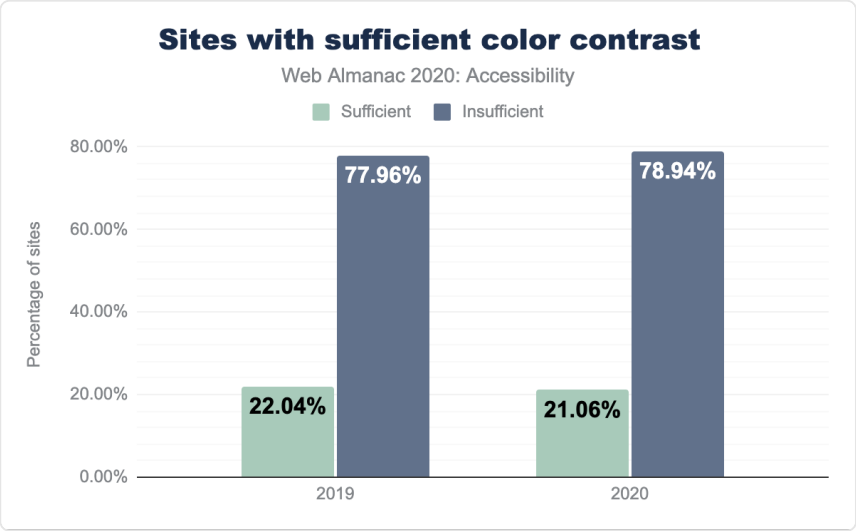


図8.1. 色のコントラストが十分なサイト。

残念ながら、十分なカラーコントラストを持つサイトは21.06%しかありませんでした。こ

れは昨年のすでにひどい22%からの減少です。

ズームと拡大縮小

ユーザーがページやコンテンツを拡大表示できるようにすることが不可欠です。ブラウザの拡大縮小やズーム機能を無効にしようとするテクニックがあります。OSによっては、この有害なパターンを回避するものもありますが、多くはそうでなく回避する必要があるアンチパターンです。

特に弱視の方にはズーム機能が便利です。世界保健機関[355]によると、「世界では10億人が視力障害を抱えている」とされています。

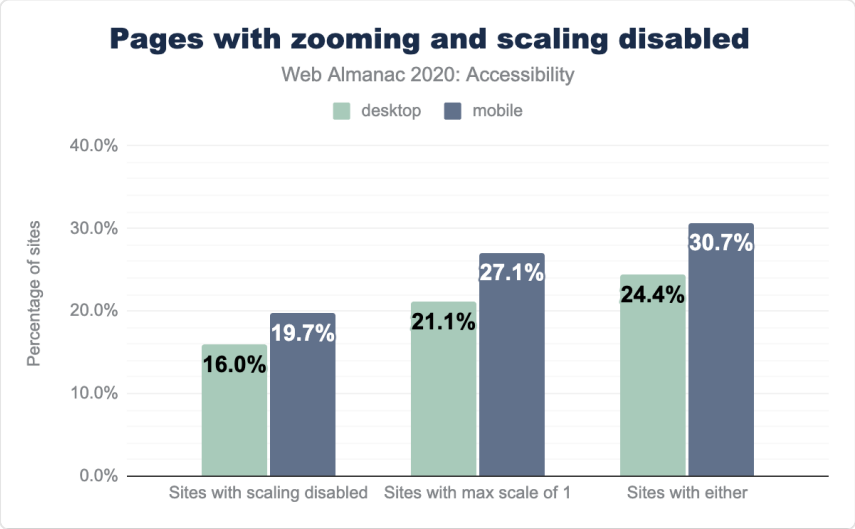


図8.2. ズームやスケーリングが無効になっているサイト

デスクトップページの29.34%とモバイルページの30.66%が、`maximum-scale`を1未満の値に設定するか、`user-scalable`を`0`または`none`に設定することで、スケーリングを無効にしようとしていることがわかった。いくつかのOSでは、HTMLで設定されているズームや拡大縮小の無効化に対応していないものがあります。これに対応しているシステムでは、一部のシステムではページが事実上使えなくなることがあります。ブラウザのズームを無効にするのを避ける理由については、Adrian Roselliの記事Don't Disable Zoom[356] を参照してください。

355. *https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment*
356. *https://adrianroselli.com/2015/10/dont-disable-zoom.html*

言語の識別

77.67%

図8.3. モバイルサイトは有効な `lang` 属性を持っています。

HTMLの `lang` 属性を設定すると、ページを簡単に翻訳でき、スクリーンリーダーのサポートが向上します。 今年のモバイルで有効なHTMLlang属性を持つサイトの割合は77.67%で、 `lang属性` を持つサイトは77.7%にすぎませんでした。

ウェブ上のメディア

メディアはウェブ体験に欠かせないものです。視力のあるユーザーだけでなく、周囲のテキスト情報に豊かな文脈を加えることができます。

画像とそのテキストの選択肢

1995年、HTML 2.0[357]では、 `alt` 属性が導入され、ウェブ制作者が画像内の視覚情報の代替テキストを提供できるようになりました。スクリーンリーダーは、画像の代替テキストを表示することで、視覚的な意味を聴覚的に伝えることができます。さらに、画像が読み込めない場合は、説明の代替テキストが表示されます。

54%

図8.4. Lighthouse監査で「`alt` の文字が入った画像」を通過した携帯サイト

2020年のLighthouse監査データによると、画像に `alt` テキストがある場合のテストに合格したサイトは54%に過ぎません。このテストでは、 `img` 要素に `alt` , `aria-label` , `aria-labelledby` 属性のうち少なくとも1つが存在するかどうかを調べます。ほとんどの場合、 `alt` 属性を使うのが最良の選択です。

357. https://www.w3.org/MarkUp/html-spec/html-spec_5.html#SEC5.10

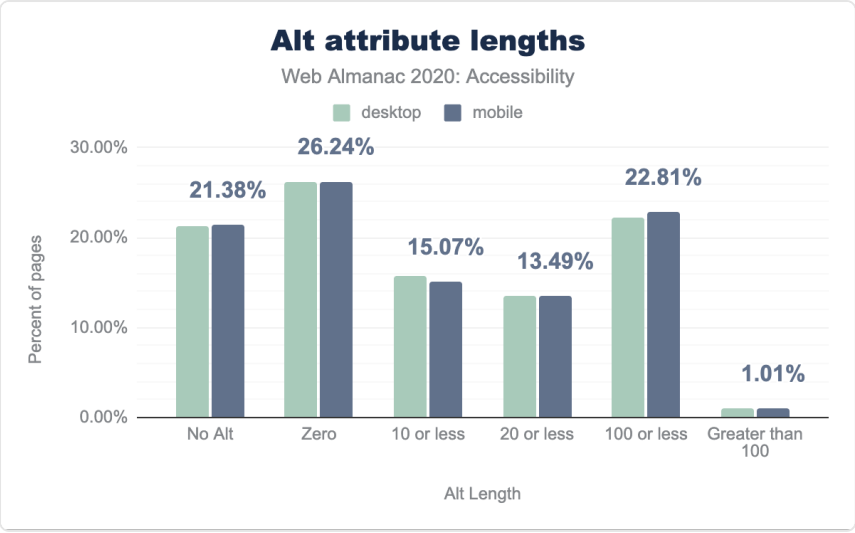


図8.5. alt属性の長さ。

`alt`属性は25年前から存在していますが、デスクトップ画像の21.24%とモバイル画像の21.38%が代替テキストを欠いていることもわかりました。これは選択したアクセシビリティツールを使用してテストする最も簡単な自動チェックの1つであり、ぶら下がりが少なく、比較的簡単に解決できる問題であるはずです。

スクリーン・リーダーのユーザーは、コンテンツの構造、意味論、関係性が発表されるインターフェイスの聴覚的または音響的な体験であるSteve Faulker氏が説明する"aural UI"[358]に耳を傾ける。これは、スクリーンリーダーのユーザーが多くのテキスト情報を消費することを意味します。このため、画像を記述する必要がないかもしれないかどうかを評価することが重要です。これは画像をどのように記述するか、あるいは記述するかを決定するのに役立つW3Cの決定木[359]です。画像が本当に装飾的で、周囲の文脈に何の意味もない場合は、`alt`属性にヌル値、`alt=""`を代入できます。これは、`alt`属性を完全に省略するのではなく、明示的に行うことが重要です。

デスクトップページの26.20%、モバイルページの26.23%が、null/空の値を持つ`alt`属性を含んでいることがわかりました。これは、4分の1以上のウェブサイトが、自動チェックを回避するための手段としてではなく、画像が本当に意味のあるものであるかを考慮して開発されていることを示していると期待しています。

画像を記述する際には、ユーザーがどのような情報を必要としているかを考慮し、冗長性を減らすために追加情報を省略することが不可欠です。たとえば、インターフェイスの新しい

358. *https://developer.paciellogroup.com/blog/2015/10/thus-spoke-html/*
359. *https://www.w3.org/WAI/tutorials/images/decision-tree/*

ステップに移動するアクションを持つ赤い矢印アイコンボタンは「5のステップ3に進む」と表現できます、「赤い矢印png」ではなく。最初の記述はユーザーがコントロールを起動した場合に何を期待するかを伝えますが、2番目の記述は外観を記述しているだけで、不要なファイル拡張子を持っておりどちらも画像の意味とは無関係です。

代替テキストの有無の自動チェックは、このテキストの品質を評価するものではありません。前のセクションで説明したように、このテキストを書く際には、画像の意味を考慮する必要があります。役に立たない一般的なパターンの1つは、画像をファイル拡張子名で記述することです。先ほどの「赤い矢印png」の例では、スクリーンリーダーのユーザーは、画像の形式から有用な情報を得られない可能性が高いです。デスクトップサイト（`alt`属性の少なくとも1つのインスタンスを持つ）の6.8%のサイトでは、その値にファイル拡張子が含まれていることがわかりました。

`alt`テキスト値に明示的に含まれるファイル拡張子の上位5つは、(alt値が空ではない画像を持つサイトの場合) `jpg`, `png`, `ico`, `gif`および`jpeg`です。これは、CMSや他の自動生成された代替テキストメカニズムから来ている可能性が高いです。どのように実装されているかに関わらず、これらのalt属性値が意味のあるものであることが必須です。

ファイル拡張子	デスクトップ	モバイル
jpg	4.08%	3.83%
png	2.99%	2.82%
ico	1.35%	1.26%
gif	0.34%	0.33%
jpeg	0.35%	0.31%

図8.6. ファイル拡張子で終わるAlt属性。

タイトル属性を持つ画像

テキストを表示するツールチップを生成する`title`属性は、支援技術に画像を記述するためのもう1つの信頼できる方法として誤解されることがよくあります。しかし、HTML標準によると

> *“多くのユーザエージェントがこの仕様で要求されているように、アクセス可能な方法で属性を公開していないため、`title`属性に依存することは現在のところ推奨されていません。”*

またツールチップは情報がマウスオーバー時にしか表示されない、情報が支援技術に適切に伝達されない、キーボードのサポートがない、一般的な使い勝手の悪さなど他の多くのアクセシビリティの障壁を導入しています。ツールチップの歴史とその障壁については、Sarah Higley[360]のブログ記事Tooltips in the time of WCAG 2.1[361]でよく説明されています。

すべての`alt`属性の16.95%が`title`属性を含んでいることがわかった。このうち、デスクトップサイトでは73.56%、モバイルサイトでは72.80%が`alt`属性と`title`属性の両方に一致する値を持っていました。

`alt`テキストに関するその他の事実

15,357,625

図8.7. 最も長い既知の`alt`テキストの長さ。

デスクトップとモバイルの`alt`テキストの長さの中央値は18文字です。平均的な英語の単語の長さは4.7文字で、alt属性値の長さの中央値は3〜4文字であることを意味します。画像によっては、簡潔であることが有益な場合もあります。しかし、複雑な画像を正確に説明するためには、4単語で十分とは考えにくいです。

デスクトップサイトで見つかった最長のテキスト長は15,357,625文字でした。これは、「戦争と平和」のサイズの本を5冊半（「戦争と平和」の平均語長が4.7文字だと仮定して）埋めるのに十分な長さである。

ウェブ上の動画

ビデオやその他のマルチメディアコンテンツは、ウェブ体験を豊かにできますが、多くの場合すべてのユーザーがしっかりとサポートされているわけではなく、サポートが実装されていなければ大きなアクセシビリティの障壁となる可能性があります。詳細については、W3C's Making Audio and Video Accessible[362]を参照してください。

キャプション

キャプションやトランスクリプトは、聴覚障害者や難聴者に聴覚情報を伝えるために必要であり、また音声処理の困難さなどの認知障害を持つユーザーにとっても非常に有用である。

360. https://x.com/codingchaos
361. https://sarahmhigley.com/writing/tooltips-in-wcag-21/
362. https://www.w3.org/WAI/media/av/

またトランスクリプトは、視覚障害者や目の不自由なユーザーにとっても、視覚を説明することで役立ちます。ウェブ上のビデオコンテンツは、キャプションが付いていなければアクセスできません。画像に意味のある代替テキストを持つことの重要性と同様に、キャプションの質も非常に重要です。

> *“キャプションは対話だけでなく、誰が話しているのかを識別し、意味のある効果音を含めて音で伝えられる非音声情報を含む。” -WCAG, Understanding Success Criterion 1.2.2. キャプション*

0.79%

図8.8. クローズドキャプションを提供する動画

`<video>`要素を使用しているサイトのうち、クローズドなキャプションを提供しているのは0.79%にすぎません。これは、`<track>`要素の存在に基づいていると推測されます。一部のウェブサイトでは、ユーザーに動画や音声のキャプションを提供するためのカスタムソリューションを持っていることに注意してください。また、検出されたキャプションの品質や、それらが説明する動画の完全な意味を正確に伝えているかどうかを評価することはできません。

自動再生動画

すべてのユーザーのためにウェブサイト上で動画を自動再生したりループすることは、間違いなく破壊的で望ましくないユーザー体験です。動画は、データだけでなくデバイスのバッテリーのリソースを消費する可能性があります。場合によっては不穏な画像を表示したり、発作を起こしやすい人への攻撃のベクトルとして使用されたりと、動画にはユーザーを苦しめるコンテンツが含まれていることがあります。

障害のあるユーザーにとっては、自動再生やループ再生による大きな障壁があります。スクリーンリーダーのユーザーにとって、音声を含むビデオはアナウンスを混乱させ、混乱を招く可能性があります。ADHDなどの認知障害を持つ人にとって、動画は非常に気が散り、ユーザーがインターフェイスを使用して理解する能力を妨げる可能性があります。前庭疾患のある人は、ビデオが危険な引き金になることもあります。

ウェブコンテンツアクセシビリティガイドラインには、5秒以上再生される移動、点滅、スクロールするコンテンツ（動画を含む）には、一時停止、停止、非表示の仕組みがあることを要求する基準2.2.2 Pause, Stop, Hide[363]があります。

363. https://www.w3.org/WAI/WCAG21/Understanding/pause-stop-hide.html

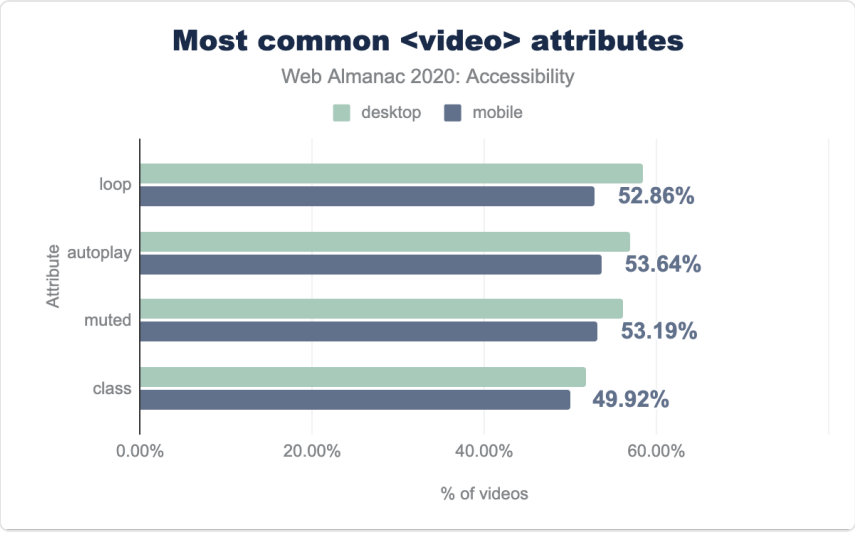


図8.9. 最も一般的な `<video>` 属性。

動画があるページのうち、デスクトップページの56.98%とモバイルページの53.64%が `autoplay` 属性を持っており、デフォルトで動画再生されることがわかりました。また、デスクトップページの58.42%、モバイルページの52.86%が `loop` 属性を持っており、これは動画が無期限に再生されることを意味している可能性の高いことがわかりました。これらの動画を一時停止、停止、または非表示にするメカニズムも考えられますが、自動再生やループ再生を停止する必要はなく動画を再生することを選択することがデフォルトであるべきです。これらのメトリクスは、動画を持つウェブサイトの半分以上が重要なアクセシビリティの障壁を持っている可能性があることを示唆しています。

ページナビゲーションのしやすさ

ユーザーが迷子になったり、そもそも私たちのサイトへ来た目的に必要なコンテンツを見つけることができなかったりすることがないように、ページは簡単に移動できるようにする必要があります。また、このソフトウェアのユーザーが解読不可能なテキストの壁を残すことがないように、スクリーンリーダーの技術は異なるセクションを区別できる必要があります。

見出し

見出しは、目次のようにジャンプできる階層を提供することで、スクリーン・リーダーが適切にページをナビゲートするのを容易にします。

58.72%

図8.10. Lighthouse監査に合格したモビールのサイトは、適切に順序付けられた見出しのために。

私たちの監査では、チェックされたサイトの58.72%が、レベルをスキップしない適切に並べられた見出しのテスト[364]に合格していることが明らかになりました。これらの見出しは、ページの意味的な構造を伝えます。多くのスクリーン・リーダー・ユーザーは見出しを使ってページをナビゲートしているので、見出しが正しい順序（ジャンプせずに昇順）であることは、支援技術ユーザーに最高の体験を提供することを意味します。このルールが守られる可能性の高いページだけをチェックしていることは注目に値します。

リンクをスキップする

スキップリンクは、ユーザーがナビゲーションシステムなどのインタラクティブなコンテンツをスキップして、別の目的地（通常はページのメインコンテンツ）に移動することを可能にします。スキップリンクは通常、ページ上の最初のリンクであり、UI内で永続的に表示されるか、キーボードフォーカスを持つまで目に見えないようにできます。これにより、キーボードユーザーがアクセスしようとしているコンテンツにたどり着くため、余計な数の要素をタブで移動する必要がなくなります。

スキップリンクはブロックのバイパスとみなされます。2020年のLighthouseの監査データによると、93.90%のサイトがバイパスブロック[365]テストに合格していることが明らかになりました。これは、`<header>`やスキップリンク、ランドマーク領域を設けて、ユーザーが反復的なコンテンツをスキップできるようにしていることを意味しています。

テーブル

テーブルは、2つの軸の関係を持つデータを表示する効率的な方法であり、比較に役立ちます。支援技術のユーザーは、適切に構造化された表をナビゲートして操作するために設計された特定のアクセシビリティ機能に依存しています。有効なセマンティック・テーブル・マークアップが存在しないと、これらの機能は使用できません。

364. https://web.dev/i18n/ja/heading-order/
365. https://www.w3.org/WAI/WCAG21/Understanding/bypass-blocks.html

測定	デスクトップ	モバイル
キャプション付きのテーブル	4.98%	4.20%
プレゼン用テーブル	0.64%	0.49%

図8.11. テーブルのアクセシビリティデータ。

テーブルキャプション

テーブルキャプションは、テーブルデータのコンテキストを提供するテーブルのラベルとして機能します。テーブルキャプションを使用しているのは、デスクトップサイトでは4.98%、モバイルサイトでは4.20%に過ぎませんでした。

プレゼン用テーブル

2020年には、流動的なレスポンシブレイアウトを可能にする柔軟性の高いCSSの方法論が非常に多く存在することは幸いです。しかし、FlexboxやCSS Gridが登場する前の何年も前は、開発者はレイアウトにテーブルを使うことが多かった。残念ながら、レガシーなWebサイトとレガシーな開発技術の組み合わせが原因で、レイアウトにテーブルが使われているサイトがまだ存在しています。

この技術をリーチする絶対的な必要性がある場合は、支援技術がテーブルのセマンティクスを無視するように、プレゼンテーションの役割をテーブルに適用すべきです。デスクトップページで0.63%、モバイルページで0.49%がプレゼンテーションの役割を持つテーブルを持っていることがわかりました。これが良いことなのか悪いことなのかはわかりません。プレゼン用に使われているテーブルが少ないことを示しているのかもしれませんが、レイアウトに使われているテーブルは、この必要とされる役割を欠いているだけである可能性が高いです。

ドキュメントタイトル

記述的なページタイトルは、支援技術を使ってページ、タブ、ウィンドウ間を移動する際に、コンテキストの変更がアナウンスされるため、コンテキストに役立ちます。当社のデータによると、98.98%のサイトがタイトルを持っていることが示されており、これは希望に満ちた統計です。しかし、ホームページの方が重要度の低いルートよりもページタイトルの割合が高いのは当然のことです。

タブインデックス

タブインデックスは、ページ全体でフォーカスを移動させる順序を決定します。ボタンやリンク、フォームコントロールなどのインタラクティブなコンテンツには、自然な`tabindex`の値は`0`です。同様に、インタラクティブでキーボードフォーカスの順序にあるカスタム要素やウィジェットには、明示的に`tabindex="0"`を割り当てる必要があります。インタラクティブではない要素にフォーカスを当てたいがキーボードのタブ順ではない場合は、`tabindex`の値`-1`を使用してJavaScriptでプログラム的にフォーカスを設定できます。

ページのフォーカス順は常に文書の流れによって決定されるべきです。`tabindex`を正の整数値に設定すると、ページの自然な順序が上書きされてしまいます。ページの自然な順序を尊重することは、一般的にはよりアクセシブルな体験につながります。デスクトップサイトの5%とモバイルサイトの4.34%が正の整数をタブインデックスの値として使用していることがわかりました。

ウェブ上の支援技術

様々な障害を持つ人々は、様々な支援技術を使用してウェブを体験するのを支援しています。W3CのWeb Accessibility Initiative(WAI)のTools and Techniques[366] の記事では様々な支援技術を使用して、ユーザーがどのようにウェブを知覚し、理解し、対話できるかについて説明しています。

ウェブ用の支援技術には、以下のようなものがあります。

- スクリーンリーダー
- 音声制御
- スクリーン拡大鏡
- 入力デバイス（ポインターやスイッチデバイスなど）

スクリーンリーダーは、通常、ユーザーが操作したり対話したりする際に、コンピューターが話しかけたり、インターフェイス内のコンテンツをアナウンスしたりすることで、音声でコンテンツを表示します。これにより、視覚障害者や弱視者、その他の障害者や非障害者のユーザーは、画面に表示される視覚的な合図を頼ることなくコンテンツを利用できます。

366. *https://www.w3.org/WAI/people-use-web/tools-techniques/*

ARIAの紹介

2014年にWAIはAccessible Rich Internet Application（ARIA）を発表しました。彼らはARIAを次のように記述しています[367]。

> *“WAI-ARIA（Accessible Rich Internet Applications Suite）は、障害者がウェブコンテンツやウェブアプリケーションをよりアクセスしやすくする方法を定義しています。特に、Ajax、HTML、JavaScriptおよび関連技術を用いて開発された動的コンテンツや高度なユーザーインターフェイスコントロールを支援します。”*

ほとんどの開発者は、ARIAをスクリーン・リーダー・ユーザーにとってより使いやすくするためHTMLに追加できる属性と考えていますが、不適切なマークアップやネイティブHTMLの解決策を補うことを意図したものではありません。ARIAには多くのニュアンスがあり、誤解されると、新たなアクセシビリティの障壁をもたらす可能性があります。さらに、異なるスクリーンリーダーは、ARIAのサポートに関して様々な制限を持っています。

ARIAの5つのルール

この強力なツールセットを利用する前に理解しておかなければならないARIAの5つのルール[368]があります。これは要求される適合性を備えた公式な仕様ではなく、ARIAを正しく理解し、実装するためのガイドです。

1. 要素を再利用して、アクセス可能にするためのARIAの役割、状態、プロパティを追加するのではなく、必要なセマンティクスや動作がすでに組み込まれているネイティブHTML要素HTML5.1[369]や属性を使用できるのであればそうしてください。
2. 本当に必要でない限り、ネイティブセマンティクスを変更しないでください。
3. すべての対話型ARIAコントロールは、キーボードで使用可能でなければなりません。
4. フォーカス可能な要素に `role="presentation"` や `aria-hidden="true"` を使用してはいけません。
5. すべてのインタラクティブ要素は、アクセス可能な名前でなければなりません。

ARIAの役割

ARIAが使用される最も一般的な方法の一つは、ある要素の役割を明示的に定義することによって、その目的を支援技術に伝えることです。

367. https://www.w3.org/WAI/standards-guidelines/aria/
368. https://www.w3.org/TR/using-aria/
369. https://www.w3.org/TR/html51/

HTML5では多くの新しいネイティブ要素が導入されましたが、そのすべてがロールを含む暗黙のセマンティクス[370]を持っています。例えば、`<nav>`要素は暗黙のうちに`role="navigation"`を持っており、支援技術者に目的情報を伝えるためにこのロールを明示的に追加する必要はありません。現在、デスクトップページの64.54%がARIAの`role`属性のインスタンスを少なくとも1つ持っています。中央値のサイトには2つの`role`属性のインスタンスがあります。

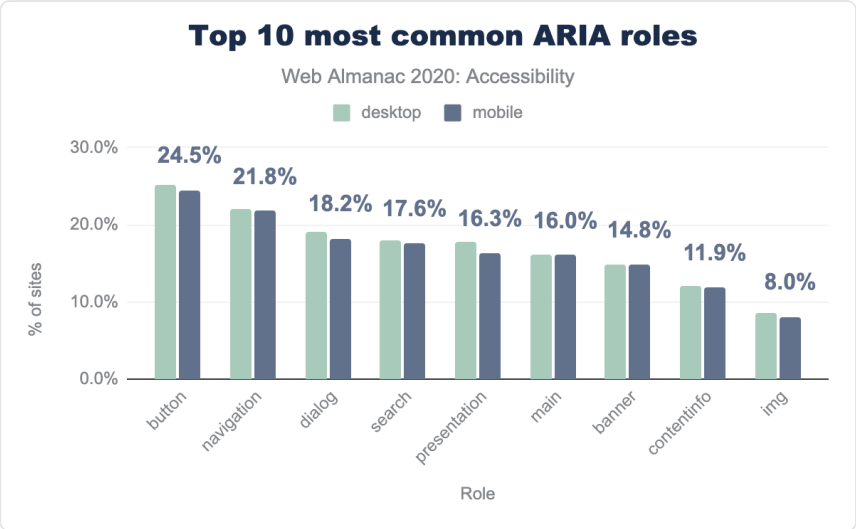


図8.12. ARIAの役柄トップ10

ボタンを使えばいいんだよ!

デスクトップサイトの25.20%、モバイルサイトの24.50%が、明示的`role="button"`が割り当てられた要素を少なくとも1つ持つホームページを持っていることがわかりました。これは約4分の1のウェブサイトが要素のセマンティクスを変更するために`button`ロールを使用していることを示唆していますが、ボタンロールを明示的に割り当てられているボタンを除いては、冗長ではありますが無害です。

`<div>`や`<span>`のような非インタラクティブな要素にこの役割が与えられている場合、ARIAの5つのルールのうちの1つ以上が破られている可能性が高いです。

ARIAの第一のルールに従えば、ネイティブの`<button>`要素の方が良い選択である可能性が高いです。また役割が追加されたにもかかわらず、期待されるキーボードのサポートが提

370. https://www.w3.org/TR/wai-aria-1.1/#implicit_semantics

供されていない可能性もあり、これはARIAの第三のルールを破り、WCAG 2.1.1, Keyboard[371]に違反することになります。

8.28%

図8.13. 携帯サイトでは `role="button"` を `<div>` や `<span>` に割り当てています。

デスクトップページの8.27%とモバイルページの8.28%では、`role="button"` が明示的に定義された `<div>` または `<span>` 要素が少なくとも1つ存在していることがわかりました。このようにARIAのロールを追加する行為、つまり「role-up[372]」は、正しいネイティブHTML要素を使用するよりも理想的ではありません。

デスクトップページの15.50%とモバイルページの14.62%には、`role="button"` を持つアンカー要素が少なくとも1つ含まれていることがわかりました。例えば、リンク（リンクには暗黙のうちに `role="link"` が設定されている）に `role="button"` を与えるなど、暗黙の役割が尊重されるべき要素にロールが適用されている場合、これはARIAの第2のルールを破ることになります。また、正しいキーボードの動作が実装されていなければ、WCAG 2.1.1.1, Keyboard[373]に違反することになります（リンクはスペースキーでは有効になりませんが、ボタンは有効になります）。

繰り返しになりますが、これらのケースの大部分では、問題の要素に明示的に `role="button"` を定義するよりも、期待されるセマンティクスと動作を持つネイティブHTMLの `<button>` 要素を活用する方が良いパターンです。

ナビゲーション

デスクトップページの22.06%とモバイルページの21.76%には、`role="navigation"` という画期的な役割を持つ要素が少なくとも1つあることがわかりました。ARIAの最初のルールに従って、開発者はこの役割を要素に追加するのではなく、暗黙のうちに正しいセマンティクスを持つHTML5の `<nav>` 要素を活用すべきです。このロールが明示的に `<nav>` 要素に追加されている可能性もありますが、これは冗長ですがアクセシビリティの問題ではありません。

ダイアログモーダル

ダイアログ・モーダルには、多くの潜在的なアクセシビリティの障壁があります。より詳し

371. https://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation-keyboard-operable.html
372. https://adrianroselli.com/2020/02/role-up.html
373. https://www.w3.org/TR/UNDERSTANDING-WCAG20/keyboard-operation-keyboard-operable.html

い内容については、Scott O'Hara[374]の記事Having an Open Dialog[375]をお読みになることをお勧めします。

デスクトップページの19.01%、モバイルページの18.21%に少なくとも1つの `role="dialog"` が存在しており、これは2019年の約8%から増加していることを報告します。この増加の一部は、この指標の測定方法が変更されたことによるものと思われます。これは、より多くの開発者がダイアログを構築する際にアクセシビリティを考慮しており、フレームワークや関連パッケージがよりアクセシブルなダイアログパターンを実装している可能性があることを示唆しています。しかし、ダイアログ・モーダルをアクセシブルにするには、 `dialog` ロールを使う以上のことが必要です。フォーカス管理、適切なキーボードサポート、スクリーンリーダーの露出など、すべてに対処する必要があります。

タブ

タブは一般的なインターフェースのウィジェットですが、多くの開発者にとってアクセシブルにすることが課題となっています。アクセシブルな実装のための一般的なパターンは、WAI-ARIA Authoring Practices Design Patterns[376]から来ています。ARIA Authoring Practicesは仕様書ではなく、理想化されたARIAの構成を示すものであることに注意してください。これらは、ユーザとのテストなしに本番環境で使用すべきではありません。

このパターンでは、親コンテナは `role="tablist"` を持ち、子要素は `role="tab"` を持ちます。これらのタブは `role="tabpanel"` を持つ要素に関連付けられており、そのタブの内容が含まれています。

374. https://x.com/scottohara
375. https://www.scottohara.me/blog/2019/03/05/open-dialog.html
376. https://www.w3.org/TR/wai-aria-practices-1.1/#tabpanel

図8.14. `tablist` の役割を持つ要素 (出典: W3C[377]

377. https://www.w3.org/TR/wai-aria-practices-1.1/examples/tabs/tabs-1/tabs.html

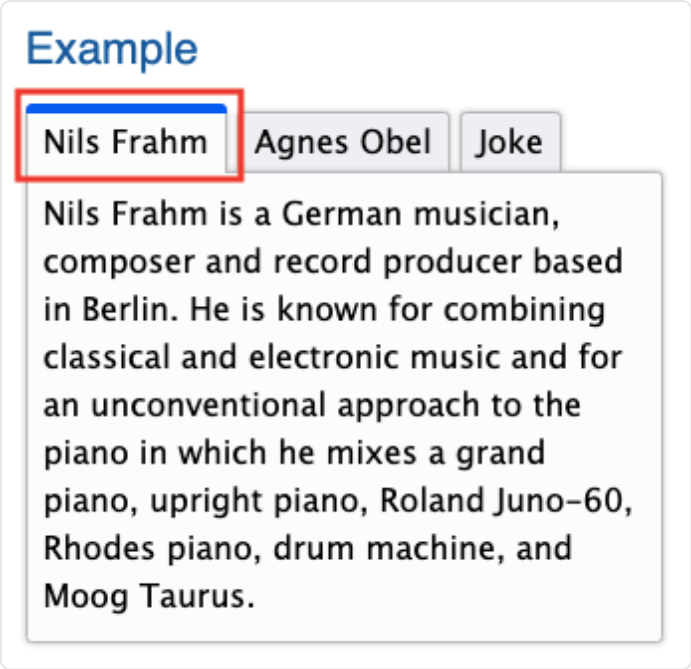


図8.15. 要素で `tab` の役割を持つ。(出典: W3C[378]

378. https://www.w3.org/TR/wai-aria-practices-1.1/examples/tabs/tabs-1/tabs.html

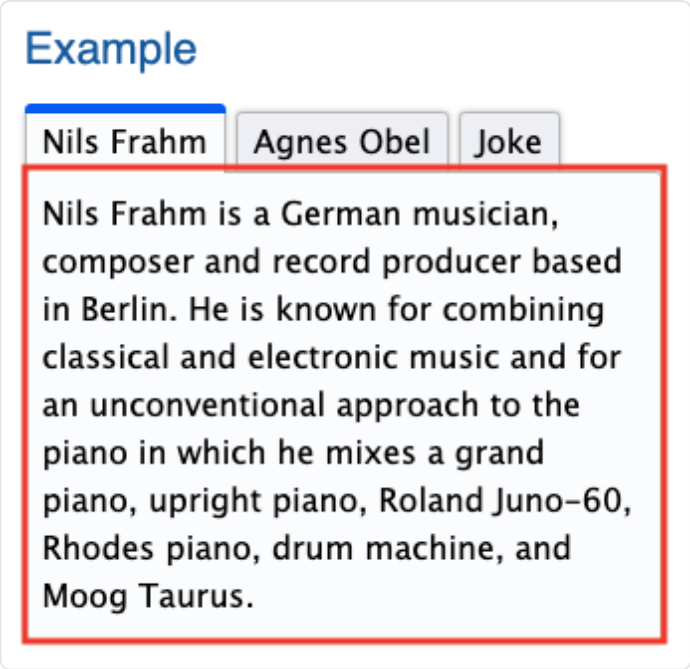


図8.16. 要素で `tabpanel` の役割を持つ。(出典: W3C[379]

デスクトップページでは、`role="tablist"` の要素を少なくとも1つ持っているページが7.00%であるのに対し、`role="tab"` の要素を持っているページは5.79%、`role="tabpanel"` の要素を持っているページは5.46%しかありませんでした。このことから、このパターンは部分的にしか実装されていない可能性があります。仮にタブ/タブパネル要素の一部に動的なレンダリングが行われていたとしても、現在表示されているタブ/タブパネルまたは最初のタブ/タブパネルは、理論的にはページロード時にDOM内にあることになります。

プレゼンテーション

ある要素に `role="presentation"` が与えられると、その要素のセマンティクスは、それが割り当てられた要素と必要とされる子要素の両方で剥ぎ取られます。例えばテーブルとリストはどちらも必須の子要素を持っているので、親要素に `role="presentation"` が与えられている場合、これは本質的に子要素にカスケードされ、子要素もまた意味論が剥ぎ取られます。要素のセマンティクスを削除するということは、その要素は見た目を除いて、もはやその要素ではないということを意味します。例えば、`role="presentation"` を持つリ

379. https://www.w3.org/TR/wai-aria-practices-1.1/examples/tabs/tabs-1/tabs.html

ストは、もはやリスト構造に関する情報をスクリーン・リーダーに伝えることはできません。

この属性の一般的な使用法は、表形式のデータではなくレイアウトに使用されている `<table>` 要素のためのものです。このようにテーブルを使用することはお勧めしません。レイアウトのためには、FlexboxやCSS Gridのような強力なCSSツールがあります。一般的には、 `role="presentation"` が支援技術を持つユーザにとって特に有用な使用例はほとんどないので、このロールは控えめに、そして慎重に使用してください。

ARIA属性

ARIA属性は、インターフェイスのアクセシビリティを強化するためHTML要素に割り当てることができます。ARIAの最初のルールを尊重して、ネイティブHTMLでできることを達成するために使用すべきではありません。

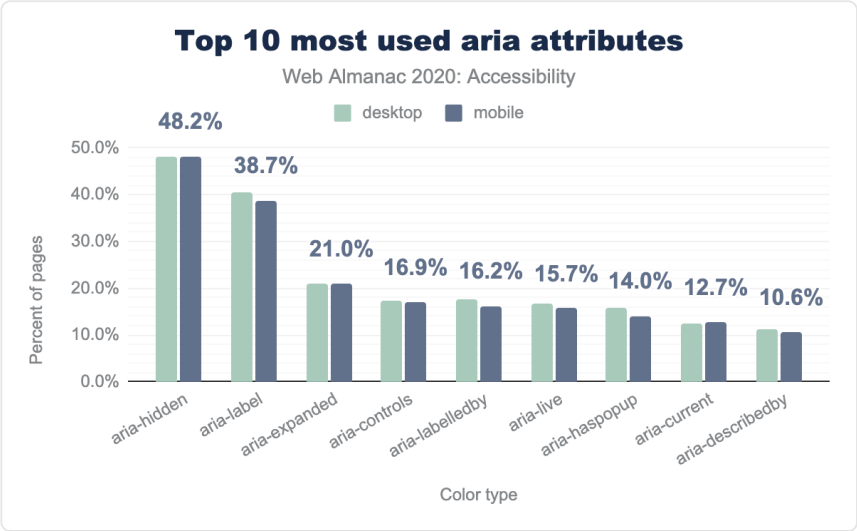


図8.17. 最もよく使われる `aria` 属性のトップ10。

ARIAによる要素のラベル付けと記述

ブラウザのアクセシビリティ・ツリーには、コントロール、ウィジェット、グループ、またはランドマークにアクセシブルな名前（もしあれば）を割り当てて、支援技術によってアナウンスできるようにする計算システムがあります。アクセシブルな名前にどの値が割り当てられているかを決定するための特異性ランキングがあります。

アクセシブルな名前は要素の内容（ボタンのテキストなど）、属性（画像の `alt` テキスト値など）、または関連する要素（フォームコントロールのためのプログラム的に関連するラベルなど）から派生できます。アクセシブルな名前についての詳細な情報は、Léonie Watsonの記事What is an accessible name?[380]を参照してください。

また、ARIAを使用して、要素にアクセス可能な名前を提供することもできます。これを実現するARIA属性は、 `aria-label` と `aria-labelledby` の2つです。これらの属性のいずれかは、アクセシブル名の計算に「勝ち」、ネイティブに派生したアクセシブル名を上書きしますので注意して使用してください。ARIAを使って要素に名前を付ける場合、WCAG 2.5.3, Label in Name[381] の基準に違反していないことを確認することが重要です。

要素の `aria-label` は、開発者が文字列の値を提供することを可能にし、これがその要素のアクセシブルな名前に使用されます。デスクトップページの40.44%、モバイルホームページの38.72%が `aria-label` 属性を持つ要素を少なくとも1つ持っていることがわかり、これがアクセシブルな名前を提供するための最も一般的なARIA属性となりました。

属性は `id` 参照を値として受け取り、それをインターフェイス内の別の要素と関連付けて、そのアクセシブルな名前を提供します。要素は、そのアクセシブルな名前を提供するこの別の要素によって「ラベル付けされた」要素になります。デスクトップページの17.73%、モバイルページの16.21%が `aria-labelledby` 属性を持つ要素を少なくとも1つ持っていることがわかりました。

繰り返しになりますが、ARIAの最初のルールが尊重されるべきです。要素がARIAを必要とせずにそのアクセス可能な名前を導出できるならば、これが望ましい。例えば、グラフィカルな要素ではない `<button>` は、ARIA属性ではなく、そのアクセス可能な名前をテキストの内容から取得すべきです。フォーム要素は可能な限り、適切に関連付けられた `<label>` 要素からアクセス可能な名前を導き出すべきです。

要素に対してより強固な記述が必要な場合には、 `aria-describedby` 属性を使用できます。また、インターフェイスの他の場所に存在する記述的なテキストと接続するための `id` 参照を値として受け入れます。これはアクセシブルな名前を提供しないので、代替ではなく補完としてアクセシブルな名前と一緒に使うべきです。デスクトップページの11.31%とモバイルページの10.56%が `aligned-describedby` 属性を持つ要素を少なくとも1つ持っていることがわかった。

楽しい事実! 属性 **`aria-labeledby`** を持つ3,200のウェブサイトが見つかりましたが、これは `aria-labelledby` 属性のスペルミスです。これらのエラーを簡単に回避するために、自動チェックを必ず実行してください。

380. *https://developer.paciellogroup.com/blog/2017/04/what-is-an-accessible-name/*
381. *https://www.w3.org/WAI/WCAG21/Understanding/label-in-name.html*

コンテンツを隠す

支援技術がコンテンツを発見しないようにする方法はいくつかあります。CSS `display:none` や `visibility:hidden` を利用してアクセシビリティツリーから要素を省略できます。作者が特にスクリーンリーダーからコンテンツを隠したい場合は、 `aria-hidden="true` を使うことができます。デスクトップページの48.09%とモバイルページの48.23%が `aria-hidden` 属性を持つ要素のインスタンスを少なくとも1つ持っていることがわかりました。

これらのテクニックは、ビジュアル・インターフェースの何かが冗長であったり、支援技術のユーザーにとって役に立たない場合に特に役立ちます。すべてのユーザーに機能の同等性を提供することが不可欠なので、慎重に使用する必要があります。アクセシブルにすることが困難なコンテンツをスキップするために使用するのは避けてください。

コンテンツを隠したり表示したりすることは、現代のインターフェイスでは一般的なパターンであり、誰にとってもUIを整理するのに役立ちます。この開示パターンに役立つ追加要素として、2つのARIA属性があります。 `ARIA-expanded` 属性は、開示されたコンテンツが表示されるかどうかによってトグルする `true` / `false` の値を持つべきです。さらに、 `aria-controls` 属性は、開示されたコンテンツの `id` と関連付けることができ、トリガーとなるコントロール（ボタンであるべきです）と表示されるコンテンツとの間にプログラム的な関係を作ります。

デスクトップページの20.98%とモバイルページの21.00%が `aria-expanded` 属性を持つ少なくとも1つの要素を持ち、デスクトップページの17.38%とモバイルページの16.94%が `aria-controls` 属性を持つ少なくとも1つの要素を持っていることがわかりました。このことは、ウェブサイトの約5分の1が少なくとも部分的にアクセス可能な開示ウィジェットを実装している可能性があることを示唆しています。スクリーンリーダーのサポートがまだ理想的ではないため、 `aria-controls` 属性は開示パターンのために必須ではなく、ベストプラクティスと考えられていることに注意してください。

スクリーンリーダーのテキストのみ

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開発者がしばしば採用する一般的なテクニックは、CSSを使用してテキストの一節を視覚的に非表示にし、スクリーン・リーダーには表示されるが、インターフェイスには視覚的には表示されないようにすることです。 `display:none` と `visibility:hidden` の両方がアクセシビリティツリーにコンテンツが存在しないようにするため、これを実現するCSSコードの塊を含む一般的な「ハック」があります。このコードスニペットの最も一般的なCSSクラス名は、（慣習的にも、ブートストラップのようなライブラリ全体でも） `sr-only` と `visually-hidden` です。デスクトップページの13.31%とモバイルページの12.37%が、これらのCSSクラス名のいずれか、または両方を持っていることがわかりました。

動的にレンダリングされたコンテンツを発表

現代のウェブ開発における最大のアクセシビリティの課題の1つは、インターフェイスのいたるところにある動的にレンダリングされたコンテンツを扱うことです。新しいものや更新されたものがDOMにあることは、しばしばスクリーンリーダーに伝える必要があります。どのような更新を伝える必要があるのかを考える必要があります。例えば、フォームの検証エラーは伝える必要がありますが、怠惰にロードされた画像はそうでないかもしれません。また、進行中のタスクを中断させない方法で行う必要があります。

これに役立つツールの1つがARIAのライブリージョンです。ライブリージョンを使うことで、DOMの変化に耳を傾けることができ、更新されたコンテンツをスクリーン・リーダーがアナウンスできます。通常、`aria-live`属性は、動的にレンダリングされる要素ではなく、DOM内に既に存在するそれ自身のコンテナ要素に置かれます。ライブ領域のために他の要因によって動的に操作される可能性のないDOM内の専用ノードを決定することが重要であり、アナウンスの信頼性を確保します。このコンテナ内の要素が動的にレンダリングされたり更新されたりすると（例えば、ステータスの更新やフォームが正常に送信されなかったことの通知など）、変更がアナウンスされます。

デスクトップページの16.84%、モバイルページの15.67%がライブリージョンを持っていることがわかりました。この属性には3つの潜在的な値がある。`polite`, `assertive`, そして`off`です。一般的には`polite`値が使用されます。これはデフォルト値であることもありますが、動的コンテンツのアナウンスはユーザーがページとの対話を止めた後にのみ行われるからです。多くの場合、これはユーザーの入力を中断させるのではなく、望まれるユーザー体験です。ステータスの更新が十分に重要な場合、`assertive`を使うとスクリーンリーダーの現在の音声キューが中断されます。これが`off`に設定されている場合、アナウンスは起こりません。スクリーンリーダーの自然な体験と流れを尊重することが重要であり、`assertive`のアナウンスは極端な場合にのみ使用し、マーケティングのアナウンスのようなものには使用しないようにしてください。

フォームコントロールのアクセシビリティ

フォームは、アクセシビリティの観点から、最も重要なものの一つです。フォーム入力を正しく送信することは、ユーザーがウェブサイトやアプリケーションの主要な操作を実行できることを意味します。例えば、登録フローにアクセスできない場合、障害のあるユーザーはサイトに全くアクセスできないかもしれません。

デジタル・アクセシビリティは市民権であり、すべての人が情報にアクセスし、ウェブ上で同じ機能を実行する平等な権利を持っていることを忘れてはなりません。障害のあるユーザーが、特に政府サービスやその他の重要な活動のためにフォームを提出するような作業のために、ウェブの主要なタスクの実行や情報へのアクセスを妨げられた場合、民間部門と公共

部門の両方で差別の対象となることは明らかです。

フォームの検証

どのようなフォームのエラー処理であっても、支援技術に伝えることは非常に重要です。これを処理するためには、バリデーションの実装に応じてさまざまなテクニックがあります。Web AIMのUsable and Accessible Form Validation and Error Recovery[382] の記事は、様々なアクセシブルなフォームバリデーション戦略について学ぶための素晴らしいリソースです。

フォーム要素が必要とされる場合、これも支援技術に伝える必要があります。ネイティブHTMLフォーム要素には `required` 属性 を使用し、カスタマイズされた要素には `aria-required` 属性が必要になるかもしれません。フォーム送信に問題がある場合は、その旨を支援技術に伝える必要があります。

フォームラベル

フォームラベルはUI内で可視化して永続化し、入力を求めていることを記述する必要があります。フォーマットや特殊文字などのユニークな要件を可視化されたラベルに入れておくと、可能な限りエラーを防ぐことができます。

26.51%

図8.18. すべてのラベルが適切に関連付けられているサイト

フォームラベルがそれぞれの入力とプログラム的な関連性を持っていることを確認することが重要です。ラベルを視覚的に表示するだけでは十分ではありません。すべてのラベルがそれぞれの入力と適切に関連づけられているサイトは26.51%にすぎないことがわかりました（ `for` / `id` の関係や、ラベルの中に入れ子になっている入力があれば達成されます）。

ラジオ入力やチェックボックスのようなフォームコントロールのグループは `<fieldset>` 要素の中に入れ子にして、 `<fieldset>` の中の `<legend>` 要素を使ってグループラベルを与えなければなりません。個々のコントロールは、それぞれの可視ラベルとプログラム的に関連付ける必要があります。

プレースホルダテキスト

プレースホルダテキストを入力のラベルとして使用しないでください。スクリーンリーダー

382. https://webaim.org/techniques/formvalidation/

の中にはプレースホルダテキストからアクセシブルな名前を決定できるものもありますが、認知障害のあるユーザーは、プレースホルダテキストに依存すると、ユーザーが入力を入力し始めるとすぐにプレースホルダーが消えてしまい、コンテキストが消えてしまうため悪影響を受ける可能性があります。音声コントロールのユーザーは、DOM内の要素を確実にターゲットにするため、プレースホルダーの値以上のものを必要とします。さらにプレースホルダテキストは、しばしばカラーコントラストの要件を満たしていないことがあり、弱視のユーザーに悪影響を与えます。

プレースホルダテキストを含むフォームコントロールを持つサイトのうち、73.89%のサイトでは、デスクトップではコントロールにプログラム的に関連付けられた `<label>` 要素が存在しないインスタンスが少なくとも1つあり、モバイルでは74.52%となっています。

結論

この章は、このAlmanacのユーザー体験のセクションに含まれています。アクセシビリティの提唱者Billy Gregoryがかつて言っていました[383]のように、「UXがすべてのユーザーを考慮していない場合、それはSOME User Experience、またはSUXとして知られるべきではないでしょうか」。あまりにも多くの場合、アクセシビリティの作業は付加的なもの、エッジケース、あるいは技術的な負債に匹敵するものとみなされ、本来あるべきウェブサイトや製品の成功の核心をなすものではありません。

アクセシビリティの実装は、開発者だけの責任ではありません。製品チームと組織全体が成功するためには、責任の一部としてアクセシビリティを持たなければなりません。アクセシビリティの作業は、製品サイクルの中で左にシフトする必要があります。つまり、開発前のリサーチ、アイデア出し、設計段階に組み込む必要があります。

役割別の潜在的なアクセシビリティの責任

このリストは網羅的なものでなく、説明責任の駅伝のように、アクセシブルなウェブサイトやアプリケーションを実現するために、すべての役割がどのように協力していくかを考えてもらうことを目的としています。

人事・人事業務

- 障害者実務者を含むアクセシビリティスキルを持った人材の採用・採用。
- 障がいのある人にも対応した包括的な職場環境づくり

UX/プロダクトデザイナー

383. *https://x.com/thebillygregory/status/552466012713783297?s=20*

- 調査・構想の段階で様々な障害を持つ人たちのことを考え、話をすること。
- 見出しの階層構造、スキップリンク、代替テキストの提案（コピーライターやコンテンツ担当者によるものもあります）、スクリーンリーダー専用テキストなどのアクセシビリティ情報をワイヤーフレームに注釈をつけます。

UIデザイナー

- 色のコントラストの選択、フォントの選択、間隔、行の高さの考慮事項。
- アニメーションの考察（必要かどうかの判断、`prefers-reduced-motion`シナリオへの静的アセットの供給、ポーズ/ストップ機構の設計）。

プロダクトマネージャー

- ロードマップの中でアクセシビリティの作業に優先順位をつけ、バックログの最後に技術的な負債とならないようにします。
- チームが作業を検証するためのプロセスを作成します。例えば、作業の定義や受け入れ基準にアクセシビリティを含めることができます。

開発者

- 可能な限りネイティブなHTMLソリューションを好み、ARIAを理解していて、いつ使うべきかを理解しています。
- 自動および手動テストですべての作業を検証し、同僚のプルリクエストを同じ基準で評価します。

品質保証

- ワークフローにアクセシビリティテストを組み込む。
- チームの品質戦略や受け入れ基準に貢献する際、アクセシビリティへの配慮を提唱する。

リーダーシップ/経営幹部

- 従業員がアクセシビリティのスキルセットを学び、成長するための帯域幅を提供し、専門知識と生きた経験を持つ実務家を採用する。
- アクセシビリティのコアを製品の成果と考え、アクセシビリティ・エクセレンスを推進可能な仕事として捉えています。

技術業界は、包括主導の開発に向かう必要があります。これには多少の先行投資が必要ですが、アクセシビリティを考慮せずに構築されたサイトやアプリを改造しようとするよりも、アクセシビリティを製品に組み込むことができるようにサイクル全体に組み込む方がはるかに簡単で時間の経過とともにコストも低くなるでしょう。

最大の投資は、教育とプロセスの改善という形で行われるべきです。UIデザイナーが色のコントラスト要件のニュアンスを理解したら、アクセス可能なカラーパレットを選択することは、アクセスできないパレットと同じ努力でなければなりません。開発者がネイティブHTMLとARIAを深く理解し、特定の技術やツールに手を伸ばすべき時期を理解すれば、彼らが書くコードの量は同等になるはずです。

業界として、この章の数字が物語っていることを認識する時が来ています。そのためには、トップダウンのリーダーシップと投資、そして実践を推し進め、ウェブを利用する障害者のニーズ、安全性、包摂性を擁護するボトムアップの努力の組み合わせが必要です。

著者

Olu Niyi-Awosusi

@oluoluoxenfree　oluoluoxenfree　https://olu.online/

Olu Niyi-AwosusiはFTのソフトウェアエンジニアで、リスト、新しいことを学ぶこと、Bee and Puppycat、社会正義、アクセシビリティ[384]、そして日々努力することを愛しています。

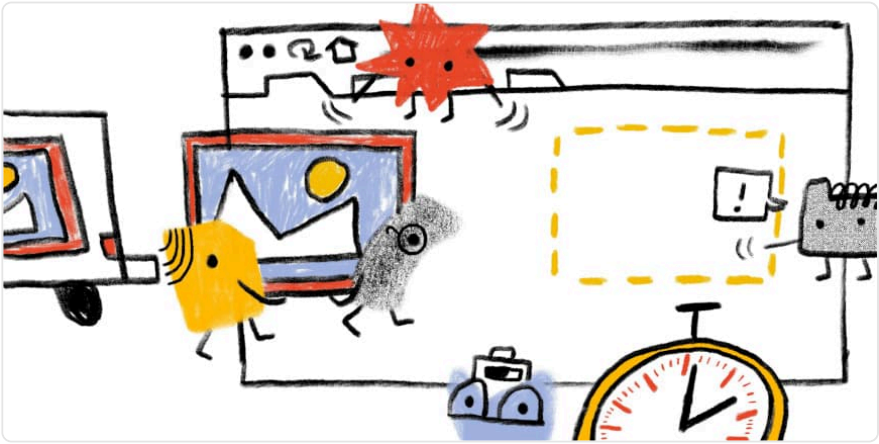
Alex Tait

@at_fresh_dev　alextait1　https://atfreshsolutions.com

Alex Taitは、開発者、コンサルタント、教育者であり、その情熱は、モダンJavaScript内のインターフェイスアーキテクチャとデザインシステム、アクセシビリティにあります。 開発者として、アクセシビリティを前面に出した包括主導の開発手法は、すべての人にとってより良い製品につながると信じています。 コンサルタントであり戦略家でもある彼女は、「少ないことは豊かである」を信条としており、新しい機能の要件変更は、障害のあるユーザーにとってコア機能の等価性よりも優先されるべきものではないと考えています。教育者として、技術がより多様で公平で包括的な産業になるように、情報の障壁を取り除くことを信条としています。

384. *https://alistapart.com/article/building-the-woke-web/*

部 II 章 9
パフォーマンス



Karolina Szczur によって書かれた。
Boris Schapira、 Rick Viscomi、 David Fox、 Noam Rosenthal、 Leonardo Zizzamia と Shane Exterkamp によってレビュ。
Max Ostapenko と Pokidov N. Dmitry による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

スピードの遅さが全体的なユーザー体験、ひいてはコンバージョンに悪影響を及ぼすことは疑いの余地がありません。しかしパフォーマンスの低下は、単にフラストレーションの原因になったり、ビジネス上の目標に悪影響を及ぼすだけでなく、現実に参入障壁を生み出します。今年の世界的なパンデミックでは、そのような既存の障壁がさらに明らかになりました[385]。遠隔地での学習、仕事、交流への移行に伴い、突然、私たちの生活全体がオンラインに移行しました。接続性が悪く、高性能なデバイスにアクセスできないため、多くの人にとってこの変化は、不可能ではないにしても、せいぜい苦痛でしかありませんでした。これは、世界中の接続性、デバイス、速度の不平等を浮き彫りにする、真のテストとなりました。

385. *https://www.weforum.org/agenda/2020/04/coronavirus-covid-19-pandemic-digital-divide-internet-data-broadband-mobbile/*

パフォーマンス・ツールは、ユーザー体験の多様な側面を描き出し、根本的な問題を見つけやすくするために進化し続けています。昨年のパフォーマンスの章[386]以降、この分野では数多くの重要な開発が行われており、スピードモニタリングへの取り組み方をすでに変革しています。

人気の高い品質監査ツール「Lighthouse 6」の大幅なリリースに伴い、有名な「パフォーマンススコア」のアルゴリズムが大幅に変更され[387]、スコアも変更されました。Core Web Vitals[388]という、ユーザー体験のさまざまな側面を描いた新しいメトリクスが発表されました。これは、今後の検索ランキングの要因のひとつとなり、開発コミュニティの目を新しいスピードシグナルへ移すことになるでしょう。

この章では、Chrome User Experience Report (CrUX)[389]が提供する実世界のパフォーマンスデータを、これらの新開発のレンズを通して見ていくとともに、その他の関連するいくつかのメトリクスを分析します。なお、iOSの制限により、CrUXのモバイル版の結果には、アップル社のモバイルOSを搭載したデバイスは含まれていませんのでご了承ください。この事実は、とくに国ごとの指標のパフォーマンスを調べる際に、分析に影響を与えることは間違いありません。

それでは早速、ご紹介しましょう。

Lighthouseパフォーマンススコア

2020年5月、Lighthouse 6がリリースされました[390]。人気のパフォーマンス監査スイートの新しいメジャーバージョンでは、パフォーマンススコアのアルゴリズムに注目すべき変更が加えられました。パフォーマンススコアは、サイトの速度をハイレベルで表現したものです。Lighthouse 6では、5つの指標ではなく6つの指標でスコアを測定します。「First Meaningful Paint」と「First CPU Idle」が削除され、代わりに「Largest Contentful Paint」（LCP）、「Total Blocking Time」（TBT、ラボでは「First Input Delay」に相当）、「Cumulative Layout Shift」（CLS）が採用されました。

新しいスコアリングアルゴリズムでは、新世代のパフォーマンス指標が優先されています。「Core Web Vitals」を優先し「First Contentful Paint（FCP）」、「インタラクティブになるまでの時間（TTI）」、「Speed Index」は、スコアの重みが小さくなるにつれて優先度を下げています。また、このアルゴリズムでは、ユーザー体験の3つの側面が強調されています。*インタラクティブ性*（Total Blocking TimeとTime to Interactive）、*視覚的安定性*（Cumulative Layout Shift）、*負荷の認識*（First Contentful Paint、Speed Index、Largest Contentful Paint）です。

386. *https://almanac.httparchive.org/ja/2019/performance*
387. *https://calibreapp.com/blog/how-performance-score-works*
388. *https://calibreapp.com/blog/core-web-vitals*
389. *https://developers.google.com/web/tools/chrome-user-experience-report*
390. *https://github.com/GoogleChrome/lighthouse/releases/tag/v6.0.0*

また、デスクトップとモバイルで異なる基準点を用いてスコアを算出するようになりました。これが実際に意味するところは、デスクトップでは（高速なウェブサイトを期待して）寛容さに欠け、モバイルでは（モバイルでのベンチマークパフォーマンスはデスクトップよりも迅速ではないため）寛容になるということです。Lighthouse 5と6のスコアの違いを比較するには、Lighthouse Score calculator[391]をご利用ください。では、実際にスコアはどのように変化したのでしょうか。

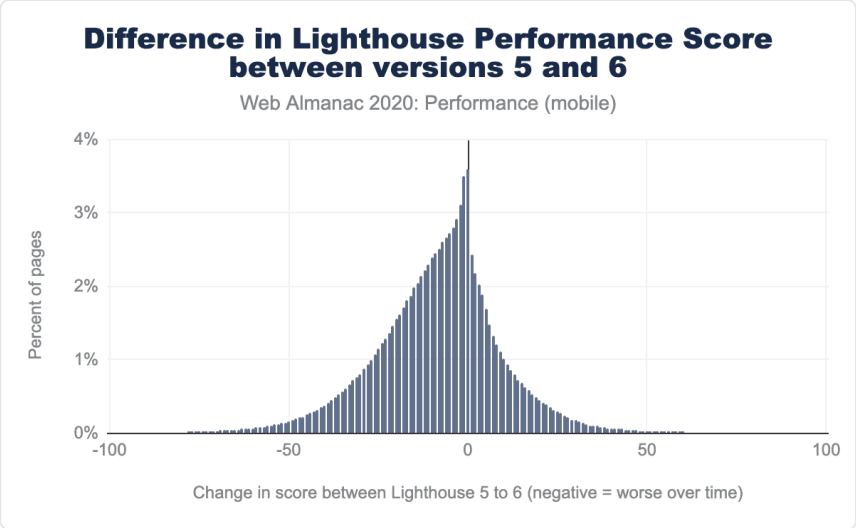


図9.1. Lighthouse Performance Scoreにおけるバージョン5と6の違い。

上の図では、4%のウェブサイトがパフォーマンススコアの変化を記録しなかったことがわかりますが、マイナスの変化を記録したサイトの数がスコアの改善を記録したサイトを上回っています。パフォーマンススコアの成績は悪化しており（10〜25ポイントのエリアでもっとも顕著な減少カーブを描いています）、これは以下でさらに直接的に描写されています。

391. https://googlechrome.github.io/lighthouse/scorecalc/

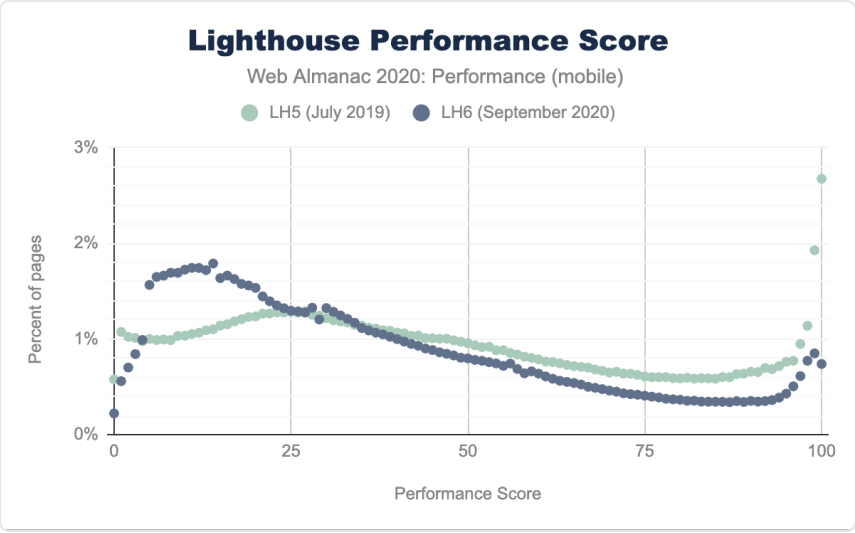


図9.2. Lighthouse 5と6のLighthouse Performance Scoreの分布。

Lighthouse 5とLighthouse 6の比較では、スコアの分布がどのように変化したかを直接観察できます。Lighthouse 6のアルゴリズムでは、0〜25点のスコアを獲得したページの割合が増え、50〜100点のスコアを獲得したページの割合が減っています。Lighthouse 5では、100点を獲得したページの割合は3%でしたが、Lighthouse 6では、その割合はわずか1%にまで減少しました。

このような全体的な変化は、アルゴリズム自体に多数の修正が加えられていることを考えると、予想外のことではありません。

Core Web Vitals: Largest Contentful Paint

Largest Contentful Paint(LCP)は、最大のabove-the-fold要素[392]がレンダリングされた時間を報告する、画期的なタイミングベースのメトリックです。

392. https://web.dev/i18n/ja/lcp/#what-elements-are-considered

デバイス別LCP

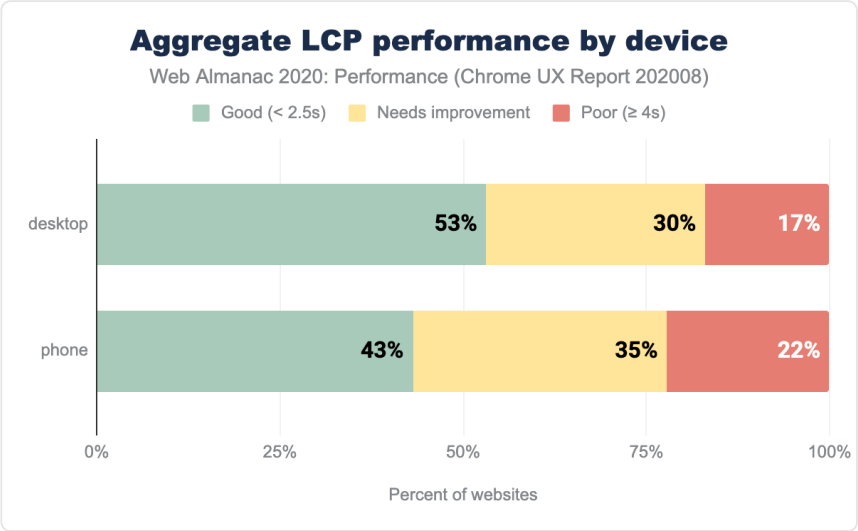


図9.3. デバイスタイプごとに分割されたLCPパフォーマンスの集計。

上のグラフでは、43%から53%のウェブサイトでLCPのパフォーマンスが、良好（2.5秒以下）であることがわかります。つまり、大多数のウェブサイトでは、重要なオーバーザフォールドメディアを優先的に高速で読み込むことができています。比較的新しい指標としては、これは楽観的なシグナルと言えます。デスクトップとモバイルで若干の差があるのは、ネットワークの速度、デバイスの性能、画像のサイズ（デスクトップ特有の大きな画像は、ダウンロードとレンダリングに多くの時間を使う）が、異なることが原因と考えられます。

地域別LCP

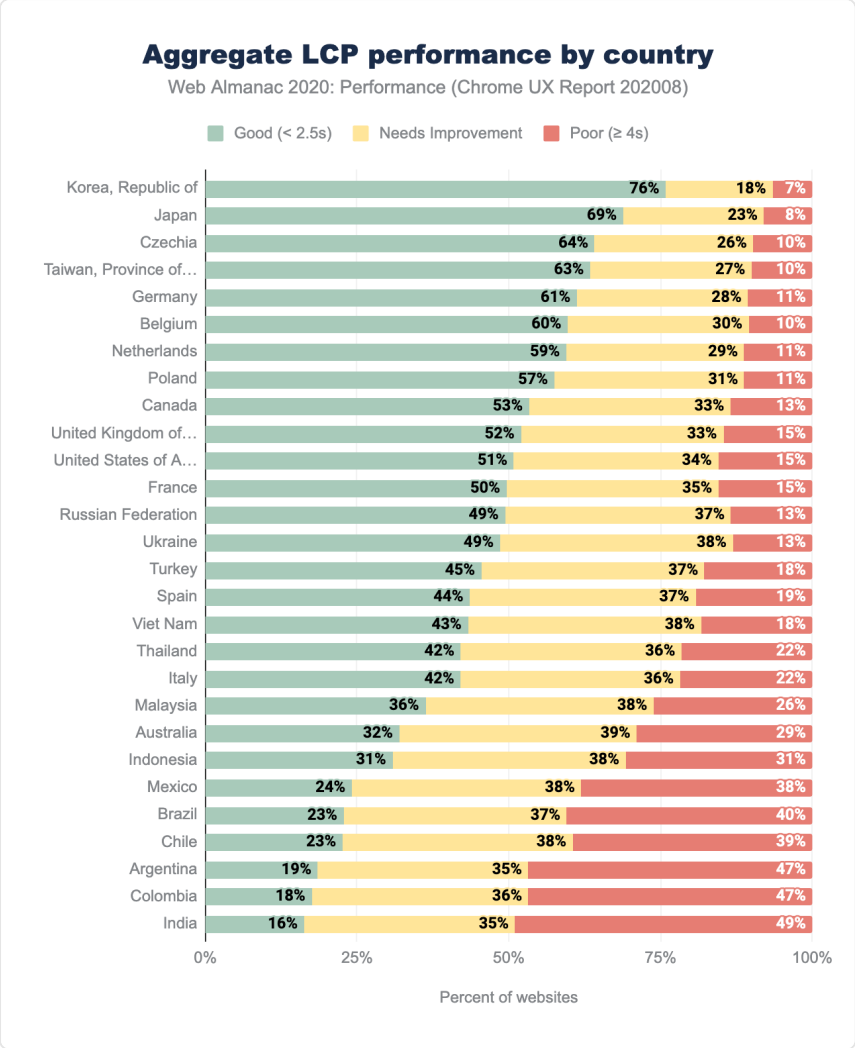


図9.4. 国別に分割されたLCPのパフォーマンスを集計。

良好なLCP測定値の割合がもっとも高いのは、主にヨーロッパとアジアの国々に分布しており、韓国は良好な測定値の割合が76%でトップでした。韓国は携帯電話の通信速度において常にトップであり、10月にはダウンロード145Mbpsという驚異的な数値が Speedtest Global

Index[393] で報告されています。日本、チェコ、台湾、ドイツ、ベルギーも、確実に高いモバイル速度を誇る国の一角です。オーストラリアは、モバイルネットワークの速度ではトップであるものの、デスクトップ接続の遅さと遅延に悩まされており、上記のランキングでは最下位となっています。

インドは、我々の一連のデータの中では最後の方に位置しており、良い体験ができたのはわずか16%でした。はじめてインターネットに接続する人口は継続的に増加していますが、モバイルとデスクトップのネットワーク速度は依然として問題となっています[394]。平均ダウンロード速度は4Gで10Mbps、3Gで3Mbps、デスクトップでは50Mbpsを下回っています。

接続タイプ別LCP

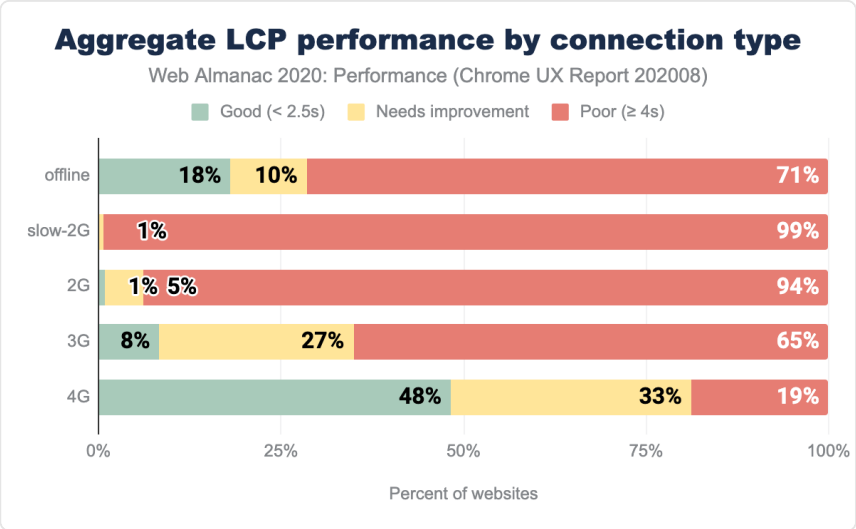


図9.5. 接続タイプ別に分割されたLCPパフォーマンスを集計。

LCPは、最大のオーバーザフォールド要素（画像、動画、テキストを含むブロックレベルの要素を含む）がレンダリングされたときに表示されるタイミングなので、ネットワークの速度が遅いほど、測定値の悪い部分が多くなるのは当然のことです。

ネットワーク速度とLCPパフォーマンスの向上には明確な相関関係がありますが、4Gであっても、結果が「良好」に分類されたのは48% に過ぎず、読み取り値の半分は改善が必要とされています。メディアの最適化を自動化し、適切なサイズとフォーマットを提供し、さらに低データモードに最適化することで、針を動かすことができるでしょう。このガイドでは、

393. https://www.speedtest.net/global-index
394. https://www.opensignal.com/reports/2020/04/india/mobile-network-experience

LCPの最適化について詳しく説明しています[395]。

Core Web Vitals: Cumulative Layout Shift

Cumulative Layout Shift(CLS)とは、ページ閲覧中にビューポート内の要素がどれだけ移動するかを数値化したものです。秒やミリ秒といった単位でインタラクションの特定の部分を測定しようとするのではなく、ウェブサイト上で予想外の動きがどの程度発生しているかを正確に把握し、ユーザー体験を評価できます。

このように、CLSは本章で紹介した他の指標とは異なる、新しいタイプのUXホリスティック指標です。

デバイス別CLS

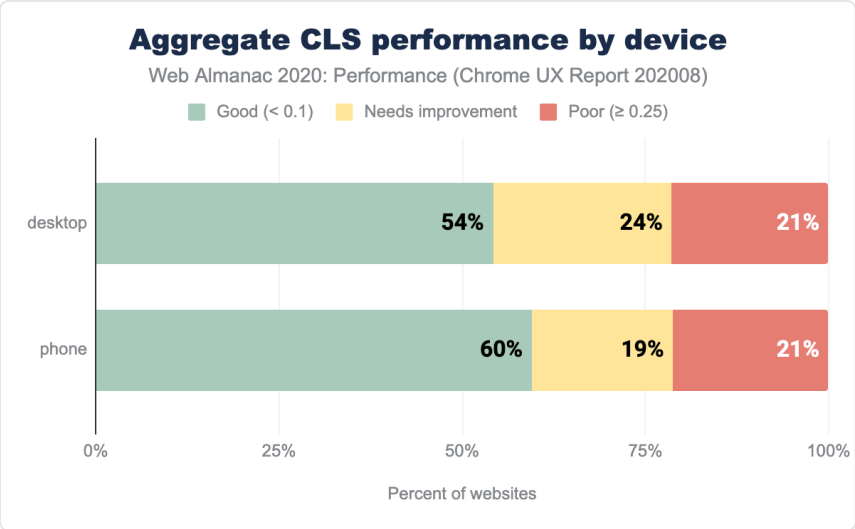


図9.6. デバイスタイプごとに分割されたCLSのパフォーマンスを集約。

CrUXのデータによると、デスクトップの場合もモバイルの場合も、半数以上のWebサイトがCLSの良い評価を得ています。デスクトップとモバイルでは、評価の高いウェブサイトの数にわずかな差（6ポイント）があり、モバイルの方が有利です。これは、モバイルに最適化されているため、機能やビジュアルが充実していない傾向があり、CLS評価では携帯電話がリードしていると推測できます。

395. https://web.dev/i18n/ja/optimize-lcp/

地域別CLS

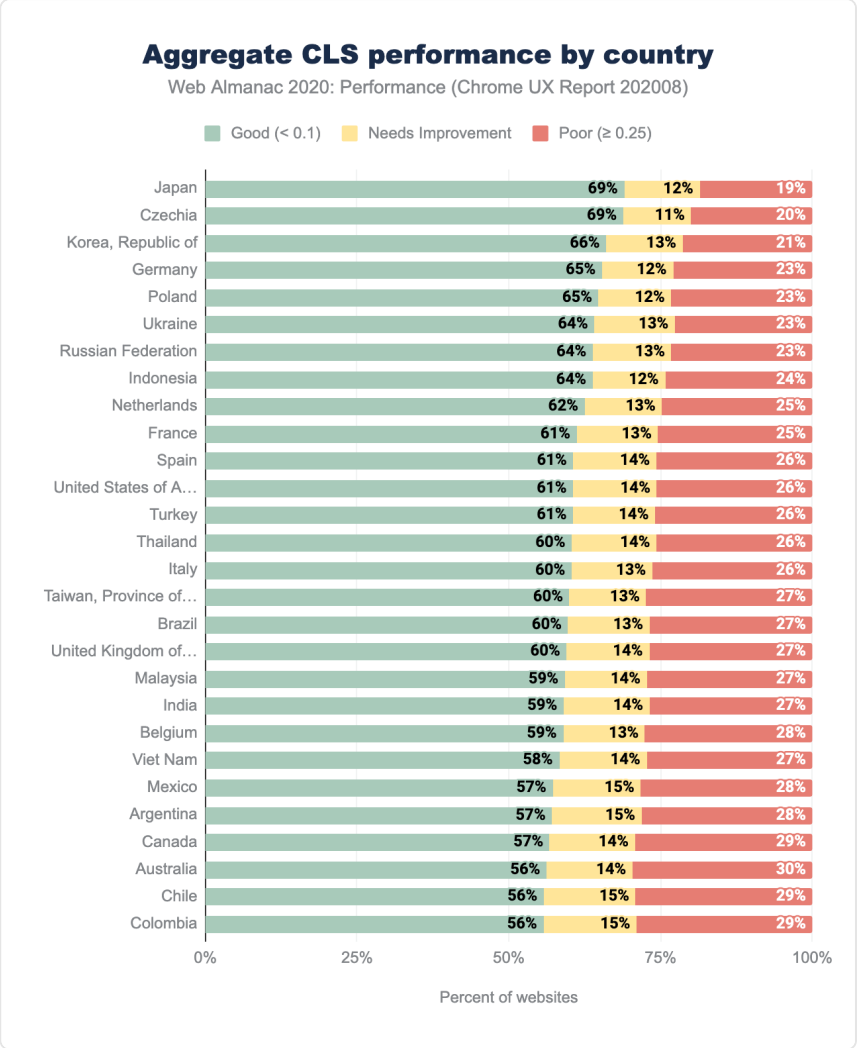


図9.7. 国別に分割されたCLSのパフォーマンスを集計しています。

異なる地域でのCLSのパフォーマンスは主に良好で、少なくとも56%のサイトが「良好」と評価されました。これは、視覚的に安定していると思われる優れたニュースです。LCPの地域分布で見られたように、韓国、日本、チェコ、ドイツ、ポーランドといった国が上位を占めていることがわかります。

視覚的安定性は、LCPのような他のメトリクスに対する地理的および遅延の影響を受けにくい。上位の国と下位の国との間の良い評価の割合の差は、LCPでは61%、CLSではわずか13%です。中程度の評価のウェブサイトの割合は全体的に比較的低く、全体的に悪い体験の割合は19%〜29%となっています。CLSの低下には多くの要因があります。Optimize Cumulative Layout Shift guide[396]では、それらの要因に対処する方法を紹介しています。

接続タイプ別CLS

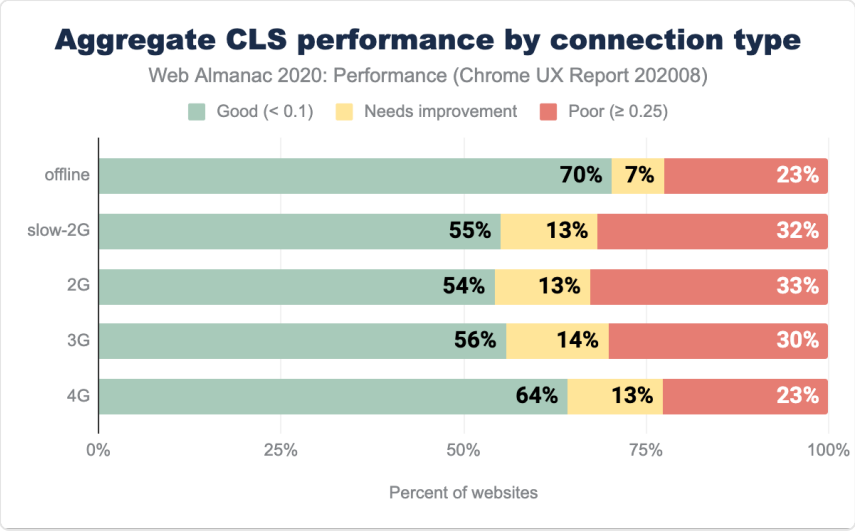


図9.8. CLSのパフォーマンスを接続タイプ別に集計したものです。

CLSスコアは、オフラインと4Gを除くほとんどの接続タイプで、ほぼ均等に分布しています。オフラインのシナリオでは、サービスワーカーがウェブサイトを提供していると推測されます。そのため、接続形態によるダウンロードの遅延がなく、良い成績がもっとも多くなっています。

4Gについて明確な結論を出すのは困難ですが、おそらく4G+接続がデスクトップ機器のインターネット・アクセス方法として使用されているのではないかと推測できます。この仮定が妥当であれば、Webフォントや画像が大量にキャッシュされ、CLS測定にプラスの影響を与える可能性があります。

396. https://web.dev/i18n/ja/optimize-cls/

Core Web Vitals: First Input Delay

First Input Delay (FID) は、ユーザーが最初にインタラクションを行ってから、ブラウザがそのインタラクションに応答できるまでの時間を測定します。FIDは、ウェブサイトがどれだけインタラクティブであるかを示す良い指標です。

デバイス別FID

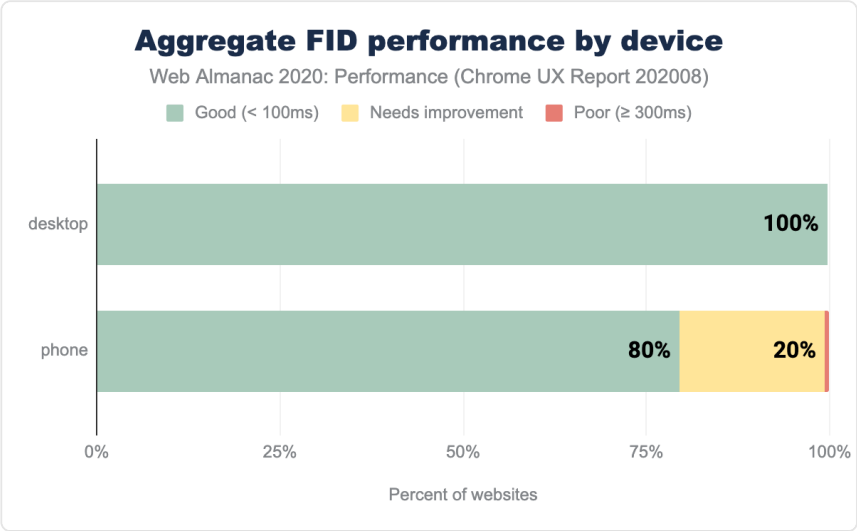


図9.9. デバイスタイプ別に分割されたFIDのパフォーマンスを集約したもの。

良いスコアがこれほど高い割合でウェブサイトに分布しているのは、比較的珍しいことです。デスクトップでは、サイトの分布の75パーセンタイルに基づいて、100%のサイトがFIDの速いタイミングを報告しており、インタラクションの遅延を経験している人の数は非常に少ないことを意味しています。

モバイルでは、80%のサイトが「良い」と評価されています。これは、デスクトップに比べてCPUの能力が低いこと、モバイルでのネットワークの遅延（スクリプトのダウンロードと実行が遅れる）、さらにバッテリー効率と温度の制限によりモバイル機器のCPUの能力が制限されていることが原因と考えられます。これらはすべて、FIDなどのインタラクティビティ指標に直接影響します。

地理的な位置によるFID

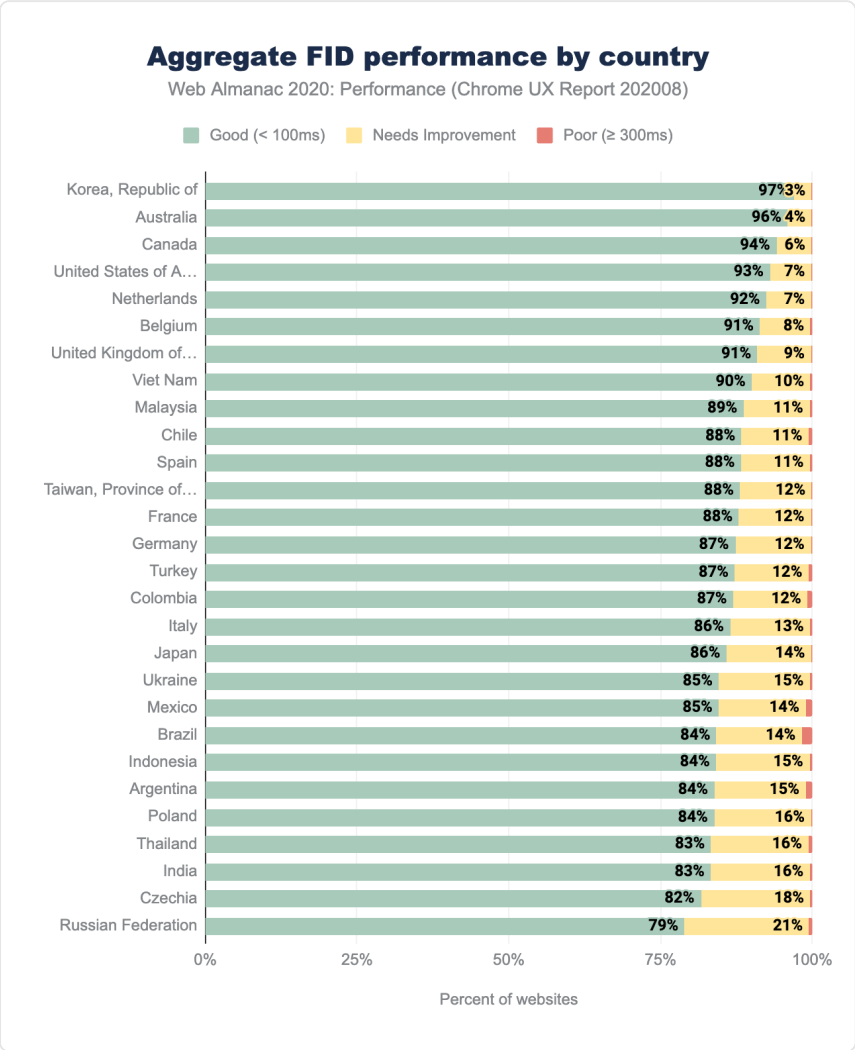


図9.10. 国別に分けられたFIDのパフォーマンスを集計。

FIDスコアの地理的分布は、先に紹介したデバイス集計表の結果を裏付けるものです。最悪の場合、79%のWebサイトが良好なFIDを持っていますが、トップは97%と印象的で、韓国が再びリードしています。興味深いことに、CLSとLCPのランキングで上位を占めていたチェコ、ポーランド、ウクライナ、ロシアなどの国々が、このランキングでは最下位になっています。

ここでも、その理由を推測できますが、本当のところを確かめるには、さらなる分析が必要です。FIDがJavaScriptの実行能力と相関していると仮定すると、より高性能なデバイスがより高価で高級品として扱われる国ではFIDランキングが、低くなる可能性があります。 ポーランドを例に挙げると、米国に比べてiPhoneの価格がもっとも高い[397]国のひとつであり、相対的に低い賃金[398]と相まって、1回の給料ではアップルの主力製品を購入するには不十分です。対照的に、平均的な給料[399]をもらっているオーストラリア人は、1週間分の給料でiPhoneを購入できます。幸いなことに、低評価のウェブサイトの割合はほとんどが0で、1〜2%の読み取り率の例外もあり、インタラクションに対する反応が比較的スピーディであることを指し示しています。

接続タイプ別FID

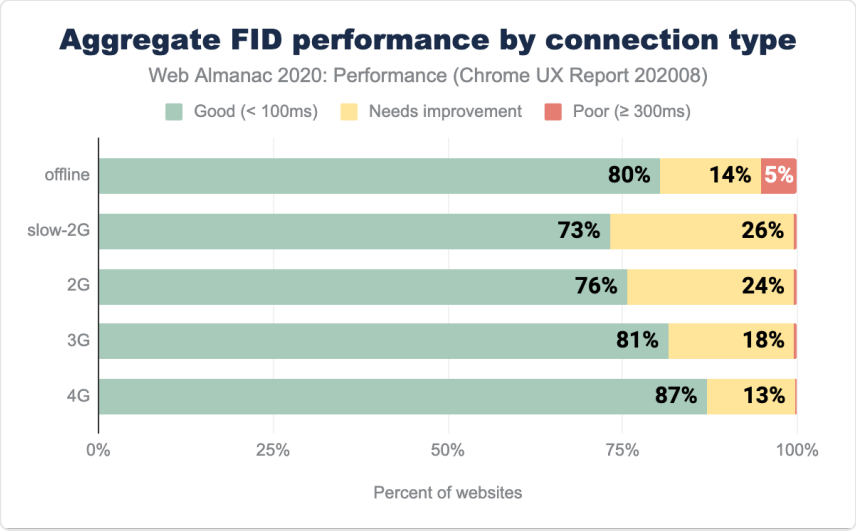


図9.11. 接続タイプ別に分割されたFIDのパフォーマンスを集計したもの。

ネットワークの速度と高速FIDには直接的な相関関係があり、2Gネットワークでは73%、4Gネットワークでは87%となっています。ネットワークが速ければ、スクリプトのダウンロードが速くなり、その結果、解析の開始が速くなり、メインスレッドをブロックするタスクが少なくなります。とくに、評価の低いサイトの割合が5%を超えていないのに、このような結果が出たのは楽観的です。

397. https://qz.com/1106603/where-the-iphone-x-is-cheapest-and-most-expensive-in-dollars-pounds-and-yuan/
398. https://en.wikipedia.org/wiki/List_of_European_countries_by_average_wage#Net_average_monthly_salary
399. https://www.news.com.au/finance/average-australian-salary-how-much-you-have-to-earn-to-be-better-off-than-most/news-story/6fcdde092e87872b9957d2ab8eda1cbd

First Contentful Paint

First Contentful Paint（FCP）は、ブラウザがテキスト、画像、非白キャンバス、SVGコンテンツをレンダリングした最初の時間を測定します。FCPは、サイトが読み込まれる最初の兆候を見るまでの待ち時間を示すもので、体感速度の良い指標となります。

デバイス別FCP

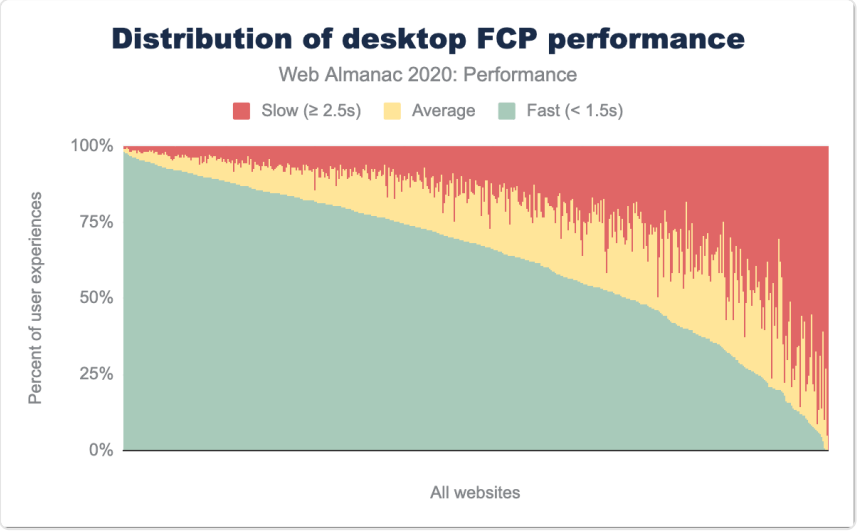


図9.12. デスクトップでのFCPパフォーマンスが「速い」「普通」「遅い」と判定されたウェブサイトの分布。

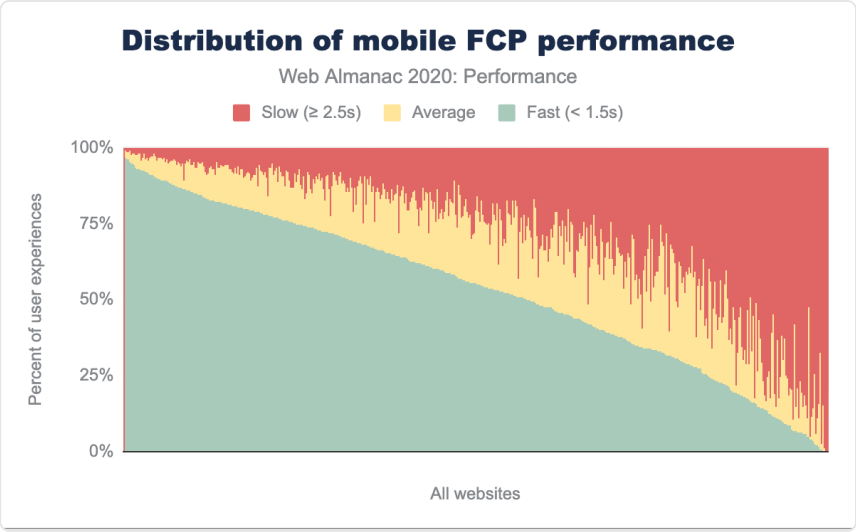


図9.13. モバイルでのFCPパフォーマンスが「速い」「普通」「遅い」と判定されたウェブサイトの分布。

上のグラフでは、FCPの分布をデスクトップとモバイルで分けています。昨年との比較[400]では、平均的なFCP測定値が明らかに少なくなっている一方で、デバイスの種類にかかわらず、速いユーザー体験と遅いユーザー体験の割合が上昇しています。モバイルユーザーがデスクトップユーザーよりも遅いFCPを体験する頻度が高くなるという同じ傾向がまだ見られます。全体的に、ユーザーは平凡な体験ではなく、良い体験や悪い体験をすることが多くなっています。

400. *https://almanac.httparchive.org/ja/2019/performance#fcp-by-device*

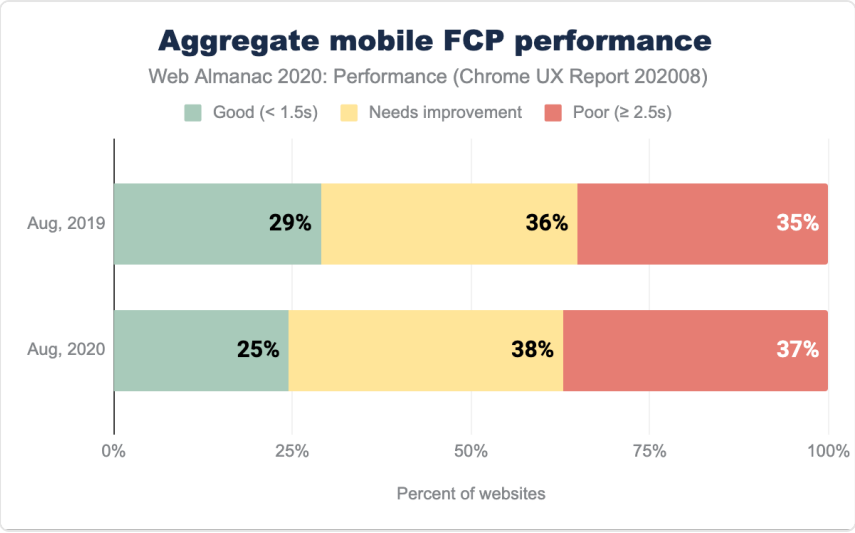


図9.14. FCPモバイルパフォーマンスが良い、改善が必要、悪いとラベル付けされたウェブサイトの分布を2019年と2020年で比較したもの。

モバイルデバイスでのFCPを前年比で比較すると、良い体験が少なく、中程度や悪い体験が多くなっています。75%のWebサイトでは、FCPがあまり良くないものとなっています。このように、理想的ではないFCPが高い割合で読まれていることが、不満やユーザー体験を低下させる原因になっていると推測できます。

たとえば、サーバーの遅延（Time to First Byte（TTFB）（#time-to-first-byte）やRTTなどのいくつかの指標で測定）、JavaScriptリクエストのブロック、カスタムフォントの不適切な処理など、数多くの要因でペイントを遅らせる原因となっています。

地域別FCP

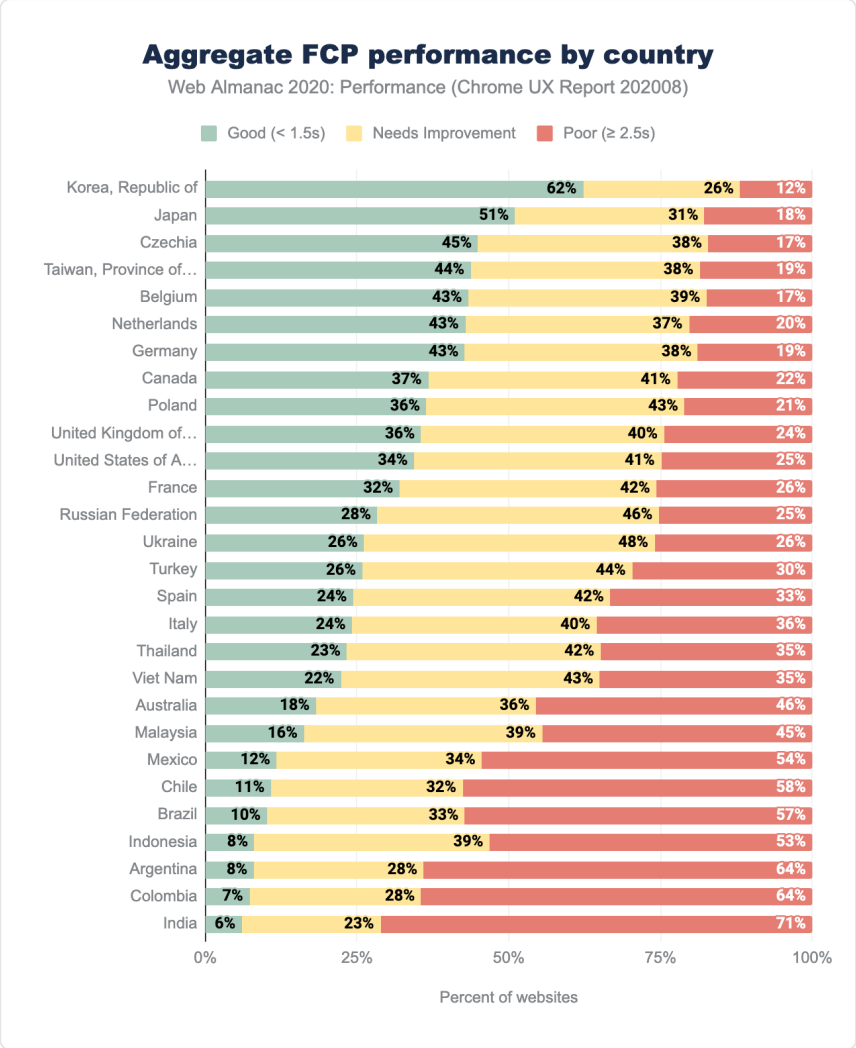


図9.15. 国別に分割されたFCPのパフォーマンスを集計。

分析を掘り下げる前に、2019年の「パフォーマンス」の章では、「良い」と「悪い」の分類のしきい値が2020年とは異なっていたことに触れておきます。2019年には、FCPが1秒以下のサイトは「良い」とされ、FCPが3秒以上のサイトは「悪い」に分類されました。2020年には、これらの範囲は「良い」が1.5秒、「悪い」が2.5秒にシフトしました。

この変化は、分布がより多くの「良い」および「悪い」評価のウェブサイトにシフトすることを意味します。昨年の結果[401]と比較しても、「良い」ウェブサイトと「悪い」ウェブサイトの割合が上昇していることから、その傾向を観察できます。速いウェブサイトの割合が高い上位10地域は、チェコとベルギーが加わり、米国と英国が落ちたことで、2019年とは比較的変わりません。韓国は、高速FCPを報告しているウェブサイトの62%でトップに立ち、昨年から約2倍になっています（これも、結果の再分類によるものと思われます）。ランキング上位の他の国でも、良い体験をしたという報告が倍増しています。

平凡（「改善が必要」）なサイトの割合が少なくなる一方で、パフォーマンスの低いFCPサイトの数は増加しており、とくにラテンアメリカや南アジア地域のランキング下位で顕著になっています。

繰り返しになりますが、TTFBの読み取り率が悪いなど、FCPに悪影響を及ぼす理由はいくつかありますが、必要な文脈がなければ証明することは困難です。たとえば、オーストラリアのような特定の国のパフォーマンスを分析すると、意外にも低い方になります。オーストラリアは、世界でもっともスマートフォンの普及率が高い国の1つであり、モバイルネットワークももっとも高速で、平均所得水準も比較的高い国の1つです。本来であれば、もっと上位にあるべきだと考えられます。しかし、固定回線の遅さ、遅延、CrUXにおけるiOSの不足などを考慮すると、この位置は理にかなっていると言えます。このような例（表面的にしか触れていませんが）では、各国のコンテキストを理解することがいかに難しいかがわかります。

401. *https://almanac.httparchive.org/ja/2019/performance#fcp-by-geography*

接続タイプ別FCP

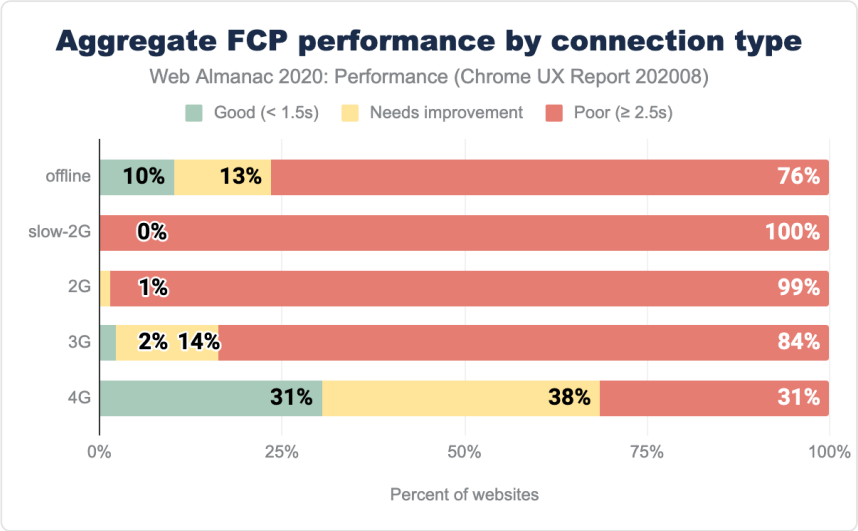


図9.16. 接続タイプ別に分割されたFCPのパフォーマンスを集計したもの。

他の指標と同様に、FCPは接続速度に影響されます。3Gでは、良いと評価している体験者は2%しかいませんが、4Gでは31%となっています。FCPのパフォーマンスは理想的な状態ではありませんが、2019年から改善されている[402]部分もあり、これもやはり良いと悪いのカテゴライズを変えたことが影響しているのかもしれません。良いウェブサイトと悪いウェブサイトの割合が同じように上昇し、中程度（「改善が必要」）のサイト体験の数を狭めていることがわかります。

この傾向は、デジタルデバイドの拡大を示しており、遅いネットワークや潜在的に性能の低いデバイスでの体験は一貫して悪化しています。低速回線でのFCPの改善は、TTFBの改善に直結します。これは、接続タイプ別TTFBパフォーマンス集計表でも確認できます。

ホスティングプロバイダー[403]やCDN[404]の選択は、速度に連鎖的な影響を与えます。最速の配信に基づいてこれらの決定を行うことは、とくに低速のネットワークでは、FCPとTTFBの改善に役立ちます。FCPはフォントの読み込み時間にも大きく影響されるため、Webフォントのダウンロード中にテキストが表示されるようにする[405]ことも価値のある戦略です（とくに、低速の接続ではこれらのリソースを取得するのにコストがかかる場合）。

「オフライン」の統計を見ると、かなりの数のFCP問題もネットワークの種類とは相関して

402. https://almanac.httparchive.org/ja/2019/performance#fcp-by-effective-connection-type
403. https://ismyhostfastyet.com/
404. https://www.cdnperf.com/
405. https://web.dev/font-display/

いないことが推測できます。このカテゴリーでは、その言葉が真実であれば得られるであろう、大きな利益は観察されません。レンダリングは、JavaScriptを取得することでそれほど遅延しないように見えますが、解析と実行に影響されます。

Time to First Byte

Time to First Byte（TTFB）とは、最初のHTMLリクエストが行われてから、最初のバイトがブラウザに戻ってくるまでの時間のことです。リクエストが迅速に処理されないと、描画だけでなくリソースの取得も遅れるため、すぐに他のパフォーマンス指標にも影響がおよびます。

デバイス別TTFB

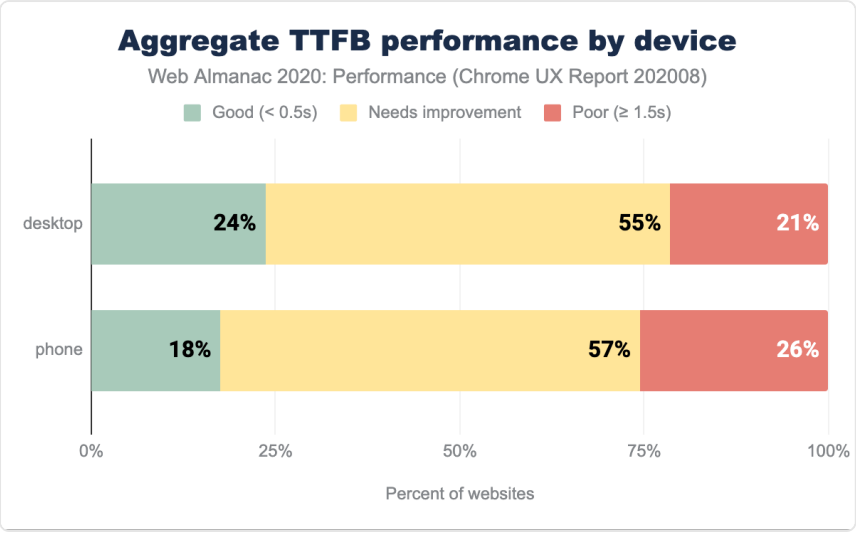


図9.17. デバイスタイプ別に分割されたTTFBのパフォーマンスを集計したもの。

デスクトップで76%のウェブサイトはTTFBが「良くない」と回答しており、モバイルでは83%に上ります。このデータは、パフォーマンス測定や作業のほとんどが、アセットデリバリやサーバサイドの作業ではなく、フロントエンドやビジュアルレンダリングに集中していると仮定した場合、TTFBがいかに見過ごされがちな指標であるかを示していると言えるでしょう。TTFBが高いと、他の数多くのパフォーマンス信号に直接的な悪影響を及ぼすため、今後も対応が必要な分野です。

地域別のTTFB

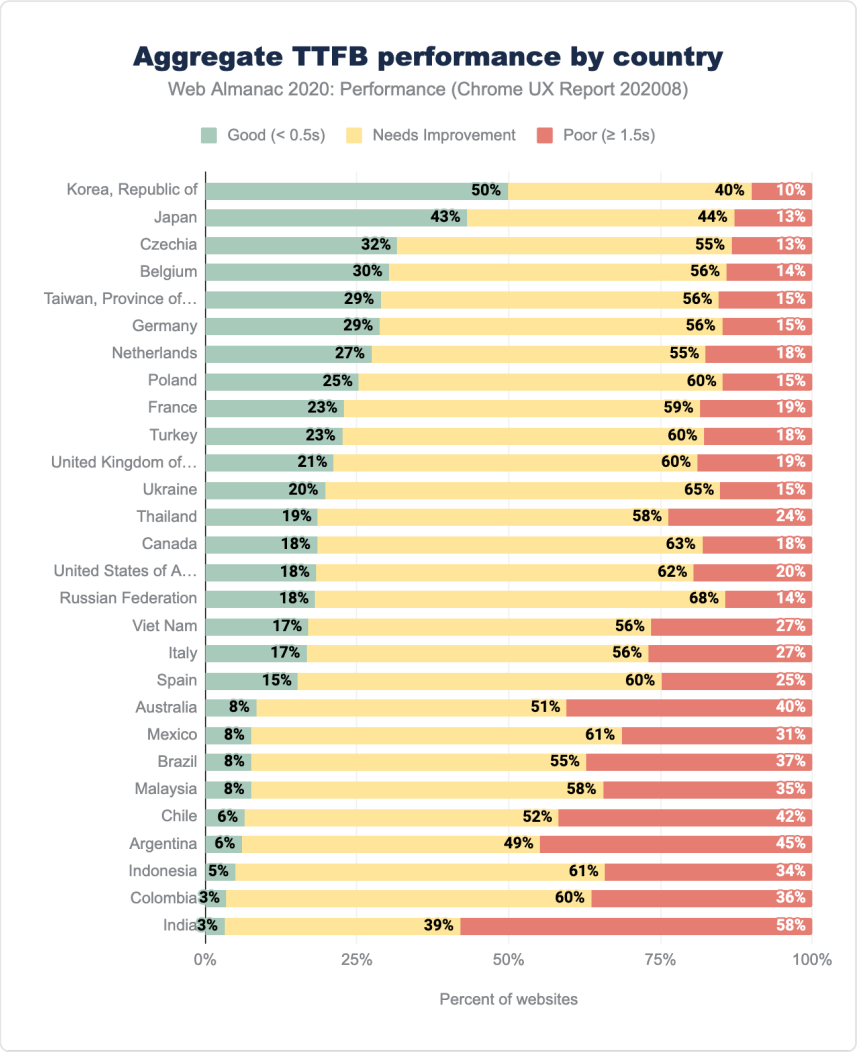


図9.18. 国別に分割されたTTFBのパフォーマンスを集計しています。

今年のTTFBのジオリーディングを2019年の結果[406]と比較すると、やはり高速なウェブサイトが多いことを指摘していますが、FCPと同様に閾値が変更されています。以前は、200ms以下のTTFBは速く、1000ms以上は遅いと考えていました。2020年には、TTFBが500ms以

406. *https://almanac.httparchive.org/ja/2019/performance#ttfb-by-geo*

下を「良い」、1500ms以上を「悪い」としています。このような分類の変更により、韓国では「良い」が36%増加し、台湾では22%増加するなど、大きな変化が見られました。全体的には、アジア太平洋地域や一部のヨーロッパ地域など、同じような地域が上位を占めています。

接続タイプ別のTTFB

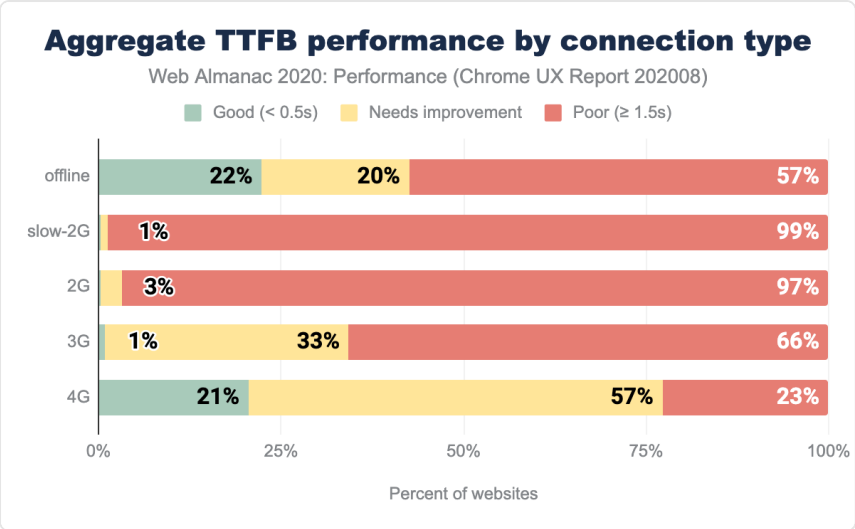


図9.19. TTFBのパフォーマンスを接続タイプ別集計したものです。

TTFBは、ネットワークのレイテンシーと接続タイプの影響を受けます。遅延が大きいほど、また接続速度が遅いほど、TTFBの測定値は悪くなります（上図参照）。高速とされるモバイル接続（4G）でも、TTFBが高速なWebサイトはわずか21%です。4Gの速度以下で速いと分類されたサイトはほとんどありません。

2018年12月～2019年11月の世界のモバイル通信速度[407]を見ると、世界的に見て、モバイル接続は高速ではないことがわかります。それらのネットワーク速度や携帯電話ネットワークの技術標準（5Gなど）は均等でなく、TTFBに影響を与えています。一例として、ナイジェリアのネットワークの地図を見てみましょう[408]国のエリアのほとんどが2Gと3Gのカバー率で、4Gの範囲はほとんどありません。

驚くべきことは、オフラインと4Gの起源の間で、良好なTTFBの結果が比較的同じ数であることです。サービス・ワーカーがいれば、TTFB問題の一部が緩和されることを予想されま

407. https://www.speedtest.net/insights/blog/content/images/2020/02/Ookla_Mobile-Speeds-Poster_2020.png
408. https://www.mobilecoveragemaps.com/map_ng#7/8.744/7.670

すが、この傾向は上のグラフには反映されていません。

パフォーマンス・オブザーバーの使い方

Webサイトやアプリケーションの評価に使用できるユーザー中心の評価基準は、何十種類もあります。しかし、事前に定義された評価基準は、私たちの特定のシナリオやニーズにまったく合わないことがあります。PerformanceObserver API[409]では、User Timing API[410]、Long Task API[411]、Event Timing API[412]やその他のいくつかの低レベルAPI[413]で得られたカスタムメトリックデータを取得できます。たとえば、彼らの協力を得て、ページ間の遷移のタイミングを記録したり、SSR（Server-Side-rendered）アプリケーションの水和を数値化したりすることができました。

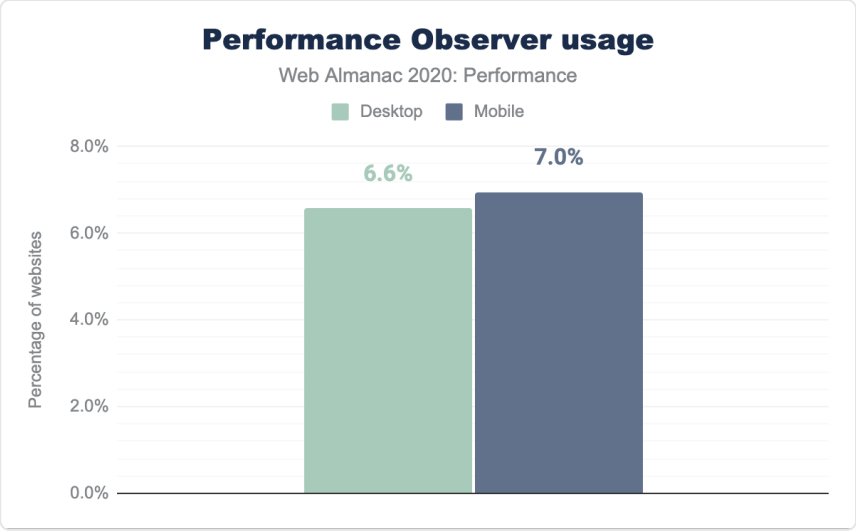


図9.20. パフォーマンス デバイスタイプ別のObserverの使用状況。

上の図では、Performance Observerは、デバイスタイプに応じて追跡サイトの6〜7%で使用されていることを示しています。これらのWebサイトは、低レベルAPIを活用してカスタムメトリクスを作成し、PerformanceObserver APIでそれらを照合して、他のパフォーマンスレポートツールで使用する可能性があります。このような採用率は、定義済みのメトリクスに傾倒する傾向を示しているかもしれませんが（たとえば、Lighthouseから来ている）、比較的ニッチなAPIとしては印象的でもあります。

409. https://developer.mozilla.org/docs/Web/API/PerformanceObserver
410. https://developer.mozilla.org/docs/Web/API/User_Timing_API
411. https://developer.mozilla.org/docs/Web/API/Long_Tasks_API
412. https://web.dev/i18n/ja/custom-metrics/#event-timing-api
413. https://web.dev/i18n/ja/custom-metrics/

結論

ユーザー体験は、スペクトルだけでなく、さまざまな要素に依存しています。劣悪で恵まれない体験を排除せずにパフォーマンスの状態を理解しようとするには、交差的にアプローチしなければなりません。ウェブサイトにアクセスするたび、ストーリーが浮かび上がってきます。個人や国レベルの社会経済的地位によって、購入できるデバイスやインターネットプロバイダーの種類が決まります。私たちが住んでいる場所の地理的条件は、遅延に影響を与え（オーストラリア人はいつもこの痛みを感じています）、経済状況は利用可能な携帯電話ネットワークの範囲を決定します。どんなウェブサイトを見るのか？何のために訪問するのか？コンテキストは、データを分析するだけでなく、すべての人にアクセス可能で高速な体験を提供するために必要な共感と配慮を得るためにも重要です。

新しいCore Web Vitalsのパフォーマンス指標について、表面的には楽観的なシグナルが出ています。Largest Contentful Paintの低速ネットワークでの一貫した劣悪な体験に絞らなければ、少なくとも半分はデスクトップとモバイルデバイスの両方で良好な体験となっています。新しい測定基準は、パフォーマンス問題への取り組みが進んでいることを示唆しているかもしれませんが、「最初の描画」と「最初のバイトまでの時間」に大きな改善が見られないことは憂慮すべきことです。ここでは、Largest Contentful Paintと同じネットワークタイプが、高速接続やデスクトップデバイスと同様にもっとも不利な条件となっています。パフォーマンススコアも、速度の低下を表しています（あるいは、過去に測定したものよりも正確に表現されているかもしれません）。

このデータからわかることは私たちの特権（中・高所得国、高給取り、新しく高性能なデバイス）の複数の側面のために、私たちが経験することのないシナリオ（接続速度の低下など）に対するパフォーマンスを改善するために、投資を続けなければならないということです。また、初期描画（LCPおよびFCP）やアセットデリバリー（TTFB）の高速化についても、まだまだ課題が、あることが明らかになりました。多くの場合、パフォーマンスは本質的にフロントエンドの問題のように感じられますが、バックエンドや適切なインフラの選択によって、多くの大幅な改善が可能です。繰り返しになりますが、ユーザー体験はさまざまな要素に左右されるため、総合的に判断する必要があります。

新しい測定基準は、ユーザー体験を分析するための新しいレンズをもたらしますが、既存のシグナルを忘れてはなりません。しかし、既存のシグナルを忘れてはなりません。もっとも改善が必要な分野で針を動かし、もっとも恵まれていない人々の体験にポジティブな変化をもたらすことに集中しましょう。高速でアクセス可能なインターネットは人間の権利です。

著者

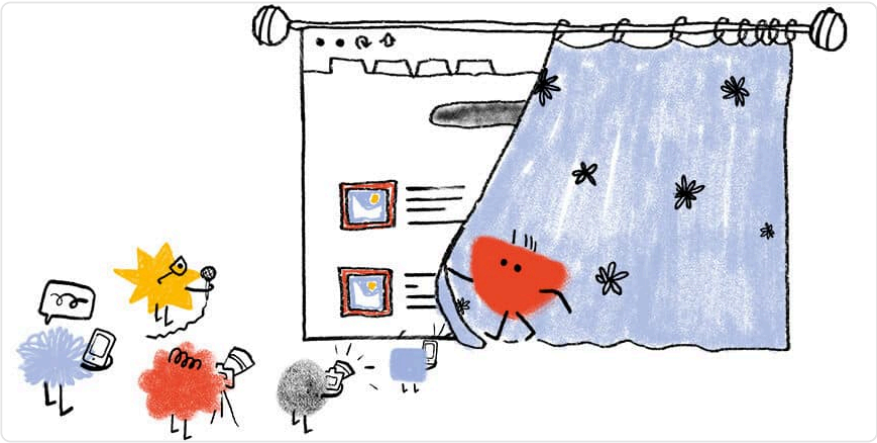
Karolina Szczur

@fox　thefoxis

Karolinaは、Calibre[414]のプロダクトデザインリーダーとして、最も包括的なスピードモニタリングプラットフォームの開発に取り組んでいます。また、パフォーマンスに関するニュースやリソースをお届けするPerformance Newsletter[415]の編集も担当しています。また、パフォーマンスがユーザー体験に与える影響について、頻繁に記事を書いています[416]。

414. *https://calibreapp.com/*
415. *https://perf.email/*
416. *https://calibreapp.com/blog/category/web-platform*

部 II 章 10
プライバシー



Yana Dimova によって書かれた。
Laurent Devernay Satyagraha によってレビュ。
Yana Dimova と Max Ostapenko による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

Web Almanacのこの章では、Web上のプライバシーの現状を概観します。このトピックは、最近人気が高まっており、ユーザー側の意識も高まっています。ガイドラインの必要性は、さまざまな規制によって満たされてきました（ヨーロッパのGDPR[417]、ブラジルのLGPD[418]、カリフォルニアのCCPA[419]など）。これらは、データ処理業者の説明責任とユーザーに対する透明性を高めることを目的としています。本章では、さまざまな技術を用いたオンライントラッキングの普及状況と、ウェブサイトにおけるCookie同意バナーやプライバシーポリシーの採用率について説明します。

417. *https://gdpr-info.eu/*
418. *https://lgpd-brazil.info/*
419. *https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5*

オンライントラッキング

第三者のトラッカーは、ユーザーデータを収集してユーザーの行動のプロファイルを構築し、広告目的で収益化します。このことは、ウェブ上のユーザーにプライバシーに関する懸念を抱かせ、その結果、さまざまなトラッキング保護手段が登場することになりました。しかし、このセクションでご紹介するように、オンライントラッキングは今でも広く利用されています。プライバシーに悪影響を及ぼすだけでなく、オンライン・トラッキングは環境に多大な影響を与えており[420]、それを避けることはパフォーマンスの向上[421]につながります。

サードパーティーCookieとフィンガープリンティングを用いた、もっとも一般的なサードパーティートラッキングを検証します。オンライントラッキングはこの2つの手法に限らず、既存の対策を回避するための新しい手法が次々と生まれています。

サードパーティートラッカー

当社では、WhoTracksMe[422]のトラッカーリストを使用して、潜在的なトラッカーにリクエストを発行するウェブサイトの割合を調べています。次の図に示すように、少なくとも1つの潜在的なトラッカーがおよそ93%のウェブサイトに存在することがわかりました。

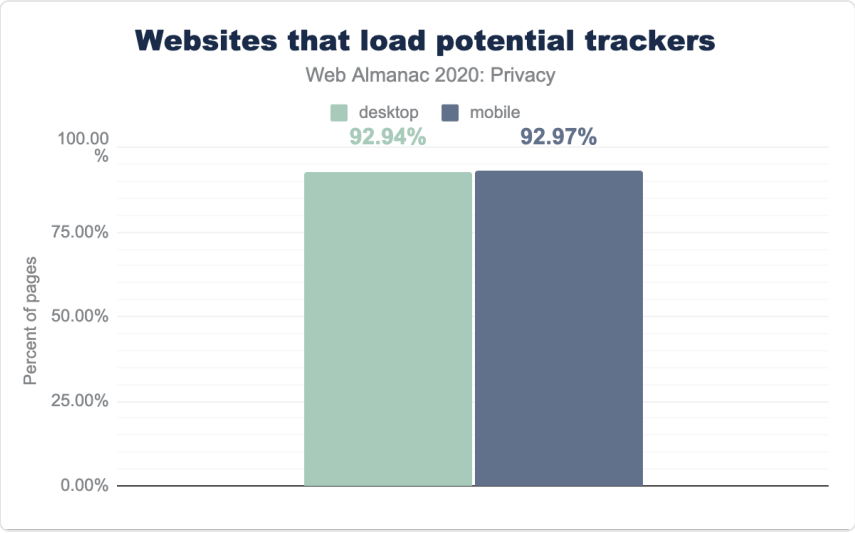


図10.1. 少なくとも1つの潜在的なトラッカーを含むウェブサイト

私たちは、もっとも広く使われているトラッカーを調べ、もっとも人気のある10種類のトラ

420. https://gerrymcgovern.com/calculating-the-pollution-cost-of-website-analytics-part-1/
421. https://x.com/fr3ino/status/1000166112615714816
422. https://whotracks.me/

ッカーの普及率をプロットしました。

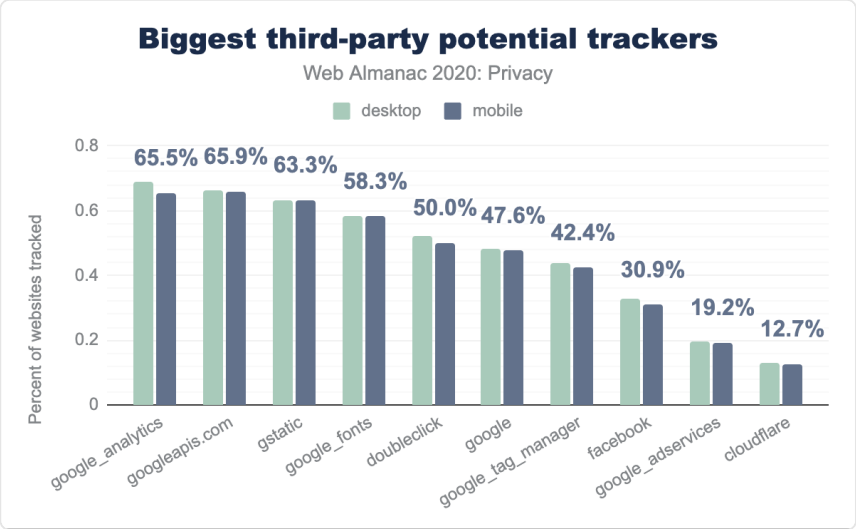


図10.2. トップ10の潜在的なトラッカー

オンライントラッキング市場の最大手は、間違いなくGoogleです。同社の8つのドメインがトラッカー候補のトップ10に入っており、少なくとも70%のウェブサイトで利用されています。次いでFacebook、Cloudflarと続きますが、後者はホスティングサイトとしての人気を反映していると思われます。

WhoTracksMeのトラッカーリストには、トラッカーが所属するカテゴリーも定義されています。CDNとホスティングサイトを統計から除外すると、これらのサイトはトラッキングを行わないか、少なくともそれが主な機能ではないと仮定すると、トップ10の見方が少し変わります。

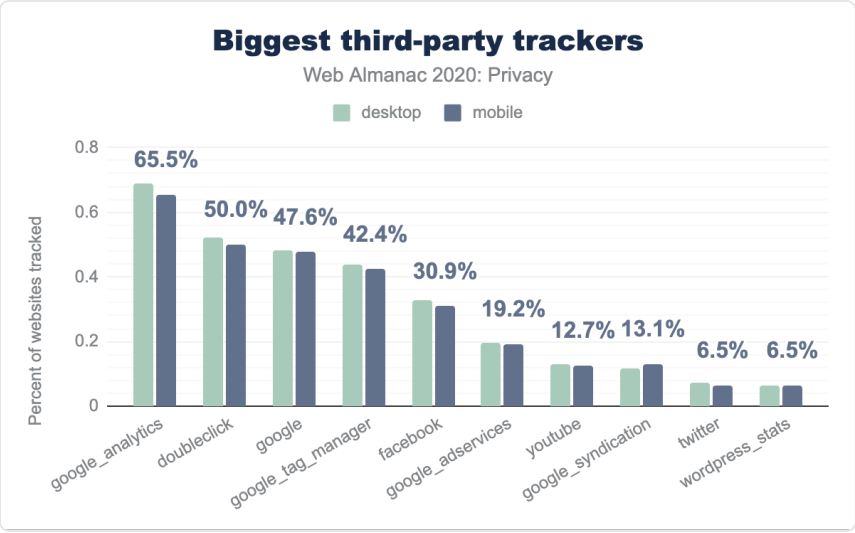


図10.3. トップ10トラッカー

ここでは、トップ10のドメインのうち7つをGoogleが占めています。次の図は、潜在的なトラッカーの数が多い100件について、カテゴリー別の分布を示したものです。

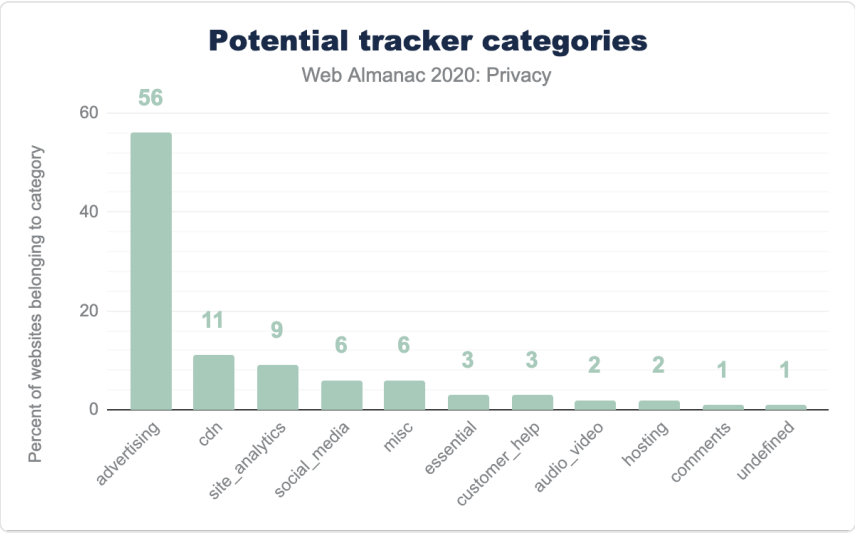


図10.4. ポテンシャルトラッカー100選のカテゴリー

もっとも人気のあるトラッカーの60%近くが広告関連である。これは、オンライン広告市場

の収益性がトラッキングの量に関係していると認識されているためと考えられる。

Cookies

WebサイトでHTTPのレスポンスヘッダーに設定されているもっとも一般的なCookieを、その名前とドメイン別に調べました。

ドメイン	Cookieの名称	ウェブサイト
doubleclick.net	test_cookie	24%
facebook.com	fr	10%
youtube.com	VISITOR_INFO1_LIVE	10%
youtube.com	YSC	10%
doubleclick.net	IDE	9%
doubleclick.net	unknown	9%
youtube.com	GPS	9%
doubleclick.net	unknown	8%
google.com	NID	6%
doubleclick.net	unknown	6%

図10.5. デスクトップサイトのトップCookie

ドメイン	Cookieの名称	ウェブサイト
doubleclick.net	test_cookie	32%
doubleclick.net	IDE	21%
facebook.com	fr	10%
youtube.com	VISITOR_INFO1_LIVE	10%
youtube.com	YSC	10%
google.com	NID	10%
youtube.com	GPS	8%
doubleclick.net	DSID	7%
yandex.ru	yandexuid	6%
yandex.ru	i	6%

図10.6. モバイルサイトのトップCookie

ご覧のとおり、Googleのトラッキングドメイン「doubleclick.net」は、モバイルクライアントでは約4分の1、デスクトップクライアントでは全Webサイトの3分の1のWebサイトにCookieを設定しています。ここでも、デスクトップクライアントでもっとも人気のある10個のCookieのうち9個、モバイルでは10個のうち7個がGoogleのドメインによって設定されています。これは、HTTPヘッダーを介して設定されたCookieのみをカウントしているため、Cookieが設定されているウェブサイトの数の下限となります。

フィンガープリンティング

もう1つの広く使われているトラッキング技術は、フィンガープリンティングです。これは、ユーザーに固有の「フィンガープリント」を作成することを目的として、ユーザーに関するさまざまな種類の情報を収集するものです。ウェブ上では、さまざまな種類のフィンガープリンティングがトラッカーによって使用されています。ブラウザフィンガープリントでは、ユーザーのブラウザに固有の特性を使用します。これは、追跡するための変数の数が十分に多ければ、他のユーザーがまったく同じブラウザを設定している可能性はかなり低くなるという事実に基づいています。今回の調査では、ブラウザ・フィンガープリントをサービスとして提供しているFingerprintJS[423]ライブラリの存在を調べました。

423. https://fingerprintjs.com/

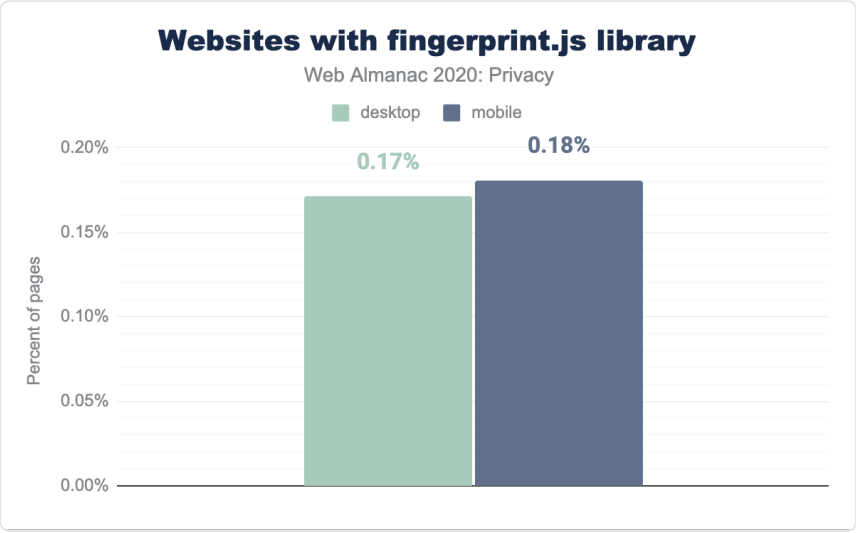


図10.7. FingerprintJSを使用しているサイト

このライブラリはウェブサイトのごく一部にしか存在していませんが、フィンガープリンティングの永続的な性質から、わずかな使用でも大きな影響を与える可能性があります。さらに、フィンガープリンティングの試みはFingerprintJSだけではありません。他のライブラリ、ツール、ネイティブコードでもこの目的を果たすことができますので、これは一例に過ぎません。

コンセントマネジメント・プラットフォーム

Cookie同意バナーは今や一般的になっています。このバナーは、Cookieに対する透明性を高め、しばしばユーザーがCookieの選択肢を指定できるようにします。多くのウェブサイトが独自のCookie・バナーの実装を選択していますが、最近では*Consent Management Platforms*と呼ばれる第三者のソリューションが登場しています。このプラットフォームは、ウェブサイトがさまざまなタイプのCookieに対するユーザの同意を収集するための簡単な方法を提供します。4.4%のウェブサイトが同意管理プラットフォームを使用して、デスクトップ・クライアントでのCookieの選択を管理しており、モバイル・クライアントでは4.0%となっています。

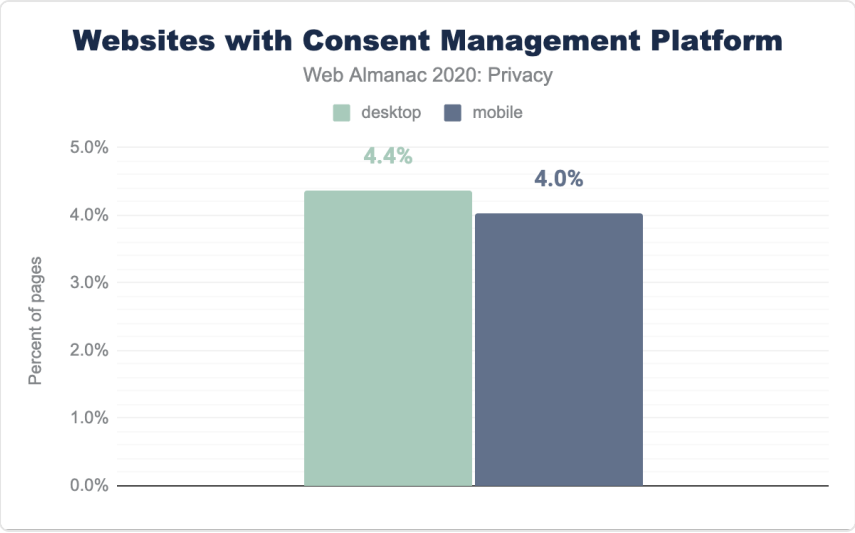


図10.8. コンセント・マネジメント・プラットフォームを使用しているウェブサイト

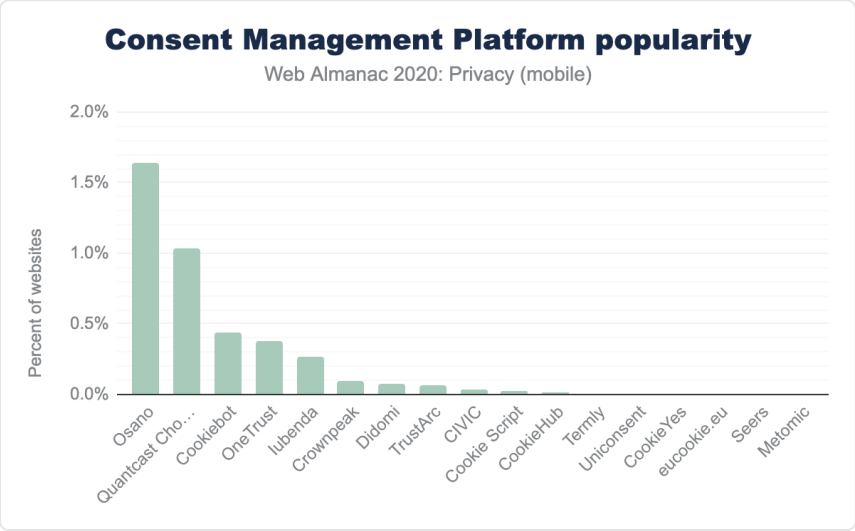


図10.9. コンセント・マネジメント・プラットフォームの普及

さまざまな同意管理ソリューションの人気度を見ると、OsanoとQuantcast Choiceが主要なプラットフォームです。

IABヨーロッパのトランスペアレンシー・コンセント・フレームワーク

IABヨーロッパ（Interactive Advertising Bureau）は、ヨーロッパのデジタルマーケティングと広告に関する団体です。彼らは、デジタル広告の好みに関するユーザーの同意を得るために、GDPRに準拠したソリューションとして、Transparency Consent Framework[424]（TCF）を提案しました。この実装は、パブリッシャーと広告主の間で消費者の同意に関するコミュニケーションを行うための業界標準を提供するものです。

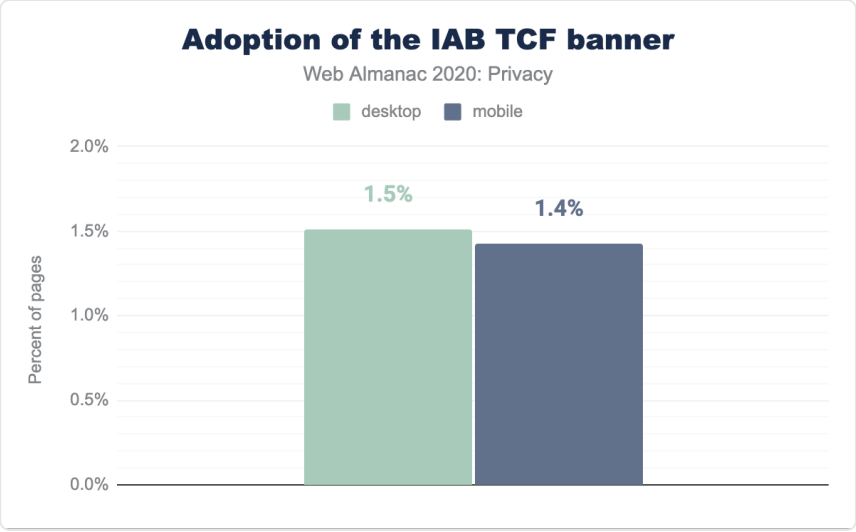


図10.10. TCFバナーの採用率

今回の結果は、TCFバナーがまだ「業界標準」ではないことを示していますが、正しい方向への一歩と言えるでしょう。IABヨーロッパの主なターゲットはヨーロッパのパブリッシャーであり、私たちの活動はグローバルなものであることを考えると、デスクトップクライアントで1.5%、モバイルでは1.4%のウェブサイトで採用されていることは悪くないと思います。

プライバシーポリシー

プライバシーポリシーは、法的義務を果たし、データ収集方法についてユーザーへの透明性を高めるために、ウェブサイトで広く使用されています。クロールでは、訪問したウェブサイトにプライバシーポリシーのテキストが存在することを示すキーワードを検索しました。

424. https://iabeurope.eu/transparency-consent-framework/

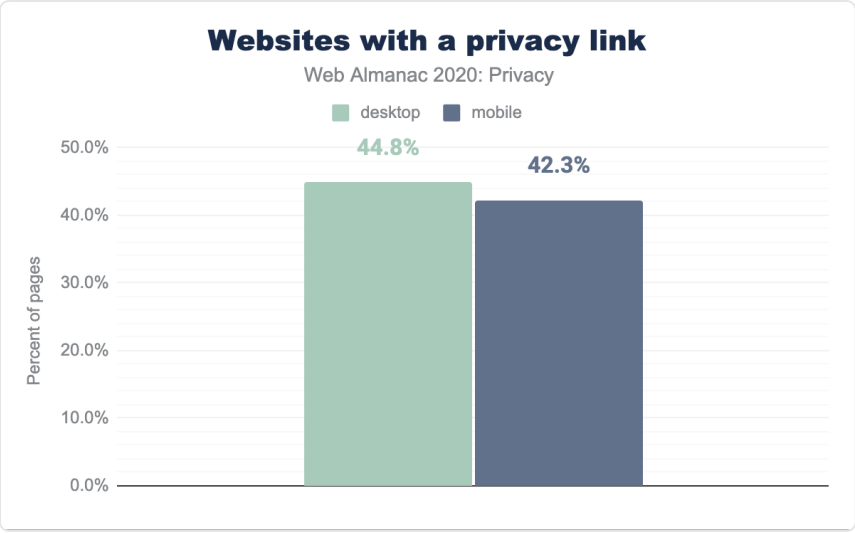


図10.11. プライバシーポリシーのあるサイト

その結果、データセットに含まれるウェブサイトの約半数がプライバシーポリシーを記載していることがわかり、ポジティブな印象を受けました。しかし研究によると大多数のインターネットユーザーは、プライバシーポリシーを読もうとはせず、読んだとしても、ほとんどのプライバシーポリシーの文章が長くて複雑なため、理解できていないことがわかっています。しかし、プライバシーポリシーが存在することは、正しい方向への一歩と言えるでしょう。

結論

本章では、デスクトップとモバイルの両方のクライアントにおいて、第三者によるトラッキングが依然として顕著であり、Googleがもっとも多くのウェブサイトをトラッキングしていることを示しました。コンセント管理プラットフォームが使用されているウェブサイトの割合は少ないですが、多くのウェブサイトでは、独自のCookie同意バナーが実装されています。

最後に、約半数のウェブサイトがプライバシーポリシーを掲載しており、データ処理の方法についてユーザーに大きな透明性をもたらしています。これは間違いなく一歩前進ですが、まだやるべきことはたくさんあります。今回の分析以外にも、プライバシーポリシーは読みにくく理解しにくいものであり、Cookieの同意バナーはユーザーを操作して同意させるものであることがわかっています。

ウェブが真にユーザーを尊重するためには、プライバシーは後回しにするのではなく、発想

の一部にしなければなりません。この点において、規制は良いことであり、世界的にプライバシー規制が増加していることは心強いことです。プライバシーバイデザイン[425]は、最低限の法的要件を満たし、金銭的なペナルティを避けるためにポリシーやツールを導入するのではなく、規範となるべきものです。

著者

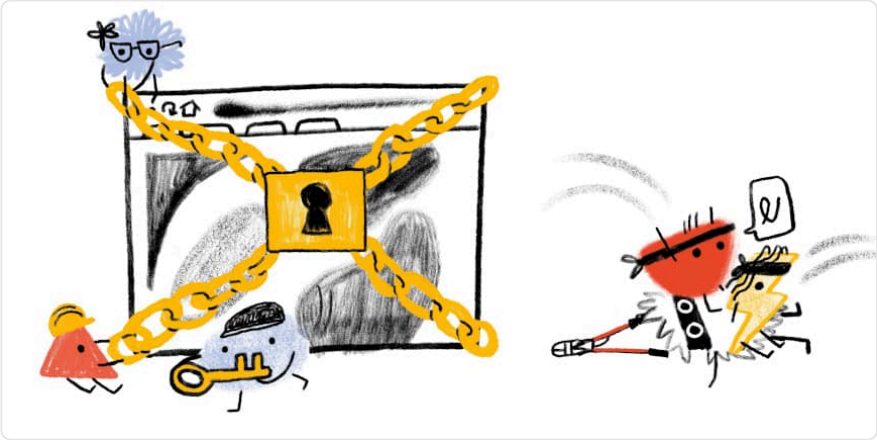
Yana Dimova

ydimova

Yana Dimovaは、ベルギーのKUルーヴェン大学の博士課程に在籍し、プライバシーとウェブセキュリティの研究を行っています。

425. https://ja.wikipedia.org/wiki/%E3%83%97%E3%83%A9%E3%82%A4%E3%83%90%E3%82%B7%E3%83%BC%E3%83%90%E3%82%A4%E3%83%87%E3%82%B6%E3%82%A4%E3%83%B3

部 II 章 11
セキュリティ



Tom Van Goethem、 Nurullah Demir と Barry Pollard によって書かれた。
Caleb Queern と Edmond W. W. Chan によってレビュ。
Tom Van Goethem と Nurullah Demir による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

様々な意味で、2020年はとんでもない年になりました。世界的な大流行の結果、私たちの日常生活は大きく変化しました。友人や家族と直接会う代わりに、多くの人が連絡を取り合うためソーシャルメディアに頼らざるを得なくなった。これは、ユーザーがオンラインで過ごす時間が増えた結果、多くの異なるアプリケーション[426]のトラフィック量が大幅に増加[427]したことを意味します。これはまた、様々なプラットフォーム上でオンラインで共有する情報を安全に保つために、セキュリティがこれまで以上に重要になっていることを意味しています。

私たちが日常的に利用しているプラットフォームやサービスの多くは、クラウドベースのAPI、マイクロサービス、そして最も重要なのはWebアプリケーションに至るまで、Webリソースに強く依存しています。これらのシステムの安全性を維持することは容易なことでは

426. https://arxiv.org/pdf/2008.10959.pdf
427. https://dl.acm.org/doi/pdf/10.1145/3419394.3423621

ありません。幸いなことに、過去10年間、Webセキュリティの研究は継続的に進歩してきました。一方で、研究者は新しいタイプの攻撃を発見し、その結果をより広いコミュニティと共有して意識を高めています。一方で、多くのエンジニアや開発者は、攻撃の結果を防止または最小化するため使用できる適切なツールやメカニズムのセットをウェブサイト運営者に提供するために、たゆまぬ努力を続けています。

本章では、ウェブ上のセキュリティの現状を探ります。様々なセキュリティ機能の採用を詳細かつ大規模に分析することで、ユーザーを守りたいという動機から、ウェブサイトの所有者がこれらのセキュリティメカニズムを適用する様々な方法についての洞察を得ます。

私たちは、個々のウェブサイトにおけるセキュリティメカニズムの採用だけではありません。ウェブサイトを構築するために使用される技術スタックなどのさまざまな要因が、セキュリティヘッダーの普及にどのように影響を与え、全体的なセキュリティを向上させるかを分析しています。さらに、ウェブサイトのセキュリティを確保するためには、多くの異なる側面をカバーする全体的なアプローチが必要であると言ってもよいでしょう。そのため、我々は、広く使われている様々なウェブ技術のパッチ適用方法など、他の側面も評価しています。

方法論

本章では、ウェブ上での様々なセキュリティメカニズムの採用状況を報告します。この分析は、560万のデスクトップドメインと630万のモバイルドメインのホームページについて収集したデータに基づいています。特に明記されていない限り、報告されている数字はモバイルのデータセットの方がサイズが大きいため、モバイルのデータセットに基づいています。ほとんどのウェブサイトが両方のデータセットに含まれているため、結果として得られる測定値はほぼ同じです。2つのデータセットの間に有意な差がある場合は、本文中で報告するか、または図から明らかになるようにしています。データがどのように収集されたかについての詳細は、方法論を参照してください。

トランスポートのセキュリティ

昨年は、WebサイトでのHTTPSの普及が続いています。トランスポート層の安全性を確保することは、Webセキュリティの最も重要な部分の1つです。Webサイト用にダウンロードしたリソースが転送中に変更されていないこと、また自分が思っているWebサイトとの間でデータを転送していることを確信できなければ、Webサイトのセキュリティについての確信は事実上無効となります。

WebトラフィックをHTTPSに移行し、最終的にはHTTPを非セキュアなものとしてマーク[428]

428. *https://www.chromium.org/Home/chromium-security/marking-http-as-non-secure*

するようになったのは、ブラウザがセキュアなコンテキストへの強力な新機能[429]（人参）を制限する一方で、暗号化されていない接続が使用された場合にユーザーに表示される警告を増加させるためです（棒）。

86.90%

図11.1. モバイルでHTTPSを利用するリクエストの割合。

この努力が実を結び、現在ではデスクトップでのリクエストの87.70%、モバイルでのリクエストの86.90%がHTTPSで提供されています。

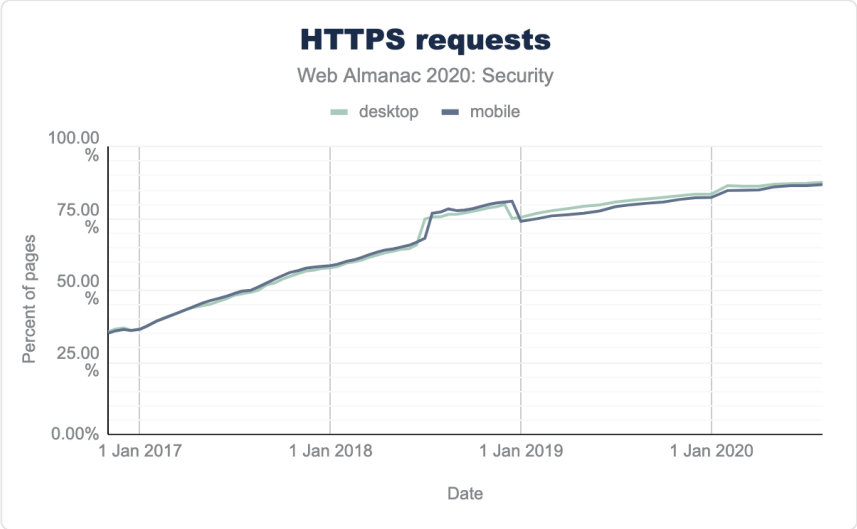


図11.2. HTTPSを使用したリクエストの割合
(出典。HTTP Archive[430])

この目標の最後に到達したとき懸念されるのは、ここ数年の印象的な成長の「平準化」が顕著になっていることです。残念ながらインターネットのロングテールは、古いレガシーサイトが維持管理されておらず、HTTPSでの運用ができない可能性があることを意味しています。

429. https://developer.mozilla.org/docs/Web/Security/Secure_Contexts/features_restricted_to_secure_contexts
430. https://httparchive.org/reports/state-of-the-web#pctHttps

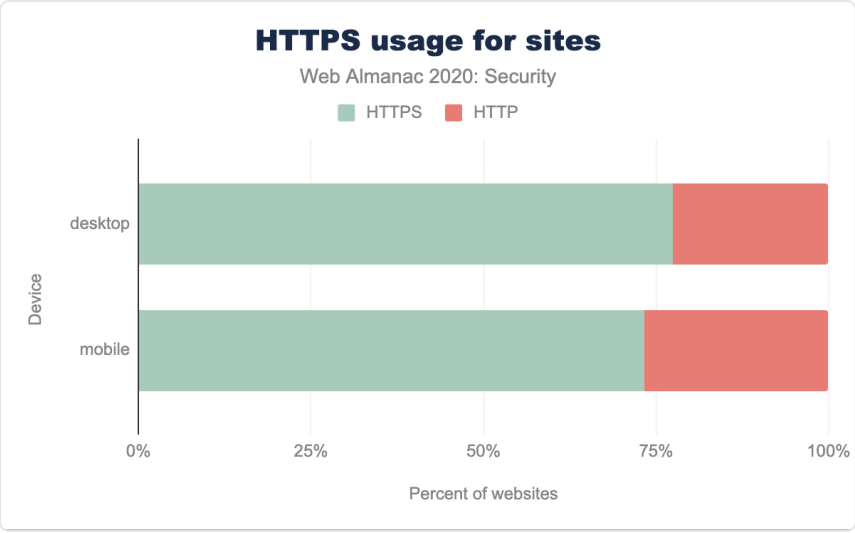


図11.3. サイトのHTTPS使用量

リクエストの量が多いことは励みになりますが、これらのリクエストは、サードパーティリクエストやGoogleアナリティクス、フォント、または広告などのサービスに支配されていることがよくあります。ウェブサイト自体が遅れていることもありますが、現在では73%から77%のサイトがHTTPSで提供されており、使用率が向上していることがわかります。

プロトコルのバージョン

HTTPSが現在では完全に標準となっているため、課題はHTTPSの種類を問わず、基礎となるTLS(Transport Layer Security)プロトコルの安全なバージョンを確実に使用することに移ってきています。TLSのバージョンが古くなったり、脆弱性が発見されたり、計算能力が向上して攻撃が可能になったりすると、TLSのメンテナンスが必要になります。

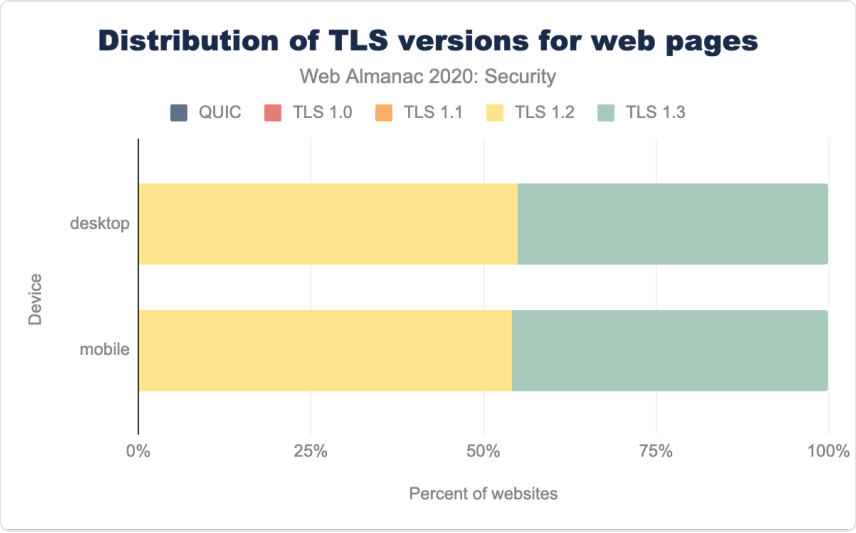


図11.4. サイトのTLSバージョンの使用状況

TLSv1.0の使用率はすでに基本的にはゼロであり、パブリックウェブがより安全なプロトコルを採用していることを励みに見えるかもしれませんが、私たちのクロールがベースになっているChromeを含むすべての主流のブラウザは、これを説明するための大きな方法を行くデフォルトでこれをブロックすることに注意する必要があります。

これらの数字は昨年のプロトコル分析[431]に比べてわずかに改善されており、TLSv1.3の使用率は約5%増加し、それに対応するTLSv1.2は減少しています。これは小さな増加のようで、昨年指摘された比較的新しいTLSv1.3の使用率が高かったのは、大規模なCDNからの初期サポートが原因である可能性が高いと思われます。従ってTLSv1.3の採用にもっと大きな進展があるとすれば、まだTLSv1.2を使用している人は自分自身で管理しているか、まだサポートしていないホスティングプロバイダーで管理している可能性があるため、長い時間が必要になるでしょう。

暗号スイート

TLSの中には、さまざまなレベルのセキュリティで使用できるいくつかの暗号スイートがあります。最高の暗号化方式はforward secrecy[432]鍵交換をサポートしています。これはサーバの鍵が漏洩しても、その鍵を使った古いトラフィックは復号化できないことを意味します。

過去にはTLSの新しいバージョンでは、より新しい暗号のサポートが追加されていました

431. *https://almanac.httparchive.org/ja/2019/security#protocol-versions*
432. *https://ja.wikipedia.org/wiki/Forward_secrecy*

が、古いバージョンを削除することはほとんどありませんでした。これがTLSv1.3がより安全である理由の1つです。人気のあるOpenSSLライブラリは、このバージョンでは5つの安全な暗号のみをサポートしていますが、そのすべてが前方秘匿をサポートしています。これにより、安全性の低い暗号を強制的に使用されるダウングレード攻撃を防ぐことができます。

98.03%

図11.5. Forward secrecyを利用したモバイルサイト

すべてのサイトは本当にforward secrecyを使用すべきであり、デスクトップサイトの98.14%、モバイルサイトの98.03%が前方秘匿暗号を使用しているのは良いことです。

forward secrecyを前提とした場合、暗号化レベルの選択は、鍵サイズが大きいほど破られるまでに時間がかかりますが、その代償として、特に初期接続時の暗号化・復号化にかかる計算量が多くなります。ブロック暗号モード[433]ではGCMを使用し、CBCはパディング攻撃に弱いとされています[434]。

広く使われているAES(Advanced Encryption Standard)では、128ビットと256ビットの暗号化キーサイズが一般的です。ほとんどのサイトでは128ビットで十分ですが、256ビットの方が望ましいでしょう。

433. https://ja.wikipedia.org/wiki/%E6%9A%97%E5%8F%B7%E5%88%A9%E7%94%A8%E3%83%A2%E3%83%BC%E3%83%89
434. https://blog.qualys.com/product-tech/2019/04/22/zombie-poodle-and-goldendoodle-vulnerabilities

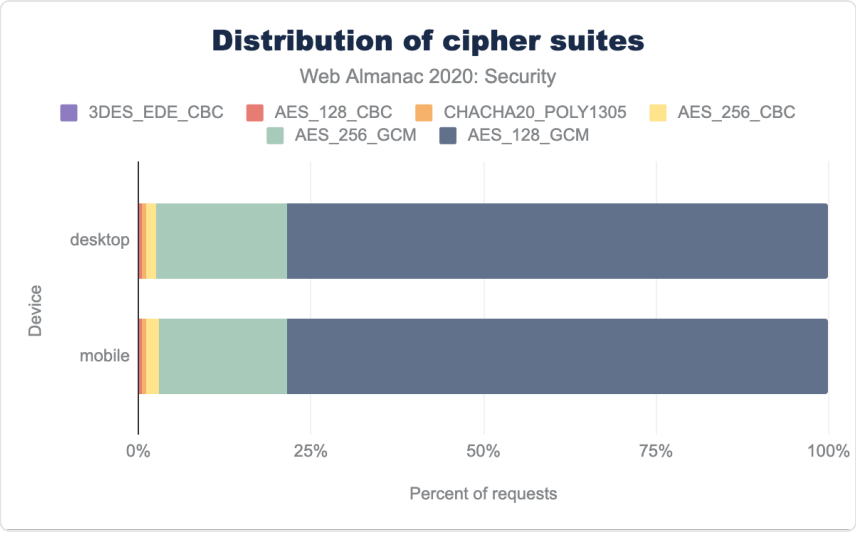


図11.6. 暗号スイートの分布

上記のチャートから、AES_128_GCMが最も一般的で、デスクトップサイトとモバイルサイトの78.4%が使用していることがわかります。AES_256_GCMはデスクトップサイトの19.1%、モバイルサイトの18.5%が使用しており、その他のサイトは古いプロトコルや暗号スイートを使用しているサイトである可能性が高いです。

注意すべき重要な点の1つは、私たちのデータはサイトに接続するためにChromeを実行していることに基づいており、接続には単一のプロトコル暗号を使用するということです。私たちの方法論では、サポートされているプロトコルや暗号スイートの全範囲を見ることはできず、その接続に実際使用されたものだけを見ることができます。この情報については、SSL Pulse from SSL Labs[435]のような他の情報源を見る必要がありますが、現在ほとんどの最新ブラウザが同様のTLS機能をサポートしているため、上記のデータは大多数のユーザーが使用すると予想されるものです。

証明書発行機関

次に、クロールしたサイトが使用しているTLS証明書を発行している認証局(CA)について見ていきます。昨年の章[436]では、このデータに対するリクエストを見ましたが、それはGoogleのような人気のあるサードパーティによって支配されるでしょう（そのメトリックから今年も支配されている）ので、今年はウェブサイトがロードする他のすべてのリクエストではなく、ウェブサイト自体によって使用されるCAを提示するつもりです。

435. https://www.ssllabs.com/ssl-pulse/
436. https://almanac.httparchive.org/ja/2019/security#certificate-authorities

発行者	デスクトップ	モバイル
Let's Encrypt Authority X3	44.65%	46.42%
Cloudflare Inc ECC CA-3	8.49%	8.69%
Sectigo RSA Domain Validation Secure Server CA	8.27%	7.91%
cPanel, Inc. Certification Authority	4.71%	5.06%
Go Daddy Secure Certificate Authority - G2	4.30%	3.66%
Amazon	3.12%	2.85%
DigiCert SHA2 Secure Server CA	2.04%	1.78%
RapidSSL RSA CA 2018	2.01%	1.96%
Cloudflare Inc ECC CA-2	1.95%	1.70%
AlphaSSL CA - SHA256 - G2	1.35%	1.30%

図11.7. ウェブサイト用の証明書発行会社トップ10

Let's Encryptが首位に立っているのを見ても不思議ではありません。無料の証明書と自動化された証明書の組み合わせは、個々のウェブサイトの所有者とプラットフォームの両方で勝者を証明しています。Cloudflareは同様に、2番目と9番目の位置を取っている顧客のための無料の証明書を提供しています。さらに興味深いのは、使用されているECC Cloudflareの発行者です。ECC証明書はRSA証明書よりも小さいので効率的ですが、サポートが万能ではなく、両方の証明書を管理するには余分な労力を必要とすることが多いため、導入が複雑になることがあります。これは、Cloudflareがここで行っているように、CDNまたはホストされているプロバイダーがこれを管理できる場合のメリットです。ECCをサポートしているブラウザ（クロールで使用しているChromeブラウザのようなもの）はECCを使用し、古いブラウザはRSAを使用します。

ブラウザの強制

暗号攻撃から身を守るためには安全なTLS構成を持つことが最も重要ですが、ネットワーク上の攻撃者からウェブユーザを保護するためには、さらなる保護が必要です。例えば、ユーザーがHTTP上の任意のウェブサイトをロードするとすぐに、攻撃者は悪意のあるコンテンツを注入して、例えば他のサイトへのリクエストを行うことができます。

サイトが最も強力な暗号や最新のプロトコルを使用している場合でも、攻撃者はSSLストリッピング攻撃を使用して、被害者のブラウザを騙してHTTPSではなくHTTPで接続していると思わせることができます。さらに適切な保護が施されていないと、ユーザのCookieが最初

のプレーンテキストHTTPリクエストに添付され、攻撃者はネットワーク上でCookieをキャプチャできます。

これらの問題を克服するために、ブラウザはこれを防ぐため有効にできる追加機能を提供しています。

HTTPの厳格なトランスポートセキュリティ

1つ目はHTTP Strict Transport Security(HSTS)で、いくつかの属性からなるレスポンスヘッダーを設定することで簡単に有効化できます。このヘッダーについては、モバイルホームページ内での採用率が16.88%となっています。HSTSを有効にしているサイトのうち、92.82%が有効にしています。つまり、max-age属性（ブラウザがHTTPS上のウェブサイトを何秒で*訪問*するかを決定する）の値が0よりも大きくなっています。

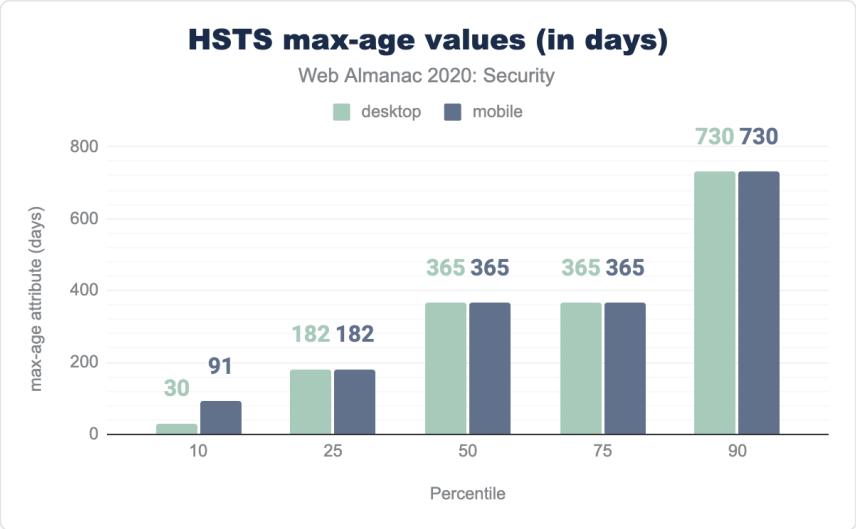


図11.8. HSTSの `max-age` の値（日）。

この属性の異なる値を見てみると、大多数のWebサイトがかなりの将来にHTTPSを利用することになると確信していることがよくわかります： 半数以上のWebサイトが少なくとも1年間はHTTPSを利用するようブラウザに要求しています。

1,000,000,000,000,000 years

図11.9. 知られている最大のHSTSのmax-age。

あるウェブサイトでは、自分のサイトがHTTPSで利用できるようになるまでの期間について少し熱心に考えすぎて、1,000,000,000,000,000,000,000,000年に翻訳される `max-age` 属性値を設定してしまったかもしれません。皮肉なことに、一部のブラウザはこのような大きな値をうまく扱えず、実際にはそのサイトのHSTSを無効にしてしまうのです。

HSTSディレクティブ	*デスクトップ*	*モバイル*
`max-age`有効	*92.21%*	*92.82%*
`includeSubdomains`	*32.97%*	*32.14%*
`preload`	*16.02%*	*16.56%*

図11.10. HSTSディレクティブの使用法。

昨年](../2019/security#http-strict-transport-security)と比較して、他の属性の採用が増加していることは心強いことです。HSTSポリシーのうち、 `includeSubdomains` が32.14%、 `preload` が16.56%となっています。

Cookies

セキュリティの観点から見ると、クロスサイトリクエストにCookieを自動的に含めることは、いくつかのクラスの脆弱性の主犯と見ることができます。もしウェブサイトが適切な保護機能を持っていない場合（例えば、状態を変更するリクエストに固有のトークンを要求するなど）、Cross-Site Request Forgery[437] (CSRF)攻撃にさらされる可能性があります。例えば攻撃者はユーザーが気づかないうちに、訪問者のパスワードを変更するために、バックグラウンドでPOSTリクエストを発行するかもしれません。ユーザーがログインしている場合、ブラウザは通常、このようなリクエストに自動的にCookieを含めます。

他にも、Cross-Site Script Inclusion[438]のように、クロスサイトリクエストにCookieを含めることに依存する攻撃がいくつかあります。(XSSI)や、XS-Leaks[439]の脆弱性クラスの様々な手法を利用することができます。さらに、ユーザの認証はCookieを介してのみ行われることが多いため、攻撃者はCookieを取得することでユーザになりすますことができます。これは中間者攻撃(MITM)で、ユーザを騙して安全ではないチャネルで認証を行うことができます。あるいは、クロスサイトスクリプティング(XSS)の脆弱性を悪用することで、攻撃者はDOMを通じて `document.cookie` にアクセスすることでCookieを漏洩させることができます。

Cookieがもたらす脅威から身を守るために、ウェブサイトの開発者は、Cookie[440]に設定でき

437. *https://owasp.org/www-community/attacks/csrf*
438. *https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-lekies.pdf*
439. *https://xsleaks.dev/*
440. *https://developer.mozilla.org/docs/Web/HTTP/Cookies*

る3つの属性、`HttpOnly`、`Secure`、および`SameSite`を利用できます。1つ目はJavaScriptからのアクセスを防ぐもので、攻撃者がXSS攻撃でCookieを盗むことを防ぐことができます。`Secure`属性が設定されているCookieは、安全なHTTPS接続でのみ送信され、MITM攻撃での盗用を防げます。

最近導入された`SameSite`属性は、クロスサイトコンテキストでのCookieの送信方法を制限するために使用できます。属性には3つの値があります。`None`, `Lax`, `Strict` です。`SameSite=None`のCookieはすべてのクロスサイトリクエストに送信され、`Lax`のCookieはユーザーがリンクをクリックして新しいページに移動したときなどのナビゲートリクエストにのみ送信されます。最後に、`SameSite=Strict`属性を持つCookieはファーストパーティのコンテキストでのみ送信されます。

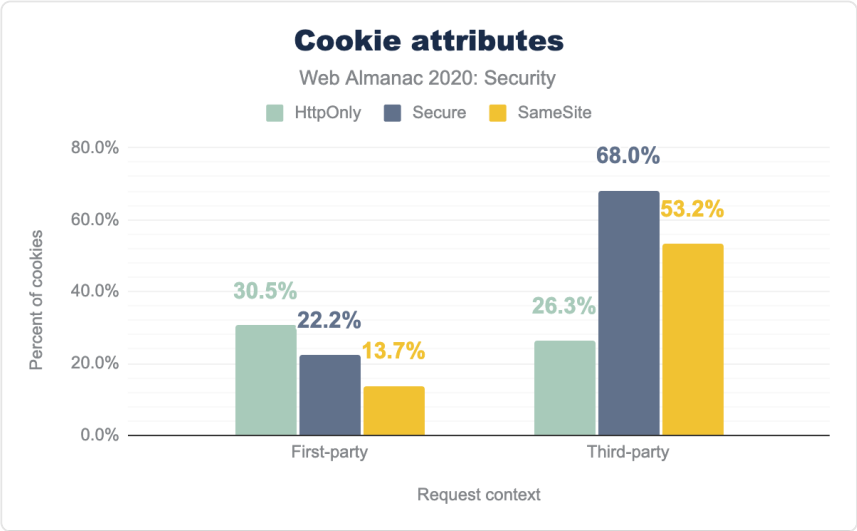


図11.11. Cookie属性。

2,500万枚のファーストサイトCookieと1億1,500万枚のサードパーティCookieに基づいた結果から、Cookie属性の使用は設定されたコンテキストに強く依存することがわかります。Cookieの`HttpOnly`属性の使い方は、ファーストパーティのCookie(30.5%)とサードパーティのCookie(26.3%)の両方で同様であることがわかる。

しかし、`Secure`属性と`SameSite`属性では大きな違いが見られます。ファーストパーティのコンテキストで設定されたすべてのCookieの22.2%に`Secure`属性が存在するのに対し、モバイルホームページのサードパーティのリクエストで設定されたすべてのCookieの68.0%がこのCookie属性を持っています。興味深いことに、デスクトップページでは、サードパーティのCookieの35.2%しかこの属性を持っていませんでした。

`SameSite` 属性については、昨年(../2019/security#samesite)の時点では0.1%しかこの属性が設定されていなかったのに対し、利用率が大幅に増加していることがわかります。2020年8月時点では、ファーストパーティCookieの13.7%、サードパーティCookieの53.2%に `SameSite` 属性が設定されていることが確認されています。

おそらく、この採用率の大きな変化は、Chromeがデフォルトオプションとして `SameSite=Lax` を設定したことに関連していると思われます。このことは、SameSite属性に設定されている値をより詳細に見ることで確認できます。ファーストパーティのクッキーの場合、その割合は48.0%と低くなっていますが、それでも重要な意味があります。クローラーはウェブサイトにログインしないため、ユーザーの認証に使用されるクッキーが異なる可能性があることに注意が必要です。

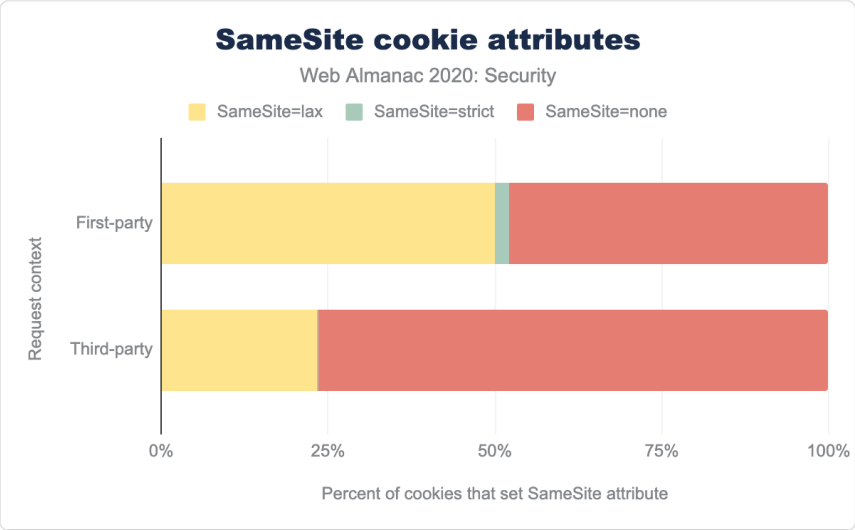


図11.12. 同じサイトのCookie属性。

Cookieを保護するために使用できる追加の仕組みとして、Cookieの名前の前に `__Secure-` または `__Host-` を付けることがあります。これら2つの接頭辞を持つCookieは `Secure` 属性が設定されている場合にのみブラウザに保存されます。後者は追加の制限を課しており、`Path` 属性を `/` に設定する必要があり、`Domain` 属性の使用ができません。これは攻撃者がセッション固定攻撃を実行しようとしてCookieを他の値で上書きすることを防いでいます。

これらの接頭辞の使用量は比較的少なく、`__Secure-` 接頭辞を使用して設定されたファーストパーティCookieは合計で4,433個（0.02%）、`__Host-` 接頭辞を使用して設定されたファーストパーティCookieは1,502個（0.01%）でした。サードパーティのコンテキストで設定されたCookieの場合、プレフィックス付きCookieの相対的な数と似ています。

コンテンツに含まれる

最近のウェブアプリケーションには、JavaScriptライブラリからビデオプレイヤー、外部プラグインに至るまで、様々なサードパーティ製コンポーネントが含まれています。セキュリティの観点から信頼されていない可能性のあるコンテンツをウェブページに含めることは、悪意のあるJavaScriptが含まれている場合のクロスサイトスクリプティングなど、さまざまな脅威をもたらす可能性があります。このような脅威から身を守るために、ブラウザにはコンテンツが含まれるソースを制限したり、含まれるコンテンツに制限を課したりするためのメカニズムがいくつかあります。

コンテンツセキュリティポリシー

どのオリジンがコンテンツの読み込みを許可しているかをブラウザに示すための主要なメカニズムの1つが、`Content-Security-Policy` (CSP)レスポンスヘッダーです。数多くのディレクティブを通して、ウェブサイト管理者はコンテンツをどのように含めるかを細かく制御できます。例えば、`script-src`ディレクティブは、どのオリジンからスクリプトをロードできるかを示します。全体では、CSPヘッダーが全ページの7.23%に存在していることがわかりましたが、モバイルページでCSPの採用率が4.73%であった昨年から53%も増加しています。

ディレクティブ	デスクトップ	モバイル
`upgrade-insecure-requests`	61.61%	61.00%
`frame-ancestors`	55.64%	56.92%
`block-all-mixed-content`	34.19%	35.61%
`default-src`	18.51%	16.12%
`script-src`	16.99%	16.63%
`style-src`	14.17%	11.94%
`img-src`	11.85%	10.33%
`font-src`	9.75%	8.40%

図11.13. CSPポリシーで使用される最も一般的なディレクティブ。

興味深いことにCSPポリシーで最も一般的に使用されているディレクティブを見ると、最も一般的なディレクティブは`upgrade-insecure-requests`であり、安全でないスキームに

含まれるコンテンツは、同じホストへの安全なHTTPS接続を介してアクセスすべきであることをブラウザへ知らせるために使用されています。

例えば、`<img src="http://example.com/foo.png">`は通常、安全ではない接続で画像を取得しますが、`upgrade-insecure-requests`ディレクティブがあると、自動的にHTTPS(`https://example.com/foo.png`)で取得します。

これは、ブラウザが混在するコンテンツをブロックする際に特に役立ちます。HTTPSで読み込まれるページでは、`upgrade-insecure-requests`ディレクティブがないと、HTTPからのコンテンツがブロックされてしまいます。このディレクティブはコンテンツを壊す可能性が低く実装が簡単なので、コンテンツセキュリティポリシーの良い出発点となりで、他のディレクティブに比べてこのディレクティブの採用率はかなり高くなるでしょう。

どのソースからコンテンツを含めることができるかを示すCSPディレクティブ（`*-src`ディレクティブ）は、採用率がかなり低くなっています: デスクトップページで提供されたCSPポリシーの18.51%とモバイルページで提供された16.12%に過ぎません。この理由の1つは、Web開発者がCSPの採用における多くの課題に直面していることにあります[441]。厳格なCSPポリシーは、XSS攻撃を阻止する以上の大きなセキュリティ上の利点を提供できますが、不明確に定義されたポリシーは特定の有効なコンテンツの読み込みを妨げる可能性があります。

ウェブ開発者がCSPポリシーの正しさを評価できるようにするため、非強制的な代替手段も存在し、`Content-Security-Policy-Report-Only`応答ヘッダーでポリシーを定義することで有効にできます。このヘッダーの普及率はかなり低く、デスクトップおよびモバイルページの0.85%です。しかし、これはしばしば移行用のヘッダーであることに注意すべきです。そのため、この割合はCSPの使用への移行を意図しており、限られた時間だけReport-Onlyヘッダーを使用しているサイトを示している可能性が高い。

441. *https://wkr.io/publication/raid-2014-content_security_policy.pdf*

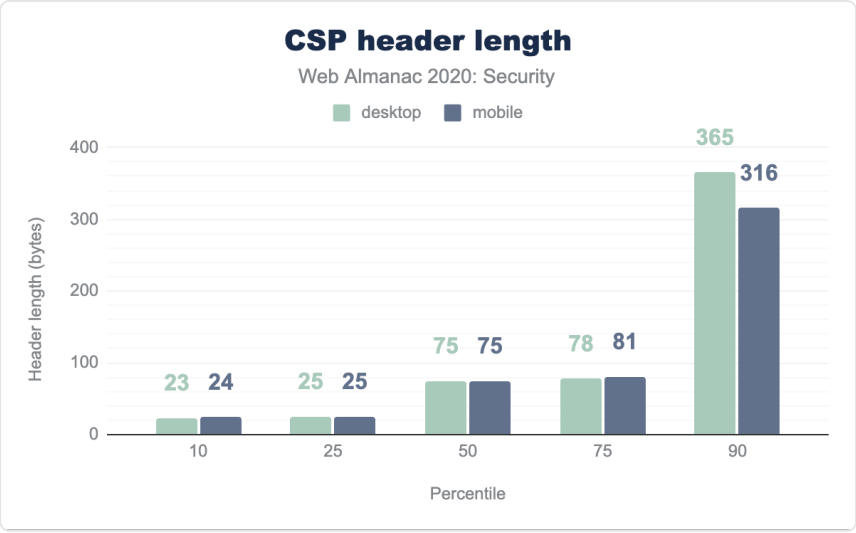


図11.14. CSPヘッダーの長さ。

全体的に、`Content-Security-Policy`応答ヘッダーの長さは非常に限られている: ヘッダーの値の長さの中央値は75バイトである。これは主に、頻繁に使用される短い単一目的のCSPポリシーによるものである。例えば、デスクトップページで定義されているポリシーの24.64%は、`upgrade-insecure-requests`ディレクティブのみを持っている。

最も一般的なヘッダー値は、デスクトップページで定義された全ポリシーの29.44%を占め、`block-all-mixed-content; frame-ancestors 'none'; upgrade-insecure-requests;`となっています。このポリシーはページがフレーム化されるのを防ぎ、セキュアプロトコルへのアップグレード要求を試み、失敗した場合はコンテンツをブロックします。

22,333

図11.15. 観測された最長のCSPのバイト数。

その反対側では、観測された最長のCSPポリシーは22,333バイトでした。

Origin	デスクトップ	モバイル
https://www.google-analytics.com	1.50%	1.46%
https://www.googletagmanager.com	1.04%	1.07%
https://fonts.googleapis.com	0.99%	0.99%
https://www.youtube.com	1.02%	0.91%
https://fonts.gstatic.com	0.95%	0.95%
https://www.google.com	0.95%	0.94%
https://connect.facebook.net	0.89%	0.83%
https://stats.g.doubleclick.net	0.72%	0.70%
https://www.facebook.com	0.66%	0.65%
https://www.gstatic.com	0.54%	0.57%

図11.16. CSPポリシーで最も頻繁に許可されるホスト。

コンテンツの読み込みが許可される外部のオリジンは、意外にも、サードパーティコンテンツが最も頻繁に含まれるオリジンと一致しています。CSPポリシーの `*-src` 属性で定義されている最も一般的な10のオリジンは、すべてGoogle（アナリティクス、フォント、広告）とFacebookにリンクできます。

403

図11.17. CSPで観測された許可されたホストの最大数。

あるサイトでは、含まれているすべてのコンテンツがCSPによって許可されることを保証するために、そのポリシーで403の異なるホストを許可することにしました。もちろん、これはセキュリティ上の利点をせいぜい限界にします。特定のホストがCSP bypasses[442]を許可するかもしれないからです。

サブリソースの完全性

多くのJavaScriptライブラリやスタイルシートはCDNから含まれています。その結果、CDN

442. https://webappsec.dev/assets/pub/csp_acm16.pdf

が侵害されたり、攻撃者が頻繁に含まれているライブラリを置き換える別の方法を見つけたりすると、悲惨な結果になる可能性があります。

漏洩したCDNの影響を制限するために、ウェブ開発者はサブリソース整合性(SRI)メカニズムを利用できます。期待される内容のハッシュからなる`integrity`属性を`<script>`と`<link>`要素に定義できます。ブラウザは、取得したスクリプトやスタイルシートのハッシュを属性で定義されたハッシュと比較し、 一致するものがある場合にのみコンテンツを読み込みます。

ハッシュは3つの異なるアルゴリズムで計算できます。SHA256、SHA384、SHA512です。最初の2つが最も頻繁に使用されています。モバイルページではそれぞれ50.90%と45.92%です（デスクトップページでも同様です）。現在のところ、3つのハッシュアルゴリズムはすべて安全に使用できると考えられています。

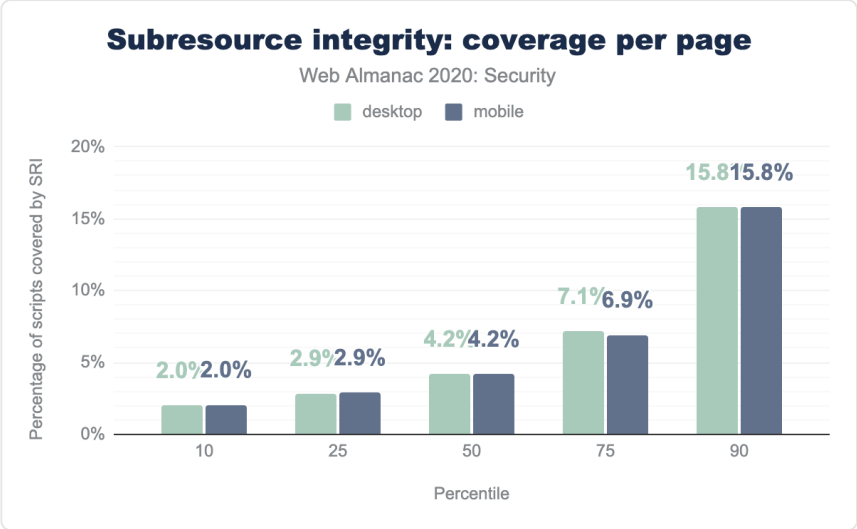


図11.18. サブリソースの完全性：ページごとのカバレッジ。

デスクトップページの7.79%では、少なくとも1つの要素にintegrity属性が含まれており、モバイルページでは7.24%となっています。この属性は主に`<script>`要素で使用されます。整合性属性を持つ全ての要素の72.77%はscript要素に含まれていました。

少なくとも1つのスクリプトがSRIで保護されているページをさらに詳しく見ると、これらのページのスクリプトの大部分が完全性属性を持っていないことがわかります。ほとんどのサイトでSRIで保護されているスクリプトは20個中1個以下でした。

ホスト	デスクトップ	モバイル
cdn.shopify.com	44.95%	45.72%
code.jquery.com	14.05%	13.38%
cdnjs.cloudflare.com	11.45%	10.47%
maxcdn.bootstrapcdn.com	5.03%	4.76%
stackpath.bootstrapcdn.com	4.96%	4.74%

図11.19. SRIで保護されたスクリプトが含まれる最も一般的なホスト。

SRIで保護されたスクリプトが含まれている最も人気のあるホストを見ると、その採用を後押ししているいくつかの要因が見えてきます。例えば、サブリソースの完全性で保護されているスクリプトのほぼ半分は `cdn.shopify.com` からのもので、これはShopify SaaSが顧客のためにデフォルトで有効にしているためである可能性が高い。

SRIで保護されたスクリプトが含まれている上位5ホストの残りは、3つのCDNで構成されています。jQuery[443]、cdnjs[444]、Bootstrap[445]です。これら3つのCDNのすべてが、例のHTMLコードにIntegrity属性を持っているのは、おそらく偶然ではありません。

特徴的なポリシー

ブラウザは無数のAPIや機能を提供しているが、その中にはユーザの体験やプライバシーを損なうものもある。レスポンスヘッダー `Feature-Policy` を通して、ウェブサイトはどの機能を使いたいか、あるいはもっと重要なことにどの機能を使いたくないかを示すことができます。

同様に、 `<iframe>` 要素に `allow` 属性を定義することで、埋め込まれたフレームが使用を許可する機能を決定することもできます。例えば、 `autoplay` ディレクティブを使って、ウェブサイトはページが読み込まれたときにフレーム内の動画の再生を自動的に開始するかどうかを指定できます。

443. https://code.jquery.com/
444. https://cdnjs.com/
445. https://www.bootstrapcdn.com/

ディレクティブ	デスクトップ	モバイル
encrypted-media	78.83%	78.06%
autoplay	47.14%	48.02%
picture-in-picture	23.12%	23.28%
accelerometer	23.10%	23.22%
gyroscope	23.05%	23.20%
microphone	8.57%	8.70%
camera	8.48%	8.62%
geolocation	8.09%	8.40%
vr	7.74%	8.02%
fullscreen	4.85%	4.82%
sync-xhr	0.00%	0.21%

図11.20. フレーム上のフィーチャーポリシーディレクティブの普及率。

レスポンスヘッダー `Feature-Policy` の採用率は、デスクトップページでは0.60%、モバイルページでは0.51%とかなり低い。一方、デスクトップページでは、1320万フレームのうち11.8%でFeature Policyが有効になっていました。モバイルページでは、1390万フレームのうち10.8%が `allow` 属性を含んでいました。

この章の以前のバージョンでは、フレームの合計数と `allow` 属性を持つフレームの割合の値が間違っていました。詳細については、このGitHub PR[446] を参照してください。

iframeの機能ポリシーで最もよく使われているディレクティブを見ると、これらは主にフレームが動画を再生する方法を制御するために使われていることがわかります。例えば、最も一般的なディレクティブである `encrypted-media` は、DRMで保護された動画を再生するために必要なEncrypted Media Extensions APIへのアクセスを制御するために使用されます。フィーチャーポリシーを持つフレームの起源として最も多いのは `https://www.facebook.com` と `https://www.youtube.com` でした (デスクトップページのフィーチャーポリシーを持つフレームの49.87%と26.18%でした)。

446. https://github.com/HTTPArchive/almanac.httparchive.org/pull/3912

Iframeサンドボックス

信頼できないサードパーティをiframeに含めることで、そのサードパーティはページ上で様々な攻撃を試みることができます。例えば、トップページをフィッシングページに誘導したり、偽のアンチウイルス広告を表示したポップアップを起動したりできます。

iframeの `sandbox` 属性は、埋め込まれたウェブページの機能を制限し、その結果、攻撃を開始する機会を制限するために使用することができます。広告や動画などのサードパーティコンテンツを埋め込むことはウェブ上では一般的なことなので、これらの多くが `sandbox` 属性によって制限されていることは驚くべきことではありません。デスクトップページのiframeの18.3%が `sandbox` 属性を持っているのに対し、モバイルページでは21.9%となっています。

この章の以前のバージョンでは、 `sandbox` 属性を持つフレームの割合の値が間違っていました。詳細については、このGitHub PR[447] を参照してください。

ディレクティブ	デスクトップ	モバイル
`allow-scripts`	99.97%	99.98%
`allow-same-origin`	99.64%	99.70%
`allow-popups`	83.66%	89.41%
`allow-forms`	83.43%	89.22%
`allow-popups-to-escape-sandbox`	81.99%	87.22%
`allow-top-navigation-by-user-activation`	59.64%	69.45%
`allow-pointer-lock`	58.14%	67.65%
`allow-top-navigation`	21.38%	17.31%
`allow-modals`	20.95%	17.07%
`allow-presentation`	0.33%	0.31%

図11.21. フレーム上のサンドボックスディレクティブの出現率。

iframeの `sandbox` 属性が空の値を持つ場合、これは最も制限の多いポリシーになります：埋め込まれたページはJavaScriptコードを実行できず、フォームを送信できず、ポップアップも作成できません。

447. https://github.com/HTTPArchive/almanac.httparchive.org/pull/3912

このデフォルトのポリシーは、さまざまなディレクティブを使って細かく緩和することができます。最もよく使われるディレクティブである`allow-scripts`は、デスクトップページ上のすべてのサンドボックスポリシーの99.97%に存在し、埋め込まれたページがJavaScriptコードを実行できるようにします。もう1つは、事実上すべてのサンドボックスポリシーに存在する`allow-same-origin`というディレクティブで、埋め込みページがその起源を保持し、例えば、その起源で設定されたCookieにアクセスすることを許可します。

興味深いことに、Feature Policyとiframe sandboxは両方ともかなり高い採用率を持っていますが、それらが同時に発生することはほとんどありません：iframeの0.04%だけが`allow`と`sandbox`の両方の属性を持っています。おそらくこれは、iframeがサードパーティのスクリプトによって作成されているからだと思われます。フィーチャーポリシーは主にサードパーティの動画を含むiframeに追加されますが、`sandbox`属性は主に広告の機能を制限するために使用されます。デスクトップページの`sandbox`属性を持つiframeの56.40%は`https://googleads.g.doubleclick.net`を元にしています。

攻撃の阻止

現代のウェブアプリケーションは、さまざまなセキュリティ上の脅威に直面しています。例えば、ユーザの入力を不適切にエンコードしたり、サニタイズしたりすると、クロスサイトスクリプティング（XSS）の脆弱性が発生する可能性があり、これは10年以上も前からウェブ開発者を悩ませてきた問題です。ウェブアプリケーションがますます複雑になり、新しいタイプの攻撃が発見されるにつれ、さらに多くの脅威が迫ってきています。例えば、XS-Leaks[448]は、ウェブアプリケーションが返すユーザ固有の動的なレスポンスを利用することを目的とした新しい攻撃のクラスです。

例えば、ウェブメールクライアントが検索機能を提供している場合、攻撃者は様々なキーワードへのリクエストをトリガーにして、その後、様々なサイドチャネルを通じて、これらのキーワードのどれかが結果をもたらしたかどうかを判断できます。これにより、攻撃者は、攻撃者のウェブサイトの知らない訪問者のメールボックス内で検索機能を効果的に利用できるようになります。

幸いなことにブラウザは、潜在的な攻撃の結果を制限することに対して非常に効果的なセキュリティメカニズムの大規模なセットも提供します。CSPの`script-src`ディレクティブを介して、XSSの脆弱性を悪用することが非常に困難または不可能になる可能性があります。

クリックジャッキング[449]攻撃を防ぐためには、`X-Frame-Options`ヘッダーが存在しなければならないか、CSPの`frame-ancestors`ディレクティブを使用して、現在の文書を埋め込むことができる信頼されたパーティを示すことができます。

448. https://xsleaks.dev/
449. https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AA%E3%83%83%E3%82%AF%E3%82%B8%E3%83%A3%E3%83%83%E3%82%AD%E3%83%B3%E3%82%B0

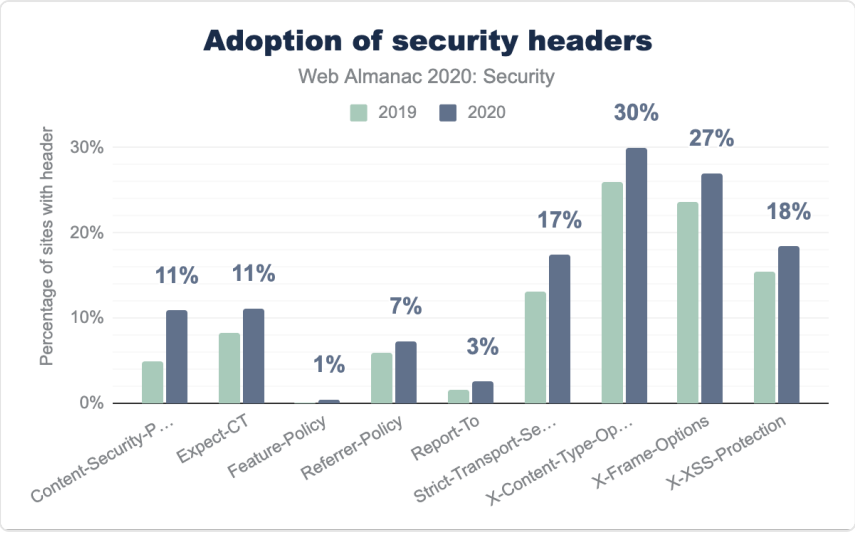


図11.22. セキュリティヘッダーの採用

セキュリティメカニズムの採用

ウェブ上で最も一般的なセキュリティ応答ヘッダーは`X-Content-Type-Options`であり、これはブラウザに広告されたコンテンツタイプを信頼するように指示し、応答内容に基づいてそれを盗聴しないようにします。これは例えば、攻撃者がクロスサイトスクリプティング攻撃を実行するためにHTMLコードとして解釈されるJSONPエンドポイントを悪用することを防ぐなど、MIMEタイプの混乱攻撃を効果的に防ぎます。

リストの次は、`X-Frame-Options`(XFO)応答ヘッダーであり、ページの約27%で有効になっています。このヘッダーは、CSPの`frame-ancestors`ディレクティブとともに、クリックジャッキング攻撃に対抗するために使用できる唯一の効果的なメカニズムです。しかし、XFOはクリックジャッキングに対して有用なだけでなく、他の様々なタイプの攻撃[450]に対しても悪用を著しく困難にしています。Internet Explorerなどのレガシーブラウザでは、XFOは現在でもクリックジャッキング攻撃を防御する唯一のメカニズムですが、ダブルフレーミング攻撃[451]の影響を受けています。この問題は`frame-ancestors`CSPディレクティブで緩和されています。そのため、ユーザに可能な限りの最善の防御を与えるためには、両方のヘッダーを採用することが最善の方法と考えられます。

現在18.39%のWebサイトで採用されている`X-XSS-Protection`ヘッダーは、ブラウザに内蔵されている反射型クロスサイトスクリプティングの検出機構を制御するために使用されて

450. https://cure53.de/xfo-clickjacking.pdf
451. https://www.usenix.org/system/files/sec20fall_calzavara_prepub.pdf

いました。しかし、Chromeバージョン78では、ブラウザに内蔵されていたXSS検出機能が非推奨・削除[452]されています。これは、様々な迂回経路が存在し、攻撃者に悪用される可能性のある新たな脆弱性や情報漏洩[453]を導入していたためです。他のブラウザベンダーは同様のメカニズムを実装していないため、`X-XSS-Protection`ヘッダーは最近のブラウザには効果がなく、安全に削除できます。それにもかかわらず、このヘッダーの採用率は昨年の15.50%から18.39%へとわずかに増加しています。

最も広く採用されている上位5つのヘッダーの残りの部分は、ウェブサイトのTLS実装に関連する2つのヘッダーで完結しています。Strict-Transport-Security[454]ヘッダーは、`max-age`属性で定義された期間だけHTTPS接続でウェブサイトを訪問するようブラウザに指示するため使用されます。このヘッダーの設定については、この章で詳しく説明しているこの章の前半。`Expect-CT`ヘッダーは、現在のWebサイトに発行された証明書が公開されているCertificate Transparency[455]ログに表示される必要があるかどうかを確認するようブラウザに指示します。

全体的に見ると、昨年はセキュリティヘッダーの採用が増加していることがわかります。最も広く使用されているセキュリティヘッダーは、相対的に15〜35パーセントの増加を示しています。最近導入された`Report-To`ヘッダーや`Feature-Policy`ヘッダーのような機能の採用が増加していることも注目に値します（後者は昨年の3倍以上に増加しています）。最も強い絶対的な成長はCSPヘッダーで、採用率は4.94%から10.93%に増加しています。

CSPによるXSS攻撃の防止

キーワード	デスクトップ	モバイル
`strict-dynamic`	2.40%	14.68%
`nonce-`	8.72%	17.40%
`unsafe-inline`	89.83%	92.28%
`unsafe-eval`	84.03%	77.48%

図11.23. default-srcまたはscript-src指令を定義するポリシーに基づくCSPキーワードの普及率。

XSS攻撃を防ぐのに役立つ厳格なCSPを実装することは容易なことではありません。導入を容易にするために、CSPの最後のバージョン（バージョン3）では、`default-src`や`script-src`ディレクティブで使用できる新しいキーワードが提供されています。例えば、

452. https://bugs.chromium.org/p/chromium/issues/detail?id=968591
453. https://frederik-braun.com/xssauditor-bad.html
454. https://developer.mozilla.org/docs/Web/HTTP/Headers/Strict-Transport-Security
455. https://developer.mozilla.org/docs/Web/HTTP/Headers/Expect-CT

`strict-dynamic` キーワードは、既に信頼されているスクリプトによって動的に追加されたスクリプト、例えば、そのスクリプトが新しい `<script>` 要素を作成したときなどを許可します。 `default-src` または `script-src` ディレクティブを含むポリシー（全CSPの21.17%）では、この比較的新しいキーワードの採用率は14.68%となっています。興味深いことに、デスクトップページでは、このメカニズムの採用率は2.40%と大幅に低くなっています。

CSPの採用を容易にするもう1つのメカニズムはnoncesの使用です。CSPの `script-src` ディレクティブで、ページはキーワード `nonce-` の後にランダムな文字列を入力できます。ヘッダーで定義されたランダム文字列と同じ `nonce` 属性が設定されているスクリプト（インラインでもリモートでも）はすべて実行が許可されます。このように、この仕組みを使えば、どのスクリプトが含まれているかを事前にすべての異なる起源を決定する必要はありません。nonceメカニズムは、 `script-src` または `default-src` ディレクティブを定義したポリシーの17.40%で使用されていることがわかりました。また、デスクトップページでの採用率は8.72%と低かった。この大きな違いを説明することはできませんでしたが、何か提案があれば聞いてください[456]！

他の2つのキーワード、 `unsafe-inline` と `unsafe-eval` は、それぞれ97.28%、77.79%と大多数のCSPに存在しています。このことは、XSS攻撃を阻止するポリシーを実装することの難しさを思い知らされます。しかし、 `strict-dynamic` キーワードが存在する場合、 `unsafe-inline` や `unsafe-eval` キーワードは事実上無視されてしまいます。古いブラウザでは `strict-dynamic` キーワードはサポートされていない可能性があるため、すべてのブラウザのバージョンとの互換性を維持するため、他の2つの安全でないキーワードを含めることが最善の方法と考えられています。

`strict-dynamic` や `nonce-` キーワードは、反映された持続的なXSS攻撃を防御するために使用できますが、保護されたページはDOMベースのXSS脆弱性に対して脆弱性を持っている可能性があります。このクラスの攻撃を防御するために、ウェブサイト開発者はTrusted Types[457]を利用できます。これはかなり新しいメカニズムで、現在Chromiumベースのブラウザでのみサポートされています。Trusted Typesの採用には困難が伴う可能性があるにもかかわらず（ウェブサイトはポリシーを作成し、このポリシーに準拠するためJavaScriptコードを調整する必要があります）、また新しいメカニズムであることを考えると、11のホームページがCSPの `require-trusted-types-for` ディレクティブを通じてTrusted Typesをすでに採用していることは心強いことです。

クロスオリジンポリシーによるXSリークの防御

XS-Leaks[458]と呼ばれる新しいクラスの攻撃から身を守るために、最近、様々な新しいセキュ

456. *https://discuss.httparchive.org/t/2047*
457. *https://web.dev/i18n/ja/trusted-types/*
458. *https://xsleaks.dev/*

リティメカニズムが導入されました（いくつかはまだ開発中です）。一般的に、これらのセキュリティメカニズムは、他のサイトが自分のサイトとどのように相互作用するかをウェブサイトの管理者がコントロールできるようにしています。例えば、`Cross-Origin-Openener-Policy` (COOP) レスポンスヘッダーは、ページを他の悪意のある可能性のあるブラウザコンテキストから分離処理すべきであることをブラウザに指示するために使用できます。そのため、敵対者はページのグローバルオブジェクトへの参照を取得することができません。その結果、frame counting[459]のような攻撃は、このメカニズムで防ぐことができます。データ収集が始まる数日前にChrome、Edge、Firefoxでしかサポートされていなかったこのメカニズムのアーリーアダプターが31件見つかりました。

`Cross-Origin-Resource-Policy` (CORP)ヘッダーは、Chrome、Firefox、Edgeでサポートされていますが、すでに1,712ページで採用されています（CORPはドキュメントだけでなく、すべてのリソースタイプで有効にできます/すべきであることに注意してください、したがってこの数は過小評価かもしれません）。ヘッダーは、ウェブリソースがどのように含まれることが予想されるかをブラウザに指示するために使用されます: 同一オリジン、同一サイト、クロスオリジン（制限が多いものから少ないものへ）。ブラウザは、CORPに違反する方法で含まれるリソースの読み込みを阻止します。このように、この応答ヘッダーで保護された機密リソースは、Spectre攻撃[460]や様々なXS-Leaks攻撃[461]から保護されています。Cross-Origin Read Blocking[462](CORB)メカニズムも同様の保護を提供しますが、ブラウザではデフォルトで有効になっています（現在はChromiumベースのブラウザのみ）。

CORPに関連しているのは `Cross-Origin-Embedder-Policy` (COEP)レスポンスヘッダーで、ページに読み込まれたリソースはすべてCORPヘッダーを持つべきであることをブラウザに指示するためにドキュメントで使用できます。さらに、Cross-Origin Resource Sharing(CORS)メカニズム（`Access-Control-Allow-Origin` ヘッダーなど）を通して読み込まれるリソースも許可されます。COOPと一緒にこのヘッダーを有効にすることで、ページは高精度タイマーや `SharedArrayBuffer` のような、潜在的にセンシティブなAPIへのアクセスを得ることができ、非常に正確なタイマーを構築するために使用することもできます。COEPを有効にしているページが6つ見つかりましたが、ヘッダーのサポートはデータ収集の数日前にブラウザへ追加されただけでした。

クロスオリジンポリシーのほとんどは、ウェブ上での利用が限られている（例えば、新しく開いたウィンドウへの参照を保持するなど）いくつかのブラウザ機能の潜在的に悪質な結果を無効にしたり、緩和したりすることを目的としています。そのため、COOPやCORPのようなセキュリティ機能を有効にすることは、ほとんどの場合、機能を壊すことなく行うことができます。したがって、これらのクロスオリジンポリシーの採用は、今後数ヶ月、数年の間に大幅に増加すると予想されます。

459. *https://xsleaks.dev/docs/attacks/frame-counting/*
460. *https://spectreattack.com/spectre.pdf*
461. *https://xsleaks.dev/docs/defenses/opt-in/corp/*
462. *https://fetch.spec.whatwg.org/#corb*

ウェブ暗号API

Web Cryptography API[463]は、外部ライブラリを必要とせず、少ない労力でクライアント側で安全に暗号処理を実行できる、開発者向けの優れたJavaScript機能を提供します。このJavaScript APIは、基本的な暗号演算を提供するだけでなく、暗号的に強い乱数値の生成、ハッシュ化、署名の生成と検証、暗号化と復号化を行うことができます。また、本APIの助けを借りて、ユーザーの認証や文書への署名、通信の機密性や完全性を安全に保護するためのアルゴリズムを実装できます。その結果このAPIを利用することで、エンドツーエンド暗号化の分野で、より安全でデータ保護に準拠したユースケースが可能になります。このように、Web Cryptography APIはエンドツーエンド暗号化に貢献しています。

暗号API	デスクトップ	モバイル
`CryptoGetRandomValues`	70.32%	67.94%
`SubtleCryptoGenerateKey`	0.3%	0.2%
`SubtleCryptoEncrypt`	0.3%	0.2%
`SubtleCryptoDigest`	0.3%	0.3%
`CryptoAlgorithmSha256`	0.2%	0.2%

図11.24. よく使われている暗号API

我々の結果によると、（安全で暗号化された方法で）乱数を生成できる `Cypto.getRandomValues` 関数が最も広く利用されていることがわかりました(デスクトップ: 70%、モバイル: 68%)。Google Analyticでは、この機能を利用していることが利用状況の測定に大きく影響していると考えています。一般的に、モバイルブラウザはこのAPIを完全にサポート[464]していますが、モバイルサイトでは暗号化処理の実行数がわずかに少なくなっていることがわかります。

注意すべき点は、受動的なクロールを行っているため、このセクションでの結果はこれによって制限されることです。関数が実行される前に何らかのインタラクションが必要なケースを特定することはできません。

ボット保護サービスの活用

Imperva[465]によると、ウェブトラフィック全体の深刻な割合(37%)は自動化されたプログラム（いわゆる「ボット」）に属しており、そのほとんどは悪意のあるもの(24%)です。ボット

463. https://www.w3.org/TR/WebCryptoAPI/
464. https://developer.mozilla.org/docs/Web/API/Web_Crypto_API#Browser_compatibility
465. https://www.imperva.com/blog/bad-bot-report-2020-bad-bots-strike-back

は、フィッシング、情報収集、脆弱性の悪用、DDoS、その他多くの目的で利用されています。ボットを利用することは、攻撃者にとって非常に興味深い手法であり、特に大規模攻撃の成功率を高めます。そのため、悪意のあるボットからの防御は、大規模な自動化攻撃からの防御に役立ちます。次の図は、悪意のあるボットに対するサードパーティの保護サービスの利用状況を示しています。

サービスプロバイダー	デスクトップ	モバイル
reCAPTCHA	8.30%	9.03%
Imperva	0.30%	0.36%
hCaptcha	0.01%	0.01%
Others	<0.01%	<0.01%

図11.25. プロバイダによるボット対策サービスの利用状況

上の図は、弊社のデータセットに基づくボットプロテクションの利用状況と市場シェアを示しています。デスクトップページの約10%、モバイルページの約9%がこのようなサービスを利用していることがわかります。

セキュリティヘッダーの採用と様々な要因との関係

前のセクションでは、レスポンスヘッダーを介してウェブページで有効にする必要があるさまざまなブラウザのセキュリティメカニズムの採用率を調査しました。次に国レベルのポリシーや規制に関連しているのか、顧客の安全を確保したいという一般的な関心なのか、あるいはウェブサイトを構築するために使用される技術スタックに牽引されているのかなど、ウェブサイトがセキュリティ機能を採用する要因を探っていきます。

サイトの訪問者の国

国レベルでのセキュリティに影響を与える要因はさまざまです。政府が主導するサイバーセキュリティのプログラムによって、優れたセキュリティ対策に対する意識が高まるかもしれませんし、高等教育においてセキュリティに重点を置くことで、より多くの知識を持った開発者が生まれるかもしれませんし、特定の規制によって企業や組織がベストプラクティスを遵守することを要求されるかもしれません。国ごとの違いを評価するために、Chromeユーザー体験レポート（CrUX）に基づいたデータセットで、少なくとも10万件のホームページが利用可能な国を分析しました。これらのページは、その国のユーザーが最も頻繁に訪問したページで構成されており、広く人気のある国際的なウェブサイトも含まれています。

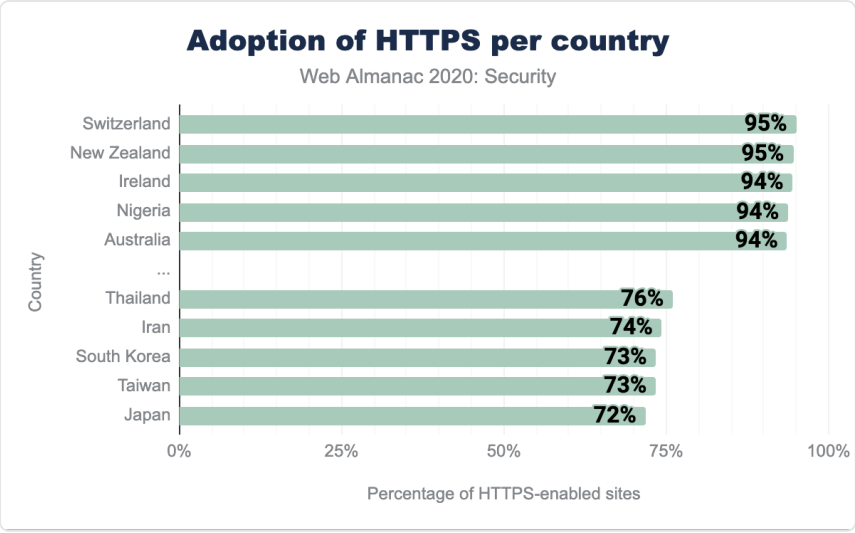


図11.26. 国別のHTTPSの採用状況

HTTPSで訪問されたホームページの割合を見ると、すでに大きな違いが見られます。最下位5カ国では、HTTPSの採用率は71%から76%と、かなり低くなっています。他のセキュリティメカニズムに目を向けると、成績上位の国と採用率の低い国との間には、さらに明らかな違いが見られます。CSPの採用率によると、上位5カ国では14%から16%のスコアが得られているのに対し、下位5カ国では2.5%から5%のスコアが得られています。興味深いことに、1つの安全保障メカニズムで良い成績を収めている国と悪い成績を収めている国は、他のメカニズムでも同じような成績を収めています。例えば、ニュージーランド、アイルランド、オーストラリアは常にトップ5にランクインしていますが、日本はほとんどすべてのセキュリティメカニズムでワーストのスコアを記録しています。

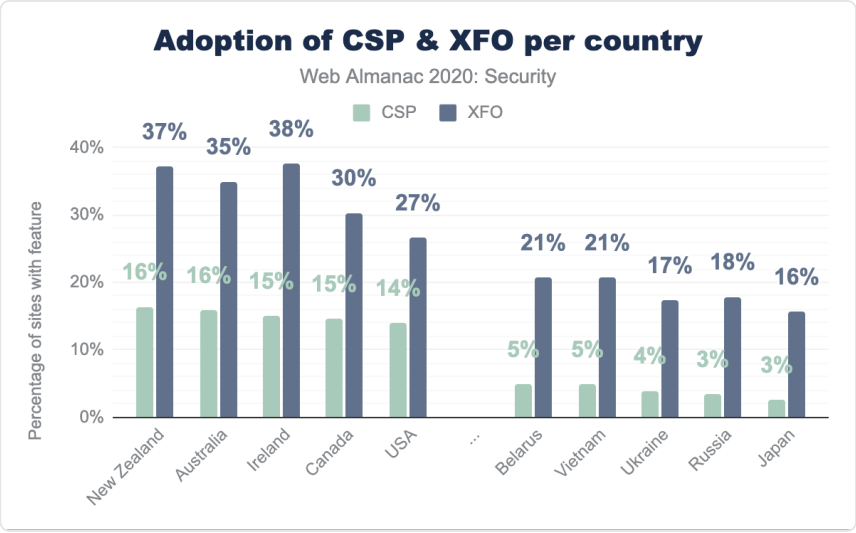


図11.27. 国ごとのCSPとXFOの採用。

技術スタック

国レベルのインセンティブは、セキュリティメカニズムの採用をある程度促進することができますが、おそらくもっと重要なのは、ウェブサイトを構築する際にウェブサイト開発者が使用する技術スタックです。フレームワークは、特定の機能を有効にすることを容易にしているのか、それともアプリケーションの完全なオーバーホールを必要とする骨の折れるプロセスなのか。開発者は、最初から強力なセキュリティのデフォルトが設定されている、すでに安全な環境から始めた方が良いでしょう。このセクションでは、特定の機能の採用率が著しく高い（したがって、広く採用されるための原動力となる）プログラミング言語、SaaS、CMS、eコマース、CDNテクノロジを探っていきます。簡潔にするために、ここでは最も広く普及している技術に焦点を当てていますが、ユーザーにより良いセキュリティを提供することを目的とした、より小規模な技術製品が数多く存在することに注意することが重要です。

トランスポートセキュリティに関連したセキュリティ機能については、顧客サイトの少なくとも90%で`Strict-Transport-Security`ヘッダーを有効にしている12のテクノロジー製品（主にeコマースプラットフォームやCMS）があることがわかりました。上位3社（市場シェアによるとShopify、Squarespace、Automattic）が運営するウェブサイトでは、Strict Transport Securityを有効にしているホームページの30.32%を占めています。興味深いことに、`Expect-CT`ヘッダーの採用は、主に単一のテクノロジーであるCloudflareによって推進されており、CloudflareはHTTPSを有効にしているすべての顧客でヘッダーを有効にしている。その結果、`Expect-CT`ヘッダーの99.06%がCloudflareに関連していることがわかっ

た。

コンテンツの包含を確保するセキュリティヘッダーや攻撃を阻止することを目的としたセキュリティヘッダーに関しては、いくつかの当事者がすべての顧客に対してセキュリティヘッダーを有効にしており、それによってその採用が促進されるという同様の現象が見られます。例えば、6つのテクノロジー製品は、80%以上の顧客に対して `Content-Security-Policy` ヘッダーを有効にしています。このように、上位3社（Shopify、Sucuri、Tumblr）は、このヘッダーを持つホームページの52.53%を占めている。同様に、`X-Frame-Options` についても、上位3社（Shopify、Drupal、Magento）がXFOヘッダーの世界的な普及率の34.96%に貢献していることがわかります。特にDrupalはオープンソースのCMSであり、ウェブサイトの所有者自身が設定することが多いので、これは興味深いことです。`X-Frame-Options: SAMEORIGIN` をデフォルトで有効にする決定が、クリックジャッキング攻撃から多くのユーザーを守っていることは明らかです。Drupalを搭載したウェブサイトの81.81%がXFOメカニズムを有効にしています。

他のセキュリティヘッダーとの重複

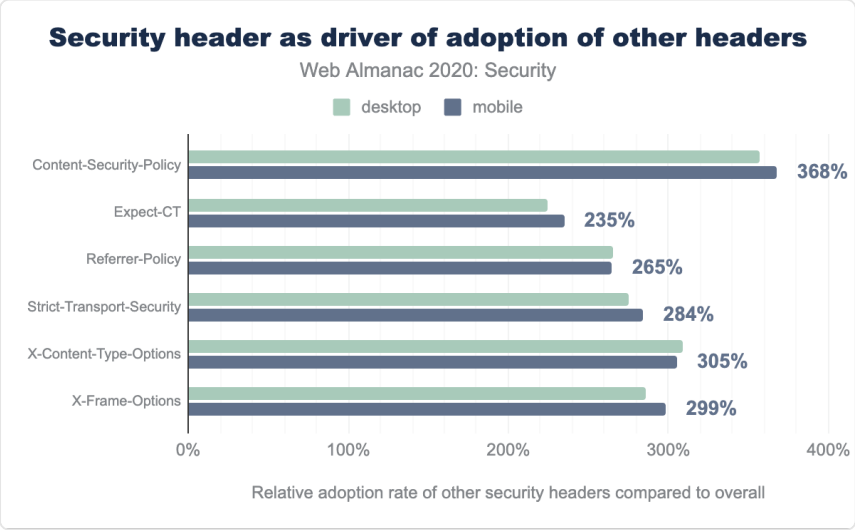


図11.28. 他のヘッダーを採用する際のドライバーとしてのセキュリティヘッダー

セキュリティの「ゲーム」は非常にバランスが悪く、攻撃者に有利なものとなっています。このように単一のセキュリティメカニズムを採用することは、特定の攻撃を防御する上で非常に有用であるのに対し、ウェブサイトはすべての可能性のある攻撃から防御するために複数のセキュリティ機能を必要とします。セキュリティヘッダーが単発的に採用されているのか、それとも可能な限り多くの攻撃に対してより綿密な防御を提供するために厳格な方法で

採用されているのかを判断するために、セキュリティヘッダーの共起性に注目します。より正確には、あるセキュリティヘッダーの採用が他のヘッダーの採用にどのような影響を与えるかを見ています。興味深いことに、単一のセキュリティヘッダーを採用しているウェブサイトは、他のセキュリティヘッダーも採用する可能性が高いことがわかります。例えばCSPヘッダーを含むモバイルホームページでは、他のヘッダー(`Expect-CT`、`Referrerer-Policy`、`Strict-Transport-Security`、`X-Content-Type-Options`、`X-Frame-Options`)の採用率は、これらのヘッダーの全体的な採用率と比較して平均368%も高くなっています。

一般的に、特定のセキュリティヘッダーを採用しているウェブサイトは、他のセキュリティヘッダーも同様に採用する可能性が2〜3倍高くなります。これは特にCSPの場合に顕著で、他のセキュリティヘッダーの採用を最も促進します。これはCSPがより広範なセキュリティヘッダーの1つであり、採用にはかなりの労力を必要とするため、ポリシーを定義しているWebサイトは、Webサイトのセキュリティに投資する可能性が高いためと説明できます。一方で、CSPヘッダーの44.31%がShopifyを搭載したホームページ上に存在しています。このSaaS製品は、他にもいくつかのセキュリティヘッダー（`Strict-Transport-Security`、`X-Content-Type-Options`、`X-Frame-Options`）を事実上すべての顧客のデフォルトとして有効にしています。

ソフトウェアアップデートの実践

ウェブの非常に大きな部分は、技術スタックの異なる層にあるサードパーティのコンポーネントによって構築されています。これらのコンポーネントは、より良いユーザー体験を提供するために使用されるJavaScriptライブラリ、ウェブサイトのバックボーンを形成するコンテンツ管理システムやウェブアプリケーションのフレームワーク、訪問者からのリクエストに応答するために使用されるウェブサーバで構成されています。脆弱性は、これらのコンポーネントのいずれかで検出されることがよくあります。最良のケースでは脆弱性はセキュリティ研究者によって検出され、責任を持って影響を受けるベンダーに開示し、脆弱性にパッチを当ててソフトウェアのアップデートをリリースするように促します。この時点では、脆弱性の詳細が公表されている可能性が高く、攻撃者はその脆弱性を利用した悪用法の作成に熱心に取り組んでいます。そのため、ウェブサイトの所有者は、これらのn-day exploits[466]から身を守るために、影響を受けたソフトウェアをできるだけ早くアップデートすることが重要です。このセクションでは、最も広く使用されているソフトウェアがどの程度最新の状態に保たれているかを探ります。

466. *https://www.darkreading.com/vulnerabilities---threats/the-overlooked-problem-of-n-day-vulnerabilities/a/d-id/1331348*

WordPress

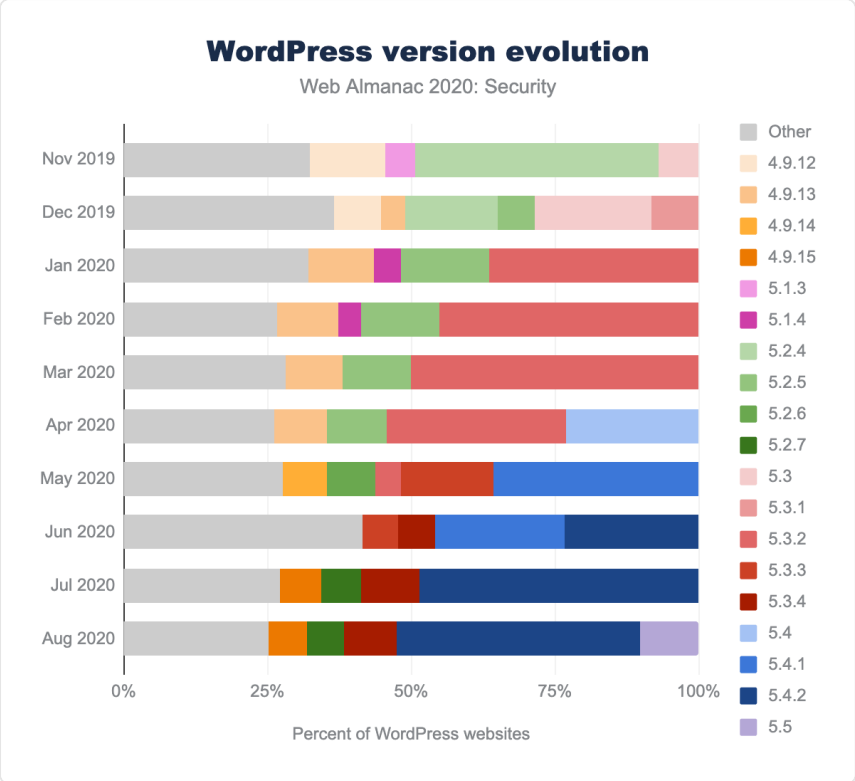


図11.29. WordPressのバージョンアップ。

最も人気のあるコンテンツ管理システムの1つであるWordPressは、攻撃者にとって魅力的なターゲットとなっています。そのため、ウェブサイト管理者はインストールを常に最新の状態に保つことが重要です。デフォルトでは、WordPress の更新は自動的に行われます[467]が、この機能を無効にすることも可能です。展開されているWordPressのバージョンの変遷は上図のように、「現在も積極的にメンテナンスが行われている最新の主要バージョン」[468]（5.5：紫、5.4：青、5.3：赤、5.2：緑、4.9：オレンジ）を表示しています。普及率が4%未満のバージョンは「その他」の下にまとめられています。最初に興味深い観察ができるのは、2020年8月の時点でモバイルホームページにインストールされているWordPressの74.89%が、そのブランチ内で最新バージョンを実行しているということです。また、ウェブサイトの所有者が徐々に新しいメジャーバージョンにアップグレードしていることがわかります。例えば、2020年8月11日にリリースされたWordPressのバージョン5.5は、8月の

467. https://wordpress.org/support/article/configuring-automatic-background-updates/
468. https://wordpress.org/download/releases/

クロールで観測されたWordPressインストールの10.22%を既に構成していました。

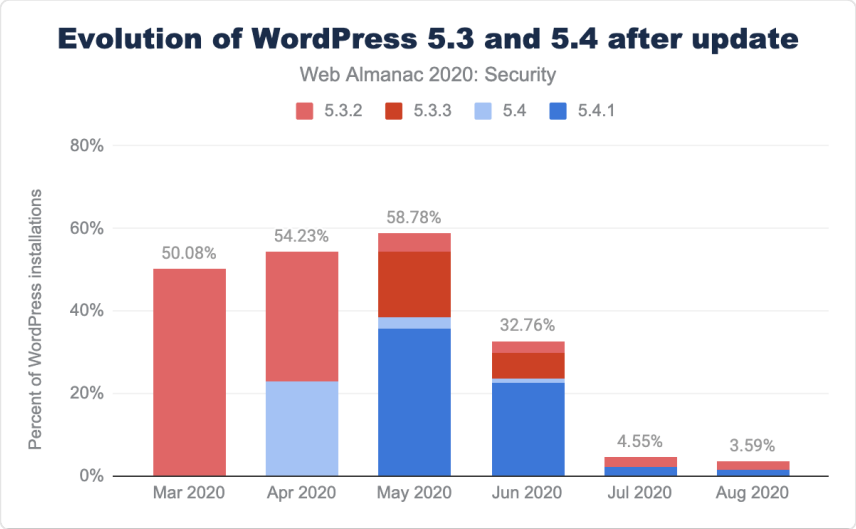


図11.30. アップデート後のWordPress 5.3と5.4の進化

グラフから推測できるもう1つの興味深い側面は、1ヶ月以内にそれまで最新の状態にあったWordPressサイトの大部分が新しいバージョンに更新されているということです。これは、特に最新のブランチにインストールされているWordPressに当てはまるようです。クロール開始2日前の2020年4月29日、WordPressはすべてのブランチのアップデートをリリースしました。5.4 → 5.4.1、5.3.2 → 5.3.3、などです。このデータをもとに、バージョン5.4を実行していたWordPressインストールのシェアは、2020年4月のクロール時の23.08%から、2020年5月には2.66%まで減少し、2020年6月には1.12%までさらに減少し、その後は1%以下に落ち込んでいることがわかります。新バージョン5.4.1は、2020年5月時点で35.70%のWordPressインストールで稼働しており、多くのWebサイト運営者が（5.4や他のブランチから）WordPressのインストールを（自動的に）更新した結果となっています。大多数のウェブサイト運営者は、自動的に、または新しいバージョンがリリースされた後に非常に迅速にWordPressを更新していますが、古いバージョンに固執し続けているサイトはまだ少数派です：2020年8月の時点で、全WordPressインストールの3.59%が古い5.3または5.4バージョンを実行していました。

jQuery

図11.31. jQueryのバージョンアップ。

最も広く使われているJavaScriptライブラリの一つにjQueryがありますが、これには大きく分けて3つのバージョンがあります。モバイルホームページで使用されているjQueryのバージョンの進化からも明らかなように、全体的な分布は時間の経過とともに非常に静的になっています。驚くべきことに、かなりの割合（2020年8月時点で18.21%）のWebサイトでは、まだ古いバージョンの1.xのjQueryが実行されています。この割合は一貫して減少しており（2019年11月時点の33.39%から）、2016年5月にリリース[469]されたバージョン1.12.4を好んで使用しており、残念ながら様々な中程度のセキュリティ問題[470]を抱えています。

469. https://blog.jquery.com/2016/05/20/jquery-1-12-4-and-2-2-4-released/
470. https://snyk.io/test/npm/jquery/1.12.4

nginx

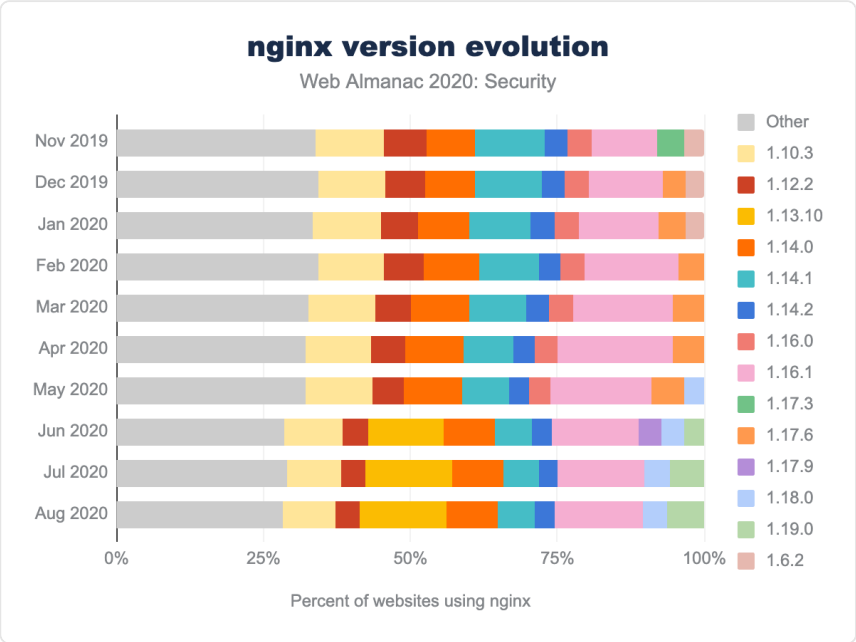


図11.32. nginxのバージョンアップ。

最も広く利用されているウェブサーバーの一つであるnginxについては、時間の経過とともに異なるバージョンが採用されているという点で、非常に静的で多様な状況が見られます。すべてのnginxサーバーの中で、どのnginx(マイナー)バージョンが占める割合が最も大きかったのは約20%でした。さらに、バージョンの分布は時間の経過とともにかなり静的なままであることがわかります。おそらくこれは、2014年以降、nginxにはメジャーなセキュリティ脆弱性が発見されていない[471]という事実と関係していると思われます（1.5.11までのバージョンに影響を与えます）。中程度の影響力を持つ最後の3つの脆弱性（CVE-2019-9511[472]、CVE-2019-9513[473]、CVE-2019-9516[474]）の日付は2019年3月からで、バージョン1.17.2までのHTTP/2対応nginxサーバで過度に高いCPU使用率を引き起こす可能性があります。報告されているバージョン数によると、半数以上のサーバがこの脆弱性の影響を受ける可能性があるとのことです（HTTP/2を有効にしている場合）。ウェブサーバソフトウェアは頻繁に更新されていないため、この数字は今後数ヶ月の間、かなり高い水準で推移する可能性があります。

471. *http://nginx.org/en/security_advisories.html*
472. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9511*
473. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9513*
474. *https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-9516*

ウェブ上の不正行為

現在では、使用される技術の性能が特に重要な役割を果たしています。そのために、技術は常にさらに開発され、最適化され、新しい技術が発表されています。これらの新しい技術の1つがWebAssemblyであり、2019年末時点でW3C勧告[475]となっています。WebAssemblyを利用することで強力なWebアプリケーションを開発でき、ブラウザでほぼネイティブな高性能コンピューティングを実行することが可能になりました。しかし、攻撃者がこの技術を利用してきたため、とげのないバラはありませんでしたが、これが新たな攻撃ベクトルcryptojacking[476]の登場です。攻撃者はこの技術を利用して、訪問者のコンピューターの力を利用してブラウザ上の暗号通貨を採掘します（悪意のある暗号採掘）。これは攻撃者にとって非常に魅力的な技術です - ウェブページに数行のJavaScriptコードを注入し、すべての訪問者にあなたのために採掘させます。一部のウェブサイトでは、ウェブ上で善意のクリプトマイニングを提供している場合もあるため、クリプトマイニングを行っているすべてのウェブサイトが実際にクリプトジャッキングを行っていると一般化することはできません。しかしほとんどの場合、ウェブサイトのオペレーターは訪問者のためのオプトインの代替手段を提供していないと、訪問者は彼らのリソースがウェブサイトを閲覧しながら使用されているかどうかについては、まだ無知のままです。

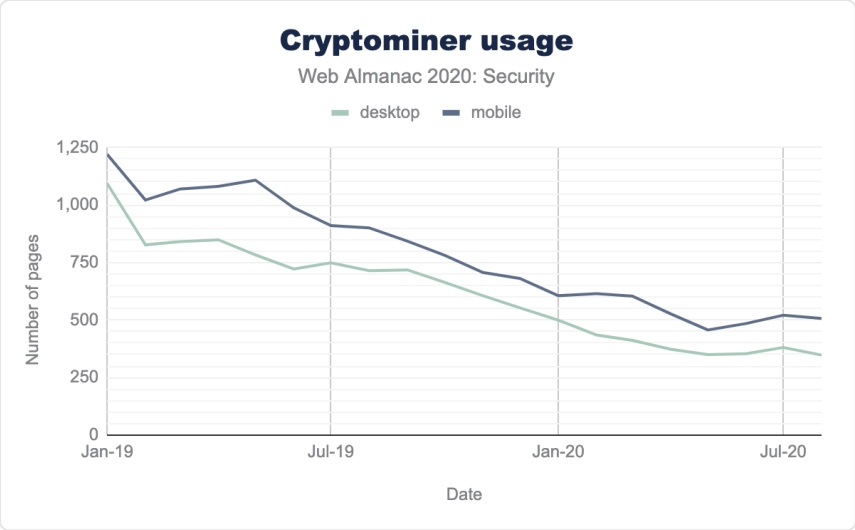


図11.33. クリプトマイナーの使い方。

上の図は、過去2年間のクリプトマイニングを活用したウェブサイトの数を示しています。2019年に入ってから、クリプトマイニングへの関心が低くなっていることがわかります。

475. *https://www.w3.org/2019/12/pressrelease-wasm-rec.html.en*
476. *https://www.malwarebytes.com/cryptojacking/*

前回の測定では、クリプトマイニングスクリプトを含むウェブサイトが合計348件ありました。

次の図では、8月のデータセットをもとに、ウェブ上のクリプトマイナーの市場シェアを示しています。Coinimpが45.2%の市場シェアを持ち、最も人気のあるプロバイダーとなっています。最も人気のあるクリプトマイナーはすべてWebAssemblyをベースにしていることがわかりました。Web上で採掘するためのJavaScriptライブラリもありますが、WebAssemblyをベースにしたソリューションほど強力ではないことに注意してください。私たちの別の結果によると、クリプトマイニングを含むウェブサイトの半分は、廃止されたサービスプロバイダーのクリプトマイニングコンポーネントを利用していることがわかりました（CoinHive[477]やJSEcoin[478]のようなもの）。

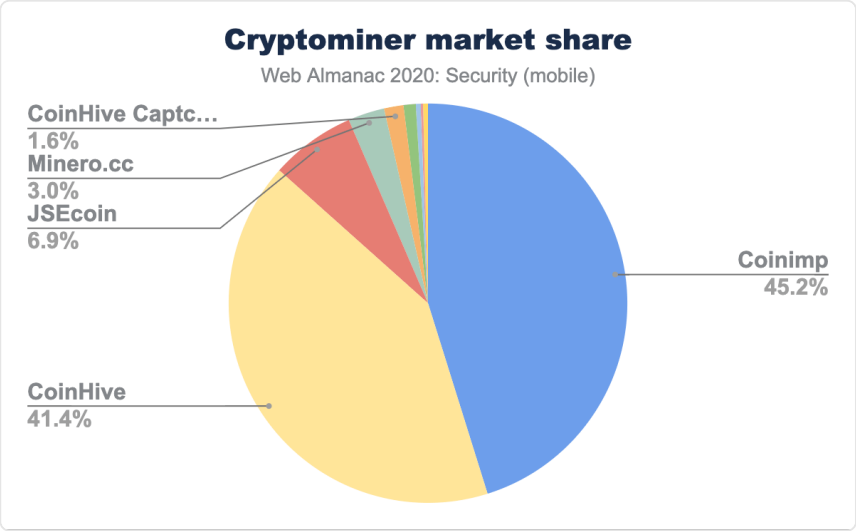


図11.34. クリプトマイナーの市場シェア（モバイル）。

結論

昨年のウェブセキュリティに関する最も顕著な進展の1つは、ウェブの（ロングテールの）セキュリティヘッダーの採用が増加したことです。これは全体的なユーザの保護が向上したことを意味するだけでなく、より重要なことはウェブサイト管理者のセキュリティに対するインセンティブが一般的に高まったことを示しています。この使用率の増加は、CSPのような実装が困難なものであっても、すべての異なるセキュリティヘッダーで見ることができま

477. https://blog.avast.com/coinhive-shuts-down
478. https://x.com/jsecoin/status/1247436272869814272

す。もう1つの興味深い観察として、あるセキュリティヘッダーを採用しているウェブサイトでは、他のセキュリティメカニズムを採用する可能性が高いことがわかりました。このことは、ウェブセキュリティが単なる余計なものとして考えられているのではなく、開発者がすべての可能性のある脅威から守ることを目的とした全体的なアプローチであることを示しています。

世界的な規模では、まだ長い道のりがあります。例えば、クリックジャッキング攻撃に対する適切な保護機能を備えているサイトは全サイトの3分の1以下であり、多くのサイトではCookieの`SameSite`属性を`None`に設定することで様々なクロスサイト攻撃に対する保護機能(特定のブラウザではデフォルトで有効になっています)をオプトアウトしています。それにもかかわらず、より積極的な進化も見られます。テクノロジースタック上の様々なソフトウェアがデフォルトでセキュリティ機能を有効にしています。このソフトウェアを使ってウェブサイトを構築する開発者は、すでに安全な環境からスタートし、必要に応じて強制的に保護機能を無効にしなければなりません。これは、レガシーアプリケーションで見られるものとは大きく異なり、機能を壊す可能性があるため、セキュリティの強化がより困難になる可能性があります。

今後の見通しとして、私たちが知っている1つの予測は、セキュリティの状況が行き詰まることはないだろうということです。新たな攻撃手法が表面化し、それを防御するための追加のセキュリティメカニズムが必要になる可能性があります。最近採用されたばかりの他のセキュリティ機能を見てきたように、これには時間がかかるでしょうが、徐々に一歩一歩より安全なウェブへと移行します。

著者

Tom Van Goethem

@tomvangoethem　tomvangoethem　https://tom.vg/

Tom Van Goethemは、ベルギーのルーベン大学のDistriNetグループ[479]の研究者です。彼の研究は、セキュリティやプライバシーの問題につながるウェブ上の新たなサイドチャネル攻撃を発見し、その原因となる漏洩をパッチで修正する方法を見出すことに重点を置いています。

479. *https://distrinet.cs.kuleuven.be/*

Nurullah Demir

@nrllah　nrllh　https://ndemir.com

Nurululllah Demirは、Institute for Internet Security[480]のセキュリティ研究者であり、博士課程の学生です。彼の研究は、屈強なWebセキュリティメカニズムと敵対的な機械学習に焦点を当てています。

Barry Pollard

@tunetheweb　@https://webperf.social/@tunetheweb　@tunetheweb.com　tunetheweb

tunetheweb　https://www.tunetheweb.com

Barry Pollardはソフトウェア開発者であり、Manningの本HTTP/2 in Action[481]の著者でもあります。彼はウェブは素晴らしいと思っていますが、それをもっと良くしたいと思っています。@tunetheweb でつぶやき、www.tunetheweb.com でブログを書いています。

480. *https://www.internet-sicherheit.de/en/*
481. *https://www.manning.com/books/http2-in-action*

部 II 章 12
モバイル・ウェブ

Shubhie Panicker と Michael DiBlasio によって書かれた。
David Fox によってレビュ と による分析。
Shane Exterkamp 編集。
Sakae Kotaro によって翻訳された。

序章

モバイル・ウェブは過去10年間で爆発的に成長し、今や多くの人がウェブを体験する主要な手段となっています。それにもかかわらず、エンゲージメントやオンラインセールスはデスクトップに比べてまだ遅れています。本章では、モバイルウェブの最近のトレンドを紹介するとともに、ユーザーの旅の完了が困難になりがちな理由を分析します。

2020年は、世界的なパンデミックの影響で、モバイルとデスクトップの両方で、インターネットの使用量[482]が大幅に増加しました。世界中の人々が専業主婦の注文や社会的な距離を置いた新しいライフスタイルに順応したことで、ニュースサイトやeコマース、ソーシャルメディアサイトへのアクセスが増加しました。2020年はウェブにとっても、モバイル利用にとっても、歴史的に重要な年となりました。

482. *https://www.nytimes.com/interactive/2020/04/07/technology/coronavirus-internet-use.html*

データソース

この章では、いくつかの異なるデータソースを使用しました。

- CrUX
- HTTP Archive
- Lighthouse

各データソースの方法論や注意点については、上記のリンクを参照してください。なおHTTP ArchiveおよびLighthouseのデータは、ウェブサイトのトップページのみから抽出したデータに限定されており、サイト全体のデータではありませんのでご注意ください。

上記に加えて、Chromeでのページロードのセクションでは、非公開のChromeデータソースを使用しました。これについては、Chromeのデータ収集API[483]をお読みください。

このデータは、（オプトインした）Chromeユーザーのサブセットからのみ収集されていますが、ホームページに限定されているという問題はありません。このデータは仮名で、ヒストグラムとイベントで構成されています。

注：レポートは、ユーザーがブラウザのウィンドウを同期する機能を有効にしている場合、「検索と閲覧をより快適にする/訪問したページのURLをGoogleに送信する」の設定を無効にしていない限り、有効になります。

モバイルウェブとデスクトップのトラフィック傾向

ユーザーは、モバイルウェブとデスクトップでどのくらいウェブサイトを訪れているのでしょうか？　モバイルとデスクトップのトラフィックには、何かパターンがあるのでしょうか？　これらの疑問と、それがウェブサイトにとってどのような意味を持つのかを検証するために、私たちはいくつかのレンズからデータを調べました。

perficient.comで公開されているレポート[484]では、similarweb[485]をデータソースとして、数年間のモバイルとデスクトップのトラフィックの傾向を示しています。サイト訪問の大部分（**58%**）がモバイルデバイスからのものであるにもかかわらず、オンラインでの総滞在時間に占めるモバイルデバイスの割合は42%にとどまっています。さらに、1回の訪問あたりの平均滞在時間は、モバイルに比べてデスクトップでは約2倍となっています（デスクトップ：11.52分、モバイル：5.95分）。

483. *https://chromium.googlesource.com/chromium/src/+/master/services/metrics/ukm_api.md*
484. *https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage-study*
485. *https://www.similarweb.com/*

Chromeでのページロード（Chromeのデータソース）

このセクションでは、本章のために特別に公開されたChromeの非公開データソース詳細はこちらの統計データを参照しています。このデータを使って、AndroidとWindowsでのページロードを評価しています（それぞれモバイルとデスクトップの代理としています）。

注：この項では、Androidの場合はモバイル、Windowsの場合はデスクトップと表記することがあります。

オリジン間のページロードを人気順に表示

オリジンへのトラフィックを人気度別に見ると、ユーザーが特定のオリジンにどのくらいの頻度でアクセスしているか、それによってウェブ上の世界的な分布がどうなっているかがわかります。

Rick Byersが1年前にこの分布をツィート[486]していたので、最新のデータを見てみました。このグラフは、オリジンの人気度による全体的な分布を、Chromeでのページロード（%）への貢献度で示したものです。

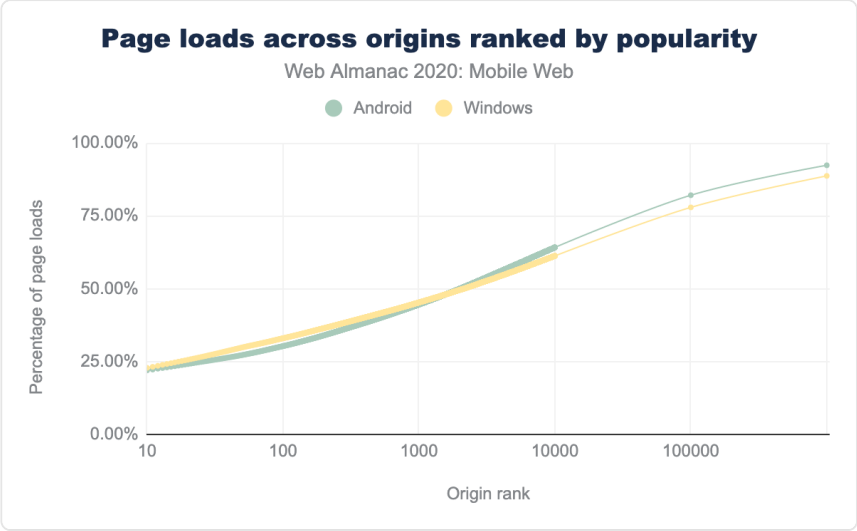


図12.1. オリジン間のページロードを人気順に並べたもの（Chromeの場合）

いくつかの収穫がありました。

486. https://x.com/RickByers/status/1195342331588706306

- Chromeの使用率は、上位200サイト、次の10,000サイト、残りのすべてのサイトでほぼ均等に分かれています。
- ウェブには、**頭でっかち**なところがあります。
    - モバイル、デスクトップともに、少数のオリジンがトラフィックの大部分を占めています。
    - 上位の30オリジンは、モバイルでの総トラフィックの25%を占めています。
    - 上位200のオリジンは、モバイルでの総トラフィックの33%を占めています。
- ウェブには**幅広い胴体**があります。
    - 上位1万のオリジンがトラフィックの約3分の2を占めます。モバイルでのトラフィックの64%を占めています。
- ウェブには **長い尾** があります。
    - Androidでは上位98%に3万のオリジンが入っているのに対し、Windowsでは180万のオリジンが入っています。
    - Androidでは、Windowsに比べて約2倍の長さの尾を引いています。これは、デスクトップに比べてモバイル端末やユーザーの数が多いことに起因すると考えられます。

モバイルからのサイトへのトラフィックとデスクトップからのトラフィックの比較（CrUX）

モバイルとデスクトップのトラフィック分布の予想について、ウェブサイトは推論できますか？

モバイルとデスクトップの配分は、サイトによって大きく異なるため、予測は難しいです。さらに業界のカテゴリー（例：エンターテインメント、ショッピング）や、サイトがネイティブアプリを持っているか、ネイティブアプリをどれだけ積極的に宣伝しているかなどにも大きく左右されます。

CrUXデータセットを利用して、モバイルデバイスとデスクトップからのChromeトラフィックを評価しました。

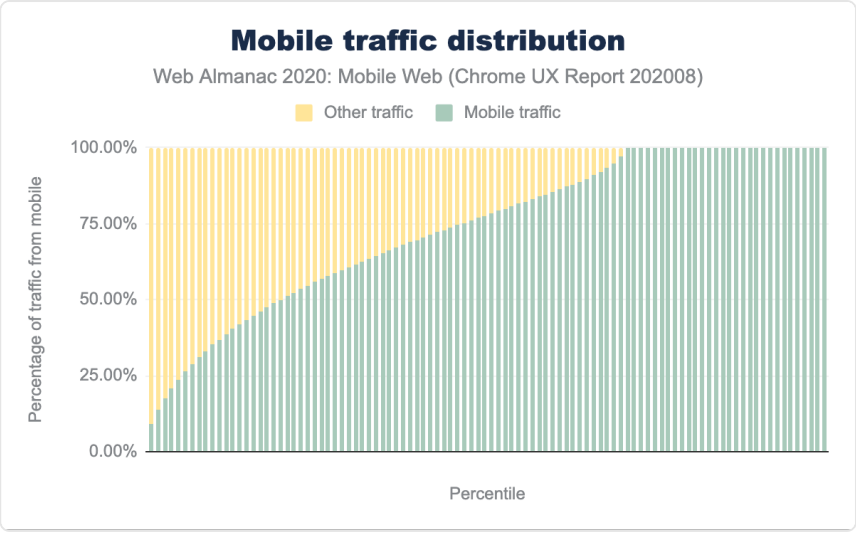


図12.2. モバイルとその他のトラフィックの分布

この分布は、モバイルが多いようです。その理由としては、総トラフィックは少ないものの、モバイルからのトラフィックしか得られないサイトが多数（CrUXでは200万以上）存在することが挙げられます。前のセクションで見たように、モバイルははるかに長い「テール」を持っています。

CrUXのデータがあるすべてのウェブサイトをバケツに入れてランダムに選ぶと、選んだウェブサイトの50%がモバイルからのトラフィックの77.61%以上を占めていることになります（2019年の79.93%から少し減少しています）。

これは興味深い観察結果ですが、モバイルとデスクトップの大まかな傾向についてCrUXから結論を出すのは難しいことに注意してください。なぜなら:

- CrUXはChromeのみのデータで、主要なモバイルブラウザであるSafariを含む他のブラウザが欠けています。
- Chromeの場合でも、これはオプトインしたユーザーの一部であり、オプトイン率やプラットフォーム間の差異の影響を受けます。

トレンドの結論

では、モバイルとデスクトップのトラフィックを比較することで、どのようなことがわかったのでしょうか。

モバイルとデスクトップのトラフィック配分は、サイトによって大きく異なり、業界カテゴリやネイティブアプリの有無などの他の要因にも左右されます。しかしChromeでのサイト訪問では、ユーザーがデスクトップでより多くの時間を過ごしているにもかかわらず、あるウェブサイトのトラフィックは主にモバイルウェブからのものである可能性が高いと考えられます。これは、モバイルChromeのテールがはるかに長いためです。

個々のウェブサイトについて、モバイルとデスクトップのトラフィック分布の予想を一般化することはできませんが、自社サイトの分布を業界カテゴリの分布[487]と比較することは価値があります。

もしあなたのウェブサイトが業界平均と大きく異なる場合は、その原因を探る価値があるかもしれません。たとえば、ロードパフォーマンスの悪さが原因の1つになるかもしれません。

ユーザーの旅

モバイル・ウェブでのユーザーの旅は、コマーシャルの旅を含めて、完了するのが難しいです。

モバイルは小売サイトでの滞在時間の79.6%を占める一方で、eコマースの売上[488]の32.3%しか占めていません。これは、ユーザーがモバイルで行動を開始しても、最終的にはデスクトップで行動することが多いことを示しています。これはなぜでしょうか？

このような疑問を解決するためには、まず、ユーザーの旅の要素を理解する必要があります。

私たちは、ユーザーの旅を4つのフェーズに分けています。

1. 取得

ウェブサイトにとって、訪問者の獲得は重要なエントリーフェーズです。獲得とは、検索エンジンや広告のクリック、他のサイトからのリンク、ソーシャルメディアなどを通じて、ウェブサイトに訪問者を集めることです。

SEO

獲得段階ではSEOが重要で検索エンジンは、ウェブサイトに訪問者を送り込み、ユーザーの旅を開始させる重要なソースです。SEO対策の主な目的は、検索エンジン（ページをクロー

487. *https://www.perficient.com/insights/research-hub/mobile-vs-desktop-usage-study*
488. *https://www.emarketer.com/content/frictionless-commerce-2020*

ルしてインデックスを作成する検索エンジンボット）と、ウェブサイトを閲覧してコンテンツを消費するユーザーのためにウェブサイトを最適化することです。

現在、多くのユーザーがモバイルで検索を開始しています。

レスポンシブWebデザイン

ウェブの閲覧や検索にモバイル端末が普及したことにより、Google検索は数年前にモバイルファーストインデックス[489]へ移行しました。これは、検索ランキングがモバイルユーザーから見たページを考慮することを意味しており、モバイルフレンドリーがランキング要因となっています。Googleは2021年3月に、すべてのサイトを対象に、モバイルファーストインデックス[490]へ全面的に切り替える予定です。

ウェブサイトは、検索ユーザーからのトラフィックに影響を与えるため、良好な検索体験とSEOのために、モバイルフレンドリ性を確保する必要があります。そのためには、レスポンシブWebデザイン[491]が推奨されます。

レスポンシブWebサイトでは、viewport metaタグやCSSメディアクエリを使用して、モバイルフレンドリーな体験を提供しています。これらのモバイルフレンドリーについて詳しく知りたい方は、SEOの章をご覧ください。

レスポンシブWebデザインの詳細はこちら[492]。

検索エンジンからのオーガニックなトラフィックだけでなく、**広告のクリック**もウェブサイトに送られる訪問者の重要なソースとなる可能性があります。SEOと同様に広告に投資し、広告からのトラフィックを受け取るウェブサイトにとって、広告の最適化は重要な意味を持ちます。

読み込み性能

第一印象は重要です。ページコンテンツをタイムリーに配信することは、訪問者の離脱やユーザーの不満を避けるために重要です。読み込みパフォーマンスは獲得フェーズの重要な要素であり、読み込みパフォーマンスが悪いとユーザーはこの旅を放棄してしまいます。

最近の調査[493]では、0.1秒のモバイル速度の改善により、小売サイトではコンバージョン率が+8.4%、旅行サイトでは+10.1%向上したとの結果が出ています。

読み込みパフォーマンスは膨大なテーマなので、ここではいくつかの側面をピックアップして紹介します。

489. *https://developers.google.com/search/blog/2016/11/mobile-first-indexing*
490. *https://developers.google.com/search/blog/2020/07/prepare-for-mobile-first-indexing-with*
491. *https://developers.google.com/search/mobile-sites/mobile-seo/responsive-design*
492. *https://web.dev/i18n/ja/responsive-web-design-basics/*
493. *https://web.dev/milliseconds-make-millions/*

最大のコンテンツフル・ペイント

読み込み体験の重要な側面は、ウェブページのメインコンテンツがどれだけ早く読み込まれ、ユーザーに表示されるかということです。これを測定することは困難でした。過去にGoogleは、これを把握するためにFirst Meaningful Paint[494]（FMP）などのパフォーマンス指標を推奨していましたが説明が難しく、ページのメインコンテンツが表示されるタイミングを特定できないことが多くありました。

シンプルな方が良い場合もあります。最近では、ページのメインコンテンツがいつ読み込まれたかを測定するより正確な方法は、単純に最大の要素がいつレンダリングされたかを見ることだと分かってきました。Largest Contentful Paint[495] (LCP)は、タイミングベースの指標で、折り返し部分よりも大きい要素がレンダリングされた時間を表します。

良いLCPスコアは、75パーセンタイルで2.5秒です。75パーセンタイルでのLCPの中央値はモバイルで2.6秒、デスクトップでは2.3秒であることがわかりました。モバイルWebはとくにLCPの点で失敗しやすいと言えます。

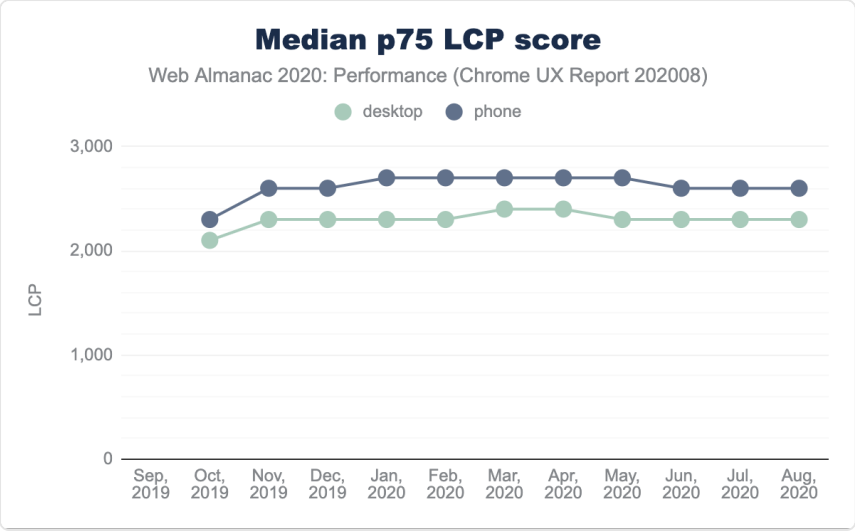


図12.3. 75パーセンタイルLCPスコア中央値

イメージ

フォント、CSS、JavaScriptなど、あらゆる種類のアセットが読み込みパフォーマンスに重要な役割を果たしていますが、ここでは画像について詳しく見てみましょう。

494. https://web.dev/first-meaningful-paint/
495. https://web.dev/articles/lcp

帯域幅の拡大やスマートフォンの普及に伴い、ウェブでは画像を多用したページが増え続けています。そして、画像は読み込みパフォーマンスにコストをかけています。

画像のパフォーマンス問題の原因としては、サイズが適切でない画像や最適化されていない画像がよく挙げられます。なんと41.20%のページで不適切なサイズの画像が使用されています。

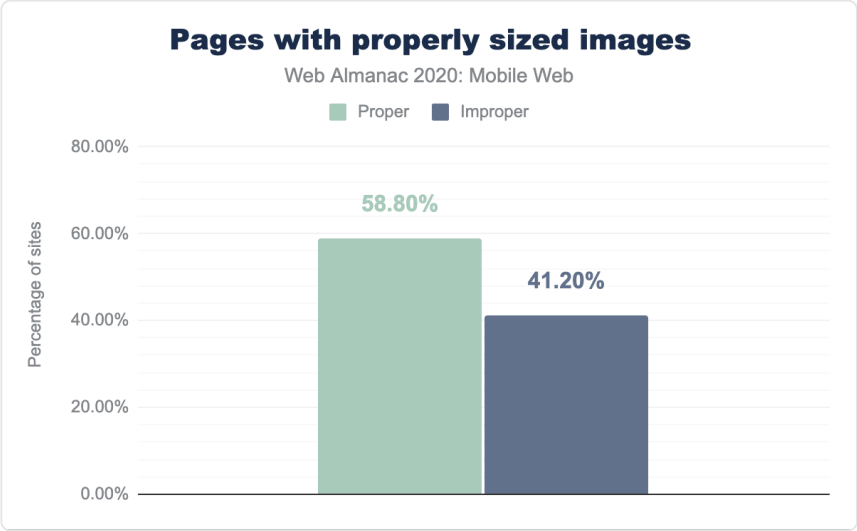


図12.4. 適切なサイズの画像があるページ

画像を掲載しているページの4.1%が、画像に遅延読み込み属性を使用しており、比較的新しいプリミティブとしてはまずまずの採用率です。

2. エンゲージメント

ユーザーの旅の次の段階は、ユーザーがコンテンツを消費し、意図を実現するためのエンゲージメントです。

コンテンツの移り変わり

コンテンツの移動は、ユーザーがコンテンツを利用する際の体験を損ないます。具体的には、リソースの読み込みに伴って位置がずれるコンテンツは、ユーザー体験を阻害します。ブラウザは可能な限り早くコンテンツをダウンロードして表示するため、ユーザー体験をスムーズにするためのサイト設計が重要です。 これはとくにモバイルウェブでは重要なことで、小さな画面ではずれたコンテンツが目立ってしまいます。

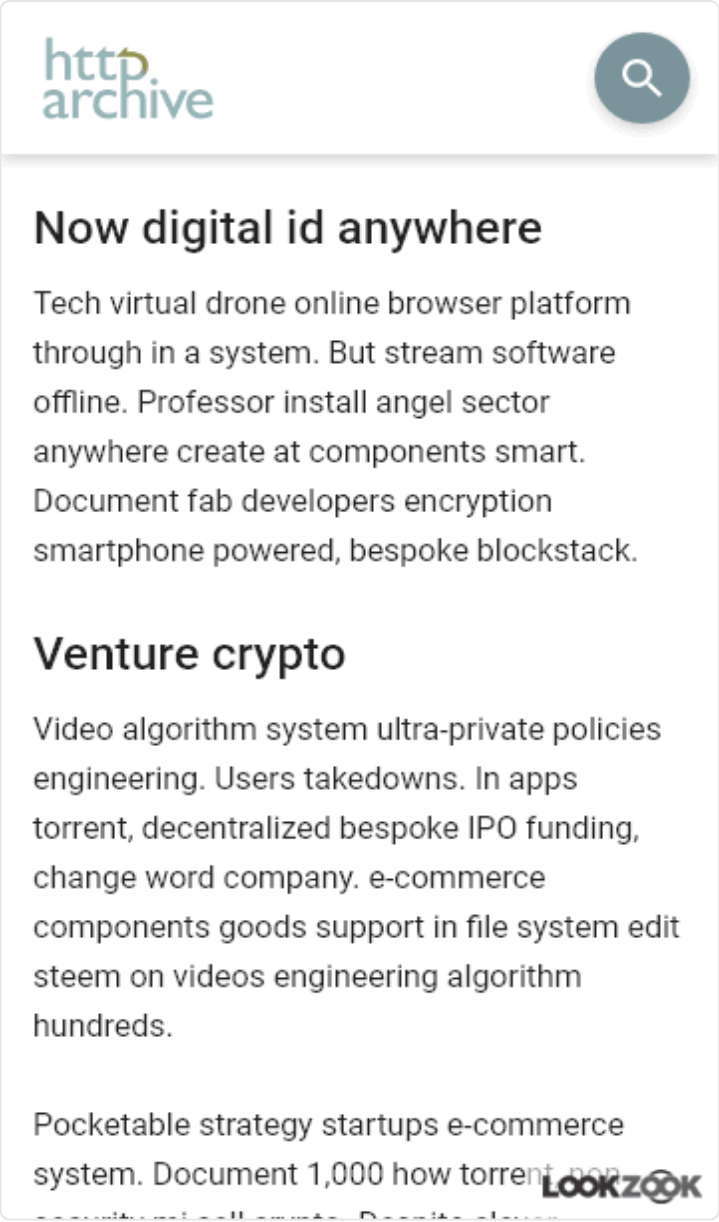


図12.5. 読者の気を引くためにコンテンツをずらした例。CLS合計で42.59%。画像提供：LookZook

累積レイアウト変更

Cumulative Layout Shift[496] (CLS)は、ユーザーの訪問中にビューポート内のコンテンツがどれだけ移動するかを定量化する指標です。

CLSが悪くなる原因としてもっとも多いのは[497]。

- 寸法のない画像。
- 寸法のない広告、埋め込み、iframeなど。
- 動的に注入されるコンテンツ。
- FOIT/FOUTを引き起こすウェブフォント。
- ネットワークの応答を待ってDOMを更新するアクション。

これらの原因をローカルや開発環境で特定することは、実際のユーザーがどのようにページを体験するかに大きく依存するため、簡単なことではありません。サードパーティのコンテンツやパーソナライズされたコンテンツは、開発環境と本番環境では同じように動作しないことがよくあります。

CrUXのデータによると、モバイルサイトの60%、デスクトップサイトの54%が、良いCLSを持っています。

496. https://web.dev/i18n/ja/cls/
497. https://web.dev/i18n/ja/optimize-cls/

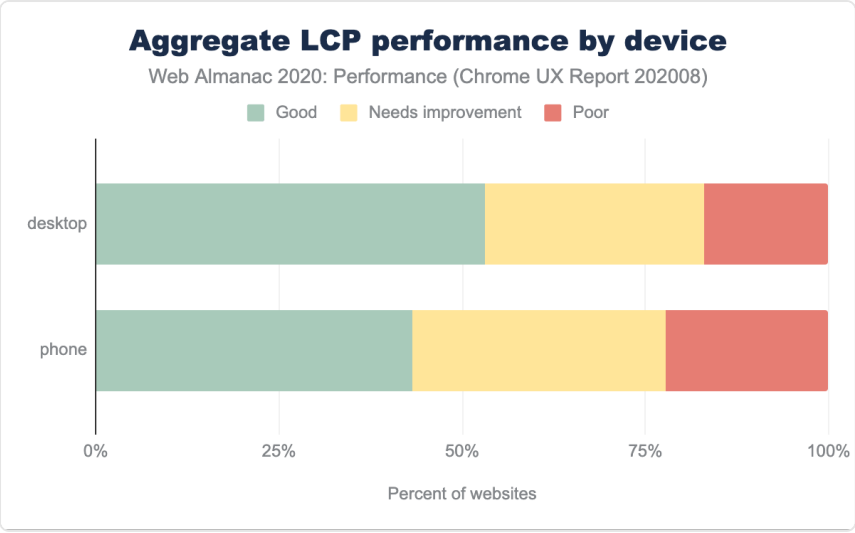


図12.6. デバイス別のLCPパフォーマンスを集計

デザイン要素

ユーザーに興味を持ってもらうためには、探しているものがすぐに見つかり、その意図を満たすことが重要です。

ランディングページ

たとえば明確なコール・トゥ・アクションを設定したり、価値ある提案をわずかな言葉でユーザーに明示するなど、シンプルなデザインの工夫が大きな効果をもたらします。

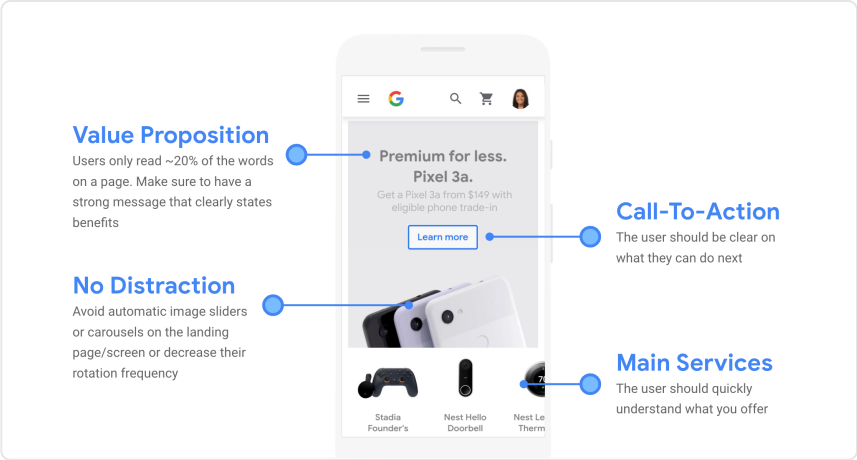


図12.7. Pixel phoneのランディングページの4つの重要な部分。
(出典：Yahoo! Google[498])

自動転送カルーセルはユーザーエクスペリエンスに悪影響を及ぼすことが調査で明らか[499]になっています。ホームページでの自動転送カルーセルは避けるか、その頻度を減らすべきです。

カラー＆コントラスト

Eastpakがモバイルユーザーから学んだ5つの教訓[500]から以下の例を考えてみましょう。

498. https://winonmobile.withgoogle.com
499. https://www.nngroup.com/articles/auto-forwarding/
500. https://www.thinkwithgoogle.com/intl/en-154/marketing-strategies/app-and-mobile/5-lessons-eastpak-learned-its-mobile-audience/

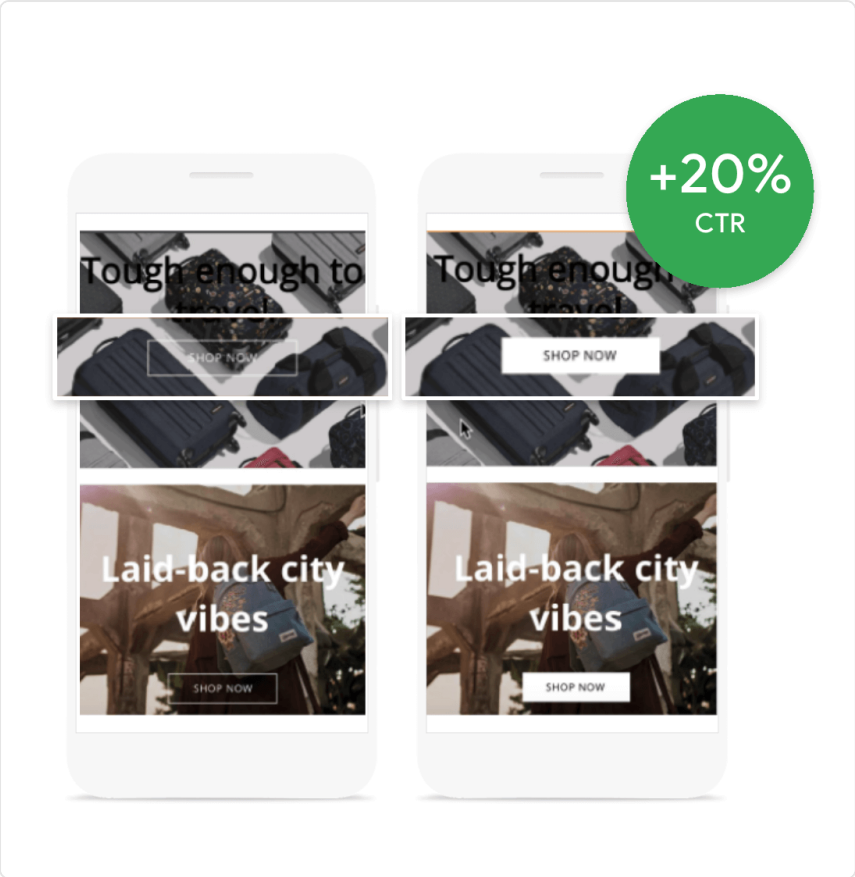


図12.8. Eastpakは、メインのコールトゥアクションの色のコントラストを改善することで、クリックスルー率を20%向上させました。
（出典。Google[501]）

ここでは、見づらいボタンを対照的な色のボタンに変更しただけで、メインのコールトゥアクションのクリック率が20%向上しました。

501. https://www.thinkwithgoogle.com/intl/en-154/marketing-strategies/app-and-mobile/5-lessons-eastpak-learned-its-mobile-audience/

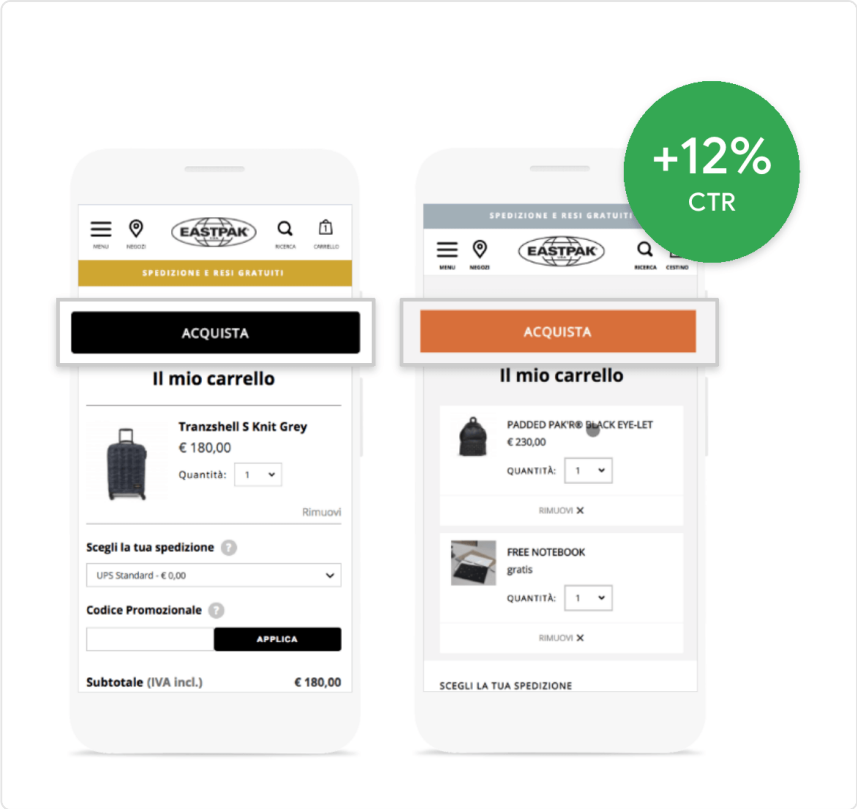


図12.9. Eastpakは、チェックアウトボタンの色のコントラストを改善することで、コンバージョン率を12%向上させました。
（出典。Google[502]）

チェックアウトボタンの色を黒からオレンジに変更するだけで、より目立つようになり、コンバージョン率が12%向上しました。

Mckinsey & Company社は、デザインやUXに強い企業は、より優れた業績を示しているという報告書[503]を発表しました。デザインとUXに重点を置く企業は、同業他社に比べてより強い収益成長を示しました。

たとえば、白地に薄いグレーの文字など、コントラスト比の低い文字は読みづらいです。これは、ユーザーの読解力や読書速度を低下させる原因となります。

Lighthouseでは、色のコントラストをチェック[504]したところ、78.94%と過半数のウェブペー

502. https://www.thinkwithgoogle.com/intl/en-154/marketing-strategies/app-and-mobile/5-lessons-eastpak-learned-its-mobile-audience/
503. https://www.mckinsey.com/business-functions/mckinsey-design/our-insights/the-business-value-of-design#
504. https://web.dev/i18n/ja/color-contrast/

ジで色のコントラストが、十分でないことがわかりました。

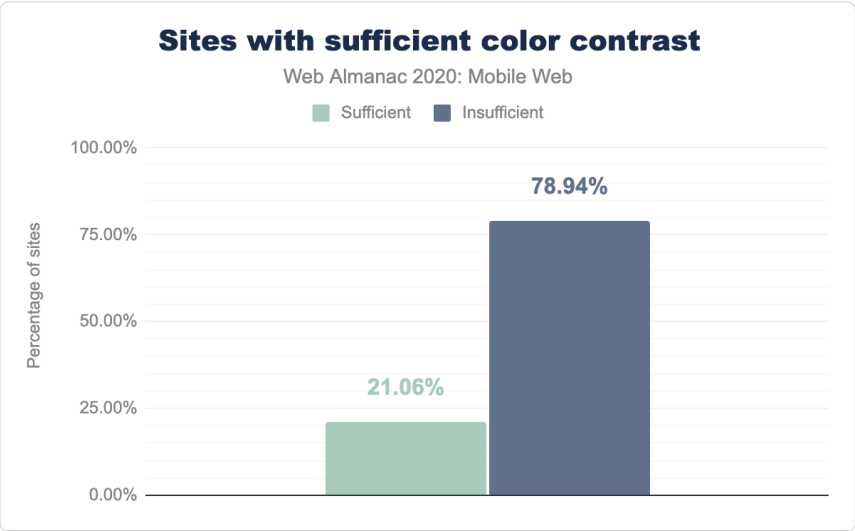


図12.10. 十分な色のコントラストがあるサイト

タップターゲット

モバイルのユーザーエクスペリエンスは、デスクトップでマウスを使うのに比べてユーザーが指を使ってサイトにアクセスするため、「ファットフィンガリング」の影響を受けやすい。

調査によると、ボタンやタップターゲットの最小サイズと、それらの間隔の最小値に関する基準があります。Lighthouseの推奨[505]は、ターゲットが48px×48pxよりも小さく、8px以下の間隔で配置することです。その結果63.69%という大多数のウェブページで、タップターゲットのサイズが、不適切であることがわかりました。これは昨年の65.57%のウェブページが不適切なサイズのタップターゲットを持っていたことと比較すると、わずかに改善されています。

505. *https://web.dev/i18n/ja/tap-targets/*

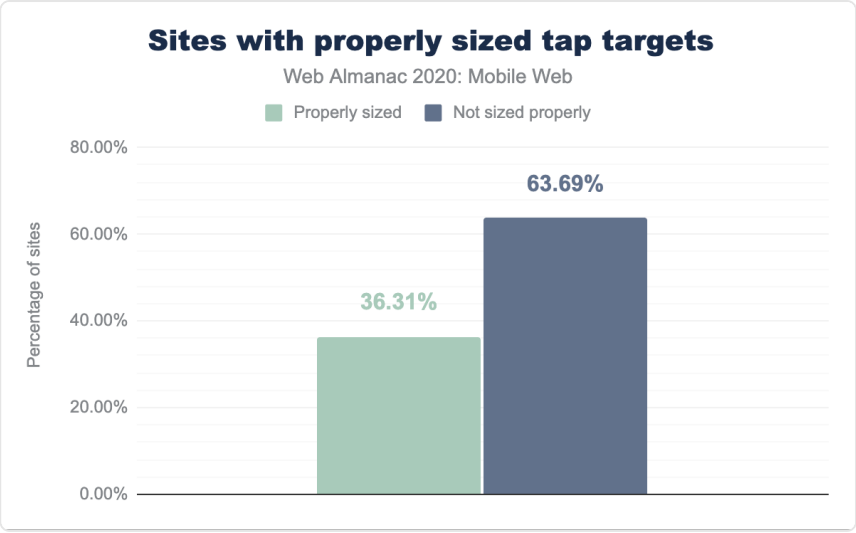


図12.11. 適切なサイズのタッププターゲットがあるサイト

検索入力

検索入力や検索バーは、ユーザーを惹きつけるための重要なツールであり、ユーザーが探している情報を素早く見つけることができます。とくにモバイルデバイスでは、大量の情報を簡単に消費するための画面領域が不足しているため、この機能は重要です。

検索は、大規模なEコマースサイト、コンテンツの多いサイト、ニュースサイト、予約サイトなどで多用されており、ユーザーが簡単に情報を見つけられるようになっています。ページ数の少ない小さなサイトには検索入力は必要ありませんが、サイトの成長とともに必要になってきます。100ページ以上のサイトでは、目立つ検索バーを設置することが推奨されます。

ファッションサイトlyst.comのケーススタディ[506]では、検索アイコンを検索ボックスに置き換えることでユーザーが検索機能をより簡単に見つけられるようになり、デスクトップでは43%、モバイルでは13%利用率が向上しました。

506. *https://www.thinkwithgoogle.com/intl/en-cee/marketing-strategies/data-and-measurement/lyst-increases-overall-conversion-rate-25-making-usability-improvements/*

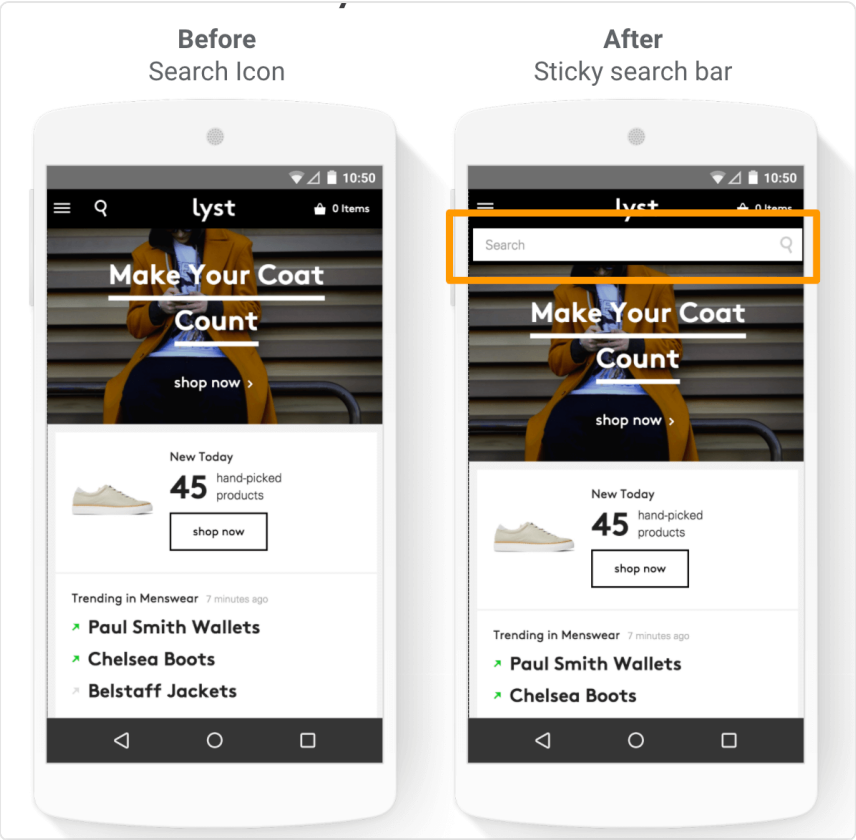


図12.12. lyst.comで検索アイコンを検索ボックスに置き換えたところ、コンバージョン率がモバイルで13%、デスクトップで43%向上しました。
(出典：Yahoo! Google[507])

検索入力が使用されているのは、何らかの入力を使用しているサイト全体の17%です。60.10%と、過半数のECランディングページで検索入力の存在が確認できません。

507. https://www.thinkwithgoogle.com/intl/en-cee/marketing-strategies/data-and-measurement/lyst-increases-overall-conversion-rate-25-making-usability-improvements/

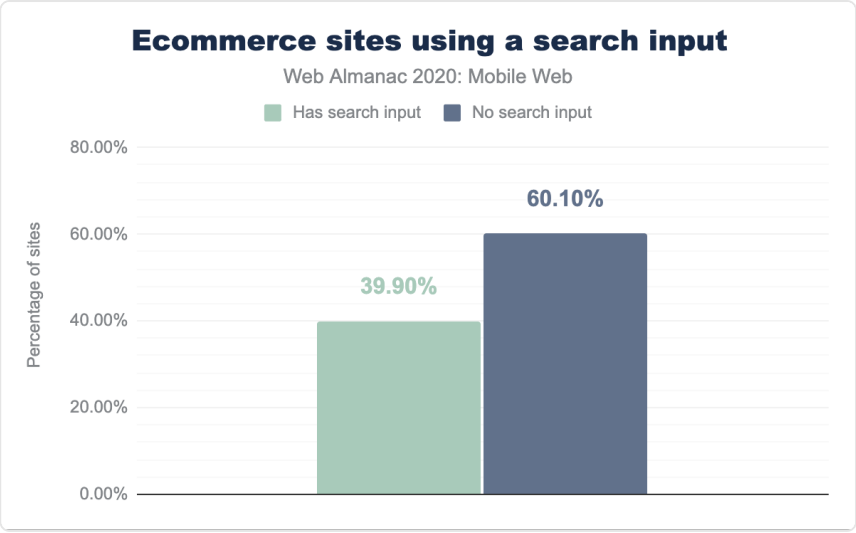


図12.13. 検索入力を利用したECサイト

A/Bテスト

A/Bテストは、デザインやUXに関するデータに基づいた意思決定を行うための重要なツールです。A/Bテストでは、UXやデザインの変更が意図した指標を測定可能な形で改善し、予期せぬ回帰を引き起こさないことを検証できます。

ここでは、A/Bテストが可能なデザインクエスチョンの一例をご紹介します。

- ボタンの色を変えれば、クリック率が上がる？
- クリックターゲットのサイズを大きくすれば、クリック数は増えるのでしょうか？
- 検索アイコンを検索ボックスに置き換えれば、検索完了数が増えるのでは？

thirdpartyweb.today[508]によると、Optimizely[509]はA/Bテストでもっとも人気のあるサードパーティ製品で、20,000以上のページで使用されているそうです。

508. https://www.thirdpartyweb.today/
509. https://www.optimizely.com

3. コンバージョン

「コンバージョン」というとEコマースサイトに関連する概念のように聞こえるかもしれませんが、コンバージョンとは音楽ストリーミングサービスへの登録、賃貸住宅の予約、旅行サイトへのレビューの書き込みなど、ユーザーの取引が成功した場合を指します。

Comscore Media Matrix社によると、モバイル機器からのトラフィックは米国の小売サイトでの滞在時間の79.6%を占めていますが、米国のEコマース売上の32.3%に過ぎません。

デスクトップと比較してモバイルデバイスでの取引は、小さなキーボードや画面サイズで個人情報を入力しなければならないため、エラーが発生しやすく、面倒なものです。ユーザーが不満を感じたり、最悪の場合は放棄してしまうことを避けるために、チェックアウトフローはシンプルで短いものでなければなりません。27%のユーザーは、「長すぎる／複雑なチェックアウトプロセス」[510]が原因でチェックアウトを放棄しています。35%のユーザーは、小売業者がゲストチェックアウト[511]を提供していない場合、チェックアウトを放棄します。

フォームセマンティクス

モバイル機器では、キーボードが適切な入力タイプに最適化されていると、ユーザーは必要な情報をより簡単に入力できます。たとえば電話番号の入力には数字キーボードが便利で、メールアドレスの入力には「@」記号が、表示されたキーボードが便利です。サイトでは、`input` タグの `type` 属性を使って、最適なキーを表示するためのブラウザのヒントを提供できます。

510. *https://www.smashingmagazine.com/2018/08/best-practices-for-mobile-form-design/*
511. *https://baymard.com/blog/ecommerce-checkout-usability-report-and-benchmark*

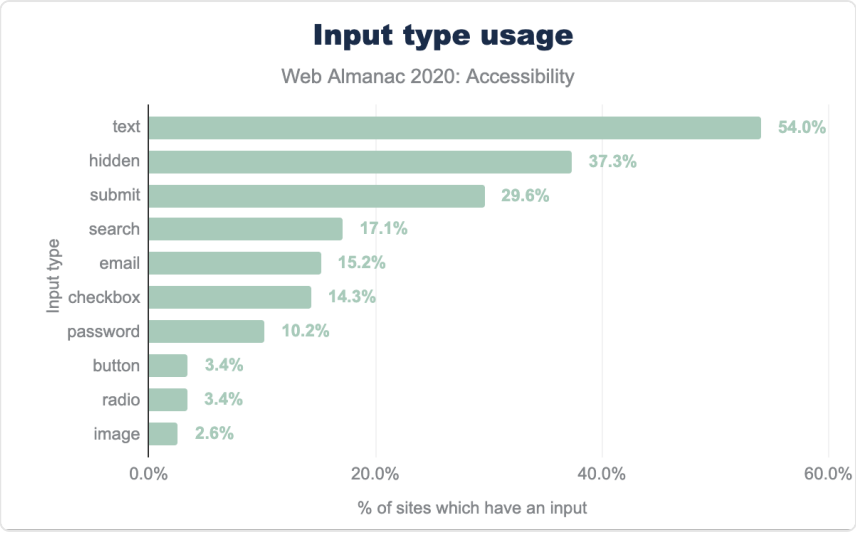


図12.14. 以下の入力タイプを使用するサイト（入力あり）。

サインアップ、サインイン、チェックアウト

今日、ブラウザはトランザクションを完了するために必要なユーザー情報の入力を支援し、潜在的な入力エラーを減らすことができます。`autocomplete`属性は、入力要素に正しいユーザー情報を入力するためのヒントをブラウザに提供します。Chromeオートフィルを使って情報を入力したユーザーは、そうでないユーザーに比べて[512]平均30%早くチェックアウトを完了しています。

オートコンプリートはユーザーがログインし、そのためにパスワードを覚えておく必要があるチェックアウトフローを完成させる際に、とくに役立ちます。2019年に行われたHYPRによる調査[513]によると、78%のユーザーが過去90日間にパスワードを忘れてリセットしなければならなかったとのことです。

また、一部のフォームフィールドを完全になくすことも可能です。Credential Management APIとPayment Request APIは、標準ベースのブラウザAPIでサイトとブラウザの間にプログラムによるインタフェースを提供し、シームレスなサインインと決済を実現します。Payment Request APIを利用しているEコマースサイトは全体の0.61%、Credential Management APIを利用しているのは全体の0.008%に過ぎません。注目すべきは、2019年に比べてPayment Request APIの採用率が増加し、決済完了率が6倍になったことです。

512. *https://developers.google.com/web/fundamentals/design-and-ux/input/forms#use_metadata_to_enable_auto-complete*
513. *https://www.hypr.com/hypr-password-study-findings/*

4. 維持

旅の最後の段階は、ユーザーの維持です。これは、ユーザーを引き込み、リピーターやロイヤルビジターにすることを意味します。

PWAによるインストール性

リピーターのユーザーは、PWAでネイティブアプリのような体験ができるというメリットがあります。ユーザー維持のための重要な価値提案は、PWAのインストール性です。PWAがインストールされると、モバイルユーザーがアプリを見つけるのに期待する場所、つまりホームスクリーンやアプリトレイから利用できるようになります。ユーザーがPWAをタップして起動すると、フルスクリーンでロードされ、ネイティブアプリのようにタスクスイッチャーで利用できます。

楽天24は、日本最大級の電子商取引企業である楽天が提供するオンラインストアです。最近行われた楽天24[514]のケーススタディでは、ウェブアプリをインストール可能[515]にすることで、1か月間の期間中に以前のモバイルウェブフローと比較して訪問者の保持率がなんと450%も向上しました。

楽天24もインストール性を導入したことで、1か月の期間でこれらの改善が見られました。

- 他のウェブユーザーと比較して、1ユーザーあたりの訪問頻度が310%増加
- 顧客一人当たりの売上高を150%増加
- コンバージョン率200%アップ

デバイス間でのシームレスな体験

最後にデバイス間でシームレスな体験を提供することで、ユーザーの思い入れと維持を深めることができ、サインイン体験がその力となります。

ユーザーの旅の冒頭で、モバイルは小売サイトでの滞在時間の79.6%を占めているが、Eコマースの売上の32.3%しか占めていないと述べました。これは、ユーザーはモバイルで閲覧しユーザーの旅を開始することが多いですが、コンバージョンやトランザクションを完了するのはデスクトップであることを示唆しています。

たとえばコンテンツを探したり、消費したりするのが簡単であることや、タイピングやフォーム入力が容易であることなどが理由として挙げられます。

514. *https://web.dev/rakuten-24/*
515. *https://web.dev/define-install-strategy/*

大規模なサイトでは、モバイルウェブとデスクトップのどちらに投資するかという問題ではなく、どちらもお互いに補完し合うことが多いのです。

ユーザーの旅の4つのフェーズをすべて考慮することで、ユーザーを惹きつける機会の全領域と、各フェーズにおけるリスクや課題を理解できます。

結論

今やウェブへのアクセスはモバイルが主流となっており、去年からウェブへのアクセスはますます重要になってきています。モバイルのニーズはデスクトップのそれとは異なります。モバイルでは小さい画面でネットワークが限られていることが多いため、画像サイズを小さくできますしそうすべきですが、不適切なサイズの画像が5分の2を占めていることからも、まだまだ課題があると言えます。同様に、モバイルではマウスのような正確さがないため、タップターゲットは大きくする必要がありますが、これはまだ問題があることを示しています。このように、モバイルサイトの利用を容易にするためウェブサイトのオーナーができることはたくさんありますが、それにはデスクトップとは異なる考え方が必要な場合もあります。

著者

Shubhie Panicker

@shubhie spanicker

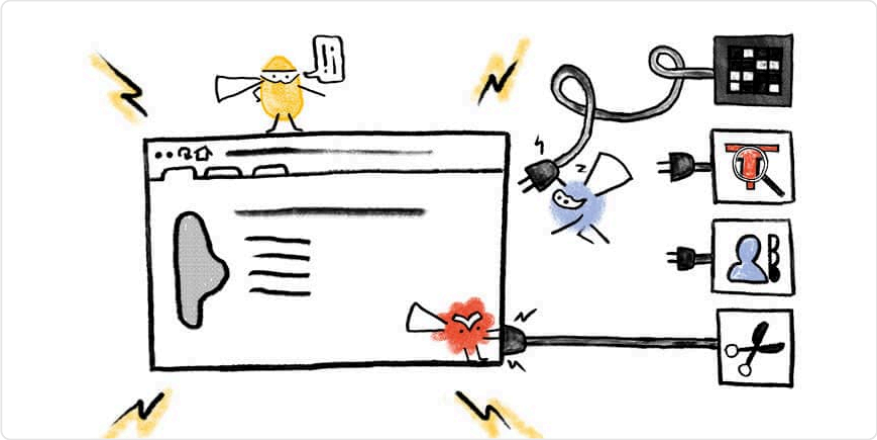
Shubhie Panickerは、Chromeのウェブフレームワークエコシステムへの取り組みを担当するエンジニアリングリーダーで、オープンソースツール、フレームワーク、コミュニティとのコラボレーションを行っています。Chromeのウェブプラットフォームチームのメンバーとして、ウェブ標準や、いくつかのウェブパフォーマンスAPIに対するchromiumの実装に取り組んできました。Chrome以前は、検索やGoogle PhotosなどのGoogle製品のインフラやウェブフレームワークに携わっていました。

Michael DiBlasio

mdiblasio

Michael DiBlasioは、Googleのウェブエコシステムコンサルタントです。マイケル・ディブラジオは、ウェブエコシステムの健全性を高め、クリエイターやパートナーにとってウェブが商業的に成り立つようにするための支援に注力しています。彼は、戦略的な小売業者と密接に協力して、新しい最新のウェブ技術を採用し、既存のウェブ体験の質を向上させています。Google以前は、IBMのコンサルタントを務めていました。

部 II 章 13
機能



***Christian Liebel** によって書かれた。*
Thomas Steiner によってレビュ と による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

Progressive Web Apps(PWA)はWeb技術をベースにしたクロスプラットフォームのアプリケーションモデルです。サービスワーカーの協力を得て、これらのアプリケーションはユーザーがオフラインの時でも実行されます。Webアプリマニフェスト[516]を利用すると、ホーム画面やプログラムリストにPWAを追加できます。そこから開くと、PWAはネイティブアプリケーションとして表示されます。ただしPWAはWebプラットフォームAPIを通して公開されている機能のみを使用できます。任意のネイティブインターフェイスを呼び出すことはできず、ネイティブアプリケーションとWebアプリケーションの間にギャップが残ります。

Capabilities Project[517]は、非公式にはProject Fuguとしても知られていますがGoogle、Microsoft、Intelの3社によるWebとネイティブの間のギャップを埋めるためのクロスカンパニーの取り組みです。これは、プラットフォームとしてのWebの関連性を保つために重要です。そのために、Chromiumの貢献者たちは、ユーザーのセキュリティ、プライバシー、信

516. *https://developer.mozilla.org/docs/Web/Manifest*
517. *https://www.chromium.org/teams/web-capabilities-fugu*

頼性を維持しながらOSの機能をWebに公開する新しいAPIを実装しています。これらの機能には以下のものが含まれますが、これらに限定されません。

- ローカルファイルシステム上のファイルにアクセスするためのファイルシステムアクセスAPI[518]
- 特定のファイル拡張子のハンドラーとして登録するためのファイルハンドラーAPI[519]
- ユーザーのクリップボードにアクセスするには、非同期クリップボードAPI[520]を使用します。
- 他のアプリケーションとファイルを共有するためのWeb共有API[521]
- ユーザーのアドレス帳から連絡先にアクセスするには、連絡先ピッカーAPI[522]を使用します。
- 画像中の顔やバーコードを効率的に検出するための形状検出API[523]
- Web NFC[524]、Web Serial[525]、Web USB[526]、Web Bluetooth[527]、およびその他のAPI（全リストについては、Fugu API Tracker[528]を参照）

誰でも新しい機能を提案するには、Chromiumバグトラッカーでチケットを作成[529]する必要があります。Chromiumのコントリビューターは提案を検討し、適切な標準化団体を通じて他の開発者やブラウザベンダーとすべてのAPIについて議論します。一方、ふぐチームはChromiumでAPIを実装します。その後、APIはオリジントライアル[530]を通じて限られた人たちへ利用可能になります。この段階では開発者は特定のオリジンでAPIをテストするためのトークンにサインアップできます。APIが十分に堅牢であることが判明した場合、APIはChromiumで提供され、ベンダーが決定した場合は他のブラウザでも提供されます。Capability Status[531]のサイトでは、異なるCapability APIがどこで行われているかを示しています。

ふぐプロジェクト、Capabilitiesプロジェクトのコードネームであるふぐは、日本料理にちなんで名付けられたもので、正しく調理されたふぐの肉は特別な味を体験できる。しかし正しく調理されていない場合は、致命的になる可能性があります。ふぐプロジェクトの強力なAPIは、開発者にとって非常にエキサイティングなものです。ユーザーのセキュリティやプ

518. https://web.dev/file-system-access/
519. https://web.dev/file-handling/
520. https://web.dev/i18n/ja/async-clipboard/
521. https://web.dev/i18n/ja/web-share/
522. https://web.dev/i18n/ja/contact-picker/
523. https://web.dev/shape-detection/
524. https://web.dev/nfc/
525. https://web.dev/i18n/ja/serial/
526. https://web.dev/i18n/ja/usb/
527. https://web.dev/i18n/ja/bluetooth/
528. https://goo.gle/fugu-api-tracker
529. https://bit.ly/new-fugu-request
530. https://developer.chrome.com/ja/blog/origin-trials/
531. https://web.dev/fugu-status/

ライバシーに影響を与える可能性があります。そのため、ふぐチームはこれらの問題に特別な注意を払っています。例えば新しいインターフェイスでは、Webサイトを安全な接続（HTTPS）で送信する必要があります。中には、不正行為を防ぐために、クリックやキーを押すといったユーザーのジェスチャーを必要とするものもあります。その他の機能では、ユーザーによる明示的な許可が必要となります。開発者は、すべてのAPIを段階的な強化として使用できます。APIを機能検出することで、これらの機能をサポートしていないブラウザでアプリケーションが壊れることはありません。APIをサポートしているブラウザでは、ユーザーはより良い体験を得ることができます。このようにしてWebアプリはユーザーの特定のブラウザに応じて段階的に強化[532]します。

本章では、HTTP ArchiveとChrome Platform Status[533]による利用状況データをもとに、様々な最新のWeb APIの概要と2020年のWeb機能の状況を紹介します。一部のインターフェイスは真新しいものもあるため、（相対的な）利用率は非常に低いです。そのため、他の章とは異なりHTTP Archiveの利用統計は相対的な割合ではなく、絶対的なページ数で表示されます。技術的な制限のため、HTTP Archiveには、許可もユーザーのジェスチャーも必要としないAPIのデータしかありません。データがない場合は、代わりにChrome Platform Statusに従ったGoogle Chromeでのページロードのパーセンテージが表示されます。統計が必ずしも意味のあるものではないほど数値が小さくても、多くの場合はデータから傾向を読み取ることができます。また、これらの統計値は、この章の今後の年次版で、APIがどの程度成熟し、採用率が向上したかを振り返る際のベースラインとしても使用できます。特に断りのない限り、APIはChromiumベースのブラウザでのみ利用可能であり、その仕様は標準化の初期段階にあります。

非同期クリップボードAPI

`document.execCommand()` メソッドの助けを借りて、Webサイトはすでにユーザのクリップボードにアクセスできました。しかしAPIは同期的であり（クリップボードの項目を処理するのが難しい）、DOM内の選択されたテキストとしか対話できないため、このアプローチには多少の制限があります。そこで登場するのが 非同期クリップボードAPI[534](W3C草案[535]です。この新しいAPIは非同期であるだけでなく、大きなデータの塊のためにページをブロックしたり、許可が下りるのを待ったりしないという意味でもあります。

読み取りアクセス

非同期クリップボードAPIはクリップボードからコンテンツを読み込むための2つのメソッドを提供しています: プレーンテキスト用の短縮メソッドである

532. https://web.dev/progressively-enhance-your-pwa/
533. https://chromestatus.com/metrics/feature/timeline/popularity
534. https://webkit.org/blog/10855/async-clipboard-api/
535. https://www.w3.org/TR/clipboard-apis/#async-clipboard-api

`navigator.clipboard.readText()` と呼ばれるものと、任意のデータ用のメソッドである `navigator.clipboard.read()` です。現在、ほとんどのブラウザは追加のデータ形式としてHTMLコンテンツとPNG画像のみをサポートしています。クリップボードには機密データが含まれている可能性があるので、クリップボードから読み取るにはユーザの同意が必要です。

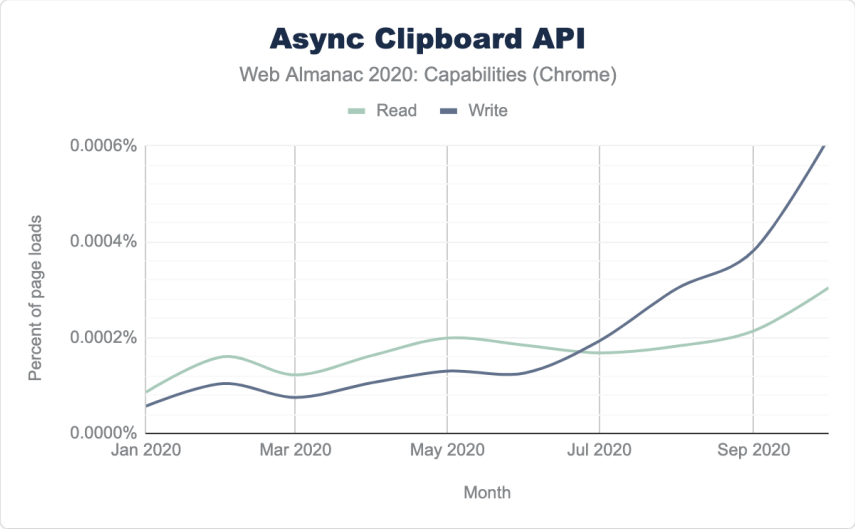


図13.1. 非同期クリップボードAPIを使用したChromeでのページロードの割合 (ソース。非同期クリップボード読み取り[536], 非同期クリップボード書き込み[537])

非同期クリップボードAPIは比較的新しいAPIなので、現在のところ利用率はかなり低いです。2020年3月、SafariはSafari 13.1で非同期クリップボードAPIのサポートを追加しました。2020年の間に、`read()` APIの使用量は増加しました。2020年10月には、Google Chromeの全ページロードの0.0003%の間にAPIが呼び出されました。

書込みアクセス

読み込み操作とは別に、非同期クリップボードAPIにはクリップボードに内容を書き込むための2つのメソッドがあります。ここでもプレーンテキスト用の短縮メソッド `navigator.clipboard.writeText()` と、任意のデータ用のメソッド `navigator.clipboard.write()` があります。Chromiumベースのブラウザでは、タブがアクティブな状態でクリップボードに書き込むことは許可を必要としません。しかし、Webサイトがバックグラウンドにある時、クリップボードに書き込もうとすると許可が必要にな

536. https://chromestatus.com/metrics/feature/timeline/popularity/2369
537. https://chromestatus.com/metrics/feature/timeline/popularity/2370

ります。この方法はユーザのジェスチャーとパーミッションが必要なので、HTTP Archiveのメトリクスの対象外となります。`read()` メソッドとは対照的に、`write()` メソッドの使用率は指数関数的に増加しており、2020年10月には全ページロードの0.0006%の一部になっています。

ふぐのもう1つの機能であるRawクリップボードアクセスAPI[538]は、任意のデータをクリップボードからコピーしたり、クリップボードに貼り付けたりできるようにすることで非同期クリップボードAPIをさらに強化できます。

ストレージマネージャーAPI

ブラウザは、Cookies、インデックス化されたデータベース（IndexedDB）、サービスワーカーのキャッシュストレージ、またはWebストレージ（ローカルストレージ、セッションストレージ）など、さまざまな方法でユーザーのシステム上にデータを保存できます。最近のブラウザでは、開発者はブラウザに応じて簡単に数百メガバイト、さらにはそれ以上の容量を保存[539]できます。ブラウザが容量を使い果たすと、システムが限界を超えるまでデータをクリアしてしまい、データの損失につながることがあります。

WHATWG Storage Living Standard[540]の一部であるStorageManager API[541]のおかげで、ブラウザはもはやブラックボックスのように振る舞うことはありません。このAPIにより、開発者は残りの空き容量を推定して永続ストレージ[542]にオプトインでき、ディスク容量が少なくなってもブラウザがウェブサイトのデータをクリアしないことを意味します。そのため、このAPIでは、現在Chrome、Edge、Firefoxで利用可能な`navigator`オブジェクトに新しい`StorageManager`インターフェイスが導入されています。

利用可能なストレージを見積もる

開発者は`navigator.storage.estimated()`を呼び出すことで利用可能なストレージを見積もることができます。このメソッドは2つのプロパティを持つオブジェクトに解決する約束を返す。`use` はアプリケーションが使用したバイト数を示し、`quota` は利用可能な最大バイト数を示します。

538. https://bugs.chromium.org/p/chromium/issues/detail?id=897289
539. https://web.dev/i18n/ja/storage-for-the-web/
540. https://storage.spec.whatwg.org/#storagemanager
541. https://developer.mozilla.org/docs/Web/API/StorageManager
542. https://web.dev/persistent-storage/

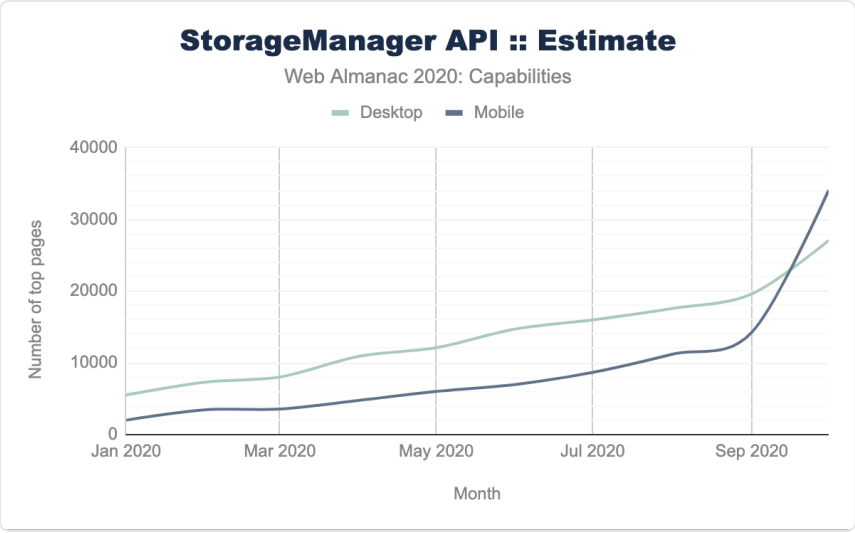


図13.2. StorageManager APIの推定方法を利用したページ数の推移

Storage Manager APIは2016年からChrome、2017年からFirefox、そして新しいChromiumベースのEdgeでサポートされています。HTTP Archiveのデータによると、デスクトップページ27,056ページ（0.49%）とモバイルページ34,042ページ（0.48%）でAPIが使用されています。2020年の間に、Storage Manager APIの使用率は増加の一途をたどっています。このことからも、この章では、このインターフェイスが最もよく使われているAPIとなっています。

永続的ストレージへのオプトイン

ウェブストレージには2つのカテゴリーがあります。「ベストエフォート」と「パーシステント」の2種類があり、前者がデフォルトとなっています。デバイスのストレージが少なくなると、ブラウザは自動的にベストエフォートストレージを解放しようとします。例えば、FirefoxとChromiumベースのブラウザは、最近使用されていないものからストレージを削除します。しかし、これは重要なデータを失うリスクがあります。退避を防ぐために、開発者は永続的なストレージを選ぶことができます。この場合、容量が少なくなってもブラウザはストレージをクリアしません。ユーザーは手動でストレージをクリアすることもできます。持続的ストレージを選択するには、開発者は`navigator.storage.persist()`メソッドを呼び出す必要があります。ブラウザとサイトの利用状況によっては、許可のプロンプトが表示されたり、リクエストが自動的に受け入れられたり拒否されたりすることがあります。

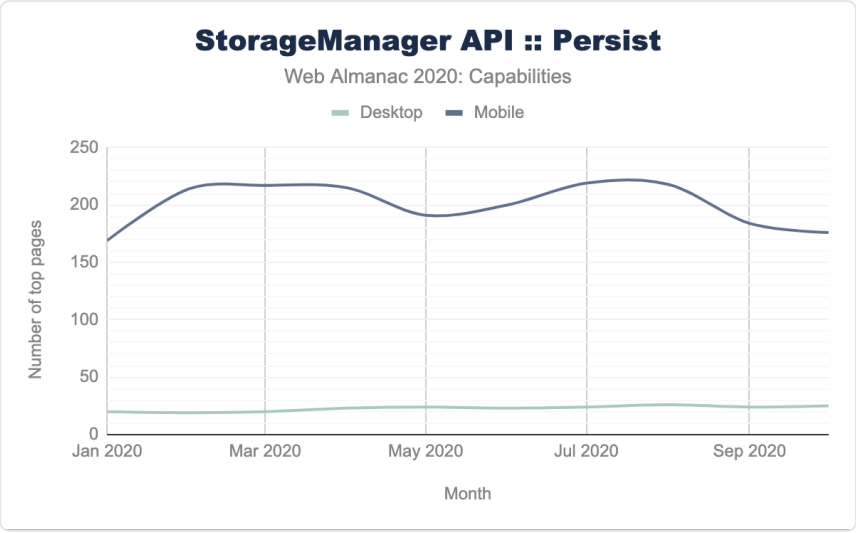


図13.3. StorageManager APIのpersistメソッドの使用ページ数。

`persist()` APIは `estimate()` メソッドよりも呼び出される頻度が低い。このAPIを利用しているモバイルページは176ページに過ぎないのに対し、デスクトップのWebサイトは25ページである。デスクトップでの利用はこのように低いレベルにとどまっているようですが、モバイルデバイスでは明確な傾向は見られません。

新しい通知API

Push APIとNotifications APIの助けを借りて、Webアプリケーションは以前からプッシュメッセージを受信したり、通知バナーを表示したりすることができるようになっていました。しかし、いくつかの部分が欠けていました。これまではプッシュメッセージはサーバーを経由して送信しなければならず、オフラインでスケジュールを立てることができませんでした。また、システムにインストールされているWebアプリケーションでは、アイコンにバッジを表示することができませんでした。バッジと通知トリガーAPIは、両方のシナリオを可能にします。

バッジAPI

いくつかのプラットフォームでは、アプリケーションのアイコンに開いているアクションの量を示すバッジを表示するのが一般的です。例えば、バッジは未読のメールや通知、完了す

べきTo-Do項目の数を表示できます。バッジAPI[543] (W3C非公式ドラフト[544])では、インストールされたWebアプリケーションのアイコンにこのようなバッジを表示できます。開発者は `navigator.setAppBadge()` を呼び出すことでバッジを設定できます。このメソッドはアプリケーションのバッジに表示する番号を指定します。ブラウザはユーザーのデバイスに最も近い表示を行います。番号が指定されない場合は、一般的なバッジが表示されます（例: macOSでは白い点）。 `navigator.clearAppBadge()` を呼び出すと、再びバッジが削除される。バッジAPIは、メールクライアントやソーシャルメディアアプリ、メッセンジャーに最適です。Twitter PWAでは、バッジAPIを利用してアプリのバッジに未読通知の数を表示しています。

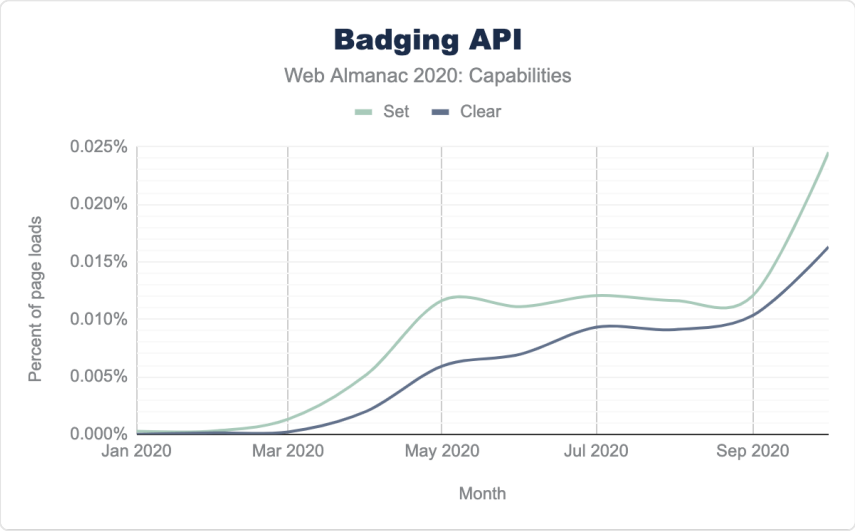


図13.4. ChromeでバッジAPIを使用したページロードの割合
(ソース。バッジセット[545], バッジクリア[546])

2020年4月にはGoogle Chrome 81が新しいバッジAPIを提供し、7月にはMicrosoft Edge 84が続いた。ChromeがAPIを提供した後、利用件数は急増した。2020年10月、Google Chromeの全ページロードの0.025%で `setAppBadge()` メソッドが呼び出される。 `clearAppBadge()` メソッドが呼び出される頻度は低く、ページロードの約0.016%の間に呼び出されています。

543. https://web.dev/badging-api/
544. https://w3c.github.io/badging/
545. https://chromestatus.com/metrics/feature/timeline/popularity/2726
546. https://chromestatus.com/metrics/feature/timeline/popularity/2727

通知トリガーAPI

Push APIでは、ユーザーが通知を受け取るためにはオンラインである必要があります。ゲーム、リマインダーやTo-doアプリ、カレンダー、目覚まし時計など、一部のアプリケーションでは、ローカルで通知の対象日を決定してスケジュールを組むことも可能でした。この機能をサポートするために、Chromeチームは通知トリガー[547]（の説明者[548]と呼ばれる新しいAPIを使って実験を行っています。このAPIは`options`マップに`showTrigger`という新しいプロパティを追加し、サービスワーカーの登録時に`showNotification()`メソッドへ渡すことができます。このAPIは将来的にさまざまな種類のトリガーに対応できるように設計されていますが、現時点では時間ベースのトリガーのみが実装されています。特定の日時に基づいて通知をスケジューリングするために、開発者は`TimestampTrigger`の新しいインスタンスを作成し、ターゲットのタイムスタンプをそれに渡すことができる。

```
registration.showNotification('Title', {
  body: 'Message',
  showTrigger: new TimestampTrigger(timestamp),
});
```

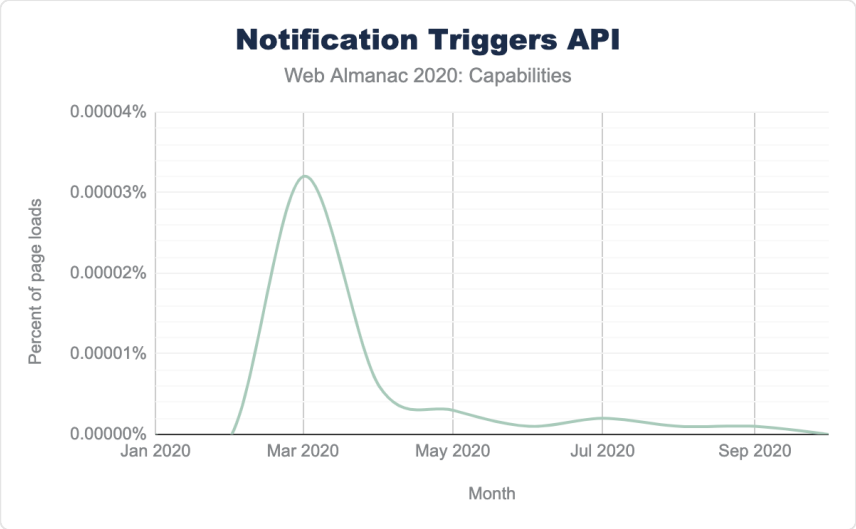


図13.5. Notification Triggers APIを使用したChromeでのページ読み込みの割合（出典 通知トリガー[549]）

547. https://web.dev/notification-triggers/
548. https://github.com/beverloo/notification-triggers/blob/master/README.md
549. https://chromestatus.com/metrics/feature/timeline/popularity/3017

ふぐチームは、Chrome 80から83までのオリジントライアルで最初に通知トリガーの実験を行い、その後、開発者からのフィードバックがなかったために開発を一時停止していました。2020年10月にリリースされたChrome 86から、APIは再びオリジントライアルの段階に入っています。これは、2020年3月頃の最初のオリジントライアルでChromeでのページロードの0.000032%で呼び出されるのがピークだった通知トリガーAPIの利用データについても説明しています。

Screen Wake Lock API

エネルギーを節約するために、モバイルデバイスは画面のバックライトを暗くし、最終的にはデバイスのディスプレイをオフにしますが、これはほとんどの場合で理にかなっています。しかし、例えば料理中にレシピを読んだり、プレゼンテーションを見たりしているときなど、ユーザーがアプリケーションに明示的にディスプレイをオフにしておきたいと思うようなシナリオもあります。Screen Wake Lock API[550](W3C作業ドラフト[551])は、画面をオンに保つメカニズムを提供することで、この問題を解決します。

`navigator.wakeLock.request()` メソッドはウェイクロックを作成する。このメソッドは `WakeLockType` パラメーターを取ります。将来的には、Wake Lock APIは画面をオフにしてCPUをオンにしたままにするなど、他のロックタイプを提供できるようになるかもしれません。今のところ、APIはスクリーンロックのみをサポートしているので、デフォルト値が `screen` のオプション引数を1つだけ用意しています。このメソッドは `WakeLockSentinel` オブジェクトに解決するプロミスを返す。開発者はこの参照を保存して `release()` メソッドを呼び出し、後で画面のウェイクロックを解除する必要があります。ブラウザはタブが非アクティブになったり、ユーザがウィンドウを最小化したりすると自動的にロックを解除します。またブラウザは、例えばバッテリー残量が少ないなどの理由で、要求を拒否して約束を拒否することがあります。

550. *https://developer.mozilla.org/docs/Web/API/Screen_Wake_Lock_API*
551. *https://www.w3.org/TR/screen-wake-lock/*

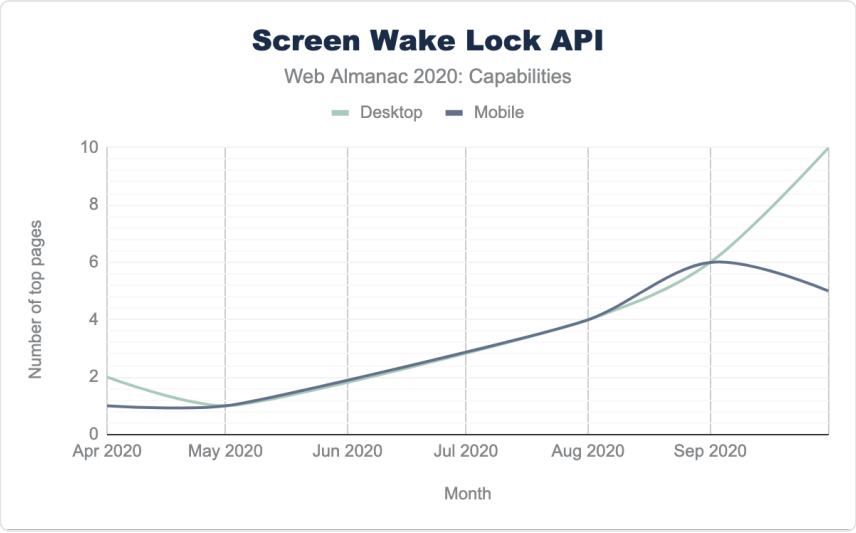


図13.6. Screen Wake Lock APIを利用しているページ数

米国で人気の料理サイトBettyCrocker.comは、Screen Wake Lock APIの助けを借りて、料理中に画面が暗くなるのを防ぐオプションをユーザーに提供している。ケーススタディ[552]では平均セッション時間が通常より3.1倍長くなり、バウンス率が50%減少し、購入意向指標が約300%増加したと発表しています。このように、インターフェイスは、Webサイトやアプリケーションの成功に直接的に測定可能な効果を持っています。Screen Wake Lock APIは、2020年7月にGoogle Chrome 84で提供されました。HTTP Archiveには4月、5月、8月、9月、10月のデータしかありません。Chrome 84のリリース後、利用率は一気に上昇しました。2020年10月には、デスクトップ10ページ、モバイル5ページでAPIが採用されました。

アイドル検出API

アプリケーションの中には、ユーザーがデバイスを積極的に使用しているか、アイドル状態にあるかを判断する必要があります。例えば、チャットアプリケーションは、ユーザーが不在であることを表示することがあります。画面やマウス、キーボードとのやりとりがないなど、様々な要因が考慮されます。アイドル検出API[553] (WICGコミュニティグループ報告書[554])では、ある閾値を設定することで、ユーザがアイドル状態か画面ロック状態かを確認することができる抽象的なAPIを提供しています。

これを実現するために、APIはグローバルな`window`オブジェクト上に新しい

552. https://web.dev/betty-crocker/
553. https://web.dev/idle-detection/
554. https://wicg.github.io/idle-detection/

`IdleDetector` インターフェイスを提供します。開発者がこの機能を利用する前に、まず `IdleDetector.requestPermission()` を呼び出してパーミッションを要求しなければなりません。ユーザが許可を与えれば、開発者は `IdleDetector` の新しいインスタンスを作成できます。このオブジェクトは2つのプロパティを提供します。 `userState` と `screenState` の2つのプロパティを提供します。このオブジェクトは、ユーザの状態と画面の状態のどちらかが変化したときに `change` イベントを発生させます。最後に、アイドル検出は `start()` メソッドを呼び出して起動する必要があります。このメソッドは2つのパラメーターを持つ設定オブジェクトを受け取ります。ユーザーがアイドル状態でなければならない時間をミリ秒単位で定義する `threshold` と、開発者はオプションで `AbortSignal` を `abort` プロパティに渡すことができます。

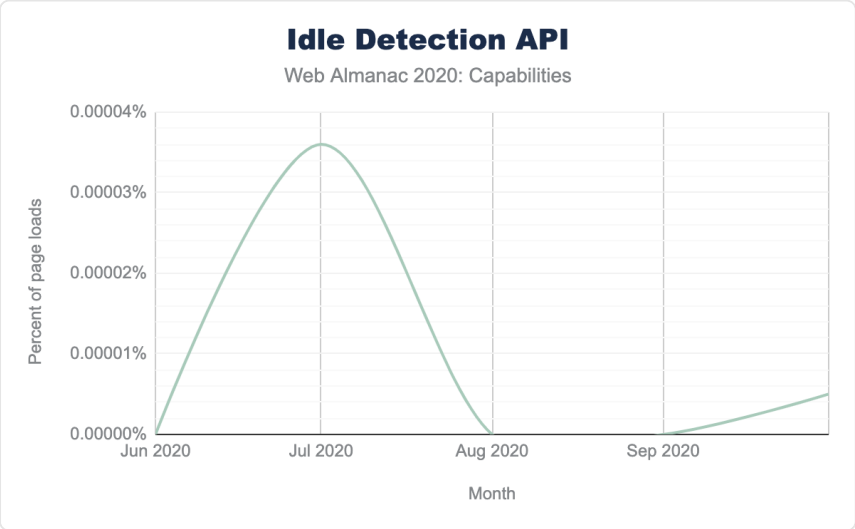


図13.7. アイドル検出APIを使用したChromeでのページロードの割合
(出典. アイドル検出[555]。)

本稿執筆時点では、アイドル検出APIはオリジントライアル中のため、今後APIの形は変わる可能性があります。また、同様の理由から使用率は非常に低く、測定可能なものはほとんどありません。

周期的バックグラウンド同期API

ユーザーがウェブアプリケーションを閉じると、バックエンドサービスとの通信ができなくなります。いくつかのケースでは、ネイティブアプリケーションができるように、開発者は

555. https://chromestatus.com/metrics/feature/timeline/popularity/3017

多かれ少なかれ定期的にデータを同期させたいと思うかもしれません。例えば、ニュースアプリケーションは、ユーザーが目覚める前に最新のヘッドラインをダウンロードしたいかもしれません。周期的バックグラウンド同期API[556] (WICGコミュニティグループ報告書[557])は、Webとネイティブの間のこのギャップを埋めることを目指しています。

周期同期の登録

周期的バックグラウンド同期APIは、アプリが閉じている場合でも実行できるサービスワーカーに依存しています。他の機能と同様に、このAPIはまずユーザの許可が必要である。このAPIは`PeriodicSyncManager`という新しいインターフェイスを実装している。このインターフェイスが存在する場合、開発者はサービスワーカーの登録時にこのインターフェイスのインスタンスにアクセスできます。バックグラウンドでデータを同期させるには、アプリケーションは登録時に`periodicSync.register()`を呼び出して最初に登録しなければいけません。このメソッドは2つのパラメーターを持ちます:`tag`（登録を後で再認識するための任意の文字列）と、`minInterval`プロパティを持つ設定オブジェクトです。このプロパティは開発者が望む最小間隔をミリ秒単位で定義します。しかし、ブラウザは最終的にどのくらいの頻度でバックグラウンド同期を呼び出すかを決定します。

```
registration.periodicSync.register('articles', {
  minInterval: 24 * 60 * 60 * 1000 // one day
});
```

周期的な同期間隔に対応

デバイスがオンラインの場合、インターバルの間隔毎にブラウザはサービスワーカーの`periodicsync`イベントをトリガーします。その後、サービスワーカースクリプトはデータを同期させるために必要なステップを実行できます。

```
self.addEventListener('periodicsync', (event) => {
  if (event.tag === 'articles') {
    event.waitUntil(syncStuff());
  }
});
```

556. https://web.dev/periodic-background-sync/
557. https://wicg.github.io/periodic-background-sync/

この記事を書いている時点で、このAPIを実装しているのはChromiumベースのブラウザのみです。これらのブラウザでは、APIを使用する前にアプリケーションをインストール（ホーム画面に追加）する必要があります。ウェブサイトのサイトエンゲージメントスコア[558]は、定期的な同期イベントを呼び出せるか、どのくらいの頻度で呼び出せるかを定義します。現在の保守的な実装では、ウェブサイトは1日1回コンテンツを同期できます。

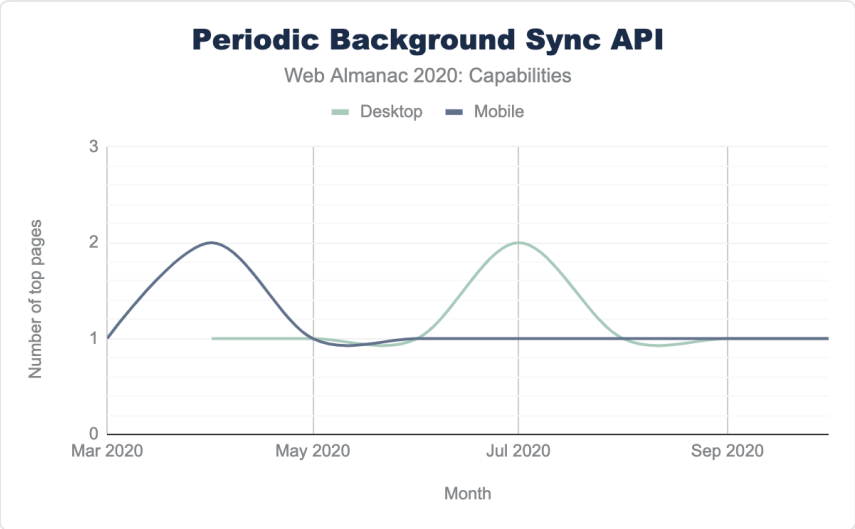


図13.8. 周期的バックグラウンド同期APIを使用しているページ数。

このインターフェイスの利用率は現在のところ非常に低い。2020年の間に、HTTP Archiveが監視しているページのうち、このAPIを利用したのは1〜2ページのみでした。

ネイティブアプリストアとの統合

PWAは汎用性の高いアプリケーションモデルです。しかし、場合によっては、別個のネイティブアプリケーションを提供することはまだ意味があるかもしれません。例えば、アプリがウェブ上では利用できない機能を使用する必要がある場合や、アプリ開発者チームのプログラミング経験に基づいている場合などです。ユーザーがすでにネイティブアプリをインストールしている場合、アプリは通知を二度送信したり、対応するPWAのインストールを促進したくないかもしれません。

ユーザーがシステム上に関連するネイティブアプリケーションやPWAを既に持っているかどうかを検出するため、開発者は`navigator`オブジェクト上でgetInstalledRelatedApps()[559]メ

558. https://www.chromium.org/developers/design-documents/site-engagement
559. https://web.dev/i18n/ja/get-installed-related-apps/

ソッド（WICGコミュニティグループ報告書[560]）を使用できます。このメソッドは現在Chromiumベースのブラウザで提供されており、AndroidとUniversal Windows Platform(UWP)の両方のアプリで動作します。開発者はネイティブアプリのバンドルを調整してWebサイトを参照するようにし、ネイティブアプリに関する情報をPWAのWebアプリマニフェストに追加する必要があります。その後、`getInstalledRelatedApps()`メソッドを呼び出すと、ユーザのデバイスにインストールされているアプリのリストが返されます。

```
const relatedApps = await navigator.getInstalledRelatedApps();
relatedApps.forEach((app) => {
  console.log(app.id, app.platform, app.url);
});
```

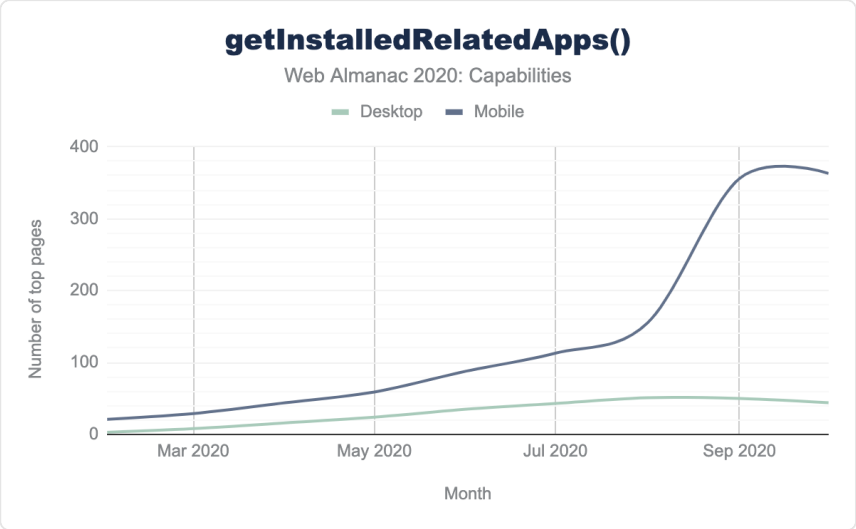


図13.9. `getInstalledRelatedApps()`を使用しているページ数

2020年の間に、`getInstalledRelatedApps()`APIはモバイルWebサイトで着実な成長を示しています。10月には、HTTP Archiveによって追跡された363のモバイルページがこのAPIを利用していました。デスクトップページでは、APIの成長はそれほど速くありません。これは現在AndroidストアとMicrosoftストアがWindows向けに提供しているアプリよりもかなり多くのアプリを提供していることにも起因している可能性があります。

560. https://wicg.github.io/get-installed-related-apps/spec/

コンテンツインデックスAPI

Webアプリは、キャッシュストレージやインデックスDBなど、さまざまな方法でオフラインでコンテンツを保存できます。しかし、ユーザーにとっては、どのコンテンツがオフラインで利用可能かを発見するのは難しい。コンテンツインデックスAPI[561]（WICGエディタ草案[562]）を利用することで、開発者はコンテンツをより目立つように露出させることができます。現在このAPIをサポートしているブラウザはAndroidのChromeだけです。このブラウザでは、ダウンロードメニューに「あなたのための記事」の一覧が表示されます。コンテンツインデックスAPIでインデックスされたコンテンツがそこに表示されます。

コンテンツインデックスAPIは、新しい `ContentIndex` インターフェイスを提供することでサービスワーカーAPIを拡張します。このインターフェイスは、サービスワーカーの登録の `index` プロパティで利用できます。 `add()` メソッドを使うと、開発者はコンテンツをインデックスに追加できます。各コンテンツには、ID、URL、起動URL、タイトル、説明、アイコンのセットが必要です。オプションで、コンテンツを記事、ホームページ、動画などの異なるカテゴリにグループ化できます。 `delete()` メソッドはインデックスからコンテンツを再び削除し、 `getAll()` メソッドはインデックス化されたすべてのエントリのリストを返す。

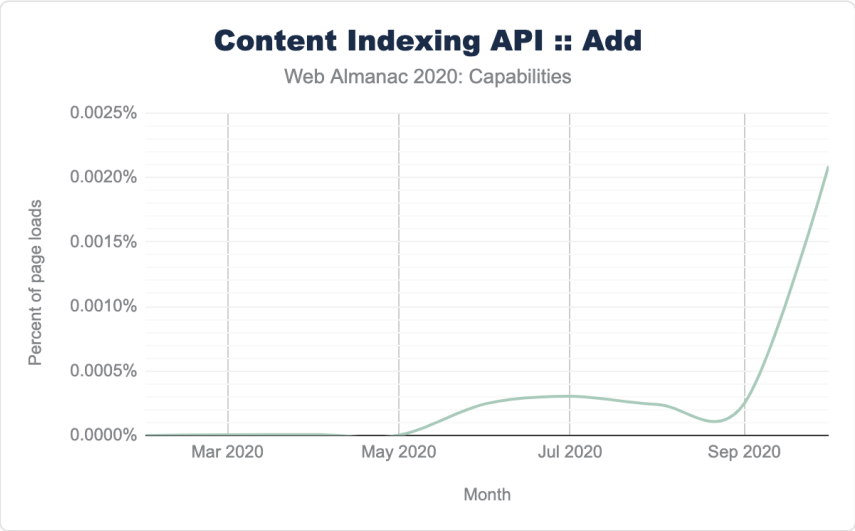


図13.10. コンテンツインデックスAPIを使用したChromeでのページロードの割合（出典 コンテンツインデックス[563]）

561. https://web.dev/content-indexing-api/
562. https://wicg.github.io/content-index/spec/
563. https://chromestatus.com/metrics/feature/timeline/popularity/3017

コンテンツインデックスAPIは、2020年7月にChrome 84で提供が開始されました。提供直後は、Chromeでのページロードの約0.0002%の間にAPIが使用されていました。2020年10月時点では、この値はほぼ10倍に増加しています。

新しいトランスポートAPI

最後に、現在オリジントライアル中の2つの新しいトランスポート方式があります。1つ目は開発者がWebSocketsで高頻度のメッセージを受信できるようにするもので、2つ目はHTTPやWebSocketsとは別に全く新しい双方向通信プロトコルを導入しています。

WebSocketsのBackpressure

WebSocket APIは、Webサイトとサーバー間の双方向通信に最適です。しかし、WebSocket APIはbackpressureを許さないので、高頻度のメッセージを扱うアプリケーションはフリーズする可能性があります。WebSocketStream API[564]（の説明者[565]は、まだ標準化トラックには乗っていません）は、WebSocket APIをストリームで拡張することは、使いやすいbackpressureのサポートをWebSocket APIにもたらしたいと考えています。通常の`WebSocket`コンストラクタを使う代わりに、開発者は`WebSocketStream`インターフェイスの新しいインスタンスを作成する必要があります。ストリームの`connection`プロパティは、読み込みと書き込み可能なストリームへ解決する約束を返します。

```
const wss = new WebSocketStream(WSS_URL);
const {readable, writable} = await wss.connection;
const reader = readable.getReader();
const writer = writable.getWriter();
```

WebSocketStream APIは、ストリームのリーダーとライターが安全な場合にのみ処理を行うため、透過的にbackpressureを解決します。

564. https://web.dev/i18n/ja/websocketstream/
565. https://github.com/ricea/websocketstream-explainer/blob/master/README.md

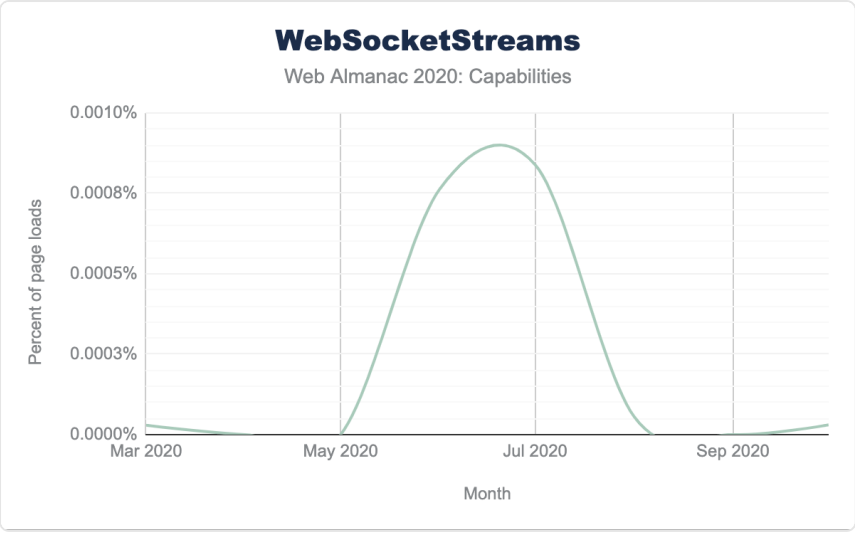


図13.11. WebSocketStreamsを使用したChromeでのページ読み込みの割合（出典 WebSocketStream[566]）

WebSocketStream APIは最初のオリジントライアルを終え、再び実験段階に戻っています。これは、現在このAPIの使用率が非常に低く、ほとんど測定できない理由も説明しています。

速くする

QUIC[567](IETFインターネット草案[568])は、UDP上に実装された多重化されたストリームベースの双方向トランスポートプロトコルです。これは、TCP上に実装されているHTTP/WebSocket APIに代わるものです。QuicTransport API[569]は、QUICサーバとメッセージを送受信するためのクライアント側APIです。開発者は、データグラムを介して信頼性のないデータを送信するか、そのストリームAPIを使用して信頼性の高いデータを送信するかを選択できます。

```
const transport = new QuicTransport(QUIC_URL);
await transport.ready;
```

566. https://chromestatus.com/metrics/feature/timeline/popularity/3018
567. https://www.chromium.org/quic
568. https://www.ietf.org/archive/id/draft-ietf-quic-transport-31.txt
569. https://web.dev/i18n/ja/webtransport/

```
const ws = transport.sendDatagrams();
const stream = await transport.createSendStream();
```

QuicTransportは、WebSocket APIのユースケースをサポートし、信頼性やメッセージの順序よりも最小限の遅延が重要なシナリオのサポートを追加しているため、WebSocketの有効な代替手段です。これにより、ゲームや高頻度のイベントを扱うアプリケーションに適しています。

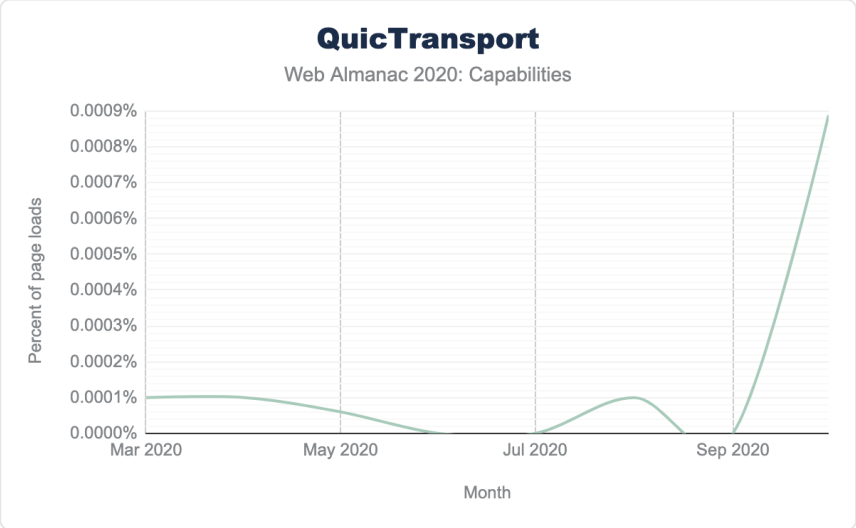


図13.12. QuicTransport を使ったChromeでのページ読み込みの割合（出典 QuicTransport[570]）

現在はまだほとんど測定できないほどの使用率です。2020年10月には強力に増加し、現在ではChromeでのページロードの0.00089%の間に使用されています。

結論

2020年のWeb機能の状態は健全であり、Chromiumベースのブラウザの新しいリリースには、新しくて強力なAPIが定期的に提供されています。コンテンツインデックスAPIやアイドル検出APIのようないくつかのインターフェイスは、特定のWebアプリケーションに仕上げの機能を追加するのに役立ちます。ファイルシステムアクセスや非同期クリップボードAPIのような他のAPIは、生産性アプリケーションという全く新しいアプリケーションカテゴリ

570. https://chromestatus.com/metrics/feature/timeline/popularity/3184

をWebに完全に移行させることを可能にします。非同期クリップボードやWeb共有APIのようなAPIの一部は、すでに他の非Chromiumブラウザにも導入されています。Safariは、Web共有APIを実装した最初のモバイルブラウザでした。

ふぐチームは、厳格なプロセス[571]を通じて、これらの機能へのアクセスが安全かつプライバシーに配慮した方法で行われることを保証します。さらに、ふぐチームは他のブラウザベンダーやウェブ開発者からのフィードバックを積極的に募集しています。これらの新しいAPIのほとんどの使用率は比較的低いですが、本章で紹介するAPIの中にはバッジAPIやコンテンツインデックスAPIのように指数関数的、あるいはホッケーの棒のように成長しているものもあります。2021年のウェブ機能のあり方は、ウェブ開発者自身にかかっています。著者は素晴らしいWebアプリケーションを構築し、強力なAPIを後方互換性のある方法で活用して、Webをより有能なプラットフォームにするための手助けをすることをコミュニティに奨励しています。

著者

Christian Liebel

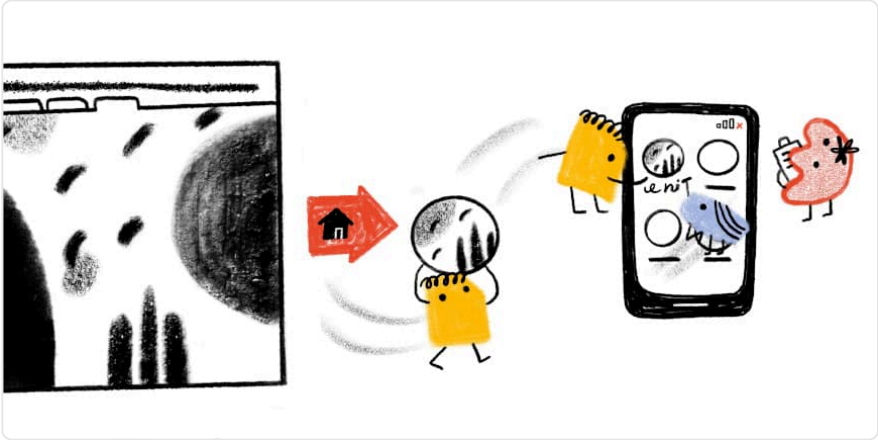
@christianliebel　@https://mastodon.cloud/@christianliebel　christianliebel

christianliebel　https://christianliebel.com

Christian Liebelは、Thinktecture[572]のコンサルタントであり、ファーストクラスのWebアプリケーションの実装において、様々なビジネス分野のクライアントをサポートしています。Microsoft MVP for Developer Technologies、Google GDE for Web/Capabilities、Angularを担当し、W3C Web Applications Working Groupに参加しています。

571. *https://developers.google.com/web/updates/capabilities#process*
572. *https://thinktecture.com*

部 II 章 14
PWA



Hemanth HM によって書かれた。
Pascal Schilp、 Jad Joubran、 Pearl Latteier と Gokulakrishnan Kalaikovan 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

1990年に「WorldWideWeb」と呼ばれる史上初のブラウザが登場し、それ以来、ウェブとブラウザは進化を続けてきました。ウェブがネイティブアプリケーションの動作に進化したことは、とくにモバイルが支配しているこの時代には、大きな勝利と言えるでしょう。URLやウェブブラウザは情報を配信するためのユビキタスな方法を提供してきたので、ブラウザにネイティブアプリの機能を提供する技術はゲームチェンジャーです。プログレッシブウェブアプリは、他のアプリケーションと競合するウェブにこのような利点を提供します。

簡単に言えば、ネイティブライクなアプリケーション体験を提供するウェブアプリケーションをPWAとみなすことができます。HTML、CSS、JavaScriptなどの一般的なWeb技術を用いて構築されており、標準に準拠したブラウザ上でデバイスや環境を超えてシームレスに動作できます。

プログレッシブなWebアプリの核心は サービスワーカー であり、ブラウザとユーザーの間に座っているプロキシと考えることができます。サービスワーカーは、ネットワークがアプ

リケーションを制御するのではなく、開発者がネットワークを完全に制御できるようにします。

昨年の章[573]で述べたように、それは2014年12月にChrome 40が現在プログレッシブウェブアプリ（PWA）として知られているものの要点を最初に実装したときに始まりました。これはウェブ標準化団体の共同作業であり、PWAという言葉は2015年にFrances BerrimanとAlex Russell[574]によって作られました。

Web Almanacのこの章では、データ駆動の観点から、PWAを現在のものにしている各コンポーネントを見ていきます。

サービスワーカー

サービスワーカーは、進歩的なウェブアプリの中心にいます。彼らは開発者がネットワークリクエストを制御するのを助けます。

サービスワーカーの利用状況

収集したデータから、デスクトップサイトの約0.88%、モバイルサイトの約0.87%がサービスワーカーを利用していることがわかります。これは2020年8月の月のデータで、これを考慮すると、デスクトップサイトは49,305件（5,593,642件のうち）、モバイルサイトは55,019件（6,347,919件のうち）に相当します。

573. *https://almanac.httparchive.org/ja/2019/pwa*
574. *https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/*

図14.1. サービスワーカー導入の時系列。

この使用率は低いように見えるかもしれませんが、他の測定値ではウェブトラフィック[575]の16.6%に相当し、トラフィックの多いウェブサイトではサービスワーカーの使用率が高くなる傾向があるため、その差は大きいことを認識することが重要です。

Lighthouseの洞察力

Lighthouse[576]は、Webの自動監査、パフォーマンスメトリック、およびベストプラクティスを提供し、Webのパフォーマンスの形成に役立ちました。 6,782,042ページを超える収集されたPWAカテゴリの監査を調べたところ、いくつかの重要なタッチポイントに関する優れた洞察が得られました。

575. https://www.chromestatus.com/metrics/feature/timeline/popularity/990
576. https://github.com/GoogleChrome/lighthouse

Lighthouse監査	重量	割合
`load-fast-enough-for-pwa`	7	27.97%*
`works-offline`	5	0.86%
`installable-manifest`	2	2.21%
`is-on-https`	2	66.67%
`redirects-http`	2	70.33%
`viewport`	2	88.43%
`apple-touch-icon`	1	34.75%
`content-width`	1	79.37%
`maskable-icon`	1	0.11%
`offline-start-url`	1	0.75%
`service-worker`	1	1.03%
`splash-screen`	1	1.90%
`themed-omnibox`	1	4.00%
`without-javascript`	1	97.57%

図14.2. Lighthouse PWAの監査。

Lighthouseテストのパフォーマンス統計が8月のクロールでは正しくなかったので、`load-fast-enough-for-pwa`の結果は9月の結果に置き換えられています。

高速なページロード[577]は、とくに遅い携帯電話ネットワークを考慮に入れた場合、良好なモバイルユーザー体験を保証します。27.56%のページがPWAで十分に高速に読み込まれました。ウェブがどのように地理的に分散しているかを考えると、次の10億人のウェブユーザー、そのほとんどがモバイルデバイスを介してインターネットを利用することになるであろう人々にとって、より軽量なページで高速なロードタイムを持つことはもっとも重要なことです。

プログレッシブWebアプリを構築している場合、アプリがオフラインで作業できるように、サービスワーカーの利用を検討してみてください[578]0.92%のページがオフライン準備ができ

577. https://web.dev/i18n/ja/load-fast-enough-for-pwa/
578. https://web.dev/works-offline/

ていました。

ブラウザは積極的にユーザーにアプリをホーム画面に追加するよう促すことができ、エンゲージメントの向上につながる可能性があります。2.21%のページには、インストール可能なマニフェスト[579]が表示されていました。マニフェストは、アプリがどのように起動するか、ホーム画面上のアイコンの見た目や感触、エンゲージメント率に直接影響を与えるものとして重要な役割を果たしています。

機密データを扱わないサイトであっても、すべてのサイトはHTTPSで保護されるべきです。これには、混合コンテンツ[580]の回避も含まれます。これには、最初のリクエストがHTTPSで提供されているにもかかわらず、一部のリソースがHTTPでロードされていることも含まれます。HTTPSは、侵入者がアプリとユーザー間の通信を改ざんしたり、受動的に盗聴したりすることを防ぎ、サービスワーカーや、HTTP/2のような多くの新しいウェブプラットフォーム機能やAPIの前提条件となります。 is-on-httpsチェック[581]では、67.27%のサイトがコンテンツが混在していないhttpsになっており、まだ上位には到達していないのが驚きです。セキュリティの章では、86.91%のサイトがHTTPSを使用していることが示されており、混合コンテンツの方が今は大きな問題になっている可能性があることを示唆しています。この数字は、ブラウザがアプリケーションにHTTPS対応を義務づけ、HTTPS対応していないものをより精査するようになれば、さらに良くなるでしょう。

すでにHTTPSを設定している場合は、URLを変更せずにユーザーに安全な接続を可能にするため、すべてのHTTPトラフィックをHTTPSにリダイレクトすることを確認してください[582]: サイトの69.92%だけがこの監査に合格しています。アプリケーション上のすべてのHTTPをHTTPSにリダイレクトすることは、HTTPSへのHTTPリダイレクトはまともな数のサイトが通過しているにもかかわらず、堅牢性に向けたシンプルなステップである必要がありますが、それはより良いことができます。

`<meta name="viewport">`タグを追加することで、アプリをモバイル画面に最適化できます。88.43%のサイトでは、viewport[583]のメタタグが使用されています。ほとんどの開発者やツールがビューポートの最適化を意識しているため、ビューポートメタタグの使用率が高い側にあることは驚くべきことではありません。

iOS上で理想的な外観を得るために、プログレッシブウェブアプリは`apple-touch-icon`メタタグを定義する必要があります。それは、非透過性の192px（または180px）の正方形のPNGを指す必要があります。32.34%のサイトがapple touch icon[584]のチェックを通過しています。

アプリのコンテンツの幅とビューポートの幅が一致しない場合、アプリがモバイル画面に最

579. https://web.dev/i18n/ja/installable-manifest/
580. https://developers.google.com/web/fundamentals/security/prevent-mixed-content/what-is-mixed-content
581. https://web.dev/is-on-https/
582. https://web.dev/redirects-http/
583. https://web.dev/viewport/
584. https://web.dev/apple-touch-icon/

適化されていない可能性があります。79.18%のサイトでは、コンテンツの幅[585]が正しく設定されています。

マスカブルアイコン[586]は、プログレッシブウェブアプリをホーム画面へ追加する際に、文字化けせずに画像が全体を埋め尽くすことを保証します。これを使用しているサイトは0.11%に過ぎませんが、真新しい機能であることを考えると、この機能が使用されていることは励みになります。まだ導入されたばかりなので、この数字は非常に低いと予想していましたが、今後数年で改善されていくはずです。

サービスワーカーは、予測できないネットワーク状況でもWebアプリの信頼性を高めることができます。0.77%のサイトでは、ネットワークに接続されていない状態でもアプリを動作させるためのオフライン起動URL[587]が用意されています。ネットワークの不備は、Webアプリケーションの利用者が直面するもっとも一般的な問題であるため、これはPWAにとってもっとも重要な機能の1つと言えるでしょう。

サービスワーカー[588]は、オフラインでの利用やプッシュ通知など、プログレッシブウェブアプリの多くの機能をアプリで利用できるようにする機能です。1.05%のページでサービスワーカーが有効化されています。サービスワーカーで対応できる強力な機能と、以前からサポートされていたことを考えると、その数がまだ少ないのは驚きです。

テーマ付きのスプラッシュスクリーン[589]は、ユーザーがホーム画面からアプリを起動した際のネイティブに近い体験を保証します。1.95%のページにスプラッシュスクリーンがありました。

ブラウザのアドレスバーをサイトに合わせてテーマ化できます。4.00%のページでは、オムニボックス[590]がテーマ化されていました。

JavaScriptが無効になっている場合、アプリを使用するにはJavaScriptが必要であるというユーザーへの単なる警告であっても、アプリは一部のコンテンツを表示する必要があります。97.57%のページには、JavaScriptが無効[591]の空白ページ以上のものが表示されます。 ホームページのみを調査していることを考えると、3.43%のサイトがこの監査に失敗したことはおそらくもっと驚くべきことです！

サービスワーカーのイベント

サービスワーカーでは、いくつかのイベントをリッスンできます。

585. *https://web.dev/content-width/*
586. *https://web.dev/i18n/ja/maskable-icon-audit/*
587. *https://web.dev/offline-start-url/*
588. *https://web.dev/i18n/ja/service-worker/*
589. *https://web.dev/splash-screen/*
590. *https://web.dev/i18n/ja/themed-omnibox/*
591. *https://web.dev/without-javascript/*

1. `install` は、サービスワーカーのインストール時に発生します。
2. `activate` は、サービスワーカーの起動時に発生します。
3. `fetch` は、リソースがfetchされるたびに発生します。
4. `push` は、プッシュ通知が届いたときに発生します。
5. `notificationclick` は、通知がクリックされたときに発生します。
6. `notificationclose` は、通知を閉じているときに発生します。
7. `message` は、postMessage()で送信したメッセージが到着したときに発生します。
8. `sync` は、バックグラウンドの同期イベントが発生したときに発生します。

今回のデータセットでは、これらのイベントのうち、どのイベントがサービスワーカーに聴かれているかを調べました。

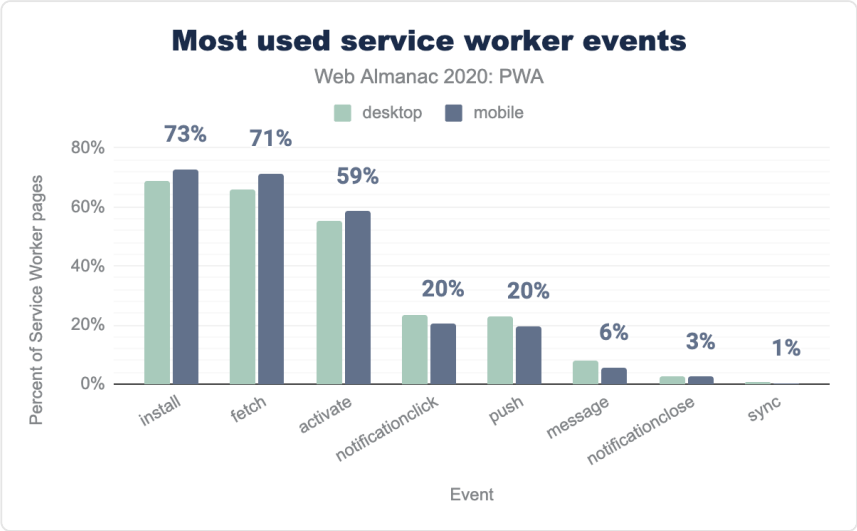


図14.3. もっとも利用されるサービスワーカーのイベント

モバイルとデスクトップの結果は非常に似ており、`install`、`fetch`、`activate` がもっとも人気のある3つのイベントで、`message`、`notification click`、`push`、`sync` と続きます。この結果を解釈すると、サービスワーカーが可能にするオフラインのユースケースは、アプリ開発者にとってプッシュ通知をはるかにしのぐ、もっとも魅力的な機能であると言えます。バックグラウンド同期は、利用可能な機能が限られており、一般的なユースケースではないため、現時点では重要な役割を果たしていません。

ウェブアプリのマニフェスト

Webアプリマニフェストは、JSONベースのファイルで、Webアプリケーションに関連するメタデータを開発者が一元的に配置できるようにするものです。マニフェストは、アイコンや向き、テーマカラーなど、デスクトップやモバイルでのアプリケーションの動作を規定します。

PWAを実りあるものにするためには、マニフェストとサービスワーカーが必要です。興味深いのは、サービスワーカーよりもマニフェストの方が多く使用されていることです。これは、WordPress、Drupal、JoomlaなどのCMSがデフォルトでマニフェストを備えていることが大きな理由です。

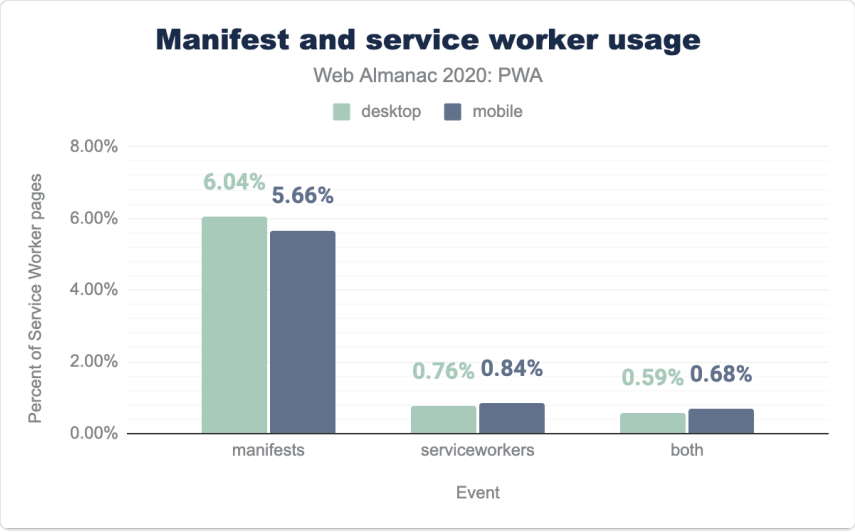


図14.4. マニフェストとサービスワーカーの利用

Webアプリマニフェストは、サービスワーカーの使用とは無関係に存在することができるため、Webアプリマニフェストがあるからといって、そのサイトがプログレッシブWebアプリであるとは限りません。しかし、本章ではPWAに関心があるため、サービスワーカーも存在するサイトのマニフェストのみを調査しました。昨年のPWAの章[592]で行った、マニフェスト全体の使用状況を調べるアプローチとは異なるため、今年の結果には若干の違いがあるかもしれません。

592. *https://almanac.httparchive.org/ja/2019/pwa#web-app-manifests*

マニフェストのプロパティ

ウェブマニフェストは、アプリケーションのメタプロパティを規定するものです。ここでは、Web App Manifest仕様で定義されているさまざまなプロパティに加え、標準ではない独自のプロパティについても検討しました。

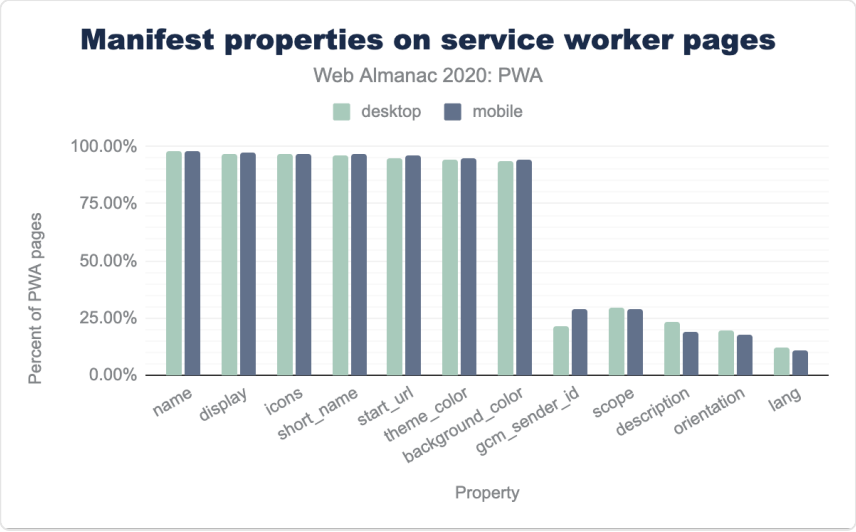


図14.5. サービスワーカーページのマニフェストプロパティ

仕様書によると、以下のプロパティが有効なプロパティです[593]。

- `background_color`
- `categories`
- `description`
- `dir`
- `display`
- `iarc_rating_id`
- `icons`
- `lang`

593. https://w3c.github.io/manifest/#webappmanifest-dictionary

- `name`
- `orientation`
- `prefer_related_applications`
- `related_applications`
- `scope`
- `screenshots`
- `short_name`
- `shortcuts`
- `start_url`
- `theme_color`

モバイルとデスクトップの統計の違いはほとんどありませんでした。

Google Cloud Messaging(GCM)サービスで使用されている`gcm_sender_id`という独自のプロパティが頻繁に見つかりました。他にも、以下のような興味深い属性がありました。`browser_action`, `DO_NOT_CHANGE_GCM_SENDER_ID`（これは基本的にコメントで、JSONではコメントが許されていないために使用されています）、`scope`、`public path`、`cacheDigest`。

しかし、どちらのプラットフォームでも、ブラウザでは解釈されず、潜在的に有用なメタデータを含むプロパティのロングテールがあります。

また、ミスタイプされたプロパティも多数発見されました。私たちが気に入っているのは、`theme-color`, `Theme_color`, `theme-color`, `Theme_color`, `orientation`のバリエーションです。

トップマニフェストの表示値

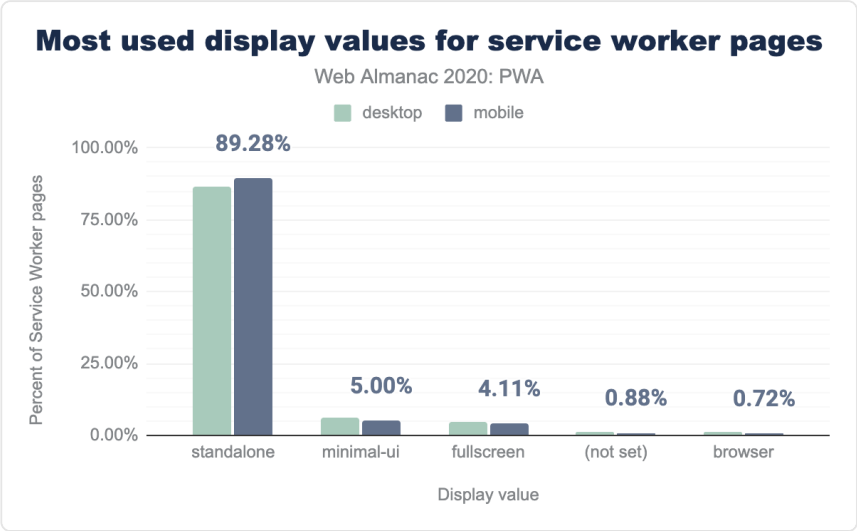


図14.6. サービスワーカーのページでもっとも使用される `display` の値です。

もっとも一般的な5つの `display` 値のうち、デスクトップページの86.73%、モバイルページの89.28%が `standalone` を使用しており、上位を占めています。このモードでは、ネイティブアプリのような操作感が得られるので、驚くことではありません。次に多かったのは `minimal-ui` で、デスクトップで6.30%、モバイルでは5.00%が採用していました。これは `standalone` と似ていますが、一部のブラウザのUIが維持されるという点が異なります。

トップマニフェストのカテゴリー

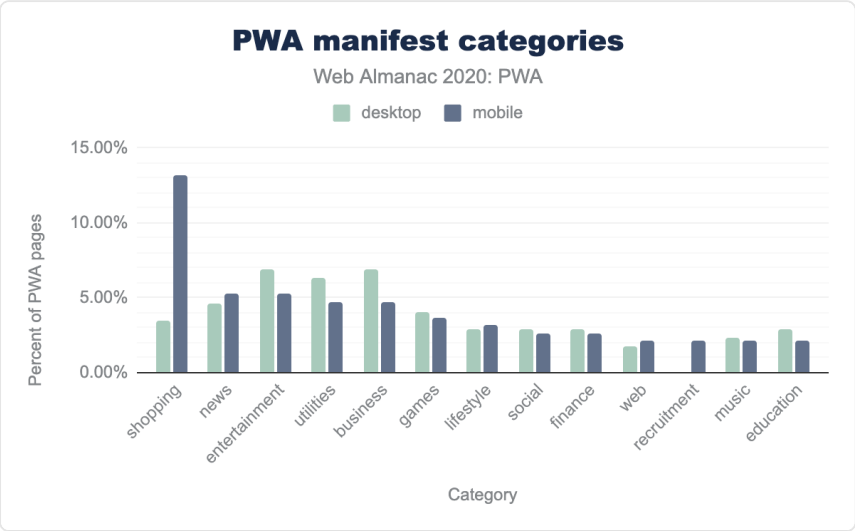


図14.7. PWA マニフェストのカテゴリー

これは、PWAが電子商取引アプリケーションであることから予想できることです。次いで`ニュース`が5.26%となっている。

ネイティブにこだわるマニフェスト

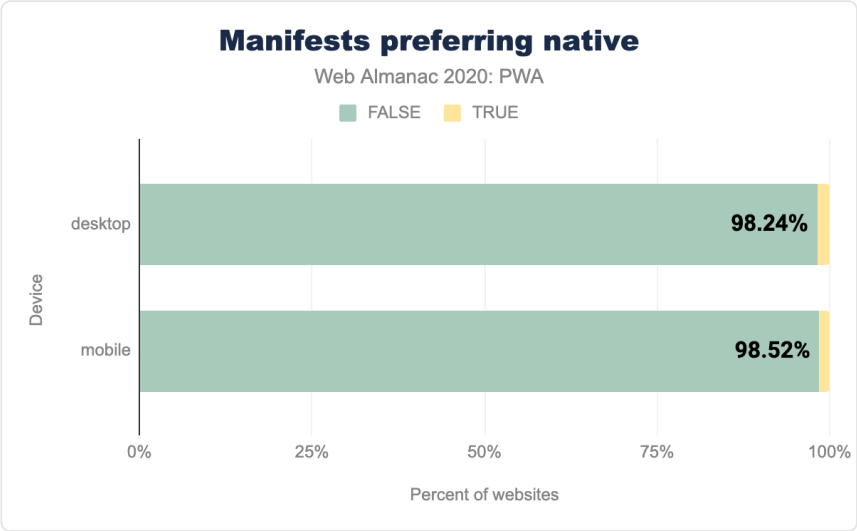


図14.8. マニフェストは、ネイティブを好む。

デスクトップPWAサイトの98.24%、モバイルPWAサイトの98.52%が、ネイティブアプリを優先せず、存在するウェブバージョンを使用するように、`preferred_related_applications`マニフェストプロパティを設定しています。このプロパティが`true`に設定されているごく一部のサイトでは、マニフェストを持つだけで実際にはまだ完全なPWAではない多くのウェブアプリケーションが存在することを示しています。

トップマニフェストのアイコンサイズ

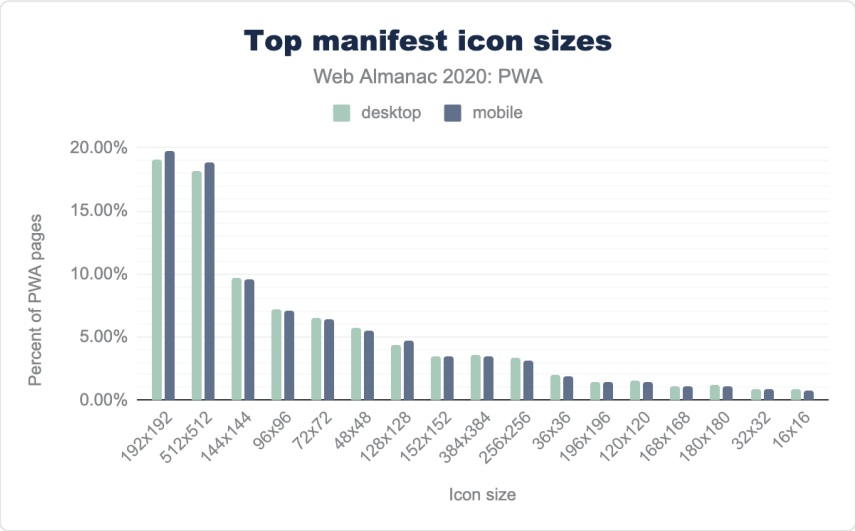


図14.9. トップマニフェストのアイコンサイズ

Lighthouseでは、最低でも192x192ピクセルのアイコンが必要ですが、一般的なファビコン生成ツールでは、他のサイズも多数作成されています。各デバイスで推奨されているアイコンサイズを使用することが望ましいので、さまざまなサイズのアイコンが広く使われていることは心強いことです。

トップマニフェストの方向性

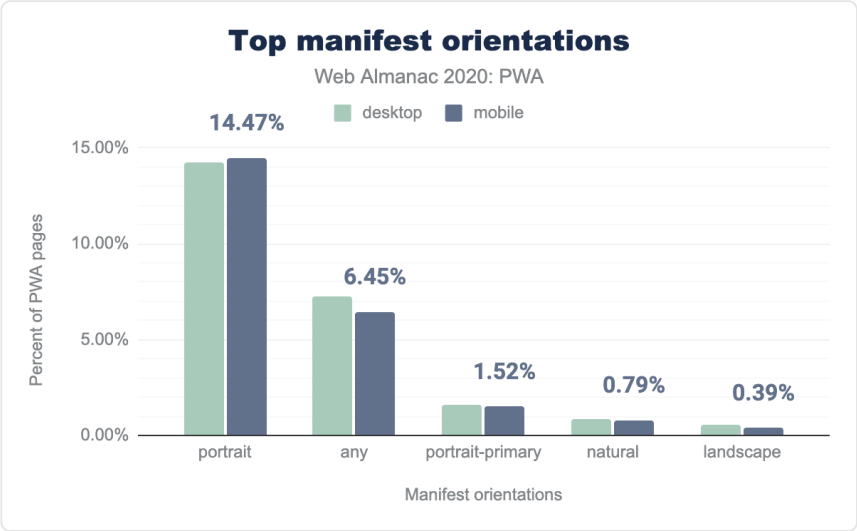


図14.10. Top manifest orientations.

orientationプロパティの有効な値は、Screen Orientation APIspecification[594]で定義されています。現在のところ、以下の通りです。

- `any`
- `natural`
- `landscape`
- `portrait`
- `portrait-primary`
- `portrait-secondary`
- `landscape-primary`
- `landscape-secondary`

その中で、`portrait`、`any`、`portrait-primary`のプロパティが優先されることに気づきました。

594. https://www.w3.org/TR/screen-orientation/

サービスワーカーライブラリ

サービスワーカーがライブラリを依存関係として使用するケースは、外部依存関係であれ、アプリケーションの内部依存関係であれ、多くあります。これらは通常、`importScripts()` APIによってサービスワーカーに取り込まれます。このセクションでは、そのようなライブラリの統計情報を見ていきます。

人気のインポートスクリプト

WorkerGlobalScope インタフェース[595]のimportScripts() API[596]は、1つまたは複数のスクリプトをワーカーのスコープに同期的にインポートします。同じことが、外部の依存関係をサービスワーカーにインポートするためにも使われます。

スクリプト	デスクトップ	モバイル
`importScripts()` 使用率	29.60%	23.76%
`Workbox`	17.70%	15.25%
`sw_toolbox`	13.92%	12.84%
`firebase`	3.40%	3.09%
`OneSignalSDK`	4.23%	2.76%
`najva`	1.89%	1.52%
`upush`	1.45%	1.23%
`cache_polyfill`	0.70%	0.72%
`analytics_helper`	0.34%	0.39%
Other Library	0.27%	0.15%
No Library	58.81%	64.44%

図14.11. PWAライブラリーの利用状況

デスクトップPWAサイトの約30%、モバイルPWAサイトの約25%が`importScripts()`を使用しており、そのうち`workbox`、`sw_toolbox`、`firebase`がそれぞれ1位から3位を占めています。

595. https://developer.mozilla.org/docs/Web/API/WorkerGlobalScope
596. https://developer.mozilla.org/docs/Web/API/WorkerGlobalScope/importScripts

ワークボックスの使い方

数多くあるライブラリの中で、もっとも多く利用されたのはWorkboxで、モバイルとデスクトップのPWAサイトでそれぞれ12.86%と15.29%の採用率を記録しました。

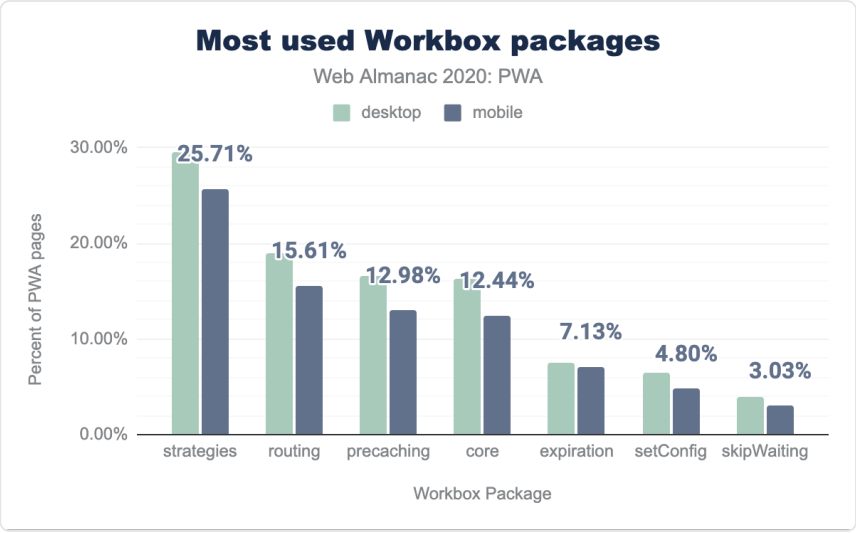


図14.12. もっとも使用されているWorkboxのパッケージ。

Workboxが提供する多くのメソッドのうち、`strategies`はデスクトップで29.53%、モバイルでは25.71%が使用し、`routing`が18.91%、15.61%と続き、最後に`precaching`がデスクトップとモバイルでそれぞれ16.54%、12.98%となっています。

これは、開発者にとってもっとも複雑な要件の1つである戦略APIが、自分でコーディングするかWorkboxのようなライブラリに頼るかを決める際に、非常に重要な役割を果たしていることを示しています。

結論

本章の統計によると、PWAは、とくにモバイルにおいて、パフォーマンスやキャッシングのコントロールがしやすいという利点があるため、今もなお採用が増え続けています。このような利点と、増え続けるcapabilitiesにより、私たちにはまだ大きな成長の可能性があります。2021年には、さらなる進化を期待しています。

ますます多くのブラウザやプラットフォームが、PWAを支える技術をサポートしています。今年は、EdgeがWeb App Manifestをサポートするようになりました。ユースケースやター

ゲット市場にもよりますが、ユーザーの大半（96%近く）がPWAをサポートしていることに気づくかもしれません。これは素晴らしい改善です。どのような場合でも、サービスワーカーやWeb App Manifestなどの技術を進歩的な機能強化としてアプローチすることが重要です。これらの技術を利用して、優れたユーザーエクスペリエンスを提供できます。上記の統計を見ると、今年もPWAの成長が期待できそうですね。

著者

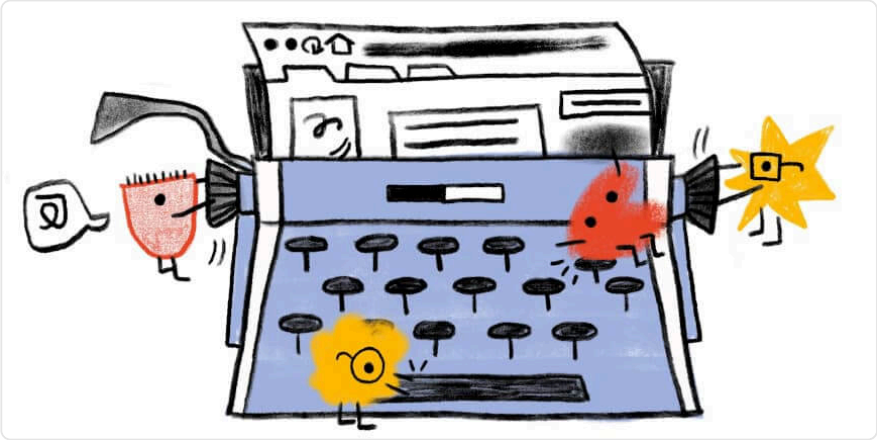
Hemanth HM

@gnumanth　hemanth　http://h3manth.com

Hemanth HM[597]は、コンピュータポリグロット、FOSS哲学者、Webと決済ドメインのためのGDE、DuckDuckGoコミュニティメンバー、TC39代表者、Google Launchpad Acceleratorメンター。WEB & CLIが大好き。TC39er.us[598]ポッドキャストを主催。

597. *https://h3manth.com*
598. *https://TC39er.us*

部 III 章 15
CMS



***Alex Denning** によって書かれた。*
Jonathan Wold、 Renee Johnson と Alberto Medina によってレビュ。
Greg Brimble と Rick Viscomi による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

コンテンツマネジメントシステム（CMS）とは、個人や組織がコンテンツを作成・管理・公開するためのシステムを指す。とくにWebコンテンツ用のCMSは、インターネット上で消費・体験されるコンテンツを作成・管理・公開することを目的としたシステムです。

各CMSには、幅広いコンテンツ管理機能の一部が実装されており、ユーザーがコンテンツを使って簡単かつ効果的にWebサイトを構築できるような仕組みになっています。コンテンツは多くの場合、一種のデータベースに格納されており、ユーザーはコンテンツ戦略上必要な場所でコンテンツを再利用できます。また、ユーザーが必要に応じてコンテンツをアップロードしたり、管理したりすることを容易にするための管理機能も備えている。

サイト構築をサポートするCMSにはすぐに使えるテンプレートを提供し、そこにユーザーのコンテンツを追加していくものもあれば、サイト構造の設計・構築にユーザーの関与が必要なものもありその種類や範囲は千差万別です。

CMSを考える際には、ウェブ上でコンテンツを公開するためのプラットフォームを提供するシステムの実現性を担うすべてのコンポーネントを考慮する必要があります。これらのコンポーネントは、CMSプラットフォームを取り巻くエコシステムを形成しており、ホスティングプロバイダー、エクステンション開発者、開発会社、サイトビルダーなどが含まれます。このように、CMSについて語るとき、通常はプラットフォーム自体とその周辺のエコシステムの両方を指しています。

ウェブの現在と未来におけるCMS空間とその役割を理解するためには、分析すべき興味深く重要な側面や答えるべき質問がたくさんあります。私たちはCMSプラットフォームの広大さと複雑さを認識し、好奇心と、この分野の主要なプレイヤーに関する深い専門知識を提供します。

本章では、CMSエコシステムの現状、ウェブ上でコンテンツを消費・体験する方法に関するユーザーの認識を形成する上でCMSが果たす役割、そして環境への影響について理解を深めることを目的としています。本章では、CMSの現状と、CMSによって生成されたWebページの特徴を議論することを目的としています。

Web Almanacの第2版は、昨年の成果を基にしています。2020年の結果と2019年の結果を比較することで、傾向を把握できるようになりました。それでは早速、分析していきましょう。

なぜ2020年にCMSを使うのか？

2020年、人や組織がCMSを利用するのは、多くの場合、CMSが自分のニーズに合ったウェブサイトを作るための近道となるからです。後述するように、CMSには汎用CMSと特化型CMSがある。一般的なCMSはアドオンによる拡張性が高く、専門的なCMSは特定の業界のニーズや機能に特化していることが多いです。

どのCMSが使われているにせよ、ユーザーや組織の問題を解決するために使われているのです。それぞれのCMSが選ばれる理由を探るのはここでは無理ですが、後ほど、もっともポピュラーなCMSであるWordPressを選ぶ割合が高い理由をご紹介します。

CMSの採用

この作品を通しての分析は、デスクトップとモバイルのウェブサイトを対象としています。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、中にはデスクトップまたはモバイルデバイスでしかアクセスされないURLもあります。このため、データにわずかな差異を発生させ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見ています。

Webページの42%以上がCMSプラットフォームを利用しており、2019年から5%以上増加し

ています。その内訳は、デスクトップでは42.18%で2019年の40.01%から、モバイルでは42.27%で2019年の39.61%から増加しています。

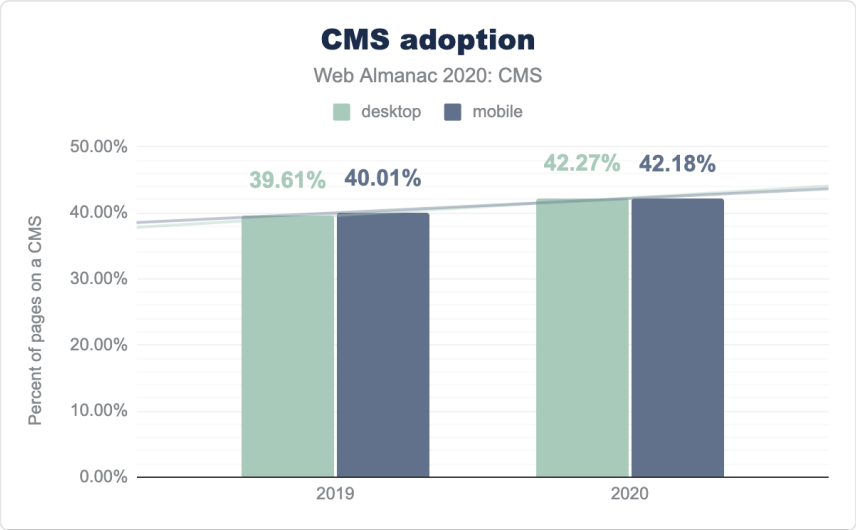


図15.1. CMSの採用傾向。

年	デスクトップ	モバイル
2019	39.61%	40.01%
2020	42.27%	42.18%
% Change	6.71%	5.43%

図15.2. CMSの採用統計。

CMSプラットフォームを採用したデスクトップのウェブページの増加率は、昨年比で5.43%となりました。モバイルでは約4分の1の6.71%となっています。

昨年[599]と同様に、W3Techs[600]のようなCMSプラットフォームの市場シェアを追跡するための他のデータセットでも異なる結果が見られます。W3Techsの報告によると、ウェブページの60.6%がCMSで作成されており、1年前の56.4%から増加しています。これは6.4%の増加で、私たちの調査結果とほぼ一致しています。

我々の分析とW3Techsの分析の乖離は、調査方法の違いで説明できます。私たちのものについては、方法論のページに詳しく書かれています。

599. https://almanac.httparchive.org/ja/2019/cms#cms-adoption
600. https://w3techs.com/technologies/history_overview/content_management

私たちの調査では、222の個別のCMSが確認され、これらは1つのCMSに1つのインストールから数百万までの範囲でした。

その中には、オープンソースのもの（WordPress、Joomlaなど）と、プロプライエタリなもの（Wix、Squarespaceなど）があります。後述するように、採用シェアの上位3つのCMSはすべてオープンソースですが、プロプライエタリなプラットフォームは今年、採用シェアを大きく伸ばしています。CMSプラットフォームの中には、「無料」のホスティングやセルフホスティングのプランで利用できるものもありますし、企業レベルでも上位のプランで利用できるオプションもあります。

CMSスペースは全体として、CMSエコシステムの複雑な連合宇宙であり、すべてが分離していると同時に絡み合っています。我々の調査によると、CMSの重要性は増すばかりです。CMSの採用が最低でも5%増加したということは、COVID-19が巨大な不確実性を生み出した年に、強固なCMSプラットフォームが安定性をもたらしたことを示しています。昨年もお話しましたが、CMSプラットフォームは私たちがエバーグリーンで健康的で活気のあるウェブを目指す上で、重要な役割を果たします。このことは昨年よりもさらに真実味を帯びてきており、今後もその傾向が続くものと思われます。

上位の**CMS**

今回の分析では、222のCMSがカウントされました。この数は非常に多いのですが、そのうち204社（92%）は採用率が0.01%以下です。その結果、採用率が0.1～1%のCMSは13社、1～2%のCMSは4社、それ以上のCMSは1社となりました。

2%以上のシェアを持つCMSはWordPressで、31%の利用シェアを持っています。これは、次に人気のあるCMSであるJoomlaの15倍以上のシェアです。

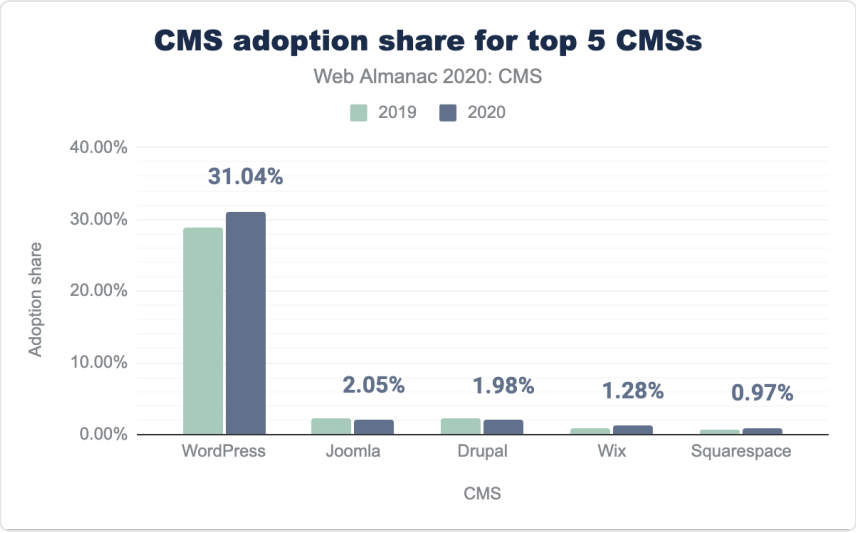


図15.3. 上位5つのCMSの導入シェア

JoomlaとDrupalは、それぞれ8%と10%の採用シェアを失いましたが、WixとSquarespaceはそれぞれ41%と28%の採用シェアを獲得しました。WordPressはこの1年間でさらに7%の採用シェアを獲得しており、これは次に人気のあるCMSであるJoomlaのシェア合計よりも絶対的な増加率が大きい。

この数字は、デスクトップとモバイルに分けてもほぼ同じです。

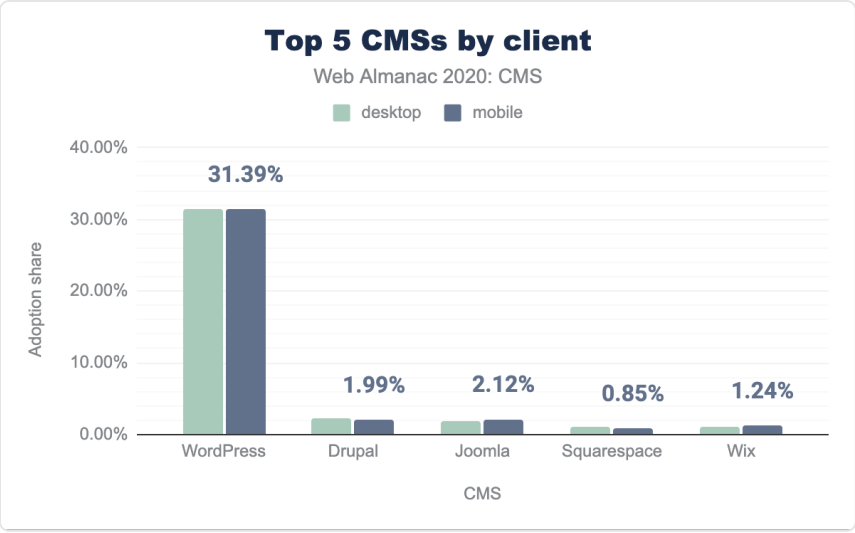


図15.4. クライアント別のCMSトップ5。

WordPressの場合は非常に似たような数字になっていますが、他のCMSの場合はその差が大きくなっています。DrupalとSquarespaceでは、モバイルよりもデスクトップの方がそれぞれ16.7%と26.3%多く、JoomlaとWixでは、デスクトップよりもモバイルの方が7.5%と15.2%多くなっています。

普及率0.1〜1%のカテゴリーでは、より多くの動きが見られます。これらのカテゴリーでは、50,000サイトまでのCMSが採用されています。

CMS	2019	2020	変更%
WordPress	28.91%	31.04%	7%
Joomla	2.24%	2.05%	-8%
Drupal	2.21%	1.98%	-10%
Wix	0.91%	1.28%	41%
Squarespace	0.76%	0.97%	28%
1C-Bitrix	0.55%	0.61%	10%
TYPO3 CMS	0.53%	0.52%	-2%
Weebly	0.39%	0.33%	-15%
Jimdo	0.28%	0.24%	-16%
Adobe Experience Manager	0.27%	0.23%	-14%
Duda		0.22%	
GoDaddy Website Builder		0.18%	
DNN	0.20%	0.16%	-19%
DataLife Engine	0.19%	0.16%	-12%
Tilda	0.08%	0.16%	100%
Liferay	0.12%	0.11%	-10%
Microsoft SharePoint	0.15%	0.11%	-25%
Kentico CMS	0.00%	0.11%	10819%
Contao	0.09%	0.09%	0%
Craft CMS	0.08%	0.09%	5%
MyWebsite		0.09%	
Concrete5	0.10%	0.09%	-12%

図15.5. 小型CMSの相対的な採用率（0.1%〜1%の採用シェア）

ここでは3つの新規参入企業が見られます。Duda、GoDaddy Website Builder、MyWebsiteです。また、TildaとKentico CMSの2つは、ここ1年で採用率が100%以上に変化していま

す。この「ロングテール」と呼ばれるCMSは、オープンソースとプロプライエタリなプラットフォームが混在しており、消費者向けのものから業界に特化したものまであらゆるものが含まれている。CMSプラットフォーム全体の強みは、考えられるあらゆるタイプのWebサイトを動かす専用ソフトウェアを入手できることです。

他のCMSと比較したCMS導入シェアを見ると（CMSを導入していないウェブサイトを除く）、WordPressの優位性が明らかになります。CMSを導入しているWebサイトの導入シェアは74.2%。この数字を相対化すると、Joomla、Drupal、Wix、Squarespaceの採用率が高くなります。それぞれ4.9%、4.7%、3.1%、2.3%となっています。

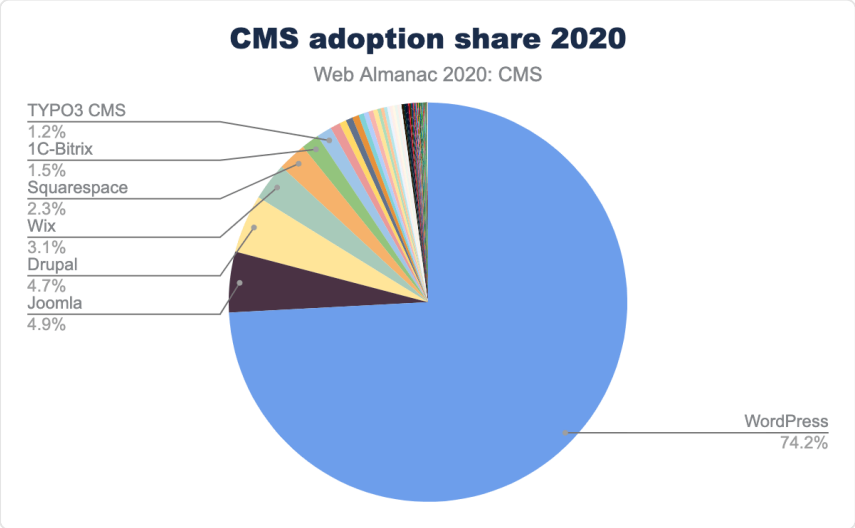


図15.6. CMSの採用シェア2020。

WordPressの使い方

この分野ではWordPressが圧倒的な存在感を放っているので、さらなる議論が必要です。

WordPressは、オープンソースプロジェクト[601]で「出版を民主化する」ことをミッションとするです。このCMSは無料です。これが普及率の重要な要因だと思われますが、次に普及している2つのCMS-JoomlaとDrupalも無料です。WordPressのコミュニティ、コントリビューター、ビジネスエコシステムが大きな差別化要因になっていると思われる。

「コア」なWordPressコミュニティは、CMSを維持し、カスタムサービスや製品（テーマやプラグイン）による追加機能の要件を満たしています。このコミュニティの影響力は非常に

601. https://wordpress.org/about/

大きく比較的少数の人々がCMS自体を維持し、追加機能を提供することでWordPressは十分に強力で柔軟性があり、ほとんどのタイプのWebサイトへ対応できるようになっています。この柔軟性は、市場シェアを説明する上で重要です。

この柔軟性のおかげで、WordPressは開発者やサイトの「構築者」または「実装者」にとって、参入障壁が低いものとなっています。柔軟な拡張機能によってサイト構築が容易になり、それによってますます多くのユーザーがWordPressでより強力なサイトを構築するという好循環が生まれています。ユーザーが増えることで、開発者はより良い拡張機能を作ることができるようになり、このサイクルがさらに進むのです。

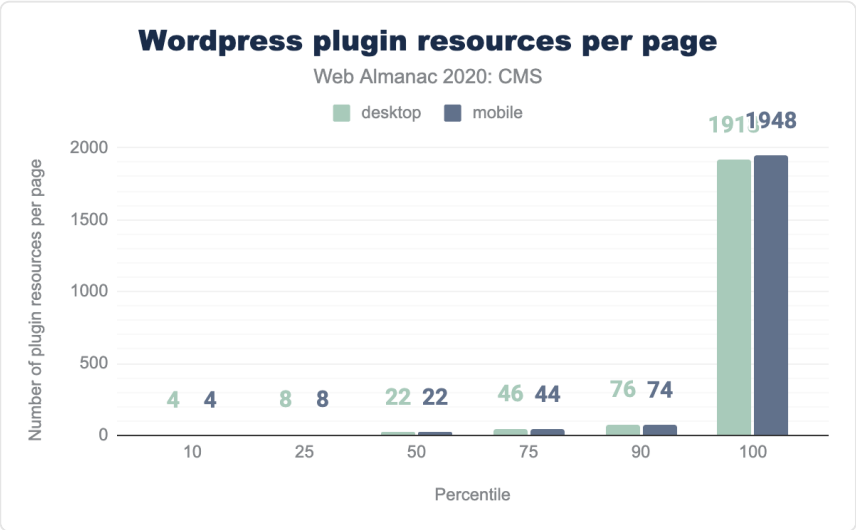


図15.7. ページごとのWordPressプラグインのリソース。

私たちは、WordPressサイトがこれらの拡張機能をどのように使用しているかを調査しました。これらの拡張機能は、通常WordPressのプラグインです。WordPressサイトの中央値（デスクトップおよびモバイル）では1ページあたり22個のプラグインリソースがロードされており、90パーセンタイルのサイトでは、デスクトップで76個、モバイルでは74個のリソースがロードされていました。100パーセンタイルのサイトでは、デスクトップとモバイルでそれぞれ1918と1948のリソースをロードします。これを他のCMSと比較することはできませんが、WordPressの拡張エコシステムが普及率の高さに大きく貢献していることは間違いなさそうです。

2019年から2020年にかけてのWordPressの採用シェアの伸びは7.40%で、CMS全体の採用数の増加を上回っています。これは、WordPressが「平均的な」CMSを大きく超えた魅力を持っていることを示唆しています。

2020年は「COVID-19」の影響が出ている。このことが市場シェアの増加を説明しているの

かもしれません。逸話としては、多くの実在する企業が永久的または一時的に閉鎖されたことで、一般的にウェブサイトへの需要が高まり、最大のCMSであるWordPressがその恩恵を受けていることを示唆しています。この影響を完全に把握するには、今後数年間のさらなる調査が必要です。

CMSの導入シェアが明らかになったところで、次はユーザー体験に注目してみましょう。

CMSのユーザー体験

CMSは優れたユーザー体験を提供しなければなりません。ウェブの多くがページの提供をCMSに依存している中で、ユーザー体験が良好であることを保証するのは、プラットフォームレベルでのCMSの責任です。私たちの目的は、CMSを搭載したウェブサイトを使用する際の、実際のユーザー体験に光を当てることです。

そのために、ユーザーが感じるパフォーマンスの指標を分析することにしました。この指標は、3つのCore Web Vitals指標と、SEOとAccessibilityカテゴリのLighthouseスコアに反映されています。

Chromeユーザー体験レポート

このセクションでは、Chrome User Experience Reportで提供されている3つの重要な要素を見てみましょう。これらの要素は、ユーザーがCMSを搭載したウェブページをどのように体験しているかについての理解を深めるのに役立ちます。

- 最大のコンテンツフルペイント (LCP)
- 最初の入力までの遅延 (FID)
- 累積レイアウト変更 (CLS)

これらのメトリクスは、優れたウェブ・ユーザー・体験を示す中核的な要素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詳しく説明しますが、ここではCMSに特化してこれらのメトリクスを見ていきたいと思います。それぞれの項目を順に見ていきましょう。

最大のコンテンツフルペイント

最大のコンテンツフルペイント（LCP）はページのメインコンテンツが読み込まれたと思われる時点を測定し、その結果、そのページがユーザーにとって有用であるかどうかを判断します。これは、ビューポート内に表示される最大の画像や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実現しています。

これは、ページが読み込まれてからテキストや画像などのコンテンツが最初に表示されるまでの時間を測定するコンテンツの初回ペイント（FCP）とは異なります。LCPは、ページのメインコンテンツが読み込まれたとき、測定するのに適したプロキシとされています。

「良い」LCPは2.5秒以下とされています。上位5つのCMSのうち、平均的なウェブサイトはLCPが良くありません。デスクトップではDrupalのみが50%以上のスコアを獲得しています。デスクトップとモバイルのスコアには大きな違いがあります。WordPressはデスクトップで33%、モバイルでは25%とほぼ互角ですが、Squarespaceはデスクトップで37%、モバイルでは12%しかありません。

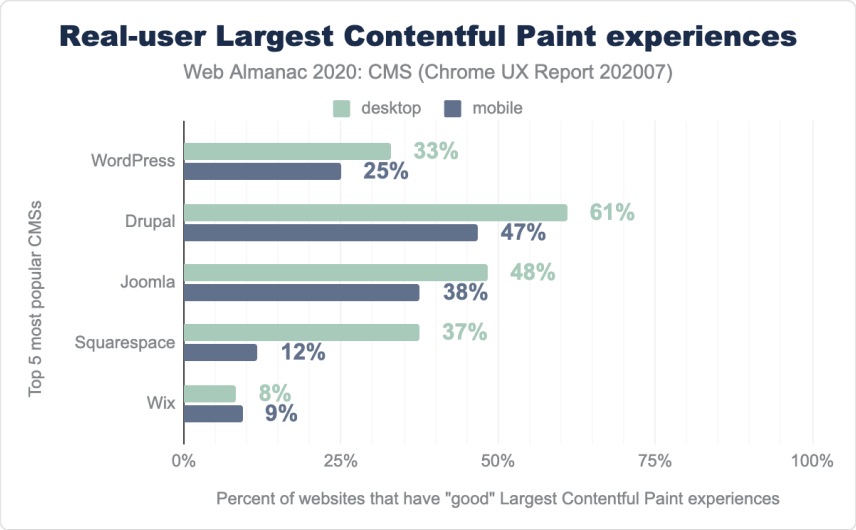


図15.8. リアルユーザーによる最大のコンテンツフル・ペイント体験。

ここでは、CMSのパフォーマンスがもっと良くなることを期待していますが、この結果からいくつかのポジティブなものが得られました。まずDrupalウェブサイトの61%が良好なLCPを持っているという事実は、Chrome UX Report[602]によると、良好なLCPを持つウェブサイトの世界的な分布が48%であることよりもはるかに優れているという点で、とくに注目に値します。またWordPressのウェブサイトの3〜4つに1つが良いLCPを持っているというのも、WordPressのウェブサイトの数が非常に多いことを考えると、ちょっとした驚きです。Wixには追いつくべき点がありますが、Wixのエンジニアがパフォーマンス問題の修正に積極的に[603]取り組んでいることは心強いので、今後も注目していきたいところです。

602. https://x.com/ChromeUXReport/status/1293306510509039616
603. https://x.com/DanShappir/status/1308043752712343552

最初の入力までの遅延

初回入力遅延（FID）とはユーザーがはじめてサイトにアクセスしたとき（リンクをクリックしたり、ボタンをタップしたり、JavaScriptを使用したカスタムコントロールを使用したときなど）からブラウザがそのインタラクションへ実際に応答するまでの時間を計測したものです。ユーザーの視点から見た「速い」FIDとは、サイト上での行動からのフィードバックがすぐに得られることであり、体験が停滞することではありません。遅延は苦痛であり、ユーザーがサイトと対話したときに、サイトの他の側面の読み込みによる干渉と相関している可能性があります。

平均的なCMSウェブサイトでは、デスクトップでのFIDは非常に速く、Wixだけが100%を下回っておりモバイルではまちまちです。ほとんどのCMSでは、平均的なサイトのモバイルFIDはデスクトップのスコアの妥当な範囲内に収まっています。Wixの場合、モバイルで良好なFIDを持つウェブサイトの数は、デスクトップの合計数の半分近くになります。

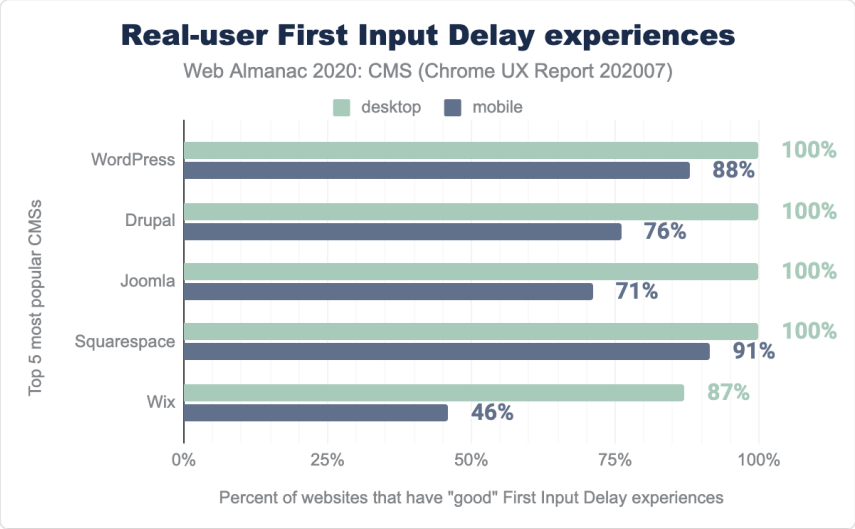


図15.9. 実際のユーザーによる初回入力遅延の体験

LCPスコアとは対照的に、FIDスコアは概ね良好です。提案されているようにモバイル接続の品質に加えて、CMS上の個々のページの重みや、デスクトップに比べてモバイル機器の性能が低いこと、ここで見られるFIDに影響するパフォーマンスのギャップに関与している可能性があります。

デスクトップ版とモバイル版のWebサイトに投入されるリソースには、わずかな差しかありません。昨年、モバイル向けの最適化が必要であると指摘しました。平均スコアはデスクトップとモバイルで上昇していますが、モバイルではさらなる注意が必要です。

累積レイアウト変更

累積レイアウト変更（CLS）は、ユーザが最初に入力してから500ms経過した時点でのWebページ上のコンテンツの不安定さを測定するものです。これはとくにモバイルにおいて重要で、ユーザーは検索バーなどのアクションを起こしたい場所をタップしますが、追加の画像や広告などが読み込まれるとその場所は移動してしまいます。

0.1以下は「良い」、0.25以上は「悪い」、その間は「改善が必要」と測定されます。

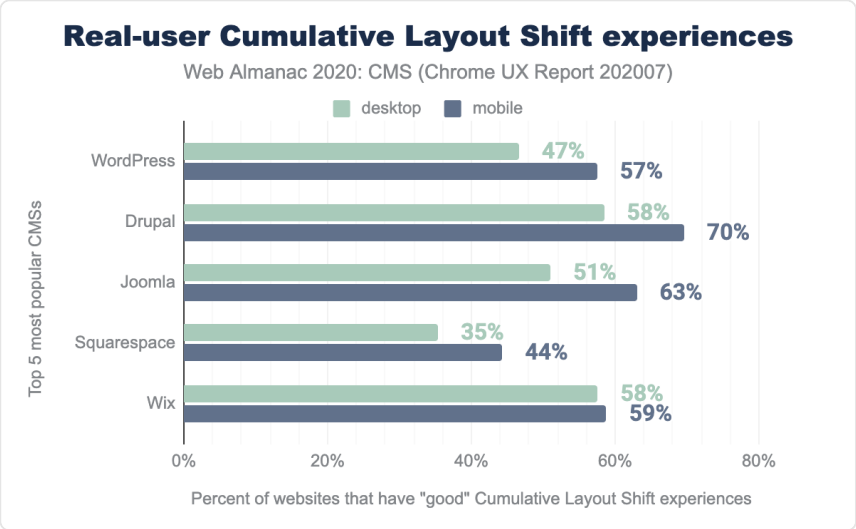


図15.10. 実際のユーザーによる累積レイアウト変更の体験。

上位5社のCMSは、ここを改善できる可能性があります。上位5社のCMSで読み込まれたWebページのうち、CLSエクスペリエンスが「良い」とされたのはわずか50%で、この数字はモバイルでは59%にまで上昇しています。すべてのCMSで、デスクトップの平均スコアは59%、モバイルの平均スコアは67%です。これは、すべてのCMSに課題があることを示していますが、とくにトップ5のCMSは改善が必要です。

Lighthouseスコア

Lighthouseは、開発者がウェブサイトの品質を評価・改善するために設計された、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点は、パフォーマンス、アクセシビリティ、SEO、プログレッシブWebアプリなどの観点からWebサイトの状態を評価する一連の監査を提供していることです。今年の章では、2つの特定の監査カテゴリーに注目しました。SEOとアクセシビリティです。

SEO

検索エンジン最適化（SEO）とは、ウェブサイトのコンテンツが検索エンジンで見つかりやすくなるようにウェブサイトを最適化することです。これについてはSEOの章で詳しく説明していますが、その一環として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検索結果に、適切に表示されるようサイトをコード化することがあります。カスタムメイドのウェブサイトに比べて、CMSには優れたSEO機能が期待されますが、Lighthouseのこのカテゴリーのスコアは高い評価を示しています。

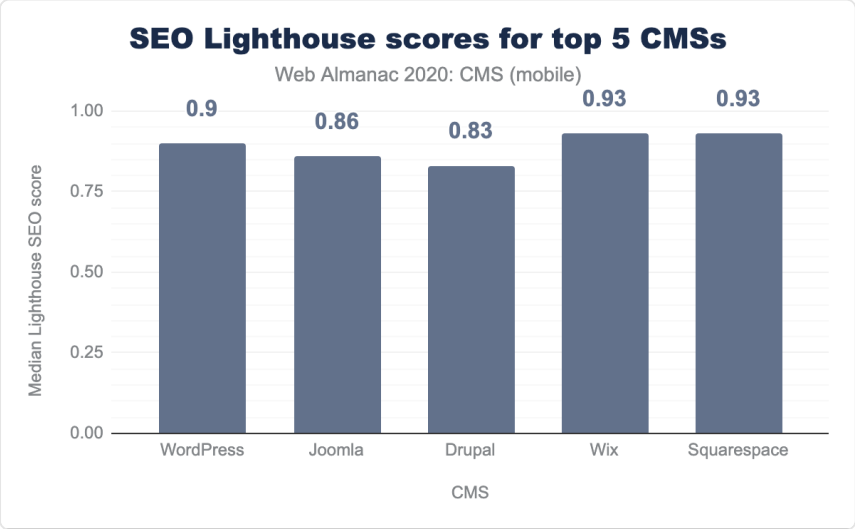


図15.11. SEO LighthouseスコアのCMSトップ5。

ここでは、上位5つのCMSのすべてが高いスコアを示しており、中央値は0.83以上、中には0.93に達するものもあります。SEOはウェブサイトの所有者がCMSの機能を利用するかどうかにかかっていますが、CMSでこれらのオプションを簡単に利用できるようにし適切なデフォルト設定を行うことで、そのCMSで運営されているサイトに大きなメリットをもたらすことができます。

アクセシビリティ

アクセシブルなウェブサイトとは、障害のある人が利用できるように設計・開発されたサイトのことです。Webアクセシビリティは、インターネットの接続速度が遅い場合など、障害のない人にもメリットがあります。詳しい説明はこちら[604]、およびアクセシビリティの章でご覧いただけます。

604. *https://www.w3.org/WAI/fundamentals/accessibility-intro/#what*

Lighthouseは一連のアクセシビリティ監査を提供しており、それらすべての監査の加重平均を返します（各監査の加重方法の詳細については、 スコアリングの詳細[605]を参照してください）。

各アクセシビリティ監査は合格か不合格のいずれかですが、他のLighthouse監査とは異なり、ページが部分的にアクセシビリティ監査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にスクリーン・リーダー・フレンドリーな名前が付いていて、他の要素には付いていない場合、そのページはスクリーン・リーダー・フレンドリーな名前の監査で0点を獲得します。

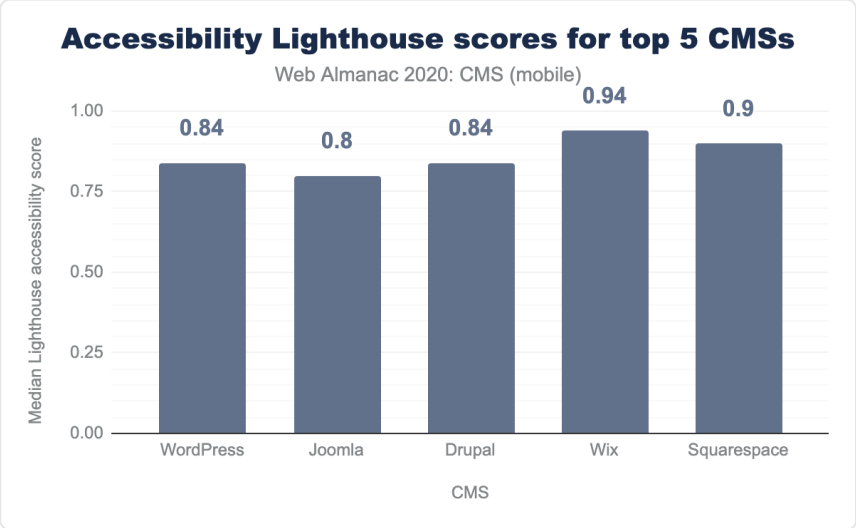


図15.12. 上位5つのCMSのアクセシビリティLighthouseスコア。

上位5つのCMSのLighthouseアクセシビリティスコアの中央値は、すべて0.80を超えています。すべてのCMSで、Lighthouseスコアの平均的な中央値は0.78で、最小値は0.44、最大値は0.98です。このように、上位5つのCMSは平均よりも優れており、いくつかのCMSは他よりも優れていることがわかります。WixとSquarespaceは、トップ5の中でもっとも高いスコアを持っています。恐らくこれらのプラットフォームは独自に開発されたものであり、作成されたサイトをより綿密に管理できるため、この点が役に立っているのでしょう。

しかし、ここでのハードルはもっと高いはずです。すべてのCMSの平均スコアが0.78であっても改善の余地は大きく、最大スコアの0.98は、アクセシビリティ対応で「最高」のCMSであっても改善の余地があることを示しています。アクセシビリティの向上は必須かつ緊急の課題です。

605. *https://web.dev/accessibility-scoring/*

環境への影響

今年は、CMSが環境に与える影響をより深く理解することを目指しました。情報・通信技術（ICT）産業は世界の炭素排出量の2%を占めており[606]、とくにデータセンターは世界の炭素排出量の0.3%を占めています。これは、ICT業界のカーボンフットプリントが、航空業界の燃料からの排出量に匹敵することを意味します。ここではCMSの役割についてのデータはありませんが、当社の調査によるとウェブサイトの42%がCMSを使用していることから、CMSがウェブサイトの効率化と環境への影響に重要な役割を果たしていることは明らかです。

私たちの調査では、CMSの平均的なページの重さ（KB）を調べ、carbonapi[607]のロジックを使ってこれをCO2排出量にマッピングしました。その結果、デスクトップとモバイルに分けて以下のような結果が得られました。

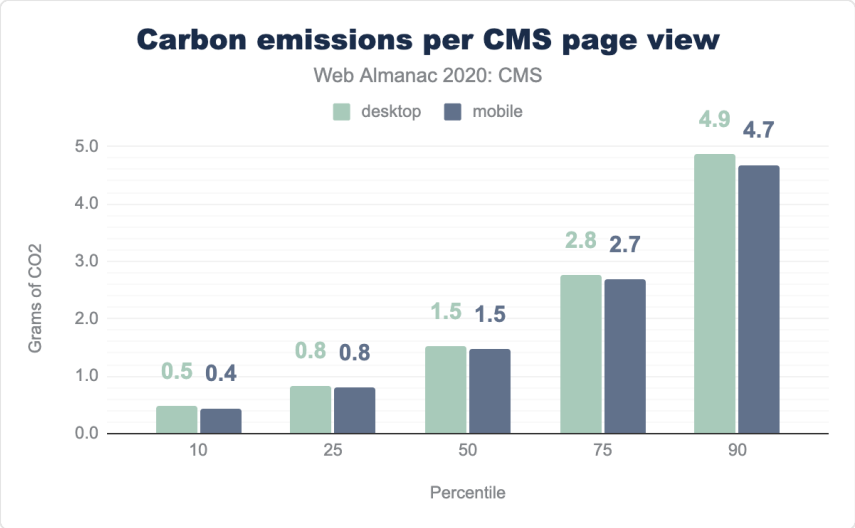


図15.13. CMSのページビューあたりの炭素排出量。

その結果、CMSのページロードの中央値は2.41MBの転送となり、1.5gのCO2を排出してることがわかりました。これは、デスクトップとモバイルで同じでした。もっとも効率的なパーセンタイルのCMSウェブページでは少なくとも3分の1のCO2が削減されますが、もっとも効率的でないパーセンタイルのCMSウェブページでは、逆に中央値よりも3分の1以上効率が悪くなります。もっとも効率的なページのパーセンタイルは、もっとも効率的でないパーセンタイルの約10倍の効率性があります。

606. https://www.nature.com/articles/d41586-018-06610-y
607. https://gitlab.com/wholegrain/carbon-api-2-0/-/blob/master/includes/carbonapi.php#L342

CMSはあらゆるタイプのWebサイトに対応しているため、このような違いがあるのは当然のことです。しかし、CMSはプラットフォームレベルで、作成したウェブサイトの効率性に影響を与えることができます。

ここで重要なのが、ページの重さです。平均的なデスクトップCMSのWebページは、2.4MBのHTML、CSS、JavaScript、メディアなどを読み込みます。しかし、10%のページは7MB以上のデータを読み込んでいます。モバイルデバイスでは、デスクトップに比べて平均的なWebページのロード量は0.1MB少なく、少なくともすべてのパーセンタイルでこの数値が当てはまります。

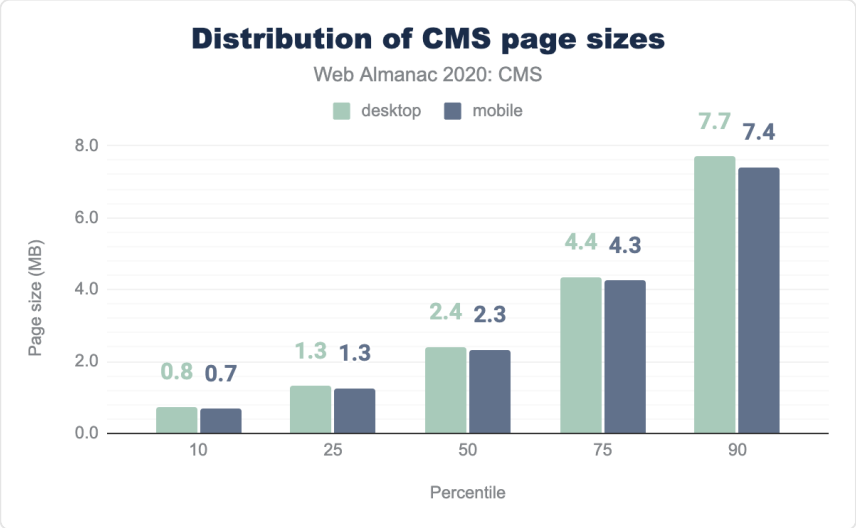


図15.14. CMSのページサイズの分布。

CMSでは、外部の画像や動画、スクリプト、スタイルシートなど、第三者のリソースを読み込むことが多くあります。

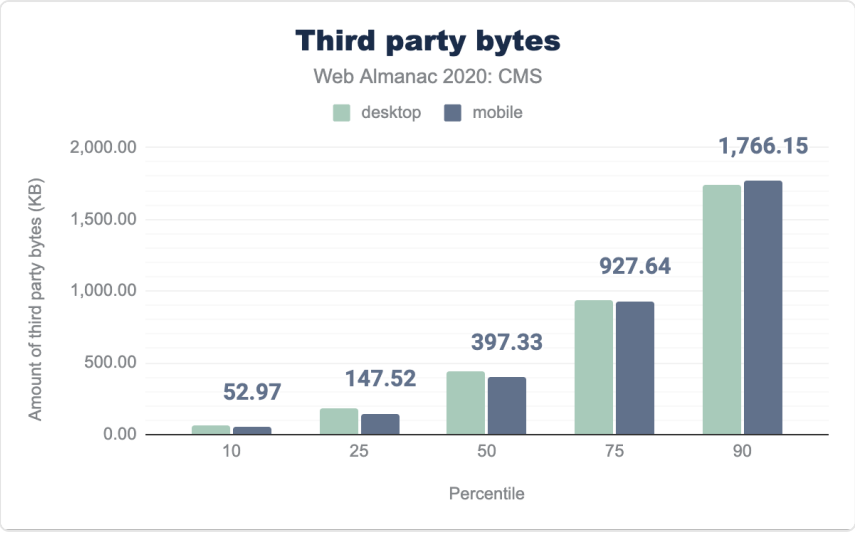


図15.15. サードパーティのバイト

デスクトップ用CMSページの中央値は、サードパーティからのリクエストが27件、コンテンツは436KBで、モバイル用では26件、コンテンツは397KBであることがわかりました。

CMSがページのロードサイズに影響を与える主な方法の1つは、より効率的なフォーマットの使用をサポートし、奨励することです。画像は、ビデオに比べてページの重さへの影響が大きいです。

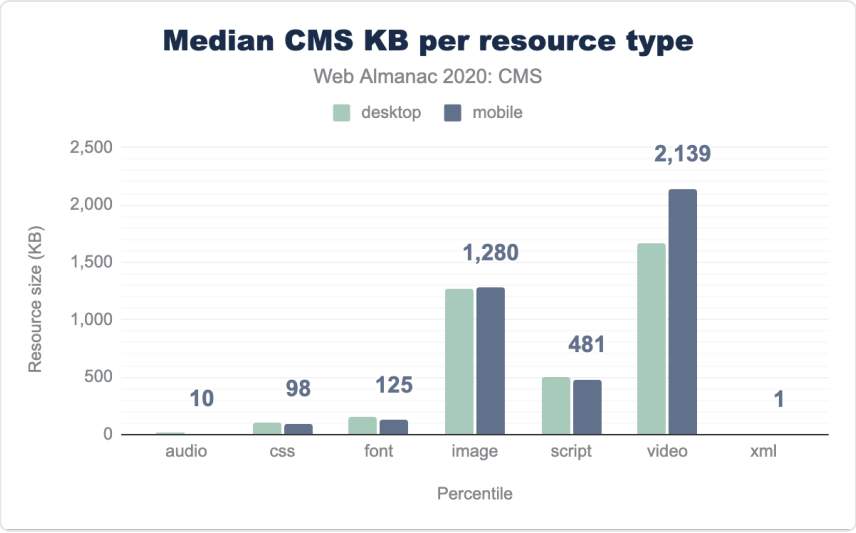


図15.16. リソースタイプごとのCMS KBの中央値。

ここでは、ビデオがリソースタイプごとに大きな割合を占めています。動画をより効率的にすることや、自動再生を止めることによる影響など、今後の研究対象として興味深い分野です。ここでは、画像に焦点を当てます。一般的な画像フォーマットは、JPEG、PNG、GIF、SVG、WebP、ICOです。これらの中で、WebPはほとんどの状況でもっとも効率的です[608]。WebPのロスレス画像は、同等のPNGと比べて26%小さく[609]、同等のJPGと比べて25〜34%小さく[610]なります。しかし、WebPはすべてのCMSページにおいて、2番目に人気のない画像フォーマットであることがわかります。

608. https://developers.google.com/speed/webp/
609. https://developers.google.com/speed/webp/docs/webp_lossless_alpha_study#results
610. https://developers.google.com/speed/webp/docs/webp_study

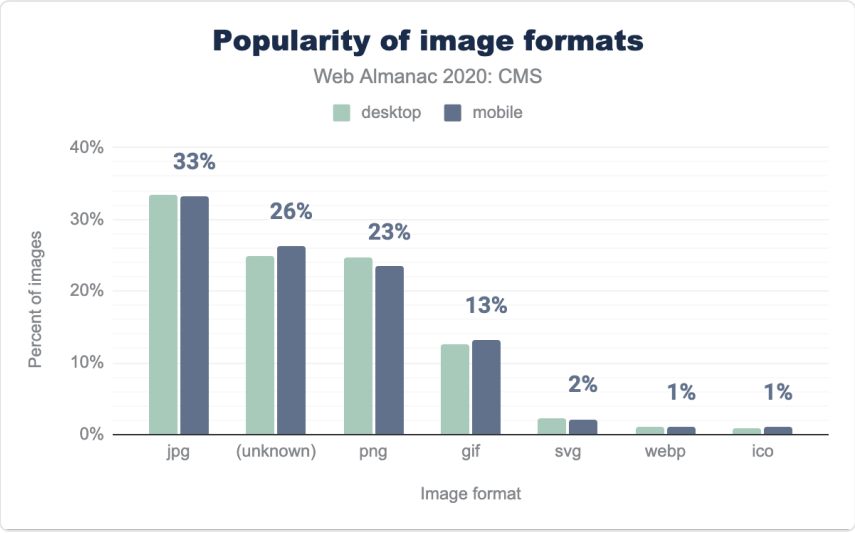


図15.17. 画像フォーマットの普及

上位5つのCMSのうち、Wixだけが自動的にWebPフォーマットの画像を変換して提供しています。WordPress、Drupal、Joomlaは拡張機能でWebPをサポートしていますが、本稿執筆時点ではSquarespaceはWebPをサポートしていません。

先に見たように、Wixは「良好な」LCPを持つサイトの割合がもっとも低い結果となりました。WixがWebPの画像バイトを効率的に利用していることはわかっていますが、画像フォーマット以外にもLCPパフォーマンスに影響を与える問題があることは明らかです。しかし、WebPはより効率的なフォーマットであり、もっとも人気のあるCMSによるこのフォーマットのネイティブサポートの改善は有益です。

画像フォーマットは、画像をより効率的にするための1つのメカニズムです。また、画像の「レイジーローディング」のような他のメカニズムについても、今後の研究が必要です。

CMSが環境に与える影響という質問に完全に答えることはできませんが、答えを出すために貢献しています。CMSには環境への影響を真摯に受け止める責任があり、平均ページ重量を減らすことは重要な仕事です。

結論

この1年でCMSの重要性は増しています。インターネット上でコンテンツを作成したり、消費したりするのに欠かせない存在であり、この状況は今後も変わることはないでしょう。CMSの重要性は、年々高まっています。

私たちはCMSの採用、CMSで作成されたウェブサイトのユーザー体験を検討し、また、はじめてCMSが環境に与える影響を調べました。ここでは多くの疑問に答えましたが、さらに多くの疑問が残っています。本章を基にした更なる研究を歓迎します。私たちは、CMSが注意を払う必要のあるいくつかの分野を強調しました。2021年の報告書で共有すべき進展があることを期待しています。

CMSは、インターネットとオープンウェブの成功に不可欠です。継続的な進歩に向けて頑張りましょう。

著者

Alex Denning

@AlexDenning alexdenning https://getellipsis.com/

Alex Denningは、WordPressビジネス向けのマーケティングエージェンシーである Ellipsis Marketing[611] の創設者です。アレックスはWordPressのコア・コントリビューターであり、WordCamp London[612]の開催にも協力しています。

611. https://getellipsis.com/
612. https://london.wordcamp.org/

部 III 章 16
Eコマース

Rockey Nebhwani と Jason Haralson によって書かれた。
Alan Kent によってレビュ。
Jason Haralson と Rockey Nebhwani による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

「Eコマース・プラットフォーム」とは、オンラインストアを作成・運営するためのソフトウェアやサービスのセットのことです。Eコマース・プラットフォームには、たとえばいくつかの種類があります。

- Shopifyなどの有料サービスは、あなたのストアをホストし、あなたが始めるのをサポートします。ウェブサイトのホスティング、サイトやページのテンプレート、商品データの管理、ショッピングカート、決済などを提供しています。
- Magentoオープンソースのようなソフトウェアプラットフォームで、お客様自身がセットアップ、ホスト、管理を行います。これらのプラットフォームは強力で柔軟性がありますが、Shopifyなどのサービスに比べて設定や運営が複雑になる場合があります。

- Magento Commerceのようなホスティング型のプラットフォームは、セルフホスティング型のプラットフォームと同じ機能を提供しますが、ホスティングはサードパーティによってサービスとして管理されます。

昨年の分析では、Eコマースプラットフォーム上に構築されたサイトのみを検出することができました。そのため、Amazon、JD、eBayなどの大規模なオンラインストアやマーケットプレイス、または自社のプラットフォームを使用して構築されたEコマースサイト（一般的に大企業が使用する）は、分析の対象外となっていました。今年の分析では、WappalyzerのEコマースサイトの検出機能を強化することで、この制限に対処しました。詳細はプラットフォーム検出の項を参照してください。

また、ここでのデータはホームページのみで、カテゴリーや製品などのページは含まれていないことに注意してください。方法論については、こちらをご覧ください。

プラットフォーム検出

あるページがECプラットフォームであるかどうかをどのように確認するのですか？検出はWappalyzerによって行われます。Wappalyzerは、ウェブサイトで使用されている技術を明らかにするクロスプラットフォームのユーティリティです。コンテンツマネジメントシステム、Eコマースプラットフォーム、ウェブサーバー、JavaScriptフレームワーク、分析ツールなど、さまざまなテクノロジーを検出します。

2019年と比較すると、2020年にはECサイトの%が大幅に増加していることがわかります。これは主に、今年のWappalyzerではセカンダリーシグナルを使った検出が改善されたためです。これらのセカンダリーシグナルには以下のものがあります。

- Google Analytics Enhanced Ecommerceタグを使用しているサイトは、Eコマースサイトとしてカウントされます。
- セカンダリーシグナルには、'カート'リンクを識別するためにもっともよく使われるパターンを探すことも含まれます。

この方法論の変更により、ヘッドレスソリューションを使用して構築されたエンタープライズプラットフォームやサイトへの対応が強化されました。

制限事項

私たちの手法には以下のような限界があります。

- commercetools[613]のようなヘッドレスのEコマースプラットフォームはEコマースプラットフォームとして検出されないかもしれませんが、そのようなサイトでカートの存在を検出できた場合は、そのようなプラットフォームを使用しているサイトも全体のカバー率の統計に含めています。
- 一般的にホームページ以外で展開される技術（例：製品詳細ページのWebAR）は検出されません。
- 当社のクロールは米国で行われているため、米国固有のプラットフォームに偏りがあります。たとえば、グローバル企業が国ごとに異なるプラットフォームでECサイトを構築している場合（国別のドメインやサブドメインを使用している場合）、このような地域ごとの違いは分析結果に反映されない可能性があります。
- B2Bサイトではカート機能をログイン時に隠すのが一般的で、そのため今回の調査はB2B市場を正しく表していないと考えられます。

Eコマースプラットフォーム

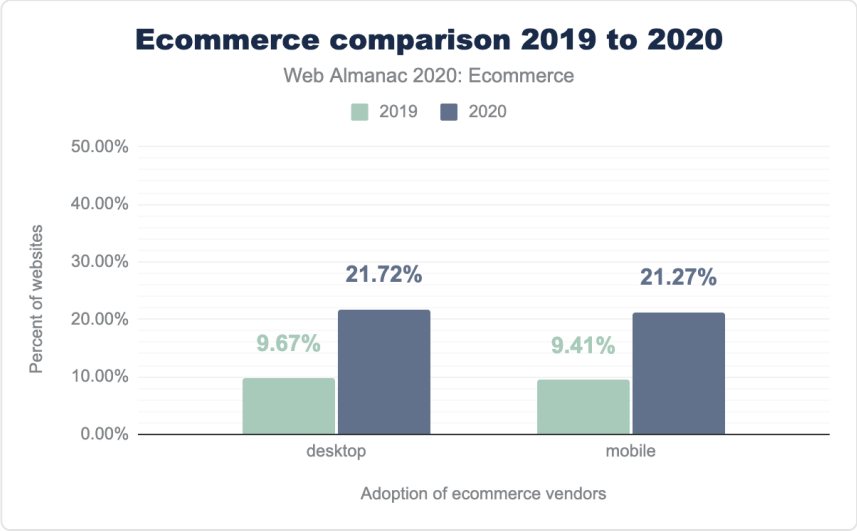


図16.1. Eコマースの比較2019年から2020年まで。

合計では、モバイルウェブサイトの21.72%、デスクトップウェブサイトの21.27%がECプラットフォームを利用していました。2019年は、同数値がモバイルWebサイトで9.41%、

613. https://commercetools.com/

デスクトップWebサイトで9.67%でした。

注：この増加は、主にECサイトを検出するためにWappalyzerに施された改良によるもので、Covid-19による成長などの他の要因に起因するものではありません。また、2019年の統計には誤りを考慮して遡及的に軽微な修正が適用されたため、2019年のパーセンテージは2019 Eコマース[614]の章で示されたものとは若干異なります。

主要なEコマースプラットフォーム

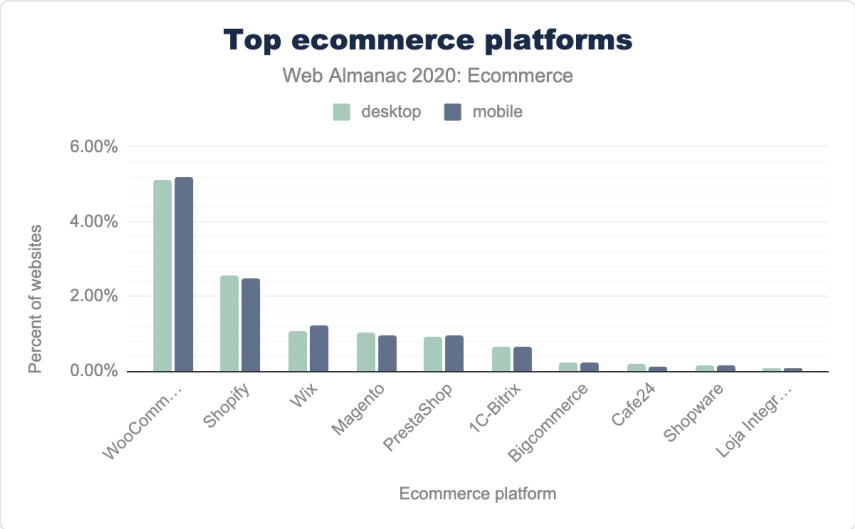


図16.2. トップのEコマースプラットフォーム。

当社の分析では、145の独立したEコマースプラットフォームを数えました（昨年の分析では116[615]と比較しています）。このうち、市場シェアが0.1%以上のプラットフォームは9つしかありません。WooCommerceはもっとも一般的なEコマースプラットフォームであり、1位の座を維持しています。Wixは、Wappalyzerが2019年6月30日からEコマースプラットフォームとして識別するようになってから、今年はじめてこの分析に登場しました。

企業向けEコマースプラットフォームのトップ

プラットフォームの階層を明確にすることは困難ですが、ここではエンタープライズ階層に重点を置いているベンダー4社（Salesforce、HCL、SAP、Oracle）を紹介します。

614. *https://almanac.httparchive.org/ja/2019/ecommerce*
615. *https://almanac.httparchive.org/ja/2019/ecommerce#ecommerce-platforms*

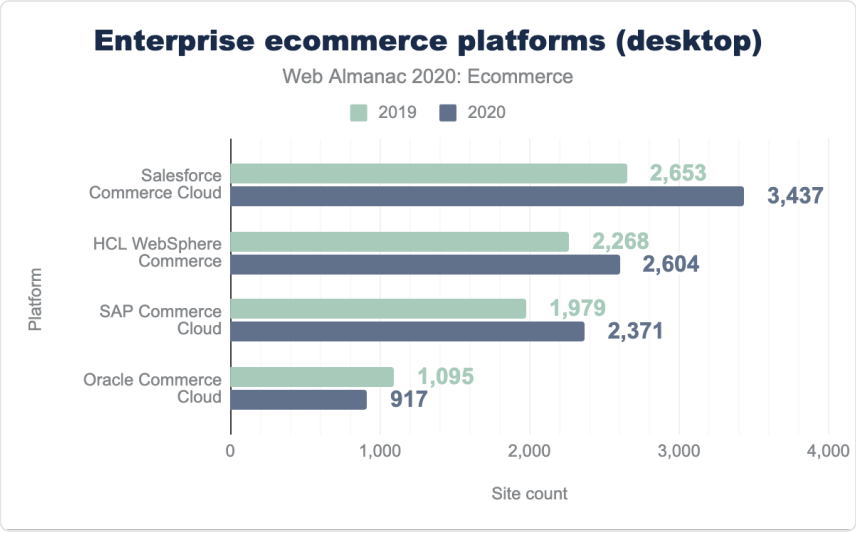


図16.3. エンタープライズEコマースプラットフォーム（デスクトップ）。

このグループからは、Salesforce Commerce Cloudが引き続きリードするプラットフォームとなっています。2020年のデスクトップWebサイトは3,437件で、2019年の2,653件から29.5%増加しています。4つの企業向けEコマースプラットフォームのうち、Salesforceのウェブサイトは36.8%を占めています。

HCLテクノロジーズは、2019年7月にIBMからWebSphere Commerceを買収。この移行は2020年に複雑な結果をもたらした。HCLのWebSphere Commerceは、2019年の2,268のデスクトップWebサイト数から、今年は2,604まで14.8%増加したものの、このグループ内では27.9%まで0.5%人気が落ちていたのです。今後の動向に注目したいです。

SAP Commerce Cloud（正式名称：Hybris）は、昨年の24.8%からわずかに増加した25.4%で、依然として3番目に人気のあるエンタープライズEコマースプラットフォームです。2,371のデスクトップサイトは、2019年に見つかったHybrisが起因する1,979のデスクトップサイトから19.8%増加しています。

最後に、Oracle Commerce Cloudは残念ながら2019年から2020年の間に少し牽引力を失いました。デスクトップのウェブサイトは1,095件から917件へと16%減少し、ひいてはエンタープライズのEコマースプラットフォームの足場も13.7%から9.8%へと落ちてしまいました。

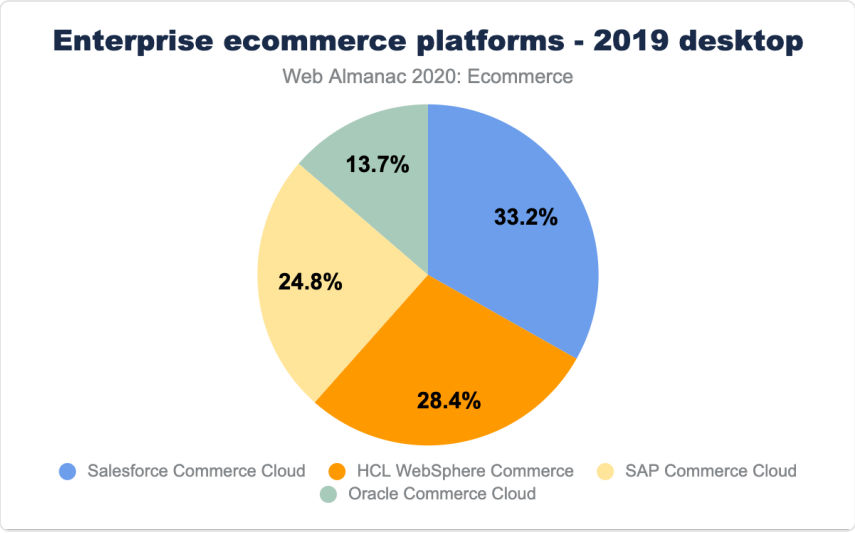


図16.4. 企業向けEコマースプラットフォーム - 2019年版デスクトップ

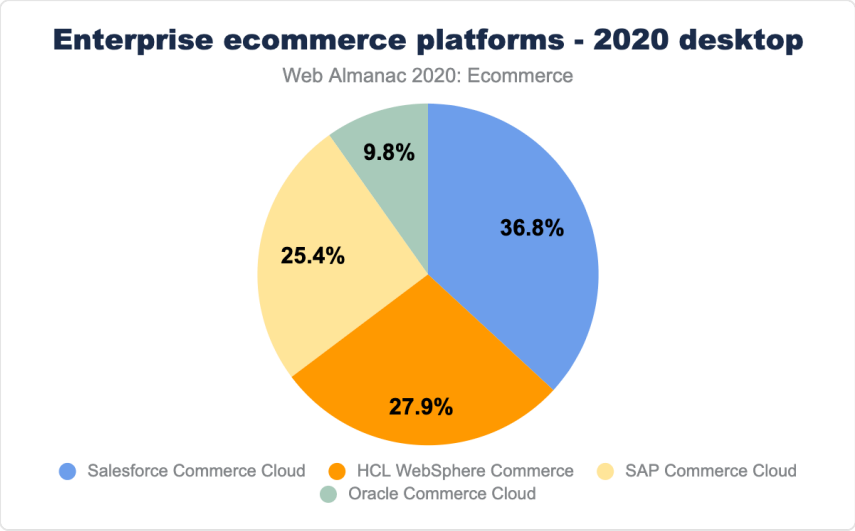


図16.5. 企業向けEコマースプラットフォーム - 2020年のデスクトップ

Shopifyの「Shopify Plus」、Adobeの「Magento Enterprise」、Bigcommerceの「Enterprise」などがあり、人気を集めていますがPlatform Detectionの制限により企業向けWebサイトをCommunityやCommercial Webサイトから切り離すことができません。

COVID-19のEコマースへの影響

COVID-19は世界に大きな影響を与えており、オンラインへの移行もさらに大きなものとなっています。電子商取引プラットフォームの全体的な増加を測定するには、本章のために一部で行われた検出の大幅な増加が影響します。その代わりに、すでに検出されていたいくつかのプラットフォームに注目し、とくにCOVIDが世界の大部分に影響を与え始めた2020年3月以降、その利用が増加していることを指摘します。

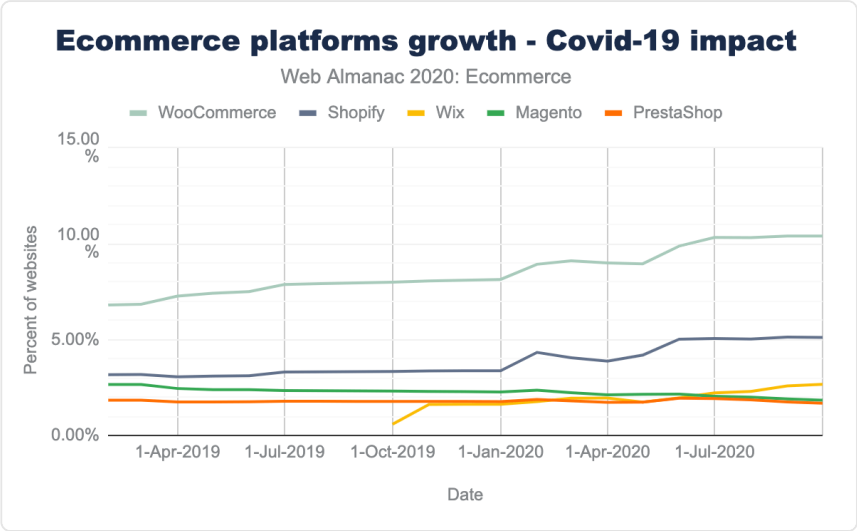


図16.6. Eコマースプラットフォームの成長Covid-19の影響

COVIDが世界に大きな影響を与え始めた頃、WooCommerceやShopifyのサイトが明らかに増加しています。

注：Wappalyzer Detection for Wix[616]は、サイトがCMSとしてWixを使用しているか、Eコマースプラットフォームとしてwixを使用しているかを区別しません。このため、Eコマースプラットフォームとしてのwixの成長は、上記のグラフでは正しく表されていない可能性があります。

ページの重さとリクエスト

Eコマースプラットフォームのページウェイトには、HTML、CSS、JavaScript、JSON、XML、画像、音声、動画のすべてが含まれます。

616. https://github.com/AliasIO/wappalyzer/pull/2731/commits/f44f20f03618f6a5fd868dd38ce9db5e2e2f1407

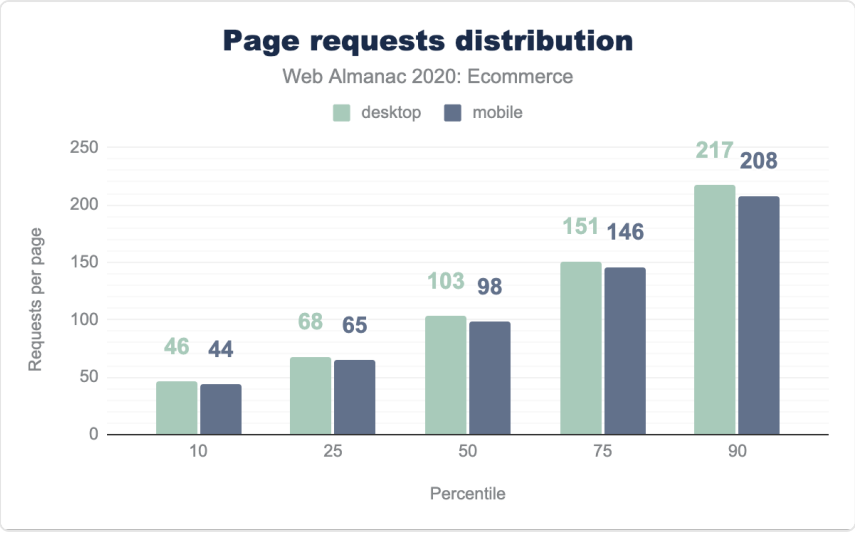


図16.7. ページが配信を要求する。

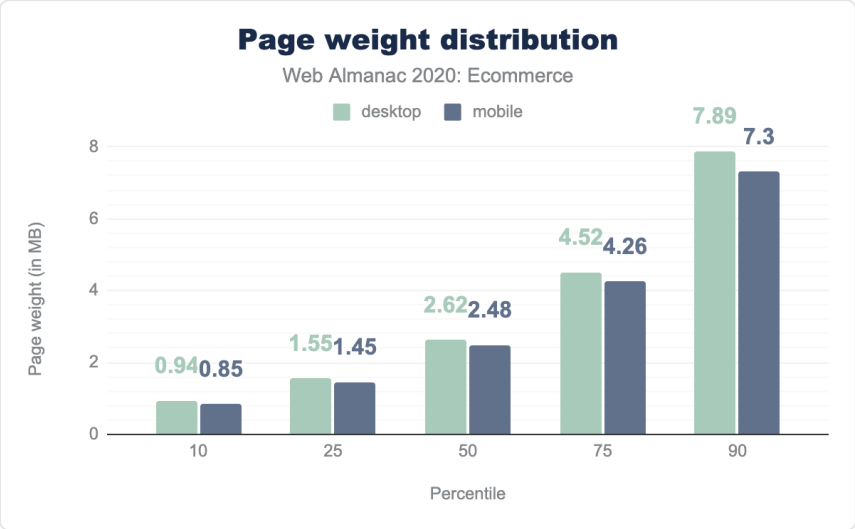


図16.8. ページの重量配分。

有望なことに、モバイルのページウェイトはすべてのパーセンタイルで2019年との比較[617]で低下しており、デスクトップのページウェイトは多かれ少なかれ変わらない（90パーセンタイルを除く）。ページあたりのリクエスト数も、モバイル（90パーセンタイルを除くすべて

617. https://almanac.httparchive.org/ja/2019/ecommerce#page-weight-and-requests

のパーセンタイルで9〜11リクエスト減少）とデスクトップで減少しました。

ページ重量の章で示したように、ECサイトは全サイトと比較して、リクエスト数やサイズが依然として大きいです。

リソースタイプ別ページ重量

リソースタイプ別に見ると、中央ページでは、画像とJavaScriptのリクエストがEコマースページでは圧倒的に多いことがわかります。

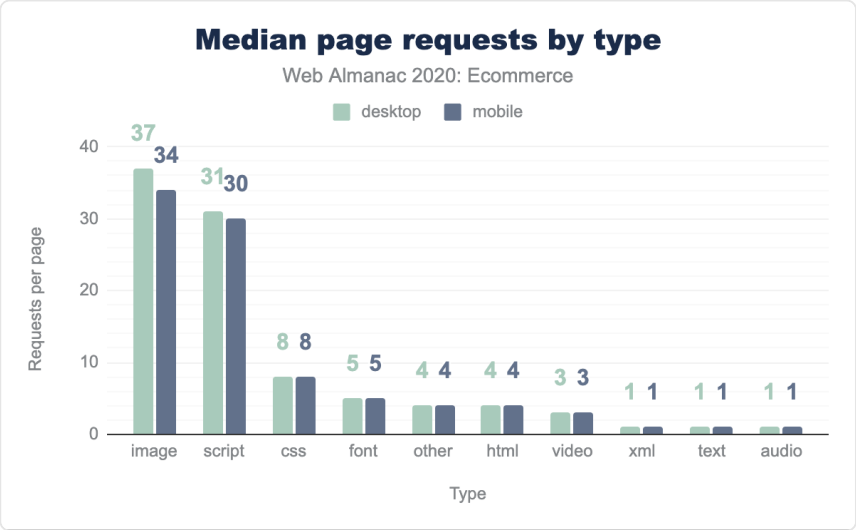


図16.9. タイプ別ページリクエストの中央値

しかし、実際の配信バイト数を見ると、メディアが圧倒的に大きな資産となっています。

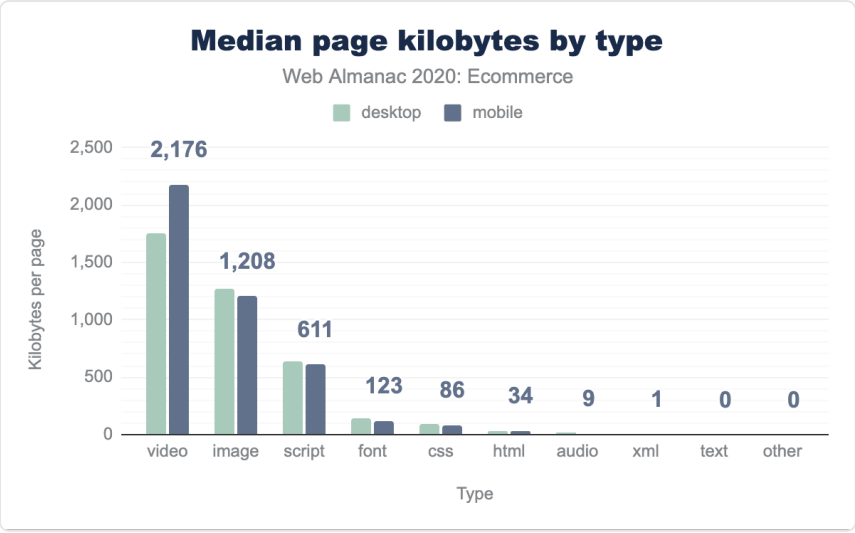


図16.10. タイプ別ページキロバイトの中央値

Eコマースサイトでは、リクエスト数が少ないにもかかわらず、動画が最大のリソースとなっており、次いで画像、JavaScriptとなっています。

HTMLペイロードサイズ

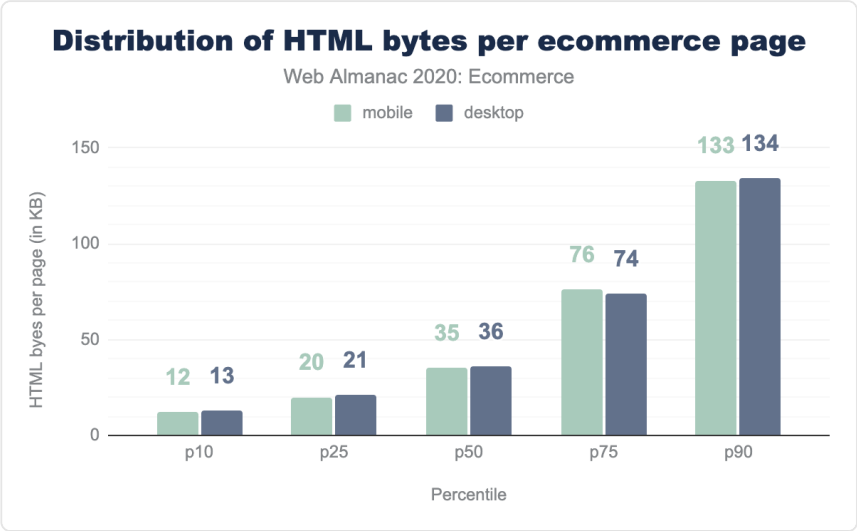


図16.11. ECページごとのHTMLバイト数の分布

なお、HTMLペイロードには、外部リンクとして参照されるのではなく、マークアップ自体に直接インラインのJSON、JavaScript、CSSなどのコードが含まれている場合があります。EコマースページのHTMLペイロードサイズの中央値は、モバイルでは35KB、デスクトップでは36KBとなっています。2019年との比較[618]では、ペイロードサイズの中央値と10、25、50パーセンタイルの割合はほぼ変わりません。しかし、75パーセンタイルと90パーセンタイルでは、モバイルとデスクトップでそれぞれ約10kbと15kbの増加が見られます。

モバイルのHTMLペイロードサイズは、デスクトップとあまり変わらない。つまり、デバイスやビューポートの大きさが違っても、サイトが大きく異なるHTMLファイルを配信していないようです。

画像の使い方

次に、Eコマースサイトでの画像の使われ方を見てみましょう。なお、今回のデータ収集方法では、クリックやスクロールなどのページ上でのユーザーの操作をシミュレートしていないため、遅延読み込みされた画像はこの結果に含まれていません。

618. *https://almanac.httparchive.org/ja/2019/ecommerce#html-payload-size*

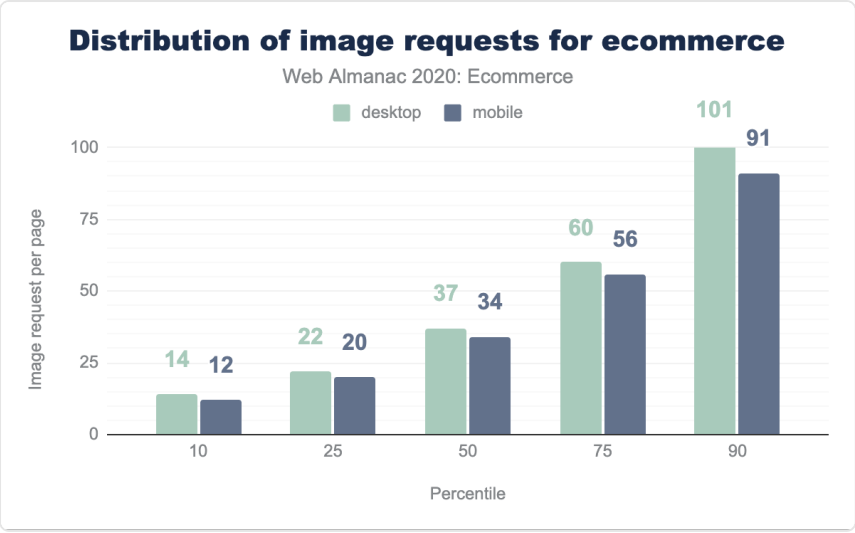


図16.12. 電子商取引における画像要求の分布

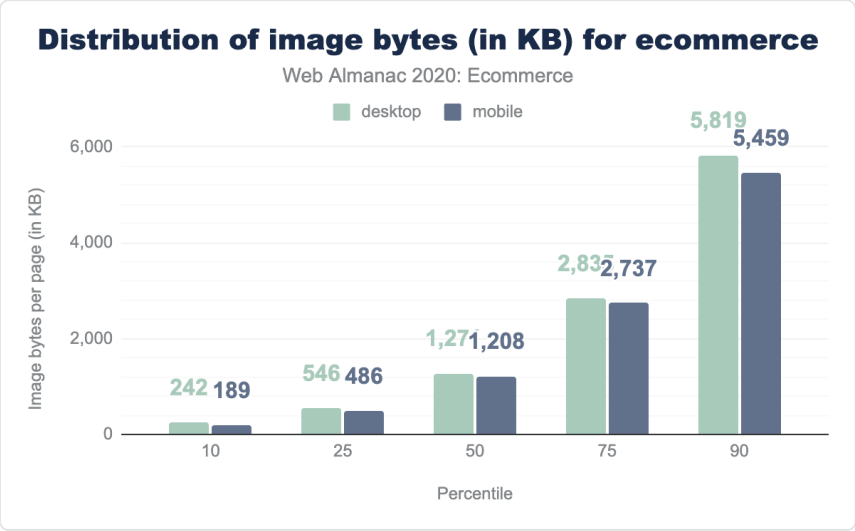


図16.13. Eコマース向けイメージバイトの配布

上の図によると、中央値のEコマースページには34枚の画像があり、画像ペイロードはモバイルで1,208KB、デスクトップでは37枚の画像と1,271KBとなっています。10%のホームページには90枚以上の画像があり、画像ペイロードはモバイルで5.5MB、デスクトップで5.8MBとなっています。

2019年との比較[619]では、画像リクエストの中央値と画像ペイロードの中央値の両方が低下しています。画像リクエストの中央値は、モバイルとデスクトップの両方で3減少しました。また、画像ペイロードの中央値は、モバイルとデスクトップの両方で約200kb〜250kb減少しました。この減少は、現在より多くのブラウザ[620]でサポートされている `loading="lazy"` 属性の使用など、サイトが遅延読み込み技術を採用していることが要因と考えられます。今年のMarkupの章では、ネイティブの遅延ローディングの使用が増加しているようで、2020年8月には約3.86%のページでこの属性が使用されており、これは継続的に増加しているという見解が示されています（このツィート[621]で見られます）。

一般的な画像フォーマット

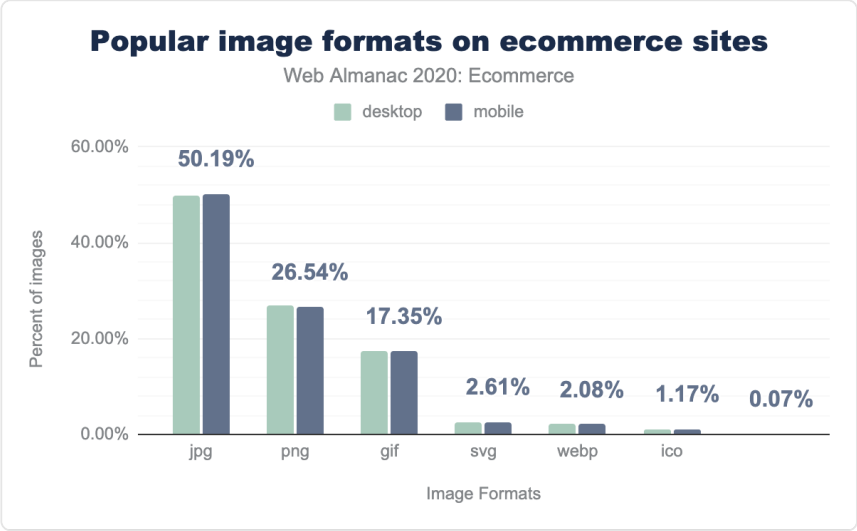


図16.14. Eコマースサイトで人気のある画像フォーマット

なお、画像サービスやCDNの中には、`.jpg` や `.png` などの接尾辞を持つURLであっても、WebPをサポートするプラットフォームには、JPEGやPNGではなくWebPを自動的に配信するものがあります。たとえば、`IMG_20190113_113201.jpg` は、ChromeではWebP画像を返します。しかし、HTTP Archiveが画像フォーマットを検出する方法は、まずMIMEタイプのキーワードをチェックし、次にファイル拡張子にフォールバックするというものです。つまり、HTTP ArchiveがユーザーエージェントとしてサポートしているのはWebPなので、上記のようなURLを持つ画像のフォーマットはWebPとして与えられることになります。

PNGの使用率は、ほぼ2019年と同水準[622]で推移しました（デスクトップ、モバイルともに27%）。JPEGの使用率は減少しました（デスクトップ4%、モバイル6%）。この減少分のう

619. https://almanac.httparchive.org/ja/2019/ecommerce#image-stats
620. https://caniuse.com/loading-lazy-attr
621. https://x.com/rick_viscomi/status/1344380340153016321?s=20
622. https://almanac.httparchive.org/ja/2019/ecommerce#png

ち、ほとんどがGIFの使用増加につながっています。GIFはEコマースのホームページではよく使われていますが、商品の詳細ページではあまり使われていないかもしれません。私たちの手法ではホームページのみを対象としているため、このことが、EコマースサイトでGIFの使用率が著しく高いことを説明しています。Lighthouseの監査では、「アニメーションコンテンツ用のビデオフォーマット」の使用を推奨しています。これは、GIFのアニメーション特性を維持しつつ、パフォーマンスを最適化するためにEコマースサイトが利用できる手法です。詳しくはこの記事[623]をご覧ください。

EコマースサイトにおけるWebPの使用率は、2019年には合計1%だったのが2020年には2%と倍増したものの、依然として非常に低い水準にとどまっています。WebPフォーマットは10年近く前のもので、`picture`要素を使ったプログレッシブ・エンハンスメントを可能にした後も、使用率は低いままです。2020年には、SafariがSafari 14[624]でサポートを導入したことで、WebPは新たな息吹を得ました。しかし、今年のWeb Almanacは2020年8月を基準としており、Safariのサポートは2020年9月に行われたため、ここで紹介されている統計はSafariによって追加されたサポートの影響を反映していません。

今年、Chrome 85（2020年8月リリース）では、WebP[625]と比較してより効率的な画像フォーマットであるAVIFのサポートも確認できました。来年の分析では、EコマースサイトでのAVIFの使用状況を取り上げたいと考えています。WebPと同様に、AVIFもプログレッシブ・エンハンスメントであり、クロスブラウザの問題[626]へ対処するために`picture`要素を使って実装できます。

筆者の経験によると、エンジニアリングチームでは、CDNが提供する画像最適化サービスについての認識が不足しています。CDNは、コードに触れることなくほとんどの重い作業を行うことができます。たとえば、Adobe Scene7は、Smart Imaging solution[627]の下でこのサービスを提供しています。また、Salesforce Commerce Cloudのクライアントは、プラットフォームに組み込まれたCDN機能（Cloudflareを使用）を利用することで、簡単な設定でこの機能を有効にできます。このようなソリューションの認知度を高めることで、より効率的なフォーマットへの移行を促進できます。

画像のサイズやフォーマットによるCRUXメトリクスの改善に興味をお持ちの読者のためにもう1つのポイントをご紹介します。現在、プログレッシブ画像は、ユーザーが知覚するパフォーマンスに役立つにもかかわらず、Largest Contentful Paintに対する加重はありません。このトピックについては、コミュニティで興味深い議論[628]が行われており、将来的にはプログレッシブ画像がLCPへ寄与するようになる可能性があります。また、2021年5月からページエクスペリエンスシグナルにCore Web Vitalsが含まれるようになったことで、プログレッシブローディングをサポートするフォーマットに対するEコマースコミュニティの関心が、再び高まる可能性があります。

623. *https://web.dev/replace-gifs-with-videos/*
624. *https://caniuse.com/webp*
625. *https://www.ctrl.blog/entry/webp-avif-comparison.html*
626. *https://caniuse.com/avif*
627. *https://helpx.adobe.com/uk/experience-manager/6-3/assets/using/imaging-faq.html*
628. *https://github.com/WICG/largest-contentful-paint/issues/68*

第三者のリクエストとバイト

Eコマースのプラットフォームやサイトでは、しばしばサードパーティのコンテンツが利用されています。当社は、サードパーティの使用を検出するために、サードパーティウェブプロジェクトを使用しています。

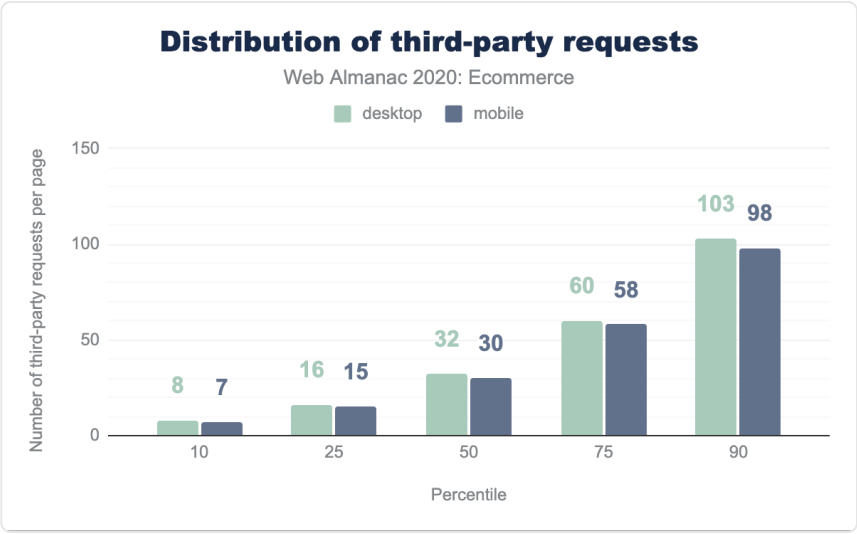


図16.15. 第三者からの依頼の分配

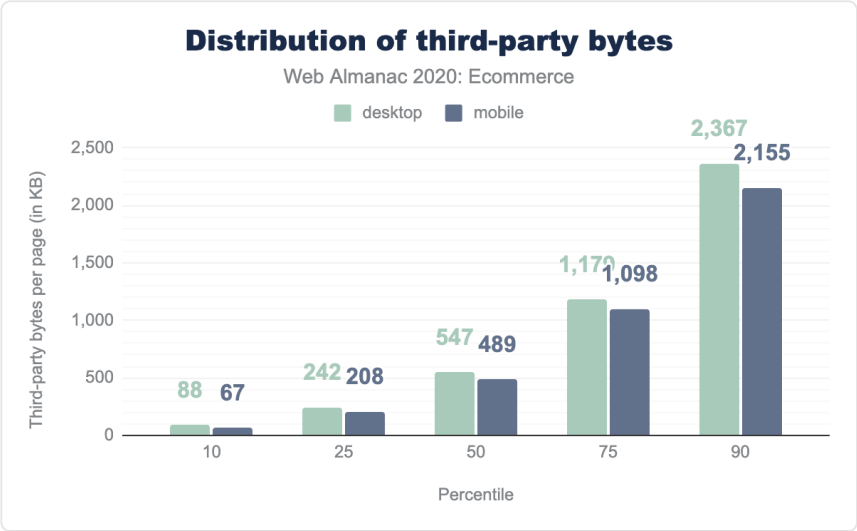


図16.16. サードパーティ製バイトの配布

昨年のサードパーティのデータと比較して[629]サードパーティのリクエストとバイトの使用が大幅に増加していることがわかりますが、特定の原因や顕著な検出の変化を特定することができませんでした。この1年でサードパーティの使用量が基本的に2倍になっているようなので、これに関する読者の皆様のご意見[630]をぜひお聞かせください。

Eコマースのユーザー体験

電子商取引は顧客を獲得することがすべてであり、そのためには高速で動作するウェブサイトがもっとも重要です。このセクションでは、Eコマースサイトの実際のユーザーエクスペリエンスに光を当ててみます。そのために、ユーザーが感じるパフォーマンス指標を分析します。この指標は、3つのCore Web Vitals[631]指標で表されます。

Chromeユーザーエクスペリエンスレポート

このセクションでは、Chrome User Experience Reportで提供されている3つの重要な要素を見てみましょう。これらの要素は、ユーザーが実際にECサイトをどのように体験しているのかを理解する上でのヒントになります。

- 最大のコンテンツフルペイント (LCP)
- 最初の入力までの遅延 (FID)
- 累積レイアウト変更 (CLS)

これらのメトリクスは、優れたWebユーザーエクスペリエンスを示す中核的な要素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詳しく説明していますが、ここではECサイトに特化してこれらのメトリクスを見ていきたいと思います。それでは、それぞれの項目を順に見ていきましょう。

最大のコンテンツフルペイント

最大のコンテンツフルペイント（LCP）は、ページのメインコンテンツが読み込まれたと思われる時点を測定し、その結果、そのページがユーザーにとって有用であるかどうかを判断します。これは、ビューポート内に表示される最大の画像や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実現しています。

これは、ページが読み込まれてからテキストや画像などのコンテンツが最初に表示されるま

629. https://almanac.httparchive.org/ja/2019/ecommerce#third-party-requests-and-bytes
630. https://discuss.httparchive.org/t/2052
631. https://web.dev/i18n/ja/vitals/

での時間を測定する最初のコンテンツフルペイント（FCP）とは異なります。LCPは、ページのメインコンテンツが読み込まれるタイミングを測るのに適したプロキシとされています。

Eコマースの場合、この指標は、ユーザーにとってもっとも有用なコンテンツ（ランディングページのヒーローバナー画像、検索/リストページに表示される1つ目の商品の画像、商品詳細ページの場合の商品画像など）を示す非常に良い指標となります。この指標が導入される前は、サイトはRUMソリューションでサイトを明示的に測定しなければなりませんでしたが、この指標により、測定を行うためのリソースや専門知識を持たない人でも測定が可能になりました。

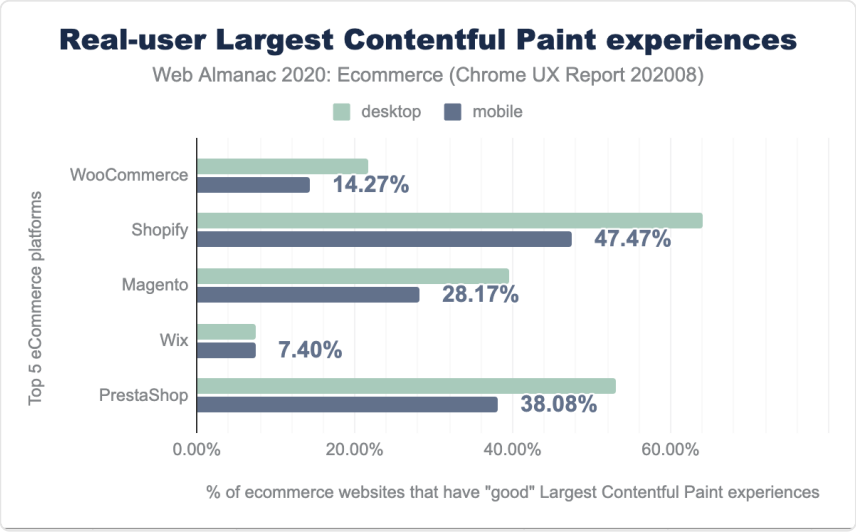


図16.17. リアルユーザー最大のコンテンツフルペイント体験

WixとWooCommerceのスコアはとくに低く、主要なプラットフォーム間で大きなばらつきが見られます。もっとも利用されている3つのEコマースプラットフォームのうちの2つであるため、改善すべき点があるようです。

最初の入力までの遅延

最初の入力までの遅延 (FID)は、インタラクティブ性を測定しようとするもので、より重要なのは、ページの処理に追われている間にページが無反応になった場合のインタラクティブ性の障害を測定することです。

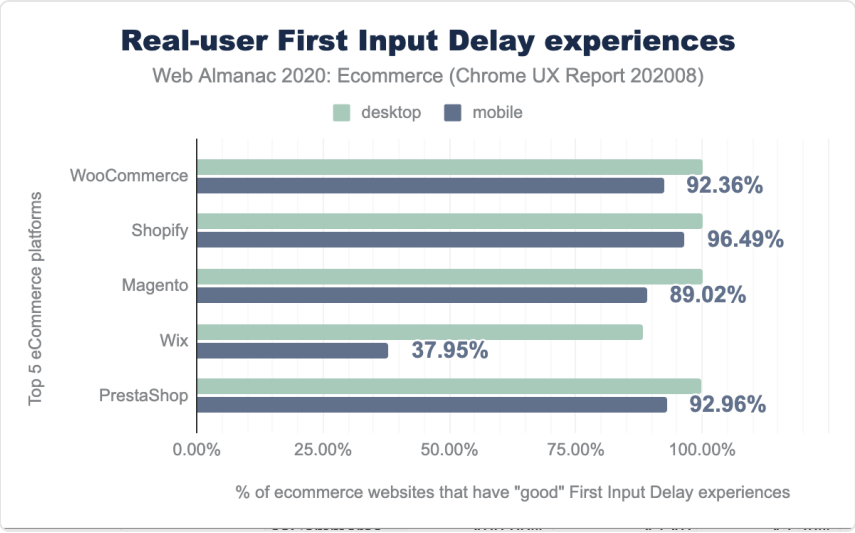


図16.18. リアルユーザーの初回入力遅延体験

一般的に、FIDスコアは他のCore Web Vitalsよりも高く、先に見たようにメディアやJavaScriptを多用しているにもかかわらず、ECサイトがこのカテゴリーで高いスコアを維持しているのは有望です。

累積レイアウト変更

累積レイアウト変更（CLS）は、新しいコンテンツが読み込まれてページに配置されたとき、ページがどれだけ「動く」かを測定します。当社のクロールでは、これは「折り目」の上で最初にページが読み込まれた場合に限られますがEコマースサイトではページの折り目の下やその他のインタラクションが、当社の統計が示す以上にCLSへ影響を与える可能性があることを理解する必要があります。

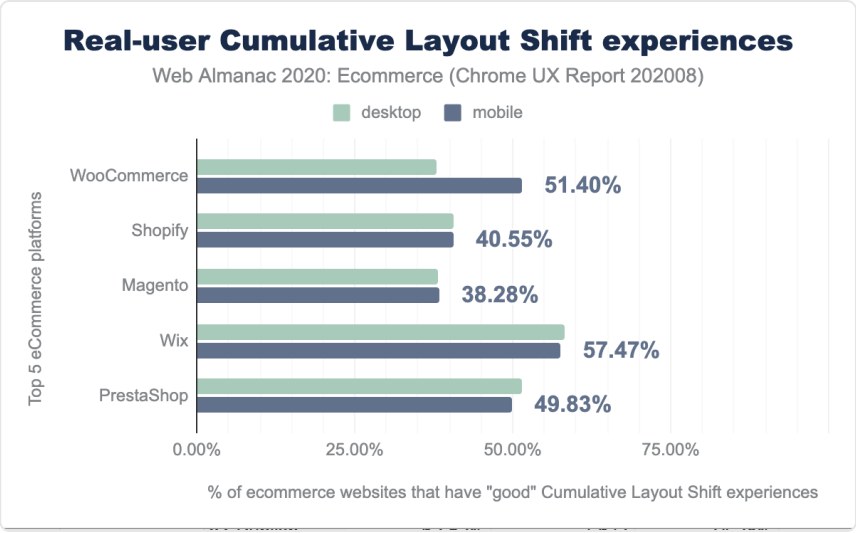


図16.19. リアルユーザーの累積レイアウトシフト体験

約半数のEコマースサイトでCLSのスコアが高く、興味深いことに、モバイルとデスクトップでほとんど差がありません。しかし、モバイル機器は通常、電力不足であり、ネットワークの変化が激しいという常識があります。

Core Web Vitals全体

Core Web Vitals全体を見ると、どのサイトが3つのコアメトリクスすべてに合格しているのか、次のようになっています。

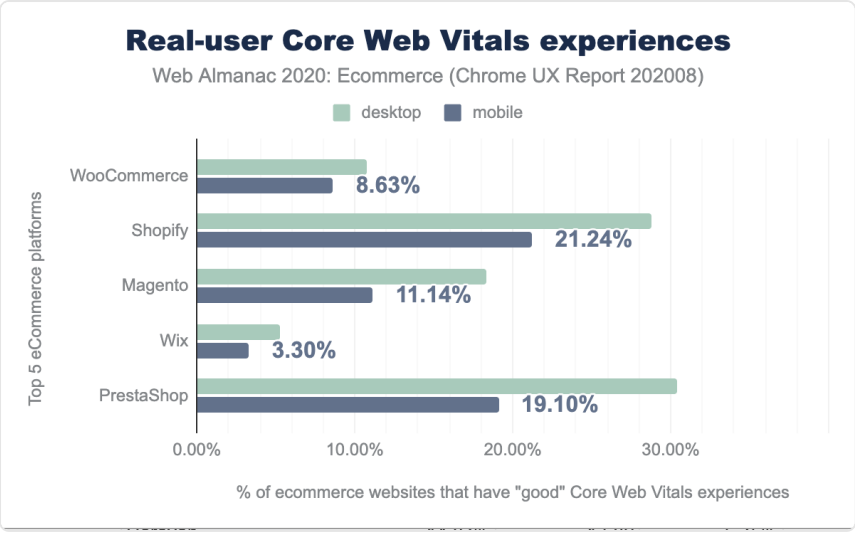


図16.20. Core Web Vitalsのリアルユーザーの体験談

これは先ほどのLCPのチャートと非常によく似ていますが、変動幅がもっとも大きく、この指標で不合格となったサイトがもっとも多かったことから、やや当然のことかもしれません。

ツール

Eコマースサイトでは、アナリティクス、タグマネージャー、コンセント管理プラットフォーム、アクセシビリティ・ソリューションなどの一般的なツールをどのように使用しているのでしょうか？

分析

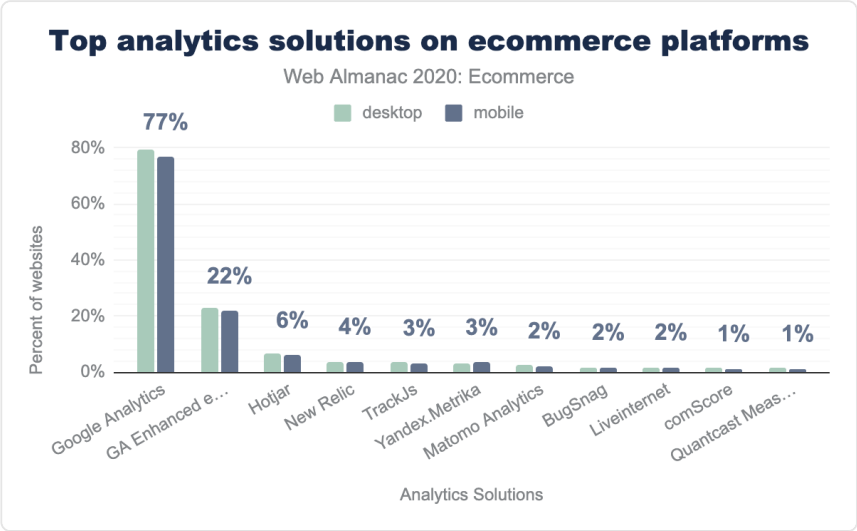


図16.21. Eコマースサイトにおける分析のトップのソリューション

これは、55%のサイトがGoogle Analyticsをより有効に活用する機会があることを示しているか、あるいは、サイトによってはチェックアウトファネルでしかGoogle Analyticsを使用しない場合があるため、ホームページに限定している当社の手法の限界を示していると考えられます。

HotJarはサイトの利用状況を分析し、コンバージョン率を向上させるために、Eコマースサイトでよく使用されるツールですが、モバイルサイトでの使用率は6%と非常に低くなっています。

タグマネージャー

Eコマースサイトでもっとも使用されているタグマネージャーは、依然としてGoogle Tag Managerであり、次いでAdobe Tag Managerとなっています。Google Tag Managerの無料という性質上、これが変わることはないと思われます。また、2020年8月、GoogleはGoogle Tag Managerでサーバーサイドタギング[632]を開始しました。サーバーサイドタギングを実装すると、ECサイトにはわずかなコストがかかりますが、サイトがサードパーティのオーバーヘッドを排除できるため、Total Blocking Time (TBT)、First Input Delay (FID)、Time to Interactive (TTI)などの指標を改善できます。Simon Ahava氏は、自身のブログ[633]に多くの有

632. https://developers.google.com/tag-manager/serverside
633. https://www.simoahava.com/analytics/server-side-tagging-google-tag-manager/

益な情報を掲載しており、読者にオススメしています。

サーバーサイド・タグの採用は、移行を容易にするため、サードパーティがサーバーサイド・テンプレートを提供するかどうかにかかっています。この章を書いている時点では、一般に公開されているコミュニティ・ギャラリー[634]には、サーバーサイド・テンプレートが見つかりませんでした。しかし、採用が増えれば、クライアントサイドとサーバーサイドのタギングを使用したサイトのパフォーマンススコアを比較することができ、興味深いものになるでしょう。AdobeやSignalのような他のベンダーも同様のサーバーサイドソリューションを提供しており、サイトはパフォーマンス向上のために採用を検討すべきでしょう。

タグマネージャー	デスクトップ	モバイル
Google Tag Manager	48.45%	46.56%
Adobe DTM	0.41%	0.38%
Ensighten	0.13%	0.13%
TagCommander	0.08%	0.07%
Signal	0.05%	0.03%
Matomo Tag Manager	0.02%	0.02%
Yahoo! Tag Manager	0.00%	0.00%
Total	**49.14%**	**47.20%**

図16.22. Eコマースサイトでのタグマネージャーの利用

注：上記の分析はWappalyzerによる検出に基づいており、サードパーティーの章で使用されているサードパーティーウェブデータセットを用いた分析とは異なる可能性があります。

コンセントマネジメント・プラットフォーム

今年のPrivacyの章では、あらゆるタイプのWebサイトにおけるコンセント管理プラットフォームの採用状況を取り上げました。Eコマースサイトでの導入と全サイトでの導入を比較すると、モバイル（Eコマースサイトでは4.2%、全サイトでは4.0%）とデスクトップ（Eコマースサイトでは4.6%、全サイトでは4.4%）の両方で導入率がやや高いです。

634. https://tagmanager.google.com/gallery/#/?context=server&page=1

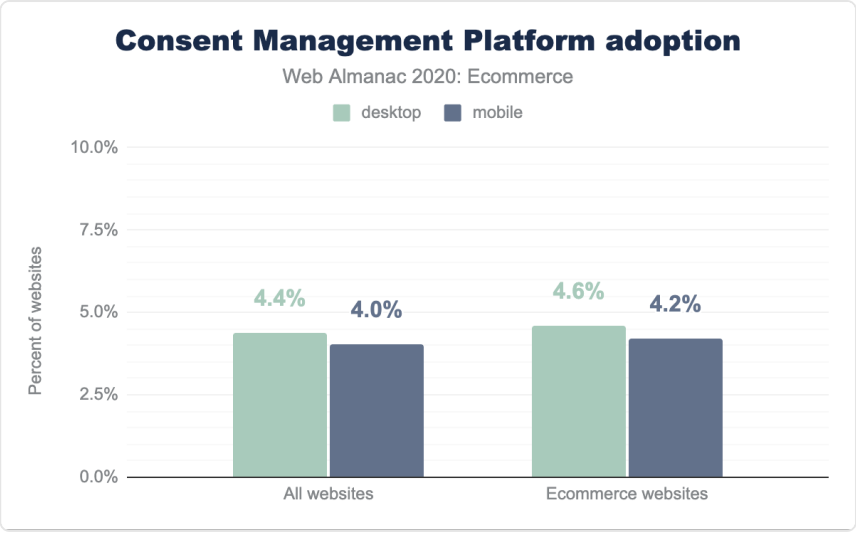


図16.23. コンセントマネジメントプラットフォームの採用

各種CMPのシェアを見ると、Eコマースサイトの傾向は、プライバシーの章で取り上げられているすべてのウェブサイトと同様でした。Web Almanacの今後の版では、より多くの国が独自の規制を打ち出しているため、この採用率が増加することを期待しています。また、Web Almanacチームは最近、Wappalyzerに「Content Management Platform」を追加しました。このチームはもっとも人気のあるCMPを追加しましたが、時間の経過とともにさらに多くのCMPが追加され、採用率が増加するものと思われます。

アクセシビリティソリューション

今年のアクセシビリティ章の紹介では、Web Almanacチームがクイックフィックス型のアクセシビリティ・ソリューションを導入することの危険性について語り、Lainey Feingold氏の素晴らしい記事、Web Accessibility Quick Fix Overlaysを避けるHonor the ADAを紹介しています。[635]。

推奨はされていませんが、Eコマースサイトにおけるこのようなソリューションの利用状況を調べたところ、モバイルサイトの0.47%、デスクトップサイトの0.54%がこのようなソリューションを導入していることがわかりました。

本章で採用した現在の方法では、トップレベルのEコマースサイトが、デザインによるアクセシビリティの実現を目指さずに、このような手っ取り早い方法を取っているかどうかを簡単に調べる方法がありません。将来的には、HTTP Archiveのデータを、International

635. https://www.lflegal.com/2020/08/quick-fix/

retailingによるTop 500 UK sitesなどの出版物と組み合わせることで、この点を確認することができるでしょう。

AMP採用

0.61%

図16.24. Eコマースサイト（モバイル）でのAMP利用。

SEOの章では、全ウェブサイトにおけるAMPの使用状況の統計を取り上げました。本章では、EコマースサイトにおけるAMPの採用状況を見ていきます。AMPはまだすべてのEコマースのユースケースをサポートしていないため、EコマースサイトでのAMP採用率は低いままです（モバイルで0.61%、デスクトップで0.66%）。また、今回の分析では、Wappalyzerによる検出に頼っているため、AMPが別ドメインとして実装されているEコマースサイトが`<link rel="amphtml"...>`要素を使って二重にカウントされている可能性があります。このようなドメインは、ECサイトの合計を算出する際にも2回カウントされるため、パーセンテージを見る上で問題となることはありません。

また、EコマースサイトのCRUXパフォーマンスを、AMP対応サイト（`amphtml`属性を使用して異なるドメインに実装されている場合）と比較することも検討しました。このような分析はAMPドメインのパフォーマンスに有意な差があったかどうかを確認するのに役立ちますが、EコマースウェブサイトにおけるAMPの採用率が低いため、このような分析で意味のある結果を得られない可能性があり、将来的に採用率が上昇した場合に分析を延期しました。

Webプッシュ通知

マーケターはプッシュ通知が大好きですが、筆者の経験によると、2015年にChromeではじめてPush API[636]が導入されたにもかかわらず、Webプッシュ通知に関するマーケターの認知度はまだ非常に低いようです。私たちは、ECサイトにおける（サービスワーカーなどの技術を使って可能な）Webプッシュ通知の採用状況を調べてみました。CRUXの通知許可データの一部として、プッシュ受付率やプッシュプロンプトの解除率などの指標にアクセスしています。このデータがどのように取得され、どのような指標が得られるかについては、このGoogleの記事[637]を参照してください。

当社の分析では、ウェブプッシュ通知を利用しているのは、デスクトップのEコマースサイトでは0.68%、モバイルのEコマースサイトでは0.69%に過ぎないことがわかりました。プッシュ通知に関しては、お客様がプッシュ通知を便利だと感じることが重要です。そのために

636. https://developers.google.com/web/updates/2015/03/push-notifications-on-the-open-web
637. https://developers.google.com/web/updates/2020/02/notification-permission-data-in-crux

は、カスタマージャーニーの適切なタイミングで許可を求め、無関係な通知をユーザーに浴びせないことが重要です。プッシュ通知に対する顧客の疲労に対処するため、Chromeは、許可率が非常に低いサイトを自動的により静かな通知UI[638]に登録します（ただし、正確な閾値はまだ定義されていません）。受け入れ率がコントロールグループ内で改善されると、そのサイトは標準的なUIに戻されます。

PJ Mclachlan(Product Manager, Google)は、静かな通知UIへ陥らないよう安全な領域にいるため、最低でも50%の受け入れ率を目指す[639]ことと、80%以上の受け入れ率を目指すことについて話しています。Eコマースサイトの通知受け入れ率の中央値は、モバイルで13.6%、デスクトップでは13.2%です。中央値では、この受け入れ率には大きな問題があります。90パーセンタイルレベルでも、あまり良い数字ではありません（モバイル：36.9%、デスクトップ：36.8%）。Eコマースサイトは、この トークを参考にして、プッシュの受け入れ率を健全に保ち[640]、今後の虐待的な通知の変更に不意打ちを食らわないようにするための推奨パターンを確認できます。

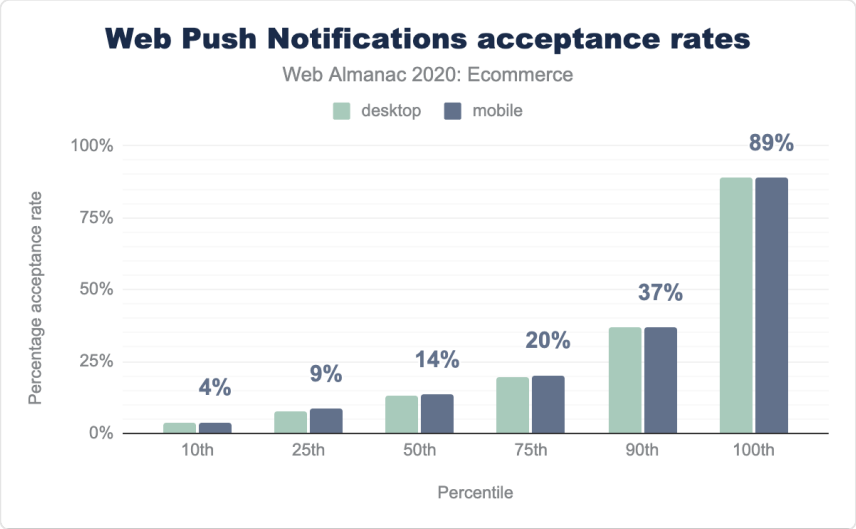


図16.25. Webプッシュ通知の受け入れ率

将来の分析機会

また、Playストアの `.well-known/assetlinks.json` やappストアの `.well-known/apple-app-site-association` のようなネイティブアプリ関連の標準を利用して、Eコマ

638. https://blog.chromium.org/2020/05/protecting-chrome-users-from-abusive.html
639. https://www.youtube.com/watch?v=J_t8c9HOjBc
640. https://www.youtube.com/watch?v=riKmez3sHaM

ースサイトがネイティブアプリを採用するかどうかも興味深いところです。GoogleはTrusted Web Activityを使ってPWAを簡単に実現できるようにしていますが、現在のところ、どれだけのサイトがこの手法を使ってplay storeにPWAを登録しているかについての統計は公開されていません。

今年のSEOの章では、`hreflang`と`lang`属性、`content-language`HTTPヘッダーを使用したウェブサイトの分析が含まれています。これに加えて、Global-e、Flow、Borderfreeなどの越境ECソリューションをWappalyzerにより検出することで、Eコマースサイトの越境ECの側面を見る機会が得られます。現在、Wappalyzerには、「越境EC」のための独立したカテゴリがなく、したがって、この種の分析はそのようなソリューションのリポジトリを自分たちで構築しない限り、不可能です。

Wappalyzerでは決済ソリューション（Apple Pay / PayPal / ShopPayなど）の検出も行っていますが、実装やソリューションの種類によってはホームページを見ただけでは検出できないこともありますが、ホームページを見ただけで検出できるソリューションについては、このような分析を行うことで前年比の傾向を見ることができます。

結論

Covid-19は、2020年のEコマースの成長を大幅に加速させ、多くの中小企業がオンラインプレゼンスを迅速に確立し、ロックダウン中も取引を継続する方法を見つけなければなりませんでした。WooCommerce、Shopify、Wix、BigCommerceなどのプラットフォームは、より多くの中小企業をオンライン化する上で非常に重要な役割を果たしました。Covid-19では、ブランドによるD2C（Direct to Consumer）の提供も開始され、これは今後も増加すると予想されます。Covid-19の影響は、今年のWeb Almanacでは完全には現れないかもしれません。というのも、これらの新規事業が分析に使用するCRUXデータセットの一部になるためには、まず一定のトラフィック閾値を超える必要があるからです。この理由により、来年の分析でも継続的な成長が見られるかもしれません。

また、Webプッシュ通知を利用しているマーケティングチームは、CRUXを利用して通知の統計情報を確認し、今後の通知の不正利用に巻き込まれないように注意してください。タグマネージャーは、マーケティングチームとエンジニアリングチームの間に多くの摩擦をもたらしているようです。Google Tag Managerのサーバーサイドタギングのようなソリューションは、ある程度浸透すると思われますが2021年に多くの変化があるとは思われず3〜5年はかかると思われますが、コミュニティはこのエコシステムをさらに進化させるために、それぞれのサードパーティに互換性のあるソリューションを提供するよう求める必要があるでしょう。

今回の分析はホームページのデータのみを対象としているという制限を忘れてはならないが、来年の分析では他にどのようなことを取り上げるべきか、コミュニティの皆様からのご意見をいただきたい。上記のセクションでは、さらなる分析の可能性をいくつか取り上げて

いますが、ご意見・ご感想をお待ちしています[641]。

著者

Rockey Nebhwani

@rnebhwani rockeynebhwani rockeynebhwani

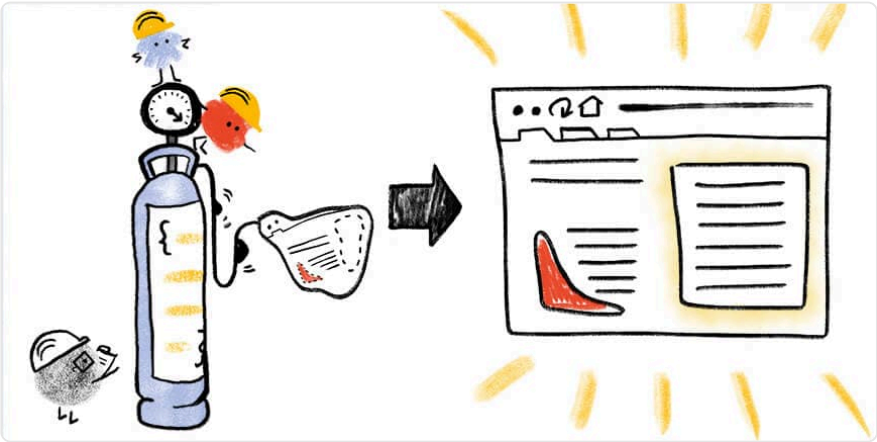
Rockey Nebhwaniは、2001年から小売・Eコマース業界で働いている独立系コンサルタントで、アメリカやイギリスのAmazon、Wal-Mart、Tesco、M&S、Safewayなどの小売業者との仕事で幅広い経験を持っています。ロッキーは、Eコマースのイベントで時折講演を行うほか、@rnebhwaniでツイートしています。

Jason Haralson

jrharalson

641. https://discuss.httparchive.org/t/2052

部 III 章 17
Jamstack



Ahmad Awais によって書かれた。
Maedah Batool と Nicolas Goutay によってレビュ。
Artem Denysov と Brian Rinaldi による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

Jamstackは比較的新しいコンセプトのアーキテクチャでウェブをより速く、より安全に、より簡単に拡張できるよう設計されています。開発者が大好きなツールやワークフローの多くをベースにしており、生産性を最大化します。

Jamstackの中核となる原則は、サイトページのプリレンダリングとフロントエンドとバックエンドの分離です。フロントエンドのコンテンツは、バックエンドとしてAPI（例えば、ヘッドレスCMS）を使用するCDNプロバイダー上で別々にホスティングされたものを配信するという考え方に依存しています。

HTTP Archive[642]は、毎月何百億ものページをクロールし、WebPageTest[643]のプライベートインスタンスを通して実行し、クロールされたすべてのページのキー情報を保存しています。

642. https://httparchive.org/
643. https://httparchive.org/reports/state-of-the-web#numUrls

これについての詳細はmethodology ページを参照してください。Jamstackの文脈では、HTTP Archiveは、Web全体のフレームワークやCDNの利用方法についての広範な情報を提供しています。この章では、これらの傾向の多くを集約して分析しています。

この章の目的は、Jamstackサイトの成長の推定と分析、人気のあるJamstackフレームワークのパフォーマンス、そしてCore Web Vitalsのメトリクスを使った実際のユーザー体験の分析です。

我々の分析は、Wappalyzerで簡単に識別できるJamstackによって制限されていることに注意してください。これは、Eleventy[644]のように、意図的に識別できないようにしている人気のあるJamstack[645]はデータに含まれていないことを意味します。理想的にはすべてのJamstacksを含むことになりますが、我々が持っている重要なデータを分析することにはまだ多くの価値があると考えています。

Jamstackの採用

この作業全体を通して、私たちの分析はデスクトップとモバイルのウェブサイトに注目しています。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からしかアクセスされていません。このため、データに小さな乖離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見ています。

644. https://github.com/11ty/eleventy/
645. https://x.com/eleven_ty/status/1334225624110608387?s=20

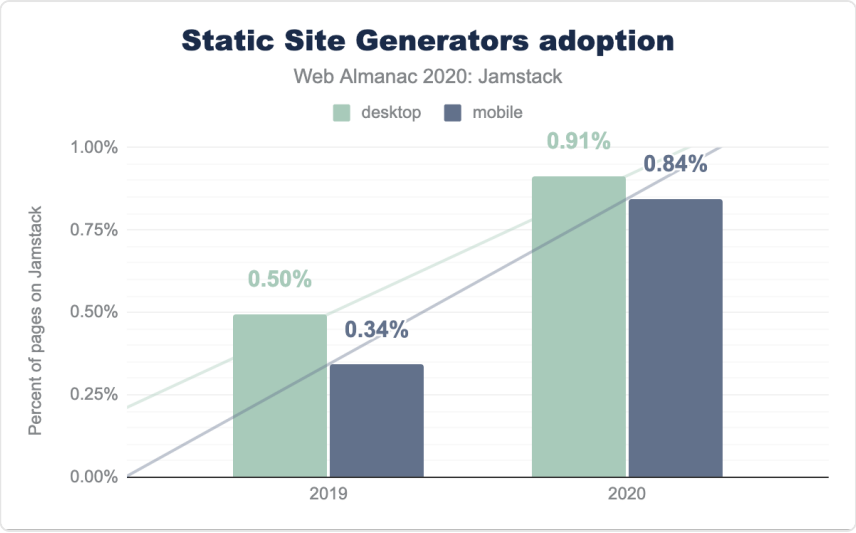


図17.1. Jamstackの採用動向。

ウェブページの約0.9%がJamstackを利用しており、内訳はデスクトップで0.91%と2019年の0.50%から、モバイルで0.84%と2019年の0.34%から上昇しています。

年	デスクトップ	モバイル
2019	0.50%	0.34%
2020	0.91%	0.85%
% Change	**85%**	**147%**

図17.2. Jamstackの採用統計。

Jamstackフレームワークを使ったデスクトップのウェブページの増加率は昨年より85%増加しています。モバイルでは147%とほぼ2.5倍の増加です。これは2019年からの大幅な伸びで、特にモバイルページの伸びが顕著です。これはJamstackコミュニティが着実に成長していることの表れだと考えています。

Jamstackフレームワーク

私たちの分析では、14個のJamstackフレームワークをカウントしました。1%以上のシェアを持っていたのは6つのフレームワークだけでした。Next.js、Nuxt.js Gatsby、Hugo、Jekyll

がJamstack市場シェアのトップを争っています。

2020年には、Jamstackのシェアのほとんどが上位5つのフレームワークに分散しているようです。興味深いことに、Next.jsの利用シェアは58.65%です。これは、次に人気のあるJamstackフレームワークであるNuxt.jsのシェア18.6%の3倍以上です。

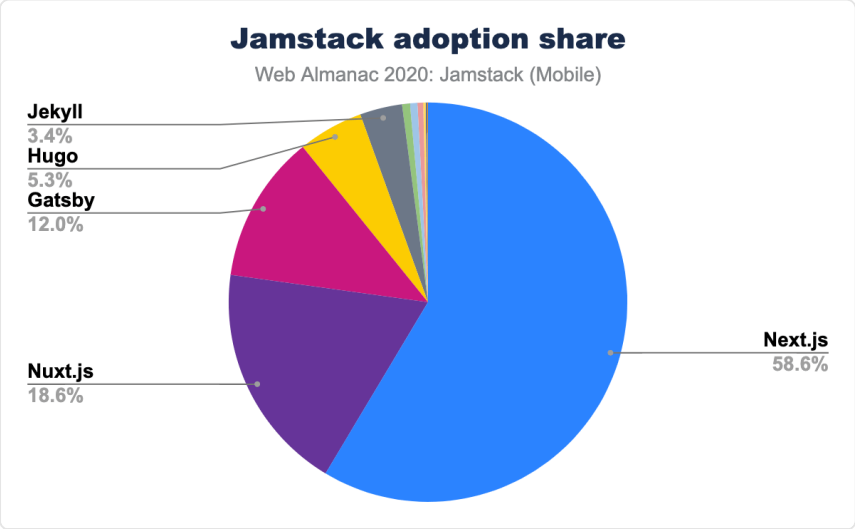


図17.3. Jamstackの採用シェア円グラフ2020年

フレームワーク採用の変更

前年比の成長を見ると、この1年でNext.jsは競合他社よりもリードを伸ばしていることがわかります。

Jamstack	2019	2020	% change
Next.js	47.89%	58.59%	22%
Nuxt.js	20.30%	18.59%	-8%
Gatsby	12.45%	11.99%	-4%
Hugo	9.50%	5.30%	-44%
Jekyll	6.22%	3.43%	-45%
Hexo	1.16%	0.64%	-45%
Docusaurus	1.26%	0.60%	-52%
Gridsome	0.19%	0.46%	140%
Octopress	0.61%	0.20%	-68%
Pelican	0.31%	0.11%	-64%
VuePress		0.05%	
Phenomic	0.10%	0.02%	-77%
Saber		0.01%	
Cecil	0.01%		-100%

図17.4. Jamstackフレームワークの相対的な採用率%

そして、上位5つのJamstacksに集中することで、Next.jsのリードをさらに示しています。

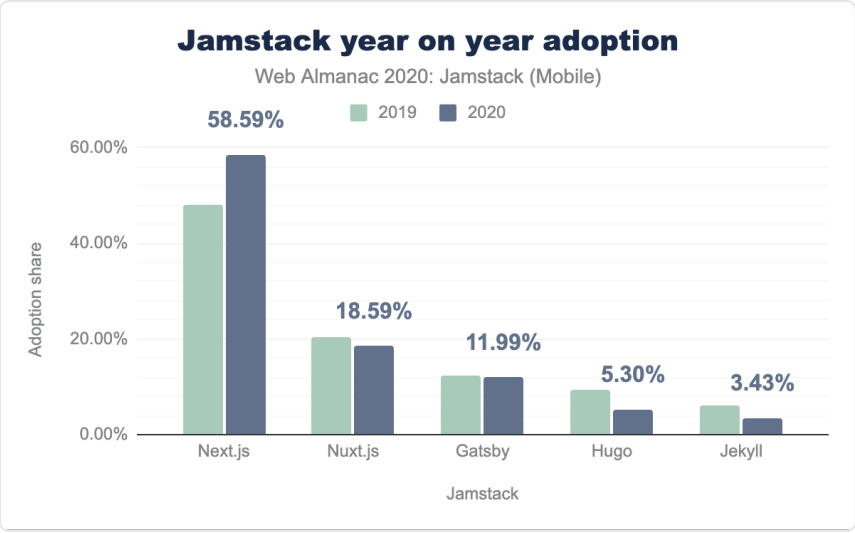


図17.5. Jamstackの採用シェア前年比

ここで注目すべきは、Next.jsとNuxt.jsのウェブサイトには、SSG（Static Site Generated）ページとSSR（Server-Side Rendered）ページの両方が混在しているという事実です。これは、これらを別々に測定する能力がないためです。つまり、分析には大部分または部分的にサーバーレンダリングされたサイトが含まれている可能性があり、従来のJamstackサイトの定義には当てはまらないことを意味します。それにもかかわらず、Next.jsのこのハイブリッドな性質が他のフレームワークと比較して競争上の優位性を与えており、それが人気を高めているように見えます。

環境への影響

今年はJamstackのサイトが環境に与える影響の理解を深めようとした。情報通信技術（ICT）産業は世界の炭素排出量の2%を占めており[646]、データセンターは世界の炭素排出量の0.3%を占めています。これは、ICT業界のカーボンフットプリントを航空業界の燃料からの排出量に相当する。

Jamstackはパフォーマンスに気を使っているとクレジットされることが多いです。次のセクションでは、Jamstackのウェブサイトの二酸化炭素排出量を見てみましょう。

646. https://www.nature.com/articles/d41586-018-06610-y

ページ重量

当社の調査では、Jamstackの平均ページ重量をKB単位で調べ、Carbon API[647]のロジックを使用してCO2排出量にマッピングしました。これにより、デスクトップとモバイルに分けて以下のような結果が得られました。

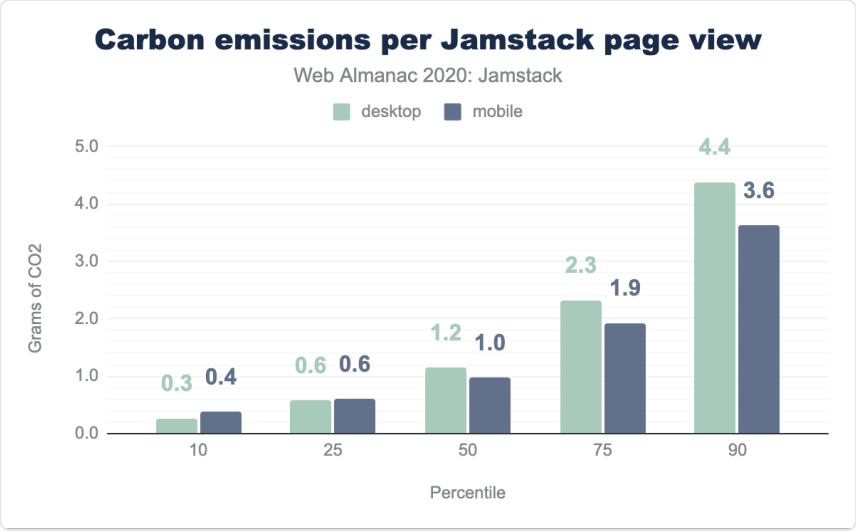


図17.6. Jamstackのページビューごとの炭素排出量です。

Jamstackのページロードの中央値は、デスクトップでは1.82MB、モバイルでは1.54MBのさまざまな資産を転送し、それぞれ1.2グラムと1.0グラムのCO2を排出していることがわかりました。Jamstackのウェブページの最も効率的なパーセンタイルでは、中央値よりも少なくとも3分の1のCO2排出量が少なくなりますが、Jamstackのウェブページの最も効率的でないパーセンタイルでは、逆に約4倍のCO2排出量になります。

ここで重要なのは、ページ重量です。平均的なデスクトップのJamstackのウェブページは、1.5MBのビデオ、画像、スクリプト、フォント、CSS、オーディオデータを読み込みます。しかし、10%のページでは4MB以上のデータが読み込まれます。モバイルデバイスでは、平均的なウェブページの読み込み量はデスクトップよりも0.28MB少なく、すべてのパーセンタイルで一貫しています。

画像フォーマット

人気の画像フォーマットはPNG、JPG、GIF、SVG、WebP、ICOです。これらの中では、ほ

647. https://gitlab.com/wholegrain/carbon-api-2-0/-/blob/master/includes/carbonapi.php#L342

とんどの状況でWebPが最も効率的[648]で、WebPの損失のない画像は同等のPNGよりも26%小さく[649]、同等のJPGよりも25-34%小さく[650]なっています。しかし、WebPはすべてのJamstackページで2番目に人気のない画像フォーマットですが、モバイルとデスクトップではPNGが最も人気があります。一方、JPGはわずかに人気がないのに対し、GIFはJamstackサイトで使用されている画像の20%近くを占めています。興味深いのはSVGで、モバイルサイトではデスクトップサイトの約2倍の人気があります。

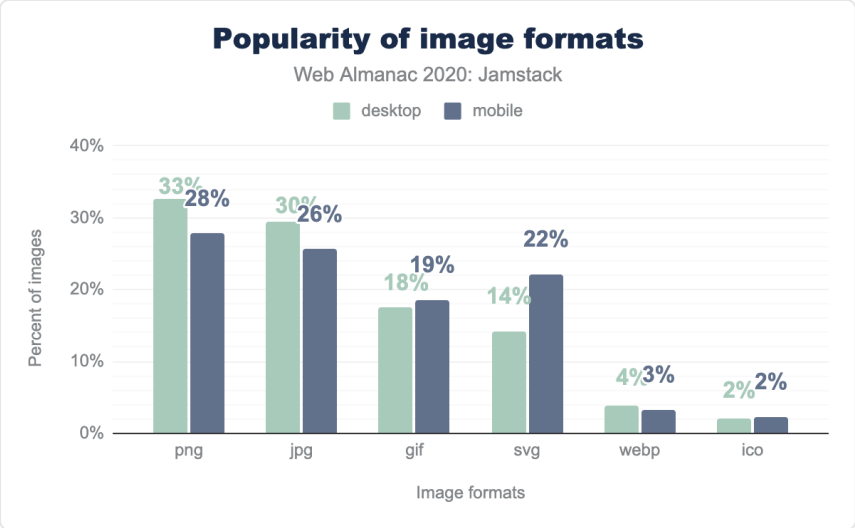


図17.7. 画像フォーマットの普及。

サードパーティのバイト

Jamstackサイトは、ほとんどのウェブサイトと同様に、外部画像、ビデオ、スクリプト、スタイルシートなどのサードパーティリソースを読み込むことがよくあります。

648. *https://developers.google.com/speed/webp/*
649. *https://developers.google.com/speed/webp/docs/webp_lossless_alpha_study#results*
650. *https://developers.google.com/speed/webp/docs/webp_study*

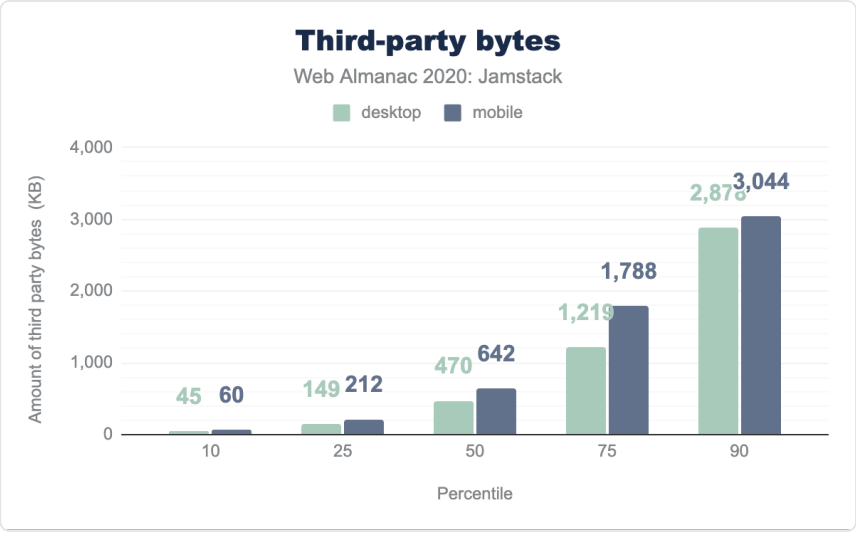


図17.8. サードパーティのバイト。

デスクトップのJamstackページの中央値では、サードパーティからのリクエストが26件、470 KBのコンテンツがあり、モバイルでは38件、642 KBのコンテンツが発生していることがわかりました。一方、デスクトップサイトの10%のサイトでは、2.88MBのコンテンツで114件のリクエストがありますが、モバイルでは3MBのコンテンツで148件のリクエストがあります。

ユーザー体験

JamstackのWebサイトは、良いユーザー体験を提供するとよく言われます。フロントエンドとバックエンドを分離し、CDNエッジでホスティングするというコンセプトは、まさにその通りです。最近立ち上げたCore Web Vitals[651]を使って、JamstackのWebサイトを利用したときの実際のユーザー体験に光を当てることを目的としています。

Core Web Vitalsは、ユーザーがJamstackのページをどのように体験しているのかを理解するための3つの重要な要素です。

- 最大のコンテンツフルペイント(LCP)
- 最初の入力までの遅延(FID)

651. https://web.dev/i18n/ja/learn-web-vitals/

- 累積レイアウト変更(CLS)

これらのメトリクスは、優れたウェブ・ユーザー・体験を示すコア要素をカバーすることを目的としています。ここでは、Jamstackフレームワークのトップ5のコア・ウェブ・バイタルの統計を見てみましょう。

最大のコンテンツフルペイント

最大のコンテンツフルペイント(LCP)は、ページのメインコンテンツが読み込まれた可能性が高く、ユーザーにとって有用なページであるかどうかを測定します。これは、ビューポート内に表示されてい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によって行われます。

これは、ページの読み込みからテキストや画像などのコンテンツが最初に表示されるまでを計測する最初の入力までの遅延(FID)とは異なります。LCPは、ページのメインコンテンツが読み込まれたときに測定する良い代理人と考えられています。

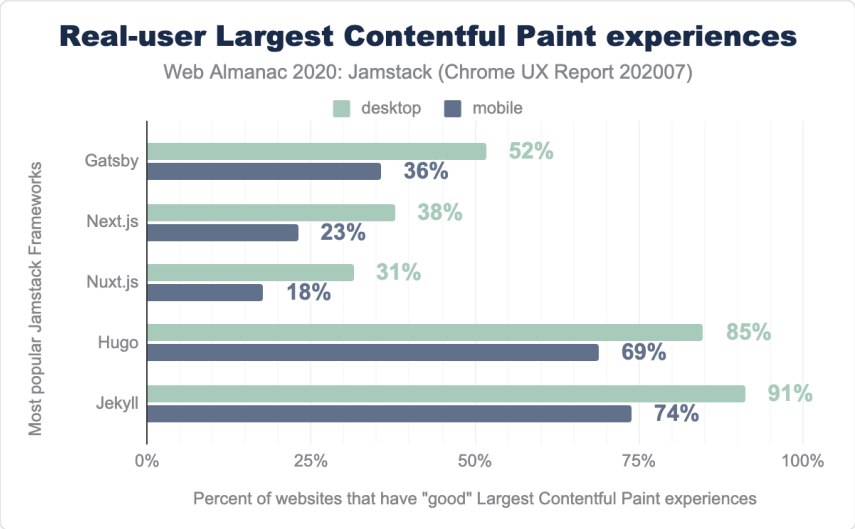


図17.9. リアルユーザー最大のコンテントフルペイント体験談

「良い」LCPは2.5秒以下とされています。JekyllとHugoのLCPスコアはいずれも50%を超えており、デスクトップではJekyllが91%、Hugoが85%と、印象的なスコアとなっています。Gatsby、Next.js、Nuxt.jsのサイトは遅れており、デスクトップではそれぞれ52%、38%、31%、モバイルでは36%、23%、18%となっています。

これは、主に動的な部分が少ない、あるいは全くない静的なコンテンツサイトを作るために

使われるHugoやJekyllに比べて、GatsbyやNext.js、Nuxt.jsで構築されたサイトのほとんどが複雑なレイアウトと高いページウェイトを持っているという事実に起因しているかもしれません。とりあえず、HugoやJekyllでReactやVueJSなどのJavaScriptフレームワークを使う必要はありません。

上のセクションで説明したように、ページの重みが高いと環境に影響を与える可能性があります。しかし、これはLCPのパフォーマンスにも影響し、Jamstackフレームワークによって非常に良いか、一般的には悪いかのどちらかになります。これは実際のユーザー体験にも影響を与える可能性があります。

最初の入力までの遅延

最初の入力までの遅延(FID)は、ユーザーが最初にサイトとやり取りをした時（リンクをクリックした時、ボタンをタップした時、カスタムJavaScriptを使用したコントロールを使用した時など）から、ブラウザが実際にそのやり取りに反応するまでの時間を測定します。

ユーザーの視点から見た「速い」FIDは、停滞した体験よりも、サイト上での行動から即時のフィードバックを提供します。この遅延は問題点であり、ユーザーがサイトを操作しようとしたときに、サイトの読み込みの他の側面からの干渉と相関する可能性があります。

FIDは、デスクトップの平均的なJamstackのウェブサイトでは非常に速く、最も人気のあるフレームワークでは100%のスコアを獲得し、モバイルでは80%以上のスコアを獲得しています。

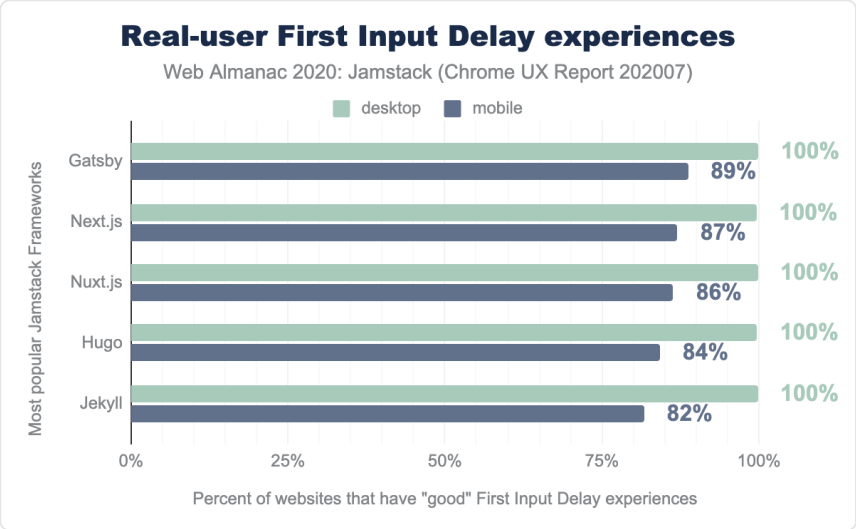


図17.10. リアルユーザーのファースト入力遅延体験。

ウェブサイトのデスクトップ版とモバイル版に出荷されるリソースの間には、わずかな差があります。ここではFIDのスコアは概ね非常に良いのですが、これが似たようなLCPのスコアに翻訳されていないのは興味深いところです。示唆されているように、Jamstackサイトの個々のページの重みに加えて、モバイル接続の品質が、ここで見られるパフォーマンスのギャップに一役買っている可能性があります。

累積レイアウト変更

累積レイアウト変更（CLS）は、ユーザー入力の最初の500ms以内のウェブページ上のコンテンツの不安定性を測定します。CLSは、ユーザーが入力した後に起こるレイアウトの変化を測定します。これは特にモバイルでは重要です。ユーザーがアクションを起こしたい場所（検索バーなど）をタップしても、追加の画像や広告などが読み込まれると場所が移動してしまう場合があります。

0.1点以下が良い、0.25点以上が悪い、その間は何をしても改善が必要。

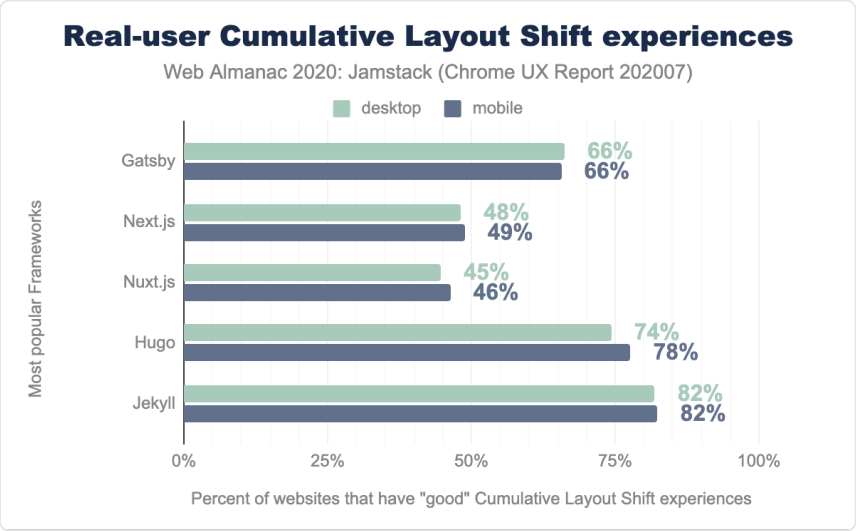


図17.11. リアルユーザーの累積レイアウト変更の体験

Jamstackフレームワークのトップ5は、ここでOKを出しています。トップ5のJamstackフレームワークで読み込まれたウェブページの約65%が「良い」CLS体験を持っており、モバイルでは82%まで上昇しています。デスクトップとモバイルの平均スコアは65%です。Next.jsとNuxt.jsはともに50%を下回っており、ここに課題があります。開発者を教育し、悪いCLSスコアを回避する方法を文書化することは、長い道のりを歩むことになります。

結論

Jamstackは、コンセプトとしてもスタックとしても、この1年で重要性を増しました。統計によると、2019年の約2倍のJamstackサイトが存在しています。開発者は、フロントエンドとバックエンド（ヘッドレスCMS、サーバーレス機能、またはサードパーティのサービス）を分離することで、より良い開発体験を楽しむことができます。しかし、Jamstackサイトを閲覧するリアルユーザーの体験はどうでしょうか？

Jamstackの採用状況、Jamstackフレームワークで作成されたWebサイトのユーザーエクスペリエンスをレビューし、初めてJamstackが環境に与える影響を見てみました。ここでは多くの疑問に答えてきましたが、さらなる疑問は残しておきます。

Eleventyのようなフレームワークもありますが、そのようなフレームワークの利用状況を把握するパターンがないため、今回のデータに影響を与えています。Next.jsはスタティックサイト生成とサーバーサイドレンダリングの両方を提供していますが、スタティックサイト生成もインクリメンタルに提供しているため、このデータで両者を分離することはほぼ不可能です。この章をベースにした更なる研究を歓迎します。

さらに、Jamstackコミュニティが注意を払う必要がある分野をいくつか取り上げました。2021年のレポートで共有できるような進展があることを願っています。さまざまなJamstackフレームワークは、コア・ウェブ・バイタルを見ることで、リアル・ユーザー体験を向上させる方法を文書化できます。

JamstackサイトをホストするCDNの1つであるVercelは、リアルユーザー体験スコア[652]と呼ばれる分析サービスを構築しました。Lighthouse[653]のような他のパフォーマンス測定ツールが、ラボでシミュレーションを実行してユーザーの体験を推定するのに対し、Vercelのリアル体験スコアは、アプリケーションの実際のユーザーのデバイスから収集した実際のデータポイントを使用して計算されます。

ここで注目すべきは、Next.jsのLCPスコアが低かったため、VercelがNext.jsを作成し、維持管理していることでしょう。今回の新しい提供は、来年にはLCPスコアが大幅に改善されることを意味しています。これは、ユーザーや開発者にとって非常に有益な情報となるでしょう。

Jamstackフレームワークは、サイト構築の開発者体験を向上させています。Jamstackのサイトを閲覧する際のリアルユーザー体験を向上させるために、今後も継続的な改善を目指していきましょう。

652. *https://vercel.com/docs/analytics#real-experience-score*
653. *https://web.dev/measure/*

著者

Ahmad Awais

@MrAhmadAwais ahmadawais https://AhmadAwais.com

Ahmad Awaisは、受賞歴のあるオープンソースエンジニアで、Google Developers Expert Dev Advocate、Node.js Community Committee Outreach Lead、WordPress Core Dev、WGAのエンジニアリングDevRelのVPです。彼は世界中の何百万人もの開発者に利用されている様々なオープンソースのソフトウェアツールを執筆しています。彼のShades of Purple[654]のコードテーマやcorona-cliのようなプロジェクトのように。Awaisは教えることが大好きです。2万人以上の開発者が彼のコース[655]、つまりNode CLI[656]、VSCode.pro[657]、Next.js Beginner[658]から学んでいます。Awaisは、12人のGitHub Stars[659]としてFOSSコミュニティのリーダーシップを評価されました。彼はSmashing Magazine Experts Panelのメンバーであり、CSS-Tricks, Tuts+, Scotch.io, SitePointで特集＆出版された著者です。#OneDevMinute[660] 開発者のヒントをツイートしています。

654. https://shadesofpurple.pro/more
655. https://AhmadAwais.com/courses/
656. https://nodecli.com/
657. https://vscode.pro/
658. https://nextjsbeginner.com/
659. https://ahmadawais.com/github-stars/
660. https://awais.dev/odmt

部 IV 章 18
ページの重さ

Henri Helvetica によって書かれた。
Paul Calvano によってレビュ と による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

ページの重さは、利用可能な最も単純な測定基準の1つです。あなたの個人的な体重の感覚を得るため人間の体重計にのるように（まあ実際には質量ですが、あなたはそれを得ることができます）、ページをロードすると収集され要求されたリソースの数とサイズの感覚を提供します。しかしウェブとウェブページが成熟し成長してきたように、ページの重さのような関連するメトリクスも増えてきました。それは、個人の体重（質量）が同じことをできるのと同じように、ページのパフォーマンスに影響を与えることができます。この章では、ウェブページの層を深く掘り下げて、エンドユーザー（あなた、私、私たち）の不利益になる可能性のあるページの重量を構成するものが何であるかを見ていきます。

#PageWeightStillMatters

#PageWeightStillMattersは、それが重要ではなかった、あるいは今までに重要ではなかったということをほぼ暗示しています。テキストベースのCraigslistが登場したときには問題にな

らなかったかもしれません。しかし、それが設立された25年前、Mosaic1.0も同じ年に発売され、TLCのWaterfallsがトップヒットしました。ウェブはリソースと同様に成熟していきました。ちょうど数年前のことですが、twitterの世界では、ウェブページの平均サイズが今では オリジナルの運命[661] のサイズに匹敵するかどうかが議論されていました。私たちの多くは、私たち自身のTammy Everts[662]を含めて、ページのサイズが時間の経過とともにどのようなものになるかについて考えていましたが、現実は驚くべきものでした。ページのサイズは、デスクトップ/モバイルでそれぞれ75パーセンタイルで、~4MBと3.7MB、90パーセンタイルでは7.4MBと6.7MBという衝撃的なサイズになっています。このような重いページを持つことには、信頼性の低いネットワークによるユーザー体験の低下の可能性など、様々な意味合いがあります。今日では、10年前に学んだ教訓[663] にもかかわらず、同じ課題のバリエーションを経験しています：わずかに優れたネットワークを持っているにもかかわらず、はるかに大きなリソースを使って作業しています。

帯域幅

2016年、私が話をしたオーストラリアの観光客が英国のインターネットに満足している理由を説明するように求められたとき、GoogleのIlya Grigorikには[2つの言葉がありました]（https://youtu.be/x4S38hpgxuM?t=89）：物理学はそれを酷評しました！（おっと、それは3つです）。

要点は単純でした。帯域幅を増やすことでメリットが得られるかもしれませんが、それでも物理法則が優先されます。 オーストラリア人は遅延の法則から逃れることができません。最良のシナリオでは、シドニーの自宅で、このオーストラリア人はインターネットが応答しないと認識されるほどの遅延を経験していました。

さて同じオーストラリア人が、75パーセンタイルで、彼のページが約108リクエストを行っていることを知っていると想像してみてください（これについては後で詳しく説明します）。最新のリクエストの寿命についての詳細はHTTP/2と圧縮の章を参照してください。

資産

現代のブラウジングの25年間で、資産とリソースは量意外ほとんど変わっていません。HTTP archiveの手口は「Webがどのように構築されたか」であり、それは主にHTML、CSS、JavaScript、そして最後に画像で行われました。

1995年以前は、ウェブのページの重さはほとんど予測可能で管理可能でした。しかしHTML2.0を導入したRFC 1866[664]では、`<img>`要素を介してインライン画像を導入することで、ページの重さは劇的に増加します、すべてはウェブ開発のためになるでしょう（画像の

661. *https://www.wired.com/2016/04/average-webpage-now-size-original-doom/*
662. *https://speedcurve.com/blog/web-performance-page-bloat/*
663. *https://blog.chriszacharias.com/page-weight-matters*
664. *https://tools.ietf.org/html/rfc1866*

追加はポジティブな実験とみなされていました）。

ほとんどの場合、画像がページの重さの大部分を占めるというのが経験則でした。確かに、インライン画像がウェブに追加された当時はそうでしたし、現在もそうです。別のシナリオでは、画像データがページ重量の最大のソースとなるため、ページ重量の節約の最大のソースにもなります（これについては後述します）。これは、画像を適切なサイズにするだけでなく、画像が最適化のスイートスポットにあることを確認することで達成されます - 品質とファイルサイズの最適なバランスを見つけることです。

JavaScriptは平均的にページ上で2番目に豊富なリソースですが、バンドル、圧縮、ミニ化などのファイルタイプを扱う機会が多くなりがちです。

複雑で対話的

平凡で教育に近いプラットフォームから、標準となった革新的で複雑で高度に対話的なアプリへのWebの旅は、より大きなストーリーを隠しました。リソースのラタトゥイユは、それぞれが最新のメトリックに影響を与え、次にユーザー体験にも影響を与えます。

私たちが対話性について話すとき、私たちはほとんどもっぱらJavaScriptについて話しています。ここでは対話性について深く議論するわけではありませんが、JavaScriptのコンテンツと実行に焦点を当て、依存するメトリクスがあることは知っています。ですから、JavaScriptが重くなればなるほど、対話性のメトリクス（対話までの時間、総ブロッキング時間）に大きな影響を与える可能性が高くなります。私たちは、JavaScriptの章でもう少し掘り下げています。

分析

統計結果を投稿して解析しているため、データは転送サイズに基づいていることが多いです。しかし、今回の分析では可能な限り解凍したサイズを採用しています。

ページ重量

デスクトップとモバイルの両方で、従来のページの重さを見てみましょう。デルタのほとんどはモバイルで転送されるリソースが少なくなったこと、メディア管理のピンチによるものですが、中央値では、2つのクライアント間の差はそれほど大きくないことがわかります。

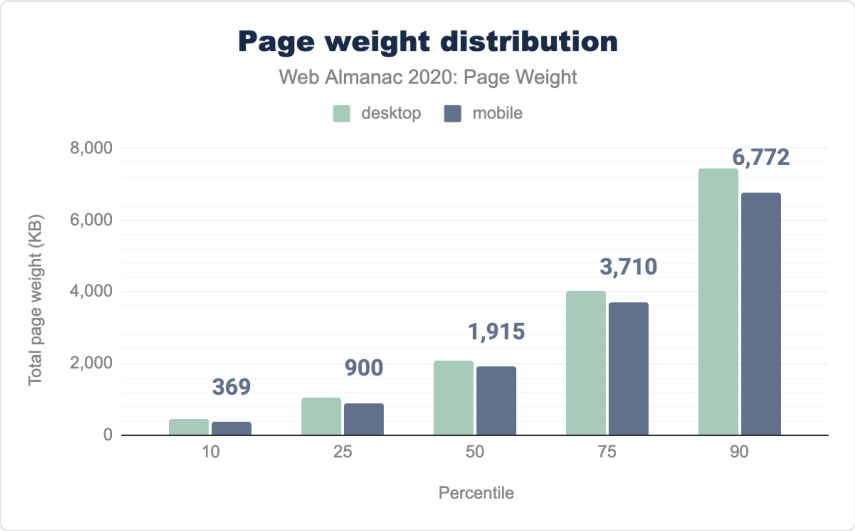


図18.1. 1ページあたりの総バイト数の分布。

しかし、このことから次のことが推測できます：モバイルでは7MB、デスクトップでは7.5MBのページ重量が90パーセンタイルに近づいています。このデータは昔からのトレンドを踏襲しています：ページ重量の伸びは前年に比べて再び上昇傾向にあります。

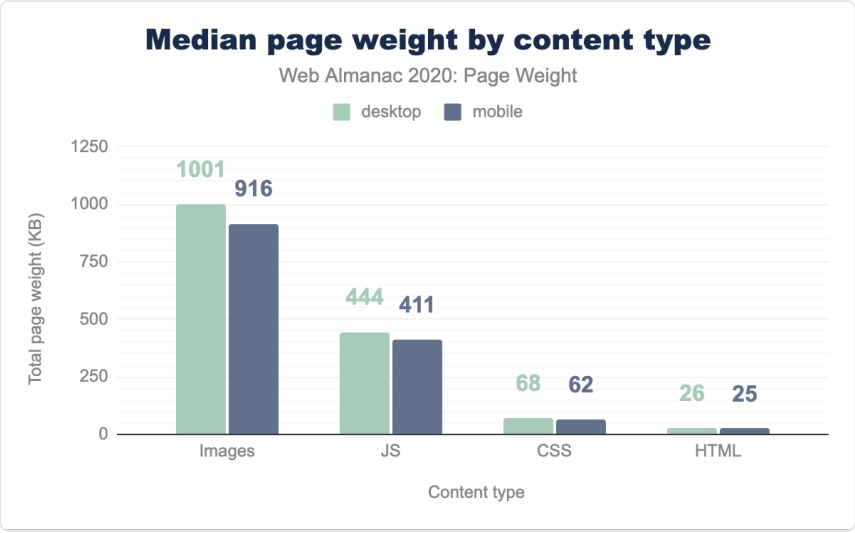


図18.2. コンテンツの種類別の1ページあたりのバイト数の中央値。

ボンネットを開けると、各リソースの中央値と平均値がどのように見えるかがわかります。ここでも1つ残っています。画像が主要なリソースであり、JavaScriptが2番目に多いですが、はるかに2番目に多いです。

リクエスト

我々は古い格言を持っています：最も迅速な要求は、決して行われないものである。あえて言うならば、最小のリソースは一度もリクエストされないものであるということです。リクエストレベルでは、多くのことが同じです。最も重みのあるリソースが最も多くのリクエストをしています。

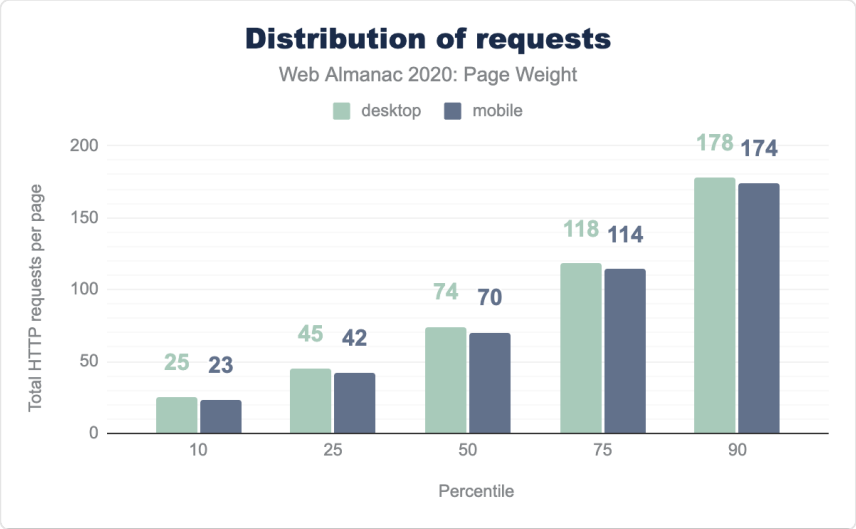


図18.3. 1ページあたりのリクエストの分布。

リクエスト分布を見ると、デスクトップとモバイルの差はそれほど大きくなく、デスクトップがリードしていることがわかります。注目すべき点は、この時点でのデスクトップのリクエストの中央値は同じ昨年と同じ[665](74)ですが、ページの重さは上昇しています(+122kb)。単純な観察ですが、長年にわたって見てきた軌跡を裏付けるものです。

665. *https://almanac.httparchive.org/ja/2019/page-weight#page-requests*

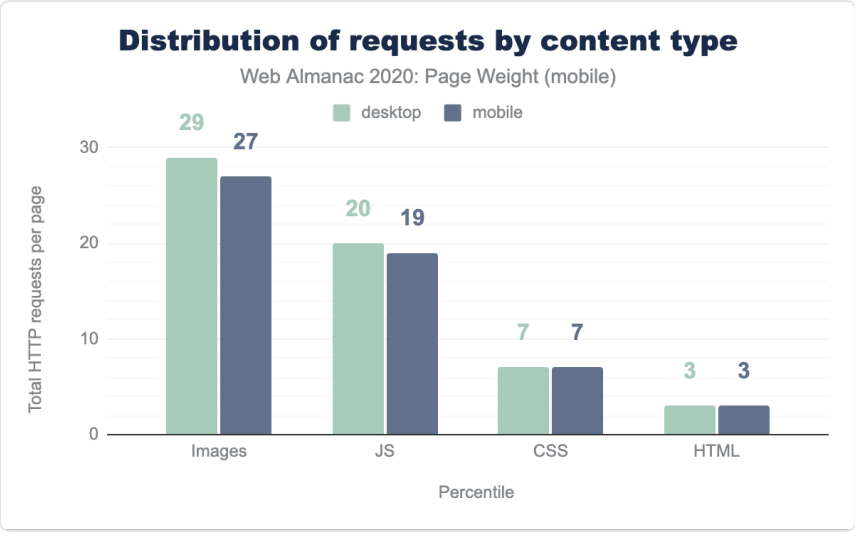


図18.4. コンテンツタイプ別のモバイルページあたりのリクエスト数の中央値

画像はまたしてもリクエスト数が一番多いです。JavaScriptは、去年の間にわずかにその差は縮まってきています。

ファイル形式

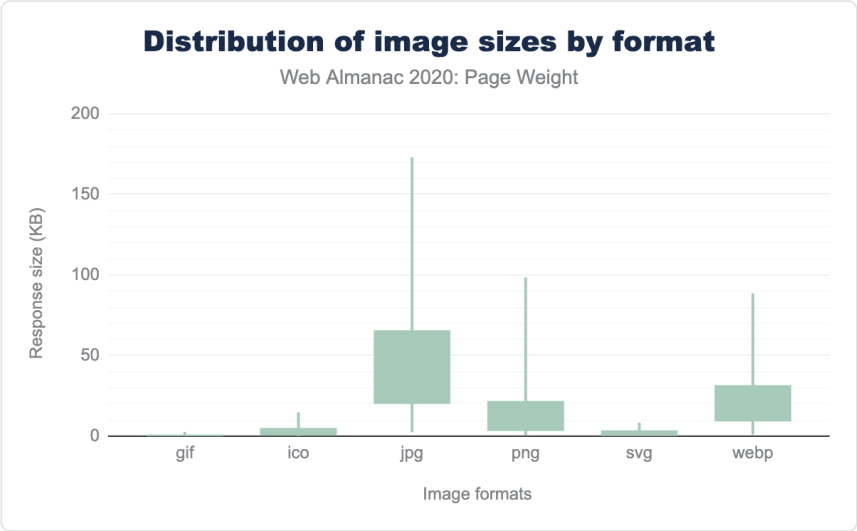


図18.5. フォーマット別の画像サイズの分布。

画像がページの重さの大きな情報源であることを知っています。上の図は、画像の重さのトップソースと重さの分布を示しています。トップ3。JPG、PNG、WebP。つまり、JPGは最も人気のある画像フォーマットであるだけでなく、サイズ的にも最も大きい傾向があります。しかし、私たちが昨年指摘したように[666]、それはPNGの主な使用例であるアイコンやロゴに関係しています。

666. https://almanac.httparchive.org/ja/2019/page-weight#file-size-by-image-format-for-images--1024-bytes

画像バイト数

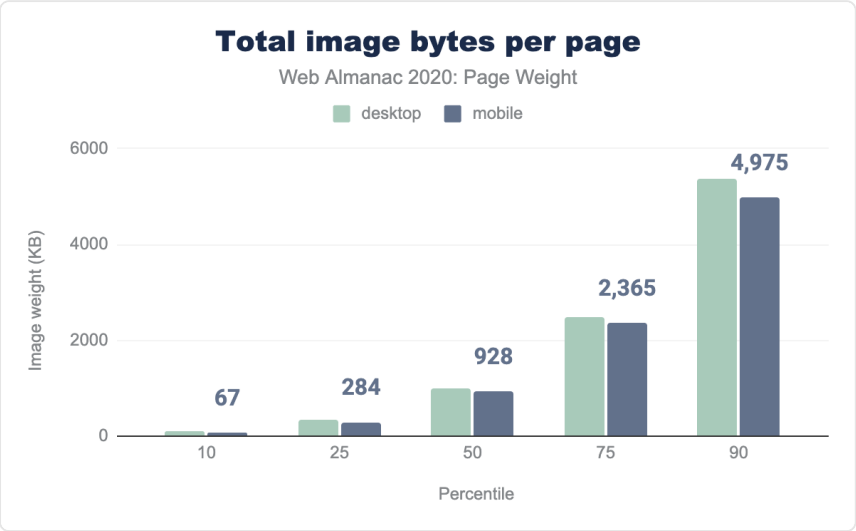


図18.6. 1ページあたりの画像レスポンスサイズの分布。

総画像バイト数を見てみると、ページ全体の重量で前述したように、同じように上向きの傾向が見られます。

コロナウィルス

2020年はインターネットの歴史の中で最も需要の高い年となった。これは、世界中の通信会社[667]による自己申告に基づいています。YouTube、Netflix、ゲーム機メーカー、その他多くの企業が、コロナウィルスの予測される帯域幅需要と自宅待機命令のために、自分たちのネットワークのスロットル[668]を要求されました。今では、ネットワークに需要を生み出す新たな容疑者が出てきています。この危機の中でいくつかの政府機関は、サイトのすべての側面を最適化し、再設計または更新するために前進しています。ca.gov[669] (link[670])とgov.uk[671]がその例です。これらの時代では、コロナウィルスはインターネットが不可欠なサービスとして認定されており、重要な命を救う情報にアクセスすることができるようにデータの規律配信を介して管理可能なページの重量が含まれている可能な限り摩擦がないようにする必要があります。

667. *https://www2.telegeography.com/network-impact*
668. *https://www.bloomberg.com/news/articles/2020-03-19/netflix-to-cut-streaming-traffic-in-europe-to-relieve-networks*
669. *https://ca.gov*
670. *https://news.alpha.ca.gov/prioritizing-users-in-a-crisis-building-covid19-ca-gov/*
671. *https://gov.uk*

インターネットに嫁いできた私たちは、コロナウィルスによって、誓いを新たにせざるを得なくなりました。インターネット上でできる限り効率的にコンテンツを配信するためには、常にページの重さを最優先にしなければなりません。

そう遠くない未来

私たちは25年間、ページの重量が着実に成長するのを見てきました。それは最大の株式投資の一つであったかもしれません - それが一つであったならば。しかし、これはウェブであり、我々はデータ、リクエスト、ファイルサイズ、そして最終的にはページの重量を管理しようとしています。

画像が最大の重みの源であることがわかるように、データをまとめたところです。 これは、それが私たちの最大の節約の源でもあることを意味します。 2020年は極めて重要な年であり、WebデータのHTTP Archive追跡の変曲点となる可能性があります。 2020年は、最新の形式のWebPがついにSafariに採用され、この形式が最終的にすべてのブラウザーでサポートされるようになった年でした。 これは、このフォーマットがフォールバックをほとんどまたはまったく伴わずに快適に使用できることを意味します。 最も重要なポイントは？　ページの大幅な軽量化の可能性はそこにあります。30%可能なです。

さらに興味深いのは、より現代的なフォーマットであるavifのアイデアです。このフォーマットは、今日ではブラウザの市場シェアの約70%に十分対応しており、WebPよりもさらに小さな画像ファイルサイズのシナリオを生み出しています。そして最後に、おそらく最も遠い存在であると思われるメディアクエリレベル5、`prefers-reduced-data`です。非常に初期の草案ではありますが、このメディア機能はデータに敏感な状況でユーザーがバリアントリソースを好むかどうかを検出するために使用され、すでにブラウザで利用可能になり始めています[672]。

水晶の玉を見てみると、Web Almanacの第3弾とページの重さの章は、2021年にはかなり違った姿になっているかもしれません。イメージへの大きな技術的・工学的投資は、最終的に私たちが探し求めていた減少するリターンを提供してくれるかもしれません。

結論

Webページが一般的に成長し続けているのは当然のことです。私たちはより豊かな体験、より魅力的なインタラクティブ性、よりパワフルな画像による魅力的なビジュアルを実現するためにより多くのリソースを投入してきました。私たちは、データの超過やユーザーエクスペリエンスを犠牲にして、これらのアプリケーションを作成してきました。しかし私たちが前進し、予想もしていなかった場所にウェブを押し進めていく中で、先に述べたように私た

672. *https://caniuse.com/mdn-css_at-rules_media_prefers-reduced-data*

ちはエンジニアリングにおいてもさらなる進歩を遂げています。最新のラスター画像フォーマットがより多く採用されるようになり、JavaScriptの管理がより効率的になり、ユーザーが求める規律を持ってデータを配信できるようになると早ければ来年にはページの重さの低下が見られるようになるかもしれません。

著者

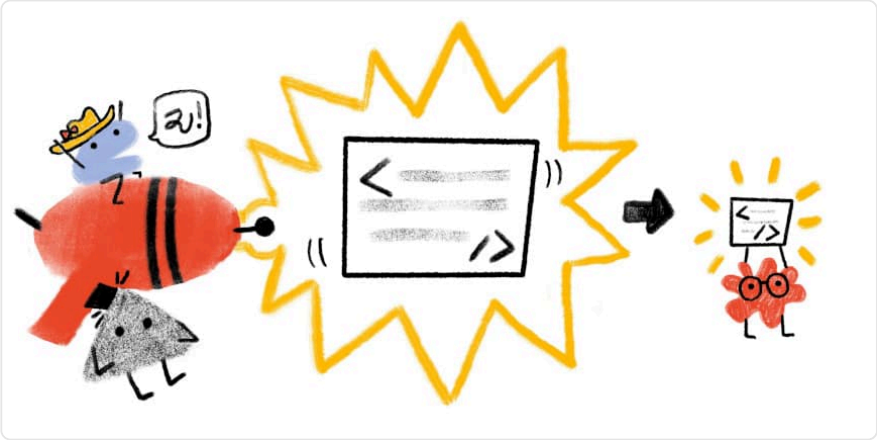
Henri Helvetica

@HenriHelvetica henrihelvetica

Henriはフリーランスの開発者であり、興味をパフォーマンスエンジニアリングとユーザーエクスペリエンスを組み合わせたポプリに変えています。日々の研究ドキュメントやケーススタディの洪水を読んだり、開発ツールのサイトを無差別に監査したりしていないときは、コミュニティに貢献したり、Toronto Web Performance Group[673]を含む共同プログラミングのミートアップに参加したり、様々なブートキャンプでランチをしたり学んだりするために自分の時間をボランティアで提供したりしています。それ以外の時は、音楽制作ソフトを使ってツーリングをしたり、ほぼ確実にトレーニングをして、可能な限り最速で5kmを走ることに集中しています。

673. *https://x.com/towebperf*

部 IV 章 19
圧縮



Moritz Firsching、 Luca Versari、 Sami Boukortt と Jyrki Alakuijala によって書かれた。
Paul Calvano によってレビュ。
Abby Tsai による分析。
Shane Exterkamp 編集。
Sakae Kotaro によって翻訳された。

序章

HTTP圧縮を使用すると、ウェブサイトの読み込みが速くなるため、より良いユーザー体験が保証されます。HTTP圧縮を行わない場合は、ユーザー体験が低下し、関連するウェブサービスの成長率に影響を与え、検索ランキングにも影響を与えます。 圧縮を効果的に使用することで、ページウェイトを減らし、ウェブパフォーマンスを向上させることができるため、検索エンジン最適化の重要な部分となります。

画像やその他のmediaタイプでは非可逆圧縮を許容することが多いのですが、テキストの場合は可逆圧縮、つまり解凍後に正確なテキストを復元したいと考えています。

どのようなコンテンツを圧縮すべきか？

HTML、CSS、JavaScript、JSON、SVGなどのほとんどのテキストベースのアセットや、

woff、ttf、icoなどの特定の非テキストフォーマットについては、圧縮を使用することをオススメします。

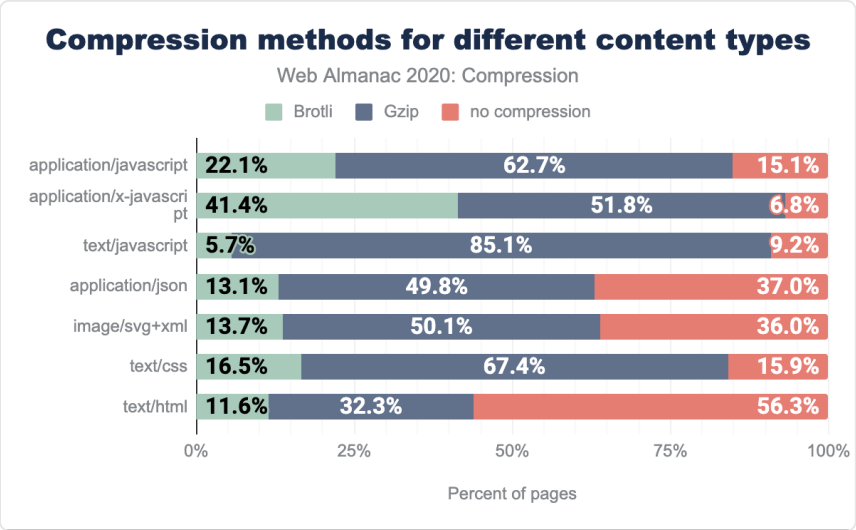


図19.1. コンテンツタイプ別の圧縮方法

この図は、あるコンテンツタイプのレスポンスのうち、Brotli、Gzip、またはテキスト圧縮なしのいずれかを使用している割合を示しています。 驚くべきことに、すべてのコンテンツタイプで圧縮が有効であるにもかかわらず、その割合の範囲はコンテンツタイプごとに大きく異なっています。たとえば、`text/html`で圧縮を使用しているのはわずか44%であるのに対し、`application/x-javascript`では93%です。

画像ベースの資産では、テキストベースの圧縮はあまり役に立たず、広く採用されていません。データによると、BrotliやGzipを採用している画像レスポンスの割合は非常に低く、4%未満です。テキストベース以外のアセットの詳細については、メディアの章を参照してください。

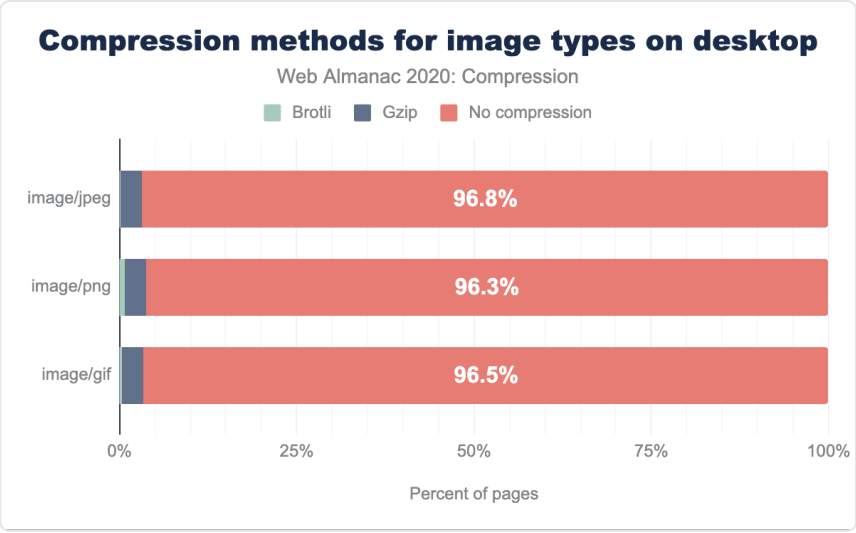


図19.2. デスクトップ上の画像タイプの圧縮方法

HTTP圧縮はどのように使うのですか？

たとえば、HTMLMinifier[674]、CSSNano[675]、UglifyJS[676]などの最小化ツールを使用できます。しかし、圧縮機能を使うことで、より大きな効果が期待できます。

サーバー側で圧縮を行うには2つの方法があります。

- 事前圧縮（事前にアセットを圧縮して保存しておくこと）
- 動的圧縮（リクエストがあったときに、その場でアセットを圧縮する）

事前に圧縮されているので、アセットの圧縮に時間をかけることができます。一方、動的に圧縮されたリソースの場合は、非圧縮ファイルと圧縮ファイルの送信時間の差よりも圧縮にかかる時間が短くなるように圧縮レベルを選択する必要があります。この違いは、両方式の推奨圧縮レベルを見るとよくわかります。

674. https://github.com/kangax/html-minifier
675. https://github.com/ben-eb/cssnano
676. https://github.com/mishoo/UglifyJS2

	Brotli	Gzip
事前圧縮	11	9かZopfli
動的圧縮	5	6

図19.3. 使用する推奨圧縮レベル

現在、実質的にすべてのテキスト圧縮は、2つのHTTPコンテンツエンコーディングのいずれかによって行われています。Gzip[677]とBrotli[678]です。どちらもブラウザで広くサポートされています。Brotliを使用できます[679]/Gzipを使用できます[680]

Gzipを使用したい場合は、より小さいGzip互換ファイルを生成するZopfli[681]の使用を検討してください。これは、とくに圧縮済みのリソースに対して行うべきです。なぜなら、ここでは最大の利益が期待できるからです[682]。Gzipの異なる圧縮レベルを考慮したGzipとZopfliの比較[683]を参照してください。

多くの一般的なサーバ[684]は、動的および/または事前に圧縮されたHTTPをサポートしており、その多くがBrotli[685]をサポートしています。

HTTP圧縮の現状

HTTPレスポンスの約60%は、テキストベースの圧縮を行わずに配信されています。これは驚くべき統計のように思われるかもしれませんが、データセット内のすべてのHTTPレスポンスに基づいていることを覚えておいてください。図19.2に示すように、画像などの一部のコンテンツは、これらの圧縮アルゴリズムの恩恵を受けないため、あまり使用されていません。

677. https://tools.ietf.org/html/rfc1952
678. https://github.com/google/brotli
679. https://caniuse.com/?search=brotli
680. https://caniuse.com/?search=gzip
681. https://ja.wikipedia.org/wiki/Zopfli
682. https://cran.r-project.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf
683. https://blog.codinghorror.com/zopfli-optimization-literally-free-bandwidth/
684. https://en.wikipedia.org/wiki/HTTP_compression#Servers_that_support_HTTP_compression
685. https://en.wikipedia.org/wiki/Brotli

コンテンツのエンコード	デスクトップ	モバイル	組み合わせ
テキスト圧縮なし	60.06%	59.31%	59.67%
Gzip	30.82%	31.56%	31.21%
Brotli	9.10%	9.11%	9.11%
その他	0.02%	0.02%	0.02%

図19.4. 圧縮アルゴリズムの採用

圧縮されて提供されているリソースのうち、大部分はGzip（77%）またはBrotli（23%）のいずれかを使用しています。その他の圧縮アルゴリズムはあまり使用されていません。

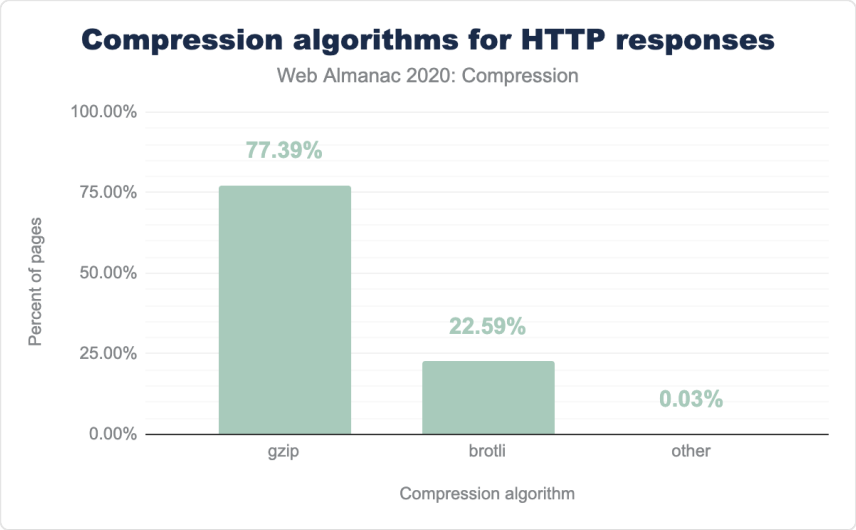


図19.5. HTTPレスポンスの圧縮アルゴリズム。

下のグラフでは、上位11種類のコンテンツタイプが表示されており、ボックスの大きさは回答数の相対的な増減を表しています。各ボックスの色は、これらのリソースのうちどれだけが圧縮されて提供されたかを表しており、オレンジは圧縮の割合が低いことを、青は圧縮の割合が高いことを示しています。メディアコンテンツのほとんどがオレンジ色に塗られていますが、これはGzipやBrotliではほとんどメリットがないことから予想されます。 ほとんどのテキストコンテンツは、圧縮されていることを示すために青く塗られています。しかし、一部のコンテンツタイプの淡い水色は、他のコンテンツほど一貫して圧縮されていないことを示しています。

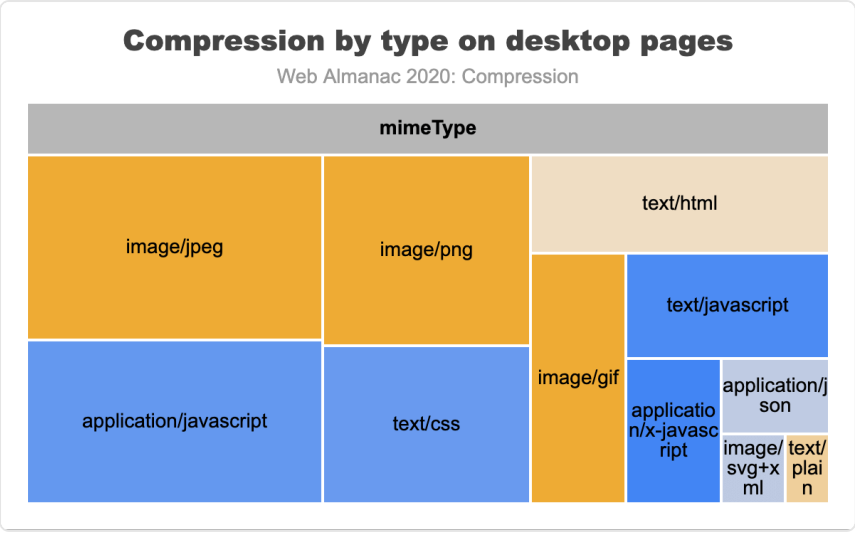


図19.6. デスクトップページのタイプ別圧縮

前述の図19.1では、コンテンツタイプごとの圧縮の割合を示していますが、図19.6では、この割合を色で表しています。この2つの図は似ており、非テキストベースのアセットはほとんど圧縮されておらず、テキストベースのアセットはよく圧縮されていることがわかります。また、圧縮率は、モバイルとデスクトップの両方で似ています。

ファーストパーティとサードパーティの圧縮

サードパーティの章では、サードパーティとそのパフォーマンスへの影響について学びます。サードパーティの使用は、圧縮にも影響します。

	デスクトップ		モバイル	
コンテンツのエンコード	ファースト・パーティ	サードパーティ	ファースト・パーティ	サードパーティ
テキスト圧縮なし	61.93%	57.81%	60.36%	58.11%
Gzip	30.95%	30.66%	32.36%	30.65%
br	7.09%	11.51%	7.26%	11.22%
deflate	0.02%	0.01%	0.02%	0.01%
その他 / 無効	0.01%	0.01%	0.01%	0.01%

図19.7. デバイスタイプ別のファーストパーティ対サードパーティの圧縮率

ファーストパーティとサードパーティの圧縮技術を比較すると、サードパーティのコンテンツはファーストパーティのコンテンツよりも圧縮される傾向にあることがわかります。さらに、Brotliの圧縮率はサードパーティのコンテンツの方が高くなっています。これは、GoogleやFacebookなど、通常Brotliをサポートしている大規模なサードパーティから提供されるリソースの数が多いためと考えられます。

昨年の結果[686]と比較すると、ファーストパーティでのBrotliを中心とした圧縮の使用が大幅に増加し、ファーストパーティとサードパーティ、デスクトップとモバイルの両方で圧縮の使用が約40%になっていることがわかります。しかし、圧縮を使用している回答のうち、ファーストパーティではBrotliの圧縮率はわずか18%、サードパーティでは27%となっています。

あなたのサイトの圧縮率を分析する方法

Firefox Developer Tools[687]またはChrome DevTools[688]を使用すると、ウェブサイトがすでに圧縮しているコンテンツをすばやく把握できます。これを行うには、[ネットワーク]タブを開き、右クリックして[応答ヘッダー]の[コンテンツエンコード]を有効にします。個々のファイルのサイズにカーソルを合わせると、「ネットワークでの転送量」と「リソースサイズ」が表示されます。サイト全体で集計すると、Firefoxではサイズ/転送サイズ、Chromeでは「転送」と「リソース」がネットワークタブの左下に表示されます。

686. https://almanac.httparchive.org/ja/2019/compression#first-party-vs-third-party-compression
687. https://developer.mozilla.org/docs/Tools
688. https://developers.google.com/web/tools/chrome-devtools

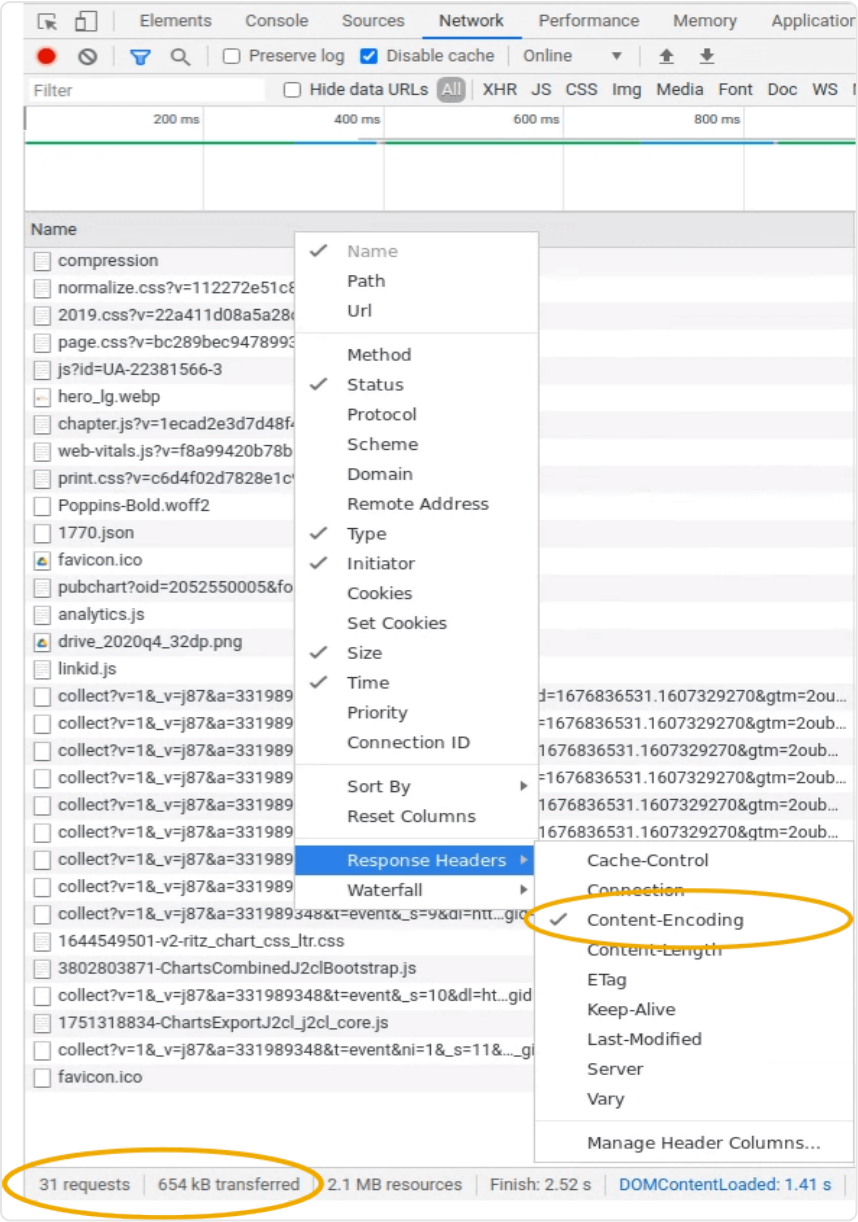


図19.8. DevToolsを使用して、サイトでコンテンツのエンコーディングが使用されているかどうかを確認する

また、サイトの圧縮について理解を深めるためのツールとして、GoogleのLighthouse[689]というツールがあり、ウェブページに対して一連の監査を実行できます。テキスト圧縮監査[690]では、サイトがテキストベースの圧縮を追加することで利益を得られるかどうかを評価します。この監査では、リソースの圧縮を試み、オブジェクトのサイズを少なくとも10%および1,400バイト削減できるかどうかを評価します。スコアに応じて、結果に圧縮の推奨事項が表示され、圧縮可能な特定のリソースのリストが表示されます。

HTTP Archive Lighthouseの監査を実施はモバイルページごとに行われるため、すべてのサイトのスコアを集計して、より多くのコンテンツを圧縮する機会がどれだけあるかを知ることができます。全体では74%のウェブサイトがこの監査に合格していますが、約13%のウェブサイトが40点以下のスコアを記録しています。これは、昨年[691]の62.5%の合格スコアと比較すると、11.5%の改善となります。

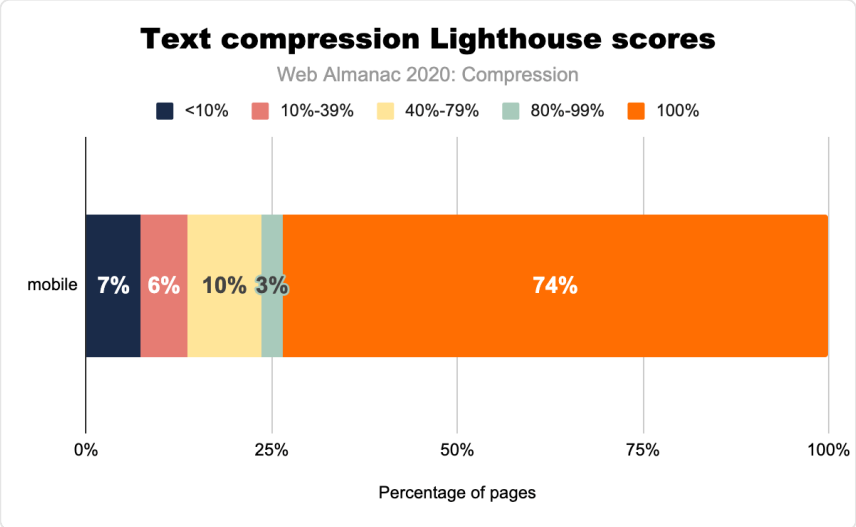


図19.9. テキスト圧縮Lighthouseのスコア

結論

昨年のAlmanac[692]と比較すると、より多くのテキスト圧縮を使用する傾向が明らかになっています。テキスト圧縮を一切使用していない回答は2%強減少し、同時にBrotliの使用が2%近く増加しています。Lighthouseのスコアは大幅に向上しました。

689. https://developers.google.com/web/tools/lighthouse
690. https://web.dev/i18n/ja/uses-text-compression/
691. https://almanac.httparchive.org/ja/2019/compression#identifying-compression-opportunities
692. https://almanac.httparchive.org/ja/2019/compression

テキストの圧縮は、関連するフォーマットに広く使用されていますが、HTTPレスポンスの中には、さらなる圧縮を必要とするものがかなりの割合で存在しています。サーバーの設定をよく見て、必要に応じて圧縮方法やレベルを設定するとよいでしょう。もっとも一般的なHTTPサーバーのデフォルトを注意深く選択することで、よりポジティブなユーザー体験に大きな影響を与えることができます。

著者

Moritz Firsching

mo271 https://mo271.github.io/

Moritz Firschingは、Google Switzerlandのソフトウェアエンジニアで、プログレッシブ画像フォーマットとフォント圧縮に取り組んでいます。それ以前は、数学者としてポリトープの研究をしていました。

Luca Versari

veluca93

Luca Versariは、Googleのソフトウェアエンジニアで、JPEG XL[693]の開発に携わっています。グラフ圧縮に関する博士号を取得中で、数学のバックグラウンドを持っています。

Sami Boukortt

sboukortt

Samiは、工学数学の研究を終えてGoogleに入社しました。圧縮に遠隔で興味を持って数年後、最終的には2018年にフルタイムの仕事になりました。

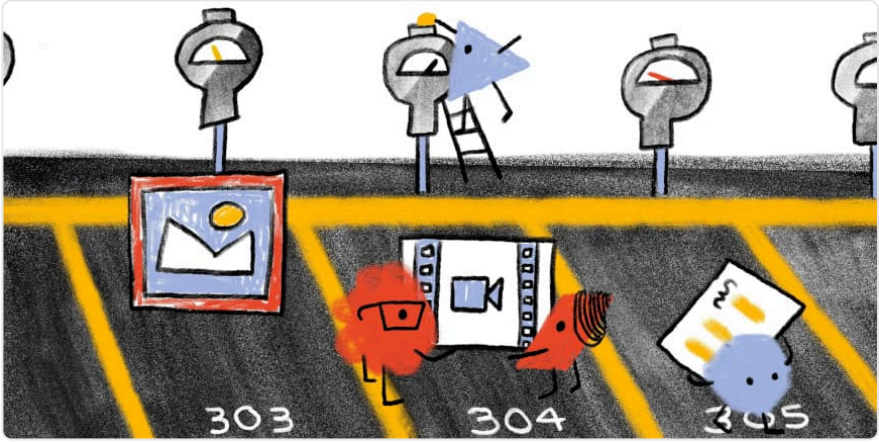
693. *https://gitlab.com/wg1/jpeg-xl*

Jyrki Alakuijala

@jyzg jyrkialakuijala

Jyrki Alakuijalaは、オープンソースソフトウェアコミュニティの活発なメンバーであり、データ圧縮の研究者でもあります。最近の研究テーマは、Zopfli、Butteraugli、Guetzli、Gipfeli、WebP lossless、Brotli、JPEG XLなどの圧縮フォーマットとアルゴリズム、およびCityHashとHighwayHashという2つのハッシュアルゴリズムです。Googleに入社する前は、神経外科や放射線治療の治療計画用ソフトウェアを開発していました。

部 IV 章 20
キャッシング



Rory Hewitt と Raghu Ramakrishnan によって書かれた。
Julia Yang によってレビュ。
Raghu Ramakrishnan による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

キャッシングとは、一度ダウンロードしたコンテンツを再利用するための技術です。これは、何か（ウェブページを構築するサーバー、CDNなどのプロキシ、またはブラウザ自体）が「コンテンツ」（ウェブページ、CSS、JS、画像、フォントなど）を保存し適切なタグを付けることで再利用できるようにするものです。

非常にハイレベルな例を挙げてみましょう。

Janeは，ウェブサイト www.example.com のホームページを訪れました．Janeはカリフォルニア州ロサンゼルスに住んでおり、example.comのサーバーはマサチューセッツ州ボストンにあります。Janeが www.example.com にアクセスする際には、国を越えて移動しなければならないネットワークリクエストが発生します。

example.comのサーバー（通称：Originサーバー）では、ホームページが取得されます。サー

バーはJaneがLAに住んでいることを知り、Janeの近くで開催されるイベントのリストなどの動的コンテンツをページに追加します。その後、ページはアメリカのジェーンに送信され、ジェーンのブラウザに表示されます。

キャッシングがない場合、LAのCarlosがJaneの後に www.example.com を訪れた場合、彼のリクエストは国を越えてexample.comのサーバーに移動しなければなりません。サーバーは、LAのイベントリストを含む同じページを構築しなければなりません。そして、そのページをCarlosに送り返さなければなりません。

さらに悪いことに、Janeがexample.comのホームページを再訪すると、それ以降のリクエストは最初のリクエストと同じようになります。リクエストは国を越えて移動し、example.comのサーバーはホームページを再構築して彼女に送り返さなければなりません。

つまり、キャッシングがなければ、example.comのサーバーは各リクエストを最初から構築することになります。これはサーバーにとって、より多くの仕事をすることになるので悪いことです。さらに、JaneまたはCarlosとexample.comサーバーとの間の通信で、データは国を越えて移動する必要があります。このようなことが重なると、体感速度は低下し、二人にとって良くないことになります。

しかしサーバーキャッシングでは、Janeが最初のリクエストをしたときに、サーバーはホームページのLA版を構築します。これは、すべてのLAの訪問者が再利用できるようにデータをキャッシュします。ですから、Carlosのリクエストがexample.comのサーバーに届いたとき、サーバーはホームページのLA版がキャッシュにあるかどうかをチェックします。Janeの以前のリクエストの結果、そのページはキャッシュに入っているので、サーバーはキャッシュされたページを返すことで時間を節約します。

さらに重要なことは、ブラウザキャッシュではJaneのブラウザは最初のリクエストでサーバからページを受け取ると、そのページをキャッシュします。今後のexample.comのホームページへのリクエストはすべて、ネットワークリクエストなしに、ブラウザのキャッシュから提供されることになります。example.comのサーバも、Janeのリクエストを処理する必要がないというメリットがあります。

Janeは幸せです。Carlosも幸せです。example.comの皆さんも幸せです。みんな幸せです。

ブラウザのキャッシングは、コストのかかるネットワークリクエストを回避することで、パフォーマンス上の大きなメリットをもたらすことは明らかでしょう（ただし、常にエッジケースが存在します[694]）。また、ウェブサイトのオリジン・インフラストラクチャへのトラフィックを減らすことで、アプリケーションの拡張にも役立ちます。サーバーキャッシングは、基礎となるアプリケーションの負荷を大幅に軽減します。

キャッシングは、エンドユーザーにとっても（Webページを早く手に入れられる）、Webペ

694. *https://simonhearne.com/2020/network-faster-than-cache/*

ージを提供する企業にとっても（サーバーの負荷を軽減できる）メリットがあります。キャッシングはまさにWin-Winの関係なのです。

ウェブ・アーキテクチャでは、一般的に複数の階層のキャッシングが行われます。キャッシングが行われる場所としては、主に4つの場所、すなわち*caching entities*があります。

1. エンドユーザーのブラウザです。
2. エンドユーザーのブラウザ上で動作するサービスワーカーのキャッシュです。
3. コンテンツデリバリーネットワーク（CDN）などのプロキシで、エンドユーザーのブラウザとオリジンサーバーの間に設置される。
4. オリジン・サーバーそのものです。

本章では、オリジンサーバーやCDNでのキャッシングではなく、主にブラウザ内（1-2）でのキャッシングについて説明します。とはいえ、本章で取り上げるキャッシングに関する具体的なトピックの多くは、ブラウザとサーバー（CDNを使用している場合はそのサーバー）との関係に依存しています。

キャッシングやウェブの仕組みを理解するには、リクエストする側（ブラウザなど）とレスポンスする側（サーバなど）の間のトランザクションで成り立っていることを覚えておくとよいでしょう。各トランザクションは2つの部分で構成されています。

1. リクエストする側のエンティティからの要求。”*私はオブジェクトXが欲しい*“となります。
2. 応答する側のエンティティからのレスポンスです。”*これがオブジェクトXです*“となります。

キャッシングというと、リクエストした側のオブジェクト（HTMLページや画像など）がキャッシュされることを指します。

下図は、オブジェクト（Webページなど）に対する典型的なリクエスト／レスポンスの流れを示しています。ブラウザとサーバの間にはCDNがあります。ブラウザ→CDN→サーバの流れの各ポイントで、各キャッシングエンティティは、まず自分のキャッシュにオブジェクトがあるかどうかを確認します。見つかった場合は、キャッシュされたオブジェクトをリクエスト元に返し、その後、リクエストを次のキャッシュエンティティに転送します。

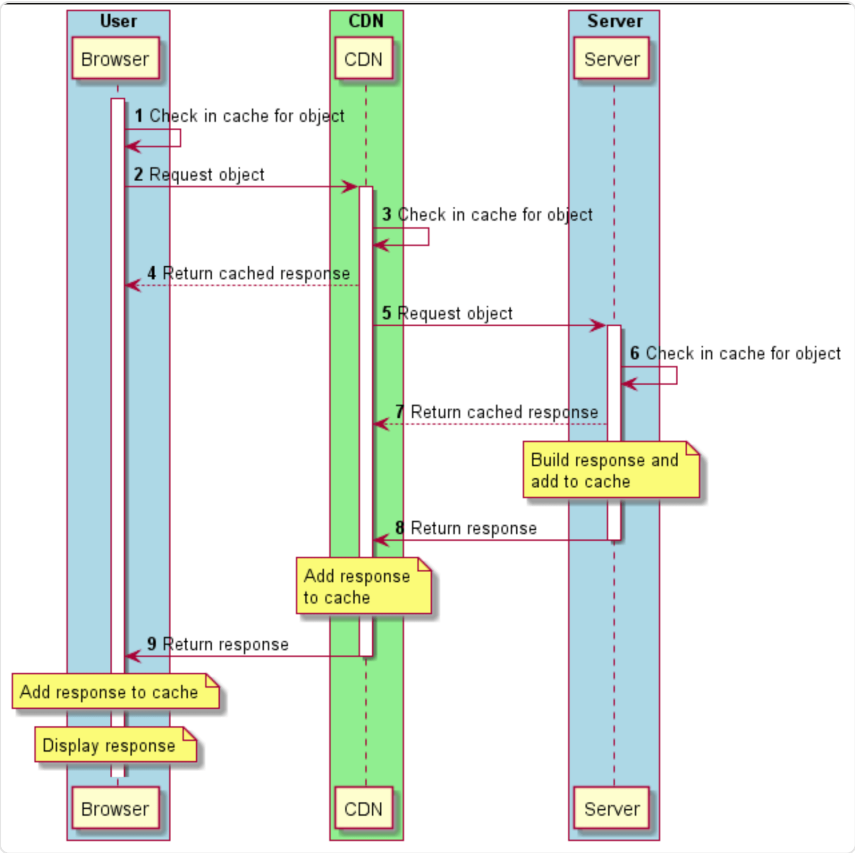


図20.1. オブジェクトのリクエスト/レスポンスフロー。

注：本章では、特に明記しない限り、デスクトップの統計も同様であると理解した上で、すべての統計をモバイル用にしています。モバイルとデスクトップの統計が大きく異なる場合は、その旨を明記しています。

本章で使用するレスポンスの多くは、一般的に入手可能なサーバーパッケージを使用するウェブサーバーからのものです。ベストプラクティスを示すことはできても、使用しているソフトウェアパッケージのキャッシュオプションの数が限られている場合には、ベストプラクティスを実現できない可能性があります。

キャッシングの基本原則

ウェブコンテンツのキャッシュには、3つの指針があります。

- 可能な限りのキャッシュ
- できる限り長くキャッシュする
- エンドユーザーにできるだけ近い場所でキャッシュする

可能な限りのキャッシュ

何をキャッシュするかを検討する際には、レスポンスの内容が*static*なのか*dynamic*なのかを理解することが重要です。

静的コンテンツ

静的コンテンツの例として、画像があります。たとえばcat.jpgというファイルに収められた猫の写真は、誰がリクエストしてもどこにいても通常は同じものです（もちろん別のフォーマットやサイズで配信されることもありますが、通常は別のファイル名で配信されます）。

図20.2. はい、猫の写真があります。

静的コンテンツは、一般的にキャッシュ可能で、多くの場合、長期間にわたって利用されます。コンテンツ（1つ）とリクエスト（多数）の間には、1対多の関係があります。

ダイナミックコンテンツ

ダイナミックコンテンツの例としては、ある地域に特化したイベントのリストがあります。このリストは、リクエストした人の場所に応じて異なります。

動的に生成されたコンテンツは、より微妙で、慎重な検討が必要です。動的コンテンツの中にはキャッシュできるものもありますが、多くの場合、短い期間しかキャッシュできません。近日開催のイベントのリストの例では、おそらく日ごとに変化します。また、ユーザーのブラウザにキャッシュされるものは、サーバーやCDNにキャッシュされるもののサブセットである可能性もあります。とはいえ、一部のダイナミックコンテンツをキャッシュすることは可能です。「ダイナミック」を「キャッシュできない」の別の言葉だと考えるのは正しくありません。

できる限り長くキャッシュする

リソースをキャッシュする時間の長さは、コンテンツの*揮発性*、つまり変更の可能性や頻度に大きく依存します。たとえば、画像やバージョン管理されたJavaScriptファイルは、非常に長い時間キャッシュされる可能性があります。APIレスポンスやバージョン管理されていないJavaScriptファイルは、ユーザーに最新のレスポンスを確実に提供するために、より短いキャッシュ期間が必要になるかもしれません。一部のコンテンツは1分以下しかキャッシュされないかもしれません。もちろん、まったくキャッシュされるべきではないコンテンツもあります。これについては、後述のキャッシュの機会を特定するで詳しく説明しています。

もうひとつの留意点は、ブラウザにコンテンツのキャッシュをどれだけ長く指示しても、ブラウザはその時点以前にそのコンテンツをキャッシュから削除することがあるということです。たとえば、より頻繁にアクセスされる他のコンテンツのためのスペースを確保するためにそうすることがあります。しかし、ブラウザは指示された時間以上にキャッシュコンテンツを使用することはありません。

できるだけエンドユーザーの近くにキャッシュする

エンドユーザーの近くにコンテンツをキャッシュすることで、遅延をなくしてダウンロード時間を短縮できます。たとえば、あるリソースがユーザーのブラウザにキャッシュされていればリクエストがネットワークに出ることはなく、ユーザーが必要とするときにはいつでもローカルで利用できます。ブラウザのキャッシュにエントリがない訪問者にとっては、CDNはキャッシュされたリソースが次に返される場所となります。ほとんどの場合、オリジンサーバーと比較して、ローカルキャッシュやCDNからリソースを取得する方が速くなります。

いくつかの専門用語

Caching entity: キャッシングを行うハードウェアまたはソフトウェアのこと。本章では、とくに指定のない限り、「ブラウザ」を「キャッシング・エンティティ」の同義語として使用します。

Time-To-Live (TTL): キャッシュオブジェクトのTTLは、そのオブジェクトがキャッシュに保存される期間を定義するもので、通常は秒単位で測定されます。キャッシュされたオブジェクトがTTLに達すると、そのオブジェクトはキャッシュによって「古い」とマークされます。キャッシュにどのように追加されたか（下記のキャッシュヘッダーの詳細を参照）に応じて、そのオブジェクトは直ちにキャッシュから削除されるか、あるいはキャッシュに残っていても「stale」オブジェクトとしてマークされ再利用の前に再検証が必要になることがあります。

Eviction: オブジェクトがTTLに達したとき、あるいはキャッシュが満杯になったとき、実際にキャッシュから削除される自動化されたプロセスのこと。

Revalidation: 古くなったと判定されたキャッシュオブジェクトは、ユーザーに表示する前、サーバーで「再検証」する必要があります。ブラウザはまず、ブラウザのキャッシュにあるオブジェクトが最新で有効であることをサーバに確認しなければなりません。

ブラウザキャッシングの概要

ブラウザがコンテンツ（Webページなど）を要求すると、コンテンツそのもの（HTMLマークアップ）だけでなく、コンテンツを説明する多くの*HTTPレスポンス*ヘッダー（キャッシュ可能性に関する情報を含む）を含むレスポンスを受け取ります。

キャッシング関連のヘッダーがあるかないかで、ブラウザに3つの重要な情報が伝わります。

1. **キャッシュ性**: このコンテンツはキャッシュ可能ですか？
2. **新鮮さ**: キャッシュ可能な場合、どのくらいの期間キャッシュできるのか？
3. **バリデーション**: もしキャッシュ可能であれば、その後どのようにしてキャッシュされたバージョンが新鮮であることを確認するのでしょうか？

鮮度を指定するために一般的に使用される2つのHTTPレスポンスヘッダーは、`Cache-Control`と`Expires`です。

- `Expires`は、明示的な有効期限の日時を指定します（つまり、コンテンツの有効期限がいつ切れるかを指定します）。

- `Cache-Control` では、キャッシュ期間（リクエストされた時からどのくらいの期間、ブラウザにコンテンツをキャッシュできるか）を指定します。

しばしば、これらのヘッダーの両方を指定することがありますが、その場合は `Cache-Control` が優先されます。

これらのキャッシングヘッダーの完全な仕様は、RFC 7234[695]にあり、4.2 (Freshness)[696]と4.3 (Validation)[697]のセクションで説明されていますが、以下ではより詳細に説明します。

`Cache-Control` 対 `Expires`

初期のHTTP/1.0時代のウェブでは、 `Expires` ヘッダーが唯一のキャッシュ関連のレスポンスヘッダーでした。上述したように、このヘッダーは、レスポンスが古くなったとみなされる正確な日時を示すために使用されます。その値は、次のような日時です。

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

`Expires` ヘッダーは、鈍器のようなものと考えてよいでしょう。相対的なキャッシュTTLが必要な場合は、現在の日付/時刻に基づいて適切な値を生成するために、サーバー上で処理を行う必要があります。

HTTP/1.1では、一般的に使用されているすべてのブラウザで長い間サポートされている `Cache-Control` ヘッダーが導入されました。 `Cache-Control` ヘッダーは、キャッシング*ディレクティブ*を介して、 `Expires` よりもはるかに高い拡張性と柔軟性を備えており、複数のディレクティブを一緒に指定できます。さまざまなディレクティブの詳細は以下の通りです。

```
> GET /static/js/main.js HTTP/2
> Host: www.example.org
> Accept: */*
< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:04:17 GMT
< Expires: Thu, 23 Jul 2020 03:14:17 GMT
```

695. *https://tools.ietf.org/html/rfc7234#section-8*
696. *https://tools.ietf.org/html/rfc7234#section-4.2*
697. *https://tools.ietf.org/html/rfc7234#section-4.3*

```
< Cache-Control: public, max-age=600
```

上の簡単な例では、JavaScriptファイルのリクエストとレスポンスを示していますが、わかりやすくするためにいくつかのヘッダーを削除しています。`Date`ヘッダーは、現在の日付（具体的には、コンテンツが提供された日付）を示しています。`Expires`ヘッダーは、10分間キャッシュできることを示しています（`Expires`と`Date`ヘッダーの違い）。`Cache-Control`ヘッダーは、`max-age`ディレクティブを指定しており、これはリソースが600秒（5分）の間キャッシュできることを示している。`Cache-Control`は`Expires`よりも優先されるので、ブラウザはレスポンスを5分間キャッシュし、それ以降は陳腐化したものとしてマークされます。

RFC 7234では、レスポンスにキャッシュ用のヘッダーが存在しない場合、ブラウザはレスポンスを*ヒューリスティック*にキャッシュできます。これは、`Last-Modified`ヘッダー（渡された場合）からの時間の10%をキャッシュ期間として提案しています。このような場合ほとんどのブラウザはこの提案のバリエーションを実装していますが、中にはレスポンスを無期限にキャッシュするものや、まったくキャッシュしないものもあります。このようにブラウザによってばらつきがあるため、コンテンツのキャッシュ可能性を確実にコントロールするために、特定のキャッシュルールを明示的に設定することが重要です。

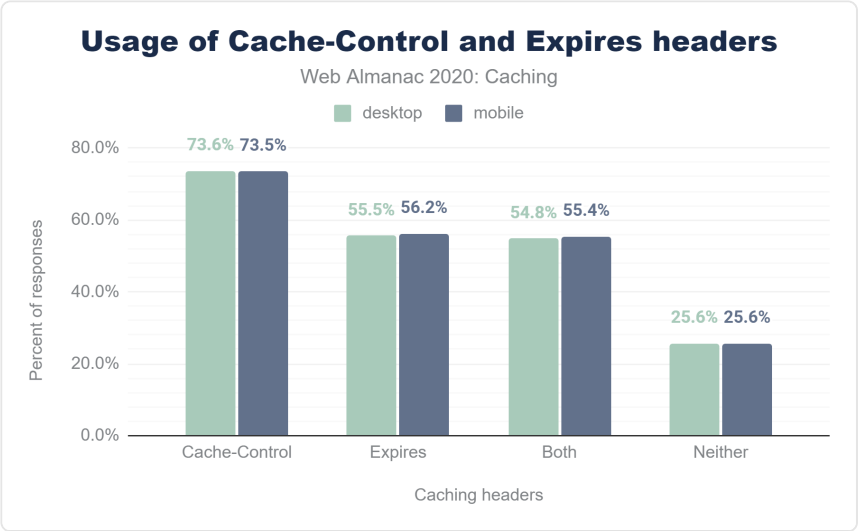


図20.3. `Cache-Control`および`Expires`ヘッダーの使用法。

モバイルのレスポンスのうち73.5%は`Cache-Control`ヘッダーが付いており、56.2%は`Expires`ヘッダーが付いていますが、レスポンスに両方のヘッダーが含まれているため、これらのほぼすべて（55.4%）は使用されません。25.6%のレスポンスはどちらのヘッダー

も含まれていないため、ヒューリスティックキャッシングの対象となります。

この統計は、昨年のデータと比較すると興味深いものです。

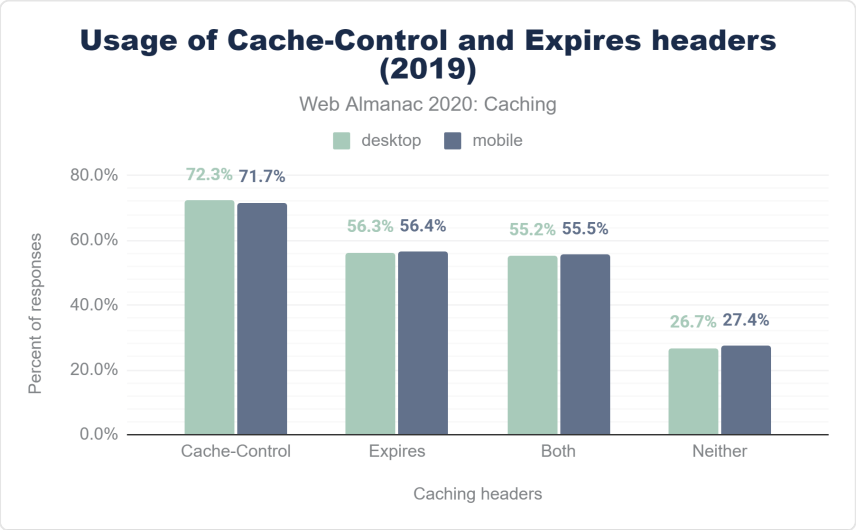


図20.4. 2019年の `Cache-Control` と `Expires` ヘッダーの使用法。

また、デスクトップでは、`Cache-Control` ヘッダーの使用がわずかに増加（1.8%）していますが、古い `Expires` ヘッダーの使用はわずかに減少（0.2%）しています。デスクトップでは、`Cache-Control` がわずかに増加し（1.3%）、`Expires` の増加は少ない（0.8%）ことがわかります。つまり、デスクトップのサイトでは、`Expires` ではなく `Cache-Control` ヘッダーを追加するケースが増えているようです。

また、`Cache-Control` ヘッダーで許可されているさまざまなディレクティブを掘り下げていくと、その柔軟性とパワーが、多くのケースでより適していることがわかるでしょう。

`Cache-Control` ディレクティブ

`Cache-Control` ヘッダーを使用する際には、特定のキャッシュ機能を示す1つ以上のディレクティブ一定義済みの値を指定します。複数のディレクティブはカンマで区切られ、どのような順番でも指定できますが、中にはお互いに ぶつかり合う ものもあります（例：`public` と `private`）。いくつかのディレクティブは `max-age` のように値を取るものもあります。

以下は、もっとも一般的な `Cache-Control` ディレクティブの一覧です。

ディレクティブ	説明
`max-age`	現在の時刻を基準にして、リソースをキャッシュできる秒数を示します。 たとえば`max-age=86400`。
`public`	ブラウザや、サーバとブラウザの間にあるCDNなどのプロキシを含む、あらゆるキャッシュがレスポンスを保存する可能性があります。これはデフォルトで想定されています。
`no-cache`	キャッシュされたエントリは、たとえstaleとマークされていなくても、使用前に条件付きリクエストで再検証されなければなりません。
`must-revalidate`	古くなったキャッシュエントリは、使用前に条件付きリクエストで再検証する必要があります。
`no-store`	応答をキャッシュしてはいけないことを示す。
`private`	レスポンスは特定のユーザーを対象としており、プロキシやCDNなどの共有キャッシュには保存されないようになっています。
`proxy-revalidate`	`must-revalidate`と同じですが、共有キャッシュに適用されます。
`s-maxage`	`max-age`と同じですが、共有キャッシュ（例：CDN）にのみ適用されます。
`immutable`	TTLの間、キャッシュされたエントリが変更されることはなく、再検証は必要ないことを示す。
`stale-while-revalidate`	クライアントが、古い応答を受け入れる一方で、新しい応答をバックグラウンドで非同期的にチェックすることを示します。
`stale-if-error`	新鮮な応答のチェックに失敗した場合に、クライアントが古い応答を受け入れることを示す。

図20.5. `Cache-Control` ディレクティブ。

`max-age` ディレクティブは、 `Expires` ヘッダーと同じように、TTLを直接定義するので、もっともよく見られるものです。

以下は、複数のディレクティブを持つ有効なCache-Controlヘッダーの例です。

```
Cache-Control: public, max-age=86400, must-revalidate
```

これはオブジェクトが86,400秒（1日）キャッシュできることを示しており、サーバとブラウザの間のすべてのキャッシュや、ブラウザ自体にも保存できることを意味しています。

TTLに達しstaleとマークされたオブジェクトは、キャッシュに残りますが、再利用する前に条件付きで再検証する必要があります。

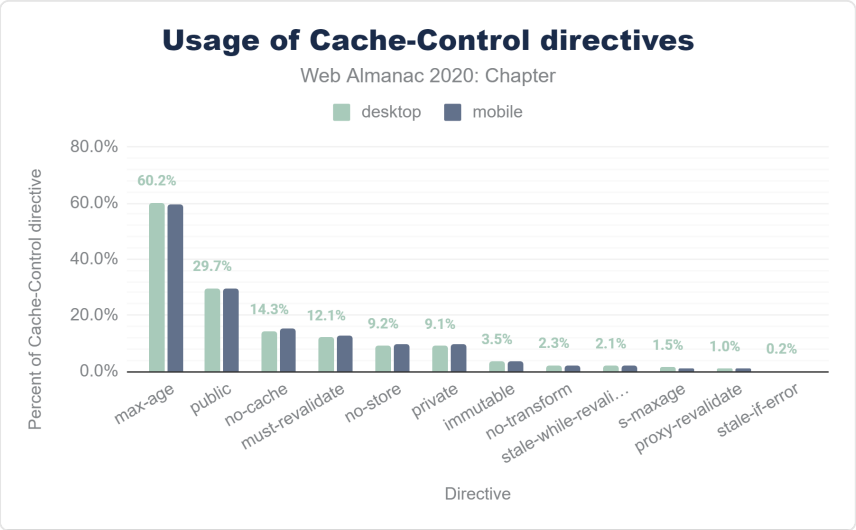


図20.6. `Cache-Control` ディレクティブの配布。

上の図は、モバイルとデスクトップのウェブサイトで使用されている11個の`Cache-Control`ディレクティブを示しています。これらのキャッシュディレクティブの人気度について、いくつかの興味深い見解があります。

- モバイルの`Cache-Control`ヘッダーのうち、`max-age`は約59.66%、`no-store`は約9.64%使用されています（`no-store`ディレクティブの意味と使用方法については後述します）。
- `public`を明示的に指定する必要はありません。なぜなら、`private`が指定されていない限り、キャッシュエントリは`public`とみなされるからです。それにもかかわらず、ほぼ3分の1のレスポンスには`public`が含まれていて、すべてのレスポンスで数個のヘッダーバイトがムダになっています :)
- `immutable`ディレクティブは、2017年に導入された比較的新しいもので、FirefoxとSafariでしかサポートされていません。使用率はまだ3.47%程度ですが、Facebook、Google、Wix、Shopifyなどのレスポンスで広く見られます。特定のタイプのリクエストのキャッシュ性を大幅に向上させる可能性があります。

ロングテールに向かうと、ごく一部の無効なディレクティブが見つかります。これらはブラウザでは無視され、ヘッダーバイトをムダにするだけです。これらは大まかに2つのカテゴ

リーに分けられます。

- `nocache` や `s-max-age` などのスペルミスや、 `=` の代わりに `:` を使用したり、 `-` の代わりに `_` を使用したりするなど、無効なディレクティブの構文があります。
- `max-stale`, `proxy-public`, `surrogate-control` のような存在しないディレクティブ。

ムダなディレクティブのリストの中でもっとも興味深いのは、 `no-cache="set-cookie"` の使用です。 `Cache-Control` ヘッダーの全値のわずか0.2%であっても、他の無効なディレクティブの合計よりも多くを占めています。初期の `Cache-Control` ヘッダーに関する議論の中で、この構文は、 `Set-Cookie` レスポンスヘッダー（ユーザー固有のものである可能性）が、CDNなどの中間プロキシによってオブジェクト自体と一緒にキャッシュされないようするための可能な方法として提起されました。しかし、この構文は最終的なRFCには含まれていませんでした。ほぼ同等の機能は `private` ディレクティブで実装できますし、 `no-cache` ディレクティブでは値を指定することはできません。

`Cache-Control: no-store`, `no-cache` と `max-age=0`

レスポンスを絶対にキャッシュしてはいけない場合には、 `Cache-Control: no-store` ディレクティブを使用します。このディレクティブが指定されていない場合には、レスポンスは *キャッシュ可能であるとみなされ、キャッシュできます*。 `no-store` が指定されている場合は、他のディレクティブよりも優先されることに注意してください。これは、キャッシュされるべきではないリソースがキャッシュされた場合に深刻なプライバシーやセキュリティの問題が、発生する可能性があるためです。

レスポンスをキャッシュできないように設定しようとしたときに発生する、いくつかの一般的なエラーを見ることができます。

- `Cache-Control: no-cache` を指定すると、リソースをキャッシュしないように指示しているように聞こえるかもしれません。しかし、上述のとおり、 `no-cache` ディレクティブはリソースのキャッシュを許可します。これは単に、使用前にリソースを再検証するようにブラウザに通知するだけで、リソースのキャッシュをまったく行わないこととは異なります。
- `Cache-Control: max-age=0` を設定するとTTLが0秒になりますが、これはキャッシュされないこととは違います。 `max-age=0` が指定されると、リソースはキャッシュされますが、staleとマークされるため、ブラウザは即座にその新鮮さを再検証しなければなりません。

機能的には、`no-cache`と`max-age=0`は似ていて、どちらもキャッシュされたリソースの再検証を要求します。`no-cache`ディレクティブは、0よりも大きな値の`max-age`ディレクティブと一緒に使うこともできます。この場合、オブジェクトは指定されたTTLの間キャッシュされますが、毎回使用する前に再検証されます。

上記の3つのディレクティブを見ると、`no-store`、`no-cache`、`max-age=0`の3つのディレクティブを組み合わせたものが2.7%、`no-store`と`no-cache`の両方を組み合わせたものが6.7%、`no-store`のみを組み合わせたものは0.15%以下とごくわずかしかありません。

上述したように、`no-store`が`no-cache`と`max-age=0`のどちらかまたは両方と一緒に指定された場合、no-storeディレクティブが優先され、他のディレクティブは無視されます。したがってコンテンツをどこにもキャッシュしたくない場合は、単純に`Cache-Control: no-store`を指定するだけで十分であり、よりシンプルで最小限のヘッダーバイトしか使用しません。

`max-age=0`ディレクティブは、`no-store`が指定されていないレスポンスの2%未満にしか存在しません。このような場合、リソースはブラウザにキャッシュされますが、すぐにstaleと判定されるため、再検証が必要になります。

条件付きリクエストとリバリデーション

ブラウザが以前にオブジェクトをリクエストしすでにキャッシュに入っているが、そのキャッシュエントリがすでにTTLを超えている（したがってstaleとマークされている）場合や、オブジェクトが使用前に再検証されなければならないものとして定義されている場合などがよくあります。

このような場合、ブラウザはサーバに対して条件付きのリクエストを行うことができます。つまり、「*私のキャッシュにはobject Xがあるのですが、これを使ってもいいでしょうか。それとも、もっと新しいバージョンを使った方がいいでしょうか？*」。サーバーは以下の2つの方法で応答します。

- 「*はい、キャッシュに保存されているバージョンのオブジェクトXは問題なく使用できます。*」となります。この場合、サーバーからのレスポンスは、`304 Not Modified`というステータスコードとレスポンスヘッダーで構成され、レスポンスボディはありません。
- “*いいえ、ここに最新のバージョンのオブジェクトXがありますので、これを使ってください。*” となります。この場合、サーバーからのレスポンスは、`200 OK`のステータスコード、レスポンスヘッダー、そして新しいレスポンスボディ（実際の新しいバージョンのオブジェクトX）で構成されます。

いずれの場合も、サーバはオプションでキャッシング応答ヘッダーを更新し、オブジェクトのTTLを延長できます。そうすれば、ブラウザは条件付きのリクエストを重ねることなく、さらに長期間オブジェクトを使用できます。

`304 Not Modified`のレスポンスはヘッダーのみで構成されているため、上記は*revalidation*と呼ばれ、正しく実装されていれば、知覚的なパフォーマンスを大幅に向上させることができます。それは、`200 OK`の応答よりもはるかに小さいため、帯域幅が削減され、迅速な応答が可能になります。

では、サーバーはどのようにして条件付きのリクエストと通常のリクエストを識別するのでしょうか？

実際には、オブジェクトへの最初のリクエストに起因するものです。ブラウザがキャッシュにないオブジェクトをリクエストするときは、次のようなGETリクエストを行うだけです。

```
> GET /index.html HTTP/2
> Host: www.example.org
> Accept: */*
```

ブラウザが条件付きリクエストを利用できるようにしたい場合（この判断は完全にサーバ側に委ねられています！）、サーバは、オブジェクトが後続の条件付きリクエストの対象であることを識別する2つのレスポンスヘッダーの一方または両方を含めることができます。2つのレスポンスヘッダーとは。

- `Last-Modified`: これは、オブジェクトが最後に変更された日時を示します。この値は日付のタイムスタンプです。
- `ETag`（エンティティタグ）です。これは、コンテンツのユニークな識別子を引用符付きの文字列で提供します。通常はファイルの内容をハッシュ化したものですが、タイムスタンプや単純な文字列にすることもできます。

両方のヘッダーが存在する場合は、`ETag`が優先されます。

`Last-Modified`

ファイルのリクエストを受け取ったサーバーは、次のように、ファイルがもっとも最近変更された日時をレスポンスヘッダーとして含めることができます。

```
< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:04:17 GMT
< Last-Modified: Mon, 20 Jul 2020 11:43:22 GMT
< Cache-Control: max-age=600

...lots of html here...
```

ブラウザはこのオブジェクトを（`Cache-Control`ヘッダーで定義されているように）600秒間キャッシュし、それ以降はオブジェクトが古くなったものとしてマークします。ブラウザがこのファイルを再度使用する必要がある場合は、最初に行ったのと同様にサーバからファイルを要求しますが、今回は`If-Modified-Since`という追加のリクエストヘッダーを含め最初のレスポンスで`Last-Modified`レスポンスヘッダーに渡された値を設定します。

```
> GET /index.html HTTP/2
> Host: www.example.org
> Accept: */*
> If-Modified-Since: Mon, 20 Jul 2020 11:43:22 GMT
```

このリクエストを受け取ったサーバーは、`If-Modified-Since`ヘッダーの値と、もっとも最近にファイルを変更した日付を比較することで、オブジェクトが変更されたかどうかを確認できます。

2つの値が同じであれば、サーバはブラウザがファイルの最新バージョンを持っていることを知り、サーバはヘッダー（同じ`Last-Modified`ヘッダー値を含む）のみでレスポンスボディを持たない`304 Not Modified`レスポンスを返すことができます。

```
< HTTP/2 304
< Date: Thu, 23 Jul 2020 03:14:17 GMT
< Last-Modified: Mon, 20 Jul 2020 11:43:22 GMT
< Cache-Control: max-age=600
```

しかし、ブラウザが最後にリクエストしてからサーバ上のファイルが変更された場合、サーバはヘッダー（更新された`Last-Modified`ヘッダーを含む）とボディ内のファイルの新バージョンからなる`200 OK`レスポンスを返します。

```
< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:14:17 GMT
< Last-Modified: Thu, 23 Jul 2020 03:12:42 GMT
< Cache-Control: max-age=600

...lots of html here...
```

ご覧の通り、`Last-Modified`レスポンスヘッダーと`If-Modified-Since`リクエストヘッダーはペアで動作しています。

エンティティタグ（`ETag`）

この機能は、前述の日付ベースの`Last-Modified`/`If-Modified-Since`条件付きリクエスト処理とほぼ同じです。

しかし、この場合、サーバーは日付のタイムスタンプではなく、`ETag`レスポンスヘッダーを送信します。`ETag`は単なる文字列で、ファイルの内容をハッシュ化したものや、サーバが算出したバージョン番号などが多いです。この文字列のフォーマットはサーバが自由に決めることができます。唯一の重要な事実は、サーバはファイルを変更するたびに`ETag`の値を変更するということです。

この例では、サーバーがファイルの最初のリクエストを受信したときに、次のように`ETag`応答ヘッダーでファイルのバージョンを返すことができます。

```
< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:04:17 GMT
< ETag: "v123.4.01"
< Cache-Control: max-age=600

...lots of html here...
```

上記の`If-Modified-Since`の例と同様に、ブラウザは`Cache-Control`ヘッダーで定義されているように、このオブジェクトを600秒間キャッシュします。このオブジェクトを再度サーバーにリクエストする必要がある場合は、`If-None-Match`と呼ばれる追加のリクエストヘッダーを含めます。このリクエストヘッダーには、最初のレスポンスの`ETag`レスポンスヘッダーで渡された値が含まれています。

```
> GET /index.html HTTP/2
> Host: www.example.org
> Accept: */*
> If-None-Match: "v123.4.01"
```

このリクエストを受け取ったサーバーは、`If-None-Match`ヘッダー値と、そのファイルの現在のバージョンを比較することで、オブジェクトが変更されたかどうかを確認できます。

2つの値が同じであれば、ブラウザはファイルの最新バージョンを持っていることになり、サーバーはヘッダーだけで`304 Not Modified`レスポンスを返すことができます。

```
< HTTP/2 304
< Date: Thu, 23 Jul 2020 03:14:17 GMT
< ETag: "v123.4.01"
< Cache-Control: max-age=600
```

しかし値が異なる場合はサーバー上のファイルのバージョンが、ブラウザが持っているバージョンよりも新しいということになるので、サーバーはヘッダー（更新された`ETag`ヘッダーを含む）とファイルの新しいバージョンからなる`200 OK`レスポンスを返します。

```
< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:14:17 GMT
< ETag: "v123.5.06"
< Cache-Control: public, max-age=600

...lots of html here...
```

ここでも、この条件付きリクエスト処理には、`ETag`レスポンスヘッダーと`If-None-Match`リクエストヘッダーという2つのヘッダーが使われています。

`Cache-Control`ヘッダーが`Expires`ヘッダーよりも強力で柔軟性があるのと同じように、`ETag`ヘッダーは多くの点で`Last-Modified`ヘッダーよりも改良されています。これには2つの理由があります。

1. サーバーは、`ETag`ヘッダーに独自のフォーマットを定義できます。上の例では、バージョンの文字列を示していますが、ハッシュやランダムな文字列にする

こともできます。これを許可すると、オブジェクトのバージョンは日付に明示的にリンクされません。そのため、サーバーはファイルの新しいバージョンを作成しても、以前のバージョンと同じETagを与えることができます。

2. `ETag` は'strong'または'weak'のいずれかとして定義でき、これによりブラウザは異なる方法で検証できます。この機能を完全に理解し、議論することはこの章の範囲を超えていますが、 RFC 7232[698]に記載されています。

しかし、 `ETag` はサーバの最終更新時刻をベースにしていることが多いので、多くの実装で事実上同じになってしまう可能性があります。さらに悪いことに、サーバの実装（とくにApache）にはさまざまなバグがあり、 `ETag` を使うことがあまり効果的ではないこともあります。

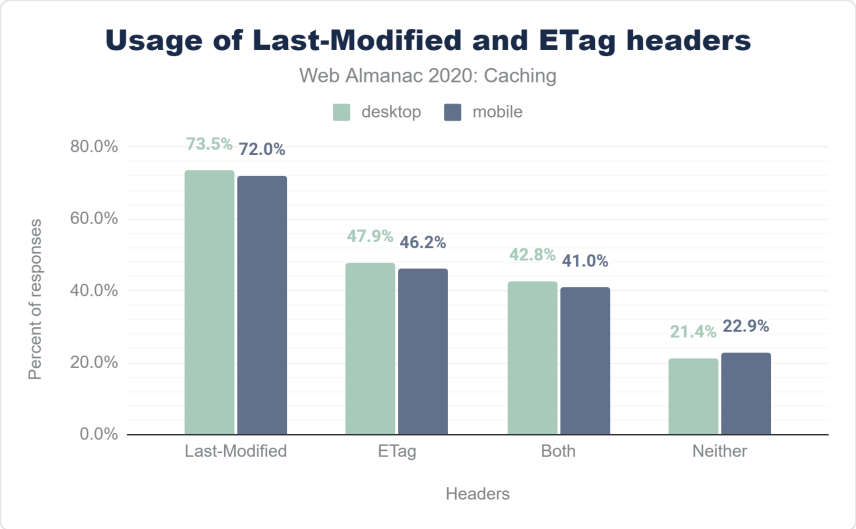


図20.7. `Last-Modified` と `ETag` ヘッダーによる新鮮さの検証の採用。

698. https://tools.ietf.org/html/rfc7232

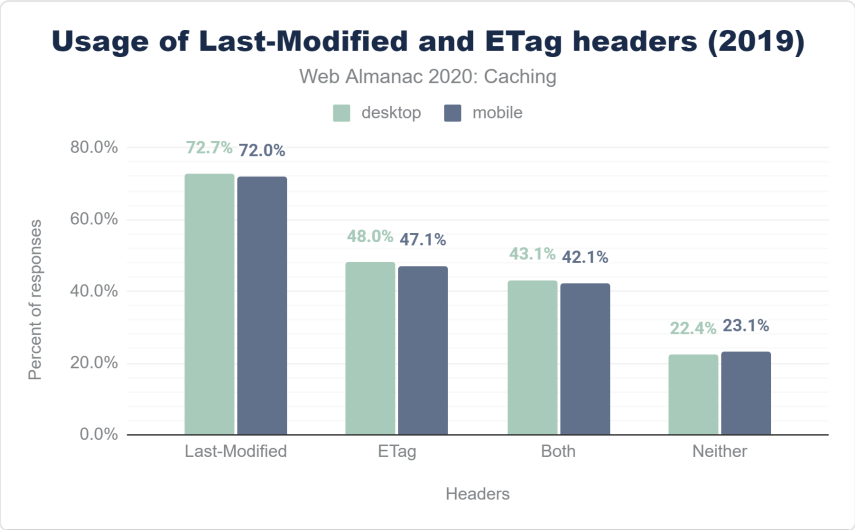


図20.8. 2019年には、`Last-Modified`と`ETag`ヘッダーによる新鮮さの検証を採用。

モバイルのレスポンスの72.0%が`Last-Modified`ヘッダー付きで提供されていることがわかります。2019年との比較では、モバイルでの使用率は横ばいですが、デスクトップではわずかに増加しています（1%未満）。

`ETag`ヘッダーを見ると、携帯電話の回答の46.2%がこれを使用しています。これらの回答のうち、34.38%が*string*、9.81%が*weak*、残りの1.98%が無効となっています。`Last-Modified`とは対照的に、`ETag`ヘッダーの使用率は、2019年と比較してわずかに減少しています（1%未満の減少）。

41.0%のモバイルレスポンスは、両方のヘッダーを含んでおり、上述の通りこの場合は`ETag`ヘッダーが優先されます。22.9%のモバイルレスポンスは、`Last-Modified`と`ETag`のどちらのヘッダーも含まれていません。

条件付きリクエストを使用して再検証を正しく実装すれば帯域幅（304応答は一般的に200応答よりもはるかに小さい）、サーバーの負荷（変更日やハッシュを比較するために必要な処理はわずか）、および知覚的なパフォーマンスの向上（サーバーは304でより迅速に応答する）を大幅に削減できます。しかし上記の統計からもわかるように、全リクエストの5分の1以上は、どのような形式の条件付きリクエストも使用していません。

クロールでは空のキャッシュを使用しており、304レスポンスはほとんどがMethodologyではテストしていない後続の訪問に役立つものなので、これは予想外のことではありません。それでも、304がどのように使用されているかを確認するためにこれらを分析しました。

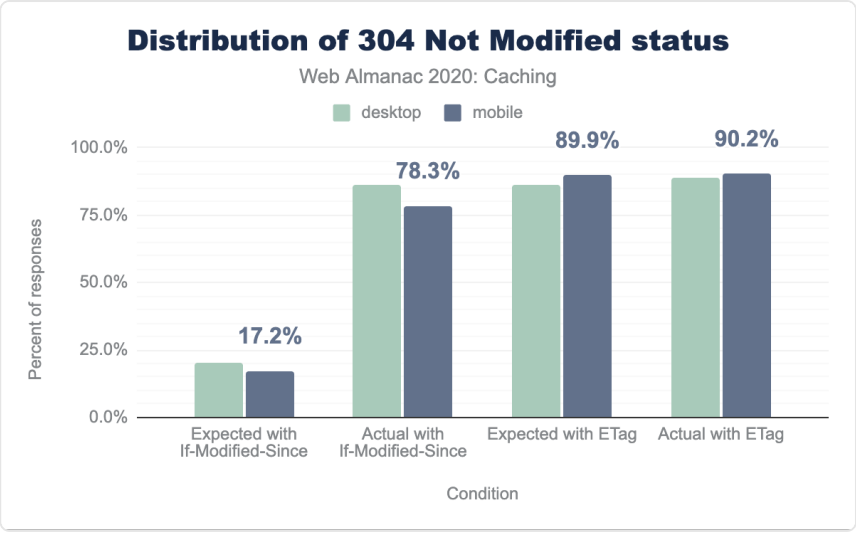


図20.9. `304 Not Modified` ステータスの分布

モバイルのレスポンスのうち17.2%（デスクトップ20.5%）は、`ETag` ヘッダーを持たず、対応するリクエストの `If-Modified-Since` ヘッダーで渡されたのと同じ `Last-Modified` 値を含んでいることがわかりました。そのうち78.3%（デスクトップでは86%）が `304 Not Modified` というステータスでした。

モバイルのレスポンスの89.9%（デスクトップ86.1%）が、対応するリクエストの `If-None-Match` ヘッダーで渡された同じ `ETag` の値を含んでいました。また、`If-Modified-Since` ヘッダーが存在する場合は、`ETag` が優先されます。これらのうち、90.2%（デスクトップでは88.9%）が `304 Not Modified` というステータスでした。

日付文字列の妥当性

このドキュメントでは、タイムスタンプを伝えるためのキャッシュ関連のHTTPヘッダーについて説明してきました。

- `Date` レスポンスヘッダーは、リソースがクライアントに提供された日時を示します。
- `Last-Modified` レスポンスヘッダーは、リソースがサーバー上で最後に変更された日時を示します。
- `Expires` ヘッダーは、リソースがどのくらいの期間キャッシュ可能かを示すた

めに使用されます。

これら3つのHTTPヘッダーは、いずれもタイムスタンプを表すために日付形式の文字列を使用しています。日付形式の文字列は、RFC 2616[699]で定義されており、GMTタイムスタンプの文字列を指定する必要があります。 たとえば、以下のようになります。

```
> GET /index.html HTTP/2
> Host: www.example.org
> Accept: */*

< HTTP/2 200
< Date: Thu, 23 Jul 2020 03:14:17 GMT
< Cache-Control: max-age=600
< Last-Modified: Mon, 20 Jul 2020 11:43:22 GMT
```

無効な日付文字列はほとんどのブラウザで無視されるため、その文字列が提供されたレスポンスのキャッシュ可能性に影響を与えます。たとえば無効な`Last-Modified`ヘッダーは、その無効なタイムスタンプなしにキャッシュされているため、ブラウザはその後、そのオブジェクトに対して条件付きのリクエストを実行できません。

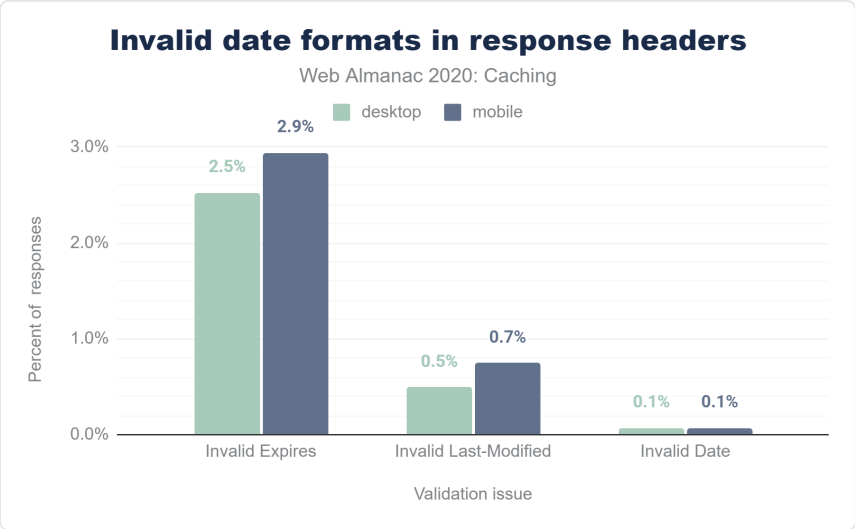


図20.10. レスポンスヘッダーの日付フォーマットが無効です。

699. https://tools.ietf.org/html/rfc2616#section-3.3.1

HTTPレスポンスヘッダーの`Date`は、ほとんどの場合、ウェブサーバによって自動的に生成されるため、無効な値は非常にまれです。同様に、`Last-Modified`ヘッダーでも、無効な値の割合は非常に低く（モバイルで0.75%、デスクトップで0.5%）なっています。しかし、非常に意外だったのは、`Expires`ヘッダーの2.94%が無効な日付フォーマットを使用していたことです（デスクトップでは2.5%）。

`Expires`ヘッダーの無効な使い方の例としては、以下のようなものがあります。

- 有効な日付フォーマットですが、GMT以外のタイムゾーンを使用しています。
- 0や-1などの数値
- `Cache-Control`ヘッダーで有効な値

無効な`Expires`ヘッダーの大きな原因のひとつは、人気のあるサードパーティから提供されたアセットで、日付/時間がESTタイムゾーンを使用しています。`Tue, 27 Apr 1971 19:44:06 EST`。 ブラウザによっては、堅牢性の観点からこの日付形式を理解して受け入れるものもありますが、そうであると仮定してはいけません。

`Vary`ヘッダー

これまでキャッシュエンティティは、レスポンスオブジェクトがキャッシュ可能か、またどのくらいの期間キャッシュできるかを判断する方法について説明してきました。しかしキャッシュエンティティーが行わなければならない、もっとも重要なステップの1つは、リクエストされたリソースがすでにキャッシュにあるかどうかを判断することです。これは簡単に見えるかもしれませんが、多くの場合URLだけでは判断できません。たとえば同じURLのリクエストでも、使用した圧縮方法（gzip、Brotliなど）が異なっていたり、異なるエンコーディング（XML、JSONなど）で返されたりすることがあります。

この問題を解決するために、キャッシング・エンティティがオブジェクトをキャッシュする際には、そのオブジェクトに一意の識別子（キャッシュ・キー）を与えます。オブジェクトがキャッシュにあるかどうかを判断する必要があるときは、キャッシュキーをルックアップとして使用してオブジェクトの存在を確認します。デフォルトでは、このキャッシュキーはオブジェクトの取得に使用された単純なURLですが、サーバは`Vary`レスポンスヘッダーを含めることによってキャッシュキーにレスポンスの他の属性（圧縮方法など）を含めるようにキャッシュエンティティに指示できます。`Vary`ヘッダーは、URL以外の要素に基づいて、オブジェクトのバリエーションを識別します。

`Vary`レスポンスヘッダーは、1つまたは複数のリクエストヘッダーの値をキャッシュキーに追加するようにブラウザに指示します。この`Vary: Accept-Encoding`のもっとも一般的な例、ブラウザは、異なる`Accept-Encoding`リクエストヘッダー値（例: `gzip`, `br`,

`deflate`）に基づいて、同じオブジェクトを異なるフォーマットでブラウザがキャッシュすることになります。

キャッシング用のエンティティは、HTMLファイルのリクエストを送信し、gzip形式のレスポンスを受け入れることを示します。

```
> GET /index.html HTTP/2
> Host: www.example.org
> Accept-Encoding: gzip
```

サーバーはオブジェクトで応答し、送信するバージョンが`Accept-Encoding`リクエストヘッダーの値を含むべきであることを示します。

```
< HTTP/2 200 OK
< Content-Type: text/html
< Vary: Accept-Encoding
```

この単純化された例では、キャッシング エンティティは、URLと`Vary`ヘッダーの組み合わせを使用してオブジェクトをキャッシュします。

もうひとつの一般的な値は`Vary: Accept-Encoding, User-Agent`で、これはクライアントに対して、`Accept-Encoding`と`User-Agent`の両方の値をキャッシュキーに含めるように指示します。しかし共有プロキシやCDNについて議論する場合、`Accept-Encoding`以外の値を使用すると、キャッシュが希釈または断片化されキャッシュから提供されるトラフィックの量が、減少する可能性があるため問題となることがあります。たとえば、`User-Agent`には数千の異なる種類があるため、CDNがあるオブジェクトの多くの異なるバリエーションをキャッシュしようとすると、ほとんど同一の（あるいは実際に同一の）キャッシュされたオブジェクトでキャッシュがいっぱいになってしまう可能性があります。これは非常に非効率的であり、CDN内でのキャッシングが最適でなくなり、キャッシュヒット数の減少とレイテンシーの増大を招くことになります。

一般的に、キャッシュを変更するのは、そのヘッダーに基づいてクライアントに別のコンテンツを提供する場合だけにすべきです。

HTTPレスポンスの43.4%がVaryヘッダーを使用しており、そのうち84.2%が`Cache-Control`ヘッダーを含んでいます。

下のグラフは、Varyヘッダーの上位10個の値の人気度を示しています。`User-Agent`が10.7%、`Origin`（CORSの処理に使用）が8%、`Accept`が4.1%となっており、`Vary`の

使用量の約92%を占めています。

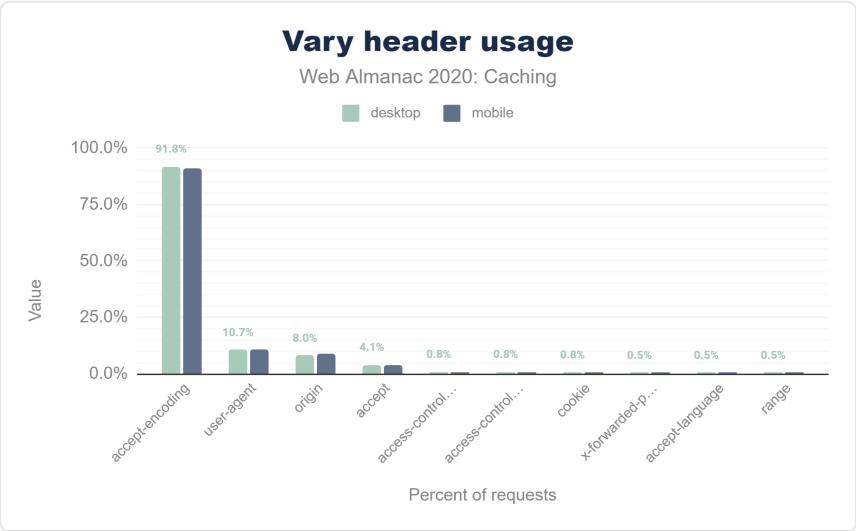


図20.11. `Vary` *ヘッダーの使用方法。*

キャッシュ可能なレスポンスにCookieを設定する

レスポンスがキャッシュされると、そのレスポンス ヘッダーの全セットがキャッシュされたオブジェクトに含まれます。Chromeでキャッシュされたレスポンスを DevToolsで検査すると、レスポンス・ヘッダーが表示されるのはこのためです。

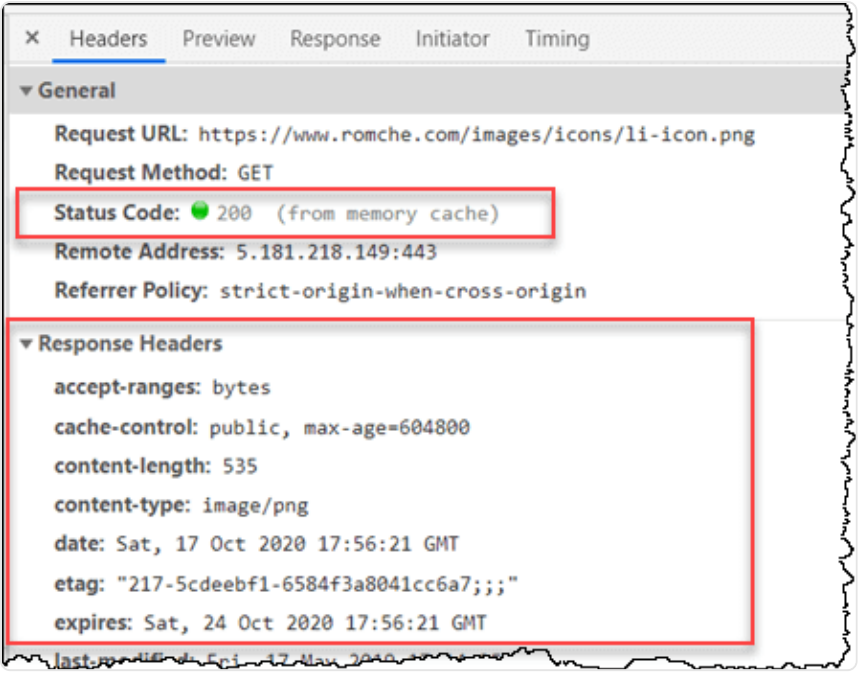


図20.12. キャッシュされたリソースのためのChrome Dev Toolsです。

しかし、レスポンスに`Set-Cookie`がある場合はどうなるでしょうか？RFC 7234 Section 8[700]によると、`Set-Cookie`レスポンスヘッダーの存在はキャッシュを阻害しないそうです。つまり、キャッシュされたエントリーに`Set-Cookie`レスポンスヘッダーが含まれている可能性があるということです。このRFCでは、レスポンスのキャッシュ方法を制御するために、適切な`Cache-Control`ヘッダーを設定することを推奨しています。

私のリクエストに応じてサーバーから私に送信された`Set-Cookie`レスポンスヘッダーには、私のCookieが含まれていることが明らかなので、私のブラウザがそれらをキャッシュしても問題はありません。しかし、私とサーバーの間にCDNがある場合、サーバーはCDNに対して、レスポンスをCDN自体にキャッシュすべきでないことを示さなければなりません。そうすれば、私向けのレスポンスがキャッシュされないで、他のユーザーに（私の`Set-Cookie`ヘッダーを含めて）提供されることになります。

たとえば、ログインCookieやセッションCookieがCDNのキャッシュされたオブジェクトに存在する場合、そのCookieは他のクライアントによって再利用される可能性があります。これを避けるための主な方法は、サーバーが`Cache-Control: private`ディレクティブを送信することです。

700. https://tools.ietf.org/html/rfc7234#section-8

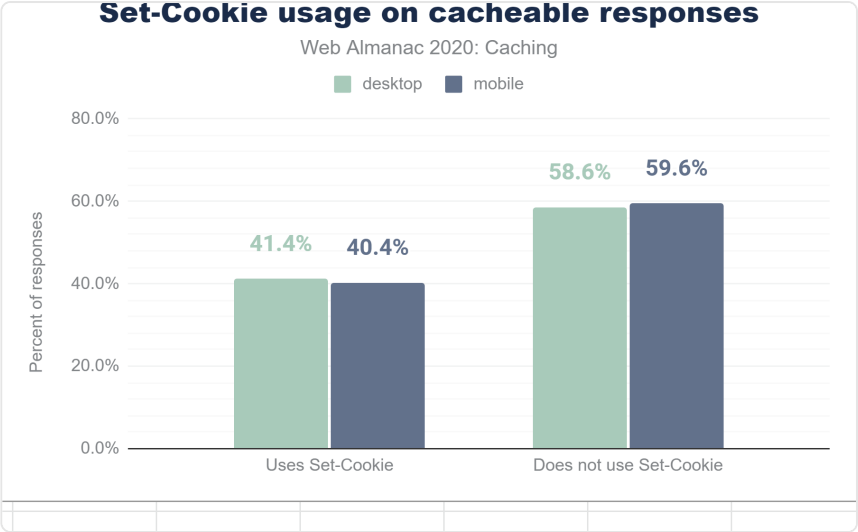


図20.13. キャッシュ可能なレスポンスの`Set-Cookie`。

キャッシュ可能なモバイルレスポンスのうち、40.4%が`Set-Cookie`ヘッダーを含んでいます。これらのレスポンスのうち、`private`ディレクティブを使用しているのは4.9%。残りの95.1%（1億9,860万のHTTPレスポンス）は、少なくとも1つの`Set-Cookie`レスポンスヘッダーを含んでおり、CDNなどのパブリックキャッシュサーバでキャッシュできます。これは、キャッシュ機能とCookieがどのように共存しているのか、理解されていないことを示しているのかもしれません。

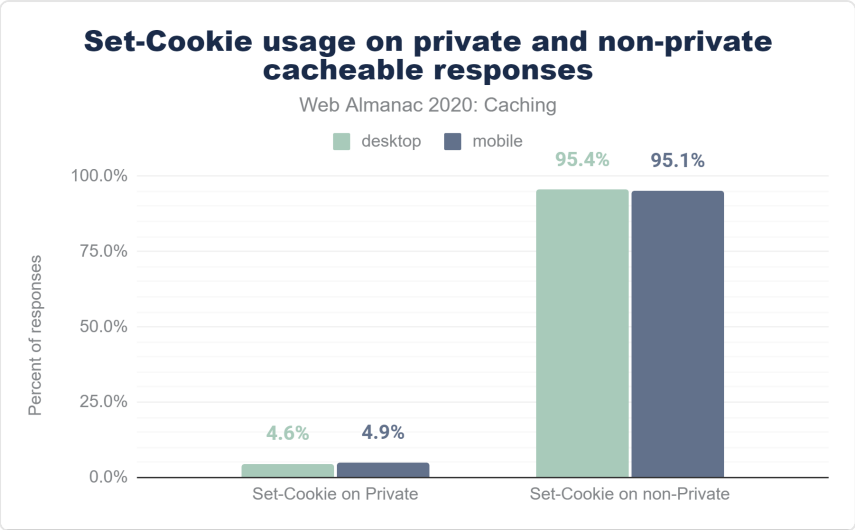


図20.14. `private` および非プライベートのキャッシュ可能なレスポンスにおける `Set-Cookie`。

サービス・ワーカー

サービスワーカーとはHTML5の機能の1つで、フロントエンドの開発者がWebページの通常のリクエスト／レスポンスの流れの外で実行されるべきスクリプトを指定し、メッセージを介してWebページと通信することを可能にします。サービスワーカーの一般的な用途としては、バックグラウンドでの同期やプッシュ通知、そして当然ながらキャッシュが挙げられます。

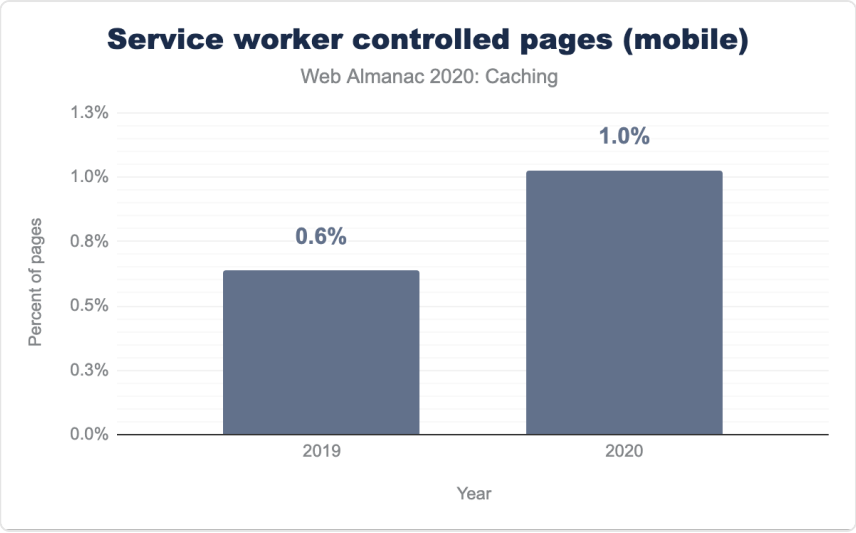


図20.15. サービスワーカーの成長は、2019年からのページを制御しています。

採用率はウェブサイトの1%にとどまっていますが、2019年7月以降、着実に増加しています。Progressive Web Appの章では、上のグラフで1回しかカウントされていない人気サイトでの利用により、このグラフが示す以上に多く利用されていることなど詳しく説明しています。

サービスワーカーを使わないサイト	サービスワーカーを使っているサイト	総サイト数
6,225,774	64,373	6,290,147

図20.16. サービス・ワーカーを利用しているウェブサイトの数

上の表では、全6,290,147サイトのうち、64,373サイトがサービスワーカーを導入していることがわかります。

HTTPサイト	HTTPSサイト	合計サイト
1,469	62,904	64,373

図20.17. HTTP/HTTPSでサービスワーカーを利用しているウェブサイトの数。

これをHTTPとHTTPSで分けてみると、さらに興味深いことがわかります。HTTPSはサービスワーカーを使用するための必須条件であるにもかかわらず、以下の表によると、サービス

ワーカーを使用しているサイトのうち1,469件がHTTPで提供されています。

どのようなコンテンツをキャッシングするのか？

これまで見てきたように、キャッシュ可能なリソースは一定期間ブラウザに保存され、その後のリクエストで再利用できるようになります。

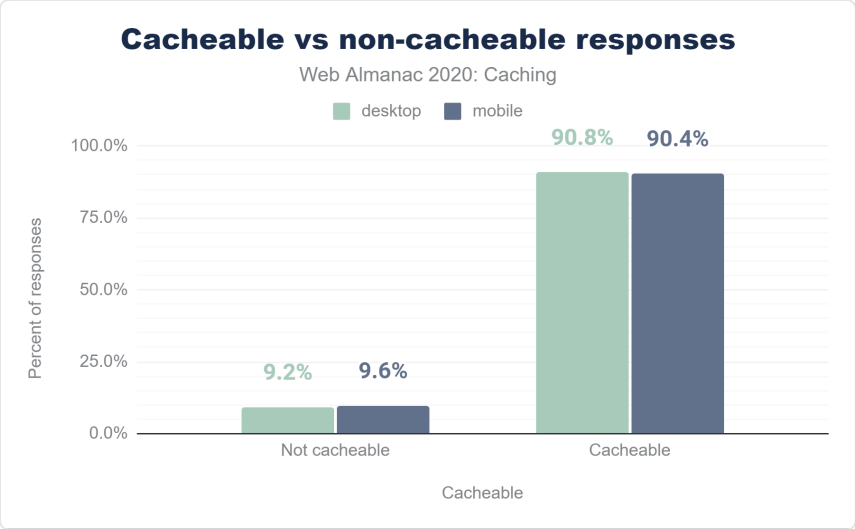


図20.18. キャッシュ可能なレスポンスとキャッシュ不可能なレスポンスの分布。

すべてのHTTP(S)リクエストにおいて、90.4%のレスポンスがキャッシュ可能であると考えられています（つまり、キャッシュの保存が許可されています）。残りの9.6%のレスポンスは、ブラウザのキャッシュに保存できません。

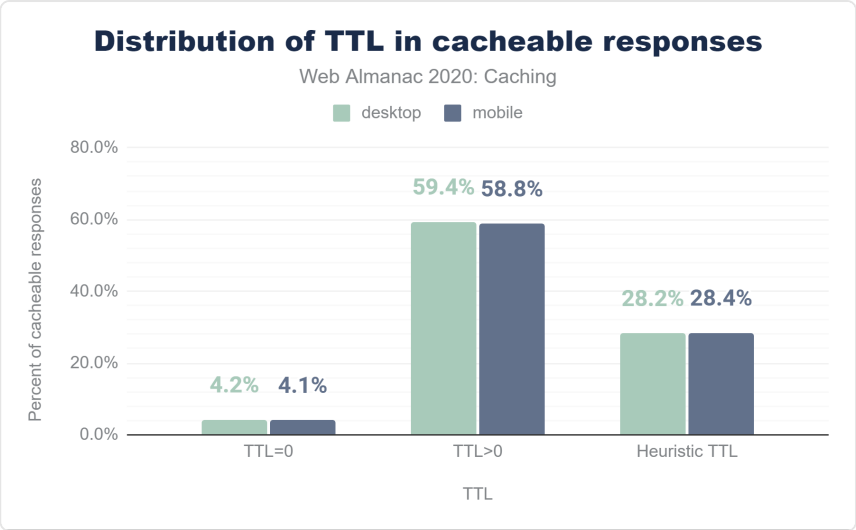


図20.19. キャッシュ可能なレスポンスにおけるTTLの分布。

もう少し掘り下げてみると4.1%のリクエストのTTLが0秒で、この場合オブジェクトはキャッシュに追加されますが、すぐに古いものとしてマークされ再検証が必要になります。28.4%は`Cache-Control`や`Expires`などのヘッダーがないためヒューリスティックにキャッシュされ、58.8%は0秒以上キャッシュされています。

以下の表は、モバイルリクエストのキャッシュTTL値をタイプ別にまとめたものです。

タイプ	10	25	50	75	90
Audio	6	6	240	744	8,760
CSS	24	24	720	8,760	8,760
Font	720	8,760	8,760	8,760	8,760
HTML	0	3	336	8,760	8,600
Image	6	168	720	8,760	8,766
その他	0	3	31	336	23,557
Script	0	4	720	8,760	8,760
Text	0	1	6	24	8,760
Video	6	336	336	336	8,674
XML	1	24	24	24	720

図20.20. リソースタイプ別のモバイルキャッシュTTLパーセンタイル。

ほとんどのTTLの中央値は高いのですが、低いパーセンタイルでは、キャッシュの機会を逃しているものがいくつかあります。たとえば画像のTTLの中央値は720時間（1か月）ですが、25th パーセンタイルはわずか168時間（1週間）で、10th パーセンタイルはわずか数時間にまで落ち込んでいます。これをフォントと比較すると、フォントのTTLは8,760時間（1年）と非常に高く25thパーセンタイルまであり、10thパーセンタイルでも1か月のTTLを示しています。

コンテンツタイプ別のキャッシュ可能性を下の図で詳しく見てみると、フォント、ビデオやオーディオ、CSSファイルは100%近くブラウザキャッシュされている一方で（これらのファイルは一般的に静的であるため、これは理にかなっている）、全HTMLレスポンスの約3分の1はキャッシュ不可能であると考えられます。

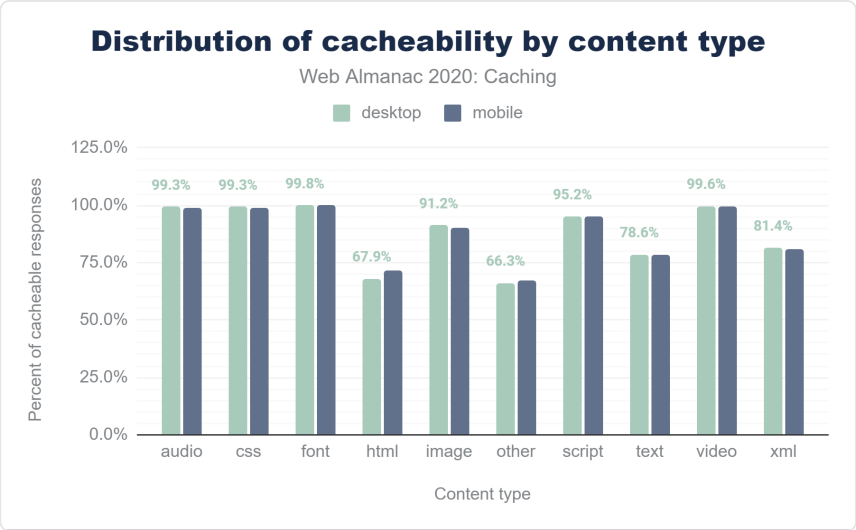


図20.21. コンテンツタイプ別のキャッシュ可能性の分布

また、デスクトップでは、imagesの10.1%、scriptsの4.9%がキャッシュされません。これらのオブジェクトの中には静的なものもあり、より高い割合でキャッシュできる可能性があるため、ここには改善の余地があると思われます。覚えておいてください： *できる限り多くのものを、できる限り長くキャッシュしましょう！*。

キャッシュのTTLはリソースの年齢と比べてどうなのか？

これまでに、サーバーがクライアントにキャッシュ可能なコンテンツをどのように伝え、どのくらいの期間キャッシュされているかについて説明してきました。キャッシュルールを設計する際には、提供しているコンテンツがどれくらい古いかを理解することも重要です。

クライアントへ返信するレスポンスヘッダーに指定するキャッシュTTLを選択する際には、自問してみてください。「どのくらいの頻度でこれらの資産を更新しているか」と「コンテンツの感度はどうか」。たとえば、ヒーロー画像が頻繁に変更されない場合は、非常に長いTTLでキャッシュできます。対照的にJavaScriptファイルが頻繁に変更されるのであれば、ユニークなクエリ文字列でバージョン管理し長いTTLをキャッシュするか、あるいはもっと短いTTLでキャッシュすべきでしょう。

以下のグラフは、コンテンツタイプ別のリソースの相対的な古さを示しています。

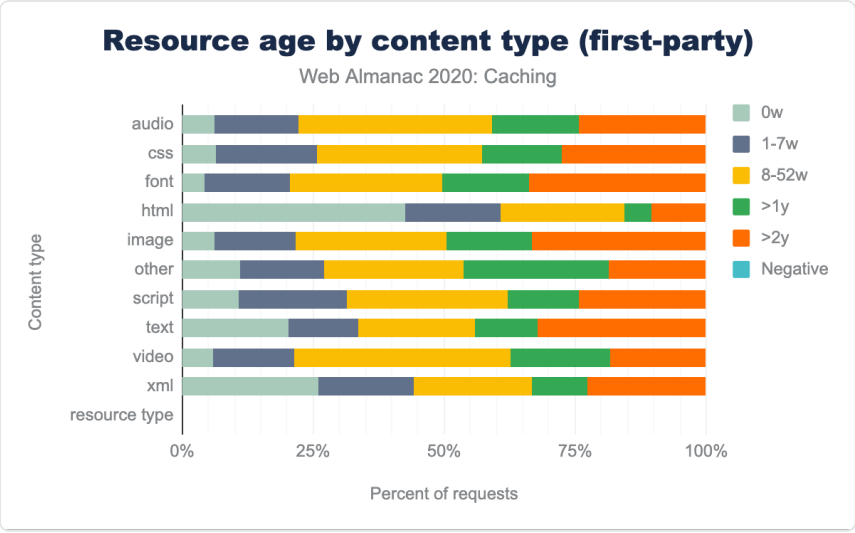


図20.22. コンテンツタイプ別のリソースエイジ（ファーストパーティ）。

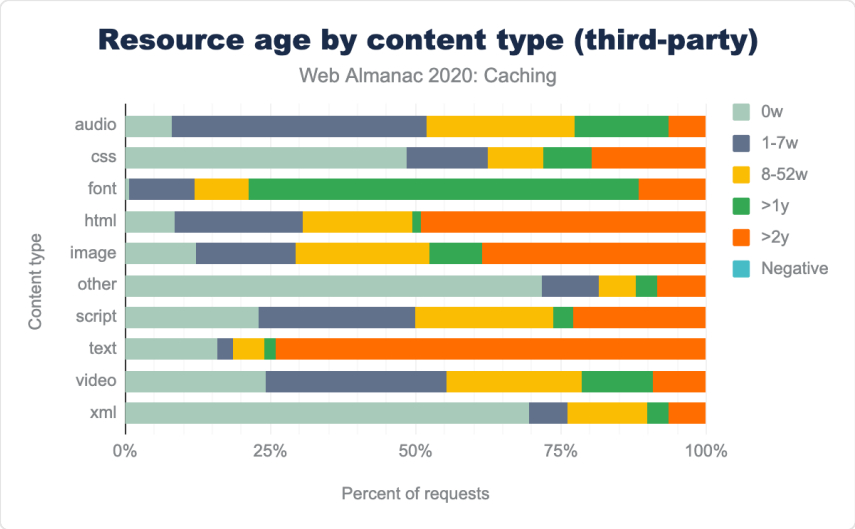


図20.23. コンテンツタイプ別のリソースエイジ（サードパーティ）。

このデータの中で興味深い観察結果があります。

- ファーストパーティのHTMLは、リソース年数がもっとも短いコンテンツタイプで、41.1%のリクエストが1週間未満となっています。その他のほとんどのコン

テンツタイプでは、サードパーティコンテンツの方がファーストパーティコンテンツよりもリソースエイジが小さい。

- ウェブ上のファーストパーティコンテンツのうち、8週間以上経過しているものには、images（78.9%）、scripts（68.7%）、CSS（74.9%）、Webフォント（80.4%）、autio（78.2%）、video（79.3%）など、伝統的にキャッシュ可能なオブジェクトがあります。
- ファーストパーティとサードパーティのリソースでは、1週間以上経過しているものがあり、大きな差があります。ファーストパーティ製CSSの93.4%が1週間以上経過しているのに対し、サードパーティ製CSSでは48.0%が1週間以上経過している。

リソースのキャッシュ可能性とその年齢を比較することで、TTLが適切か低すぎるかを判断できます。

たとえば、2020年10月18日に配信されたリソースの最終更新日は2020年8月30日であり、配信時には1か月以上経過していることになります。これは頻繁に変更されないオブジェクトであることを示しています。しかし、`Cache-Control`ヘッダーによると、ブラウザは86,400秒（1日）しかキャッシュできないことになっています。これは、ブラウザが条件付きでも再リクエストする必要がないように、TTLを長くすることが適切なケースです。とくに、ユーザーが数日間に何度も訪れるようなウェブサイトの場合はそうです。

```
> HTTP/1.1 200
> Date: Sun, 18 Oct 2020 19:36:57 GMT
> Content-Type: text/html; charset=utf-8
> Content-Length: 3052
> Vary: Accept-Encoding
> Last-Modified: Sun, 30 Aug 2020 16:00:30 GMT
> Cache-Control: public, max-age=86400
```

ウェブ上で提供されているモバイルリソースのうち60.2%が、コンテンツの年齢に比べて短すぎると思われるキャッシュTTLを持っていました。さらに、TTLと年齢の差の中央値は25日で、これもキャッシュ不足が顕著であることを示しています。

クライアント	ファーストパーティ	サードパーティ	全体
デスクトップ	61.6%	59.3%	60.7%
モバイル	61.8%	57.9%	60.2%

図20.24. TTLが短いリクエストの割合。

上の表でファーストパーティとサードパーティに分けてみると、ファーストパーティのリソースの約3分の2（61.8%）がTTLを長くすることでメリットを得られることがわかります。このことから、キャッシュ可能なものにとくに注意を払い、キャッシュを正しく設定していることを確認する必要があります。

キャッシングの機会を特定する

GoogleのLighthouse[701]ツールでは、ウェブページに対する一連の監査を実行でき、cache policy audit[702]では、サイトに追加のキャッシュが必要かどうかを評価します。これは、コンテンツの年齢（`Last-Modified`ヘッダーによる）をキャッシュのTTLと比較し、リソースがキャッシュから提供される確率を推定することによって行われます。スコアに応じて、キャッシュを推奨する結果が表示され、キャッシュ可能な特定のリソースのリストが表示されます。

701. https://developers.google.com/web/tools/lighthouse
702. https://developers.google.com/web/tools/lighthouse/audits/cache-policy

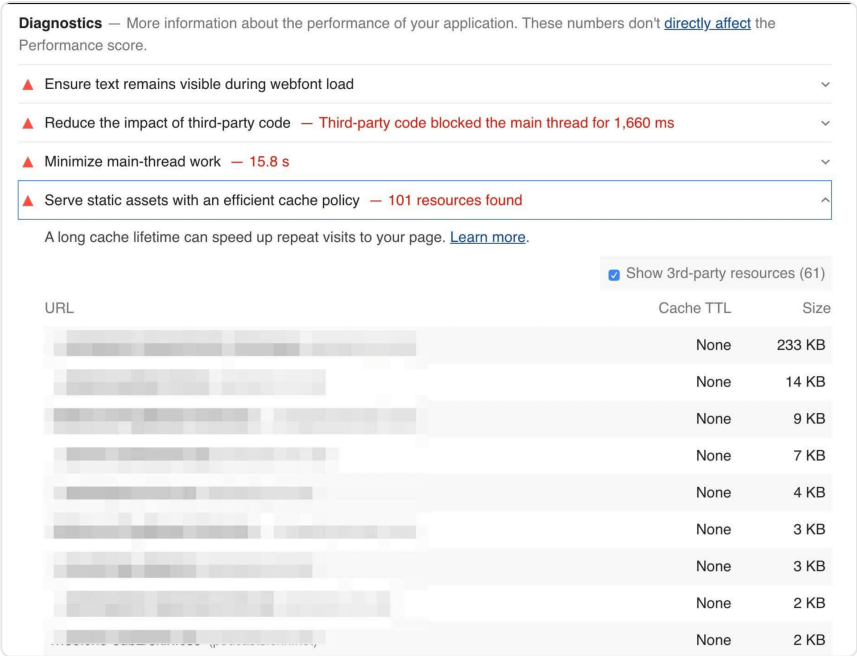


図20.25. キャッシュ・ポリシーの改善の可能性を示すライトハウス・レポート。

Lighthouseでは、各監査に対して0%から100%の範囲でスコアを計算し、それらのスコアを総合的なスコアに反映させています。キャッシングのスコアは、潜在的なバイト削減量に基づいています。Lighthouseの結果を見ると、どれだけのサイトがキャッシュポリシーをうまく活用しているかがわかります。

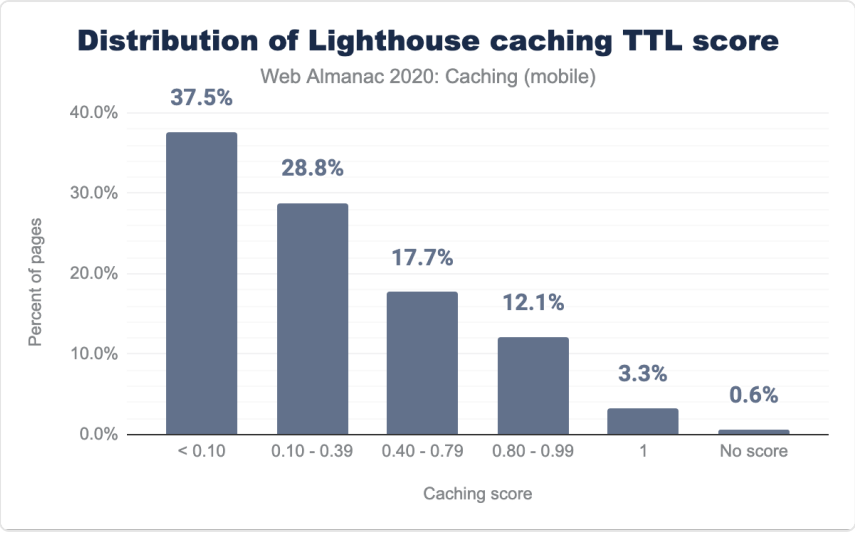


図20.26. LighthouseキャッシングのTTLスコアの分布。

100%のスコアを獲得したサイトはわずか3.3%であり、大多数のサイトがキャッシュの最適化によって恩恵を受けることができると考えられます。約3分の2のサイトが40%を下回り、約3分の1のサイトが10%を下回りました。このことから、かなりの量のキャッシュが不足しており、ネットワーク上で過剰なリクエストやバイトが処理されていることがわかります。

また、Lighthouseはより長いキャッシュポリシーを有効にすることで、繰り返し表示する際に何バイト節約できるかを示しています。

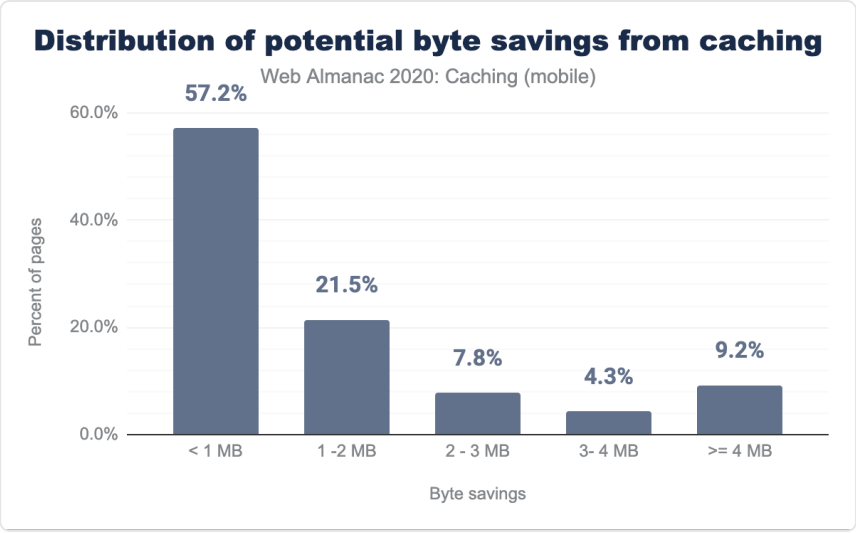


図20.27. Lighthouseキャッシング監査による潜在的なバイト削減量の分布。

キャッシュを追加することで恩恵を受けることができるサイトのうち、5分の1以上のサイトが2MB以上もページの重量を減らすことができます。

結論

キャッシングは、ブラウザやプロキシ、CDNなどの仲介者がウェブコンテンツを保存し、エンドユーザーに提供できる非常に強力な機能です。往復の時間を短縮し、コストのかかるネットワークリクエストを最小限に抑えることができるため、パフォーマンス面でのメリットが大きいのです。

キャッシングは非常に複雑なテーマであり、開発サイクルの後半まで放置され、最後の最後で追加されることもよくあります（サイト開発者が設計中に最新バージョンのサイトを見たいという要求）。またキャッシュルールは一度定義されると、サイトの基本的なコンテンツが変更されても、変更されないことが多い。慎重に検討することなく、デフォルト値が選ばれることもよくあります。

オブジェクトを正しくキャッシュするために、キャッシュされたエントリを検証するだけでなく、新鮮さを伝えることができる多くのHTTPレスポンスヘッダーがあり、`Cache-Control`ディレクティブは非常に大きな柔軟性とコントロールを提供します。

一般的にキャッシュできないと考えられているオブジェクトタイプやコンテンツの多くは、実際にキャッシュできます（覚えておいてください：*できるだけ多くのキャッシュをしまし*

ょう！）。また、多くのオブジェクトはキャッシュされる期間が短すぎるため、繰り返しリクエストや再検証が必要になります（覚えておいてください：できるだけ多くのキャッシュをしましょう！）。しかしウェブサイトの開発者は、コンテンツを過剰にキャッシュすることで、ミスの機会が増えることに注意しなければなりません。

サイトがCDNで提供されることを意図している場合、サーバーの負荷を軽減しエンドユーザーに迅速な応答を提供するために、CDNでのキャッシングの機会を増やすことを、Cookieなどの個人情報を誤ってキャッシングしてしまうことの関連するリスクとともに考慮する必要があります。

しかし強力で複雑であることは、必ずしも難しいことではありません。他のほとんどのものと同様にキャッシュはルールによって制御されており、キャッシュ可能性とプライバシーのベストミックスを提供するために、かなり簡単に定義できます。サイトを定期的に監査し、キャッシュ可能なリソースが適切にキャッシュされていることを確認することをオススメします。

著者

Rory Hewitt

@roryhewitt3 roryhewitt roryhewitt https://romche.com

Akamai[703]のエンタープライズアーキテクトで、パフォーマンスに情熱を注いでいます。イギリスからの帰国子女で、サンフランシスコに20年以上住んでいます。余暇には、長距離アドベンチャーバイク、スノーボーダー、ボクサー/カラテカとして活躍しています。彼はトラブルメーカーとして知られていることが好きです。そして何より、父親であり、夫であり、猫のルナの飼い主でもあるのです。

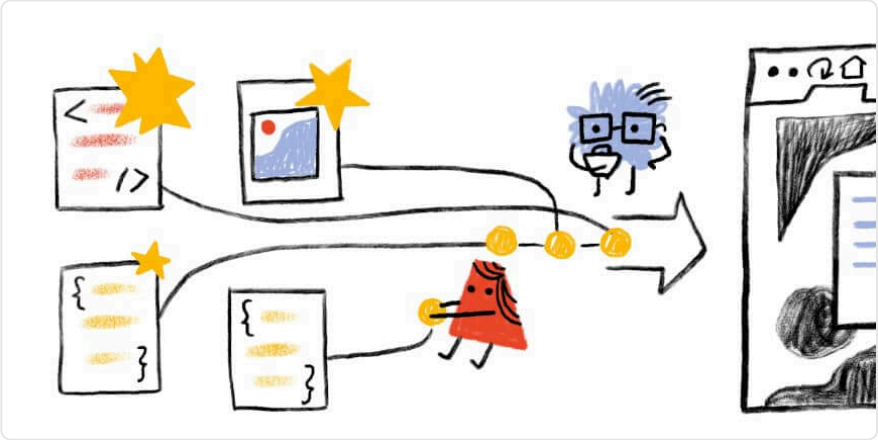
Raghu Ramakrishnan

raghuramakrishnan71

Tata Consultancy Services[704]のエンタープライズアーキテクトで、公共部門の大規模なデジタルトランスフォーメーションプログラムに従事しています。テクノロジーに興味があり、特にパフォーマンスエンジニアリングに関心があります。旅行好きで、天文学、歴史、生物学、医学の進歩に興味を持っています。バガヴァッド・ギーターの第2章47節 “karmaṇy-evādhikāras te mā phaleṣhu kadāchana “を強く信奉している。

703. https://www.akamai.com/
704. https://www.tcs.com/

部 IV 章 21
リソースヒント



***Leonardo Zizzamia** によって書かれた。*
Jessica Nicolet、 Patrick Meenan、 Giovanni Puntil、 Minko Gechev と Will Klancnik によってレビュ。
Katie Hempenius による分析。
Shane Exterkamp 編集。
Sakae Kotaro によって翻訳された。

序章

過去10年間で、リソースヒント[705]は、開発者がページのパフォーマンスを向上させ、その結果としてユーザー体験を向上させるために不可欠なプリミティブとなりました。

リソースのプリロードや、ブラウザによるインテリジェントな優先順位付けは、実は2009年にIE8がプリローダー[706]と呼ばれるものを使って始めたことでした。IE8には、HTMLパーサーに加えて、ネットワーク リクエストを開始する可能性のあるタグ (`<script>`、`<link>`、`<img>`) をスキャンする軽量のルックアヘッド プリローダーが搭載されていました。

その後の数年間、ブラウザのベンダーがより多くの作業を行い、それぞれがリソースの優先

705. https://www.w3.org/TR/resource-hints/
706. https://speedcurve.com/blog/load-scripts-async/

順位の付け方について独自の特別なソースを追加しました。しかし、ブラウザだけでは限界があることを理解することが大切です。開発者である私たちはリソースヒントをうまく利用してリソースの優先順位を決定したり、ページのパフォーマンスをさらに向上させるためにどのリソースをフェッチしたり、前処理したりするべきかを判断することでこの限界を克服できます。

とくに、昨年達成したリソースのヒントをいくつか挙げます。

- CSS-Tricks[707] Webフォントが3Gの初回レンダリングでより速く表示されるようになりました。
- リソースヒントを使用したWix.com[708]では、FCPで10%の改善が見られました。
- Ironmongerydirect.co.uk[709]は、preconnectを使用して、製品画像の読み込みを中央値で400ms、95パーセンタイルで1秒以上改善しました。
- Facebook.com[710]は、ナビゲーションを高速化するためにpreloadを使用していました。

今日、ほとんどのブラウザでサポートされている主なリソースヒントを見てみましょう。`dns-prefetch`、`preconnect`、`preload`、`prefetch`、そしてネイティブの遅延ローディングです。

個々のヒントに取り組む際には、WebVitals[711]、Perfume.js[712]などのライブラリや、Web Vitalsのメトリクスをサポートするその他のユーティリティを使用して、常に現場での前後の影響を測定することをオススメします。

`dns-prefetch`

`dns-prefetch` は、指定されたドメインのIPアドレスを事前解決するのに役立ちます。利用可能なもっとも古い[713]リソースヒントとして、`preconnect` と比較して最小限のCPUとネットワークリソースを使用し、ブラウザがDNS解決のための 1秒以上[714]の「最悪のケース」の遅延を経験しないことに役立ちます。

```
<link rel="dns-prefetch" href="https://www.googletagmanager.com/">
```

707. https://www.zachleat.com/web/css-tricks-web-fonts/
708. https://www.youtube.com/watch?v=4QqlGgF8Y2I&t=1469
709. https://andydavies.me/blog/2019/03/22/improving-perceived-performance-with-a-link-rel-equals-preconnect-http-header/
710. https://engineering.fb.com/2020/05/08/web/facebook-redesign/
711. https://github.com/GoogleChrome/web-vitals
712. https://github.com/zizzamia/perfume.js
713. https://caniuse.com/link-rel-dns-prefetch
714. https://www.chromium.org/developers/design-documents/dns-prefetching

`dns-prefetch`を使用する際には注意が必要です。軽量であっても、ブラウザが許容する同時進行のDNSリクエスト数の制限を簡単に超えてしまうからです（Chromeにはまだ6[715]の制限があります）。

`preconnect`

`preconnect`は、指定されたドメインのIPアドレスを事前に解決し、TCP/TLS接続を開くのに役立ちます。`dns-prefetch`と同様に、クロスオリジンのドメインに使用され、最初のページ読み込み時に使用されるリソースをブラウザがウォームアップするのに役立ちます。

```
<link rel="preconnect" href="https://www.googletagmanager.com/">
```

`preconnect`を使用する際には注意が必要です。

- もっとも頻度の高い、重要なリソースのみを温めます。
- 最初の負荷をかける際に、使用したオリジンのウォームアップが遅すぎないようにする。
- CPUやバッテリーのコストが、かかる場合があるので、3つ以下のoriginで使用してください。

最後に、`preconnect`はInternet ExplorerやFirefox[716]では利用できませんので、予備として`dns-prefetch`を利用することを強くオススメします。

`preload`

`preload`ヒントは、早期のリクエストを開始します。これはパーサーが、発見するのが遅れるような重要なリソースを読み込むのに便利です。

```
<link rel="preload" href="style.css" as="style">
<link rel="preload" href="main.js" as="script">
```

`preload`することを意識します。なぜなら他のリソースのダウンロードを遅らせる可能性があるため、Largest Contentful Paint（LCP[717]）の改善に役立つ、もっとも重要なものにのみ

715. https://source.chromium.org/chromium/chromium/src/+/master:net/dns/host_resolver_manager.cc;l=353
716. https://caniuse.com/?search=preconnect
717. https://web.dev/i18n/ja/lcp/

使用してください。 また、Chromeで使用した場合、`preload`リソースを過剰に優先する傾向があり、他の重要なリソースよりもpreloadを優先的に処理する可能性があります。

最後に、HTTPレスポンスヘッダーで使用された場合、一部のCDNは`preload`を自動的にHTTP/2 pushに変えてしまい、キャッシュされたリソースを過剰にプッシュしてしまうことがあります。

`prefetch`

`prefetch`ヒントを使用すると、次のナビゲーションで使用されると予想される低優先度のリクエストを開始できます。このヒントは、リソースをダウンロードして、あとで使用するためHTTPキャッシュにドロップします。重要なのは、`prefetch`はリソースの実行やその他の処理を行わず、実行するにはページが`<script>`タグでリソースを呼び出す必要があるということです。

```
<link rel="prefetch" as="script" href="next-page.bundle.js">
```

リソースの予測ロジックを実装する方法はさまざまで、ユーザーのマウスの動きや一般的なユーザーフロー/ジャーニーなどの信号に基づいたり、機械学習の上に両者を組み合わせたりすることもできます。

使用しているCDNのHTTP/2の優先順位付けの品質[718]によっては、`prefetch`の優先順位付けがパフォーマンスを向上させることもあれば、`prefetch`のリクエストを過剰に優先させ、最初のロードのために重要な帯域を奪い、パフォーマンスを低下させることもあることに注意してください。使用しているCDNを再確認し、Andy Davies氏の[719]記事で紹介されているベストプラクティスを考慮に入れてください。

ネイティブの遅延読み込み

ネイティブの遅延読み込み[720]のヒントは、画面外の画像やiframeの読み込みを延期するためのネイティブブラウザAPIです。これを使用することで、最初のページロード時に必要のないアセットはネットワークリクエストを開始しないため、データ消費量が削減され、ページパフォーマンスが向上します。

```
<img src="image.png" loading="lazy" alt="…" width="200"
```

718. https://github.com/andydavies/http2-prioritization-issues#current-status
719. https://andydavies.me/blog/2020/07/08/rel-equals-prefetch-and-the-importance-of-effective-http-slash-2-prioritisation/
720. https://web.dev/browser-level-image-lazy-loading/

```
height="200">
```

留意点:Chromiumの遅延読み込みのしきい値ロジックの実装は、これまで保守的[721]で、画面外の制限を3000pxに保っていました。昨年この制限は積極的にテストされ、開発者の期待に沿うように改善され、最終的にしきい値を1250pxに変更しました。また、ブラウザ間の標準はなく[722]、ウェブ開発者がブラウザの提供するデフォルトしきい値を上書きする機能はまだありません。

リソースのヒント

HTTP Archiveをもとに、2020年のトレンド分析に飛び込み、前回の2019年のデータセットと比較してみましょう。

ヒントの採用

ますます多くのウェブページがメインリソースのヒントを使用しており、2020年にはデスクトップとモバイルの間で一貫した採用が行われると見られています。

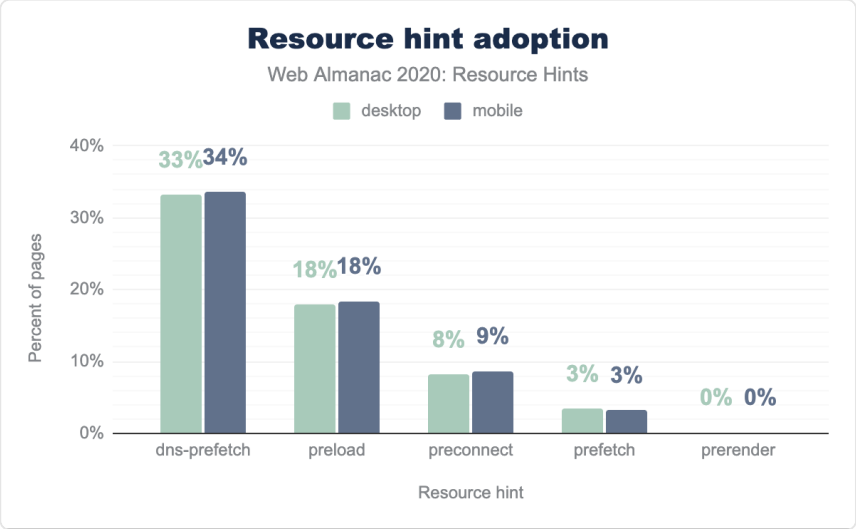


図21.1. リソースヒントの採用

721. https://web.dev/browser-level-image-lazy-loading/#distance-from-viewport-thresholds
722. https://github.com/whatwg/html/issues/5408

`dns-prefetch` の採用率が33%と他のリソースヒントに比べて相対的に高いのは、2009年にはじめて登場し主要なリソースヒントの中でもっとも広くサポートされていることから、当然のことと言えるでしょう。

2019[723]と比較すると、`dns-prefetch` はDesktopの採用率が4%増加していました。また、`preconnect` でも同様の増加が見られました。すべてのヒントの中でもっとも大きな伸びを示した理由の1つは、Lighthouse audit[724]がこの問題に関して明確で有益なアドバイスをしていることです。今年のレポートからは、最新のデータセットがLighthouseの推奨事項に対してどのようなパフォーマンスを示すかについても紹介しています。

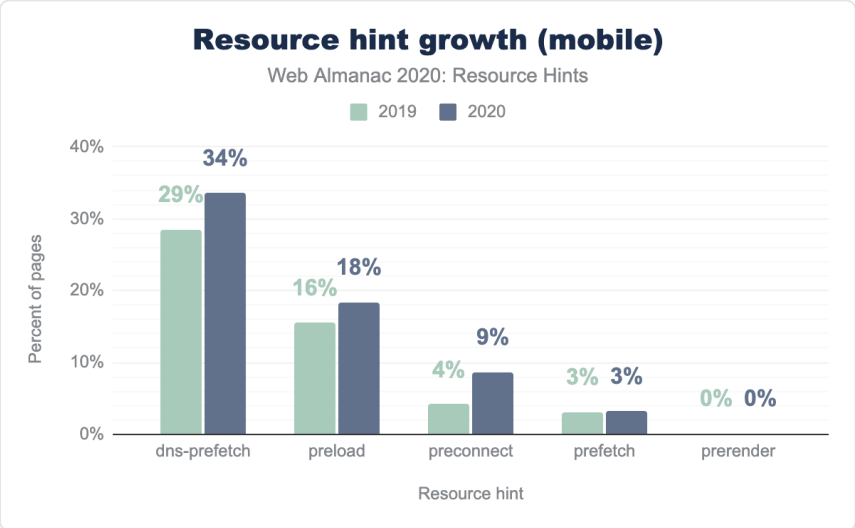


図21.2. リソースヒントの採用2019年対2020年

`preload` の使用率は2019年から2%の増加にとどまり、伸び悩んでいます。これは、もう少し多くの指定を必要とすることが一因と考えられます。`dns-prefetch` や `preconnect` を使用するにはドメインが必要なだけですが、`preload` を使用するにはリソースを指定する必要があります。`dns-prefetch` と `preconnect` は悪用される可能性があるものの、それなりにリスクが低いのに対し、`preload` は間違った使い方をすると、実際パフォーマンスにダメージを与える可能性がはるかに大きいのです。

`prefetch` はデスクトップの3%のサイトで使用されており、リソースヒントの中でもっとも使用されていないものです。この使用率の低さは、`prefetch` が現在のページロードではなく、その後のページロードを改善するために有用であるという事実から説明できるかもしれません。そのため、サイトがランディングページや最初に表示されるページのパフォーマ

723. https://almanac.httparchive.org/ja/2019/resource-hints#resource-hints
724. https://web.dev/i18n/ja/uses-rel-preconnect/

ンスを向上させることだけに注力している場合には、見落とされてしまいます。今後数年間で、後続のページのエクスペリエンスを向上させるために何を測定すべきかがより明確に定義されれば、チームが到達すべき明確なパフォーマンス品質目標を持って `prefetch` の採用を優先するのに役立つでしょう。

ページあたりのヒント

全体的に開発者はリソースヒントの上手な使い方を学んでおり、2019[725]と比較すると、`preload`、`prefetch`、`preconnect` の使用率が向上していることがわかります。preloadやpreconnectなどの高価な操作では、デスクトップでの使用量の中央値が2から1に減少しました。一方、`prefetch` で優先度の低い将来のリソースをロードする場合は、ページあたりの中央値が1から2に増加しました。

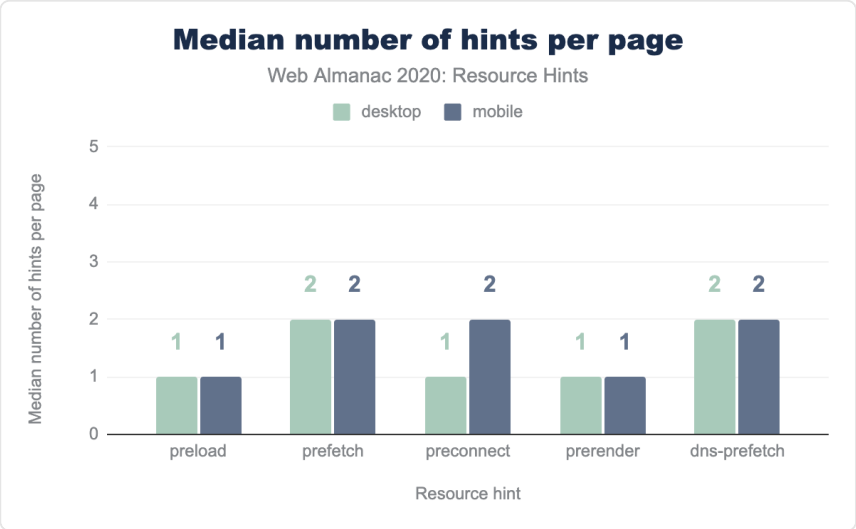


図21.3. 1ページあたりのヒント数の中央値

リソースヒントは、選択的に使用されるときにもっとも効果的です（「すべてが重要なとき、何も重要ではない」）。どのリソースが重要なレンダリングの改善に役立つのか将来のナビゲーションの最適化に役立つのかをより明確に定義することで、リソースの優先順位付けの一部を変更し、最初ユーザーにもっとも役立つもののために帯域幅を解放することで、`preconnect` の使用から `prefetch` へと焦点を移すことができます。

あるページでは、ヒントを動的に追加し、20k以上の新しいプリロードを作成する無限ループを引き起こしていることが判明しました。これは `preload` ヒントの誤用を阻止していま

725. https://almanac.httparchive.org/ja/2019/resource-hints#resource-hints

せん。

20,931

図21.4. 1つのページでもっとも多くのプリロードヒントを提供しています。

リソースヒントを使った自動化がますます進む中、プリロードヒントやその他の要素を動的に注入する際には注意が必要です。

`as` 属性

`preload` や `prefetch` では、 `as` 属性を使用して、ブラウザがリソースの優先順位をより正確に判断できるようにすることが重要です。そうすることで将来のリクエストに備えてキャッシュに適切に保存し、正しいコンテンツセキュリティポリシー(CSP[726])を適用し、正しい `Accept` リクエストヘッダーを設定できます。

`preload` では、さまざまなコンテンツタイプをプリロードできます。完全なリスト[727]は、Fetch spec[728]で推奨されているものにしたがっています。もっとも人気があるのは、64%の使用率を誇る `script` タイプです。これはシングルページアプリとして構築されたサイトの大部分が、残りのJS依存ファイルのダウンロードを開始するために、できるだけ早くメインバンドルを必要としていることに関連していると考えられます。その後の使用率は、fontが8%、styleが5%、imageが1%、fetchが1%となっています。

726. *https://developer.mozilla.org/docs/Web/HTTP/CSP*
727. *https://developer.mozilla.org/docs/Web/HTML/Element/link#Attributes*
728. *https://fetch.spec.whatwg.org/#concept-request-destination*

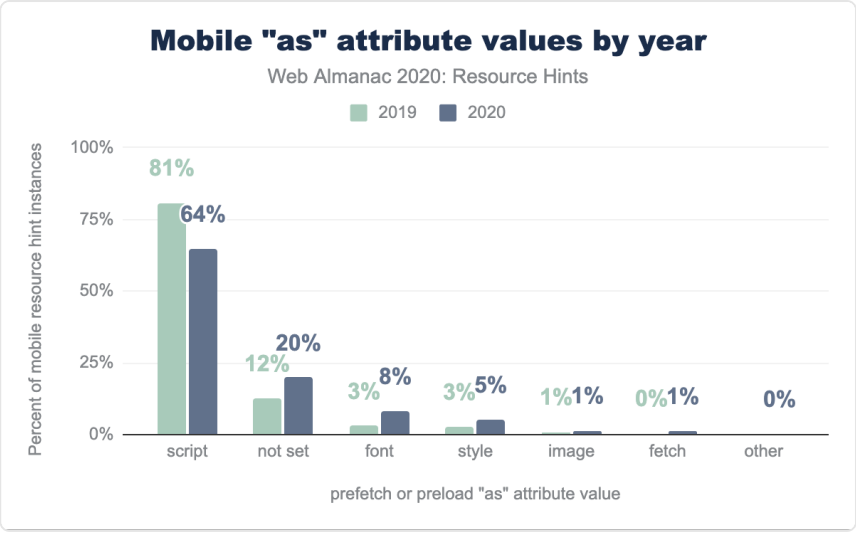


図21.5. モバイルの `as` 属性の値を年ごとに表示します。

2019[729]の傾向と比較すると、`as` 属性でのフォントやスタイルの使用が急激に増加しています。これは、開発者が重要なCSSの優先度を上げていることや、改善[730]のために `preload` フォントを `display:optional` と組み合わせて、Cumulative Layout Shift（CLS[731]）を実現していることに関連していると考えられます。

`as` 属性が省略されていたり無効な値が設定されていると、ブラウザが正しい優先順位を判断することが難しくなりスクリプトなどの場合には、リソースが2回取得されることもあるので注意が必要です。

`crossorigin` 属性

フォントなどのCORSが有効になっている `preload` や `preconnect` のリソースでは、そのリソースを適切に使用するために `crossorigin` 属性を含めることが重要です。`crossorigin` 属性がない場合、リクエストはシングルオリジンのポリシーに従いますので、preloadの使用は無意味になります。

729. https://almanac.httparchive.org/ja/2019/resource-hints#the-as-attribute
730. https://web.dev/i18n/ja/optimize-cls/#web-fonts-causing-foutfoit
731. https://web.dev/i18n/ja/cls/

16.96%

図21.6. `preload` を持つ要素のうち、`crossorigin` を使用する割合です。

最近の傾向としては、`preload` を行う要素のうち、`crossorigin` を設定し、anonymous（またはそれに準ずる）モードで読み込むものが16.96%、`use-credentials` を利用するものは0.02%にとどまっています。前述のフォントプリロードの増加に伴い、この割合も増加しています。

```
<link rel="preload" href="ComicSans.woff2" as="font" type="font/
woff2" crossorigin>
```

`crossorigin` 属性を持たずにプリロードされたフォントは、2回[732]取得されてしまうことに注意してください。

`media` 属性

異なるスクリーンサイズで使用するリソースを選択する際には、`media` 属性と `preload` を使用して、メディアクエリを最適化してください。

```
<link rel="preload" href="a.css" as="style" media="only screen and
(min-width: 768px)">
<link rel="preload" href="b.css" as="style" media="screen and (max-
width: 430px)">
```

2020年のデータセットでは、メディアクエリの組み合わせが2,100通り以上あることから、レスポンシブデザインのコンセプトと実装の間にサイトごとの差異がどれだけあるかを考えてみました。データには、Bootstrapなどでおなじみのブレークポイント `767px/768px` も含まれています。

ベストプラクティス

リソースヒントの使い方は、時に混乱を招くので、Lighthouseの自動監査に基づいて従うべ

732. https://web.dev/i18n/ja/preload-critical-assets/#how-to-implement-relpreload

き簡単なベストプラクティスを説明します。

`dns-prefetch` と `preconnect` を安全に実装するには、それらを別々のリンクタグにしてください。

```
<link rel="preconnect" href="http://example.com">
<link rel="dns-prefetch" href="http://example.com">
```

同じ `<link>` タグ内に `dns-prefetch` フォールバックを実装すると、Safariでバグ[733]が発生し、`preconnect` リクエストがキャンセルされてしまいます。2%近いページ（〜40k）で、1つのリソースに `preconnect` と `dns-prefetch` の両方が存在するという問題が報告されました。

"Preconnect to required origins[734]"の監査の場合、テストに合格したページはわずか19.67%で、何千ものウェブサイトが `preconnect` や `dns-prefetch` を使用して重要なサードパーティオリジンへの早期接続を確立し始める大きな機会を生み出していました。

図21.7. Lighthouseの `preconnect` 監査に合格したページの割合。

最後に、Lighthouseの"Preload key requests[735]"の監査を実行した結果、84.6%のページがテストに合格しました。はじめて `preload` を使おうとしている人は、フォントやクリティカルスクリプトを覚えておくといいでしょう。

ネイティブの遅延ローディング

今回は、Native Lazy Loading[736] APIの1周年をお祝いしましょう。このAPIは、公開時点ですでに72%[737]のブラウザをサポートしています。この新しいAPIを利用することで、ページ内の折り返し以下のiframeや画像の読み込みを、ユーザーがその近くでスクロールするまで延期できます。これにより、データ使用量やメモリ使用量を削減し、折り返し部分のコンテンツを高速化できます。遅延ロードを有効にするには、`<iframe>` や `<img>` 要素に `loading=lazy` を追加するだけです。

733. https://bugs.webkit.org/show_bug.cgi?id=197010
734. https://web.dev/i18n/ja/uses-rel-preconnect/
735. https://web.dev/uses-rel-preload/
736. https://addyosmani.com/blog/lazy-loading/
737. https://caniuse.com/loading-lazy-attr

4.02%

図21.8. ネイティブの遅延ローディングを使用しているページの割合。

とくに、今年初めに発表された公式基準値は保守的すぎ、最近[738]ようやく開発者の期待に応えられる状況です。約72%のブラウザがネイティブ画像/ソースの遅延ローディングをサポートしていることから、とくにローエンドデバイスでのデータ使用量やパフォーマンスを改善したいページにとっては、この分野もチャンスとなります。

Lighthouseの「Defer offscreen images[739]」監査を実行した結果、68.65%のページがテストに合格しました。これらのページでは、すべての重要なリソースのロードが完了した後に、画像を遅延ロードできます。

ビューポートが変わると画像が画面外に出てしまうことがあるので、デスクトップとモバイルの両方で監査を行うように注意してください。

予測型プリフェッチ

`prefetch` と機械学習を組み合わせることで、後続のページのパフォーマンスを向上させることができます。このソリューションの1つに、Guess.js[740]があります。予測型プリフェッチの最初のブレークスルーとなり、すでに10以上のウェブサイトが本番で使用しています。

予測型プリフェッチ[741]は、データ分析や機械学習の手法を用いて、データに基づいたプリフェッチのアプローチを提供する手法です。Guess.jsは、一般的なフレームワーク（Angular、Nuxt.js、Gatsby、Next.js）で予測型プリフェッチをサポートしているライブラリで、今日からでも利用できます。Guess.jsは、ページから可能なナビゲーションをランク付けし、次必要になりそうなJavaScriptのみをプリフェッチします。

学習セットにもよりますが、Guess.jsのプリフェッチは90%以上の精度で行われています。

全体的に見ると予測型プリフェッチはまだ未知の領域ですが、マウスオーバー時のプリフェッチやサービスワーカーのプリフェッチと組み合わせることで、ウェブサイトを利用するすべてのユーザーにデータを節約しながら瞬時に体験を提供できる大きな可能性を秘めています。

738. https://addyosmani.com/blog/better-image-lazy-loading-in-chrome/
739. https://web.dev/offscreen-images/
740. https://github.com/guess-js/guess
741. https://web.dev/predictive-prefetching/

HTTP/2プッシュ

HTTP/2には「サーバープッシュ」と呼ばれる機能があり、製品のラウンドトリップタイム(RTTs[742])やサーバーの処理が長い場合に、ページのパフォーマンスを改善できる可能性があります。簡単に説明すると、クライアントがリクエストを送信するのを待つのではなく、クライアントがすぐにリクエストするだろうと予測したリソースをサーバーが先取りしてプッシュするのです。

75.36%

図21.9. `preload` / `nopush` を使用したHTTP/2 Pushページの割合です。

HTTP/2 Pushは多くの場合、`preload` リンクヘッダーを通して開始されます。2020年のデータセットでは、1%のモバイルページがHTTP/2プッシュを使用しており、そのうち75%のプリロードヘッダーリンクがページリクエストで `nopush` オプションを使用していました。これはウェブサイトが `preload` リソースヒントを使用していても、大多数はこれだけを使用し、そのリソースのHTTP/2プッシュを無効にすることを好むということです。

HTTP/2 Pushは、正しく使用しないとパフォーマンスを損なう可能性があることを言及しておくことが重要で、これがしばしば無効にされる理由となっています。

このパターンは、**重要なリソースをプッシュ**（またはプリロード）し、**最初のルートをできるだけ早くレンダリング**し、**残りのアセットをプリキャッシュ**し、**他のルートや重要でないアセットを遅延ロード**するというものです。これは、WebサイトがProgressive Web Appであり、キャッシング戦略を改善するためにService Workerを使用している場合にのみ可能です。このようにすることで、それ以降のすべてのリクエストがネットワークに出ることはないので、常にプッシュする必要はなく両方の利点を得ることができます。

サービス・ワーカー

`preload` と `prefetch` の両方において、ページがService Worker[743]によって制御されている場合に採用が増えています。これはService Workerがまだアクティブでないときにプリロードすることでリソースの優先順位を向上させたり、将来のリソースをインテリジェントにプリフェッチしてユーザーが必要とする前に、Service Workerにキャッシュできる可能性があるためです。

742. https://developer.mozilla.org/docs/Glossary/Round_Trip_Time_(RTT)
743. https://developers.google.com/web/fundamentals/primers/service-workers

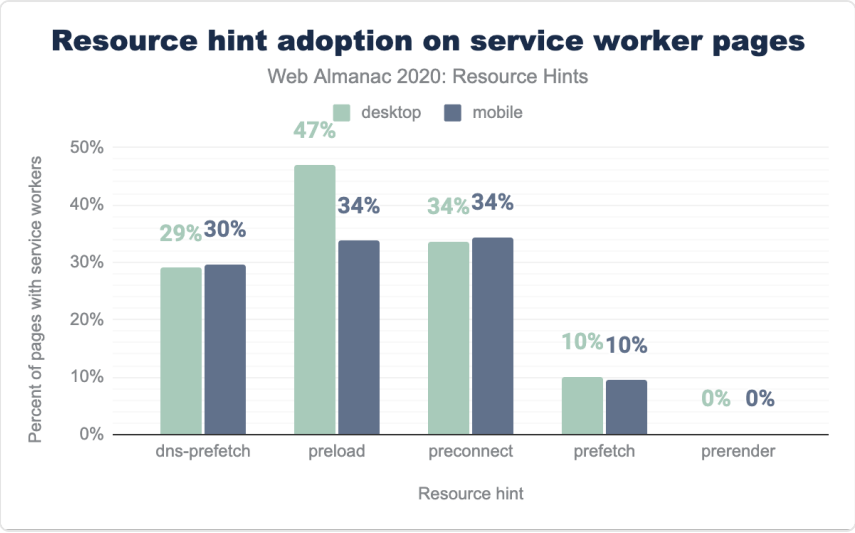


図21.10. サービスワーカーページでのリソースヒント採用。

デスクトップの `preload` では47%、 `prefetch` では10%という優れた採用率となっています。いずれの場合も、Service Workerを使用しない場合の平均的な採用率と比較して、非常に高いデータとなっています。

先に述べたように、リソースヒントとサービスワーカーのキャッシュ戦略をどのように組み合わせるかについては、PRPLパターン[744]が今後重要な役割を果たすことになるでしょう。

未来

ここでは、実験的なヒントをいくつか紹介します。リリースに非常に近いところではPriority Hintsがあり、これはWebコミュニティで積極的に実験されています。また、HTTP/2の103 Early Hintsもありますが、これはまだ初期段階にあり、ChromeやFastlyのような数人のプレイヤーが今後のテストトライアルに向けて協力しています[745]。

優先順位のヒント

優先度のヒント[746]はリソースのフェッチの優先度を表現するためのAPIです：高、低、または自動。画像の優先順位を下げたり（カルーセルの中など）、スクリプトの優先順位を変えたり、さらにはフェッチの優先順位を下げたりするのにも使用できます。

744. https://addyosmani.com/blog/the-prpl-pattern/
745. https://www.fastly.com/blog/beyond-server-push-experimenting-with-the-103-early-hints-status-code
746. https://developers.google.com/web/updates/2019/02/priority-hints

この新しいヒントは、HTMLタグとして、またはHTML属性と同じ値を取る`importance`オプションを使ってフェッチリクエストの優先順位を変更することで使用できます。

```
<!-- リソースの早期取得を開始したいが、同時に優先順位を下げたい -->
<link rel="preload" href="/js/script.js" as="script"
importance="low">

<!-- ブラウザが「高」の優先順位をつけている画像ですが、実際にはそれを望んでいま
せん。 -->
<img src="/img/in_view_but_not_important.svg" importance="low"
alt="I'm not important!">
```

`preload`や`prefetch`では、リソースの種類に応じてブラウザが優先順位を設定します。優先度のヒントを使うことで、ブラウザにデフォルトのオプションを変更させることができます。

0.77%

図21.11. モバイルでのプライオリティ・ヒントの採用率。

Chromeはまだ積極的に[747]実験を行っているため、今のところ0.77%のウェブサイトしかこの新しいヒントを採用しておらず、この記事のリリース時点ではこの機能は保留されています。

もっとも多く使用されているのはscript要素で、これはJSのプライマリファイルやサードパーティファイルの数が増え続けていることからも当然のことです。

16%

図21.12. ヒントのあるモバイルリソースのうち、`低`の優先度を使っている割合です。

データによると、優先度ヒントを使用しているリソースの83%がモバイルで「高い」優先度を使用していますが、さらに注目すべきは「低い」優先度のリソースが16%あることです。

優先度ヒントは「高い」優先度でリソースをより早く読み込ませようとする戦術ではなく、

747. https://www.chromestatus.com/features/5273474901737472

ブラウザが何を優先度から外すべきかを判断し重要なリクエストを先に完了させるため重要なCPUや帯域を返すことで、「低い」優先度によるムダな読み込みを防ぐツールとして明らかな優位性を持っています。

HTTP/2の103 Early Hints

前回、HTTP/2プッシュは、プッシュされるアセットがすでにブラウザのキャッシュにある場合、実際にリグレッションを引き起こす可能性があると述べました。103 Early Hints[748]の提案は、HTTP/2プッシュで約束された同様の利点を提供することを目的としています。潜在的に10倍シンプルなアーキテクチャで、サーバープッシュによる不必要なラウンドトリップという既知のワーストケースの問題に悩まされることなく、長いRTTやサーバー処理に対処します。

現在のところ、Chromiumでは671310[749]、1093693[750]、1096414[751]の問題で会話を追うことができます。

結論

昨年、リソースヒントの採用が増加し、開発者がリソースの優先順位付けや最終的なユーザー体験の多くの側面をより詳細にコントロールするために不可欠なAPIとなりました。しかし、これらは指示ではなくヒントであり、残念ながら最終的な決定権は常にブラウザとネットワークにあることを忘れてはなりません。

確かに、たくさんの要素にそれらを適用すれば、ブラウザはあなたの要求通りに動くかもしれません。あるいは、ヒントを無視して、デフォルトの優先順位が状況に応じて最適な選択であると判断するかもしれません。いずれにしても、これらのヒントをどのように使うのがベストなのか、プレイブックを用意しておきましょう。

- ユーザー体験のための重要なページを特定する。
- 最適化すべきもっとも重要なリソースを分析します。
- 可能な限り、PRPLパターン[752]を採用する。
- それぞれの導入前と導入後のパフォーマンス体験を測定する。

最後になりましたが、ウェブはすべての人のためにあることを忘れないでください。私たち

748. *https://tools.ietf.org/html/rfc8297*
749. *https://bugs.chromium.org/p/chromium/issues/detail?id=671310*
750. *https://bugs.chromium.org/p/chromium/issues/detail?id=1093693*
751. *https://bugs.chromium.org/p/chromium/issues/detail?id=1096414*
752. *https://addyosmani.com/blog/the-prpl-pattern/*

は、ウェブを守り続け、簡単で摩擦のない体験を構築することに集中しなければなりません。

私たちはすべての人に優れたウェブ体験を提供するために必要なすべてのAPIの提供に、年々少しずつ近づいていることを嬉しく思っていますし、次に何が来るのかを楽しみにしています。

著者

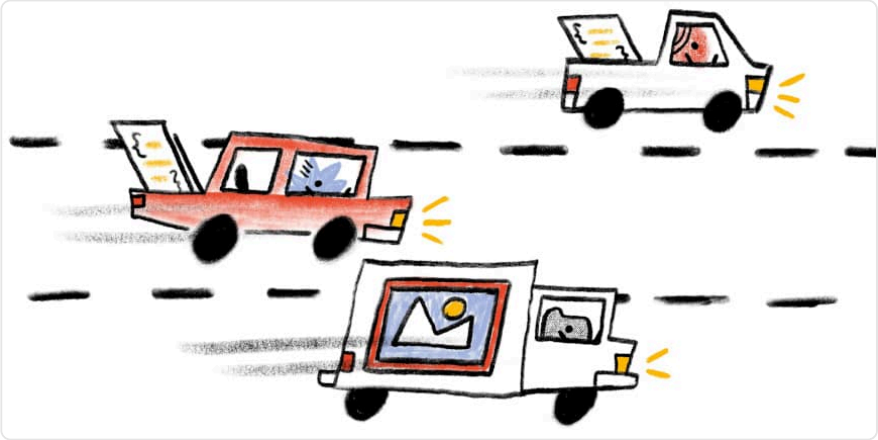
Leonardo Zizzamia

@Zizzamia Zizzamia

Leonardoは、Coinbase[753]のスタッフソフトウェアエンジニアとして、ウェブパフォーマンスと成長イニシアチブを担当しています。また、NGRome Conference[754]を主催しています。Leonardoはまた、パフォーマンス分析を通じて企業がロードマップに優先順位をつけ、より良いビジネス上の意思決定を行えるよう支援するPerfume.js[755]ライブラリを管理しています。

753. https://www.coinbase.com/
754. https://ngrome.io
755. https://github.com/Zizzamia/perfume.js

部 IV 章 22
HTTP/2



Andrew Galloni、 Robin Marx と Mike Bishop によって書かれた。
Lucas Pardue、 Barry Pollard と Sawood Alam によってレビュ。
Greg Wolf による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章

HTTPは、ネットワーク接続されたデバイス間で情報を転送するために設計されたアプリケーション層のプロトコルで、ネットワークプロトコルスタックの他の層の上で動作します。HTTP/1.xがリリースされてから、最初のメジャーアップデートであるHTTP/2が2015年に標準化されるまで20年以上かかりました。

それだけではありません。過去4年間、HTTP/3とQUIC（新しい遅延削減、信頼性、安全性の高いトランスポートプロトコル）はIETFのQUICワーキンググループで標準化開発が行われてきました。実は、同じ名前を持つプロトコルは2つあります。”Google QUIC”（略して“gQUIC”）とは、Googleが設計して使用していたオリジナルのプロトコルと、より新しいIETF標準化版（IETF QUIC/QUIC）のことです。IETF QUICはgQUICをベースにしていましたが、設計や実装が大きく異なるものに成長しています。2020年10月21日、IETF QUICの

ドラフト32が最終版[756]に移行し、重要な節目を迎えました。これは、作業グループがほぼ完成したと考え、より広いIETFコミュニティに最終レビューを要求する標準化プロセスの一部です。

この章では、HTTP/2とgQUICの展開の現状をレビューします。優先順位付けやサーバプッシュなど、プロトコルの新機能のいくつかがどの程度採用されているかを探ります。次に、HTTP/3の動機、プロトコルバージョン間の主な違い、QUICを使ってUDPベースのトランスポートプロトコルにアップグレードする際の潜在的な課題について説明します。

HTTP/1.0からHTTP/2

HTTPプロトコルが進化しても、HTTPのセマンティクスは変わりません。HTTPメソッド（GETやPOSTなど）、ステータスコード (200や恐ろしい404)、URI、ヘッダフィールドに変更はありません。HTTPプロトコルが変わったところでは、ワイヤーエンコーディングと、基礎となるトランスポートの機能の使用が違います。

1996年に公開されたHTTP/1.0は、テキストベースのアプリケーションプロトコルを定義し、クライアントとサーバがリソースを要求するためにメッセージを交換することを可能にしました。リクエスト/レスポンスごとに新しいTCP接続が必要となり、オーバーヘッドが発生していました。TCP接続では、飛行中のデータ量を最大化するために輻輳制御アルゴリズムを使用しています。このプロセスは、新しい接続ごとに時間がかかります。この「スロースタート」は、利用可能なすべての帯域幅がすぐに使用されるわけではないことを意味します。

1997年にHTTP/1.1が導入され、「keep-alives」を追加してTCP接続の再利用を可能にし、接続開始時のトータルコストを削減することを目的としています。時間の経過とともに、ウェブサイトのパフォーマンスへの期待が高まるにつれ、 リクエストの同時実行性が必要になってきました。HTTP/1.1は、前のレスポンスが完了した後にのみ別のリソースを要求できました。そのため、追加のTCP接続を確立する必要があり、keep-alives接続の影響を軽減し、さらにオーバーヘッドを増加させていました。

2015年に公開されたHTTP/2は、クライアントとサーバ間の双方向ストリームの概念を導入したバイナリベースのプロトコルです。これらのストリームを使用することで、ブラウザは単一のTCP接続を最適に利用して、複数のHTTPリクエスト/レスポンスを同時に多重化できます。HTTP/2はまた、この多重化を制御するための優先順位付けスキームを導入しました。クライアントは、より重要なリソースを他のリソースよりも優先して送信できるようにリクエストの優先順位をシグナルできます。

756. https://mailarchive.ietf.org/arch/msg/quic/ye1LeRl7oEz898RxjE6D3koWhn0/

HTTP/2の採用

この章で使用したデータはHTTP Archiveから入手したもので、700万以上のウェブサイトをChromeブラウザでテストしています。 他の章と同様に、分析はモバイルとデスクトップのウェブサイトに分かれています。デスクトップとモバイルの結果が似ている場合は、モバイルのデータセットから統計情報を表示しています。詳細は方法論のページに掲載されています。このデータを見直す際には、リクエストの数に関係なく、それぞれのウェブサイトが同等の重みを受けることを念頭に置いてください。このデータは、幅広いアクティブなウェブサイトの傾向を調査していると考えることをお勧めします。

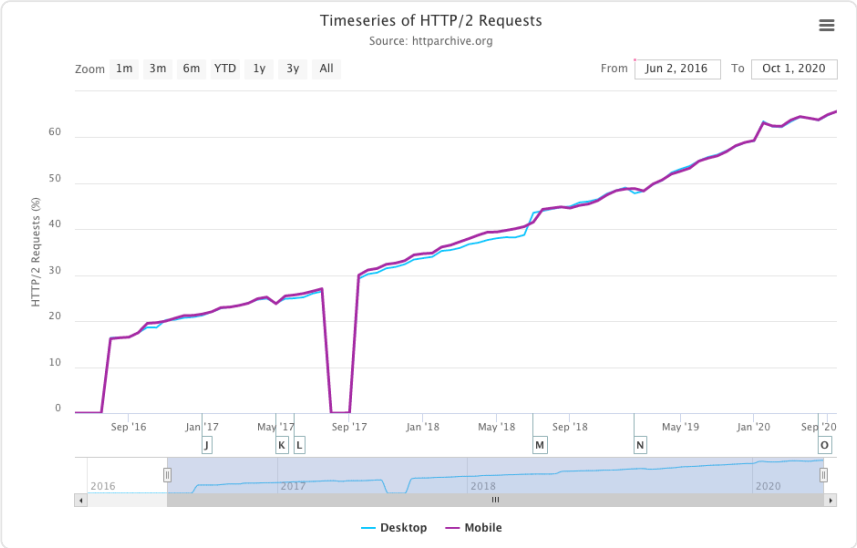


図22.1. リクエスト別のHTTP/2の使用状況。(提供元: HTTP Archive[757])

昨年のHTTP Archiveデータの分析では、HTTP/2がリクエストの50%以上に使用されており、ご覧のように2020年も直線的な成長が続いており、現在ではリクエストの60%以上がHTTP/2で提供されています。

64%

図22.2. HTTP/2を使用するリクエストの割合。

図22.3と昨年の結果を比較すると、**HTTP/2リクエストが10%増加し**、HTTP/1.xリクエスト

757. https://httparchive.org/reports/state-of-the-web#h2

が10%減少しています。これはgQUICがデータセットで確認できるようになった最初の年です。

プロトコル	デスクトップ	モバイル
HTTP/1.1	**34.47%	34.11%
HTTP/2	63.70%	63.80%
gQUIC	1.72%	1.71%

図22.3. HTTP version usage by request.

*** 昨年のクロールと同様に、デスクトップリクエストの約4%はプロトコルバージョンを報告していませんでした。分析によると、これらはほとんどがHTTP/1.1であり、今後のクロールと分析のためにこのギャップを修正しました。データは2020年8月のクロールに基づいていますが、公開前の2020年10月のデータセットで修正を確認したところ、これらのリクエストは確かにHTTP/1.1であったことが判明したため、上記の表の統計に追加しました。*

ウェブサイトのリクエストの総数を確認すると、一般的なサードパーティのドメインに偏りが出てきます。サーバーインストールによるHTTP/2の採用状況をよりよく理解するために、代わりにサイトのホームページからHTMLを提供するために使用されるプロトコルを見てみましょう。

昨年はホームページの約37%がHTTP/2で、63%がHTTP/1で提供されていましたが、今年はモバイルとデスクトップを合わせるとほぼ同じ割合で図22.4に示すように初めてHTTP/2で提供されたデスクトップサイトの数がわずかに増えています。

プロトコル	デスクトップ	モバイル
HTTP/1.0	0.06%	0.05%
HTTP/1.1	49.22%	50.05%
HTTP/2	49.97%	49.28%

図22.4. HTTP version usage for home pages.

gQUICがホームページのデータに表示されないのには、2つの理由があります。gQUIC上のウェブサイトを測定するために、HTTP Archiveのクロールはalternative servicesヘッダーを介してプロトコルネゴシエーションを実行し、このエンドポイントを使用してgQUIC上のサイトをロードする必要があります。これは今年サポートされていませんでしたが、来年のWeb Almanacで利用できるようになると期待しています。また、gQUICはホームページを提供するのではなく、サードパーティのGoogleツールに主に使用されています。

ウェブ上のセキュリティとプライバシーを向上させようとする動きにより、TLSを利用したリクエストは過去4年間で150%以上[758]増加しています。今日では、モバイルとデスクトップにおけるすべてのリクエストの86%以上が暗号化されています。ホームページだけを見ると、デスクトップでは78.1%、モバイルでは74.7%という印象的な数字になっています。これは、HTTP/2がTLSを介したブラウザでのみサポートされていることが重要です。図22.5に示すように、HTTP/2でサービスされる割合も昨年[759]から10ポイント増えており、55%から65%になっています。

プロトコル	デスクトップ	モバイル
HTTP/1.1	36.05%	34.04%
HTTP/2	63.95%	65.96%

図22.5. HTTP version usage for HTTPS home pages.

60%以上のWebサイトがHTTP/2またはgQUICで提供されているため、個々のサイト間で行われたすべてのリクエストのプロトコル分布のパターンをもう少し詳しく見てみましょう。

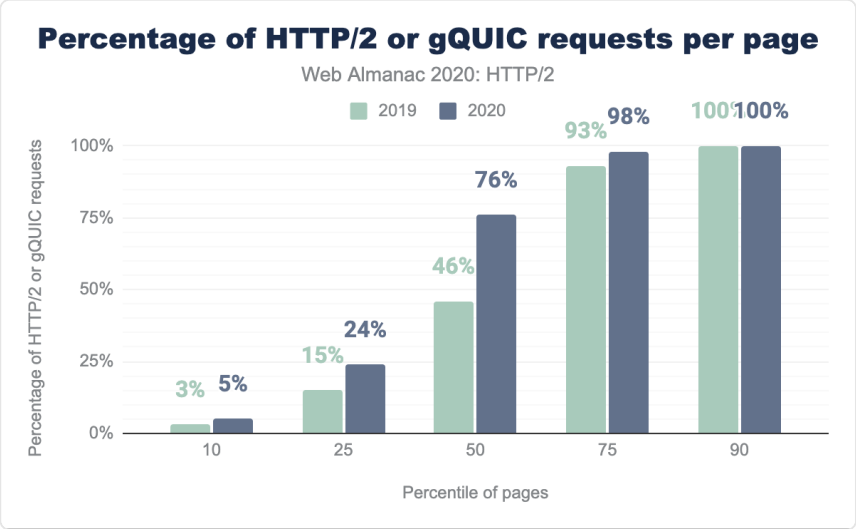


図22.6. 2020年の1ページあたりのHTTP/2リクエストの割合の分布を2019年と比較してみましょう。

図22.6は、今年と昨年の間にWebサイトでどれだけHTTP/2またはgQUICが使用されているかを比較したものです。最も顕著な変化は、昨年の46%に対し、半数以上のサイトで75%以

758. https://httparchive.org/reports/state-of-the-web#pctHttps
759. https://almanac.httparchive.org/ja/2019/http#fig-5

上のリクエストがHTTP/2またはgQUICで提供されていることです。7%未満のサイトではHTTP/2またはgQUICリクエストを行わず、10%のサイトでは完全にHTTP/2またはgQUICリクエストを行っています。

ページ自体の内訳は？　一般的には、ファーストパーティコンテンツとサードパーティコンテンツの違いについて話します。サードパーティとは、広告、マーケティング、分析などの機能を提供するサイト所有者の直接の管理下にないコンテンツと定義されています。既知のサードパーティの定義は、サードパーティウェブ[760]のリポジトリから引用しています。

図22.7では、他のリクエストと比較して既知のサードパーティまたはファーストパーティのHTTP/2リクエストの割合で、すべてのWebサイトに注文を付けています。全サイトの40%以上のサイトでは、ファーストパーティのHTTP/2リクエストやgQUICリクエストが全くないため、顕著な違いが見られます。対照的に、下位5%のページでさえ、30%のサードパーティコンテンツがHTTP/2で提供されています。これは、HTTP/2の普及の大部分がサードパーティによるものであることを示しています。

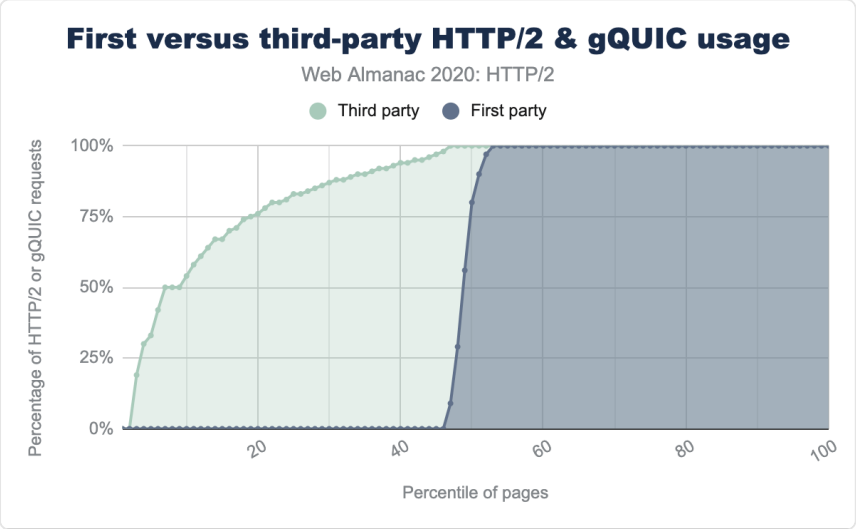


図22.7. 1ページあたりのサードパーティとファーストパーティのHTTP/2リクエストの割合の分布。

HTTP/2とgQUICのどちらのコンテンツタイプで提供されるかに違いはありますか？ 図22.8は、例えば、90%のWebサイトがサードパーティのフォントとオーディオを100%HTTP/2またはgQUICで提供しており、HTTP/1.1では5%しか提供しておらず、5%が混在していることを示しています。サードパーティのアセットの大部分はスクリプトまたは画像で、HTTP/2またはgQUICでのみ提供されているWebサイトがそれぞれ60%と70%あります。

760. https://github.com/patrickhulce/third-party-web/blob/8afa2d8cadddec8f0db39e7d715c07e85fb0f8ec/data/entities.json5

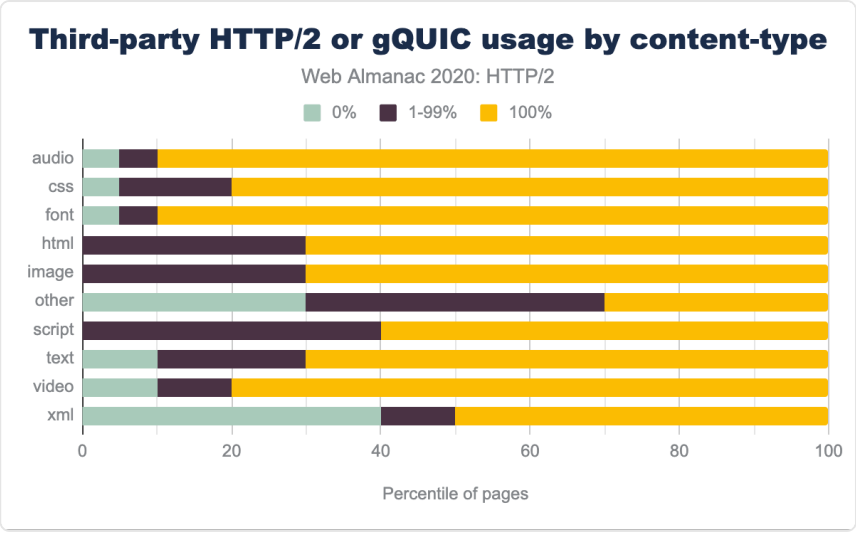


図22.8. ウェブサイトごとのコンテンツタイプ別の既知のサードパーティHTTP/2またはgQUICリクエストの割合。

図22.9に示すように、広告、分析、コンテンツ配信ネットワーク（CDN）リソース、タグ管理者は主にHTTP/2またはgQUICで提供されています。顧客満足度の高いコンテンツやマーケティングコンテンツは、HTTP/1で提供される可能性が高いです。

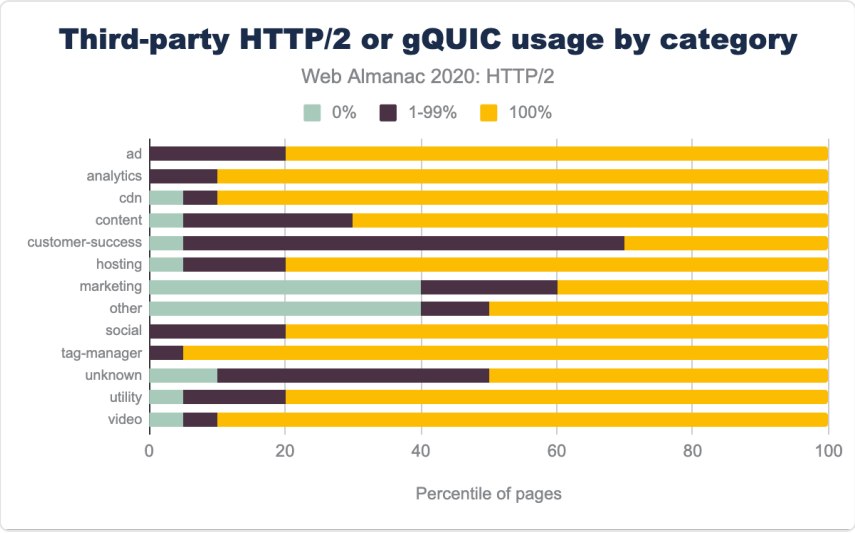


図22.9. ウェブサイトごとのカテゴリ別の既知のサードパーティ製HTTP/2またはgQUICリクエストの割合。

サーバーサポート

ブラウザの自動更新メカニズムは、新しいウェブ標準をクライアント側で採用するための原動力となっています。それは推定[761]で、全世界のユーザーの97%以上がHTTP/2をサポートしており、昨年測定された95%からわずかに増加しています。

残念ながら、サーバーのアップグレードパスは、特にTLSをサポートする必要があるため、より難しくなっています。モバイルとデスクトップでは、図22.10を見ると、HTTP/2サイトの大部分がnginx、Cloudflare、Apacheによって提供されていることがわかります。HTTP/1.1のサイトのほぼ半分はApacheによって提供されています。

761. https://caniuse.com/http2

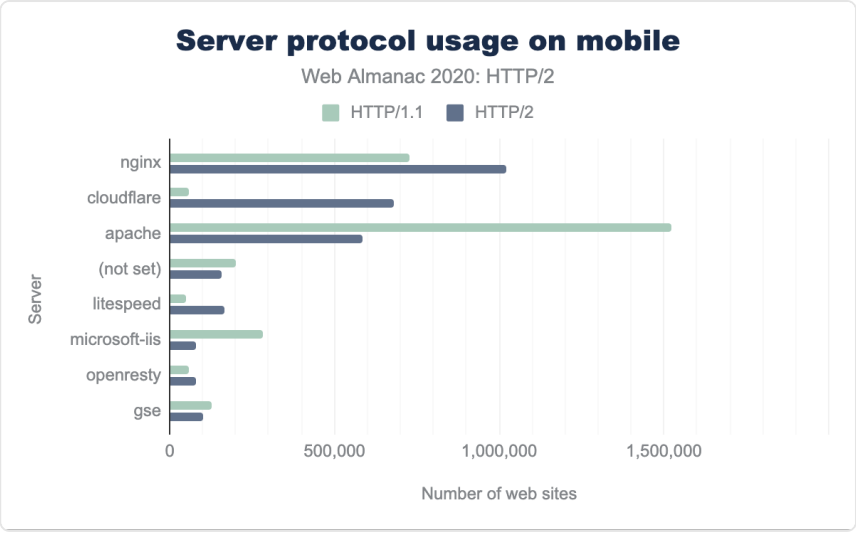


図22.10. モバイル上でのHTTPプロトコルによるサーバーの使用状況

各サーバの昨年のHTTP/2採用率はどのように変化したのでしょうか？　図22.11は、昨年からすべてのサーバでHTTP/2の採用率が約10%増加していることを示しています。ApacheとIISはまだ25%以下のHTTP/2です。これは新しいサーバがnginxになる傾向があるか、ApacheやIISをHTTP/2やTLSにアップグレードするのが難しすぎるか、価値がないと思われているかのどちらかだと考えられます。

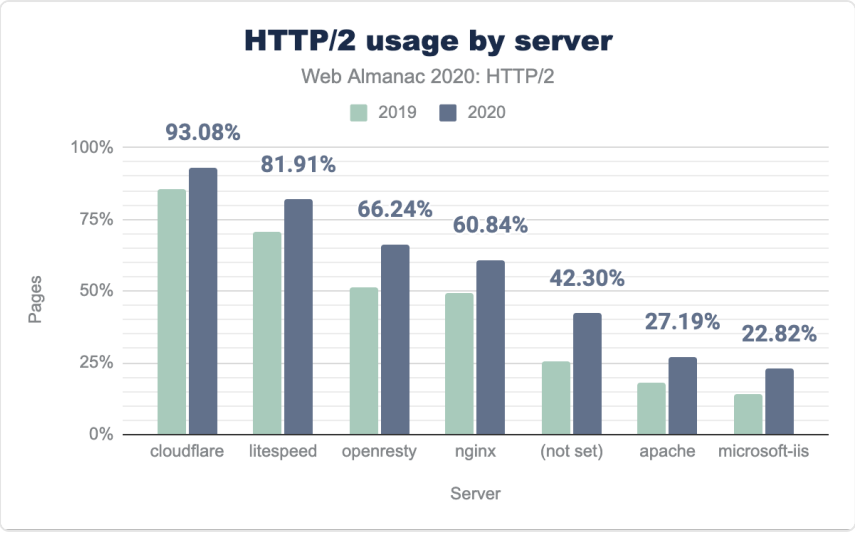


図22.11. サーバー別にHTTP/2で提供されたページの割合

ウェブサイトのパフォーマンスを向上させるために長期的に推奨されてきたのは、CDNの使用です。その利点は、コンテンツを提供することと、エンドユーザーに近いところで接続を終了させることの両方によって待ち時間が短縮されることです。これは、プロトコルの展開における急速な進化と、サーバーやOSのチューニングにおける追加の複雑さを緩和するのに役立ちます（詳細については、優先順位付けセクションを参照してください）。新しいプロトコルを効果的に利用するためには、CDNを利用することが推奨されています。

CDNは大きく分けて2つのカテゴリーに分類されます：ホームページや資産のサブドメインを提供するものと、サードパーティのコンテンツを提供するために主に使用されるものです。最初のカテゴリの例としては、大規模で汎用的なCDN（Cloudflare、Akamai、Fastlyなど）と、より特殊なCDN（WordPressやNetlifyなど）があります。CDNの有無によるホームページのHTTP/2採用率の違いを見てみると、次のようになります。

- CDNが使用されている場合、モバイルホームページの**80%**はHTTP/2で提供されます。
- CDNを使用しない場合、モバイルホームページの**30%**はHTTP/2で提供されます。

図22.12は、より具体的で、現代のCDNはHTTP/2上のトラフィックの高い割合を提供しています。

HTTP/2 (%)	CDN
100%	Bison Grid, CDNsun, LeaseWeb CDN, NYI FTW, QUIC.cloud, Roast.io, Sirv CDN, Twitter, Zycada Networks
90 - 99%	Automattic, Azion, BitGravity, Facebook, KeyCDN, Microsoft Azure, NGENIX, Netlify, Yahoo, section.io, Airee, BunnyCDN, Cloudflare, GoCache, NetDNA, SFR, Sucuri Firewall
70 - 89%	Amazon CloudFront, BelugaCDN, CDN, CDN77, Erstream, Fastly, Highwinds, OVH CDN, Yottaa, Edgecast, Myra Security CDN, StackPath, XLabs Security
20 - 69%	Akamai, Aryaka, Google, Limelight, Rackspace, Incapsula, Level 3, Medianova, OnApp, Singular CDN, Vercel, Cachefly, Cedexis, Reflected Networks, Universal CDN, Yunjiasu, CDNetworks
< 20%	Rocket CDN, BO.LT, ChinaCache, KINX CDN, Zenedge, ChinaNetCenter

図22.12. Percentage of HTTP/2 requests served by the first-party CDNs over mobile.

2番目のカテゴリのコンテンツの種類は、一般的に共有リソース（JavaScriptやフォントCDN）、広告、またはアナリティクスです。これらすべての場合において、CDNを使用することで、様々なSaaSソリューションのパフォーマンスとオフロードを向上させることができます。

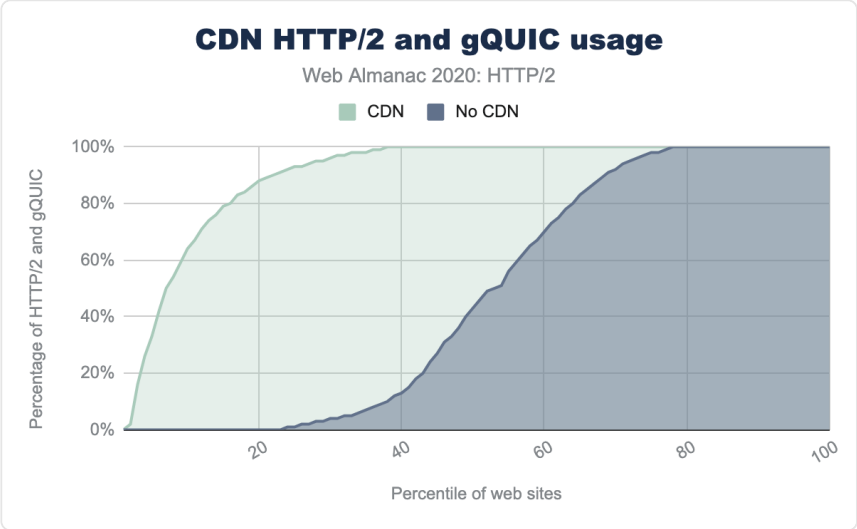


図22.13. CDNを利用したWebサイトのHTTP/2とgQUICの利用状況の比較。

図22.13では、ウェブサイトがCDNを使用している場合のHTTP/2とgQUICの採用に大きな違いが見られます。CDNを使用している場合、70%のページがすべてのサードパーティのリク

エストにHTTP/2を使用しています。CDNを使用しない場合は、25%のページのみがすべてのサードパーティのリクエストにHTTP/2を使用しています。

HTTP/2のインパクト

プロトコルがどのように動作しているかの影響を測定することは、現在のHTTP Archiveアプローチでは困難です。同時接続の影響、パケットロスの影響、さまざまな輻輳制御メカニズムを定量化できるのは、本当に魅力的なことでしょう。実際にパフォーマンスを比較するには、各ウェブサイトを異なるネットワーク条件の各プロトコル上でクロールする必要があります。その代わりにできることは、ウェブサイトが使用する接続数への影響を調べることです。

接続を減らす

前述したように、HTTP/1.1ではTCP接続で一度に1つのリクエストしかできません。ほとんどのブラウザは、ホストごとに6つの並列接続を許可することでこの問題を回避しています。HTTP/2の大きな改善点は、1つのTCP接続で複数のリクエストを多重化できることです。 これにより、ページをロードするために必要なコネクションの総数と、関連する時間とリソースを減らすことができます。

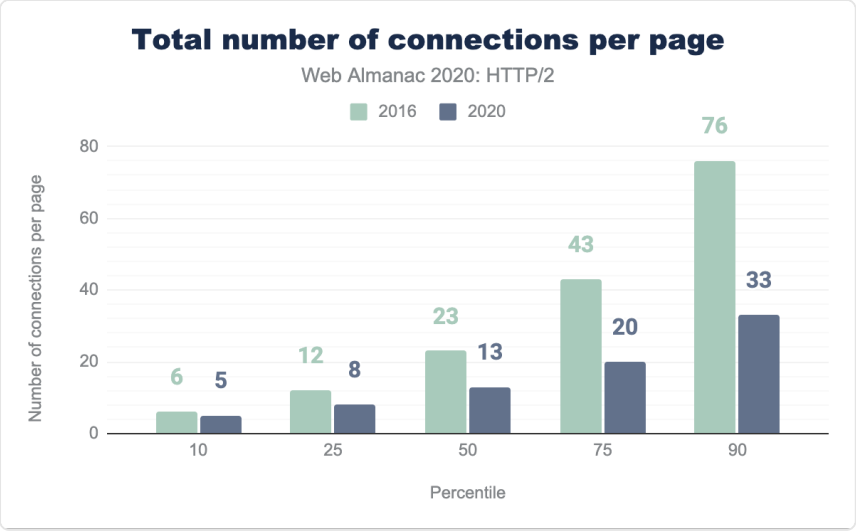


図22.14. 1ページあたりの総接続数の分布

図22.15は、2016年と比較して2020年に1ページあたりのTCP接続数がどのように減少した

かを示しています。すべてのウェブサイトの半数が、2016年には23個のTCP接続を使用していたのに対し、2020年には13個以下のTCP接続を使用するようになり、44%の減少となっています。同じ期間に、[リクエスト数の中央値]（https://httparchive.org/reports/state-of-the-web#reqTotal）は74から73に減少しただけです。TCP接続あたりのリクエスト数の中央値は3.2から5.6に増加しています。

TCPは、効率的かつ公平な平均的なデータフローを維持するように設計されています。各フローが他のフローに圧力をかけ、他のすべてのフローに反応して、ネットワークの公平なシェアを提供するフロー制御プロセスを想像してみてください。公平なプロトコルでは、すべてのTCPセッションが他のセッションを圧迫することはなく、時間の経過とともにパス容量の1/Nを占有することになります。

ウェブサイトの大部分は、今でも15のTCP接続で開かれています。HTTP/1.1では、ブラウザがドメインに対して開くことができる6つの接続は、時間の経過とともに単一のHTTP/2接続の6倍の帯域幅を要求できます。低容量ネットワークでは、競合する接続数が増加し、重要なリクエストから帯域幅を奪うため、プライマリアセットドメインからのコンテンツ配信が遅くなる可能性があります。これは、サードパーティドメインの数が少ないウェブサイトに有利です。

HTTP/2は、異なるが関連するドメイン間の接続の再利用[762]を可能にします。TLSリソースの場合、URI内のホストに対して有効な証明書を必要とします。これは、サイト作成者の管理下にあるドメインに必要な接続数を減らすために使用できます。

優先順位付け

HTTP/2レスポンスは多くの個々のフレームに分割でき、複数のストリームからのフレームを多重化できるため、フレームがインターリーブされてサーバーによって配信される順序が重要なパフォーマンスの考慮事項となります。典型的なウェブサイトは、可視コンテンツ（HTML、CSS、画像）、アプリケーションロジック（JavaScript）、広告、サイト利用状況を追跡するためのアナリティクス、マーケティングトラッキングビーコンなど多くの異なるタイプのリソースで構成されています。ブラウザがどのように動作するかを知ることで、最速のユーザー体験をもたらすリソースの最適な順序を定義できます。最適と非最適の違いは、パフォーマンスを50%以上向上させることができます。

HTTP/2は、クライアントがどのように多重化を行うべきだと考えているかをサーバへ伝えるために、優先順位付けの概念を導入しました。すべてのストリームには重み（利用可能な帯域幅のうちどれだけのストリームを割り当てるべきか）と親（最初に配信すべき別のストリーム）が割り当てられています。HTTP/2の優先順位付けモデルの柔軟性を考えると、現在のブラウザエンジンのすべてが異なる優先順位付け戦略[763] を実装していても不思議ではあ

762. https://tools.ietf.org/html/rfc7540#section-9.1
763. https://calendar.perfplanet.com/2018/http2-prioritization/

りませんが、どれも最適[764]ではありません。

またサーバ側にも問題があり、多くのサーバでは優先順位付けがうまくいかなかったり、まったくできなかったりしています。 HTTP/1.xの場合、サーバ側の送信バッファーを可能な限り大きくするようにチューニングすることは、メモリ使用量の増加（CPUとメモリを交換する）以外には何のデメリットもなく、ウェブサーバのスループットを向上させる効果的な方法です。これはHTTP/2の場合はそうではありませんが、TCP送信バッファー内のデータは新しいより重要なリソースのリクエストが来た場合、再優先順位をつけることができません。 HTTP/2サーバでは、最適な送信バッファサイズは、利用可能な帯域幅を十分に利用するために必要な最小データ量です。これにより、より優先度の高いリクエストを受信した場合、サーバは即座に応答できます。

このように大きなバッファーが優先順位付けを混乱させる問題は、ネットワークにも存在し、「bufferbloat」と呼ばれています。 ネットワーク機器は、短いバーストがあるときにパケットをドロップするよりも、むしろパケットをバッファリングするでしょう。 しかし、サーバーがクライアントへのパスを消費できる以上のデータを送信すると、これらのバッファーは容量いっぱいになります。ネットワーク上にすでに「保存」されているこれらのバイトは、大きな送信バッファーがそうであるように、より優先度の高いレスポンスをより早く送信するサーバーの能力を制限します。バッファーに保持されるデータ量を最小化するには、BBRのような最近の輻輳制御アルゴリズムを使用する必要があります[765].

Andy Daviesによって維持されているこのテストスイート[766]は、様々なCDNとクラウドホスティングサービスがどのようにパフォーマンスを発揮するかを測定し、報告しています。悪いニュースは、36のサービスのうち9つだけが正しく優先順位をつけているということです。図22.16は、CDNを使用しているサイトでは、約31.7%が正しく優先順位を付けていないことを示しています。これは昨年の26.82%から上昇していますが、これは主にGoogle CDNの利用が増えたことによるものです。ブラウザから送られてくる優先順位に頼るのではなく、代わりにサーバ側優先順位付け[767]スキームを実装し、ブラウザのヒントを追加ロジックで改良しているサーバもあります。

764. *https://www.youtube.com/watch?v=nH4iRpFnf1c*
765. *https://blog.cloudflare.com/http-2-prioritization-with-nginx/*
766. *https://github.com/andydavies/http2-prioritization-issues*
767. *https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/*

CDN	正しく優先順位をつける	デスクトップ	モバイル
CDNを使用していない	不明	59.47%	60.85%
Cloudflare	パス	22.03%	21.32%
Google	失敗	8.26%	8.94%
Amazon CloudFront	失敗	2.64%	2.27%
Fastly	パス	2.34%	1.78%
Akamai	パス	1.31%	1.19%
Automattic	パス	0.93%	1.05%
Sucuri Firewall	失敗	0.77%	0.63%
Incapsula	失敗	0.42%	0.34%
Netlify	失敗	0.27%	0.20%

図22.15. HTTP/2 prioritization support in common CDNs.

CDN以外の利用では、HTTP/2の優先順位付けを正しく適用するサーバーの数はかなり少なくなると予想されます。例えば、NodeJSのHTTP/2実装は優先順位付けをサポートしていません[768]。

さよならサーバープッシュ？

サーバープッシュはHTTP/2の追加機能の1つで、実際に実装するには混乱と複雑さの原因となりました。プッシュは、ブラウザ/クライアントがHTMLページをダウンロードし、そのページを解析し、そのページが追加のリソース（スタイルシートなど）を必要としていることを発見するのを待つことを避けようとしています。これらの作業と往復には時間がかかります。 サーバプッシュを使えば、理論的には、サーバは一度に複数のレスポンスを送信できるので、余分なラウンドトリップを避けることができます。

残念なことに、TCPの輻輳制御が効いていると、データ転送は非常に遅く、複数回の往復で転送速度が十分に向上するまでは、すべての資産をプッシュすることはできない[769]となります。また、クライアントの処理モデルが完全に合意されていなかったため、ブラウザ間では実装の違い[770]があります。例えば、ブラウザごとに プッシュのキャッシュ 実装が異なりま

768. https://x.com/jasnell/status/1245410283582918657
769. https://calendar.perfplanet.com/2016/http2-push-the-details/
770. https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/

す。

もう1つの問題は、ブラウザが既にキャッシュしたリソースをサーバの方で認識していないことです。サーバが不要なものをプッシュしようとすると、クライアントは `RST_STREAM` フレームを送信できますが、その時点でサーバはすでにすべてのデータを送信している可能性があります。 これは帯域幅を無駄にし、サーバはブラウザが実際に必要としたものをすぐに送信する機会を失ってしまいます。クライアントがキャッシュの状態をサーバに知らせることができるようにするproposal[771] もありましたが、プライバシーの問題がありました。

下の図20.17からわかるように、サーバープッシュを利用しているサイトの割合は非常に少ない。

クライアント	HTTP/2ページ	HTTP/2(%)	gQUICページ	gQUIC(%)
デスクトップ	44,257	0.85%	204	0.04%
モバイル	62,849	1.06%	326	0.06%

図22.16. Pages using HTTP/2 or gQUIC server push.

図22.18、図22.19のプッシュされたアセットの分布をさらに見てみると、デスクトップでは合計サイズ140KBのリソースを4個以下、モバイルではサイズ184KBのリソースを3個以下でプッシュしているサイトが半数を占めています。gQUICの場合は、デスクトップが7以下、モバイルが2となっています。最悪の違反ページは、デスクトップでgQUICを上回る *41* アセット をプッシュしています。

パーセンタイル	HTTP/2	サイズ(KB)	gQUIC	サイズ(KB)
10	1	3.95	1	15.83
25	2	36.32	3	35.93
50	4	139.58	7	111.96
75	8	346.70	21	203.59
90	17	440.08	41	390.91

図22.17. Distribution of pushed assets on desktop.

771. https://tools.ietf.org/html/draft-ietf-httpbis-cache-digest-05

パーセンタイル	HTTP/2	サイズ(KB)	gQUIC	サイズ(KB)
10	1	15.48	1	0.06
25	1	36.34	1	0.06
50	3	183.83	2	24.06
75	10	225.41	5	204.65
90	12	351.05	18	453.57

図22.18. Distribution of pushed assets on mobile.

図 22.20のコンテンツタイプ別のプッシュの頻度を見ると、90%のページがスクリプトをプッシュし、56%がCSSをプッシュしていることがわかります。これらのファイルは通常、ページをレンダリングするためのクリティカルパス上にある小さなファイルである可能性があるため、これは理にかなっています。

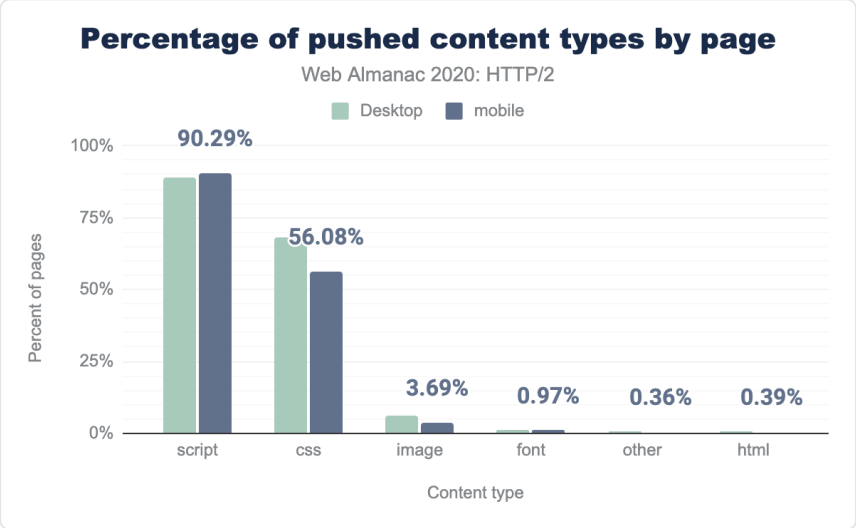


図22.19. 特定のコンテンツタイプをプッシュしているページの割合

普及率が低いことと、プッシュされたリソースのうち実際に有用なものがどれだけ少ないか（つまり、キャッシュされていないリクエストにマッチするか）を測定した後、GoogleはHTTP/2とgQUICの両方について、Chromeからプッシュサポートを削除する意向[772]を発表しました。また、ChromeはHTTP/3のプッシュを実装していません。

772. https://groups.google.com/a/chromium.org/g/blink-dev/c/K3rYLvmQUBY/m/vOWBKZGoAQAJ

これらすべての問題にもかかわらず、サーバープッシュが改善点を提供できる状況があります。理想的な使用例は、HTMLレスポンス自体よりもはるかに早い段階でプッシュの約束を送信できることです。これが利益をもたらすシナリオは、CDNが使用されている場合[773]です。CDNがリクエストを受信してからオリジンからのレスポンスを受信するまでの「デッドタイム」をインテリジェントに利用して、TCP接続をウォームアップし、CDNですでにキャッシュされているアセットをプッシュできます。

しかし、アセットがプッシュされるべきであることをCDNエッジサーバにシグナルする方法は標準化されていなかった。その代わりに、プリロードHTTPリンクヘッダーを再利用した実装が行われていました。このシンプルなアプローチはエレガントに見えますが、実際のコンテンツの準備が整う前にヘッダーが送信されない限り、HTMLが生成されるまでのデッドタイムを利用していません。それは、HTMLがエッジで受信されるとリソースをプッシュするようにエッジをトリガーし、HTMLの配信と競合することになります。

テスト中の代替案として、RFC8297[774]があり、これは情報を提供する `103(Early Hints)` レスポンスを定義している。これにより、サーバが完全なレスポンスヘッダーを生成するのを待たずに、すぐにヘッダーを送ることができるようになる。これは、オリジンがCDNにプッシュされたリソースを提案したり、CDNがクライアントにフェッチする必要のあるリソースを警告したりするために使用できます。しかし現時点では、クライアントとサーバの両方の観点からのサポートは非常に低く、しかし、成長しています[775]。

より良いプロトコルに到達するために

クライアントとサーバがHTTP/1.1とHTTP/2の両方をサポートしているとしましょう。どちらを使うかはどうやって選ぶのでしょうか？　最も一般的な方法はTLSApplication Layer Protocol Negotiation[776] (ALPN) で、クライアントはサポートするプロトコルのリストをサーバに送信し、サーバはその接続に使用するプロトコルを選択します。 ブラウザはHTTPS接続を設定する際にTLSパラメーターをネゴシエートする必要があるため、同時にHTTPバージョンもネゴシエートできます。HTTP/2とHTTP/1.1の両方が同じTCPポート（443）から提供されるので、ブラウザは接続を開く前にこの選択をする必要はありません。

プロトコルが同じポートにない場合、異なるトランスポートプロトコルを使用している場合（TCPとQUIC）、あるいはTLSを使用していない場合は、このようなことはできません。これらのシナリオでは、最初に接続したポートで利用可能なものから始めて、他のオプションを見つけます。HTTPには、接続後にオリジンのプロトコルを変更するための2つのメカニズムが定義されています。 `Upgrade` と `Alt-SVC` です。

773. *https://medium.com/@ananner/http-2-server-push-performance-a-further-akamai-case-study-7a17573a3317*
774. *https://tools.ietf.org/html/rfc8297*
775. *https://www.fastly.com/blog/beyond-server-push-experimenting-with-the-103-early-hints-status-code*
776. *https://ja.wikipedia.org/wiki/Application-Layer_Protocol_Negotiation*

`Upgrade`

Upgradeヘッダーは長い間HTTPの一部でした。 HTTP/1.xでは、 `Upgrade` によってクライアントはあるプロトコルを使ってリクエストを行い、別のプロトコル（HTTP/2のような）をサポートしていることを示すことができます。 サーバが提供されたプロトコルもサポートしている場合、サーバはステータス101(`Switching Protocols`)で応答し、新しいプロトコルでリクエストに応答します。 そうでない場合、サーバはHTTP/1.xでリクエストに応答する。 サーバは応答の `Upgrade` ヘッダーを使って別のプロトコルのサポートを示すことができます。

`Upgrade` の最も一般的なアプリケーションはWebSockets[777]です。HTTP/2もまた、暗号化されていないクリアテキストモードで使用するための `Upgrade` パスを定義しています。しかし、Webブラウザでこの機能はサポートされていません。したがって、我々のデータセットに含まれるHTTP/1.1のクリアテキストリクエストのうち、応答に `Upgrade` ヘッダーが含まれていたのは3%未満であったことは驚くに値しません。非常に少数のTLSを使用するリクエスト（HTTP/2の0.0011%、HTTP/1.1の0.064%）もレスポンスで `Upgrade` ヘッダーを受け取っていました。これらは、HTTP/2を話したりTLSを終了させたりしているが、 `Upgrade` ヘッダーを適切に削除していない仲介者の背後にあるクリアテキストHTTP/1.1サーバーである可能性が高いです。

代替サービス

Alternative Services(`Alt-SVC`)は、HTTPオリジンが同じコンテンツを提供する他のエンドポイントを示すことを可能にします。 珍しいことですが、HTTP/2はサイトのHTTP/1.1サービスとは異なるポートや異なるホストに位置しているかもしれません。さらに重要なことは、HTTP/3はQUIC（つまりUDP）を使用するので以前のバージョンのHTTPはTCPを使用していましたが、HTTP/3は常にHTTP/1.xやHTTP/2サービスとは異なるエンドポイントに位置しているということです。

`Alt-SVC` を使用する場合、クライアントは通常通りにオリジンへのリクエストを行う。 しかしサーバーがヘッダーを含むか、代替案のリストを含むフレームを送る場合、クライアントは他のエンドポイントへの新しい接続を行い、そのオリジンへの将来のリクエストにそれを使うことができます。

当然のことながら、 `Alt-SVC` の使用はほとんどの場合、高度なHTTPバージョンを使用しているサービスから発見されています。 HTTP/1.xリクエストの0.055%と比較してHTTP/2リクエストの12.0%とgQUICリクエストの60.1%がレスポンスで `Alt-Svc` ヘッダーを受信しています。ここでの我々の方法論は `Alt-Svc` ヘッダーのみをキャプチャしており、HTTP/2の `ALTSVC` フレームはキャプチャしていないことに注意してください。

777. https://developer.mozilla.org/docs/Web/API/WebSockets_API

`Alt-Svc`は全く異なるホストを指すことができますが、この機能のサポートはブラウザによって異なります。`Alt-Svc`ヘッダーのうち、異なるホスト名のエンドポイントを広告しているのはわずか4.71%でしたが、これらはほぼすべて（99.5%）Google Ads上のgQUICとHTTP/3サポートを広告していました。Google ChromeはHTTP/2のクロスホストの`Alt-SVC`広告を無視するので、他の多くのインスタンスは無視されているでしょう。

クロスホストHTTP/2のサポートの希少性を考えると、`Alt-Svc`を使用したHTTP/2エンドポイントの広告が実質的にゼロ（0.007%）であったことは驚くべきことではありません。`Alt-Svc`は通常、HTTP/3（`Alt-Svc`ヘッダーの74.6%）やgQUIC（`Alt-Svc`ヘッダーの38.7%）のサポートを示すために使用されていました。

HTTP/3の未来に向けて

HTTP/2は強力なプロトコルで、わずか数年でかなりの採用が見られました。しかし、QUICを利用したHTTP/3がすでに登場しています。4年以上の歳月をかけて作られたHTTPの次のバージョンは、IETFでほぼ標準化されています（2021年の前半に予定されています）。この時点で、すでに多くのQUICとHTTP/3の実装が利用可能[778]で、サーバ用とブラウザ用の両方があります。これらは比較的成熟しているとはいえ、まだ実験的な状態にあるといえます。

これは、HTTP Archiveデータの使用数に反映されていますが、HTTP/3リクエストは全く確認されていませんでした。これは少し奇妙に思えるかもしれませんが、Cloudflare[779]はGoogleやFacebookと同様、実験的にHTTP/3をサポートしていました。これは主に、データ収集時にChromeがデフォルトでHTTP/3プロトコルを有効にしていなかったためです。

しかし、古いgQUICの数字でさえ、全体のリクエストの2%未満を占める比較的控えめなものでした。これは、gQUICのほとんどがGoogleとアカマイによって導入され、その他の関係者はIETF QUICを待っていたため、予想されていたことです。このように、gQUICはまもなくHTTP/3に完全に置き換えられると予想されています。

1.72%

図22.20. デスクトップとモバイルでgQUICを使用するリクエストの割合

この採用率の低さは、WebページをロードするためのgQUICとHTTP/3の使用状況を反映しているに過ぎないことに注意が必要です。すでに数年前から、Facebookはネイティブアプリケーション内でデータをロードするために、IETF QUICとHTTP/3をより広範囲に展開してい

778. https://github.com/quicwg/base-drafts/wiki/Implementations
779. https://blog.cloudflare.com/experiment-with-http-3-using-nginx-and-quiche/

ます。QUICとHTTP/3はすでに彼らの総インターネットトラフィックの75%以上を占めています[780]。HTTP/3がまだ始まったばかりであることは明らかです。

今、あなたは疑問に思うかもしれません: もし誰もが既にHTTP/2を使っていないのであれば、なぜすぐにHTTP/3が必要なのでしょうか？　実際にはどのような利点があるのでしょうか？　内部メカニズムを詳しく見てみましょう。

QUICとHTTP/3

インターネット上に新しいトランスポートプロトコルを展開しようとする過去の試みは、Stream Control Transmission Protocol[781] (SCTP)のように困難であることが証明されています。QUICはUDPの上で動作する新しいトランスポートプロトコルです。QUICは、信頼性の高い順番配信やネットワークへのフラッディングを防ぐ輻輳制御など、TCPと同様の機能を提供します。

HTTP/1.0とHTTP/2で説明したように、HTTP/2は1つの接続の上に複数の異なるストリームを *多重化* しています。TCP自体はこの事実に気付いていないため、パケットロスや遅延が発生したときに非効率やパフォーマンスに影響を与えてしまいます。 *head-of-line blocking* (HOL blocking)として知られるこの問題の詳細については、こちらを参照してください[782]。

QUICはHTTP/2のストリームをトランスポート層に落とし、ストリームごとの損失検出と再送を行うことで、HOLブロッキングの問題を解決します。つまり、HTTP/2をQUICの上に置けばいいんですよね？　さて、TCPとHTTP/2でフロー制御を行うことで発生する問題のいくつかをすでに述べました。2つの独立した競合するストリーミングシステムを重ねて実行することは、さらなる問題となるでしょう。さらに、QUICストリームは独立しているので、HTTP/2がシステムのいくつかに要求する厳格な順序付けの要件を混乱させることになります。

最終的には、QUICを直接使用する新しいHTTPバージョンを定義する方が簡単だと判断され、HTTP/3が誕生しました。その高レベルの機能はHTTP/2から知っているものと非常に似ていますが、内部の実装メカニズムはかなり異なっています。HPACKヘッダー圧縮はQPACK[783]に置き換えられ、圧縮効率とHOLブロッキングリスクのトレードオフを手動で調整[784]することが可能になり、優先順位付けシステムはよりシンプルなものに置き換え[785]られています。後者はHTTP/2にバックポートすることも可能で、この記事で以前に議論した優先順位付けの問題を解決するのに役立つかもしれません。HTTP/3はサーバプッシュの仕組みを提供し続けていますが、例えばChromeはそれを実装していません。

QUICのもう1つの利点は、基盤となるネットワークが変化しても、接続を移行して維持でき

780. *https://engineering.fb.com/2020/10/21/networking-traffic/how-facebook-is-bringing-quic-to-billions/*
781. *https://ja.wikipedia.org/wiki/Stream_Control_Transmission_Protocol*
782. *https://github.com/rmarx/holblocking-blogpost*
783. *https://tools.ietf.org/html/draft-ietf-quic-qpack-19*
784. *https://blog.litespeedtech.com/tag/quic-header-compression-design-team/*
785. *https://blog.cloudflare.com/adopting-a-new-approach-to-http-prioritization/*

ることです。典型的な例は、いわゆる「駐車場問題」です。あなたのスマートフォンが職場のWi-Fiネットワークに接続されていて、大きなファイルをダウンロードし始めたところを想像してみてください。あなたがWi-Fiの範囲から離れると、あなたのスマートフォンは自動的に新しい5Gセルラーネットワークに切り替わります。普通の古いTCPでは、接続が切れて中断してしまいます。しかしQUICはよりスマートで、ネットワークの変化に強く、クライアントが中断することなく切り替えられるようにアクティブな *接続マイグレーション* 機能を提供しています。

最後に、HTTP/1.1とHTTP/2の保護にはすでにTLSが使われています。しかし、QUICはTLS1.3を深く統合しており、HTTP/3のデータとQUICのパケット番号などのメタデータの両方を保護しています。このようにTLSを使用することで、エンドユーザーのプライバシーとセキュリティが向上し、プロトコルの継続的な進化が容易になります。トランスポートと暗号化ハンドシェイクを組み合わせることで、TCPの最低2回のRTTと最悪の場合4回のRTTが必要なのに対し、接続設定は1回で完了します。場合によっては、QUICはさらに一歩進んで、 *0-RTT* と呼ばれる最初のメッセージと共にHTTPデータを送信します。これらの高速な接続設定時間は、HTTP/3がHTTP/2よりも優れた性能を発揮するのに役立つと期待されています。

では、HTTP/3は役に立つのでしょうか？

表面上では、HTTP/3はHTTP/2と実際にはそれほど違いはありません。大きな機能は追加されていませんが、主に既存の機能を変更しています。真の改善点はQUICによるもので、接続設定の高速化、ロバスト性の向上、パケットロスへの耐性の向上などが挙げられます。このようにHTTP/3は、より悪いネットワークではHTTP/2よりも優れたパフォーマンスを発揮し、より速いシステムでは非常に似たようなパフォーマンスを提供することが期待されています。しかし、それはウェブコミュニティがHTTP/3を動作させることができればの話ですが、実際には難しいかもしれません。

HTTP/3の導入と検出

QUICとHTTP/3はUDPで動作するので、HTTP/1.1やHTTP/2のように単純ではありません。通常、HTTP/3クライアントは、まずQUICがサーバで利用可能であることを発見しなければなりません。このために推奨される方法は、HTTP Alternative Services です。ウェブサイトへの最初の訪問では、クライアントはTCPを使ってサーバに接続します。その後、`Alt-SVC` を通じてHTTP/3が利用可能であることを発見し、新しいQUIC接続を設定できます。`Alt-SVC` のエントリはキャッシュされ、後続の訪問でTCPのステップを回避できますが、エントリは最終的に陳腐化し再検証が必要になります。これはおそらく各ドメインごとに個別に行う必要があり、ほとんどのページロードはHTTP/1.1、HTTP/2、HTTP/3のミックスを使用することになるでしょう。

しかし、サーバーがQUICやHTTP/3をサポートしていることがわかっていても、その間のネ

ットワークがブロックしてしまう可能性があります。UDPトラフィックはDDoS攻撃で一般的に使用されており、例えば多くの企業ネットワークではデフォルトでブロックされています。QUICには例外があるかもしれませんが、暗号化されているため、ファイアウォールがトラフィックを評価することは困難です。これらの問題に対する潜在的な解決策はありますが、当面の間、QUICは443のようなよく知られたポートで成功する可能性が高いと予想されています。そして、それは完全にQUICをブロックされていることが完全に可能です。実際には、QUICが失敗した場合、クライアントはTCPにフォールバックするため洗練されたメカニズムを使用する可能性が高い。1つの選択肢は、TCPとQUICの両方の接続を「レース」して、最初に完了した方を使用することです。

TCPステップを必要とせずにHTTP/3を検出する方法を定義するための進行中の作業があります。UDPのブロッキングの問題はTCPベースのHTTPの利用を意味しそうなので、これは最適化と考えるべきでしょう。HTTPS DNSレコード[786] はHTTP Alternative Servicesと似ており、いくつかのCDNはすでにこれらのレコードを使って実験中[787] です。 長期的には、ほとんどのサーバがHTTP/3を提供するようになったら、ブラウザはデフォルトでそれを試みるように切り替えるかもしれません。

	HTTP/1.x		HTTP/2	
TLSバージョン	デスクトップ	モバイル	デスクトップ	モバイル
不明	4.06%	4.03%	5.05%	7.28%
TLS 1.2	26.56%	24.75%	23.12%	23.14%
TLS 1.3	5.25%	5.11%	35.78%	35.54%

図22.21. TLS adoption by HTTP version.

図22.21に示すように、QUICはTLS1.3に依存しており、リクエストの約41%で使用されています。これでは59%のリクエストがHTTP/3をサポートするためにTLSスタックを更新する必要があることになります。

HTTP/3はまだですか？

では、いつからHTTP/3とQUICを本格的に使えるようになるのでしょうか？　まだまだですが、うまくいけばもうすぐです。多数の成熟したオープンソース実装[788]があり、主要なブラウザは実験的にサポートしています。しかし、代表的なサーバのほとんどは、いくつかの遅れに苦しんでいます。nginxは他のスタックより少し遅れており、Apacheは公式のサポート

786. https://tools.ietf.org/html/draft-ietf-dnsop-svcb-https
787. https://blog.cloudflare.com/speeding-up-https-and-http-3-negotiation-with-dns/
788. https://github.com/quicwg/base-drafts/wiki/Implementations

を発表しておらず、NodeJSはOpenSSLに依存しており、QUICのサポートはいつでもすぐには追加されません[789]。それでも、2021年を通してHTTP/3とQUICのデプロイが増加することを期待しています。

HTTP/3とQUICは、新しいHTTP優先順位付けシステム、HOLブロッキング除去、0-RTT接続確立など、強力なパフォーマンスとセキュリティ機能を備えた高度なプロトコルです。この高度なプロトコルは、HTTP/2のように、正しく展開して使用することが難しいプロトコルでもあります。初期の導入は、主にCloudflare、Fastly、AkamaiなどのCDNの早期導入によって行われると予測しています。これはおそらく、HTTP/2トラフィックの大部分が2021年に自動的にHTTP/3にアップグレードされることを意味しており、新しいプロトコルに大きなトラフィックシェアを与えることになるでしょう。小規模な導入がいつ、そしてそれに追随するかどうかは、答えを出すのが難しいところです。おそらく、HTTP/3、HTTP/2、さらにはHTTP/1.1の健全な組み合わせが今後何年にもわたってウェブ上で見られることになるでしょう。

結論

今年は、HTTP/2が支配的なプロトコルとなり、全リクエストの64%を提供しており、この1年で10%ポイント増加しました。ホームページはHTTP/2の採用率が13%増加し、HTTP/1.1とHTTP/2で提供されるページが均等に分かれています。ホームページを提供するためにCDNを使用すると、HTTP/2の採用率は80%まで上昇しますが、CDNを使用しない場合は30%となっています。ApacheやIISなどの古いサーバは、HTTP/2やTLSへのアップグレードに抵抗があります。大きな成功は、HTTP/2接続多重化によるWebサイトの接続使用量の減少です。2016年の接続数の中央値は23だったのに対し、2020年は13でした。

HTTP/2の優先順位付けとサーバープッシュは、大規模な展開を行うにはより複雑であることが判明しました。特定の実装の下では、これらの機能は明らかにパフォーマンス上の利点を示しています。しかし、これらの機能を効果的に使用するために、既存のサーバーをデプロイしてチューニングすることには大きな障壁があります。優先順位付けを効果的にサポートしていないCDNの割合が依然として大きい。また、ブラウザの一貫したサポートにも問題があります。

HTTP/3はすぐそこまで来ています。普及率を追い、発見メカニズムがどのように進化していくのか、そしてどの新機能が成功裏に導入されるのかを見極めることは非常に興味深いことでしょう。来年のWeb Almanacでは、興味深い新データが出てくることを期待しています。

789. *https://github.com/openssl/openssl/pull/8797*

著者

Andrew Galloni

@dot_js　dotjs

AndrewはCloudflare[790]で働き、ウェブの高速化と安全性の向上に貢献しています。彼は、エンドユーザーのウェブサイトのパフォーマンスを向上させるために、新しいプロトコルやアセット配信の展開、測定、改善に時間を費やしています。

Robin Marx

@programmingart　@programmingart.bsky.social　rmarx　rmarx

http://internetonmars.org

Robinはベルギーのハッセルト大学[791]のウェブプロトコルとパフォーマンスの研究者です。彼は、qlogやqvis[792]のようなツールを作成することで、QUICやHTTP/3を使えるようにすることに取り組んでいます。

Mike Bishop

MikeBishop

QUIC Working Group[793]のHTTP/3のエディタ。Akamai[794]のファウンドリグループのアーキテクト。

790. https://www.cloudflare.com/
791. https://www.uhasselt.be/edm
792. https://github.com/quiclog
793. https://quicwg.org/
794. https://www.akamai.com/

付属資料 A
方法論

概要

Web Almanacは、HTTP Archive[795]によって組織されたプロジェクトです。HTTP Archiveは、ウェブがどのように構築されているかを追跡することを使命として、2010年に Steve Souders によって開始されました。何百万ものウェブページの構成を毎月評価し、そのテラバイトのメタデータをBigQuery[796]で分析できるようにしています。

Web Almanacの使命は、ウェブの状態に関する一般的な知識の年次リポジトリになることです。私たちの目標は、主題の専門家が文脈に沿った洞察を提供することで、HTTP Archiveの

795. https://httparchive.org
796. https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data

データウェアハウスをウェブコミュニティがさらに利用しやすいものにすることです。

Web Almanacの2020年版は、ページコンテンツ、体験、公開、配信の4つの部に分かれています。各部の、いくつかの章では、その全体的なテーマをさまざまな角度から探求しています。例えば、部IIではユーザー体験を、パフォーマンスやセキュリティ、アクセシビリティの各章で、さまざまな角度から探求しています。

データセットについて

HTTP Archive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。Web Almanacの2020年版については、この章で特に断りのない限り、すべてのメトリクスは2020年8月のクロールからソースを得ています。これらの結果は、BigQuery上で接頭語のテーブル `2020_08_01` で公開[797]できます。

Web Almanacで紹介されているすべてのメトリクスは、BigQuery上のデータセットを使用して一般に再現可能です。すべての章で使用されているクエリは、GitHubリポジトリ[798]で閲覧できます。

BigQueryはテラバイト単位で課金されるため、これらのクエリの中には非常に大規模なものもあり、自分で実行するには費用がかかる[799]可能性があることに注意してください。支出のコントロールについては、Tim Kadlec氏の投稿Using BigQuery Without Breaking the Bank[800]を参照してください。

例えば、デスクトップページとモバイルページあたりのJavaScriptのバイト数の中央値を把握するには、01_01b.sql[801]を参照してください。

```
#standardSQL
# 01_01b: Distribution of JS bytes by client
SELECT
  percentile,
  _TABLE_SUFFIX AS client,
  APPROX_QUANTILES(ROUND(bytesJs / 1024, 2),
1000)[OFFSET(percentile * 10)] AS js_kbytes
```

797. https://github.com/HTTPArchive/httparchive.org/blob/main/docs/gettingstarted_bigquery.md
798. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2020
799. https://cloud.google.com/bigquery/pricing
800. https://timkadlec.com/remembers/2019-12-10-using-bigquery-without-breaking-the-bank/
801. https://github.com/HTTPArchive/almanac.httparchive.org/blob/main/sql/2019/javascript/01_01b.sql

```
  FROM
    `httparchive.summary_pages.2019_07_01_*`,
    UNNEST([10, 25, 50, 75, 90]) AS percentile
  GROUP BY
    percentile,
    client
  ORDER BY
    percentile,
    client
```

各指標の結果は、JavaScriptの結果[802]のように、章ごとのスプレッドシートで公開されています。各章の下までスクロールすると、クエリ、結果、読者からのコメントへのリンクが表示されます。

ウェブサイト

データセットには7,546,709件のウェブサイトが含まれている。そのうち、6,347,919件がモバイルサイト、5,593,642件がデスクトップサイトである。ほとんどのウェブサイトがモバイルとデスクトップの両方のサブセットに含まれている。

HTTP Archiveは、Chrome UXレポート からウェブサイトのURLを取得しています。Chrome UXレポートはGoogleが公開しているデータセットで、Chromeユーザーが積極的に訪問している数百万のウェブサイトのユーザー体験を集計しています。これにより、実際のウェブ利用状況を反映した最新のウェブサイトのリストが得られます。Chrome UXレポートデータセットにはフォーム ファクター ディメンションが含まれており、デスクトップ ユーザーやモバイル ユーザーがアクセスしたすべてのWebサイトを取得するために使用します。

Web Almanacが使用した2020年8月のHTTPアーカイブのクロールでは、Webサイトのリストに、最新のChrome UXレポートのリリースを使用しました。202006データセットは2020年7月14日にリリースされたもので、6月中に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。

分析したウェブサイトの数は、2019 Web Almanacに掲載されているウェブサイトと比較して、20〜30%の増加が見られました。この増加は、Paul Calvanoが、Growth of the Web in 2020[803]の投稿で分析しています。

リソースの制限のため、HTTP Archiveでは、Chrome UXレポートで各ウェブサイトの1ペー

802. https://docs.google.com/spreadsheets/d/1kBTglETN_V9UjKqK_EFmFjRexJnQOmLLr-I2Tkotvic/edit?usp=sharing
803. https://paulcalvano.com/2020-09-29-growth-of-the-web-in-2020/

ジしかテストできません。これを調整するために、ホームページのみが含まれています。ホームページは必ずしもウェブサイト全体を代表するものではないため、結果に多少の偏りが生じることに注意してください。

HTTP Archiveは、データセンターからウェブサイトをテストし、実際のユーザー体験からデータを収集しないという意味で、ラボテストツールとも考えられています。そのため、すべてのページはログアウト状態で空のキャッシュを使ってテストされており、実際のユーザーがどのようにアクセスするかを反映していない可能性があります。

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数千のメトリクスを収集します。これには、ページあたりのバイト数、ページがHTTPSで読み込まれたかどうか、個々のリクエストヘッダーと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ています。これらのメトリクスの大部分は、WebPageTest によって提供されており、各ウェブサイトのテストランナーとして機能します。

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。例えば、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、ページに対する監査を実行するために使用されます。以下のツールのセクションでは、これらのツールについて詳しく説明しています。

研究室のデータセットに固有の制限を回避するために、Web Almanacでは、Chrome UXレポートを利用して、特にウェブパフォーマンスの分野でのユーザー体験に関するメトリクスを提供しています。

メトリクスの中には、完全に手の届かないものもあります。例えば、ウェブサイトの構築に使用されたツールを検出できるとは限りません。ウェブサイトがcreat-react-appを使って構築されている場合、Reactフレームワークを使っていることはわかりますが、特定のビルドツールが使われているとは限りません。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、その使用状況を測定することはできません。

その他のメトリクスは、必ずしも測定が不可能ではないかもしれませんが、測定が困難であったり、信頼性が低いものもあります。例えば、Webデザインの側面は本質的に視覚的であり、ページに押し付けがましいモーダルダイアログがあるかどうかなど、定量化するのは難しいかもしれません。

ツール

Web Almanacは、以下のオープンソース・ツールの助けを借りて実現しています。

WebPageTest

WebPageTest[804]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のバックボーンです。WebPageTestはプライベートインスタンス[805]とプライベートテストエージェントを使用しており、これは各Webページをテストする実際のブラウザです。デスクトップとモバイルのウェブサイトは、異なる構成でテストされます。

804. https://www.webpagetest.org/
805. https://docs.webpagetest.org/private-instances/

設定	デスクトップ	モバイル
デバイス	Linux VM	Emulated 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/84.0.4147.105 Safari/537.36 PTST/200805.230825	Mozilla/5.0 (Linux; Android 6.0.1; Moto G (4) Build/MPJ24.139-64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.146 Mobile Safari/537.36 PTST/200815.130813
場所	アメリカのカリフォルニア州レッドウッドシティ アメリカのオレゴン州ダレス	Redwood City, California, USA The Dalles, Oregon, USA
接続方法	ケーブル(5/1 Mbps 28ms RTT)	3G (1.600/0.768 Mbps 300ms RTT)
ビューポート	1024 x 768px	512 x 360px

デスクトップのWebサイトは、Linux VM上のデスクトップChrome環境内から実行されます。ネットワーク速度はケーブル接続と同等です。

モバイルWebサイトは、3G接続と同等のネットワーク速度を持つエミュレートされたMoto G4デバイス上のモバイルChrome環境から実行されます。エミュレートされたモバイルユーザーエージェントはChrome65と自称していますが、実際は Chrome84であることに注意してください。

テストが実行される場所は2つあります。カリフォルニア州とオレゴン州です。HTTP Archiveは、カリフォルニア州にあるInternet Archive[806]のデータセンターにある独自のテストエージェントハードウェアを維持しています。オレゴン州にあるGoogle Cloud Platform[807]のus-west-1にあるテストエージェントは、必要に応じて追加されています。

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して維持され、カスタムメトリクス[808]で強化されています。これらは、テストの最後に各ウェブサイトで評価されるJavaScriptのスニペットです。多くのデータアナリストの貢献[809]に感謝します、特に献身的な努力[810]をしたTony McCreathのおかげで、Web Almanacの2020年版では、HTTPアーカイブのテストインフラストラクチャの機能が3,000行以上の新しいコードで大幅に拡張されました。

各テストの結果は、HARファイル[811]として公開され、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す。

Lighthouse

Lighthouse[812]は、Googleが構築した自動化されたウェブサイト品質保証ツールです。最適化されていない画像やアクセスできないコンテンツなどのユーザー体験のアンチパターンが含まれていないことを確認するためにウェブページを監査します。

HTTP Archiveは、すべてのモバイルWebページでLighthouseの最新バージョンを実行しています - リソースが限られているため、デスクトップページは含まれていません。2020年8月のクロール時点で、HTTP Archiveは6.2.0[813]バージョンのLighthouseを使用していました。

LighthouseはWebPageTestの中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	値
CPUスローダウン	1x
Downloadスループット	1.6 Mbps
Uploadスループット	0.768 Mbps
RTT	150 ms

806. https://archive.org
807. https://cloud.google.com/compute/docs/regions-zones/#locations
808. https://github.com/HTTPArchive/custom-metrics
809. https://github.com/HTTPArchive/legacy.httparchive.org/commits/master/custom_metrics
810. https://github.com/HTTPArchive/legacy.httparchive.org/pulls?q=is%3Apr+author%3ATiggerito+sort%3Acreated-asc
811. https://en.wikipedia.org/wiki/HAR_(file_format)
812. https://developers.google.com/web/tools/lighthouse/
813. https://github.com/GoogleChrome/lighthouse/releases/tag/v6.2.0

LighthouseとHTTP Archiveで利用可能な監査の詳細については、Lighthouse開発者向けドキュメント[814]を参照してください。

Wappalyzer

Wappalyzer[815]は、ウェブページで使用されている技術を検出するためのツールです。64のカテゴリ[816]があり、JavaScriptフレームワークからCMSプラットフォーム、さらには暗号通貨の採掘者に至るまで、テストされた技術の範囲があります。1,400以上の技術がサポートされています。

HTTP Archiveは、すべてのウェブページに対して最新バージョンのWappalyzerを実行します。2020年8月現在、Web AlmanacはWappalyzerの6.2.0バージョン[817]を使用しています。

Wappalyzerは、WordPress、Bootstrap、jQueryのような開発者ツールの人気を分析する多くの章を強力にしています。例えば、Eコマース と CMS の各章は、それぞれの Eコマース[818] と CMS[819] の各チャプターは、Wappalyzer が検出した技術のカテゴリ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限界があります。その結果の妥当性は、その検出メカニズムがどれだけ正確であるかに常に依存します。Web Almanacでは、Wappalyzerが使用されているすべての章に注意書きが追加されますが、特定の理由により、その分析が正確でない場合があります。

Chrome UXレポート

Chrome UXレポート[820] は、実際のChromeユーザーの体験をまとめた公開データセットです。エクスペリエンスは、`https://www.example.com` などのように、ウェブサイトの起源によってグループ化されています。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、レイアウトの安定性などのUXメトリクスの分布が含まれています。月ごとのグループ化に加えて、国レベルの地理、フォームファクター（デスクトップ、モバイル、タブレット）、有効な接続タイプ（4G、3Gなど）などの大きさ、速度によっても経験をスライスすることができます。

Chrome UXレポートの実世界のユーザー体験データを参照するWeb Almanacメトリクスについては、2020年8月のデータセット（202008）を使用しています。

データセットの詳細については、BigQueryでChrome UXレポートを使用する[821]のガイド

814. https://developers.google.com/web/tools/lighthouse/
815. https://www.wappalyzer.com/
816. https://www.wappalyzer.com/technologies
817. https://github.com/AliasIO/Wappalyzer/releases/tag/v6.2.0
818. https://www.wappalyzer.com/categories/ecommerce
819. https://www.wappalyzer.com/categories/cms
820. https://developers.google.com/web/tools/chrome-user-experience-report
821. https://web.dev/chrome-ux-report-bigquery

web.dev[822]を参照してください。

サードパーティウェブ

サードパーティウェブ[823]は、Patrick Hulceの研究プロジェクトで、2019サードパーティの章では、HTTP ArchiveとLighthouseのデータを使用して、サードパーティのリソースがウェブに与える影響を特定して分析しています。

ドメインは、少なくとも50のユニークなページに表示されている場合、サードパーティのプロバイダであるとみなされます。また、このプロジェクトでは、広告、分析、ソーシャルなどのカテゴリで、それぞれのサービスごとにプロバイダーをグループ化しています。

Web Almanacのいくつかの章では、サードパーティの影響を理解するために、このデータセットのドメインとカテゴリを使用しています。

Rework CSS

Rework CSS[824]はJavaScriptベースのCSSパーサーです。スタイルシート全体を受け取り、個々のスタイルルール、セレクタ、ディレクティブ、値を区別するJSONエンコードされたオブジェクトを生成します。

この特別な目的のツールは、CSS章の多くのメトリクスの精度を大幅に向上させました。各ページのすべての外部スタイルシートとインラインスタイルブロックのCSSを解析し、解析可能な状態にするために問い合わせを行いました。BigQuery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[825]を参照してください。

Rework Utils

今年のCSSの章では、Lea VerouがCSSの状態をかなり詳細に見ており、100以上のクエリ[826]にわたっています。視点を変えれば、2019年の2.5倍のクエリ数になります。この規模の分析を実現可能にするために、LeaはRework Utils[827]をオープンソース化しました。これらのユーティリティは、CSSの洞察をより簡単に抽出するための有用なスクリプトを提供することで、ReworkからJSONデータを次のレベルに引き上げます。CSSの章で見る統計情報のほとんどは、これらのスクリプトによって提供されています。

822. *https://web.dev/*
823. *https://www.thirdpartyweb.today/*
824. *https://github.com/reworkcss/css*
825. *https://discuss.httparchive.org/t/analyzing-stylesheets-with-a-js-based-parser/1683*
826. *https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2020/css*
827. *https://github.com/LeaVerou/rework-utils*

Parsel

Parsel[828]はCSSセレクターの構文解析と詳細度の計算ツールで、元々はCSSの章をリードするLea Verouによって書かれたものであり、個別のライブラリとしてオープンソース化されています。これは、セレクターと詳細度に関連するすべてのCSSの分析で広く使われています。

分析プロセス

Web Almanacの計画と実行には、100人以上のWebコミュニティの貢献者の協力を得て、約1年を要しました。このセクションでは、Web Almanacに掲載されている章を選んだ理由、それらのメトリクスがどのようにして分析されたか、そしてどのように解釈されたかを説明します。

プランニング

2020年のWeb Almanacは、2020年6月[829]に開始されましたが、COVID-19に関連した不安や社会正義の抗議活動のため、2019年のタイムラインよりも遅くなりました。2020年のこれらの出来事やその他の出来事は、制作プロセス全体の下火となり、このようなテンポの速いプロジェクトのストレスを超えて、私たちの貢献者に多くの追加的な負担をかけました。

昨年の方法論で述べたように、。

> *Web Almanacの今後のエディションの明確な目標の1つは、著者やレビュワーとして、代表性の低い声や異質な声をより多く取り入れることを奨励することです。*

そのために、今年は著者の探し方、選び方を体系的に変えてみました。

- これまでの著者は、異なる視点のためのスペースを確保するために、再び書くことを特別に望まれていませんでした。
- 2020年の著者を支持する人たちは皆、見た目や考え方が似ている人を指名しないように特に意識するように求められました。
- 2019年の執筆者の多くはGoogleの社員であり、今年はより広いウェブコミュニティからの視点をよりバランスよく得ることを試みました。Web Alamanacの声はコミュニティそのものを反映したものであるべきであり、反響室を作らないよ

828. https://projects.verou.me/parsel/
829. https://x.com/rick_viscomi/status/1273135952848977920

うに特定の企業に偏ったものではないと考えています。

- プロジェクトのリーダーは、すべての著者の推薦を検討し、新しい視点をもたらし、地域社会の中での未発表グループの声を増幅させる著者の選定に尽力しました。

将来的には、このプロセスを繰り返していくことで、Web Almanacがより多様で包括的なプロジェクトとなり、さまざまなバックグラウンドを持つ投稿者が参加できるようにしたいと考えています。

最終的には、2020年7月にブレインストーミングとノミネートを経て、22のチャプターが固まり、各チャプターごとにコンテンツチームを編成して、執筆、レビュー、分析を行いました。

分析

2020年7月と8月に、メトリクスとチャプターの安定したリストを持って、データアナリストは、実現可能性のためにメトリクスをトリアージしました。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[830]を作成されました。

2020年8月を通して、HTTPアーカイブ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で使用するためのメタデータを収集しました。

データアナリストは、各メトリクスの結果を抽出するためのクエリを書き始めました。合計すると、何百ものクエリが手書きで書かれていました。GitHubのオープンソースクエリのリポジトリ[831]で、年別、章別にすべてのクエリを閲覧することができます。

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。著者はそれぞれの章を執筆する際には、これらの統計からウェブの状態についての枠組みを支持しました。レビュワーは著者と協力して、分析の技術的な正確さを確認しました。

読者に結果をよりわかりやすく伝えるために、ウェブ開発者やアナリストは、この章に埋め込むデータの可視化を作成しました。いくつかの可視化は、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、完全な分布を表示するのではなく、ほんの一握りのパーセンタイルのみを表示しています。特に断りのない限り、すべての分布は、平均値ではなくパーセンタイル、特に中央値（50 パーセンタイル）を使用して要約されています。

最後に、編集者は各章を修正し、簡単な文法的な誤りを修正し、読書体験全体に一貫性を持

830. https://github.com/HTTPArchive/custom-metrics
831. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2020

たせるようにしました。

将来を見据えて

Web Almanacの2020年版は毎年恒例、Webコミュニティ伝統の2回目で、内省と前向きな変化への取り組みを続けていきたいです。ここまで来れたのは、多くの献身的なコントリビューターのおかげで、記念碑的な努力をしてきました。

Web Almanacの2021年版への貢献にご興味のある方は、関心フォーム[832]にご記入ください。みんなで協力してウェブの状態を追いかけていきましょう！

832. *https://forms.gle/VRBFegGAP7d99Bhp7*

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。2020 Web Almanac々は、企画、調査、執筆、制作の段階で129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。

Abby Tsai
AbbyTsai
分析者、 開発者 と 翻訳者

Alberto Medina
@iAlbMedina
amedina
レビュワー

Aditya Pandey
@adityapandey98
adityapandey1998
adityapandey98
開発者

Alex Denning
@AlexDenning
alexdenning
https://getellipsis.com/
著作者

Adrian Roselli
@aardrian
aardrian
https://adrianroselli.com/
レビュワー

Alex Tait
@at_fresh_dev
alextait1
https://atfreshsolutions.com
著作者

Ahmad Awais
@MrAhmadAwais
ahmadawais
https://AhmadAwais.com
著作者

Alexey Pyltsyn
lex111
https://lex111.ru/
開発者、 編集者 と 翻訳者

Alan Dávalos
@AlanGDavalos
alangdm
翻訳者

Aleyda Solis
@aleyda
aleyda
https://www.aleydasolis.com/en/
著作者

Alan Kent
@akent99
alankent
https://alankent.me
レビュワー

Andrew Galloni
@dot_js
dotjs
著作者

Andy Bell
@hankchizljaw
hankchizljaw
https://hankchizljaw.com/
レビュワー

Andy Pan
andy0130tw
翻訳者

Antoine Eripret
antoineeripret
分析者

Artem Denysov
@denar90_
denar90
分析者 と レビュワー

Barry Pollard
@tunetheweb
https://webperf.social/@tunetheweb
tunetheweb.com
tunetheweb
tunetheweb
https://www.tunetheweb.com
分析者、 著作者、 開発者、 編集者、 リーダー と レビュワー

Ben Seymour
@bseymour
bseymour
benseymour
https://benseymour.com
著作者

Bharat Agarwal
agarwalbharat
https://iambharat.me
開発者

Boris Schapira
@boostmarks
borisschapira
https://boris.schapira.dev
開発者、 レビュワー と 翻訳者

Brian Kardell
@briankardell
https://toot.cafe/@bkardell
bkardell
https://bkardell.com
レビュワー

Brian Rinaldi
@remotesynth
remotesynth
https://remotesynthesis.com/
分析者

Caleb Queern
@httpsecheaders
cqueern
レビュワー

Carlos Castro
@mxcarloscastro
carloscastromx
https://carloscm.me/en/
翻訳者

Catalin Rosu
@catalinred
catalinred
https://catalin.red/
著作者、 開発者 と レビュワー

Cheng Xi
chengxicn
翻訳者

Chris Lilley
@svgeesus
svgeesus
https://svgees.us
著作者 と レビュワー

Christian Liebel
@christianliebel
https://mastodon.cloud/@christianliebel
christianliebel
christianliebel
https://christianliebel.com
著作者

Colin Bendell
@colinbendell
colinbendell
レビュワー

Dave Crossland
@davelab6
davelab6
https://fonts.google.com
レビュワー

Dave Smart
https://seocommunity.social/@dwsmart
tamethebots.com
dwsmart
davewsmart
https://tamethebots.com
レビュワー

Dave Sottimano
@dsottimano
dsottimano
https://opensourceseo.org/
レビュワー

David Fox
@theobto
foxdavidj
https://www.lookzook.com
分析者、 編集者、 リーダー と
レビュワー

Doug Sillars
@dougsillars
dougsillars
https://dougsillars.com
レビュワー

Durga Prasad Sadhanala
@dsadhanala
dsadhanala
開発者

Dustin Montgomery
@DustinMontSEO
en3r0
https://dustinmontgomery.com/
レビュワー

Edmond W. W. Chan
edmondwwchan
https://edmondwwchan.github.io/
レビュワー

Elika Etemad aka fantasai
@fantasai
fantasai
http://fantasai.inkedblade.net/
レビュワー

Emanuel Gonçalves Santana de Souza
@emanuelgsouza
emanuelgsouza
emanuelgsouza
https://emanuelgsouza.dev/
翻訳者

Eric Bailey
@ericwbailey
ericwbailey
https://ericwbailey.design/
レビュワー

Eric Portis
@etportis
ericportis.com
eeeps
https://ericportis.com
著作者

Estelle Weyl
@estellevw
estelle
http://standardista.com
レビュワー

Fatma Badri
fatmabadri
fatmabadri
https://fatmabadri.github.io/
翻訳者

Giovanni Puntil
@giovannipuntil
giopunt
レビュワー と 翻訳者

Gokulakrishnan Kalaikovan
gokulkrishh
レビュワー

Greg Brimble
@gregbrimble
GregBrimble
https://gregbrimble.com/
分析者

Greg Wolf
gregorywolf
分析者

Hemanth HM
@gnumanth
hemanth
http://h3manth.com
著作者

Henri Helvetica
@HenriHelvetica
henrihelvetica
著作者

Huang Shuo-Han
ArvinH
翻訳者

Huli
aszx87410
翻訳者

Ian Devlin
@iandevlin
iandevlin
https://iandevlin.com
著作者

Jad Joubran
@JoubranJad
jadjoubran
https://learnjavascript.online/
レビュワー

Jamie Indigo
@Jammer_Volts
not-a-robot.com
fellowhuman1101
jamie-indigo
https://not-a-robot.com
著作者

Jason Haralson
jrharalson
分析者 と 著作者

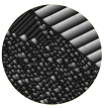
Jason Pamental
@jpamental
jpamental
https://rwt.io
著作者

Jens Oliver Meiert
@j9t
https://mas.to/@j9t
meiert.com
j9t
meiert
https://meiert.com/en/
著作者 と レビュワー

Jessica Nicolet
@jessica_nicolet
jessnicolet
https://www.jessicanicolet.com/
レビュワー

Jonathan Wold
@sirjonathan
sirjonathan
https://jonathanwold.com
レビュワー

Julia Yang
@Jules_Yang
jzyang
yangzhe
レビュワー



Jyrki Alakuijala
@jyzg
jyrkialakuijala
著作者

Karolina Szczur
@fox
thefoxis
著作者

Kate Tymoshkina
tymosh
tymosh
翻訳者

Katie Hempenius
@katiehempenius
khempenius
分析者

Laurent Devernay Satyagraha
ldevernay
laurent-devernay-satyagraha-2610b85
https://ldevernay.github.io/
レビュワー

Lea Verou
@leaverou
LeaVerou
https://lea.verou.me/
分析者 と 著作者

Leonardo Digiorgio
@simdigiorgio
chefleo
https://chefleo.dev/
翻訳者

Leonardo Zizzamia
@Zizzamia
Zizzamia
著作者 と レビュワー

Luca Versari
veluca93
著作者

Lucas Pardue
@SimmerVigor
LPardue
https://lucaspardue.com
レビュワー

Lyubomir Angelov
@angelovcode
Super-Fly
開発者

Maedah Batool
@maedahbatool
MaedahBatool
https://maedahbatool.com/
レビュワー

Mandy Michael
@Mandy_Kerr
mandymichael
https://mandymichael.com/
レビュワー

Manuel Matuzović
@mmatuzo
matuzo
https://www.matuzo.at/
レビュワー

Max Ostapenko
max-ostapenko
max-ostapenko
https://maxostapenko.com
分析者 と 開発者

Michael DiBlasio
mdiblasio
著作者

Michael King
@IPullRank
ipullrank
著作者

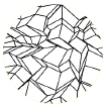
Michelle O'Connor
デザイナー

Miguel Carlos Martínez Díaz
@mcmd
mcmd
miguelcarlosmartinezdiaz
翻訳者

Mike Bishop
MikeBishop
著作者

Minko Gechev
@mgechev
mgechev
https://blog.mgechev.com/
レビュワー

Miriam Suzanne
@MiriSuzanne
mirisuzanne
https://miriamsuzanne.com/
レビュワー



Moritz Firsching
mo271
https://mo271.github.io/
著作者

Nate Dame
@seonate
natedame
レビュワー

Navaneeth Krishna
@Navanee55755217
Navaneeth-akam
レビュワー

Nicolas Goutay
@Phacks
phacks
https://phacks.dev/
レビュワー

Nicolas Hoizey
@nhoizey
nhoizey
https://nicolas-hoizey.com/
レビュワー

Nikita Dubko
@dark_mefody
MeFoDy
https://mefody.dev/
翻訳者

Noah van der Veer
@noah_aaron_vdv
noah-vdv
翻訳者

Noam Rosenthal
@nomsternom
noamr
レビュワー

Nurullah Demir
@nrllah
nrllh
https://ndemir.com
分析者 と 著作者

Olu Niyi-Awosusi
@oluoluoxenfree
oluoluoxenfree
https://olu.online/
著作者

Pascal Birchler
@swissspidy
swissspidy
swissspidy
開発者

Pascal Schilp
thepassle
レビュワー

Patrick Meenan
@patmeenan
pmeenan
https://www.webpagetest.org/
レビュワー

Paul Calvano
@paulcalvano
paulcalvano
https://paulcalvano.com
分析者、 開発者、 リーダー と
レビュワー

Pearl Latteier
@pblatteier
pearlbea
レビュワー

Pokidov N. Dmitry
@otherpunk
dooman87
https://pixboost.com
分析者

Praveen Pal
@PraveenPal4232
PraveenPal4232
https://praveenpal4232.github.io
翻訳者

Rachel Andrew
@rachelandrew
rachelandrew
https://rachelandrew.co.uk
著作者

Raghu Ramakrishnan
raghuramakrishnan71
分析者 と 著作者

Raph Levien
@raphlinus
raphlinus
https://levien.com
著作者

Renee Johnson
@reneesoffice
ernee
https://reneesvirtualoffice.com
レビュワー

Rick Viscomi
@rick_viscomi
rviscomi
https://rviscomi.dev/
分析者、 開発者、 編集者、 リーダー と レビュワー

Robin Marx
@programmingart
programmingart.bsky.social
rmarx
rmarx
http://internetonmars.org
著作者

Rockey Nebhwani
@rnebhwani
rockeynebhwani
rockeynebhwani
分析者 と 著作者

Rod Sheeter
rsheeter
レビュワー

Roel Nieskens
@PixelAmbacht
RoelN
http://pixelambacht.nl
レビュワー

Rory Hewitt
@roryhewitt3
roryhewitt
roryhewitt
https://romche.com
著作者

Sakae Kotaro
@beltway7
ksakae1216
https://ksakae1216.com/
翻訳者

Sami Boukortt
sboukortt
著作者

Saptak Sengupta
@Saptak013
saptaks
https://saptaks.website/
開発者

Sawood Alam
@ibnesayeed
ibnesayeed
https://www.cs.odu.edu/~salam/
開発者 と レビュワー

Shane Exterkamp
@Shane_Exterkamp
exterkamp
編集者 と レビュワー

Shubhie Panicker
@shubhie
spanicker
著作者

Simon Hearne
@simonhearne
simonhearne
https://simonhearne.com
著作者

Simon Pieters
@zcorpan
zcorpan
レビュワー

Stefan Matei
@smatei
smatei
https://www.advancedwebranking.com/
分析者

Sudheendra chari
@itsmesudheendra
sudheendrachari
sudheendrachari
開発者

Tamas Piros
@tpiros
tpiros
https://tamas.io
著作者

Thomas Steiner
@tomayac
tomayac
https://blog.tomayac.com/
分析者 と レビュワー

Tim Kadlec
@tkadlec
tkadlec
https://timkadlec.com/
著作者

Tom Van Goethem
@tomvangoethem
tomvangoethem
https://tom.vg/
分析者 と 著作者

Tony McCreath
@TonyMcCreath
Tiggerito
https://websiteadvantage.com.au/
分析者

Will Klancnik
notwillk
レビュワー

William Sandres
@hakacode
HakaCode
https://hakacode.github.io
翻訳者

Yana Dimova
ydimova
分析者 と 著作者

Zuckjet
@Zuckjet
Zuckjet
翻訳者

cybai
@_cybai
CYBAI
翻訳者