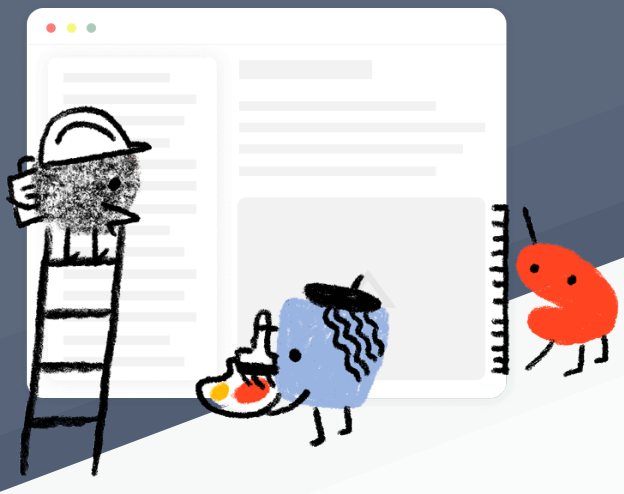
2019
Web Almanac

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: JavaScript 1
章 2: CSS 21
章 3: マークアップ 55
章 4: メディア 73
章 5: サードパーティ 99
章 6: フォント 113

部 II. ユーザー体験

章 7: パフォーマンス 131
章 8: セキュリティ 151
章 9: アクセシビリティ 175
章 10: SEO 195
章 11: PWA 215
章 12: モバイルウェブ 229

部 III. コンテンツ公開

章 13: Eコマース 245
章 14: CMS 267

部 IV. コンテンツ配信

章 15: 圧縮 291
章 16: キャッシング 305
章 17: CDN 335
章 18: Page Weight 369
章 19: リソースのヒント 383

章 20: HTTP/2 393

付属資料

方法論 413
貢献者 425

序章

オープンウェブは、驚くほど複雑で進化するテクノロジーのネットワークです。あらゆる産業やキャリアはウェブの上に構築されており、成功するためにはウェブの活気あるエコシステムに依存しています。ウェブが非常に重要であるにもかかわらず、ウェブがどのように機能しているかを理解することは、驚くほど難しいものでした。2010年以来、HTTP Archiveプロジェクトの使命は、ウェブがどのように構築されているかを追跡することであり、それは素晴らしい仕事をしてきました。それは、HTTP Archiveプロジェクトが収集してきたデータに意味を持たせ、コミュニティがウェブのパフォーマンスを簡単に理解できるようにすることです。そこでWeb Almanacの出番です。

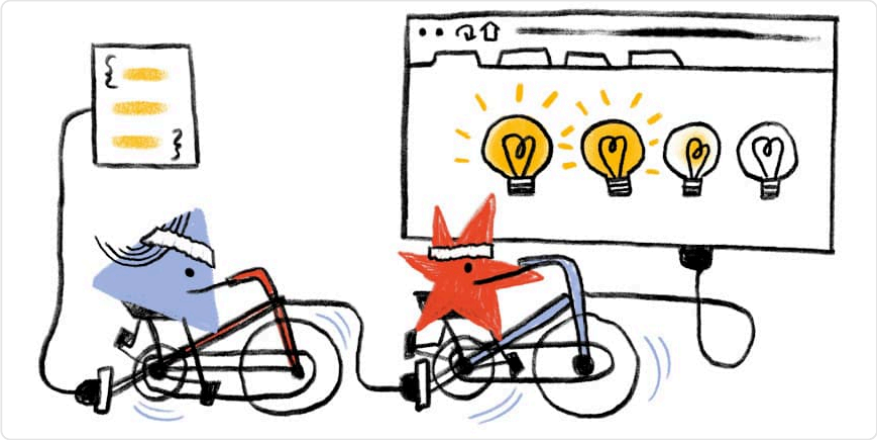
Web Almanacの使命は、データを掘り起こす勇気のある人しかアクセスできないような洞察の宝庫を、理解しやすい形でパッケージ化することにあります。これは、データの意味を理解し、それが何を意味するのかを教えてくれる業界の専門家の助けを借りて可能にしたものです。Web Almanacの20の章はそれぞれ、ウェブの特定の側面に焦点を当てており、各章はそれぞれの分野の専門家によって執筆され、レビューを受けています。Web Almanacの強みは、執筆者の専門知識から生まれます。

Web Almanacに掲載されている調査結果の多くは、賞賛に値するものですが、高品質のユーザー体験を提供するためには、まだまだ必要な作業があることを思い出させてくれる重要なものでもあります。各章のデータに基づいた分析は、より良いWebを開発するために私たちが共有している説明責任の一形態です。それは、間違ったやり方をしている人たちを非難することではなく、ベストプラクティスの道筋に光を当てることで、明確で正しいやり方があることを示しています。ウェブコミュニティの継続的な支援を得て、これを毎年の恒例行事にしたいと考えています。

このレポートには学ぶべきことがたくさんありますので、あなたが得たものをコミュニティで共有することで、ウェブの状態についての理解を深めることができます。

— *Rick Viscomi、Web Almanacの作成者。*

部 I 章 1
JavaScript



***Houssein Djirdeh** によって書かれた。*
David Fox、 Paul Calvano、 Mathias Bynens と Rick Viscomi によってレビュ。
Rick Viscomi による分析。
David Fox 編集。
Sakae Kotaro によって翻訳された。

序章

JavaScriptはウェブ上で、対話可能で複雑な体験を構築することを可能にするスクリプト言語です。これには、ユーザーの会話への応答、ページ上の動的コンテンツの更新などが含まれます。イベントが発生したときにウェブページがどのように振る舞うべきかに関係するものはすべて、JavaScriptが使用されています。

言語の仕様自体は、世界中の開発者が利用している多くのコミュニティビルドのライブラリやフレームワークとともに、1995年に言語が作成されて以来、変化と進化を続けてきました。JavaScriptの実装やインタプリタも進歩を続けており、ブラウザだけでなく多くの環境で利用できるようになっています。

HTTP Archive[1]は毎月数百万ページ[2]をクロールし、WebPageTest[3]のプライベートインスタン

1. https://httparchive.org/
2. https://httparchive.org/reports/state-of-the-web#numUrls
3. https://webpagetest.org/

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方法論 で学べます）。JavaScriptのコンテキストでは、HTTP Archiveはウェブ全体の言語の使用法に関する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ます。

どのくらいのJavaScriptを使っているのか？

JavaScriptは、私たちがブラウザに送るリソースの中で最もコストのかかるものでダウンロード、解析、コンパイル、そして最終的に実行されなければなりません。ブラウザはスクリプトの解析とコンパイルにかかる時間を大幅に短縮しましたが、WebページでJavaScriptが処理される際には、ダウンロードと実行が最もコストのかかる段階になっています[4]。

ブラウザに小さなJavaScriptのバンドルを送ることは、ダウンロード時間を短縮し、ひいてはページパフォーマンスを向上させるための最良の方法です。しかし、実際にどのくらいのJavaScriptを使っているのでしょうか？

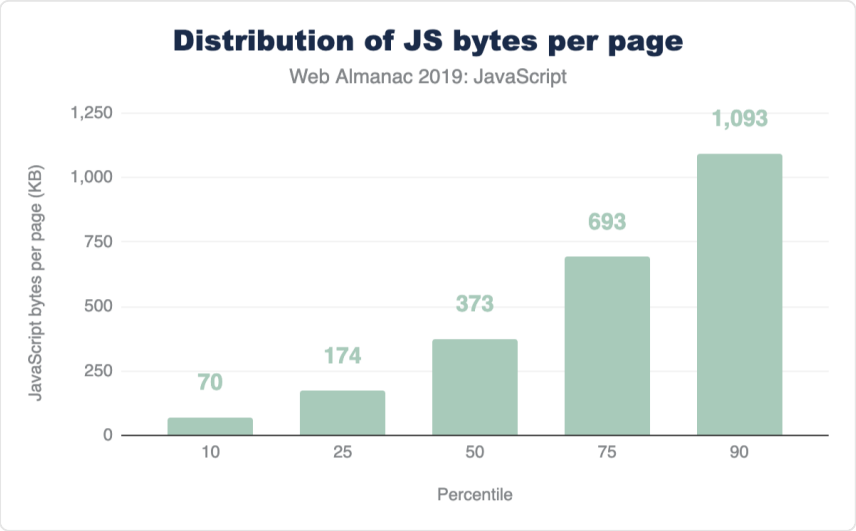


図1.1. ページあたりのJavaScriptバイト数の分布

上の図1.1を見ると、JavaScriptを373KB使用しているのは、50パーセンタイル（中央値）であることがわかります。つまり、全サイトの50%がこれだけのJavaScriptをユーザーに提供していることになります。

4. *https://v8.dev/blog/cost-of-javascript-2019*

この数字を見ると、これはJavaScriptの使いすぎではないかと思うのは当然のことです。しかし、ページのパフォーマンスに関しては、その影響はネットワーク接続や使用するデバイスに完全に依存します。モバイルクライアントとデスクトップクライアントを比較した場合、どのくらいのJavaScriptを提供しているのでしょうか？

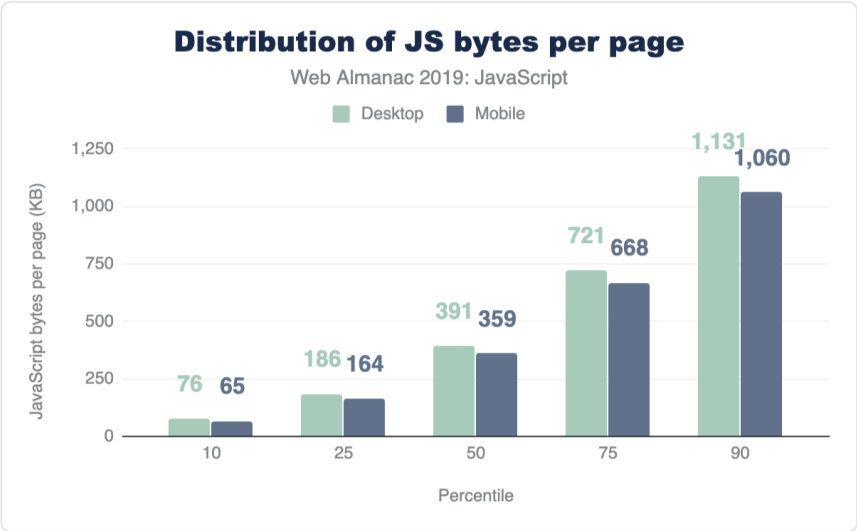


図1.2. デバイス別のページあたりのJavaScriptの分布。

どのパーセンタイルでも、モバイルよりもデスクトップデバイスに送信するJavaScriptの数がわずかに多くなっています。

処理時間

解析とコンパイルが行われた後、ブラウザによって取得されたJavaScriptは、利用する前に処理（または実行）される必要があります。デバイスは様々であり、その計算能力はページ上でのJavaScriptの処理速度に大きく影響します。ウェブ上での現在の処理時間は？

V8のメインスレッドの処理時間を異なるパーセンタイルで分析すると、アイデアを得ることができます。

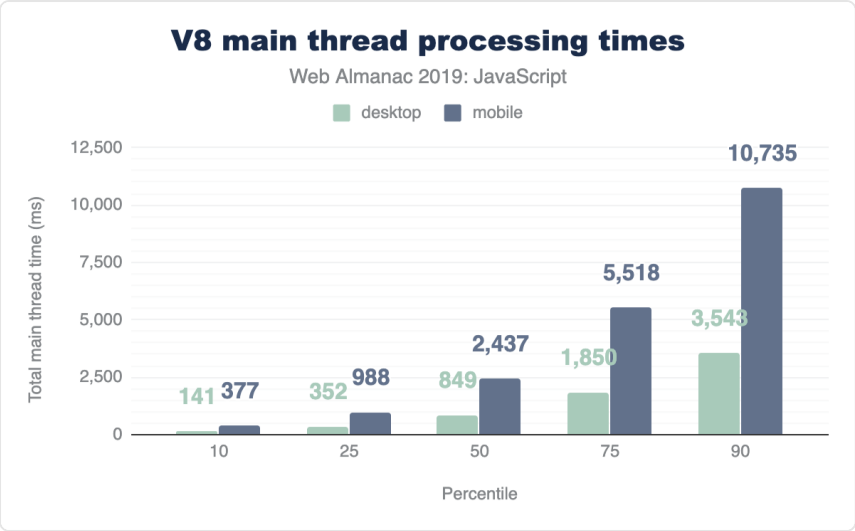


図1.3. デバイス別のV8メインスレッド処理時間。

すべてのパーセンタイルにおいて、処理時間はデスクトップよりもモバイルの方が長くなっています。メインスレッドの合計時間の中央値はデスクトップでは849msであるのに対し、モバイルでは2,437msと大きくなっています。

このデータはモバイルデバイスがJavaScriptを処理するのにかかる時間が、より強力なデスクトップマシンに比べてどれだけ長いかを示していますが、モバイルデバイスは計算能力の点でも違いがあります。次の表は、1つのWebページの処理時間がモバイルデバイスのクラスによって大きく異なることを示しています。

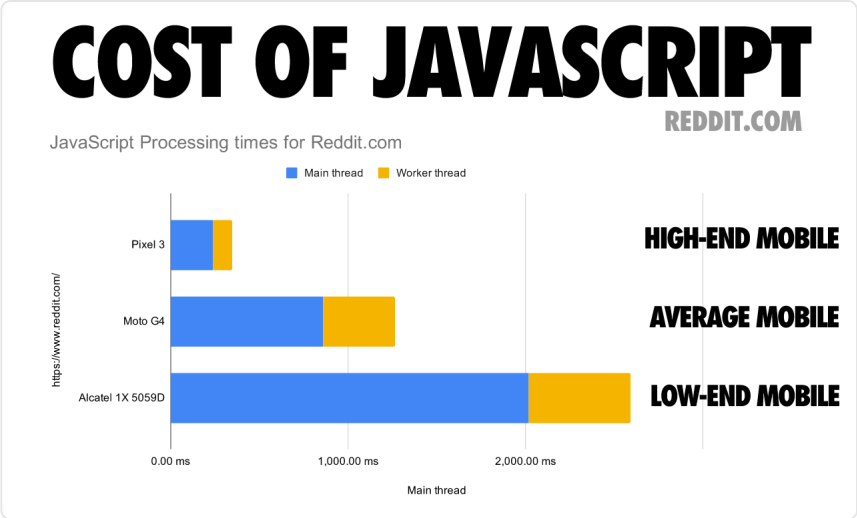


図1.4. reddit.comのJavaScript処理時間。2019年のJavaScriptのコスト[5]より。

リクエスト数

Webページで使用されているJavaScriptの量を分析しようとする場合、1つの方法として、送信されたリクエスト数を調べる価値があります。HTTP/2では、複数の小さなチャンクを送信することで、より大きなモノリシックなバンドルを送信するよりもページの負荷を改善できます。また、デバイスクライアント別に分解してみると、どのくらいのリクエストがフェッチされているのでしょうか。

5. *https://v8.dev/blog/cost-of-javascript-2019*

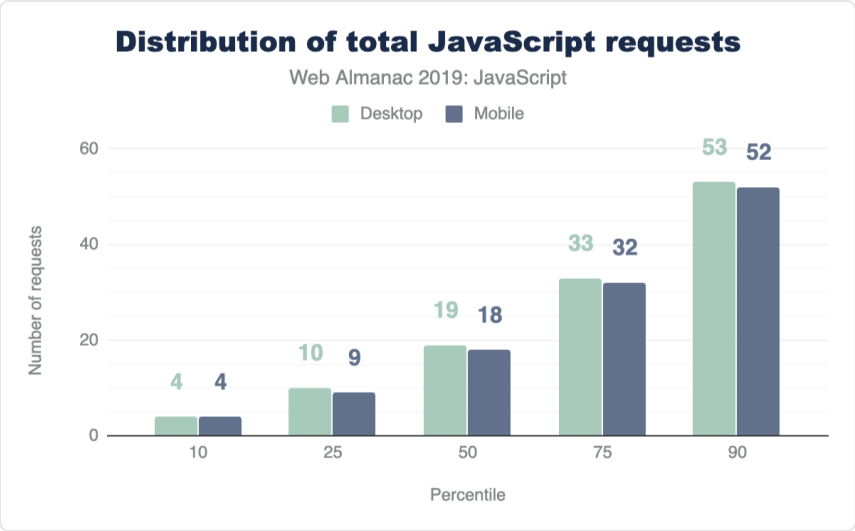


図1.5. 総JavaScriptリクエスト数の分布。

中央値では、デスクトップ用に19件、モバイル用に18件のリクエストが送信されています。

ファーストパーティ対サードパーティ

これまでに分析した結果のうち、全体のサイズとリクエスト数が考慮されていました。しかし、大多数のウェブサイトでは、取得して使用しているJavaScriptコードのかなりの部分がサードパーティのソースから来ています。

サードパーティのJavaScriptは、外部のサードパーティのソースから取得できます。広告、分析、ソーシャルメディアの埋め込みなどは、サードパーティのスクリプトを取得するための一般的なユースケースです。そこで当然のことながら、次の質問に移ります。

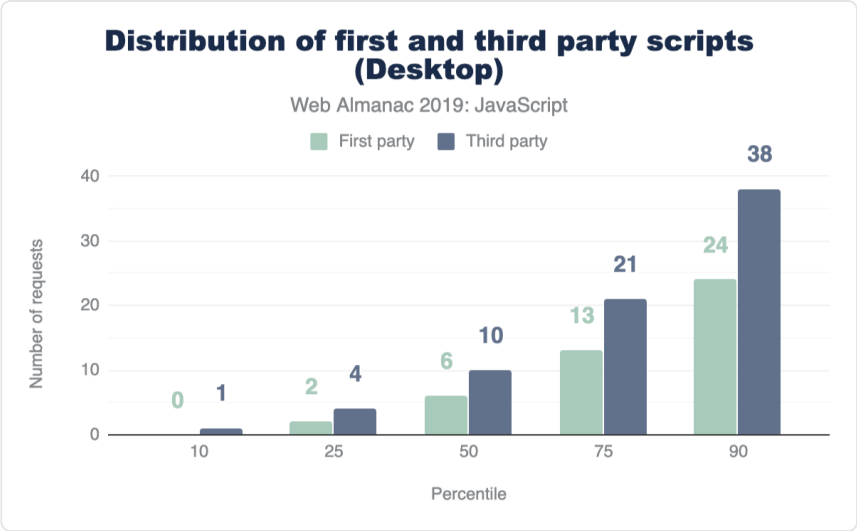


図1.6. デスクトップ上のファーストスクリプトとサードパーティスクリプトの分布。

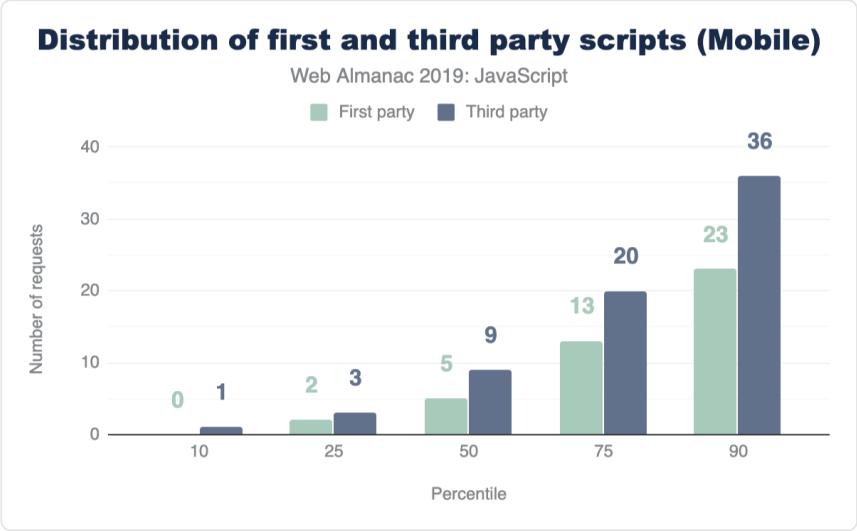


図1.7. モバイル上のファーストパーティとサードパーティのスクリプトの分布。

モバイルクライアントとデスクトップクライアントの両方において、すべてのパーセンタイルにおいて、ファーストパーティよりもサードパーティのリクエストの方が多く送信されています。これが意外に思える場合は、実際に提供されるコードのうち、サードパーティのベンダーからのものがどれくらいあるのかを調べてみましょう。

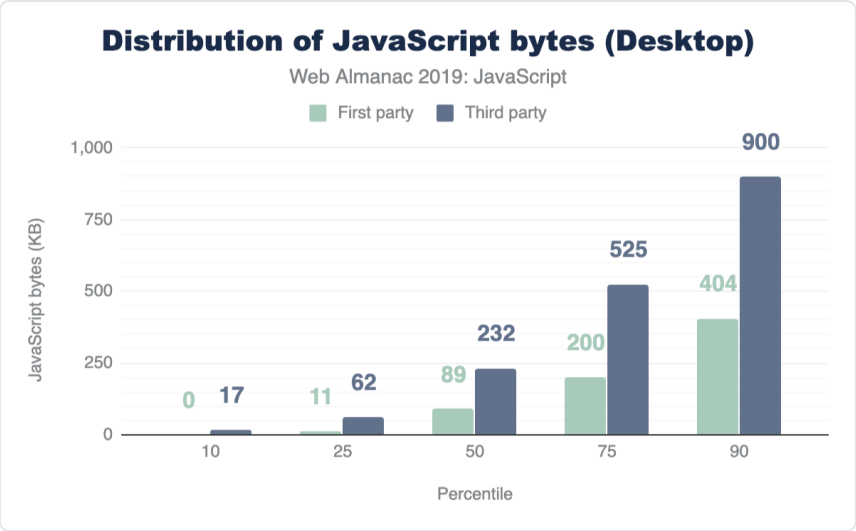


図1.8. デスクトップ上でダウンロードされたJavaScriptの総ダウンロード数の分布。

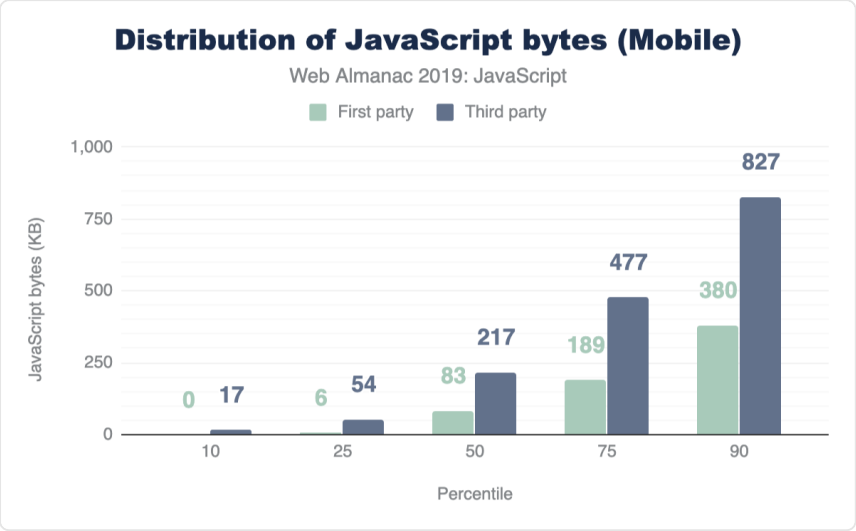


図1.9. モバイルでダウンロードされたJavaScriptの総ダウンロード数の分布。

中央値では、モバイルとデスクトップの両方で、開発者が作成したファーストパーティのコードよりもサードパーティのコードの方が89%多く使用されています。これは、サードパーティのコードが肥大化の最大の要因の1つであることを明確に示しています。サードパーティの影響についての詳細は、“サードパーティ”の章を参照してください。

リソース圧縮

ブラウザとサーバの会話のコンテキストで、リソース圧縮とは、データ圧縮アルゴリズムを使用して変更されたコードを指します。リソースは事前に静的に圧縮することも、ブラウザからの要求に応じて急ぎ圧縮することもでき、どちらの方法でも転送されるリソースサイズが大幅に削減されページパフォーマンスが向上します。

テキスト圧縮アルゴリズムは複数ありますが、HTTPネットワークリクエストの圧縮（および解凍）に使われることが多いのはこの2つだけです。

- Gzip[6] (`gzip`): サーバーとクライアントの相互作用のために最も広く使われている圧縮フォーマット。
- Brotli[7] (`br`): 圧縮率のさらなる向上を目指した新しい圧縮アルゴリズム。90%のブラウザ[8]がBrotliエンコーディングをサポートしています。

圧縮されたスクリプトは、一度転送されるとブラウザによって常に解凍される必要があります。これは、コンテンツの内容が変わらないことを意味し、実行時間が最適化されないことを意味します。しかし、リソース圧縮は常にダウンロード時間を改善しますが、これはJavaScriptの処理で最もコストのかかる段階の1つでもあります。JavaScriptファイルが正しく圧縮されていることを確認することは、サイトのパフォーマンスを向上させるための最も重要な要因の1つとなります。

JavaScriptのリソースを圧縮しているサイトはどれくらいあるのでしょうか？

6. *https://www.gzip.org/*
7. *https://github.com/google/brotli*
8. *https://caniuse.com/#feat=brotli*

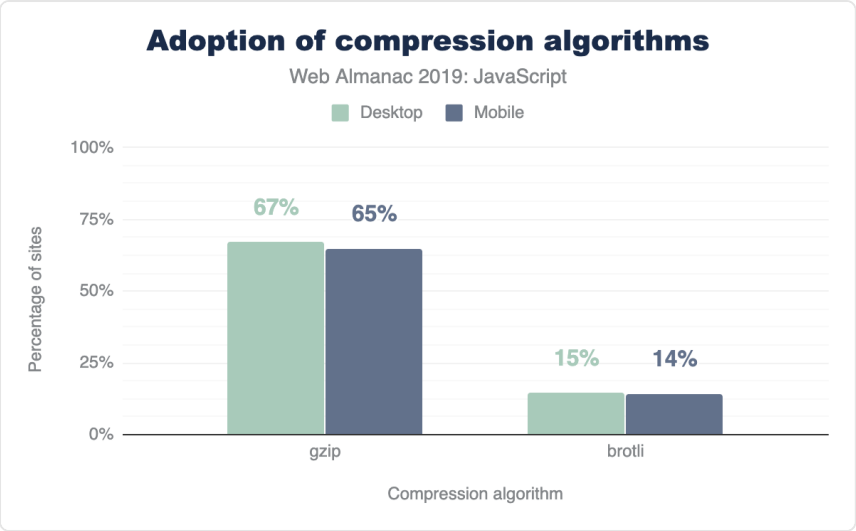


図1.10. JavaScript リソースをGzipまたはBrotliで圧縮しているサイトの割合。

大多数のサイトではJavaScriptのリソースを圧縮しています。Gzipエンコーディングはサイトの～64-67%で、Brotliは～14%で使用されています。圧縮率はデスクトップとモバイルの両方でほぼ同じです。

圧縮に関するより深い分析については、“圧縮”の章を参照してください。

オープンソースのライブラリとフレームワーク

オープンソースコード、または誰でもアクセス、閲覧、修正が可能な寛容なライセンスを持つコード。小さなライブラリから、Chromium[9]やFirefox[10]のようなブラウザ全体に至るまで、オープンソースコードはウェブ開発の世界で重要な役割を果たしています。JavaScriptの文脈では、開発者はオープンソースのツールに依存して、あらゆるタイプの機能をWebページに組み込んでいます。開発者が小さなユーティリティライブラリを使用するか、アプリケーション全体のアーキテクチャを決定する大規模なフレームワークを使用するかにかかわらずオープンソースのパッケージに依存することで、機能開発をより簡単かつ迅速にできます。では、どのJavaScriptオープンソースライブラリが最もよく使われているのでしょうか？

9. https://www.chromium.org/Home
10. https://www.openhub.net/p/firefox

ライブラリ	デスクトップ	モバイル
jQuery	85.03%	83.46%
jQuery Migrate	31.26%	31.68%
jQuery UI	23.60%	21.75%
Modernizr	17.80%	16.76%
FancyBox	7.04%	6.61%
Lightbox	6.02%	5.93%
Slick	5.53%	5.24%
Moment.js	4.92%	4.29%
Underscore.js	4.20%	3.82%
prettyPhoto	2.89%	3.09%
Select2	2.78%	2.48%
Lodash	2.65%	2.68%
Hammer.js	2.28%	2.70%
YUI	1.84%	1.50%
Lazy.js	1.26%	1.56%
Fingerprintjs	1.21%	1.32%
script.aculo.us	0.98%	0.85%
Polyfill	0.97%	1.00%
Flickity	0.83%	0.92%
Zepto	0.78%	1.17%
Dojo	0.70%	0.62%

図1.11. デスクトップとモバイルでのトップJavaScript ライブラリ

これまでに作成された中で最も人気のあるJavaScriptライブラリであるjQuery[11]は、デスクトップページの85.03%、モバイルページの83.46%で使用されています。Fetch[12]や

11. https://jquery.com/
12. https://developer.mozilla.org/docs/Web/API/Fetch_API

querySelector[13]など、多くのブラウザAPIやメソッドの出現により、ライブラリが提供する機能の多くがネイティブ形式に標準化されました。jQueryの人気は衰退しているように見えるかもしれませんが、なぜ今でもウェブの大部分で使われているのでしょうか？

理由はいくつか考えられます。

- WordPress[14]は、30%以上のサイトで使用されているため、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を要する場合があり、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。

他にもjQueryの亜種（jQuery migrate、jQuery UI）、Modernizr[15]、Moment.js[16]、Underscore.js[17]などがトップで使用されているJavaScriptライブラリです。

フレームワークと**UI**ライブラリ

方法論で述べたように、HTTP Archive(Wappalyzer)で使用されているサードパーティ製の検出ライブラリには、特定のツールを検出する方法に関して多くの制限があります。JavaScriptライブラリやフレームワークの検出を改善するための未解決の問題があります[18]、それがここで紹介した結果に影響を与えています。

過去数年の間に、JavaScriptのエコシステムでは、**シングルページアプリケーション** (SPA) の構築を容易にするオープンソースのライブラリやフレームワークが増えてきました。シングルページアプリケーションとは、単一のHTMLページを読み込み、サーバーから新しいページを取得する代わりにJavaScriptを使用してユーザーの対話に応じてページを修正するWebページのことを指します。これはシングルページアプリケーションの大前提であることに変わりはありませんが、このようなサイトの体験を向上させるために、異なるサーバーレンダリングアプローチを使用できます。これらのタイプのフレームワークを使用しているサイトはどれくらいあるのでしょうか？

13. *https://developer.mozilla.org/docs/Web/API/Document/querySelector*
14. *https://wordpress.org/*
15. *https://modernizr.com/*
16. *https://momentjs.com/*
17. *https://underscorejs.org/*
18. *https://github.com/AliasIO/wappalyzer/issues/2450*

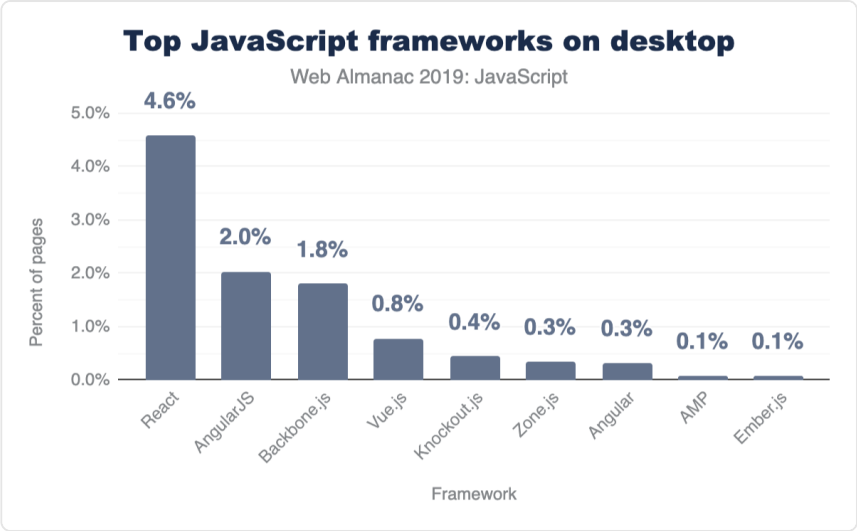


図1.12. デスクトップで最もよく使われるフレームワーク

ここでは人気のあるフレームワークのサブセットのみを分析していますが、これらのフレームワークはすべて、これら2つのアプローチのいずれかに従っていることに注意することが重要です。

- モデルビューコントローラ[19]（またはモデルビュービューモデル）アーキテクチャー
- コンポーネントベースアーキテクチャ

コンポーネントベースモデルへの移行が進んでいるとはいえ、MVCパラダイムを踏襲した古いフレームワーク（AngularJS[20]、Backbone.js[21]、Ember[22]）は、いまだに何千ページにもわたって使われています。しかし、React[23]、Vue[24]、Angular[25]はコンポーネントベースのフレームワークが主流です（Zone.js[26]は現在Angular coreの一部となっているパッケージです）。

19. *https://developer.chrome.com/apps/app_frameworks*
20. *https://angularjs.org/*
21. *https://backbonejs.org/*
22. *https://emberjs.com/*
23. *https://reactjs.org/*
24. *https://vuejs.org/*
25. *https://angular.io/*
26. *https://github.com/angular/zone.js*

差分ロード

JavaScriptモジュール[27]、またはESモジュールは、すべての主要ブラウザ[28]でサポートされています。モジュールは、他のモジュールからインポートおよびエクスポートできるスクリプトを作成する機能を提供します。これにより、サードパーティのモジュールローダーに頼ることなく、必要に応じてインポートとエクスポートを行い、モジュールパターンで構築されたアプリケーションを誰でも構築できます。

スクリプトをモジュールとして宣言するには、スクリプトタグが`type="module"`属性を取得しなければなりません。

```
<script type="module" src="main.mjs"></script>
```

ページ上のスクリプトに`type="module'`を使用しているサイトはどれくらいあるでしょうか？

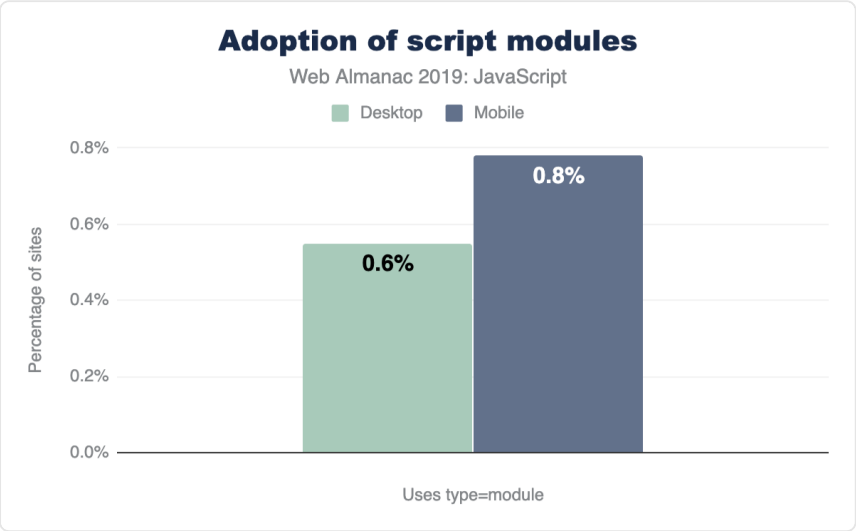


図1.13. type=moduleを利用しているサイトの割合。

ブラウザレベルでのモジュールのサポートはまだ比較的新しく、ここでの数字は、現在スクリプトに`type="module"`を使用しているサイトが非常に少ないことを示しています。多くのサイトでは、コードベース内でモジュールを定義するためにモジュールローダー（全デス

27. https://v8.dev/features/modules
28. https://caniuse.com/#feat=es6-module

クトップサイトの2.37%がRequireJS[29]を使用しています）やバンドラー（webpack[30]を使用しています）にまだ依存しています。

ネイティブモジュールを使用する場合は、モジュールをサポートしていないブラウザに対して適切なフォールバックスクリプトを使用することが重要です。これは、`nomodule`属性を持つ追加スクリプトを含めることで実現できます。

```
<script nomodule src="fallback.js"></script>
```

併用すると、モジュールをサポートしているブラウザは`nomodule`属性を含むスクリプトを完全に無視します。一方、モジュールをサポートしていないブラウザは¥`type="module"`属性を持つスクリプトをダウンロードしません。ブラウザは`nomodule`も認識しないので、`type="module"`属性を持つスクリプトを普通にダウンロードします。このアプローチを使うことで、開発者は最新のコードを最新のブラウザに送信してページ読み込みを高速化する[31]できます。では、ページ上のスクリプトに`nomodule`を使っているサイトはどれくらいあるのだろうか。

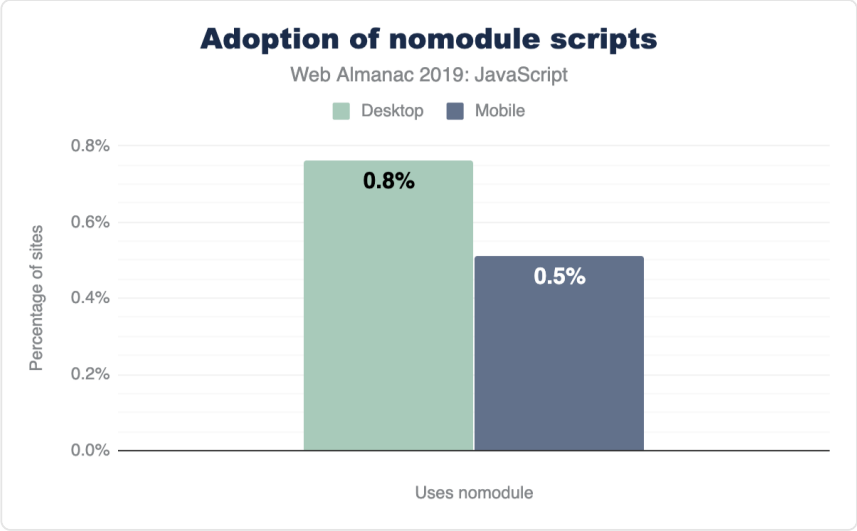


図1.14. nomoduleを使用しているサイトの割合。

同様に、スクリプトに`nomodule`属性を使用しているサイトはほとんどありません（0.50%-0.80%）。

29. https://github.com/requirejs/requirejs
30. https://webpack.js.org/
31. https://web.dev/serve-modern-code-to-modern-browsers/

プリロードとプリフェッチ

プリロード[32] と プリフェッチ[33]はリソースヒントであり、どのリソースをダウンロードする必要があるかを判断する際にブラウザを助けることができます。

- `<link rel="preload">`でリソースをプリロードすると、ブラウザはこのリソースをできるだけ早くダウンロードするように指示します。これは、ページの読み込みプロセスの後半に発見され、最後にダウンロードされてしまう重要なリソース（例えば、HTMLの下部にあるJavaScriptなど）に特に役立ちます。
- `<link rel="prefetch">`を使用することで、ブラウザが将来のナビゲーションに必要なリソースを取得するためのアイドル時間を利用できるようにします。

では、プリロードやプリフェッチディレクティブを使っているサイトはどれくらいあるのでしょうか？

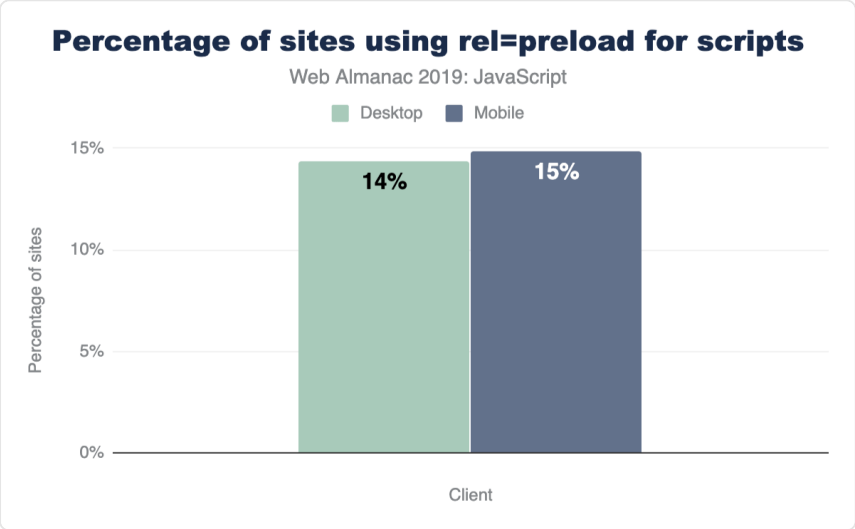


図1.15. スクリプトにrel=preloadを使用しているサイトの割合。

HTTP Archiveで測定したすべてのサイトで、デスクトップサイトの14.33%、モバイルサイトの14.84%が`<link rel="preload">`をページ上のスクリプトに使用しています。

プリフェッチについて以下のようなものがあります。

32. https://developer.mozilla.org/docs/Web/HTML/Link_types/preload
33. https://developer.mozilla.org/docs/Web/HTTP/Link_prefetching_FAQ

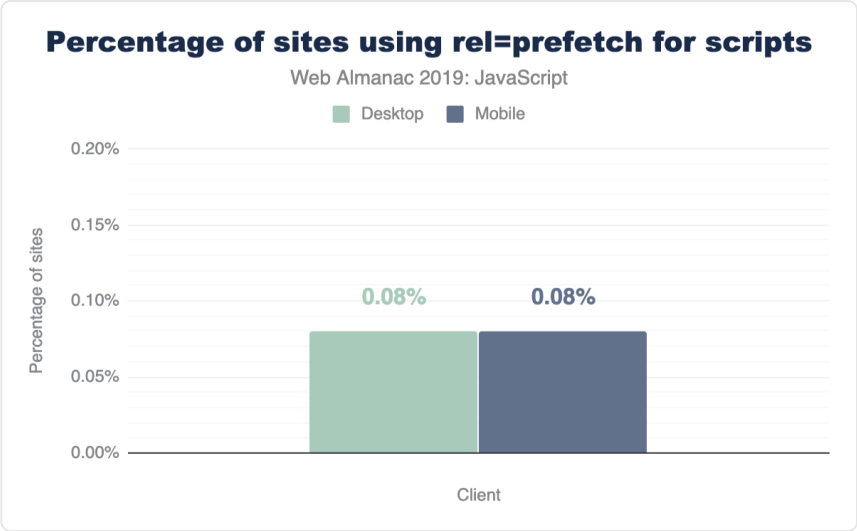


図1.16. スクリプトにrel=prefetchを使用しているサイトの割合。

モバイルとデスクトップの両方で、0.08%のページがスクリプトのいずれかでプリフェッチを利用しています。

新しいAPI

JavaScriptは言語として進化を続けています。ECMAScriptと呼ばれる言語標準そのものの新バージョンが毎年リリースされ、新しいAPIや機能が提案段階を通過して言語そのものの一部となっています。

HTTP Archiveを使用すると、サポートされている（あるいはこれからサポートされる）新しいAPIを調べて、その使用法がどの程度普及しているかを知ることができます。これらのAPIは、サポートしているブラウザで既に使用されているかもしれませんし、すべてのユーザに対応しているかどうかを確認するためにポリフィルを添付しています。

以下のAPIを使用しているサイトはどれくらいありますか？

- Atomics[34]
- Intl[35]

34. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Atomics
35. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Intl

- Proxy[36]
- SharedArrayBuffer[37]
- WeakMap[38]
- WeakSet[39]

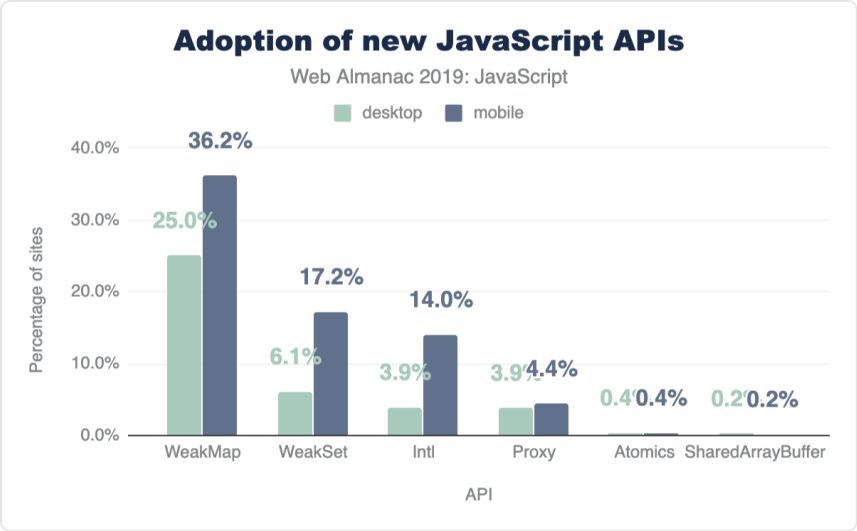


図1.17. 新しいJavaScript APIの利用法

Atomics(0.38%)とSharedArrayBuffer(0.20%)は、使用されているページが少ないので、このチャートではほとんど見えません。

ここでの数値は概算であり、機能の使用状況を測定するためのUseCounter[40]を活用していないことに注意してください。

ソースマップ

多くのビルドシステムでは、JavaScriptファイルはサイズを最小化し、多くのブラウザではまだサポートされていない新しい言語機能のためにトランスパイルされるようにミニ化されています。さらに、TypeScript[41]のような言語スーパーセットは、元のソースコードとは明ら

36. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Proxy
37. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer
38. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/WeakMap
39. https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/WeakSet
40. https://chromium.googlesource.com/chromium/src.git/+/master/docs/use_counter_wiki.md
41. https://www.typescriptlang.org/

かに異なる出力へコンパイルされます。これらの理由から、ブラウザに提供される最終的なコードは読めず、解読が困難なものになることがあります。

ソースマップとは、JavaScriptファイルに付随する追加ファイルで、ブラウザが最終的な出力を元のソースにマップできます。これにより、プロダクションバンドルのデバッグや分析をより簡単にできます。

便利ではありますが多くのサイトが最終的な制作サイトにソースマップを入れたくない理由は、完全なソースコードを公開しないことを選択するなど、いくつかあります。では、実際にどれくらいのサイトがソースマップを含んでいるのでしょうか？

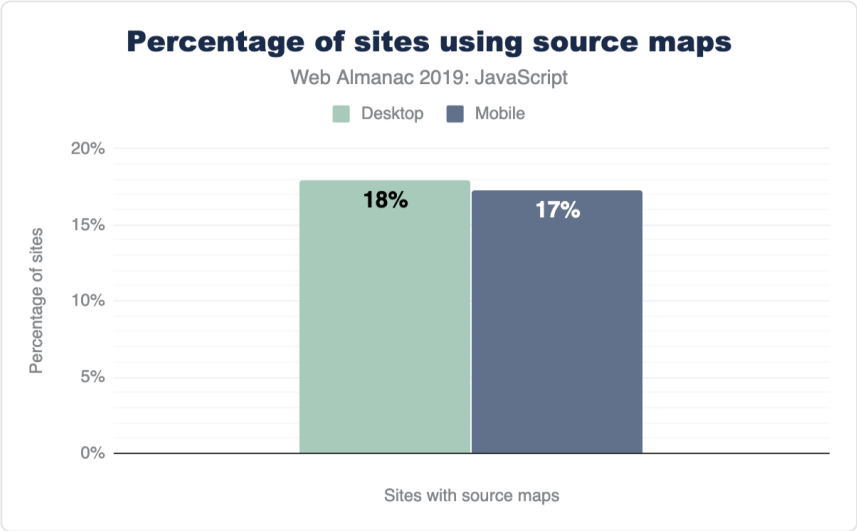


図1.18. ソースマップを使用しているサイトの割合。

デスクトップページでもモバイルページでも、結果はほぼ同じです。17〜18%は、ページ上に少なくとも1つのスクリプトのソースマップを含んでいます（`sourceMappingURL`を持つファーストパーティスクリプトとして検出されます）。

結論

JavaScriptのエコシステムは毎年変化し続け、進化し続けています。新しいAPI、改良されたブラウザエンジン、新しいライブラリやフレームワークなど、私たちが期待していることは尽きることがありません。HTTP Archiveは、実際のサイトがどのようにJavaScriptを使用しているかについての貴重な洞察を提供してくれます。

JavaScriptがなければ、ウェブは現在の場所にはなく、この記事のために集められたすべてのデータがそれを証明しているに過ぎません。

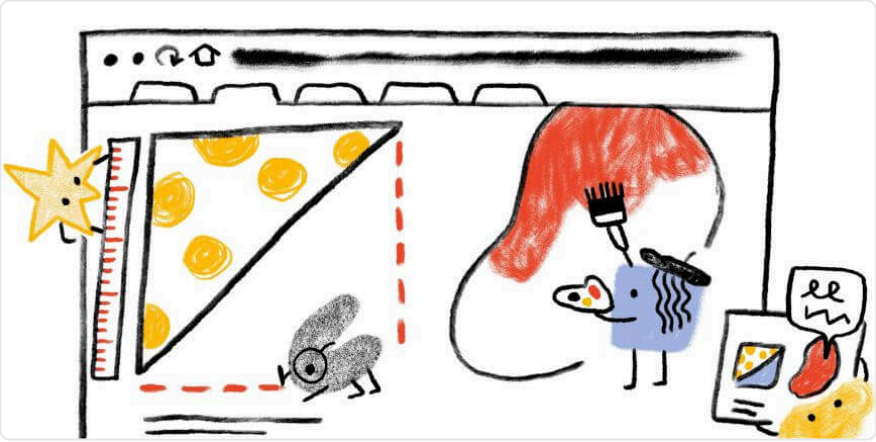
著者

Houssein Djirdeh

@hdjirdeh housseindjirdeh https://houssein.me

HousseinはGoogleの開発提唱者で、スピード、パフォーマンス、ウェブフレームワークのエコシステムに取り組んでいます。彼は @hdjirdeh でツイートし、https://houssein.me/ でブログを書いています。

部 I 章 2
CSS



Una Kravets と Adam Argyle によって書かれた。
Eric A. Meyer と Chen Hui Jing によってレビュ。
Rick Viscomi による分析。
Rachel Costello 編集。
Sakae Kotaro によって翻訳された。

導入

カスケードスタイルシート（CSS）は、Webページの描画、書式設定、およびレイアウトに使用されます。それらの機能は、テキストの色から3Dパースペクティブまでの単純な概念に及びます。また、さまざまな画面サイズ、コンテキストの表示、印刷を処理する開発者を支援するフックもあります。 CSSは、開発者がコンテンツを絞り込み、ユーザーに適切に適合させることを支援します。

CSSをWebテクノロジーに慣れていない人に説明するときは、CSSを家の壁にペイントする言語と考える事が役立ちます。窓やドアのサイズと位置、および壁紙や植物などが栄える装飾と説明できる。そのストーリーの面白いひねりは、ユーザーが家の中を歩いているかどうかに応じて、開発者はその特定のユーザーの好みやコンテキストに家を作り替えることができるということです！

この章では、WebでのCSSの使用方法に関するデータを検査、集計、および抽出します。私

たちの目標はどの機能が使用されているか、どのように使用されているか、CSSがどのように成長し採用されているかを全体的に理解することです。

魅力的なデータを掘り下げる準備はできましたか?! 以下の数値の多くは小さい場合がありますが、重要ではないと誤解しないでください! 新しいものがウェブを飽和させるには何年もかかることがあります。

色

色は、Webのテーマとスタイリングに不可欠な部分です。ウェブサイトが色を使用する傾向を見てみましょう。

色の種類

16進数は、色を説明する最も一般的な方法であり93%の使用率、RGB、HSLが続きます。興味深いことに、開発者はこれらの色の種類に関してアルファ透明度の引数を最大限に活用しています。HSLAとRGBAは、HSLとRGBよりもはるかに人気があり、使用量はほぼ2倍です。アルファ透明度は後でWeb仕様に追加されましたが、HSLAとRGBAはIE9までさかのぼって[42]サポートされているため、先に進んで使用することもできます！

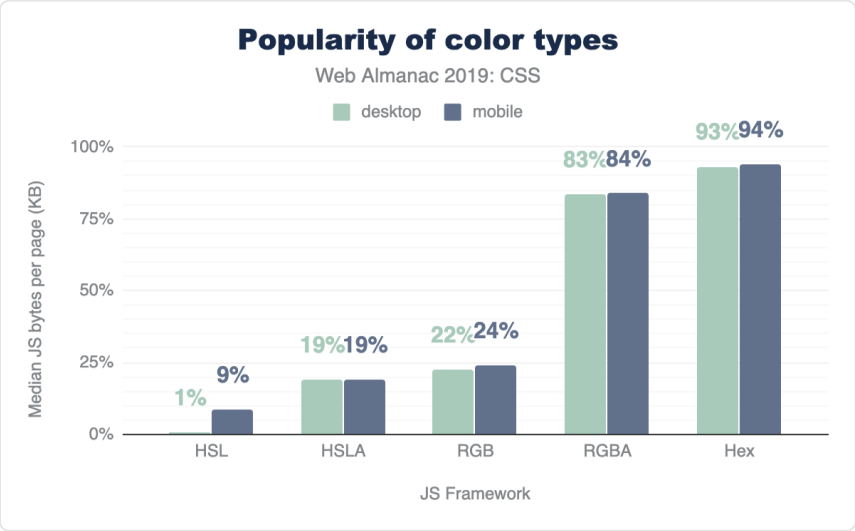


図2.1. カラー形式の人気。

42. *https://caniuse.com/#feat=css3-colors*

色の選択

CSSの名前付きカラーは148個[43]あり、`transparent`および`currentcolor`の特別な値は含まれていません。これらを文字列名で使用して、読みやすくできます。最も人気がある名前の付いた色は`黒`と`白`であり、当然のことながら`赤`と`青`が続きます。

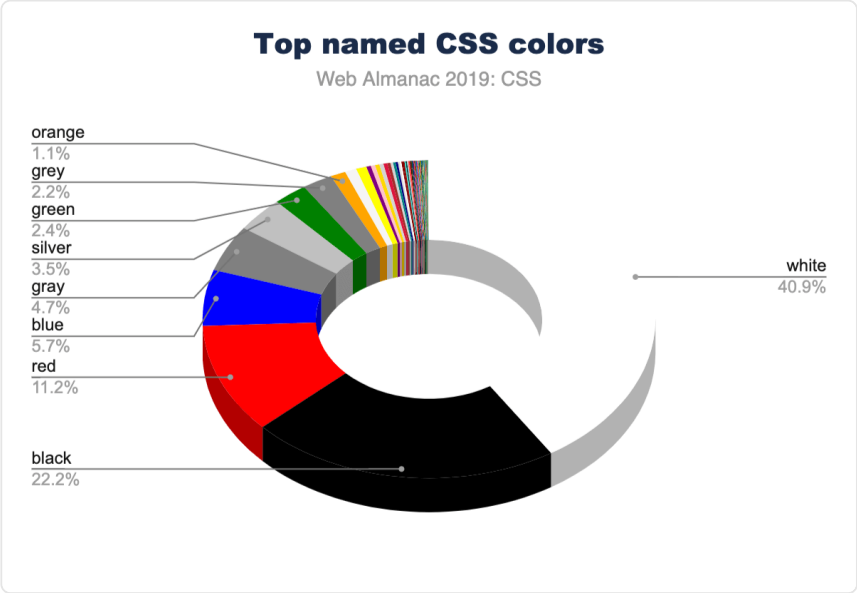


図2.2. 上位の名前付き色。

言語は色によっても興味深いことに推測されます。イギリス式の「grey」よりもアメリカ式の「gray」の例が多くあります。グレー色[44]（`グレー`、`ライトグレー`、`ダークグレー`、`スレートグレー`など）のほぼすべてのインスタンスは、「e」ではなく「a」で綴ると、使用率がほぼ2倍になりました。 gr [a/e] ysが組み合わされた場合、それらは青よりも上位にランクされ、＃4スポットで固まります。これが、チャートで銀がグレーよりも高いランクになっている理由です。

色の数

ウェブ全体でいくつの異なるフォントの色が使用されていますか？　これは一意の色の総数ではありません。むしろ、テキストに使用される色の数です。このグラフの数値は非常に高く、経験からCSS変数なしでは間隔、サイズ、色がすぐに離れて、スタイル全体で多くの小

43. https://www.w3.org/TR/css-color-4/#named-colors
44. https://www.rapidtables.com/web/color/gray-color.html

さな値に断片化することがわかります。これらの数値はスタイル管理の難しさを反映しており、あなたがチームやプロジェクトに持ち帰るための何らかの視点を作り出すのに役立つことを願っています。この数を管理可能かつ合理的な量に減らすにはどうすればよいですか？

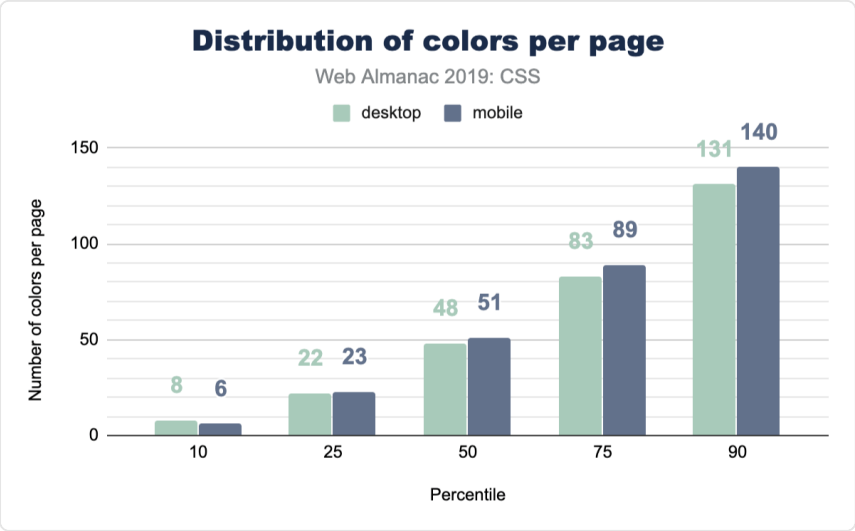


図2.3. ページごとの色の分布。

色の複製

さて、私たちはここで興味を持ち、ページにいくつの重複色が存在するかを調べたいと思いました。しっかり管理された再利用可能なクラスCSSシステムがなければ、複製はものすごく簡単に作成できます。中央値には十分な重複があるため、パスを実行してそれらをカスタムプロパティと統合する価値があるかもしれません。

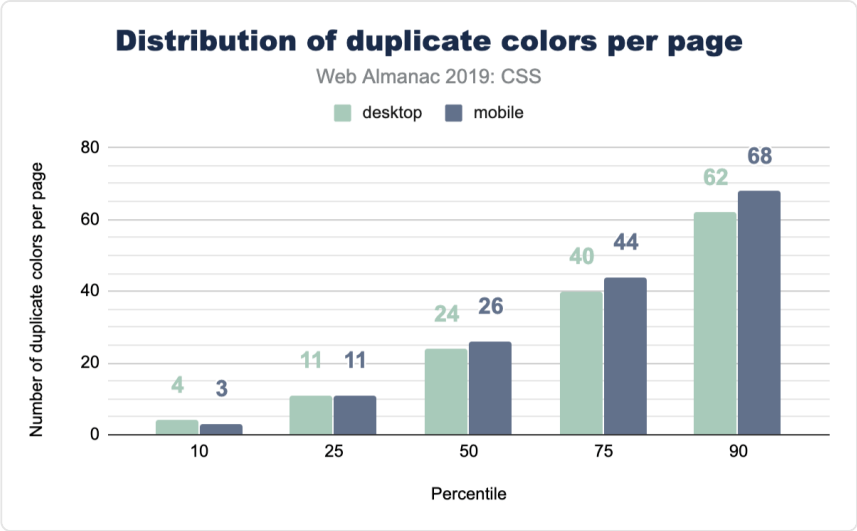


図2.4. ページごとの複製色の分布。

ユニット

CSSには、異なるユニットタイプ（`rem`、`px`、`em`、`ch`、または`cm`）を使用して同じ視覚的結果を達成するためのさまざまな方法があります！　それで、どのユニットタイプが最も人気ですか？

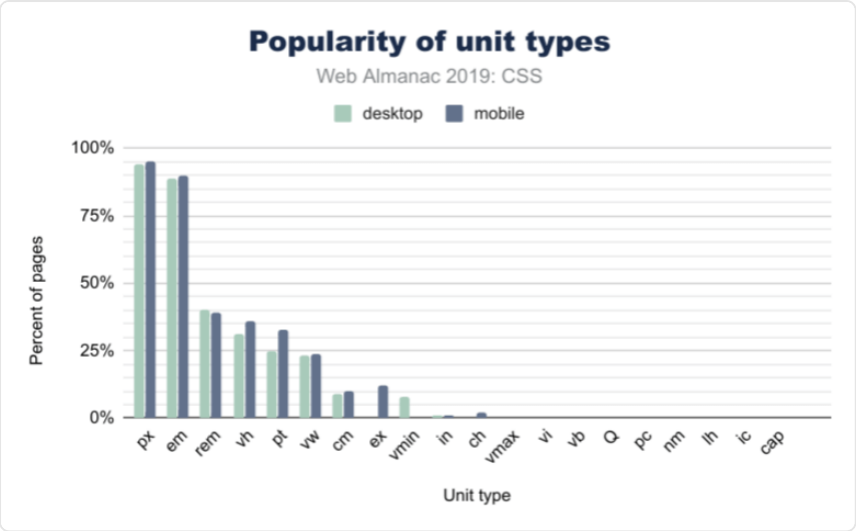


図2.5. ユニットタイプの人気。

長さとサイズ

当然のことながら、上の図2.5では、`px`が最もよく使用されるユニットタイプであり、Webページの約95%が何らかの形式のピクセルを使用しています（これは要素のサイズ、フォントサイズなどです）。ただし、`em`ユニットの使用率はほぼ同じで約90%です。これは、Webページで40%の頻度しかない`rem`ユニットよりも2倍以上人気があります。違いを知りたい場合は、`em`は親フォントサイズに基づいており、`rem`はページに設定されている基本フォントサイズに基づいています。`em`のようにコンポーネントごとに変更されることはないため、すべての間隔を均等に調整できます。

物理的な空間に基づいた単位となると、`cm`（またはセンチメートル）ユニットが最も人気であり、次に`in`（インチ）、`Q`が続きます。これらのタイプのユニットは、印刷スタイルシートに特に役立つことがわかっていますが、この調査まで`Q`ユニットが存在することさえ知りませんでした！　知ってましたか？

この章の以前のバージョンでは、`Q`ユニットの予想外の人気について説明しました。この章を取り巻くコミュニティの議論[45]のおかげで、これは分析のバグであることがわかり、それに応じて図2.5を更新しました。

45. https://discuss.httparchive.org/t/chapter-2-css/1757/6

ビューポートベースのユニット

ビューポートベースのユニットのモバイルとデスクトップの使用に関しては、ユニットタイプに大きな違いが見られました。モバイルサイトは36.8%が `vh`（ビューポートの高さ）を使用していますが、デスクトップサイトは31%しか使用していません。また、 `vh` は `vw`（ビューポートの幅）よりも約11%一般的です。 `vmin`（ビューポートの最小値）は `vmax`（ビューポートの最大値）よりも人気があり、モバイルでの `vmin` の使用率は約8%で、 `vmax` はWebサイトの1%のみが使用しています。

カスタムプロパティ

カスタムプロパティは、多くの場合CSS変数と呼ばれます。ただし、通常の静的変数よりも動的です！　CSS変数は非常に強力であり、コミュニティとして私たちはまだ彼らの可能性を発見しています。

5%

図2.6. カスタムプロパティを使用しているページの割合。

私たちのお気に入りはCSS追加の1つが健全な成長を示しており、これは刺激的な情報だと感じました。これらは2016年または2017年以降、すべての主要なブラウザで利用可能であったため、かなり新しいと言っても過言ではありません。多くの人々は、CSSプリプロセッサ変数からCSSカスタムプロパティに移行しています。カスタムプロパティが標準になるまであと数年かかると推定されます。

セレクター

ID vs クラスセレクター

CSSには、スタイリングのためにページ上の要素を見つける方法がいくつかあるのでIDとクラスを互いに比較して、どちらがより一般的であるかを確認しましょう。結果は驚くべきものでありません。クラスの方が人気です！

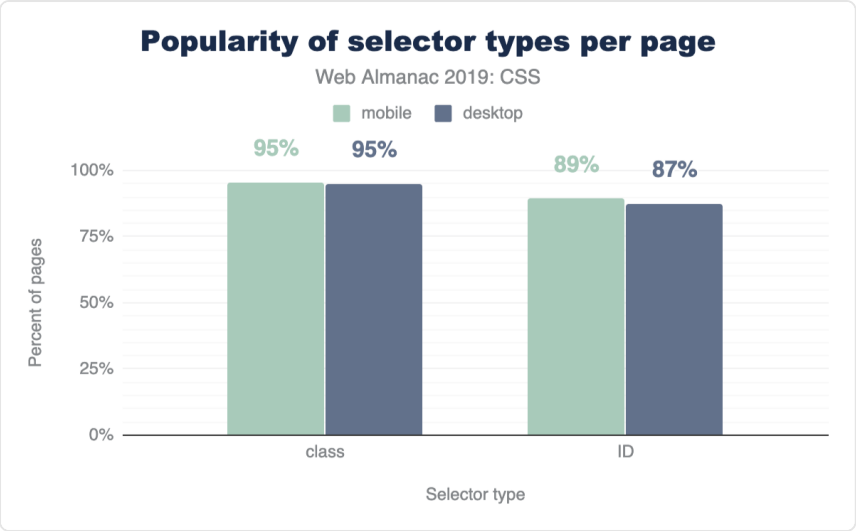


図2.7. ページごとのセレクタータイプの人気。

素敵なフォローアップチャートはこれです。スタイルシートで見つかったセレクタの93%がクラスを占めることを示しています。

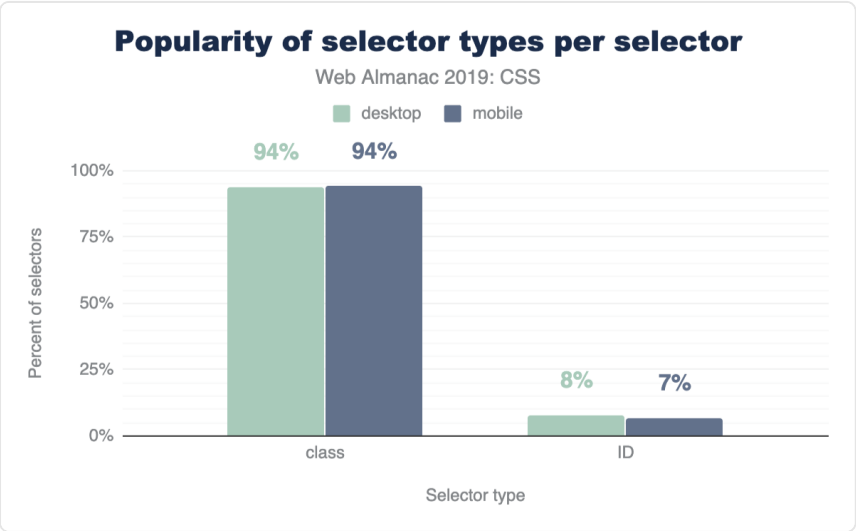


図2.8. セレクタごとのセレクタタイプの人気。

属性セレクター

CSSには、非常に強力な比較セレクターがいくつかあります。これらは、`[target="_blank"]`、`[attribute^="value"]`、`[title~="rad"]`、`[attribute$="-rad"]`または`[attribute*="value"]`などのセレクターです。それらを使用しますか？　よく使われていると思いますか？　それらがWeb全体でIDとクラスでどのように使用されるかを比較しましょう。

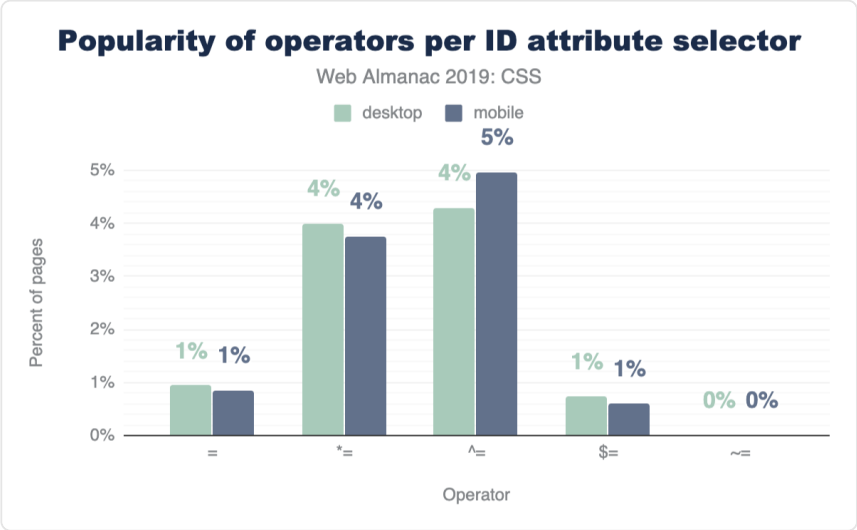


図2.9. ID属性セレクターごとの演算子の人気。

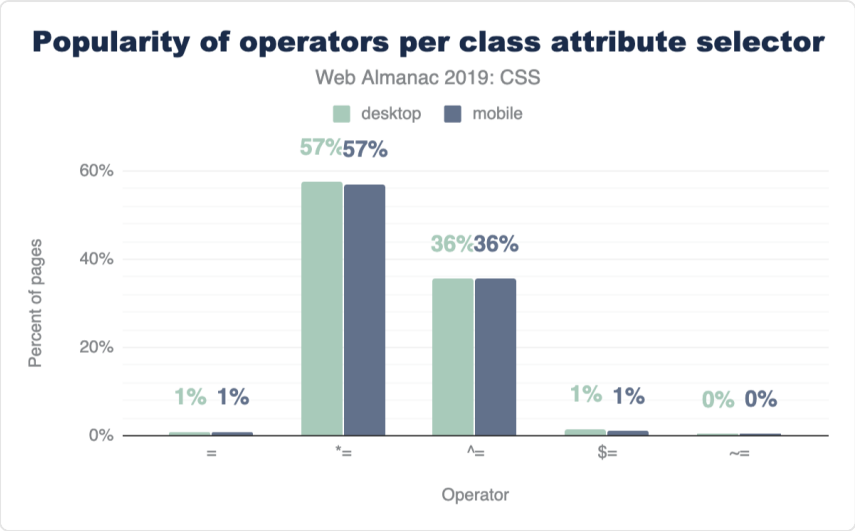


図2.10. クラス属性セレクタごとの演算子の人気。

スタイルシートのIDセレクターは通常クラスセレクターよりも少ないため、これらの演算子はIDよりもクラスセレクターではるかに人気がありますが、これらすべての組み合わせの使用法は見た目にも優れています。

要素ごとのクラス

OOCSS、アトミック、および機能的なCSS戦略の登場により要素に10以上のクラスを構成してデザインの外観を実現できるため、おそらく興味深い結果が得られるでしょう。クエリは非常に刺激的でなく、モバイルとデスクトップの中央値は要素ごとに1クラスでした。

1

図2.11. クラス属性ごとのクラス名の中央値数（デスクトップおよびモバイル）。

レイアウト

Flexbox

Flexbox[46]は、子を指示、整列するコンテナスタイルです。つまり制約ベースの方法でレイアウトを支援します。 2010年から2013年の間に仕様が2〜3の大幅な変更を経たため、Webでの開始は非常に困難でした。幸いなことに、2014年までにすべてのブラウザに落ち着き実装されました。その歴史を考えると採用率は低かったのですが、それから数年が経ちました！　今では非常に人気があり、それに関する多くの記事とそれを活用する方法がありますが、他のレイアウト戦術と比較してまだ新しいです。

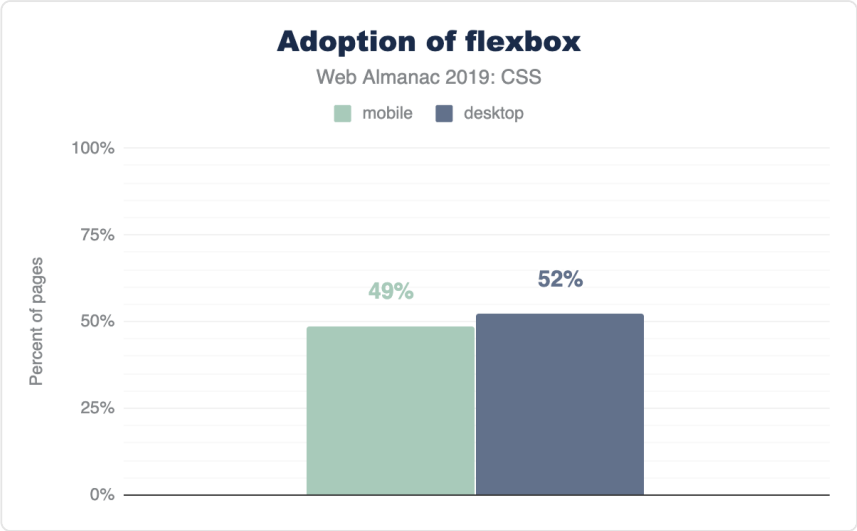


図2.12. フレックスボックスの採用。

Webのほぼ50%がスタイルシートでflexboxを使用しているため、ここに示したかなりの成功事例です。

Grid

flexboxと同様に、グリッド[47]も早い段階でいくつかの仕様変更を経ましたが、公的に展開されたブラウザの実装を変更することはありませんでした。 Microsoftは、水平方向にスクロールするデザインスタイルの主要なレイアウトエンジンとして、Windows 8の最初のバージョンにグリッドを備えていました。最初にそこで検証され、Webに移行し、その後、2017

46. https://developer.mozilla.org/docs/Web/CSS/CSS_Flexible_Box_Layout/Basic_Concepts_of_Flexbox
47. https://developer.mozilla.org/docs/Web/CSS/CSS_Grid_Layout

年の最終リリースまで他のブラウザーによって強化されました。ほぼすべてのブラウザーが同時に実装をリリースしたため、Web開発者はある日目覚めただけで優れたグリッドサポートを得ることができました。今日2019年の終わりに、グリッドはまだ子供のように感じています。人々がまだその力と能力に気付き始めているため。

2%

図2.13. グリッドを使用するWebサイトの割合。

これは、Web開発コミュニティが最新のレイアウトツールを使用して調査したことがどれほど少ないかを示しています。主要なレイアウトエンジンの人々がサイトを構築する際に頼るので、グリッドの最終的な引継ぎを楽しみにしています。著者にとって、私たちはグリッドを書くのが大好きです。通常、最初にグリッドへ到達し、次にレイアウトを実現し、繰り返しながら複雑さをダイヤルバックします。今後数年間で、この強力なCSS機能を使用して他の地域がどうなるかは今後の課題です。

書き込みモード

WebとCSSは国際的なプラットフォーム機能であり、書き込みモードはHTMLとCSSが要素内でユーザーの好みの読み取りと書き込みの方向を示す方法を提供します。

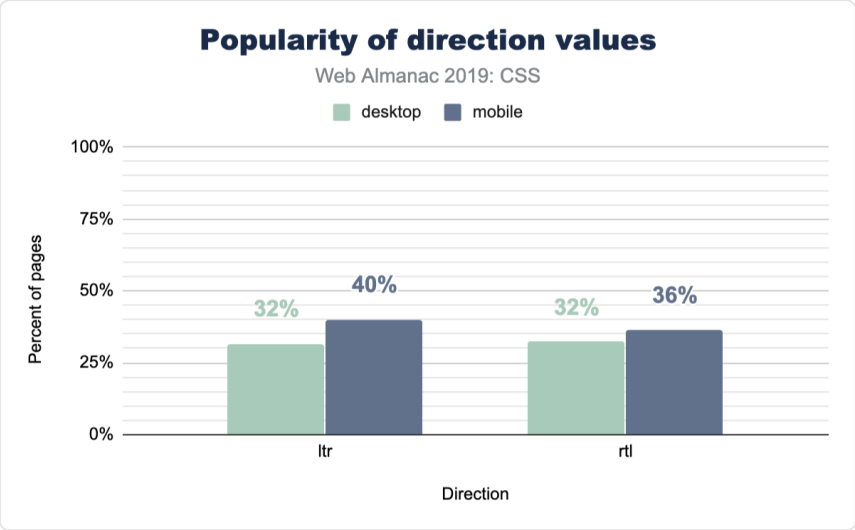


図2.14. 方向の値の人気。

タイポグラフィ

ページごとのWebフォント

Webページにいくつのwebフォントをロードしていますか：0？　10？　1ページあたりのWebフォントの中央値は3です！

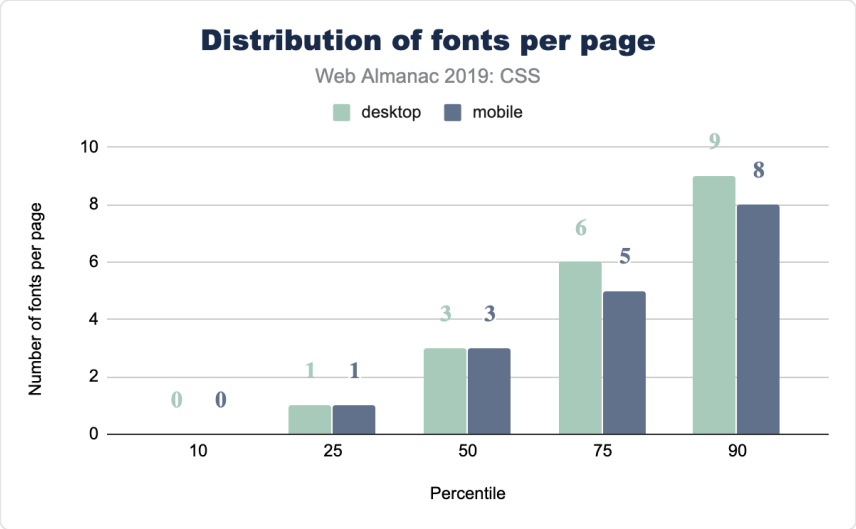


図2.15. ページごとにロードされるWebフォントの数の分布。

人気のあるフォントファミリー

ページあたりのフォントの総数の問い合わせに対する自然な回答は、次のとおりです。それらはどのフォントですか?!　デザイナーは、あなたの選択が人気のあるものと一致しているかどうかを確認できるようになります。

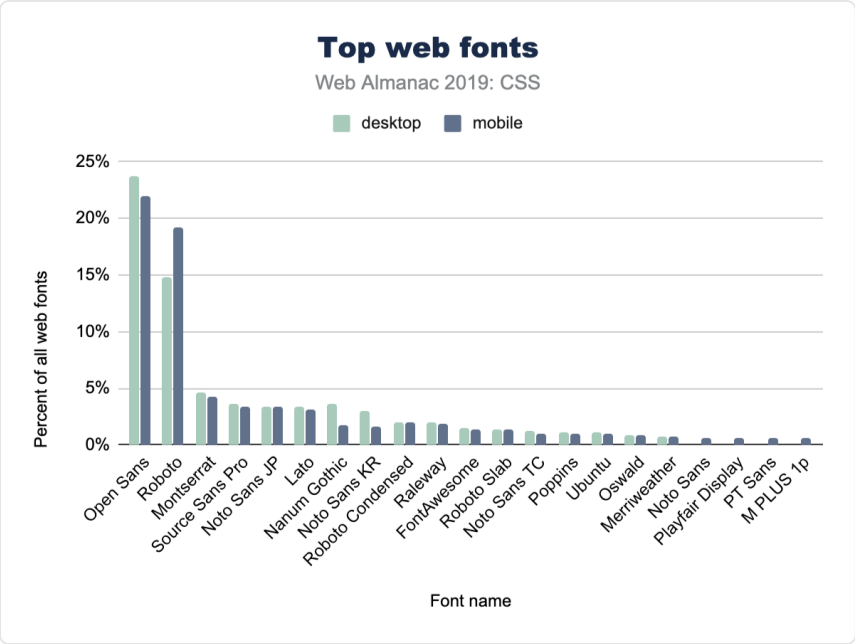


図2.16. 上位のWebフォント。

ここではOpen Sansが大きな勝者であり、CSSの `@font-family` 宣言の4分の1近くがそれを指定しています。私たちは間違いなく、プロジェクトでOpen Sansを使用しています。

また、デスクトップ導入とモバイル導入の違いに注目することも興味深いです。たとえば、モバイルページはデスクトップよりもOpen Sansの使用頻度がわずかに低いです。一方、デスクトップはRobotoを少しだけ頻繁に使用します。

フォントサイズ

これは楽しいものです。ユーザーがページ上にあると感じるフォントサイズの数をユーザーに尋ねた場合、通常5または10未満の数値が返されるからです。デザインシステムでフォントサイズはいくつありますか？　Webに問い合わせたところ、中央値はモバイルで40、デスクトップで38でした。タイプランプの配布に役立つカスタムプロパティや再利用可能なクラスの作成について真剣に考える時間になるかもしれません。

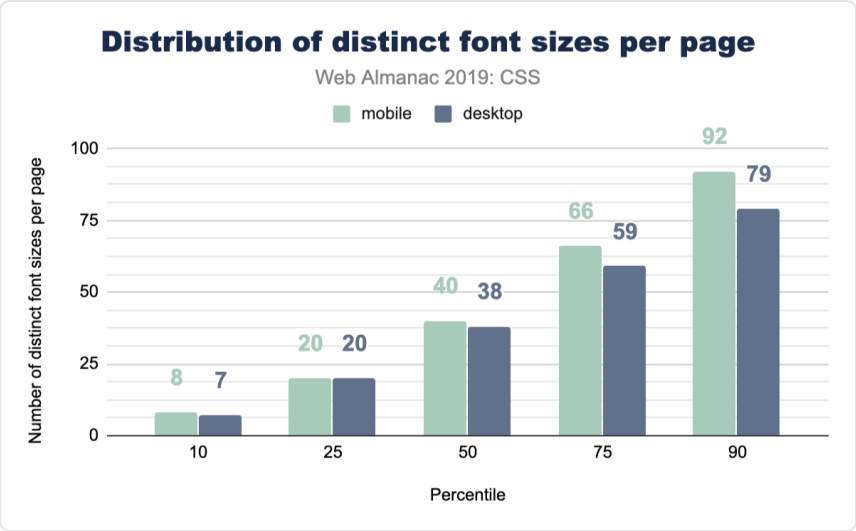


図2.17. ページごとの異なるフォントサイズの数の分布。

間隔

マージン

マージンとは、自分の腕を押し出すときに要求するスペースのような要素の外側のスペースです。これは多くの場合、要素間の間隔のように見えますが、その効果に限定されません。Webサイトまたはアプリでは、間隔はUXとデザインで大きな役割を果たします。スタイルシートにどのくらいのマージン間隔コードが入るか見てみましょうか？

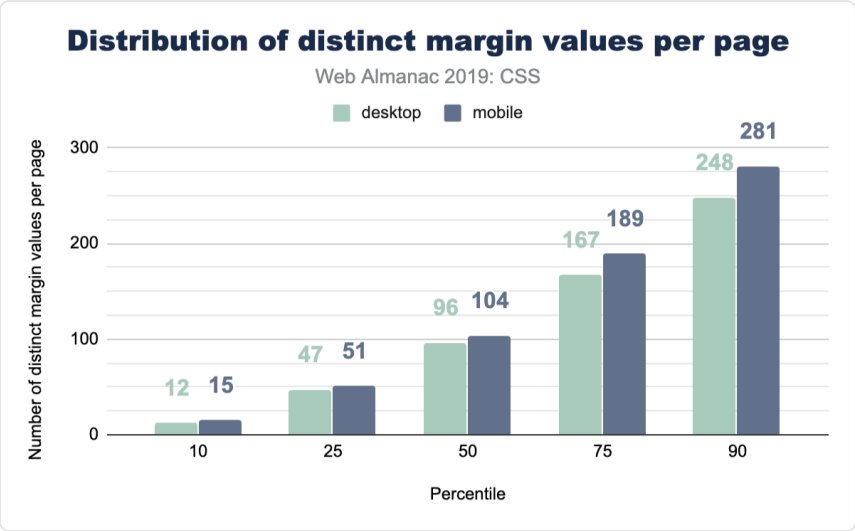


図2.18. ページごとの異なるマージン値の数の分布。

かなりたくさんのようです！　デスクトップページの中央値には96の異なるマージン値があり、モバイルでは104です。これにより、デザインに多くのユニークな間隔が生まれます。あなたのサイトにいくつのマージンがあるか知りたい？　この空白をすべて管理しやすくするにはどうすればよいですか？

論理プロパティ

0.6%

図2.19. 論理プロパティを含むデスクトップおよびモバイルページの割合。

`margin-left` と `padding-top` の覇権は限られた期間であり、書き込み方向に依存しない、連続した論理プロパティ構文によりまもなく補完されると推定します。楽観的ではありますが、現在の使用量は非常に低く、デスクトップページでの使用量は0.67%です。私たちにとって、これは業界として開発する必要がある習慣の変化のように感じられますが、新しいシンタックスを使用するために新しい開発者を訓練することを願っています。

z-index

CSSの `z-index` を使用して、垂直の階層化またはスタックを管理できます。私たちは、人々

が自分のサイトでどれだけ多くの価値を使用しているかに興味がありました。 `z-index` が受け入れる範囲は理論的には無限であり、ブラウザーの可変サイズの制限によってのみ制限されます。それらすべてのスタック位置が使用されていますか？　では見てみよう！

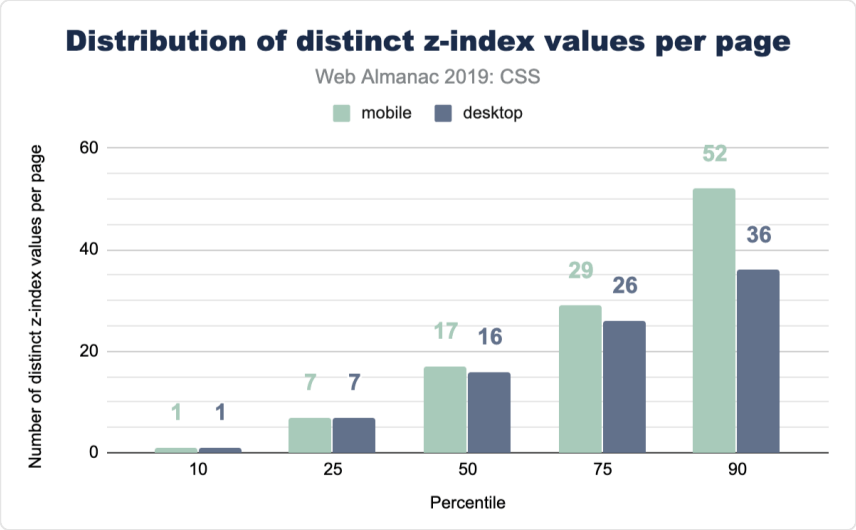


図2.20. ページごとの個別の `z-index` 値の数の分布。

人気のあるz-index値

私たちの仕事の経験から、9の任意の数が最も一般的な選択肢であると思われました。可能な限り少ない数を使用するように教えたにもかかわらず、それは共同の基準ではありません。じゃあ何ですか?!　人々が一番上のものを必要とする場合、最も人気のあるZインデックス番号は何ですか？　飲み物を置いてください。これはあなたがそれを失うかもしれないので十分面白いです。

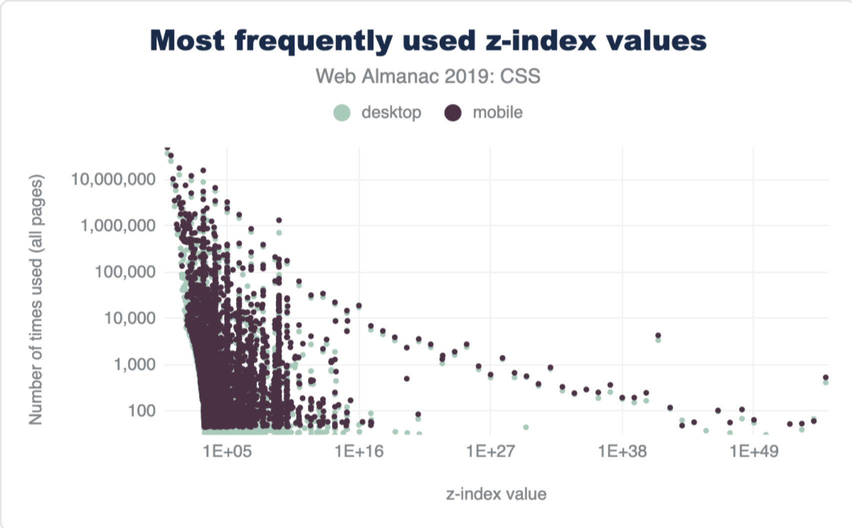


図2.21. 最も頻繁に使用される `z-index` 値。

99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
99999999999999999999999999999999999999999999999
9999999999999999999999999999 !important

図2.22. 最も頻繁に使用される `z-index` 値。

デコレーション

フィルター

フィルターは、ブラウザーが画面に描画するピクセルを変更するための楽しくて素晴らしい方法です。これは、適用対象の要素、ノード、またはレイヤーのフラットバージョンに対して実行される後処理効果です。 Photoshopによって使いやすくなり、Instagramによって、オーダーメイドの定型化された組み合わせによって大衆がアクセスできるようになりました。それらは2012年頃から存在し、10個あります。それらを組み合わせて独自の効果を作成できます。

78%

図2.23. `フィルター` プロパティを持つスタイルシートを含むページの割合。

スタイルシートの78%に `フィルター` プロパティが含まれていることがわかりました。その数も非常に高かったので、少し怪しいように思えたので、私たちは深掘りしてその高い数を説明しようとしました。正直に言って、フィルターはきちんとしていますが、すべてのアプリケーションやプロジェクトに組み込まれているわけではありません。しない限り！

さらなる調査の結果、FontAwesome[48]のスタイルシートには `フィルター` の使用法とYouTube[49]埋め込みが含まれていることがわかりました。そのため、非常に人気のあるいくつかのスタイルシートに便乗することで、バックドアに `フィルター` が入り込むと考えています。また、`-ms-filter` の存在も含まれている可能性があり、使用率が高くなっていると考えられます。

ブレンドモード

ブレンドモードは、ターゲット要素のフラットバージョンに対して実行される後処理効果であるという点でフィルターに似ていますが、ピクセル収束に関係しているという点で独特です。別の言い方をすれば、ブレンドモードとは、2つのピクセルが重なり合ったときに互いに影響を与える方法です。上部または下部のどちらの要素でも、ブレンドモードがピクセルを操作する方法に影響します。 16種類のブレンドモードがあります。どのモードが最も人気かを見てみましょう。

48. https://fontawesome.com
49. https://youtube.com

8%

図2.24. `*-blend-mode` プロパティを持つスタイルシートを含むページの割合。

全体的に、ブレンドモードの使用はフィルターの使用よりもはるかに低いですが、適度に使用されていると見なすのに十分です。

Web Almanacの今後のエディションでは、ブレンドモードの使用法にドリルダウンして、開発者が使用している正確なモード（乗算、スクリーン、カラーバーン、ライトなど）を把握することをお勧めします。

アニメーション

トランジション

CSSには、トランジションのこれらの値の方法に関する単一のルールを記述するだけで簡単に使用できるこの素晴らしい補間機能があります。アプリの状態を管理するためにCSSを使用している場合、タスクを実行するためにトランジションを使用する頻度はどれくらいですか？　Webに問合せしましょう！

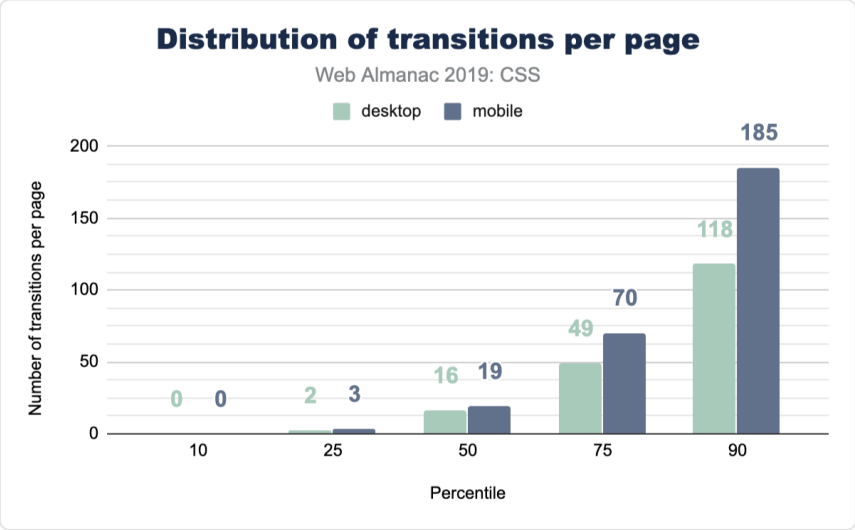


図2.25. ページごとの遷移数の分布。

それはかなり良いです！　私たちは`animate.css`を含めるべき人気のあるライブラリと考えていました。これはたくさんのトランジションアニメーションをもたらしますが、人々がUIのトランジションを検討しているのを見るのは今でも素晴らしいことです。

キーフレームアニメーション

CSSキーフレームアニメーションは、より複雑なアニメーションやトランジションに最適なソリューションです。これにより、効果をより明確に制御できるようになります。 1つのキーフレームエフェクトのように小さくすることも、多数のキーフレームエフェクトを堅牢なアニメーションに合成して大きくすることもできます。ページあたりのキーフレームアニメーションの数の中央値は、CSSトランジションよりもはるかに低くなっています。

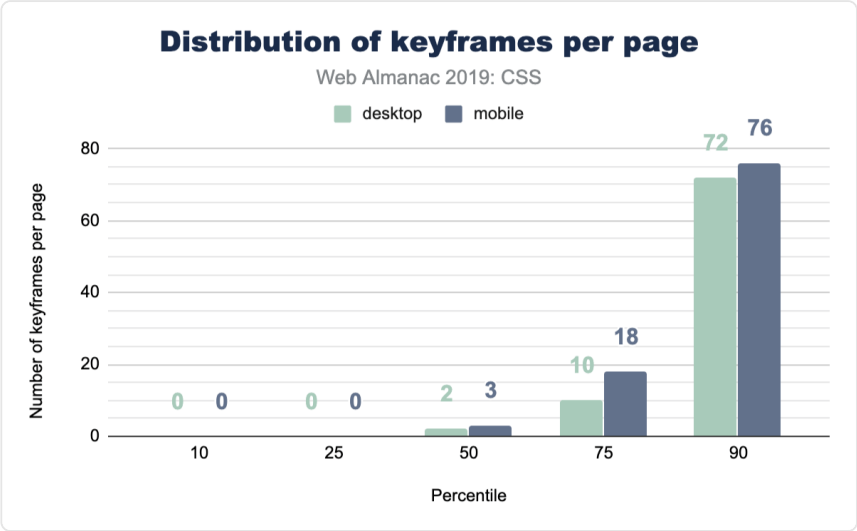


図2.26. ページごとのキーフレーム数の分布。

メディアクエリ

メディアクエリを使用すると、CSSをさまざまなシステムレベルの変数にフックして、訪問ユーザーに適切に適応させることができます。これらのクエリの一部は、印刷スタイル、プロジェクタースクリーンスタイル、ビューポート/スクリーンサイズを処理できます。長い間、メディアクエリは主にビューポートの知識のために活用されていました。デザイナーと開発者は、小さな画面、大きな画面などにレイアウトを適合させることができます。その後、ウェブはますます多くの機能とクエリを提供し始めました。つまり、メディアクエリはビューポート機能に加えてアクセシビリティ機能を管理できるようになりました。

メディアクエリから始めるのに適した場所は、1ページあたりの使用数です。典型的なページが応答したいと感じるのは、いくつの瞬間やコンテキストですか？

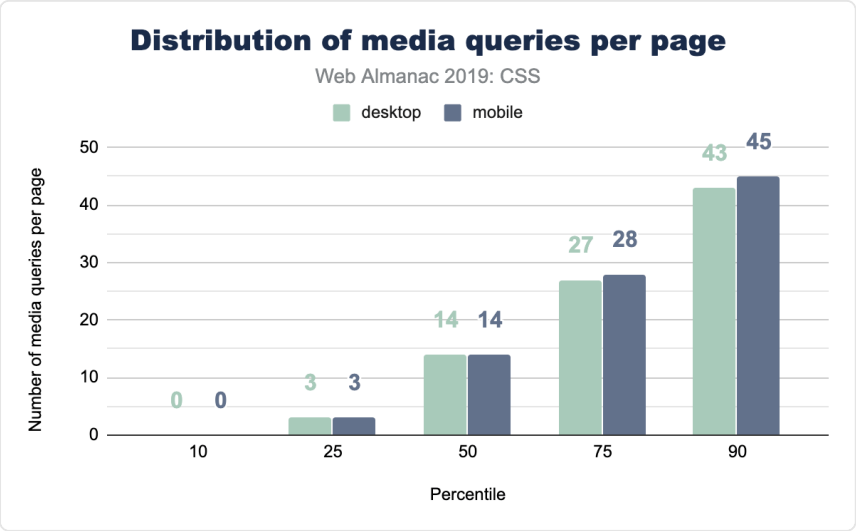


図2.27. ページごとのメディアクエリ数の分布。

一般的なメディアクエリブレークポイントサイズ

ビューポートメディアクエリの場合、任意のタイプのCSSユニットを評価用のクエリ式に渡すことができます。以前、人々は`em`と`px`をクエリに渡していましたが、時間がたつにつれて単位が追加され、Webで一般的に見られるサイズの種類について非常に興味を持ちました。ほとんどのメディアクエリは一般的なデバイスサイズに従うと想定していますが、想定する代わりにデータを見てみましょう。

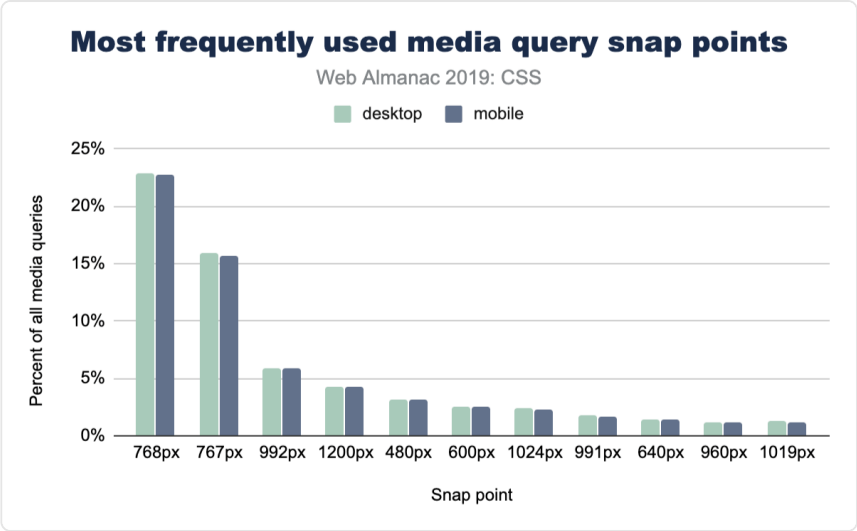


図2.28. メディアクエリで使用される最も頻繁に使用されるスナップポイント。

上記の図2.28は、前提の一部が正しいことを示しています。確かに、大量のモバイル固有のサイズがありますが、そうでないものもあります。また、このチャートの範囲を超えて`em`を使用するいくつかのトリックエントリで、非常にピクセルが支配的であることも興味深いです。

ポートレートとランドスケープの使用

人気のあるブレークポイントサイズからの最も人気のあるクエリ値は`768px`であるため、興味をそそられました。この値は、`768px`が一般的なモバイルポートレートビューポートを表すという仮定に基づいている可能性があるため、主にポートレートレイアウトへ切り替えるために使用されましたか？　そこで、ポートレートモードとランドスケープモードの使用の人気を確認するために、フォローアップクエリを実行しました。

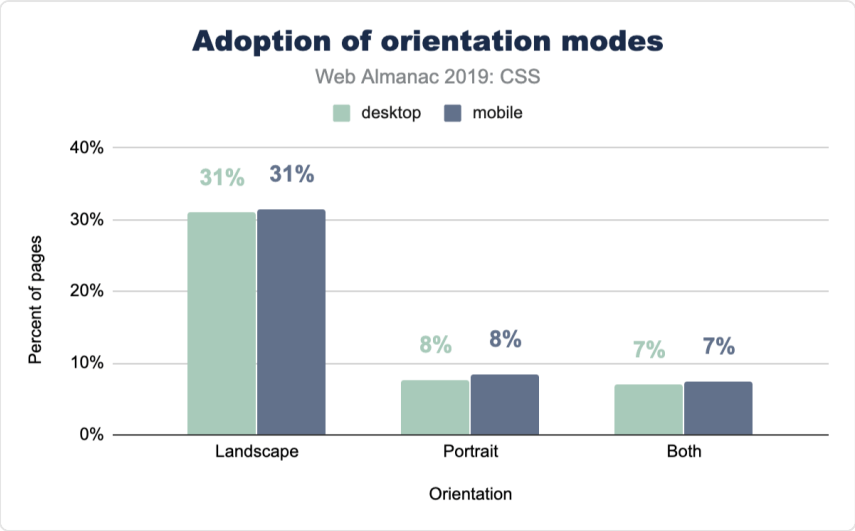


図2.29. メディアクエリの方向モードの採用。

興味深いことに、`ポートレート`はあまり使用されませんが、`ランドスケープ`はより多く使用されます。`768px`はポートレートレイアウトのケースとして十分に信頼できるものであり、到達できるコストははるかに少ないと想定できます。また、デスクトップコンピューターで作業をテストしているユーザーは、ブラウザーを押しつぶすほど簡単にモバイルレイアウトを見るためにポートレートをトリガーできないと想定しています。わかりにくいですが、データは魅力的です。

最も人気のあるユニットタイプ

これまで見てきたメディアクエリの幅と高さでは、ピクセルはUIをビューポートに適合させることを考えている開発者にとって主要な選択単位のように見えます。ただし、これを排他的にクエリしたいので、実際に人々が使用するユニットのタイプを見てみましょう。これは私たちが見つけたものです。

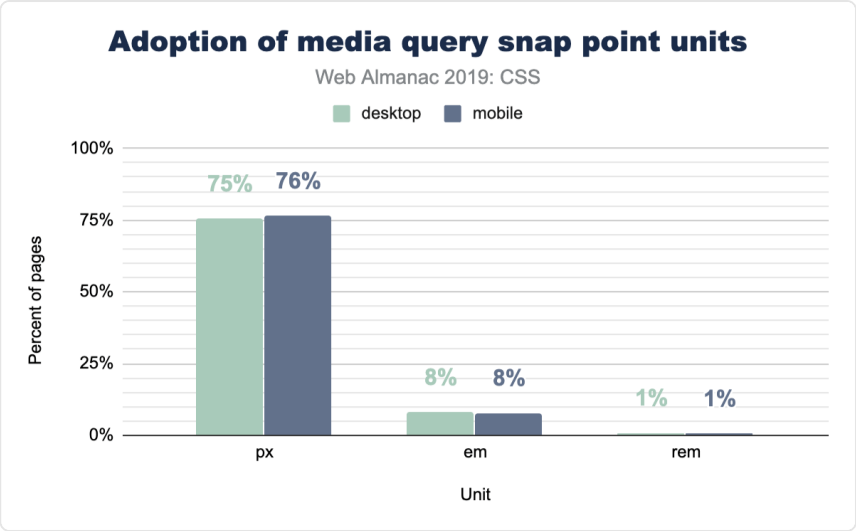


図2.30. メディアクエリスナップポイントでのユニットの採用。

`min-width` と `max-width`

人々がメディアクエリを書くとき、彼らは通常、特定の範囲を超えているか下にあるビューポート、またはその両方をチェックして、サイズの範囲内にあるかどうかをチェックしてるでしょうか？　ウェブに聞いてみましょう！

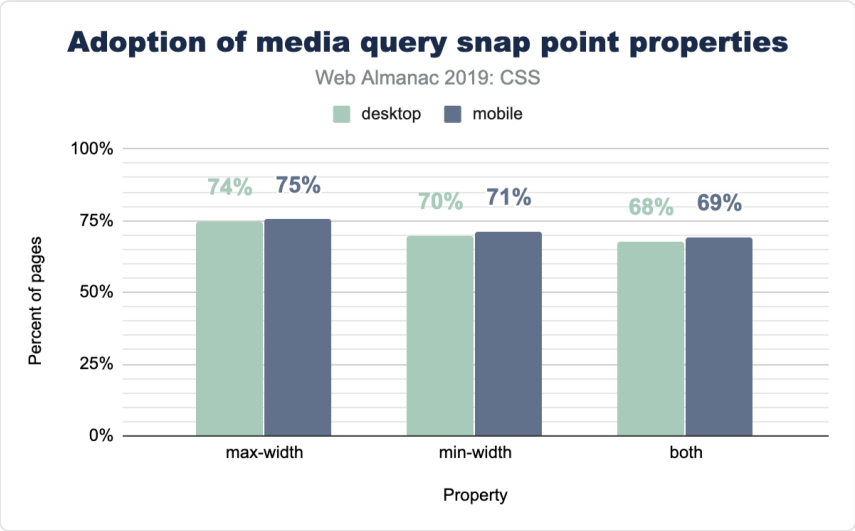


図2.31. メディアクエリスナップポイントで使用されるプロパティの採用。

ここには明確な勝者はありません。 `max-width` と `min-width` はほぼ同じように使用されます。

printとspeech

Webサイトはデジタルペーパーのように感じますか？　ユーザーとしては、ブラウザーから印刷するだけで、そのデジタルコンテンツを物理コンテンツに変換できることが一般的に知られています。 Webサイトは、そのユースケースに合わせて変更する必要はありませんが、必要に応じて変更できます。あまり知られていないのは、ツールまたはロボットによって読み取られるユースケースでWebサイトを調整する機能です。では、これらの機能はどれくらいの頻度で活用されていますか？

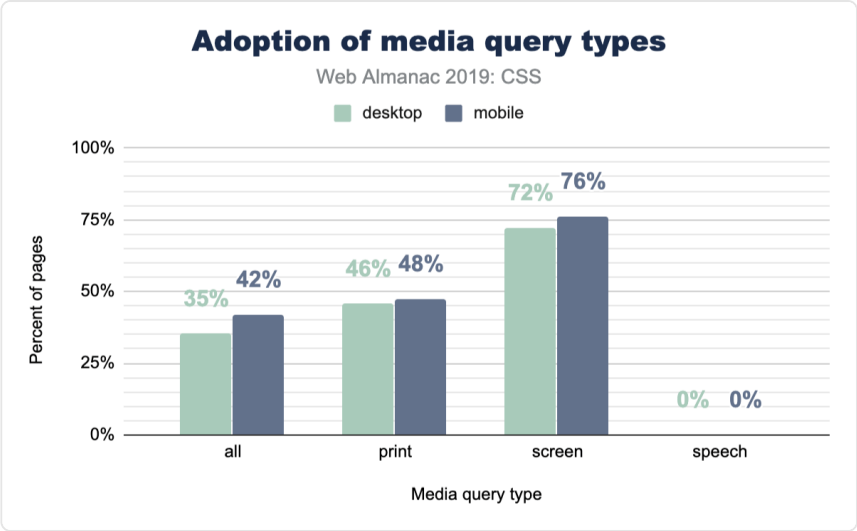


図2.32. メディアクエリのall、print、screen、およびspeechタイプの採用。

ページレベルの統計

スタイルシート

ホームページから何枚のスタイルシートを参照していますか？　アプリからはどのくらい？　モバイルとデスクトップのどちらにサービスを提供していますか？　ここに他のみんなのチャートがあります！

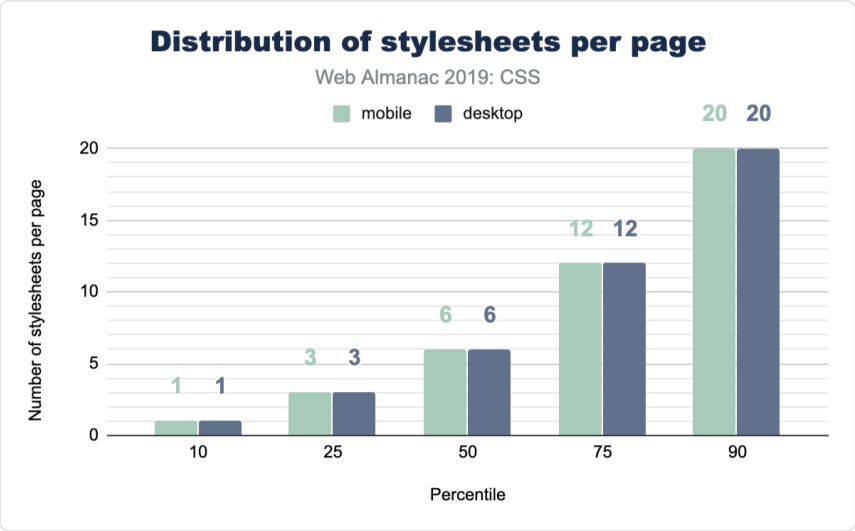


図2.33. ページごとにロードされるスタイルシートの数の分布。

スタイルシート名

スタイルシートの名前は何ですか？　あなたのキャリアを通して一貫した名前にしましたか？　ゆっくり収束したか、一貫して発散しましたか？　このチャートは、ライブラリの人気を少し垣間見せています。また、CSSファイルの一般的な名前を垣間見ることもできます。

スタイルシート名	デスクトップ	モバイル
style.css	2.43%	2.55%
font-awesome.min.css	1.86%	1.92%
bootstrap.min.css	1.09%	1.11%
BfWyFJ2Rl5s.css	0.67%	0.66%
style.min.css?ver=5.2.2	0.64%	0.67%
styles.css	0.54%	0.55%
style.css?ver=5.2.2	0.41%	0.43%
main.css	0.43%	0.39%
bootstrap.css	0.40%	0.42%
font-awesome.css	0.37%	0.38%
style.min.css	0.37%	0.37%
styles__ltr.css	0.38%	0.35%
default.css	0.36%	0.36%
reset.css	0.33%	0.37%
styles.css?ver=5.1.3	0.32%	0.35%
custom.css	0.32%	0.33%
print.css	0.32%	0.28%
responsive.css	0.28%	0.31%

図2.34. 最も頻繁に使用されるスタイルシート名。

それらすべてのクリエイティブなファイル名を見てください！　スタイル、スタイル、メイン、デフォルト、すべて。しかし目立ったのは、あなたはわかりますか？ `BfWyFJ2Rl5s.css` は、最も人気のある4位になります。少し調べてみましたが、Facebookの「いいね」ボタンに関連していると思われます。そのファイルが何であるか知っていますか？　話を聞きたいので、コメントを残してください。

スタイルシートのサイズ

これらのスタイルシートはどれくらいの大きさですか？　CSSのサイズは心配する必要がありますか？　このデータから判断すると、CSSはページ膨張の主な攻撃者ではありません。

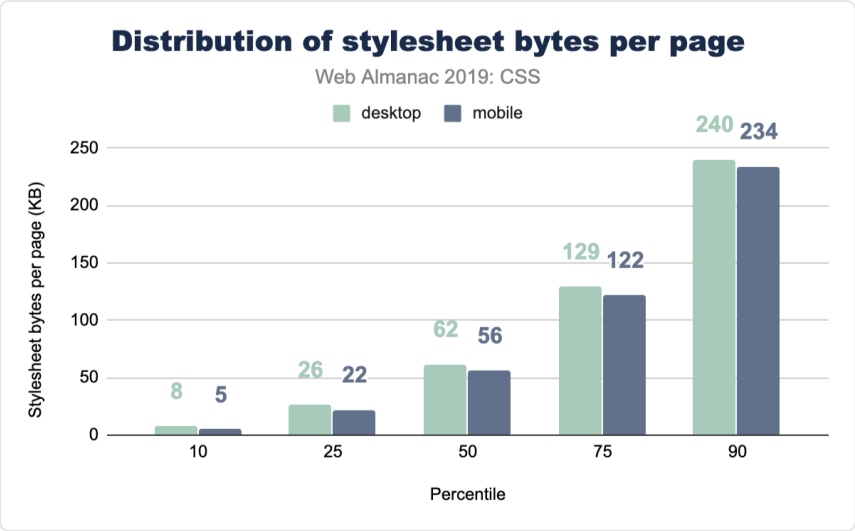


図2.35. ページごとにロードされるスタイルシートのバイト数（KB）の分布。

Webサイトが各コンテンツタイプにロードするバイト数の詳細については、ページウェイトの章を参照してください。

ライブラリ

新しいプロジェクトをスタートする事にCSSライブラリへ手を出すのは一般的で、人気があり、便利で強力です。あなたはライブラリに手を伸ばす人ではないかもしれませんが、私たちは2019年にウェブへ問い合わせて、どれが群を抜いているか調べました。彼らが私たちと同じように結果に驚くなら、開発者バブルがどれだけ小さいかを知る手がかりになると思います。物事は非常に人気がありますが、ウェブに問い合わせると、現実は少し異なります。

ライブラリ	デスクトップ	モバイル
Bootstrap	27.8%	26.9%
animate.css	6.1%	6.4%
ZURB Foundation	2.5%	2.6%
UIKit	0.5%	0.6%
Material Design Lite	0.3%	0.3%
Materialize CSS	0.2%	0.2%
Pure CSS	0.1%	0.1%
Angular Material	0.1%	0.1%
Semantic-ui	0.1%	0.1%
Bulma	0.0%	0.0%
Ant Design	0.0%	0.0%
tailwindcss	0.0%	0.0%
Milligram	0.0%	0.0%
Clarity	0.0%	0.0%

図2.36. 特定のCSSライブラリを含むページの割合。

このチャートは、Bootstrap[50]がプロジェクトを支援するために知っておくべき貴重なライブラリであることを示唆しています。支援する機会があるすべてを見てください！　すべてのサイトがCSSフレームワークを使用しているわけではないので、これはポジティブなシグナルチャートにすぎないことも注目に値します。100%に達することはありません。すべてのサイトの半分以上が、既知のCSSフレームワークを使用していません。とても面白いですよね！

リセットユーティリティ

CSSリセットユーティリティは、ネイティブWeb要素のベースラインを正規化または作成することを目的としています。あなたが知らなかった場合、各ブラウザはすべてのHTML要素に対して独自のスタイルシートを提供し、それら要素の外観、動作について独自の決定を下すことができます。リセットユーティリティはこれらのファイルを調べ、共通点を見つけた

50. https://getbootstrap.com/

（もしくは見つけなかった）ため、開発者が1つのブラウザーでスタイルを設定し、別のブラウザーでも同じように見える合理的な自信を持たせるため、相違点を解決しました。

それで、どれだけのサイトがそれを使っているかを見てみましょう！　彼らの存在はかなり理にかなっているように思えるので、何人の人々が彼らの戦術に同意し、彼らのサイトでそれらを使用しますか？

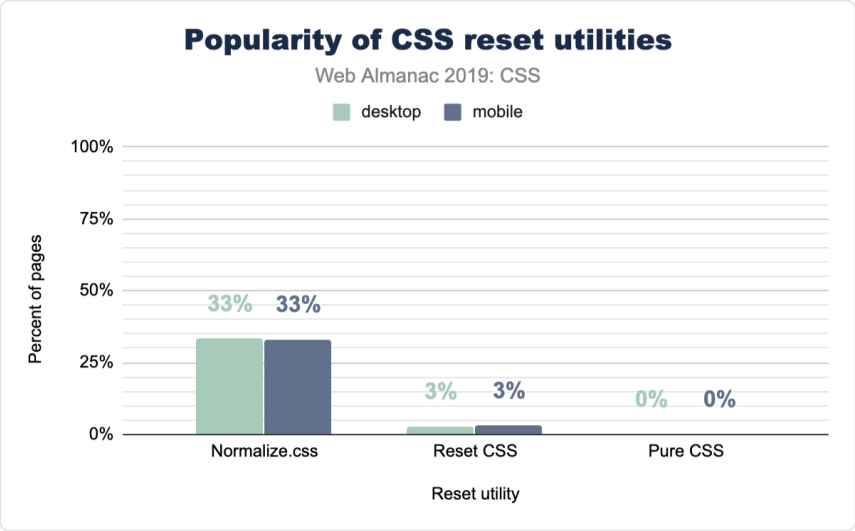


図2.37. CSSリセットユーティリティの採用。

Webの約3分の1が`normalize.css`を使用していることがわかります。これは、リセットよりもタスクへのより穏やかなアプローチと考えることができます。少し詳しく見てみると、Bootstrapには`normalize.css`が含まれていることがわかりました。`normalize.css`がBootstrapよりも多く採用されていることも注目に値するので、それを単独で使用する人がたくさんいます。

`@supports`と`@import`

CSS `@supports`は、ブラウザが特定のプロパティと値の組み合わせが有効であると解析されたかどうかをチェックし、チェックがtrueを返した場合にスタイルを適用する方法です。

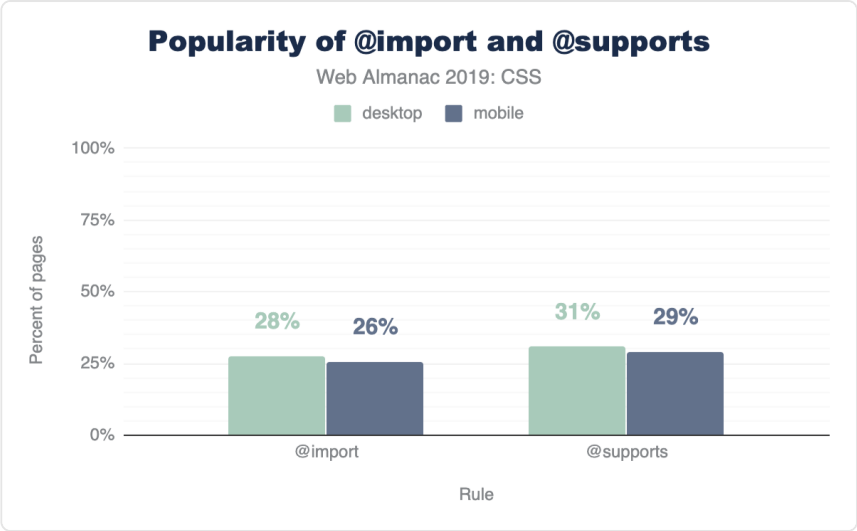


図2.38. CSS「@」ルールの人気

2013年にほとんどのブラウザで `@supports` が実装されたことを考慮すると、大量の使用と採用が見られることはそれほど驚くことでありません。ここでは、開発者のマインドフルネスに感銘を受けています。これは思いやりのあるコーディングです！　すべてのWebサイトの30%は、使用する前にディスプレイ関連のサポートをチェックしています。

これの興味深いフォローアップは、 `@imports` より `@supports` の使用が多いことです！ 私たちはそれを期待していませんでした！ `@import` は1994年以来ブラウザに存在しています。

結論

ここには、データマイニングするための非常に多くのものがあります！　結果の多くは私たちを驚かせました、そして同様にあなたも驚いたことを願っています。この驚くべきデータセットにより、要約が非常に楽しくなり、結果の一部がそうである理由を追い詰めたいかどうかを調査するための多くの手がかりと追跡の跡が残されました。

どの結果が最も驚くべきものでしたか？ どの結果を使用して、コードベースにすばやくクエリを移動しますか？

これらの結果からの最大のポイントは、スタイルシートのパフォーマンス、乾燥、スケーラビリティの点で、カスタムプロパティが予算に見合った価値を提供することだと感じました。インターネットのスタイルシートを再度スクラブし、新しいデータムと挑発的なチャー

トの扱いを探しています。クエリ、質問、アサーションを含むコメントで@unaまたは@argyleinkに連絡してください。私たちはそれらを聞きたいです！

著者

Una Kravets

@una　una　http://una.im

Una Kravetsは、ブルックリンを拠点とする国際的な講演者であり、テクニカルライターであり、Googleのマテリアルデザインの開発提唱者でもあります。Unaは、Designing the Browser[51] のウェブシリーズと Toolsday[52] の開発者向けポッドキャストを主催しています。Twitter[53] で彼女をフォローして、クリエイティブなCSS、ユーザー体験、ウェブ開発のベストプラクティスについての彼女の考察を見つけてください。

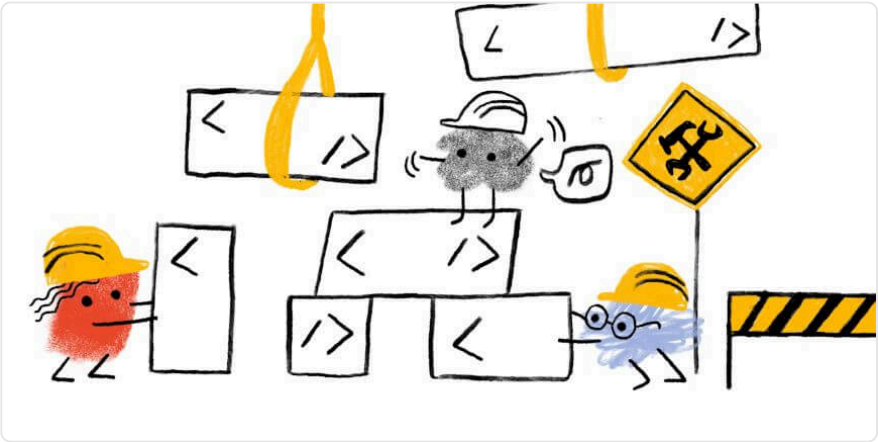
Adam Argyle

@argyleink　argyleink　https://nerdy.dev

Adam ArgyleはGoogle Chrome開発者リレーションズのメンバーで、CSSを中心に活動しています。優れたUX & UIを求める飽くなき欲望を持つウェブ中毒者です。@argyleink or checkout his website https://nerdy.dev。

51. *https://www.youtube.com/watch?v=YK8GZBx3hpg*
52. *https://spec.fm/podcasts/toolsday*
53. *https://x.com/una*

部 I 章 3
マークアップ



***Brian Kardell** によって書かれた。*
Simon Pieters、 Tommy Hodgins と Matthew Phillips によってレビュ。
Rick Viscomi による分析 と 編集。
M.Sakamaki によって翻訳された。

導入

2005年にIan “Hixie” Hicksonはこれまでの研究に基づいたマークアップデータの分析[54]を投稿しました。この作業のほとんどは、クラス名を調査して、内々で開発者が採用しているセマンティクスを確認し、標準化する意味があるかの確認をすることが目的でした。この研究の一部は、HTML5の新要素の参考として役に立ちました。

14年すぎて、新しい見方をする時が来ました。以降、カスタム要素(Custom Elements)[55]とExtensible Web Manifesto[56]の導入により、開発者は要素そのものの空間を探し、標準化団体が辞書編集者のようになる[57]ことで、牛の通り道を舗装する(pave the cowpaths)よりよい方法を見つけることを推奨しています。様々なものに使われる可能性があるCSSクラス名とは異なり、非標準の要素は作成者が要素であることを意識しているため、さらに確実なものとなります。

54. *https://web.archive.org/web/20060203035414/http://code.google.com/webstats/index.html*
55. *https://developer.mozilla.org/docs/Web/Web_Components/Using_custom_elements*
56. *https://extensiblewebmanifesto.org/*
57. *https://bkardell.com/blog/Dropping-The-F-Bomb-On-Standards.html*

2019年7月の時点で、HTTP Archiveは、約440万のデスクトップホームページと約530万のモバイルホームページのDOMで使用されているすべての*要素*名の収集を開始しました。*(方法論の詳細を御覧ください)*

このクロールの結果、*5,000種類を超える非標準の要素*が検出されたため、計測する要素の合計数を「トップ」（以下で説明）5,048種類に制限しました。

方法論

各ページの要素の名前は、JavaScriptの初期化後DOMより収集されました。

現実の要素出現数を確認することは標準の要素であっても有用ではありません、検出されたすべての要素の約25%は `<div>` です。そして、約17%が `<a>` で、11%が `<span>` となっており、これらは10%以上を占める唯一の要素たちです。言語は一般的にこのようなもの[58]ですが、これと比較してみると驚くほど少ない用語が使われています。さらに、非標準の要素の取り込みを検討してみると、１つのサイトが特定の要素を1000回も使用しているために、とても人気があるように見えてしまい、大きな誤解を招く可能性があります。

そのような方法を取らず、私達はHixieの元の研究のようにホームページに各要素が少なくとも1回は含まれているサイトの数に着目しました。

注意: この方法は潜在的なバイアスが無いとは言い切れません。人気のある製品は複数のサイトで使われています。これにより個々の作成者は意識していない非標準のマークアップが導入されるでしょう。したがって、この方法は一般的なニーズに対応するのと同じように、作成者の直接的な知識や意識的な採用を意味しないことに注意する必要があります。調査中に、このような例はいくつか見つかりました。

上位の要素と概説

2005年、Hixieはページ中に最もよく使用されていて、頻度の少ない上位要素を調査しました。トップ３は `html`、`head`、`body` でした、これらはオプションなので省略されてもパーサーによって作成されており、彼は興味深いと述べています。パーサーによる解析後のDOMを使って調査すると、データは普遍的に表示されます。なので、４番目に使用頻度の高い要素からはじめました。以下は、その時点から現在までのデータの比較です。(ここでは面白いので出現数を含めました)

58. *https://www.youtube.com/watch?v=fCn8zs912OE*

2005 (サイト毎)	2019 (サイト毎)	2019 (出現数)
title	title	div
a	meta	a
img	a	span
meta	div	li
br	link	img
table	script	script
td	img	p
tr	span	option

図3.1. 2005年から2019年までの上位要素の比較。

ページ毎の要素

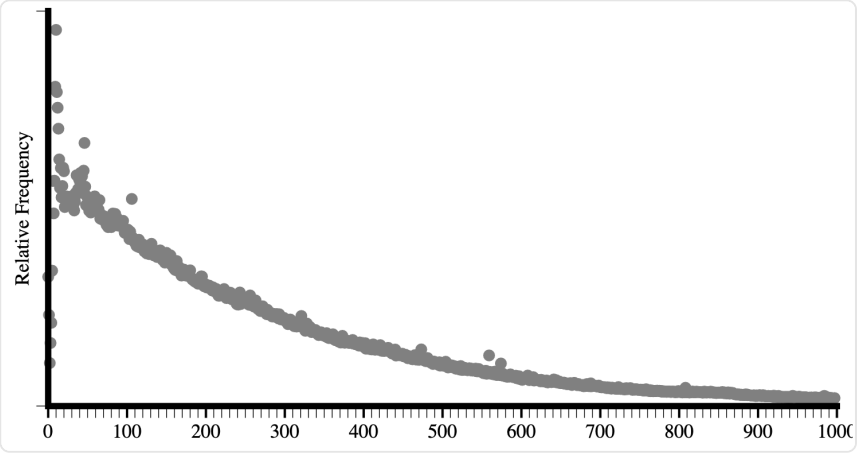


図3.2. Hixieによる2005年の要素頻度の分布。

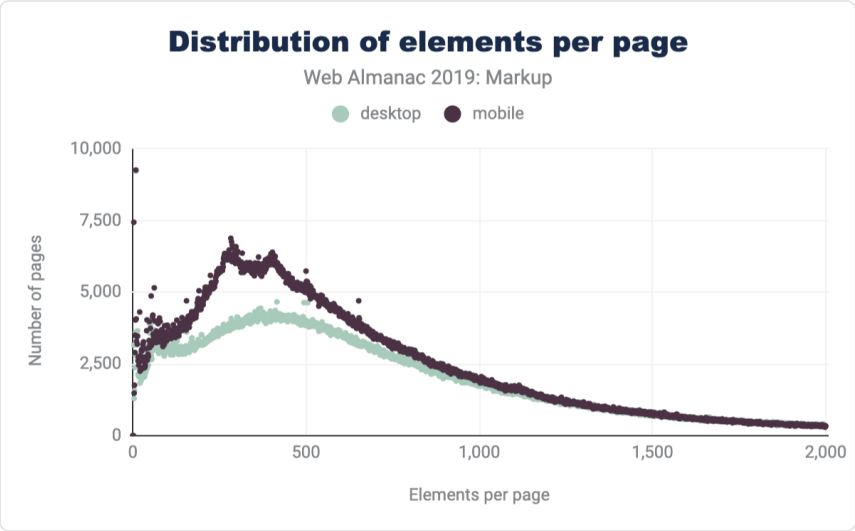


図3.3. 2019年の要素頻度。

図3.2の2005年のHixieのレポートと図3.3の最新データを比較すると、DOMツリーの平均サイズが大きくなっていることがわかります。

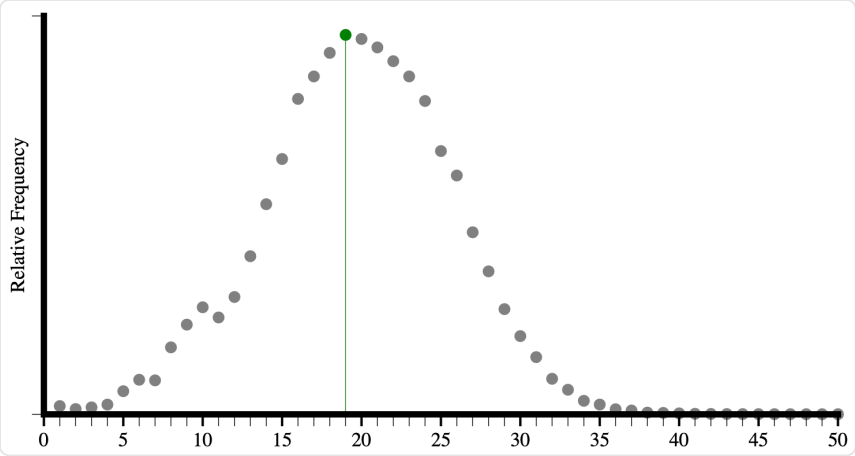


図3.4. 2005年にHixieが分析したページ毎の要素タイプのヒストグラム。

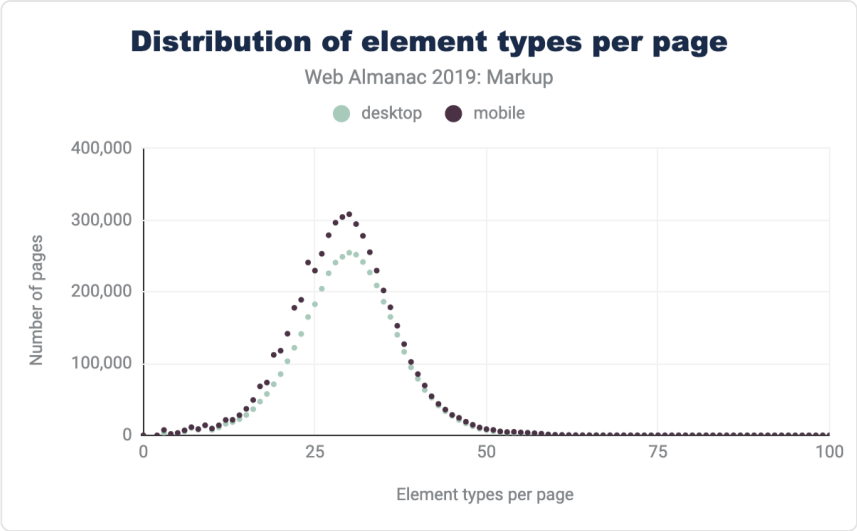


図3.5. 2019年時点でのページ毎の要素タイプのヒストグラム。

ページあたりの要素の種類の平均数と、ユニークな要素数の最大値の両方が増加していることがわかります。

カスタム要素

記録された要素のほとんどはカスタム（単純に「非標準」となる物）でした。しかし、どの要素がカスタムであるか、カスタムではないかを議論するのは少し面倒です。実際にかなりの数の要素が仕様や提案のどこかに書かれています。今回、244個の要素を標準として検討しました。（ただし、一部は非推奨またはサポート対象外のものです）

- 145個のHTML要素
- 68個のSVG要素
- 31個のMathML要素

実際は、これらのうち214だけに遭遇しました。

- 137個のHTML要素
- 54個のSVG要素
- 23個のMathML要素

デスクトップのデータセットでは、検出された4,834個の非標準要素のデータを収集しました。次がそれに当たります。

- 155個（3%）は、非常に高い確率でマークアップまたはエスケープの例外として識別できます（解析されたタグ名にマークアップが破損していることを暗示する文字が含まれていました）
- 341個（7%）はXMLスタイルのコロン名前空間を使っています（ただし、HTMLとしてはXML名前空間は使っていません）
- 3,207個（66%）は有効なカスタム要素の名前です
- 1,211個（25%）はグローバルな名前空間にあります（非標準であり、ダッシュもコロンもありません）
    - うち、216個は2文字以上で、`<cript>`、`<spsn>`または`<artice>`などの標準要素名からレーベンシュタイン距離が1であるため、タイプミスの*可能性*としてフラグを立てました。ただし、これらの一部（`<jdiv>`など）には意図的なものも含まれています。

付け加えると、デスクトップページの15%とモバイルページの16%には、既に廃止された要素が含まれています。

注意：この結果は、それぞれのの作成者がマークアップを手動で作成しているのではなく、何らかの製品を使っている為と考えられます。

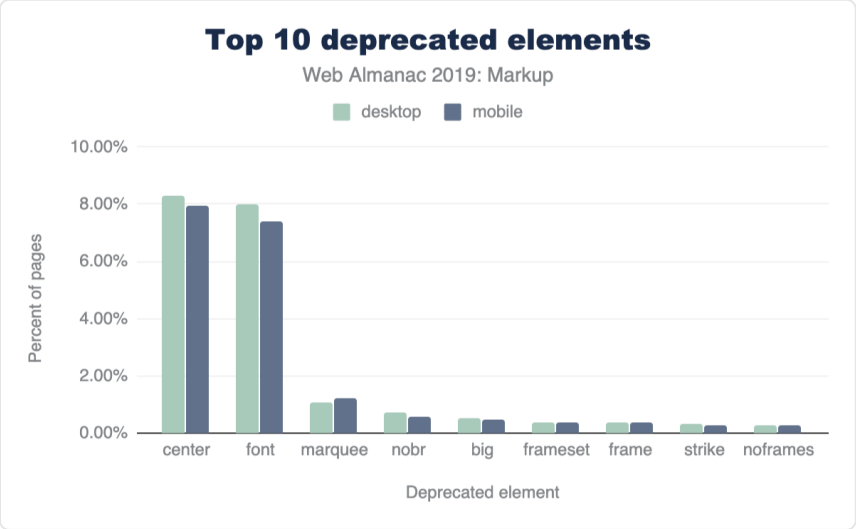


図3.6. 最も頻繁に使われている非推奨の要素。

上記の図3.6は、最も頻繁に使われている非推奨の要素トップ10を表しています。これらは非常に小さな数値に見えますが、この観点は重要です。

価値観と使用法

要素の使い方に関する数値（標準、非推奨、またはカスタム）を議論する為には、まず何らかの観点を確立する必要があります。

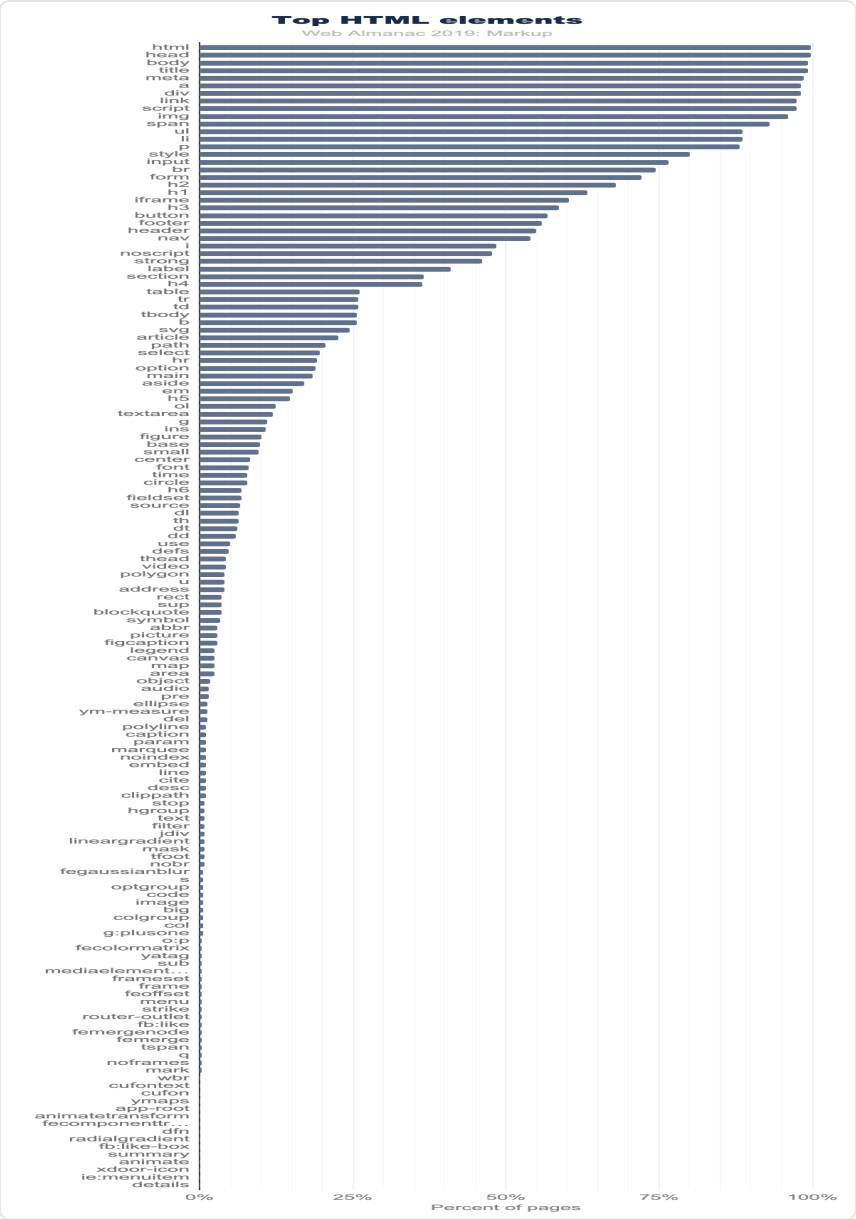


図3.7. トップ150の要素（詳細）。

上記の図3.7は、ページ中に現れたかどうかでカウントされた要素のトップ150を表示しています。利用率がどのように落ちていくかに着目してください。

ページの90%以上で使われている要素は11個しかありません。

- `<html>`
- `<head>`
- `<body>`
- `<title>`
- `<meta>`
- `<a>`
- `<div>`
- `<link>`
- `<script>`
- `<img>`
- `<span>`

上を除き、ページ中50%以上使われている要素は15個だけです。

- `<ul>`
- `<li>`
- `<p>`
- `<style>`
- `<input>`
- `<br>`
- `<form>`
- `<h2>`
- `<h1>`
- `<iframe>`
- `<h3>`
- `<button>`

- `<footer>`
- `<header>`
- `<nav>`

また、ページ中に5%以上使われている要素は40個のみでした。

`<video>` でさえ、ぎりぎりその範囲内に収まっていません。デスクトップデータセット内の4%という結果で現れています（モバイルでは3%）。この数字はとても低いように聞こえますが、実のところ4%はかなり人気だったりします。事実、ページ中1%以上の要素は98個しかありません。

これらの要素の分布を抑え、どの要素が1%以上使われているのかを見るのは興味深いことです。

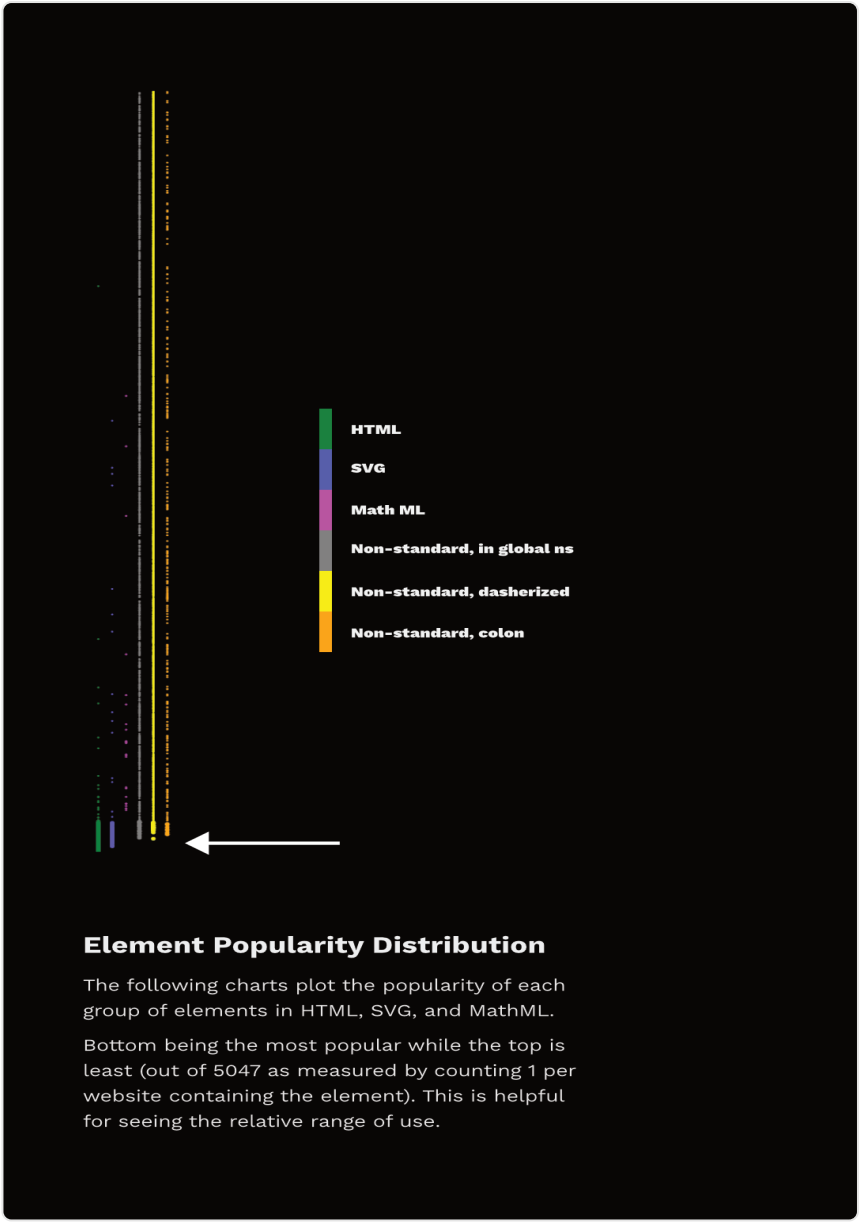


図3.8. 標準化によって人気になった要素の分類。

図3.8は、各要素の順位とそれらがどのカテゴリに属するかを示しています。データポイントを単純に見ることができるように、個別の塊へ分割しました（そうしなければ、全データ

を表現するために十分なピクセル領域がありませんでした）、これは人気を一つの「線」として表します。一番下が最も一般的で、上が一般的では無いものです。矢印は、ページの1%以上に表示される要素の終端を指しています。

ここでは2つのことを確認できます。まず、使用率が1%を超える要素の塊は、HTMLだけではありません。実際、*最も人気のある100個の要素のうち27個はHTMLではなく*SVGです！さらに、*その隙間または近くには非標準のタグもあります*！そして多くのHTML要素がページ毎に1%未満として現れている事に注意してください。

では、これらのページで1%の利用率となっている要素はすべて無駄ですか？、いいえ絶対にあえりえません。これが視点を確立することが重要な理由です。Webには約20億のWebサイトがあります。データセットのすべてのWebサイトの0.1%に何かが表示される時、これはWeb全体で*約200万のWebサイト*を表していると推定できます。0.01%でさえ*20万のサイト*を推定できます。これは、良い思想では無い古い要素であったとしても、めったに要素のサポートを打ち切らない理由でもあります。数十万または数百万のサイトを壊すことは、ブラウザベンダーが簡単にできることではありません。

ほとんどの要素は、ネイティブの物も含めてページの1%未満として現れていますが、それでも非常に重要であり成功しています。たとえば `<code>` は私が頻繁に使用する要素です。これは間違いなく便利で重要ですが、ページの0.57%でしか使われていません。この部分は私達の測定対象に基づいているため偏っています。通常、ホームページは特定の種類のもの（たとえば `<code>` など）が含まれる可能性は低いでしょう。例えば、ホームページでは見出し、段落、リンク、リスト以外はあまり一般的ではないでしょう。ただし、データには一般的に価値があります。

また、著者が定義した（ネイティブではない） `.shadowRoot` を含むページに関する情報も収集しました。デスクトップページの約0.22%とモバイルページの約0.15%にシャドウルートが確認できています。数が少ないように聞こえるかもしれませんが、これはモバイルデータセット内の約6.5kサイトとデスクトップ上の10kサイトであり、いくつかのHTML要素よりも多くなっています。たとえば、 `<summary>` はデスクトップ上で同レベルで利用されており、146番目に人気のある要素です。 `<datalist>` はホームページの0.04%に使われており、201番目に人気のある要素です。

実際、HTMLで定義されている要素の15%以上は、デスクトップデータセットのトップ200から圏外です。 `<meter>` は、HTMLがLiving Standardモデルに移行する前、2004-2011頃の最も人気のない「HTML5時代」の要素です。そしてこの要素の人気は1,000番目です。最近導入された要素（2016年4月）である `<slot>` の人気は1,400番目となっています。

大量データ：実際のWeb上の実際のDOM

データセット中のネイティブ/標準機能をどのように使っているかと言う観点を念頭に置い

て、非標準のものについて話しましょう。

測定したほとんどの要素は単一のWebページでのみ使用されると思われるかもしれませんが、実際には5,048個の要素すべてが複数のページに出現しています。データセット中、最も出現数が少ない要素は15ページに存在しています。そして、約5分の1は100ページ以上に存在します。約7%は1,000ページ以上に存在します。

データ分析を支援するためにGlitchで小さなツール[59]を共同で作りました。このツールはあなたも使うことができます。そして、あなたの観測した内容をパーマリンクと共に@HTTPArchiveへシェアしてください。（Tommy Hodginsは、同じように洞察に使えるCLIツール[60]を作成しています。）

それでは、いくつかのデータを見ていきましょう。

製品（およびライブラリ）とそのカスタムマークアップ

いくつかの標準でない要素の普及率については、ファーストパーティの採用をしたというより、人気のあるサードパーティのツールに含まれていることが関係しているでしょう。たとえば `<fb:like>` 要素は0.03%のページで見つかります。これはサイト所有者が意図的に記述しているのではなく、Facebookウィジェットに含まれているためです。Hixieが14年前に言及した[61]要素のほとんどは減少しているように見えますが、大部分が残っています。

- Claris Home Page[62]（最後の安定版は21年前）で作られた一般的要素は、100ページ以上にまだ現れてます。 たとえば、 `<x-claris-window>` は130ページに現れています。
- 英国のeコマースプロバイダーであるOxatis[63]の `<actinic:*>` 要素の一部はさらに多くのページに出現しています。たとえば、 `<actinic:basehref>` はデスクトップデータ中の154ページに出現しています。
- Macromediaの要素はほとんど消えたようです。一覧にはたった一つの要素 `<mm:endlock>` だけが現れており、その数はわずか22ページだけです。
- Adobe Go-Liveの `<csscriptdict>` は、デスクトップデータセットの640ページに引き続いて現れています。
- Microsoft Officeの `<o:p>` 要素は、2万ページ以上のデスクトップページとなる0.5%に引き続いて現れています。

59. *https://rainy-periwinkle.glitch.me*
60. *https://github.com/tomhodgins/hade*
61. *https://web.archive.org/web/20060203031245/http://code.google.com/webstats/2005-12/editors.html*
62. *https://en.wikipedia.org/wiki/Claris_Home_Page*
63. *https://www.oxatis.co.uk*

そして、Hixieのオリジナルレポートにはなかった多くの新しい要素も現れました。

- `<ym-measure>` は、YandexのMetrica analytics package[64]によって挿入されるタグです。デスクトップとモバイルページの1%以上で使われており、最も利用されている要素トップ100でその地位を確立しています。すごい！
- 今は亡きGoogle Plusの `<g:plusone>` は、2万1千ページ以上で出現しています。
- Facebookの `<fb:like>` は、14,000のモバイルページで出現しています。
- 同様に、 `<fb:like-box>` は7.8kモバイルページで出現しています。
- `<app-root>` は、Angularなどのフレームワークで一般的に含まれており、8.2kモバイルページに出現しています。

これらを5%未満のネイティブHTML要素と比べてみましょう。

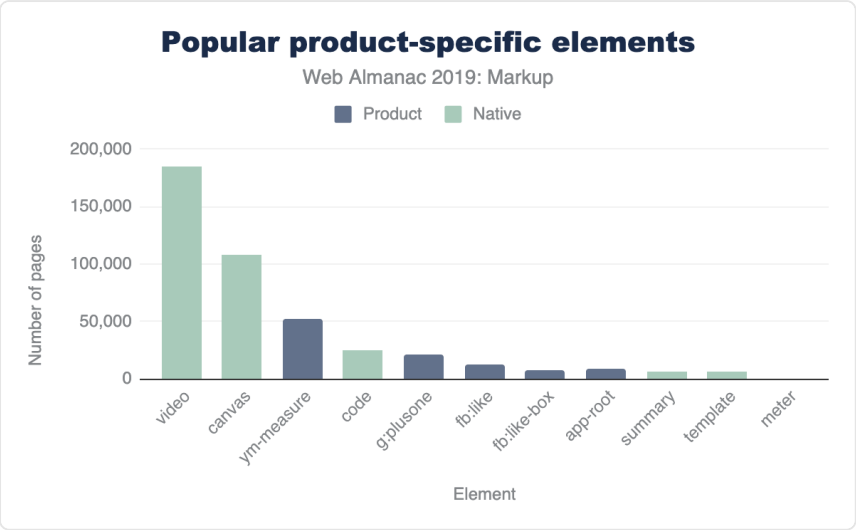


図3.9. 採用率が5%以下での、製品固有とネイティブで人気の要素。

このような興味深い洞察を一日中行うことができます。

これは少々違うものですが、人気のある要素には製品のエラーによって引き起こされる可能性もあります。たとえば1,000を超えるサイトで `<pclass="ddc-font-size-large">` が出現しています。しかしこれは、これは人気のある”as-a-service”製品がスペースを取り忘れているために発生していました。幸いなことに、このエラーは調査中に報告されて、すぐに

64. https://www.npmjs.com/package/yandex-metrica-watch

修正されました。

Hixieはオリジナルの論文で次のように述べています。

> *この非標準マークアップに対して楽天的でいられる間は少なくとも、これらの要素にはベンダープレフィックスを明確に利用しているため、これは良い考えだと言えます。これにより、標準化団体が新しく作る要素と属性が、それらと衝突する可能性を大幅に減らすことができます。*

ただし、上で述べた通りこれは一般的ではありません。記録できた非標準要素の25%以上は、グローバル名前空間の汚染を避けるために、いかなる名前空間戦略も使っていません。例えば、モバイルデータセットにある1157個の要素一覧[65]を表示します。見ての通り、これらの多くは曖昧な名前やつづりの間違など、問題がない可能性があります。しかし、少なくともこれはいくつかの挑むべき課題を示しています。例えば、`<toast>`（Google社員が`<std-toast>`として最近提案しようとした仕様[66]）がこの一覧に含まれています。

それほど難しくない一般的な要素もいくつかあります。

- Yahoo Mapsの`<ymaps>`は、～12.5kのモバイルページに出現します。
- 2008年のフォント置換ライブラリである`<cufon>`と`<cufontext>`は、～10.5kモバイルページに出現しています。
- `<jdiv>`要素は、Jivo chatの製品によって挿入されており、～40.3kモバイルページに出現しています。

前回のチャートに今回のデータを配置すると、次のようになります（改めて、データセットに基づいて少しだけ変わっています）

65. *https://rainy-periwinkle.glitch.me/permalink/53567ec94b328de965eb821010b8b5935b0e0ba316e833267dc04f1fb3b53bd5.html*
66. *https://www.chromestatus.com/feature/5674896879255552*

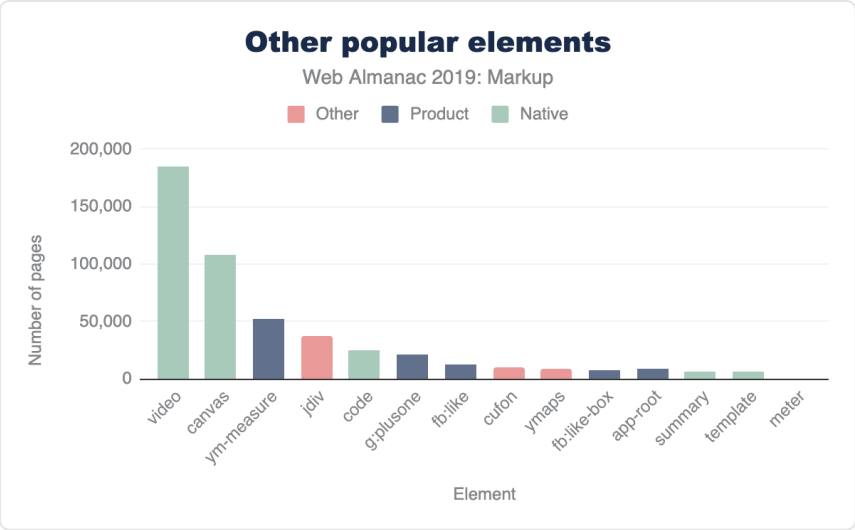


図3.10. 採用率が5％以下での、製品固有のコンテキストまたはネイティブで人気のあるその他の要素。

この結果には興味深い点があります、それは一つのツールが他の便利になる手段も提供していると言うことです。データ空間を調べることに興味がある場合に、具体的なタグ名は想定される尺度の一つでしかありません。良い「俗語」の発展を見つけることができれば、それは間違いなく最強の指標でしょう。しかし、それが私たちの興味の範囲外である場合はどうなりますか？

一般的なユースケースとソリューション

たとえば、一般的なユースケースの解決に興味が人々の場合はどうでしょうか？これは、現在抱えているユースケースに対応したソリューションを探している場合や、標準化を促進するために人々が解決しようとしている一般的なユースケースをさらに研究するようなものがあります。一般的な例として「タブ」を取り上げます。長年にわたって、タブのようなものに対して多くの要求がありました。あいまいな検索をしてみるとタブには多くのバリエーション[67]があることがわかります。同一のページに2つの要素が存在しているかを簡単に識別できないため、利用されているかどうかを数えるのは少し難しくなります。そのためこの計測条件は地味ですが、最大のカウントを持つものを単純に使用します。ほとんどの場合、実際のページ数はそれより大幅に増えるでしょう。

67. *https://rainy-periwinkle.glitch.me/permalink/c6d39f24d61d811b55fc032806cade9f0be437dcb2f5735a4291adb04aa7a0ea.html*

また、数多くのアコーディオン[68]やダイアログ[69]、少なくとも65種類のカルーセル[70]、それとポップアップ[71]に関するもの、そして最低でも27種類存在するトグルとスイッチ[72]があります。

おそらくですが、非ネイティブである92種類のボタン要素[73]が必要な理由を調査することで、ネイティブとのギャップを埋めようとすることができます。

人気の有るものがポップアップ（`<jdiv>`などのチャット）という事に気付く事ができれば、私達の知っている知識（たとえば、`<jdiv>`についての目的や`<olark>`）を知り、それらに取り組むために作り上げた43のこと[74]を見て、そこを辿りながら空間を調査することができます。

結論

なのでここには多くのデータがありますが、要約すると。

- ページ中の要素は、14年前よりも平均と最大の両方で増えています。
- ホームページ上に存在する要素の寿命は*非常に*長いでしょう。要素を非推奨やサポート停止にしたとしても、消えることはありません。
- 世の中には多くの壊れたマークアップがあります。（タグのつづり間違い、スペースの抜け、エスケープ間違い、誤解）
- 「有益」である事を測定するのは難しいでしょう。多くのネイティブ要素は5%の敷居、さらには1%敷居を超えることはありませんが、多くのカスタム要素は多くの理由でその敷居を越えます。少なくとも1%を超えたものには注目をするべきかもしれませんが、データを見る限りは割と*良い*成功を収めているようなので、おそらく0.5%とすべきでしょう。
- 既に大量のカスタムマークアップがあります。様々な形式がありますが、ダッシュを含む要素は確実に削除されたようです。
- 牛の通り道を舗装する(pave the cowpaths)ために、このデータをさらに調査して、適切な観測結果を出す必要があります。

最後はあなたの出番です。大規模なコミュニティの創造性と好奇心を利用し、いくつかのツール(https://rainy-periwinkle.glitch.me/など)を使うことでこのデータを探索することができ

68. https://rainy-periwinkle.glitch.me/permalink/e573cf279bf1d2f0f98a90f0d7e507ac8dbd3e570336b20c6befc9370146220b.html
69. https://rainy-periwinkle.glitch.me/permalink/0bb74b808e7850a441fc9b93b61abf053efc28f05e0a1bc2382937e3b78695d9.html
70. https://rainy-periwinkle.glitch.me/permalink/651e592cb2957c14cdb43d6610b6acf696272b2fbd0d58a74c283e5ad4c79a12.html
71. https://rainy-periwinkle.glitch.me/permalink/981967b19a9346ac466482c51b35c49fc1c1cc66177ede440ab3ee51a7912187.html
72. https://rainy-periwinkle.glitch.me/permalink/2e6827af7c9d2530cb3d2f39a3f904091c523c2ead14daccd4a41428f34da5e8.html
73. https://rainy-periwinkle.glitch.me/permalink/5ae67c941395ca3125e42909c2c3881e27cb49cfa9aaf1cf59471e3779435339.html
74. https://rainy-periwinkle.glitch.me/permalink/db8fc0e58d2d46d2e2a251ed13e3daab39eba864e46d14d69cc114ab5d684b00.html

ます。興味深い観察結果を共有して、知識と理解の共有の場を作ってください。

著者

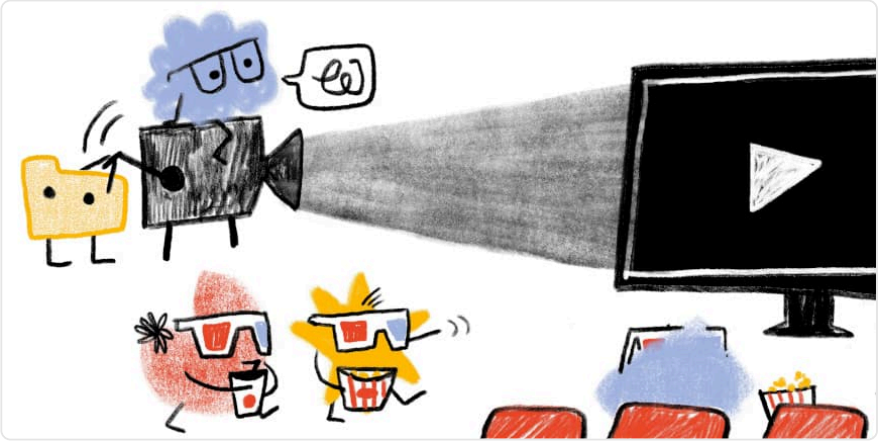
Brian Kardell

@briankardell @https://toot.cafe/@bkardell bkardell https://bkardell.com

Brian Kardellは、Igalia[75] の開発者提唱者、標準化貢献者、ブロガー[76]であり、現在は Open JS Foundation[77] の W3C 諮問委員会代表を務めています。彼は Extensible Web Community Groupの創設者であり、The Extensible Web Manifesto[78] の共著者でもあります。

75. *https://igalia.com*
76. *https://bkardell.com*
77. *https://openjsf.org/*
78. *https://extensiblewebmanifesto.org*

部 I 章 4
メディア



Colin Bendell と Doug Sillars によって書かれた。
Ahmad Awais と Eric Portis によってレビュ。
Doug Sillars 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

画像、アニメーション、動画はウェブ体験の重要な一部です。それらが重要な理由はたくさんあります。ストーリーを伝えたり、視聴者の関心を引きつけたり、他のウェブ技術では簡単には作れないような芸術的な表現を提供したりするのに役立ちます。これらのメディアリソースの重要性は、2つの方法で示すことができます。1つは、1ページのダウンロードに必要なバイト数の多さ、もう1つはメディアで描かれたピクセル数の多さです。

純粋なバイトの観点から見ると、HTTP Archiveは歴史的に報告されている[79]メディアから関連付けられたリソースバイトの平均3分の2を持っています。分布の観点から見ると、事実上すべてのウェブページが画像や動画に依存していることがわかります。10パーセンタイルでさえ、我々はバイトの44%がメディアからであり、ページの90パーセンタイルで総バイトの91%に上昇できることを参照してください。

79. *https://legacy.httparchive.org/interesting.php#bytesperpage*

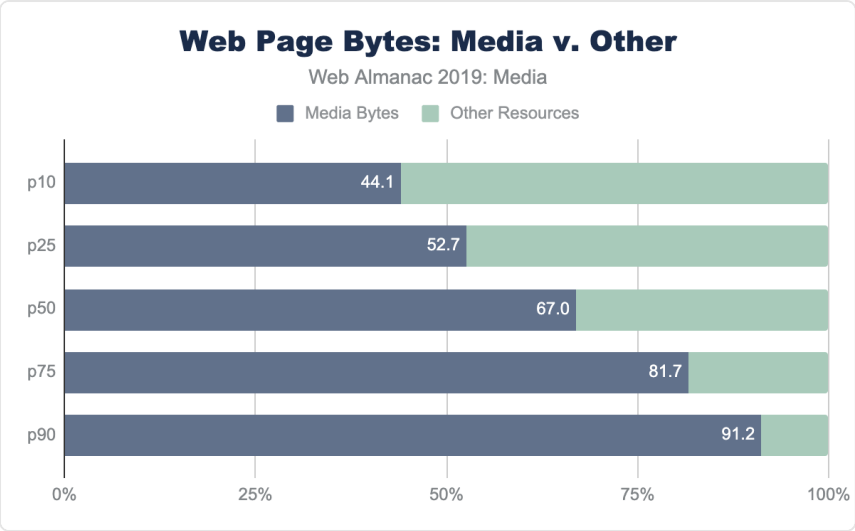


図4.1. Webページのバイト：画像と動画対その他。

メディアは視覚体験には欠かせないものですが、この大量のバイトのインパクトには2つの副作用があります。

まず、これらのバイトをダウンロードするために必要なネットワークのオーバーヘッドは大きく、携帯電話や低速ネットワーク環境（コーヒーショップやUberに乗っているときのテザリングのような）では劇的にページのパフォーマンスを遅くできます。画像はブラウザによる優先度の低いリクエストですが、ダウンロード中のCSSやJavaScriptを簡単にブロックできます。これ自体がページのレンダリングを遅らせることになります。しかし、画像コンテンツは、ページの準備ができたことをユーザーに視覚的に伝える手がかりとなります。そのため、画像コンテンツの転送が遅いと、ウェブページが遅いという印象を与えることがあります。

2つ目の影響は、ユーザーへの金銭的なコストです。これはウェブサイトの所有者の負担ではなく、エンドユーザーの負担となるため、しばしば無視されがちな側面です。逸話として、日本のような[80]という市場では、データの上限に達した月末近くは学生の購買意欲が低下し、ユーザーはビジュアルコンテンツを見ることができなくなるということが伝えられています。

さらに、世界のさまざまな地域でこれらのウェブサイトを訪問するための金銭的コストは不釣り合いです。中央値と90パーセンタイルでは、画像のバイト数はそれぞれ1MBと1.9MBです。WhatDoesMySiteCost.com[81]を使用すると、マダガスカルのユーザーの一人当たりの国

80. https://x.com/yoavweiss/status/1195036487538003968?s=20
81. https://whatdoesmysitecost.com/#gniCost

民総所得（GNI）コストは90パーセンタイルでウェブページを1回読み込んだだけで、一日の総所得の2.6%になることがわかります。対照的に、ドイツでは、これは1日の総所得の0.3%になります。

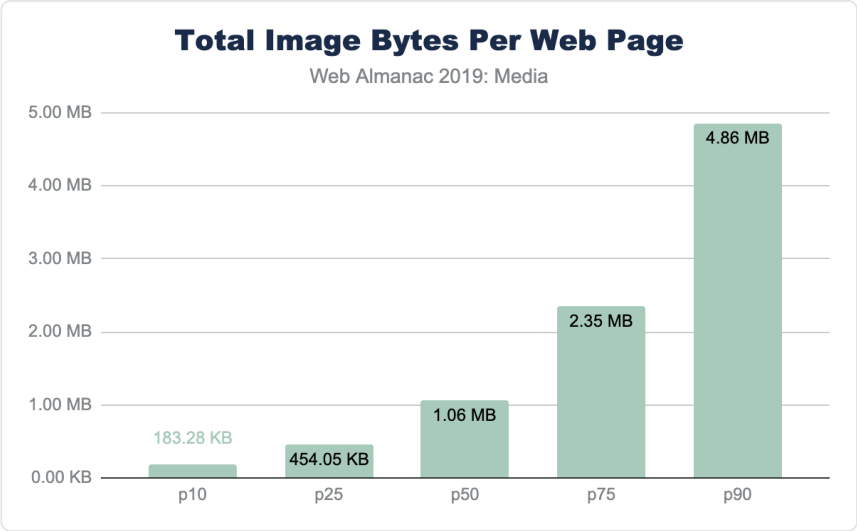


図4.2. ウェブページあたりの総画像バイト数（モバイル）。

ページあたりのバイト数を見ると、ページパフォーマンスとユーザーに対するコストだけを見ることになりますが、それは利点を見落としています。これらのバイトは、画面上のピクセルをレンダリングするために重要です。このように、1ページあたりに使用されるメディアのピクセル数を見ることで、画像や動画リソースの重要性を見ることができます。

ピクセル量を見るときに考慮すべき3つのメトリクスがあります。CSSピクセル、ナチュラルピクセル、スクリーンピクセルです。

- *CSSピクセルボリューム* はCSSの観点からのレイアウトです。この尺度は、画像や動画を引き伸ばしたり、押し込んだりできる境界ボックスに焦点を当てています。また、実際のファイルのピクセルや画面表示のピクセルは考慮されていません。
- *ナチュラルピクセル* とは、 ファイル内で表現される論理的なピクセルのことを指します。この画像をGIMPやPhotoshopで読み込んだ場合、ピクセルファイルの寸法は自然なピクセルとなります。
- *スクリーンピクセル* とは、ディスプレイ上の物理的な電子機器を指します。携帯電話や最新の高解像度ディスプレイが登場する以前は、CSSピクセルとスクリーン上のLEDポイントの間には1:1の関係がありました。しかし、モバイルデバイス

は目に近づけられ、ノートPCの画面は昔のメインフレーム端末よりも近づけられているため、現代のスクリーンは従来のCSSピクセルに対する物理ピクセルの比率が高くなっています。この比率は、Device-Pixel-Ratio、または口語でRetina™ディスプレイと呼ばれています。

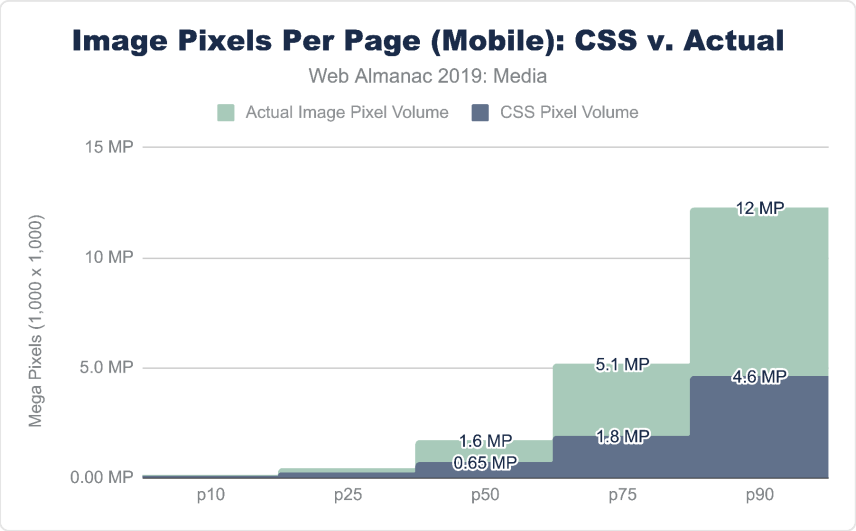


図4.3. 1ページあたりのピクセル画像（モバイル）。CSS対実物。

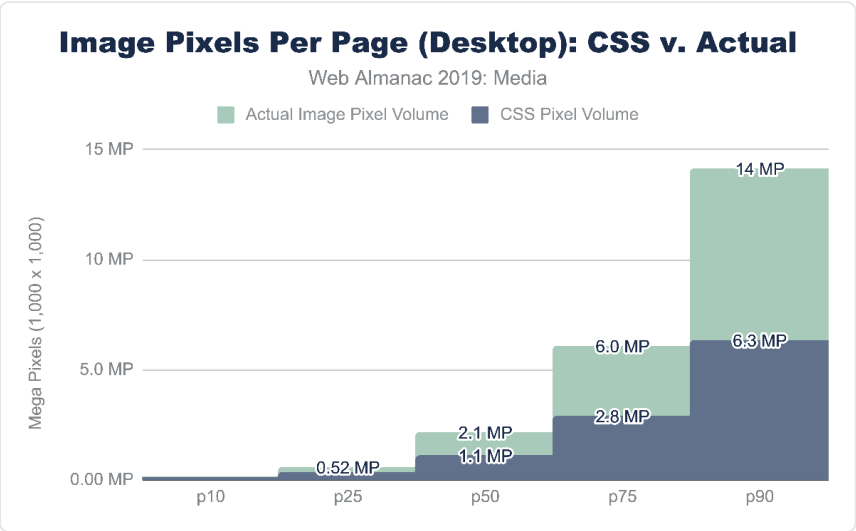


図4.4. 1ページあたりのピクセル画像（デスクトップ）。CSS対実物。

CSSピクセルと自然ピクセル量を見ると、中央値のウェブサイトは1メガピクセル (MP) のメディア コンテンツを表示するレイアウトになっていることがわかります。90パーセンタイルでは、CSSレイアウトのピクセル量はモバイルで4.6MP、デスクトップで6.3MPに増加しています。これはレスポンシブレイアウトが、異なる可能性が高いだけでなく、フォームファクターが異なることも興味深い。要するに、モバイルレイアウトはデスクトップに比べてメディアに割り当てられるスペースが少ないということです。

対照的に、ナチュラル（ファイル）ピクセル量はレイアウト量の2〜2.6倍です。デスクトップウェブページの中央値は2.1MPのピクセルコンテンツを送信し、1.1MPのレイアウトスペースに表示されます。モバイルでは、90パーセンタイルの割合で12MPが4.6MPに圧縮されていることがわかります。

もちろん、モバイルデバイスのフォームファクターはデスクトップとは異なります。デスクトップが大きく主に横向きで使用されるのに対し、モバイルデバイスは小さく通常縦向きで使用されます。前述したように、モバイルデバイスは目から近い位置にあるため、一般的にデバイスピクセル比（DPR）が高く、タイムズスクエアのビルボードに必要なピクセル数と比べて1インチあたりのピクセル数が多く必要となります。これらの違いにより、レイアウトの変更を余儀なくされ、モバイルのユーザーはコンテンツの全体を消費するためにサイトをスクロールするのが一般的です。

メガピクセルは、主に抽象的な指標であるため、難しい指標です。ウェブページで使用されているピクセルの量を表現するのに便利な方法は、ディスプレイサイズに対する比率として表現することです。

ウェブページのクロールで使用したモバイル端末では、`512 x 360`の表示で、0.18MPのCSSコンテンツが表示されています（物理的な画面である`3x`や3^2以上の画素である1.7MPと混同しないように）。このビューアーのピクセル量を画像に割り当てられたCSSピクセルの数で割ると、相対的なピクセル量が得られます。

もし、画面全体を完璧に埋め尽くす画像が1枚あったとしたら、これは1xピクセルの塗りつぶし率になります。もちろん、ウェブサイトが1枚の画像でキャンバス全体を埋め尽くすことはほとんどありません。メディアコンテンツは、デザインや他のコンテンツと混在する傾向があります。1xよりも大きい値はレイアウトが追加の画像コンテンツを見るため、ユーザーが、スクロールする必要があることを意味します。

注：これは、DPRとレイアウトコンテンツのボリュームの両方のCSSレイアウトを見ているだけです。レスポンシブ画像の効果や、DPRの高いコンテンツを提供することの効果を評価しているわけではありません。

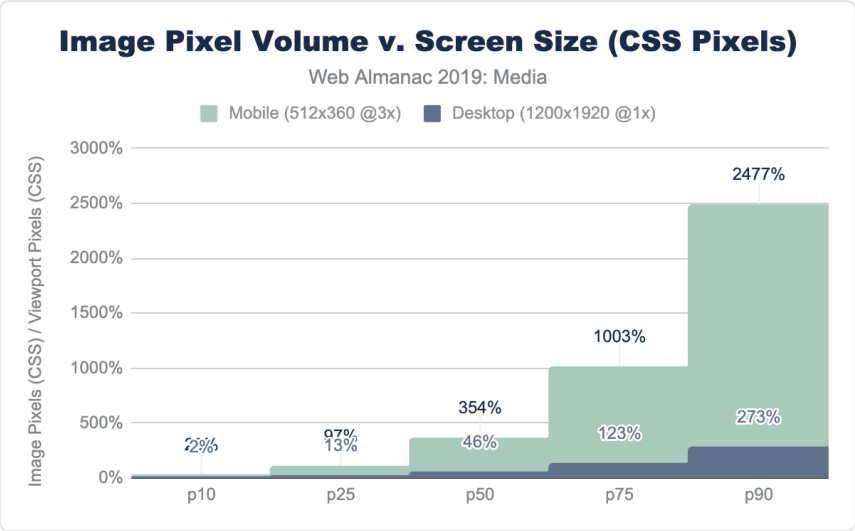


図4.5. 画像のピクセル量と画面サイズ（CSSピクセル）の関係。

デスクトップの中央値のウェブページでは、画像や動画を含むレイアウトが表示されるのはディスプレイの46%に過ぎません。対照的に、モバイルでは、メディアピクセルの量が実際のビューポートサイズの3.5倍を埋めています。レイアウトは、1つの画面で埋められる以上のコンテンツがあり、ユーザーはスクロールする必要があります。最低でも、1サイトあたり3.5ページ分のコンテンツがスクロールしていることになります（飽和度100%を想定）。モバイルの90パーセンタイルでは、これはビューポートサイズの25倍にまで大幅に拡大します！

メディアリソースは、ユーザーエクスペリエンスにとって非常に重要です。

画像

バイトの削減とユーザー体験の最適化に役立つ画像の管理と最適化については、すでに多くのことが書かれています。ブランド体験を定義するのはクリエイティブなメディアであるため、多くの人にとって重要かつクリティカルなトピックとなっています。したがって画像や動画コンテンツの最適化は、意図した体験の忠実性を維持しながら、ネットワーク上で転送されるバイト数を減らすのに役立つベストプラクティスを適用することとの間のバランスをとる行為です。

画像、動画、アニメーションに利用されている戦略は、大まかに似ていますが、具体的なアプローチは大きく異なります。一般的には、これらの戦略は次のようなものです。

- **ファイル形式** - 最適なファイル形式を利用
- **レスポンシブ** - レスポンシブ画像技術を適用して、画面に表示されるピクセルだけを転送する
- **遅延ローディング** - 人が見たときだけコンテンツを転送する
- **アクセシビリティ** - 全人類に一貫した体験を提供する

これらの結果を解釈する際には注意が必要です。Web Almanacのためにクロールされたウェブページは、Chromeブラウザでクロールされました。これは、SafariやFirefoxに適したコンテンツネゴシエーションが、このデータセットでは表現されていない可能性があることを意味しています。例えば、JPEG2000、JPEG-XR、HEVC、HEICなどのファイル形式は、Chromeではネイティブにサポートされていないため、使用されていません。これは、ウェブにこれらの他の形式や経験が含まれていないことを意味するものではありません。同様に、Chromeには遅延読み込みのネイティブサポートがあります（v76 以降）が、他のブラウザではまだ利用できません。これらの注意事項については、方法論をご覧ください。

画像を利用していないウェブページを見つけることは稀です。長年にわたり、ウェブ上でコンテンツを表示するためのさまざまなファイルフォーマットが登場してきましたが、それぞれが異なる問題に対処してきました。主に4つの普遍的な画像フォーマットがあります。JPEG、PNG、GIF、およびSVGです。さらに、Chromeではメディア パイプラインが強化され、5つ目の画像フォーマットのサポートが追加されました。WebP。他のブラウザでも同様にJPEG2000（Safari）、JPEG-XL（IEとEdge）、HEIC（SafariではWebViewのみ）のサポートが追加されています。

それぞれのフォーマットにはそれぞれメリットがあり、Web上での理想的な使い方があります。とても簡単にまとめると以下のようになります。

フォーマット	ハイライト	欠点
JPEG	• ユビキタスに対応 • 写真コンテンツに最適	• 常に品質の損失がある • ほとんどのデコーダは、最新のカメラからの高ビット深度の写真を扱うことができません（チャンネルあたり8ビット以上） • 透明性のサポートはありません
PNG	• JPEGやGIFのように、幅広いサポートを共有しています • ロスレスです • 透明度、アニメーション、高ビット深度をサポート	• JPEGに比べてはるかに大きなファイル • 写真コンテンツには理想的ではない
GIF	• PNGの前身Gは、アニメーションで最もよく知られています。 • ロスレス	• 256色の制限があるため、変換による視覚的な損失が常にあります • アニメーション用の非常に大きなファイル
SVG	• ファイルサイズを増やさずにリサイズできるベクターベースのフォーマット • ピクセルではなく数学に基づいており、滑らかな線を作成します	• 写真やその他のラスターコンテンツには有用ではない
WebP	• PNGのようなロスレス画像とJPEGのようなロスレス画像を生成することができる新しいファイル形式です • JPEGと比較して平均30%のファイル削減率を誇っています[62]が、他のデータでは、ファイル削減率の中央値はピクセル量に基づいて10-28%の間であることが示	• JPEGとは異なり、一部の画像がぼやけて見えるクロマサブサンプリングに限定されます • 普遍的にサポートされているわけではありません。Chrome、Firefox、Androidのエコシステムのみ • ブラウザのバージョンに応じて

図4.6. 主流のファイル形式の説明。

画像フォーマット

すべてのページを見てみると、これらのフォーマットの普及率が、高いことがわかります。ウェブ上でもっとも古いフォーマットの1つであるJPEGは画像リクエストの60%、全画像バイトの65%で圧倒的に、もっとも一般的に使用されている画像フォーマットです。興味深いことに、PNGは画像要求とバイト数の28%で2番目によく使われている画像フォーマットです。色の正確さやクリエイティブなコンテンツの精度に加えて、サポートがどこにでもあることが広く使われている理由と考えられます。対照的に、SVG、GIF、WebPは4%とほぼ同じ使用率です。

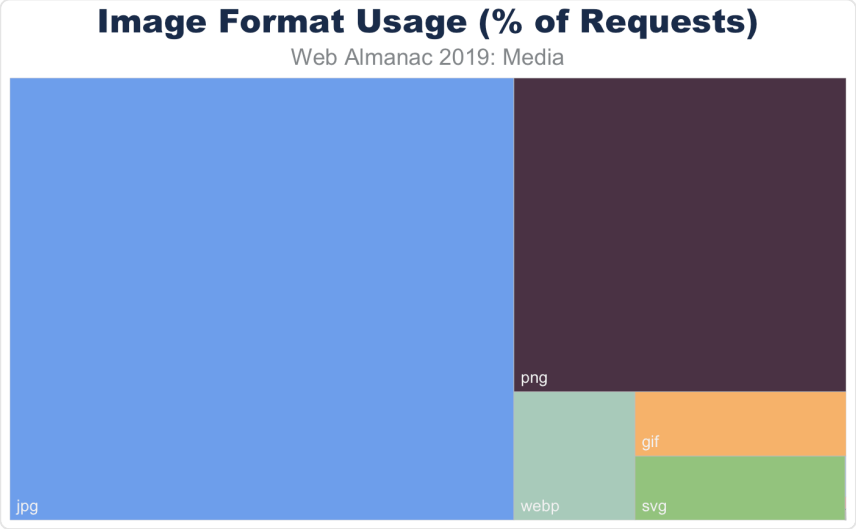


図4.7. 画像フォーマットの使用法

もちろん、ウェブページの画像コンテンツの使い方は一様でありません。画像に依存しているページもあれば、いくつかは他よりも画像に依存しています。`google.com`のホームページを見てみると、一般的なニュースサイトに比べて画像はほとんどないことがわかります。実際、中央値のウェブサイトには13枚の画像があり、90パーセンタイルでは61枚、99パーセンタイルでは229枚の画像があります。

82. https://developers.google.com/speed/webp/faq
83. https://cloudinary.com/state-of-visual-media-report/

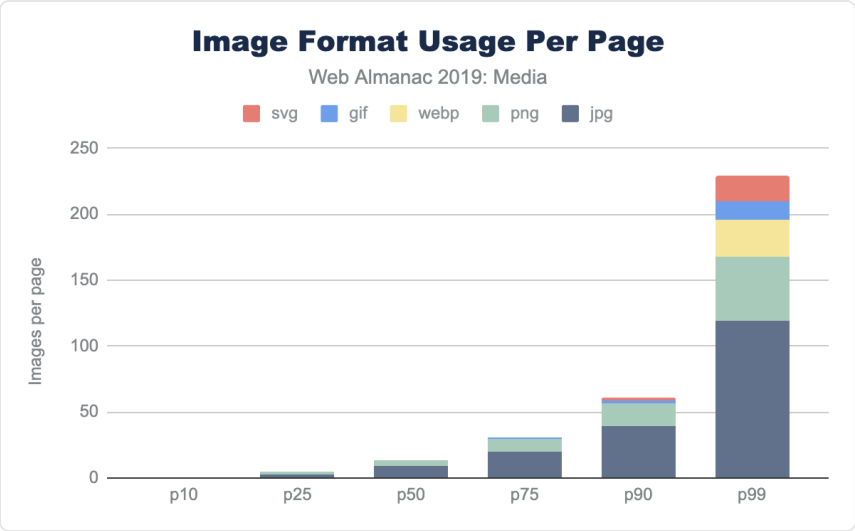


図4.8. 1ページあたりの画像フォーマットの使用量

中央値のページではJPEGが9枚、PNGが4枚となっており、GIFが使用されているのは上位25%のページのみで、採用率は報告されていません。1ページあたりの各フォーマットの使用頻度は、より近代的なフォーマットの採用についての洞察を提供していません。具体的には、各フォーマットに少なくとも1枚の画像が含まれているページの割合は？

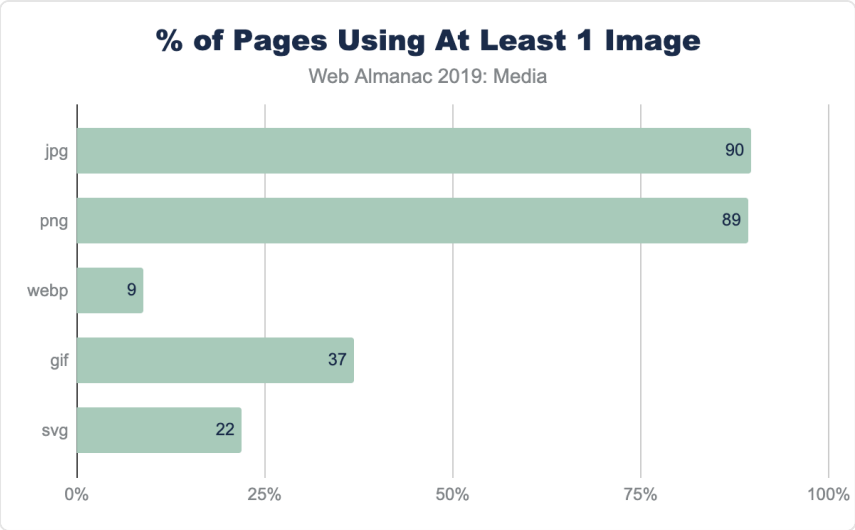


図4.9. 少なくとも1枚の画像を使用しているページの割合。

これは、90パーセンタイルのページでさえWebPの頻度がゼロである理由を説明するのに役立ちます。WebPがイメージに適していない理由はたくさんありますが、メディアのベストプラクティスの採用は、WebP自体の採用のようにまだ初期段階にとどまっています。

画像ファイルのサイズ

画像ファイルのサイズを見るには、リソースあたりの絶対バイト数とピクセルあたりのバイト数の2つの方法があります。

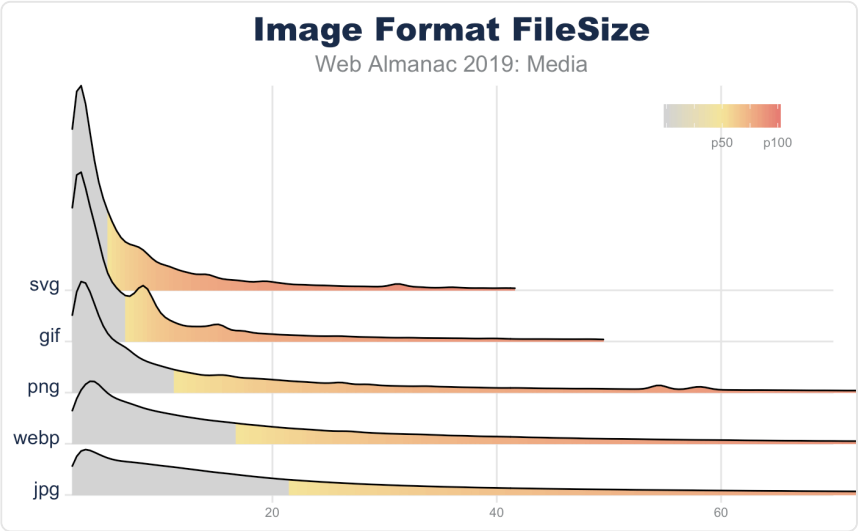


図4.10. 画像形式別のファイルサイズ（KB）。

このことから、ウェブ上の典型的なリソースの大きさや小ささを知ることができます。しかし、これではこれらのファイル分布の画面上で表現されているピクセルの量を知ることはできません。これを行うには、各リソースのバイト数を画像の自然なピクセル数で割ることができます。1ピクセルあたりのバイト数が低いほど、視覚コンテンツの伝送効率が高いことを示しています。

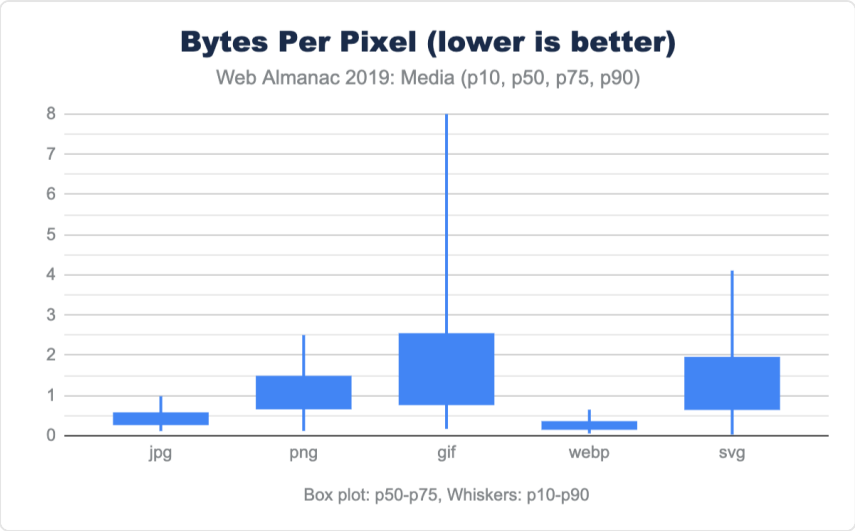


図4.11. ピクセルあたりのバイト数。

以前はGIFファイルがJPEGよりも小さいと思われていましたが、今ではJPEGのリソースが大きくなった原因はピクセルボリュームにあることがはっきりとわかります。おそらく、GIFが他のフォーマットと比較して非常に低いピクセル密度を示していることは驚きではありません。さらにPNGは高いビット深度を扱うことができ、クロマサブサンプリングのぼやけに悩まされることはありませんが、同じピクセルボリュームではJPGやWebPの約2倍のサイズになります。

なお、SVGに使用されるピクセル量は、画面上のDOM要素のサイズ（CSSピクセル）です。ファイルサイズの割にはかなり小さいですが、これは一般的にSVGがレイアウトの小さい部分で使用されていることを示唆しています。これが、PNGよりも1ピクセルあたりのバイト数が悪く見える理由です。

繰り返しになりますが、この画素密度の比較は、同等の画像を比較しているわけではありません。むしろ、典型的なユーザー体験を報告しているのです。次に説明するように、これらの各フォーマットでも、ピクセルあたりのバイト数をさらに最適化して減らすために使用できる技術があります。

画像フォーマットの最適化

体験に最適なフォーマットを選択することは、フォーマットの能力のバランスをとり、総バイト数を減らすことです。ウェブページの場合、画像を最適化することでウェブパフォーマンスを向上させることが1つの目標です。しかし、それぞれのフォーマットには、バイト数を減らすのに役立つ追加機能があります。

いくつかの機能は、総合的な体験に影響を与えることができます。たとえばJPEGやWebPでは、*量子化*（一般的には*品質レベル*と呼ばれる）や *クロマサブサンプリング*を利用でき、視覚的な体験に影響を与えることなく、画像に格納されているビット数を減らすことができます。音楽用のMP3のように、この技術は人間の目のバグに依存しており、カラーデータが失われるにもかかわらず同じ体験を可能にします。しかし、すべての画像がこれらの技術に適しているわけではありません。

他のフォーマット機能は、単にコンテンツを整理するだけで、時には文脈に沿った知識を必要とします。たとえばJPEGのプログレッシブエンコーディングを適用すると、ピクセルはスキャンレイヤーに再編成されブラウザはより早くレイアウトを完成させることができ、同時にピクセル量を減らすことができます。

1つのLighthouseテストは、ベースラインとプログレッシブにエンコードされたJPEGをA/Bで比較するものです。これは画像全体がロスレス技術でさらに最適化されるか、また異なる品質レベルを使用するなど、潜在的には不可逆技術で最適化されるかどうかを示すための気付きを提供しています。

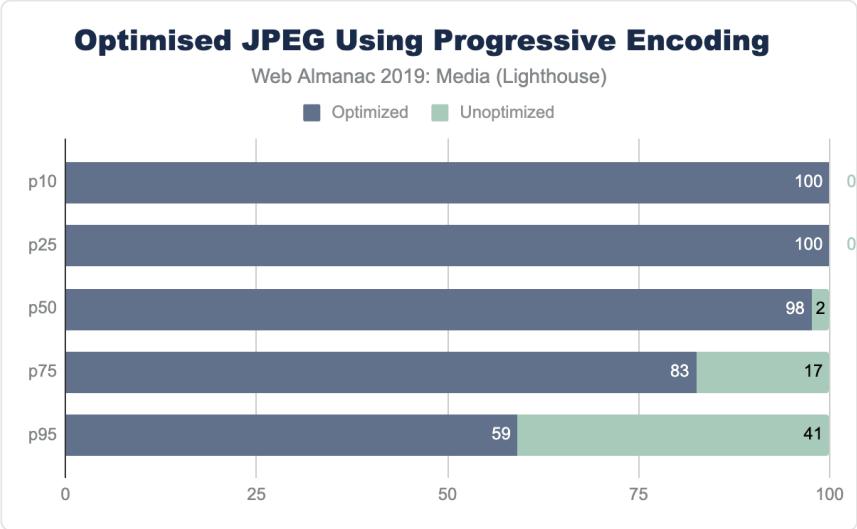


図4.12. 「最適化された」画像の割合。

このAB Lighthouseテストでの節約は、p95で数MBに達することができる潜在的なバイトの節約だけでなく、ページパフォーマンスの向上を実証しています。

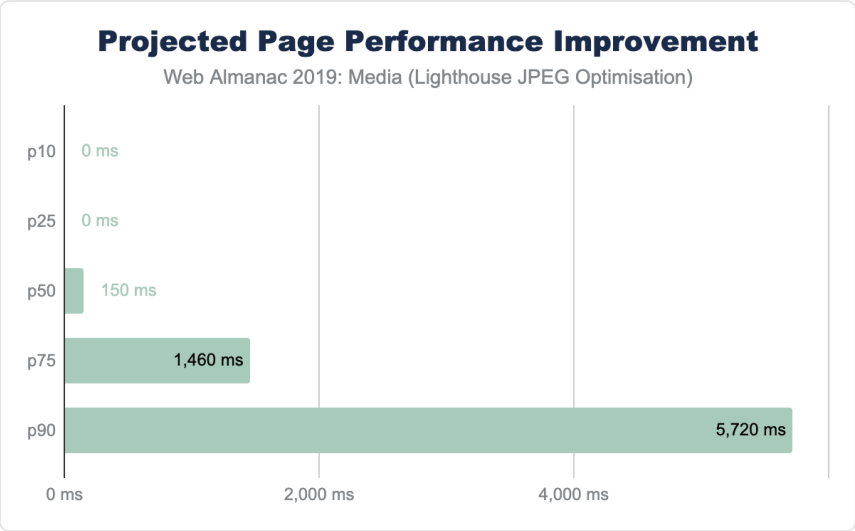


図4.13. Lighthouseからの画像最適化によるページパフォーマンスの向上を予測。

レスポンシブ画像

ページパフォーマンスを向上させるもう1つの軸として、レスポンシブ画像の適用があります。これは、画像の縮小によってディスプレイに表示されない余分なピクセルを減らすことで、画像のバイト数を減らすことに重点を置いた手法です。この章の最初の方でデスクトップの中央のウェブページでは、1MPの画像プレースホルダーが使用されているにもかかわらず、実際のピクセル量は2.1MP転送されていることを見ました。これは1xDPRテストだったので、1.1MPのピクセルがネットワーク経由で転送されましたが、表示されませんでした。このオーバーヘッドを減らすために、2つの技術のうちの1つを使用できます（3つの可能性もあります）。

- **HTMLマークアップ** - `srcset` 要素と `sizes` 要素を組み合わせて使用することで、ブラウザはビューポートの寸法とディスプレイの密度に基づいて最適な画像を選択できます。
- **クライアントヒント** - これは、リサイズ可能な画像の選択をHTTPコンテントネゴシエーションに移行させます。
- **特典**: JavaScriptライブラリーは、JavaScriptがブラウザーDOMを実行して検査し、コンテナーに基づいて正しい画像を挿入できるようになるまで画像のロードを遅らせます。

HTMLマークアップの使用

レスポンシブ画像を実装するもっとも一般的な方法は、`<img srcset>` または `<source srcset>` のいずれかを用いて代替画像のリストを作成することです。`srcset` がDPRに基づいている場合、ブラウザは追加情報なしでリストから正しい画像を選択できます。しかし、ほとんどの実装では、`srcset` のピクセルサイズに基づいて正しい画像を選択するため必要なレイアウト計算の方法をブラウザへ指示するため `<img sizes>` を利用しています。

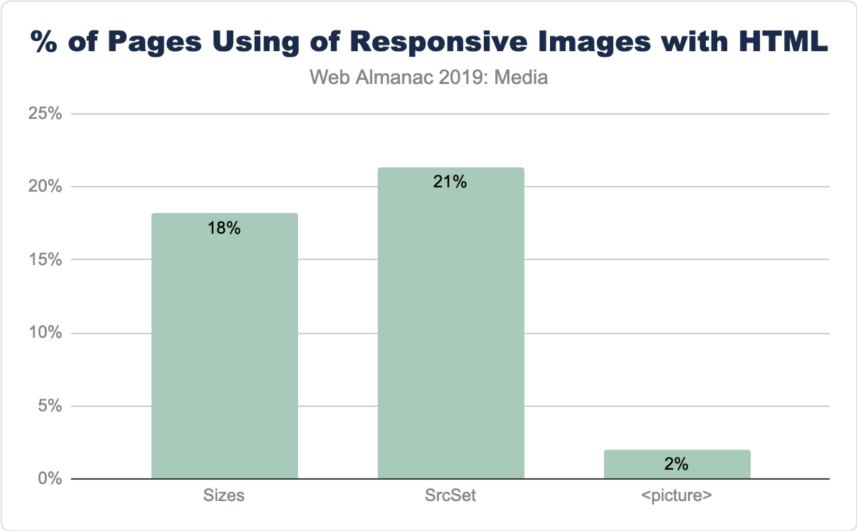


図4.14. HTMLでレスポンシブ画像を使用しているページの割合。

アートディレクション[84]のような高度なレスポンシブウェブデザイン（RWD）レイアウトによく使われていることを考えると、`<picture>` の使用率が著しく低いのは驚くべきことでありません。

サイズの使用

`srcset` の有用性は、通常は `sizes` メディアクエリの精度に依存します。`sizes` がないと、ブラウザは `<img>` タグが小さいコンポーネントではなくビューポート全体を埋め尽くすと仮定します。興味深いことに、ウェブ開発者が `<img sizes>` に採用している共通のパターンは5つあります。

- **`<img sizes="100vw">`** - これは画像がビューポートの幅を埋め尽くすことを示

84. https://developer.mozilla.org/docs/Learn/HTML/Multimedia_and_embedding/Responsive_images#Art_direction

します（デフォルトでもあります）。

- `<img sizes="200px">` - これは、DPRに基づいてブラウザを選択する際に便利です。
- `<img sizes="(max-width: 300px) 100vw, 300px">` - これは2番目に人気のあるデザインパターンです。これはWordPressとおそらく他のいくつかのプラットフォームで自動生成されるものです。元の画像サイズ（この場合は300px）に基づいて自動生成されているように見えます。
- `<img sizes="(max-width: 767px) 89vw, (max-width: 1000px) 54vw, ...">` - このパターンは、CSSレスポンシブレイアウトに合わせてカスタムビルドしたデザインパターンです。ブレークポイントごとに使用するサイズの計算が異なります。

<img sizes>	*頻度（百万）*	*%*
(max-width: 300px) 100vw, 300px	*1.47*	*5%*
(max-width: 150px) 100vw, 150px	*0.63*	*2%*
(max-width: 100px) 100vw, 100px	*0.37*	*1%*
(max-width: 400px) 100vw, 400px	*0.32*	*1%*
(max-width: 80px) 100vw, 80px	*0.28*	*1%*

図4.15. 最も人気のある `sizes` パターンを使用しているページの割合。

- `<img sizes="auto">` - これはもっともよく使われている使い方ですが、実際には非標準であり、`lazy_sizes` JavaScriptライブラリの使用によるものです。これはクライアント側のコードを使って、ブラウザのためにより良い `sizes` の計算を注入します。これの欠点は、JavaScriptの読み込みとDOMの準備が完全に整っているかどうかに依存し、画像の読み込みが大幅に遅れることです。

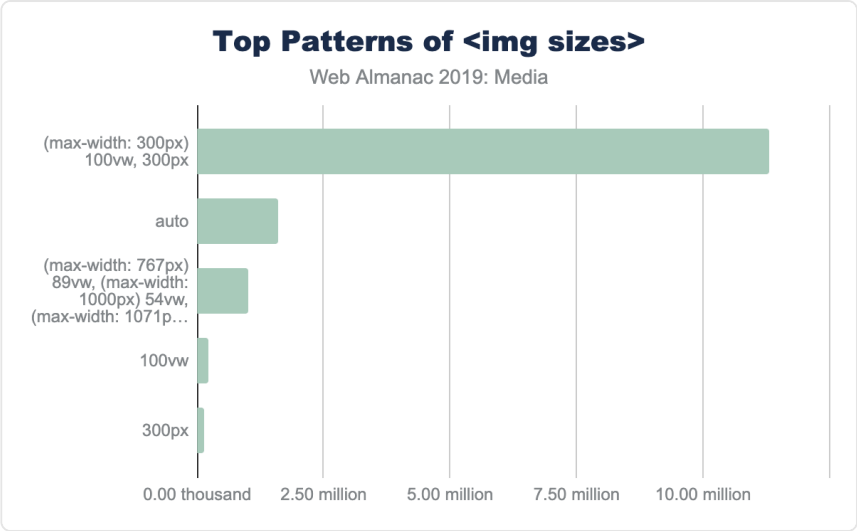


図4.16. `<img sizes>` のトップパターン。

クライアントヒント

クライアントヒント[85]は、コンテンツ制作者が画像のリサイズをHTTPコンテンツネゴシエーションに移すことを可能にします。この方法では、HTMLはマークアップを乱雑にするための追加の `<img srcset>` を必要とせず、代わりにサーバや最適な画像を選択するための画像CDN[86]に依存できます。これによりHTMLの簡素化が可能になり、オリジンサーバが時間の経過とともに適応し、コンテンツ層とプレゼンテーション層を切り離すことが可能になります。

クライアントヒントを有効にするには、ウェブページでブラウザに追加のHTTPヘッダー `Accept-CH: DPR, Width, Viewport-Width` を使ってシグナルを送る必要があります。*または*HTML `<meta http-equiv="Accept-CH" content="DPR, Width, Viewport-Width">` を追加します。どちらか一方の手法の利便性は実装するチームに依存し、どちらも利便性のために提供されています。

85. *https://web.dev/user-agent-client-hints/*
86. *https://cloudinary.com/blog/client_hints_and_responsive_images_what_changed_in_chrome_67*

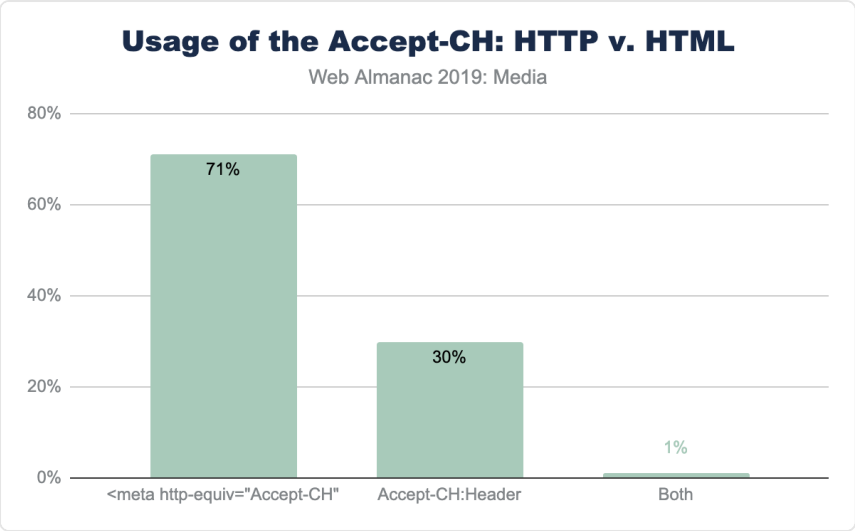


図4.17. `Accept-CH` ヘッダーと同等の `<meta>` タグの使用法。

HTMLでクライアントヒントを呼び出すために `<meta>` タグを使うのは、HTTPヘッダーに比べてはるかに一般的です。これは、ミドルボックスにHTTPヘッダーを追加するよりも、マークアップテンプレートを変更する方が便利であることを反映していると思われます。しかし、HTTPヘッダーの利用状況を見ると、50%以上が単一のSaaSプラットフォーム（Mercado）からのものです。

呼び出されたクライアントヒントのうち、大部分のページでは `DPR`, `ViewportWidth`, `Width` の3つのユースケースで使用されている。もちろん、 `Width` のクライアントヒントでは、ブラウザがレイアウトに関する十分なコンテキストを持つために `<img sizes>` を使用する必要があります。

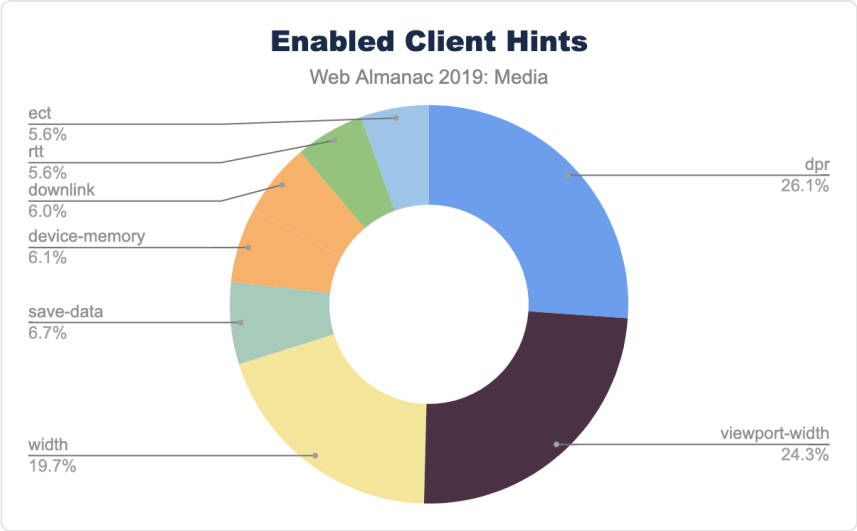


図4.18. 有効化されたクライアントヒント。

ネットワーク関連のクライアントヒント`downlink`、`rtt`、`ect`はAndroid Chromeでのみ利用可能です。

遅延ローディング

ウェブページのパフォーマンスを改善することは、部分的にはイリュージョンのゲームとして特徴付けることができます。このように遅延読み込み画像は、ユーザーがページをスクロールしたときにのみ画像やメディアコンテンツが読み込まれる、これらのイリュージョンの1つです。これにより、遅いネットワークでも知覚パフォーマンスが向上し、ユーザーが他の方法で表示されていないバイトをダウンロードする手間が省けます。

以前、図4.5で、75パーセンタイルの画像コンテンツの量が、理論的には単一のデスクトップやモバイルのビューポートで表示できる量をはるかに超えていることを示しました。オフスクリーン画像[87]Lighthouseの監査は、この疑念を裏付けています。ウェブページの中央値では、折り目の下に27%の画像コンテンツがあります。これは、90パーセンタイルの割合で84%に増加しています。

87. *https://developers.google.com/web/tools/lighthouse/audits/offscreen-images*

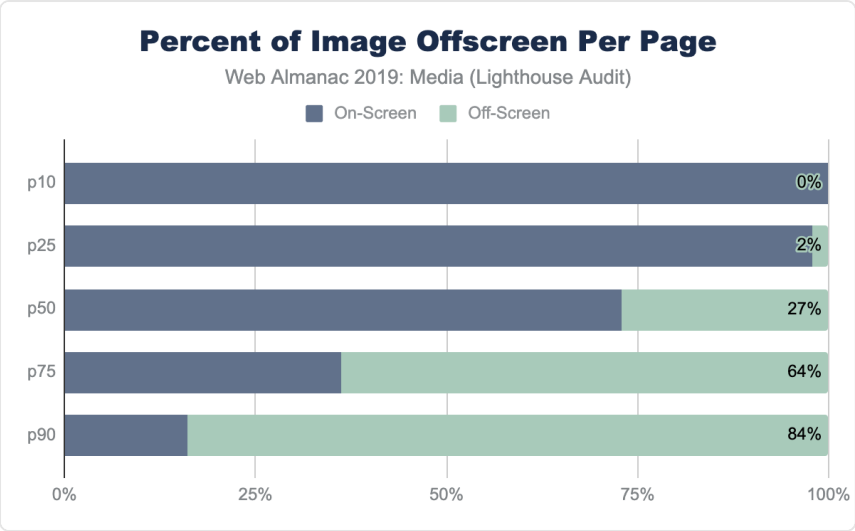


図4.19. Lighthouse監査：オフスクリーン。

Lighthouseの監査では、質の高いプレースホルダーを使用するなど、油断できない状況がいくつもあるため、臭いを嗅ぎ分けてくれます。

遅延ローディングはIntersection Observers、Resize Observersの組み合わせを含め実装可能[88]です、またはlazySizes[89]、lozad[90]などのJavaScriptライブラリの使用などさまざまな方法で実装できます。

2019年8月、Chrome76では`<img>`を使用したマークアップベースの遅延ローディングのサポートが開始されました。2019年のWeb Almanacに使用されたウェブサイトのスナップショットは2019年7月のデータを使用していますが、2,509以上のウェブサイトがすでにこの機能を利用していました。

アクセシビリティ

画像アクセシビリティの中心にあるのは`alt`タグです。画像に`alt`タグが追加されると、このテキストは画像を見ることができない（障害のある、インターネット接続が悪いのいずれかの理由で）ユーザーに画像を説明するために使用されます。

データセットのHTMLファイルに含まれるすべての画像タグを検出できます。デスクトップでは1,300万個、モバイルでは1,500万個の画像タグのうち、91.6%の画像に`alt`タグが存在

88. https://developers.google.com/web/fundamentals/performance/lazy-loading-guidance/images-and-video
89. https://github.com/aFarkas/lazysizes
90. https://github.com/ApoorvSaxena/lozad.js

しています。一見すると、ウェブ上では画像のアクセシビリティは非常に良好な状態にあるように見えます。しかし、よく調べてみると、見通しはあまり良くありません。データセットに存在する`alt`タグの長さを調べると、`alt`タグの長さの中央値は6文字であることがわかります。これは空の`alt`タグ（`alt=""`のように見える）に対応する。6文字以上の長さの`alt`テキストを使用している画像は全体の39%にすぎない。実際の」`alt`テキストの中央値は31文字で、そのうち25文字が実際に画像を説明しています。

動画

ウェブページで提供されるメディアは画像が主流ですが、ウェブ上でのコンテンツ配信では動画が大きな役割を果たし始めています。HTTP Archiveによると、デスクトップサイトの4.06%、モバイルサイトの2.99%が動画ファイルをセルフホスティングしていることがわかります。つまり、動画ファイルはYouTubeやFacebookのようなウェブサイトがホストしているわけではないということです。

動画フォーマット

動画は、多くの異なるフォーマットやプレイヤーで配信できます。モバイルおよびデスクトップ向けの主要なフォーマットは、`.ts`（HLSストリーミングのセグメント）と`.mp4`(H264 MPEG) です。

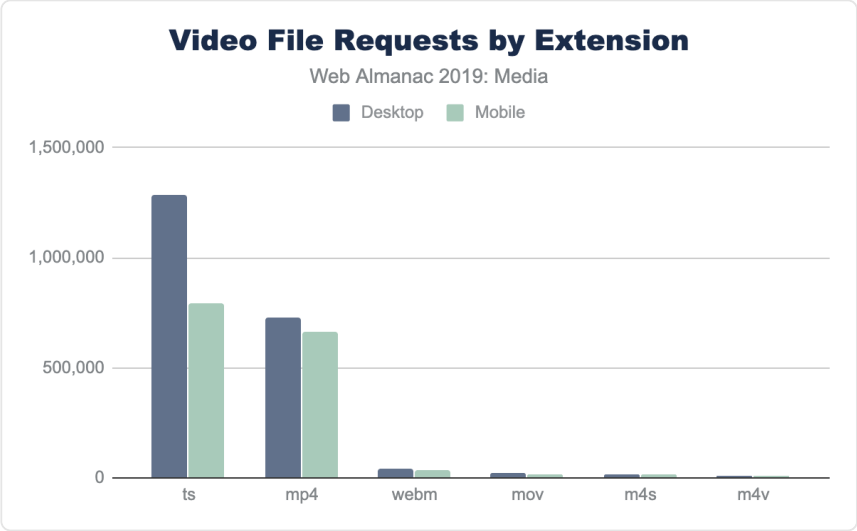


図4.20. 拡張子別の動画ファイルの数

他にも、`webm`、`mov`、`m4s`、`m4v`（MPEG-DASHストリーミングセグメント）などのフォーマットが見られます。ウェブ上のストリーミングの大部分はHLSであり、静的動画の主要なフォーマットは`mp4`であることが明らかです。

各フォーマットの動画サイズの中央値は以下の通りです。

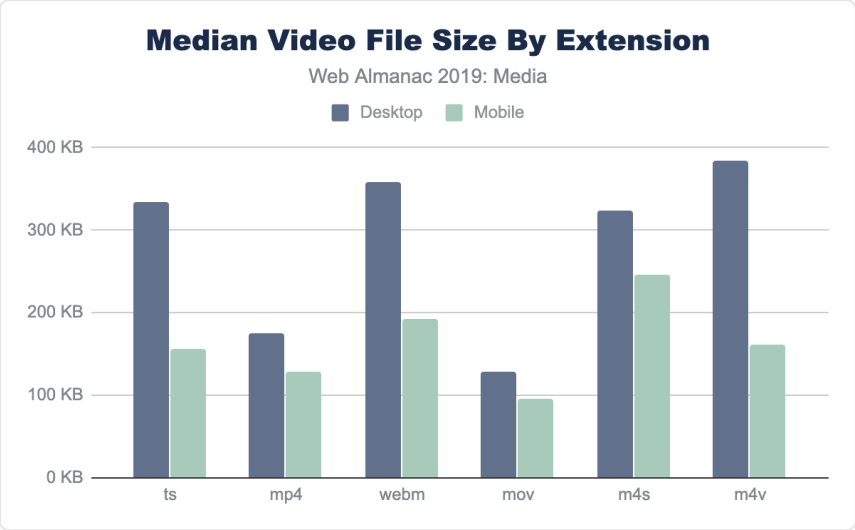


図4.21. 動画の拡張子別のファイルサイズの中央値。

中央値はモバイルの方が小さくなっていますが、これはおそらくデスクトップで非常に大きな動画を持っているサイトがモバイル用へ無効化していたり、動画ストリームが小さい画面に小さいバージョンの動画を提供していたりすることを意味していると思われます。

動画ファイルのサイズ

ウェブ上で動画を配信する場合、ほとんどの動画はHTML5動画プレイヤーで配信されます。HTML動画プレイヤーは、さまざまな目的で動画を配信するために非常にカスタマイズが可能です。たとえば、動画を自動再生するには、パラメーター`autoplay`と`muted`を追加します。`controls`属性は、ユーザーが動画を開始/停止したり、スキャンしたりすることを可能にします。HTTP Archiveの動画タグを解析することで、これらの属性の使用状況を確認できます。

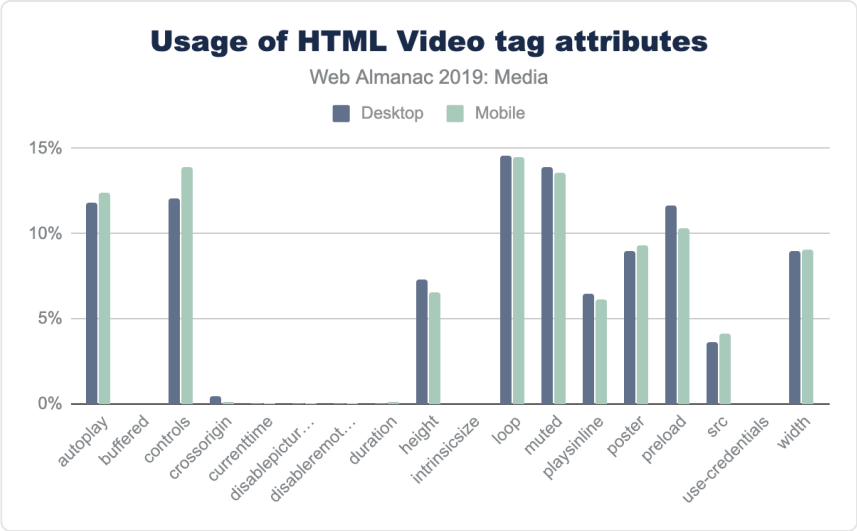


図4.22. HTML動画タグ属性の使用法。

もっとも一般的な属性は`autoplay`、`muted`、`loop`で、続いて`preload`タグ、そして`width`と`height`です。`loop`属性の使用は背景の動画や、動画をアニメーションGIFの代わりに、使用する場合に使用されるのでウェブサイトのホームページでよく使用されても不思議ではありません。

ほとんどの属性はデスクトップとモバイルで似たような使い方をしていますが、いくつかの属性には大きな違いがあります。モバイルとデスクトップの間でもっとも大きな違いがあるのは`width`と`height`の2つの属性で、モバイルではこれらの属性を使用しているサイトが4%少なくなっています。興味深いことに、モバイルでは`poster`属性（再生前に動画ウィンドウの上に画像を配置する）が少しだけ増加しています。

アクセシビリティの観点からは、`<track>`タグはキャプションや字幕を追加するために使用できます。HTTP Archiveには`<track>`タグの使用頻度に関するデータがありますが、調査したところ、データセットのほとんどのインスタンスはコメントアウトされているか、`404`エラーを返すアセットを指していました。多くのサイトでは、JavaScriptやHTMLのボイラプレートを使用しており、trackが使用されていない場合でもtrackを削除しないようになっているようです。

動画プレイヤー

より高度な再生（および動画ストリームの再生）を行うには、HTML5ネイティブ動画プレイヤーは動作しません。再生に使用する一般的な動画ライブラリがいくつかあります。

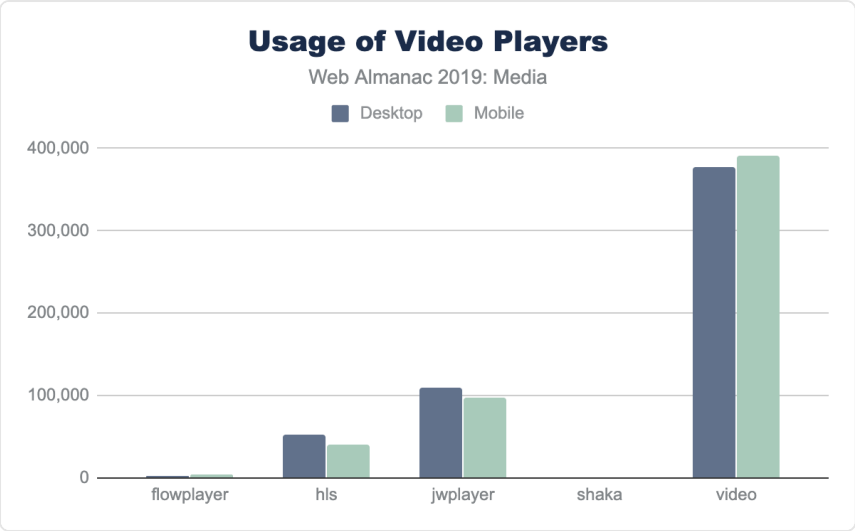


図4.23. トップの JavaScript 動画プレイヤー

もっとも人気があるのは（圧倒的に）video.jsで、JWPLayerとHLS.jsがそれに続いています。著者は、「video.js」という名前のファイルが、同じ動画再生ライブラリではない可能性があることを認めています。

結論

ほぼすべてのウェブページではユーザー体験を向上させ、意味を生み出すために、画像や動画をある程度使用しています。これらのメディアファイルは大量のリソースを利用し、ウェブサイトのトン数の大部分を占めています（そして、それらがなくなることはありません！）代替フォーマット、遅延ロード、レスポンシブ画像、画像の最適化を利用することは、ウェブ上のメディアのサイズを小さくするために長い道のりを行くことができます。

著者

Colin Bendell

@colinbendell colinbendell

Colinは、Cloudinary[91]のCTOオフィスの一員であり、オライリーの本High Performance Images[92]の共著者でもあります。彼は、大容量データ、メディア、ブラウザ、標準の交差点で多くの時間を過ごしています。@colinbendell や https://bendell.ca のブログで彼を見つけることができます。

Doug Sillars

@dougsillars dougsillars https://dougsillars.com

Doug Sillarsはフリーランスのデジタルノマドで、パフォーマンスとメディアの交差点で活動しています。彼は@dougsillarsをツイートし、dougsillars.com[93]で定期的にブログを更新しています。

91. https://cloudinary.com/
92. https://www.oreilly.com/library/view/high-performance-images/9781491925799/
93. https://dougsillars.com

部 I 章 5
サードパーティ

Patrick Hulce によって書かれた。
Simon Pieters、 David Fox と Vamsee Jasti によってレビュ。
Patrick Hulce による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

導入

オープンWebは広大で、リンク可能で、設計により相互運用可能です。他の誰かの複雑なライブラリを取得し、単一の `<link>` または `<script>` 要素を使用してサイトで使用する機能により開発者の生産性が大幅に向上し、素晴らしく新しいWeb体験を実現しました。反対に、一部のサードパーティプロバイダーが非常に人気があるため、パフォーマンス、プライバシー、およびセキュリティに関する重要な懸念が生じています。この章では、2019年のWebに対するサードパーティコードの普及と影響、サードパーティソリューションの人気につながる使用パターンと、Web体験の将来への影響の可能性について検討します。

定義

“サードパーティ”

サードパーティとは、サイトとユーザーの主要な関係の外にあるエンティティです。つまり、サイトの側面はサイトの所有者の直接の管理下になく、承認を得て存在します。たとえば、Googleアナリティクススクリプトは、一般的なサードパーティリソースの例です。

サードパーティのリソースは次のとおりです。

- 共有および公開オリジンでホストされています
- さまざまなサイトで広く使用されています
- 個々のサイト所有者の影響を受けない

これらの目標を可能な限り正確に一致させるために、この章で使用されるサードパーティリソースの正式な定義は、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です。

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たファーストパーティのコンテンツは、サードパーティのコンテンツとしてカウントされます。たとえば、サードパーティ・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・コンテンツとみなされます。

プロバイダーカテゴリー

この章では、サードパーティプロバイダをこれらの大まかなカテゴリのいずれかに分類しています。以下に簡単な説明を記載し、ドメインとカテゴリのマッピングについては、サードパーティ・ウェブ・リポジトリ[94]を参照してください。

- **Ad** - 広告の表示と測定
- **Analytics** - トラッキングサイト訪問者の行動
- **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提供

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

- **Content** - 公開を容易にし、シンジゲートされたコンテンツを提供
- **Customer Success** - サポートと顧客関係管理の機能
- **Hosting** - ユーザー任意のコンテンツを提供
- **Marketing** - 営業、見込み客獲得、メールマーケティング機能
- **Social** - ソーシャルネットワークとその連携
- **Tag Manager** - 他の第三者の包含を管理することを唯一の役割を提供
- **Utility** - 開発を補助するためのコード
- **Video** - ユーザーの任意の動画コンテンツ
- **Other** - 未分類、上記に分類されないカテゴリ

***CDNについての注意事項**: 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。
- 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。

データ

93.59%

図5.1. 少なくとも1つのサードパーティ製リソースを含むデスクトップページの割合。

サードパーティのコードは至る所にあります。ページの93%が少なくとも1つのサードパーティリソースを含み、ページの76%がアナリティクスドメインへのリクエストを発行しています。中央のページでは、ネットワークアクティビティ全体の35%を占める少なくとも9つのユニークサードパーティドメインからコンテンツをリクエストしており、最もアクティブ

な10%のページでは175以上のサードパーティリクエストを発行しています。サードパーティはウェブの不可欠な部分であると言っても過言ではありません。

55.63%

図5.2. 少なくとも1つの広告リソースを含むデスクトップページの割合。

カテゴリー

サードパーティ製コンテンツの普及が驚くに値しないとすれば、おそらくもっと興味深いのは、サードパーティ製コンテンツのプロバイダタイプ別の内訳です。

ウェブ上でのサードパーティの存在は広告が最もユーザーの目につきやすい例かもしれませんが、アナリティクスプロバイダーが最も一般的なサードパーティのカテゴリーであり、76%のサイトで少なくとも1つのアナリティクスリクエストが含まれています。CDNが63%、広告が57%、Sentry、Stripe、Google Maps SDKなどの開発者向けユーティリティが56%で、最も多くのウェブプロパティに表示されているのは僅差の2位、3位、4位と続いています。これらのカテゴリの人気は、本章で後述するウェブ利用パターンの基礎を形成しています。

プロバイダー

プロバイダーの比較的小さなセットがサードパーティの状況を支配しています：トップ100ドメインは、ウェブ全体のネットワーク要求の30%を占めています。Google、Facebook、YouTubeのような大企業は、それぞれのシェアの完全なパーセンテージポイントでここの見出しを作るが、WixやShopifyのような小さな事業体は同様にサードパーティの人気のかなりの部分を指揮します。

個々のプロバイダの人気とパフォーマンスへの影響については、多くのことが言えるかもしれませんが、このより意見の多い分析は読者やサードパーティ製Web[95]のような他の目的のために構築されたツールの練習として残されています。

95. *https://thirdpartyweb.today*

ランク	サードパーティドメイン	リクエストの割合
1	fonts.gstatic.com	2.53%
2	www.facebook.com	2.38%
3	www.google-analytics.com	1.71%
4	www.google.com	1.17%
5	fonts.googleapis.com	1.05%
6	www.youtube.com	0.99%
7	connect.facebook.net	0.97%
8	googleads.g.doubleclick.net	0.93%
9	cdn.shopify.com	0.76%
10	maps.googleapis.com	0.75%

図5.3. サードパーティドメインの人気トップ10

ランク	サードパーティURL	リクエストの割合
1	https://www.google-analytics.com/analytics.js	0.64%
2	https://connect.facebook.net/en_US/fbevents.js	0.20%
3	https://connect.facebook.net/signals/plugins/inferredEvents.js?v=2.8.51	0.19%
4	https://staticxx.facebook.com/connect/xd_arbiter.php?version=44	0.16%
5	https://fonts.gstatic.com/s/opensans/v16/mem8YaGs126MiZpBA-UFVZ0b.woff2	0.13%
6	https://www.googletagservices.com/activeview/js/current/osd.js?cb=%2Fr20100101	0.12%
7	https://fonts.gstatic.com/s/roboto/v18/KFOmCnqEu92Fr1Mu4mxK.woff2	0.11%
8	https://googleads.g.doubleclick.net/pagead/id	0.11%
9	https://fonts.gstatic.com/s/roboto/v19/KFOmCnqEu92Fr1Mu4mxK.woff2	0.10%
10	https://www.googleadservices.com/pagead/conversion_async.js	0.10%

図5.4. サードパーティからのリクエストが多いトップ10

リソースの種類

サードパーティコンテンツのリソースタイプの内訳を見ると、サードパーティのコードがWeb全体でどのように使用されているかを知ることができます。ファーストパーティのリクエストが、56%の画像、23%のスクリプト、14%のCSS、4%のHTMLにすぎないのに対し、サードパーティのリクエストはスクリプトとHTMLの割合が高く32%のスクリプト、34%の画像、12%のHTML、6%のCSSとなっています。このことは、サードパーティのコードがデザインを支援するために使用される頻度が低く代わりにファーストパーティのコードよりもインタラクションを促進したり観察したりするために使用される頻度が高いことを示唆していますがパーティの状態別のリソースタイプの内訳を見ると、よりニュアンスのあ

るストーリーがわかります。CSSと画像がそれぞれ70%、64%と圧倒的にファーストパーティであるのに対し、フォントはほとんどがサードパーティのプロバイダによって提供されており、ファーストパーティのソースから提供されているのは28% にすぎません。この使用パターンの概念については、この章で後ほど詳しく説明します。

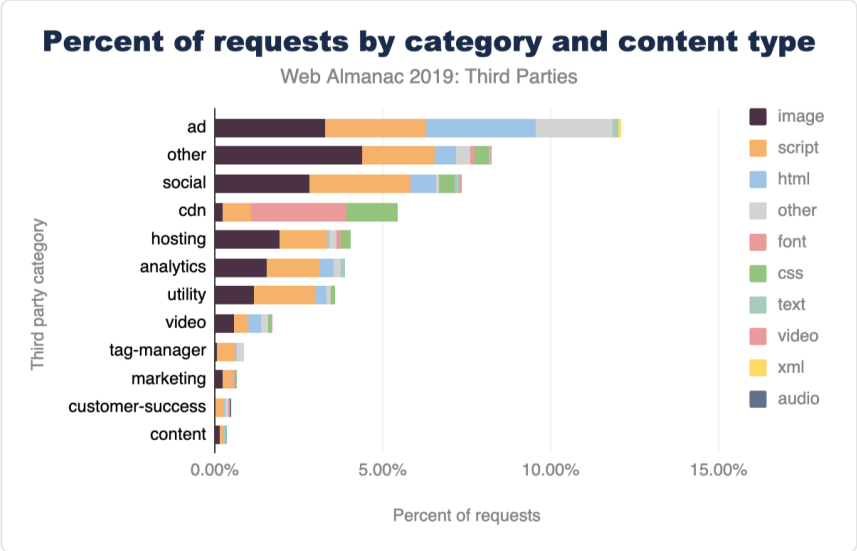


図5.5. カテゴリーとコンテンツタイプ別の第三者リクエストの割合。

このデータからは、他にもいくつかの興味深い事実が飛び出してきます。トラッキングピクセル（解析ドメインへの画像リクエスト）は全ネットワークリクエストの1.6%を占め、FacebookやTwitterなどのソーシャルネットワークへの動画リクエストの数は、YouTubeやVimeoなどの専用動画プロバイダーの6倍にもなります（YouTubeのデフォルトの埋め込みはHTMLとプレビューサムネイルで構成されていて自動再生動画ではないためと思われます）。

リクエスト数

全リクエストの49%がサードパーティです。ファーストパーティは2019年にも、51%と、ウェブリソースの大部分を占めるという王座にあと一歩まで迫ることができます。全リクエストの半分弱がサードパーティ製であるにもかかわらず、まったく含まれていないページが少数であることを考えると最もアクティブなサードパーティ製ユーザーは、自分の公平なシェアよりもかなり多くのことをしているに違いありません。実際、75%、90%、99%の割合で、ほぼすべてのページがサードパーティのコンテンツで構成されていることがわかります。実際、WixやSquareSpaceのような分散型WYSIWYGプラットフォームに大きく依存しているサイトでは、ルートドキュメントが唯一のファーストパーティのリクエストになってい

るかもしれません。

各サードパーティプロバイダーが発行するリクエストの数も、カテゴリーによって大きく異なります。アナリティクスはウェブサイトで最も普及しているサードパーティのカテゴリーですが、サードパーティのネットワークリクエスト全体のわずか7%にすぎません。一方、広告は、サイト数が20%近く少ないにもかかわらず、サードパーティのネットワークリクエスト全体の25%を占めています。彼らの人気に比べてリソースへの影響が桁違いに大きいことは、残りのデータからも明らかにしていくテーマになるでしょう。

バイトの重さ

リクエストの49%がサードパーティ製のものであるのに対し、ウェブのバイト数でのシェアは28%とかなり低くなっています。複数のリソースタイプ別の内訳も同様です。サードパーティのフォントはフォント全体の72%を占めていますが、フォントのバイト数に占める割合は53%にすぎません。これらはすべて、サードパーティのプロバイダがレスポンスサイズを低く抑える責任ある管理人であることを示唆しているように見えます。

スクリプトの57%を提供しているにもかかわらず、サードパーティはスクリプトバイトの64%を占めています。つまり、サードパーティのスクリプトはファーストパーティのスクリプトよりも平均で大きくなっています。これは、次のいくつかのセクションで述べるパフォーマンスへの影響を示す早期警告の兆候です。

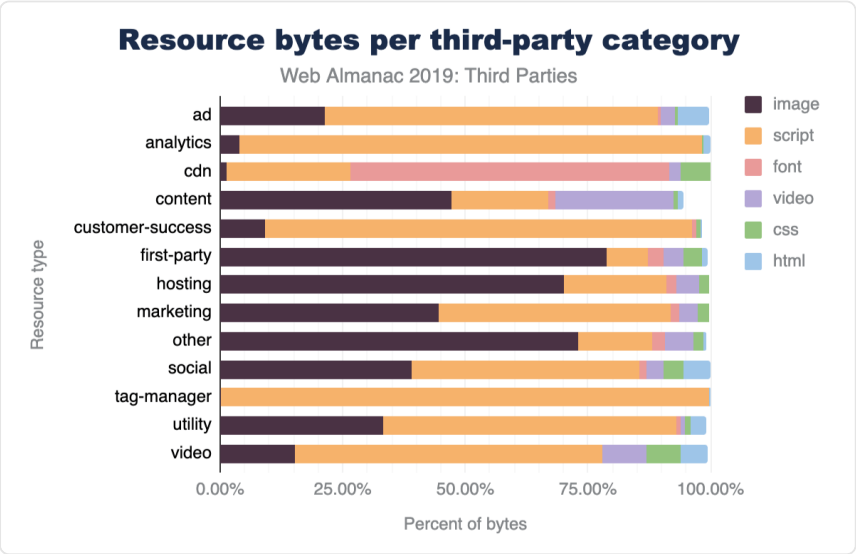


図5.6. サードパーティカテゴリ毎のリソースバイトの分布。

具体的なサードパーティプロバイダについては、リクエスト数リーダーボードの上位にいる大手プロバイダがバイト数でも登場しています。注目すべき動きは、YouTube、Shopify、Twitterのようなメディアを中心とした大手プロバイダがバイトインパクトチャートの上位にランクインしていることくらいです。

スクリプトの実行

スクリプトの実行時間の57%はサードパーティ製のスクリプトによるもので、トップ100のドメインはすでにウェブ上のスクリプト実行時間の48%を占めています。このことは、少数のエンティティがウェブのパフォーマンスに与える影響が実際にどれほど大きいかを明確に示しています。このトピックについては、反響 > パフォーマンスセクションで詳しく説明しています。

スクリプト実行の間のカテゴリの内訳は、主にリソース数の内訳に従っています。ここでも広告が最大の割合を占めています。広告スクリプトはサードパーティのスクリプト実行時間の25%を占めており、ホスティングとソーシャルプロバイダーは12%で2位と大きく引き離されています。

個々のプロバイダの人気とパフォーマンスの影響については、多くのことが言えるかもしれませんが、より意見の多い分析は読者のための演習として残されていますし先に述べた サードパーティウェブ[96] のような他の目的のために構築されたツールもあります。

使用パターン

サイトオーナーはなぜサードパーティのコードを使うのか？　サードパーティのコンテンツがネットワークリクエストの半分近くを占めるようになったのはなぜでしょうか？　これらのリクエストは何をしているのか？　これらの疑問に対する答えは、サードパーティのリソースの3つの主要な使用パターンにあります。大まかに言えば、サイト所有者はユーザーからデータを生成して消費し、サイト体験を収益化しWeb開発を簡素化するためにサードパーティを利用しています。

データの生成と消費

アナリティクスは、ウェブ上で最も人気のあるサードパーティのカテゴリですが、ユーザーの目に触れることはほとんどありません。ユーザーのコンテキスト、デバイス、ブラウザ、接続品質、ロケーション、ページのインタラクション、セッションの長さ、再訪問者のステータスなどが継続的に生成されています。このような大規模な時系列データをウェアハウスし、正規化し分析するツールを維持するのは困難で面倒で、コストがかかります。アナリテ

96. *https://thirdpartyweb.today*

ィクスがサードパーティプロバイダーの領域に入ることを明確に必要とするものはありませんが、ユーザーを理解することの魅力、問題空間の複雑さ、データを尊重し責任を持って管理することの重要性が増していることから、アナリティクスはサードパーティの人気のある使用パターンとして自然に表面化しています。

しかし、ユーザーデータには消費という裏返しの側面もあります。アナリティクスはサイトの訪問者からデータを生成することですが、他のサードパーティのリソースは、他の人しか知らない訪問者に関するデータを消費することに重点を置いています。ソーシャルプロバイダーは、この利用パターンにぴったりと当てはまります。サイト所有者は、訪問者のFacebookプロフィールからの情報をサイトに統合したい場合、Facebookのリソースを使用する必要があります。サイトオーナーがソーシャルネットワークのウィジェットを使って体験をパーソナライズし、訪問者のソーシャルネットワークを活用してリーチを増やすことに興味がある限り、ソーシャル統合はサードパーティの領域であり続けると思われます。

ウェブトラフィックの収益化

ウェブのオープンモデルは、コンテンツ制作者の金銭的利益を必ずしも満足させるものではなく、多くのサイト所有者は広告でサイトを収益化することに頼っています。広告主との直接の関係を構築し、価格契約を交渉するのは比較的難しく時間のかかるプロセスであるため、この懸念はターゲット広告とリアルタイム入札を行うサードパーティのプロバイダーによって主に処理されています。否定的な世論の広がり、広告ブロッキング技術の普及、ヨーロッパなどの主要な世界市場での規制措置は、収益化のためにサードパーティのプロバイダを継続的に使用する最大の脅威となっています。サイト所有者が突然独自の広告契約を結んだり特注の広告ネットワークを構築したりすることは考えにくいですが、ペイウォールやBraveのBasic Attention Token[97]のような実験のような代替的なマネタイズモデルは、将来のサードパーティの広告業界を揺るがす可能性を秘めています。

開発の簡素化

何よりもサードパーティのリソースは、ウェブ開発の経験を単純化するために使用されます。以前の使用パターンでさえも、おそらくこのパターンに当てはまる可能性があります。ユーザーの行動分析、広告主とのコミュニケーション、ユーザー体験のパーソナライズなど、サードパーティのリソースはファーストパーティの開発を容易にするため使用されます。

ホスティングプロバイダは、このパターンの最も極端な例です。これらのプロバイダーの中には、技術的な専門知識がなくても、地球上の誰もがサイトのオーナーになれるようしているところもあります。これらのプロバイダーは、資産のホスティング、コーディングの経験がなくてもサイトを構築できるツール、ドメイン登録サービスを提供しています。

97. *https://basicattentiontoken.org/*

サードパーティ・プロバイダの残りの部分も、この使用パターンに当てはまる傾向がある。フロントエンド開発者が使用するためのjQueryなどのユーティリティライブラリのホスティングであれ、Cloudflareのエッジサーバーにキャッシュされているものであれ人気の高いGoogleCDNから提供されている一般的なフォントの膨大なライブラリであれサードパーティのコンテンツはサイトオーナーが心配することを1つ減らし、もしかしたら素晴らしい体験を提供する仕事を少しだけ楽にしてくれるもう1つの方法です。

反響

パフォーマンス

サードパーティコンテンツのパフォーマンスの影響は、カテゴリ的に良い悪いはありません。善良な行為者と悪良な行為者が存在し、カテゴリーの種類によって影響力のレベルが異なります。

良い点：サードパーティ製のフォントやスタイルシートの共有ユーティリティは、平均的にファーストパーティ製のものよりも効率的に提供されます。

ユーティリティ、CDN、およびコンテンツの各カテゴリーは、サードパーティのパフォーマンスにおいて最も輝かしい存在です。これらのカテゴリは、ファーストパーティのソースから提供されるコンテンツと同じ種類のコンテンツの最適化されたバージョンを提供しています。GoogleFontsとTypekitはファーストパーティのフォントよりも平均的に小さい最適化されたフォントを提供し、CloudflareCDNはサイト所有者によっては開発モードで誤って提供されてしまう可能性のあるオープンソースのライブラリをミニ化したバージョンを提供し、GoogleMapsSDKはそうでなければ素朴に大きな画像として提供されてしまう複雑な地図を効率的に配信します。

悪い：非常に小さなエンティティのセットは、ページ上の機能の狭いセットを実行するJavaScriptの実行時間の非常に大きなチャンクを表しています。

広告、ソーシャル、ホスティング、および特定の分析プロバイダーは、ウェブパフォーマンスへの最大の悪影響を表します。ホスティングプロバイダはサイトのコンテンツの大部分を提供しており、他のサードパーティのカテゴリよりもパフォーマンスへの影響が大きいのは当然ですが、ほとんどの場合、JavaScriptをほとんど必要としない静的なサイトを提供していてスクリプトの実行時間を正当化することはできません。しかし、パフォーマンスに影響を与える他のカテゴリーは言い訳できません。これらのカテゴリは、それぞれのページに表示される役割が非常に狭いにもかかわらず、すぐにリソースの大部分を占有してしまいます。例えば、Facebookの「いいね！」ボタンと関連するソーシャルウィジェットは、画面の面積が非常に小さく、ほとんどのウェブ体験の何分の1かしか占めていませんがソーシャルサードパーティのあるページへの影響の中央値はJavaScriptの総実行時間の20%近くになり

ます。状況はアナリティクスについても同様です。トラッキングライブラリは知覚されたユーザー体験に直接貢献しませんが、アナリティクスのサードパーティがあるページへの影響度は90パーセンタイルで、JavaScriptの総実行時間の44%です。

このような少数の事業体がこれほど大きな市場シェアを享受していることの裏には、非常に限られた集中的な努力がウェブ全体に大きな影響を与えることができるということがあります。上位数社のホスティングプロバイダのパフォーマンスを改善するだけで、全ウェブリクエストの2〜3%を改善できます。

プライバシー

サードパーティの最大の利用例は、サイト所有者がユーザーを追跡することであり、一握りの企業がウェブトラフィックの大部分に関する情報を受け取っています。

ユーザーの行動を理解して分析することに対するサイト所有者の関心、それ自体は悪意のあるものではありませんが、ウェブ解析の普及した比較的裏方的な性質は有効な懸念を引き起こし、ヨーロッパのGDPR[98]やカリフォルニア州のCCPA[99]などのプライバシー規制によりユーザー、企業、法律家は近年注目を集めています。開発者がユーザーデータを責任を持って扱い、ユーザーを尊重して扱い、収集されたデータが透明であることを保証することは、アナリティクスを最も人気のあるサードパーティのカテゴリーとして維持し、将来のユーザー価値を提供するためにユーザーの行動を分析するという共生的な性質を維持するための鍵となります。

スクリプトの実行が上位に集中していることは、パフォーマンス向上の潜在的な影響を考えると素晴らしいことですが、プライバシーへの影響を考えるとあまり刺激的ではありません。ウェブ上の全スクリプト実行時間の29%は、GoogleやFacebookが所有するドメイン上のスクリプトだけです。これは、たった2つの事業体によって制御されているCPU時間の非常に大きな割合です。アナリティクスプロバイダーに適用されているのと同じプライバシー保護が、他の広告、ソーシャル、開発者向けユーティリティカテゴリにも適用されるようにすることが重要です。

セキュリティ

セキュリティのトピックについては セキュリティ の章で詳しく説明していますが、サイトに外部の依存関係を導入することによるセキュリティへの影響は、プライバシーへの懸念と密接に関連しています。第三者が任意のJavaScriptを実行できるようにすることは、あなたのページを完全に制御できます。スクリプトがDOMと`window`を制御できれば、すべてのことができるようになります。たとえコードにセキュリティ上の懸念がなくても、単一の障

98. 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
99. https://en.wikipedia.org/wiki/California_Consumer_Privacy_Act

害点を導入できますこれは以前から潜在的な問題として認識されていました[100]。

サードパーティのコンテンツをセルフホスティングする[101]は、ここで述べた懸念事項のいくつかとその他の懸念事項に対応しています。さらに、ブラウザがHTTPキャッシュのパーティショニング[102]を増やしていることから、サードパーティから直接読み込むことのメリットはますます疑問視されています。おそらく多くのユースケースでサードパーティのコンテンツを利用するには、その影響を測定することが難しくなってもこの方法の方が良いでしょう。

結論

サードパーティのコンテンツはどこにでもあります。これは驚くべきことでありません。ウェブの基本は、相互接続とリンクを可能にすることです。この章では、メインドメインから離れてホストされている資産という観点から、サードパーティコンテンツを調べてみました。もし、自己ホスト型のサードパーティ・コンテンツ（例えば、メイン・ドメインにホストされている一般的なオープンソース・ライブラリなど）を含めるとサードパーティの利用率はさらに高まっていたでしょう。

コンピュータ技術での再利用[103]は一般的にベストプラクティスですが、ウェブ上のサードパーティは、ページのパフォーマンス、プライバシー、セキュリティにかなりの影響を与える依存関係を導入します。セルフホスティングと慎重なプロバイダの選択は、これらの影響を軽減するために長い道のりを歩むことができます。

第三者のコンテンツがどのようにページに追加されるかという重要な問題に関わらず、結論は同じです。サードパーティはWebの不可欠な部分です！

著者

Patrick Hulce

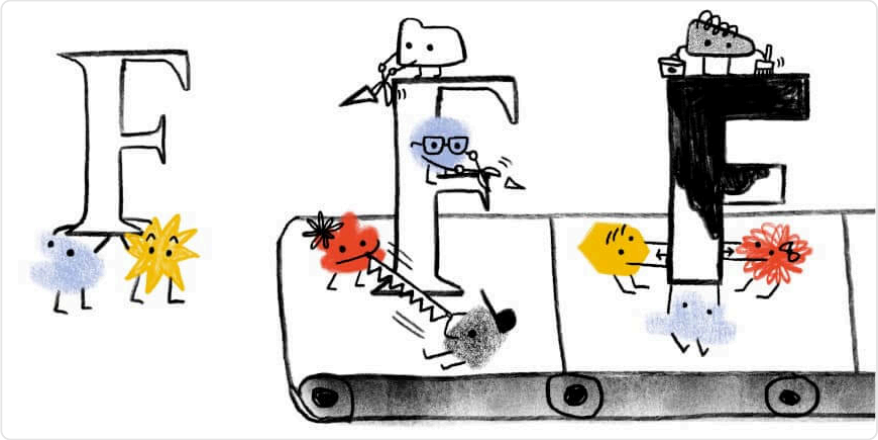
@patrickhulce　patrickhulce　http://patrickhulce.com

Patrick Hulceは元Chromeのエンジニアであり、Eris Ventures[104]の創設者であり、Lighthouse[105]のコアチームメンバーであり、Lighthouse CI[106], DallasJS[107] meetup の共同主催者、third-party-web[108] プロジェクトの執筆者。

100. https://www.stevesouders.com/blog/2010/06/01/frontend-spof/
101. https://csswizardry.com/2019/05/self-host-your-static-assets/
102. https://chromestatus.com/feature/5730772021411840
103. https://en.wikipedia.org/wiki/Code_reuse
104. https://eris.ventures/
105. https://github.com/GoogleChrome/lighthouse
106. https://github.com/GoogleChrome/lighthouse-ci
107. https://www.meetup.com/DallasJS/

108. *https://github.com/patrickhulce/third-party-web*

部 I 章 6
フォント



Zach Leatherman によって書かれた。
John Teague と Aymen Loukil によってレビュ。
TJ Monserrat 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

ウェブフォントは、ウェブ上で美しく機能的なタイポグラフィを可能にします。ウェブフォントを使用することは、デザインに力を与えるだけでなく、デザインのサブセットを民主化します。しかし、どんなに良いことがあってもウェブフォントが適切に読み込まれていないと、サイトのパフォーマンスに大きな悪影響を及ぼすこともあります。

それらはウェブにとってプラスになるのか？　それらは害よりも多くの利益を提供しているか?　Web標準の牛道は、デフォルトでWebフォントの読み込みのベストプラクティスを奨励するために十分に舗装されているだろうか？　そうでない場合、何を変える必要があるのでしょうか？　今日のウェブ上でウェブフォントがどのように使用されているかを調べることで、これらの疑問に答えられるかどうかをデータ駆動型で覗いてみましょう。

そのウェブフォントはどこで手に入れたの？

最初の、そして最も顕著な問題は、パフォーマンスです。パフォーマンスに特化した章がありますが、ここではフォント固有のパフォーマンスの問題について少し掘り下げてみましょう。

ホストされたWebフォントを使用すると、実装やメンテナンスが容易になりますが、セルフホスティングは最高のパフォーマンスを提供します。Webフォントはデフォルトで、Webフォントの読み込み中にテキストを非表示にする（Flash of Invisible Text[109]、またはFOITとしても知られています）ことを考えると、Webフォントのパフォーマンスは画像のような非ブロッキング資産よりも重要になる可能性があります。

フォントは同じホストでホストされているのか、それとも別のホストでホストされているのか？

サードパーティのホスティングに対するセルフホスティングの差別化は、HTTP/2の世界ではますます重要になってきています。同一ホストのリクエストには、ウォーターフォール内の他の同一ホストのリクエストに対して優先順位をつける可能性が高いという大きな利点があります。

別のホストからウェブフォントを読み込む際のパフォーマンスコストを軽減するための推奨事項としては、`preconnect`、`dns-prefetch`、`preload` リソースのヒントの使用がありますが、優先度の高いウェブフォントは、ウェブフォントのパフォーマンスへの影響を最小限に抑えるため、同一ホストからのリクエストにすべきです。これは視覚的に、非常に目立つコンテンツやページの大部分を占める本文コピーで使用されるフォントへ対して特に重要です。

109. *https://css-tricks.com/fout-foit-foft/*

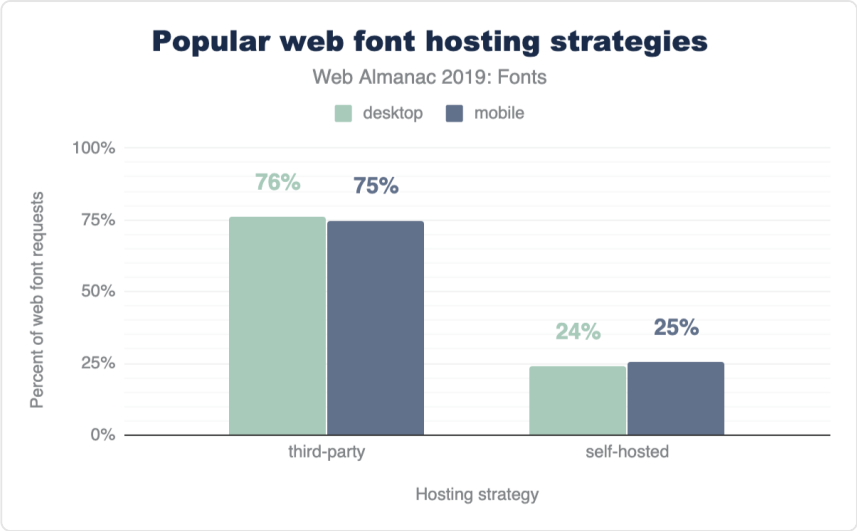


図6.1. 人気のあるウェブフォントのホスティング戦略。font hosting strategies.

4分の3がホストされているという事実は、おそらく我々が議論するGoogle Fontsの優位性を考えると意外と知られていません以下。

Googleは`https://fonts.googleapis.com`でホストされているサードパーティのCSSファイルを使ってフォントを提供しています。開発者は、マークアップの`<link>`タグを使ってこれらのスタイルシートにリクエストを追加します。これらのスタイルシートはレンダーブロッキングされていますが、そのサイズは非常に小さいです。しかし、フォントファイルは`https://fonts.gstatic.com`という別のドメインでホストされています。2つの異なるドメインへの2つの別々のホップを必要とするモデルでは、CSSがダウンロードされるまで発見されない2つ目のリクエストには`preconnect`が最適な選択肢となります。

`preload`はリクエストのウォーターフォールの上位にフォントファイルをロードするための素晴らしい追加機能ですが（`preconnect`は接続を設定するもので、ファイルの内容をリクエストするものではないことを覚えておいてください）、`preload`はGoogle Fontsではまだ利用できません。Google Fontsはフォントファイル用のユニークなURLを生成しますこれは変更される可能性があります[110]。

110. *https://github.com/google/fonts/issues/1067*

最も人気のあるサードパーティ製のホストは何ですか？

ホスト	デスクトップ	モバイル
fonts.gstatic.com	75.4%	74.9%
use.typekit.net	7.2%	6.6%
maxcdn.bootstrapcdn.com	1.8%	2.0%
use.fontawesome.com	1.1%	1.2%
static.parastorage.com	0.8%	1.2%
fonts.shopifycdn.com	0.6%	0.6%
cdn.shopify.com	0.5%	0.5%
cdnjs.cloudflare.com	0.4%	0.5%
use.typekit.com	0.4%	0.4%
netdna.bootstrapcdn.com	0.3%	0.4%
fast.fonts.net	0.3%	0.3%
static.dealer.com	0.2%	0.2%
themes.googleusercontent.com	0.2%	0.2%
static-v.tawk.to	0.1%	0.3%
stc.utdstc.com	0.1%	0.2%
cdn.jsdelivr.net	0.2%	0.2%
kit-free.fontawesome.com	0.2%	0.2%
open.scdn.co	0.1%	0.1%
assets.squarespace.com	0.1%	0.1%
fonts.jimstatic.com	0.1%	0.2%

図6.2. リクエストの割合による上位20のフォントホスト。

ここでのGoogle Fontsの優位性は、同時に驚くべきことであると同時に意外性のないものであった。期待していたという点では予想外でしたが、サービスの圧倒的な人気の高さには驚きました。フォントリクエストの75%というのは驚異的だ。TypeKitは一桁台の遠い2位で、Bootstrapライブラリがさらに遠い3位を占めていました。

29%

図6.3. ドキュメント内にGoogle Fontsスタイルシートのリンクを含むページの割合 `<head>`。

ここでのGoogle Fontsの使用率の高さは非常に印象的だが、Google Fonts `<link>` 要素を含むページが29%しかなかったことも注目に値する。これはいくつかのことを意味しているかもしれない。

- ページがGoogle Fontsを使用する場合、彼らはGoogle Fontsを多く使用しています。それらは結局のところ、金銭的なコストなしで提供されています。おそらく、人気のあるWYSIWYGエディタで使われているのではないでしょうか？　これは非常に可能性の高い説明のように思えます。
- あるいは、もっとありそうにない話としては、多くの人が `<link>` の代わりに `@import` を使ってGoogle Fontsを使っているということかもしれません。
- あるいは、超ありえないシナリオにまで踏み込んでみたいのであれば、多くの人が代わりにHTTP `Link:` ヘッダーを使ってGoogle Fontsを使っているということになるかもしれません。

0.4%

図6.4. ドキュメントの最初の子としてGoogle Fontsスタイルシートのリンクを含むページの割合 `<head>`。

Google Fontsのドキュメントでは、Google Fonts CSSの `<link>` はページの `<head>` の最初の子として配置することを推奨しています。これは大きなお願いです！　実際、これは一般的でありません。全ページの半分のパーセント（約20,000ページ）しかこのアドバイスを受けていないので、これは一般的でありません。

さらに言えば、ページが `preconnect` や `dns-prefetch` を `<link>` 要素として使用している場合、これらはいずれにしてもGoogle Fonts CSSの前へ来ることになります。これらのリソースのヒントについては、続きを読んでください。

サードパーティホスティングの高速化

上述したように、サードパーティホストへのウェブフォント要求を高速化する超簡単な方法

は、 `preconnect` リソースヒントを使用することです。

1.7%

図6.5. モバイルページがウェブフォントホストにプリコネクティングしている割合。

うわー！　2%未満のページが `preconnect` を使用している！　Google Fontsが75%であることを考えると、これはもっと高いはずです！　開発者の皆さん: Google Fontsを使うなら、`preconnect` を使いましょう！　Google Fonts: `preconnect` をもっと宣伝しよう！

実際、もしあなたがGoogle Fontsを使っているのであれば、 `<head>` にこれを追加してください。

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

最も人気のある書体

ランク	フォントファミリー	デスクトップ	モバイル
1	Open Sans	24%	22%
2	Roboto	15%	19%
3	Montserrat	5%	4%
4	Source Sans Pro	4%	3%
5	Noto Sans JP	3%	3%
6	Lato	3%	3%
7	Nanum Gothic	4%	2%
8	Noto Sans KR	3%	2%
9	Roboto Condensed	2%	2%
10	Raleway	2%	2%
11	FontAwesome	1%	1%
12	Roboto Slab	1%	1%
13	Noto Sans TC	1%	1%
14	Poppins	1%	1%
15	Ubuntu	1%	1%
16	Oswald	1%	1%
17	Merriweather	1%	1%
18	PT Sans	1%	1%
19	Playfair Display	1%	1%
20	Noto Sans	1%	1%

図6.6. 全フォント宣言に占める上位20のフォントファミリーの割合。

ここでの上位のエントリがGoogle Fontsの人気順フォント一覧[111]と非常によく似ていることは驚くに値しません。

111. https://fonts.google.com/?sort=popularity

どのようなフォント形式を使用していますか？

今日のブラウザではWOFF2はかなりサポートされています[112]。Google FontsはWOFF2というフォーマットを提供していますが、これは前身のWOFFよりも圧縮率が向上したフォーマットで、それ自体はすでに他の既存のフォントフォーマットよりも改善されていました。

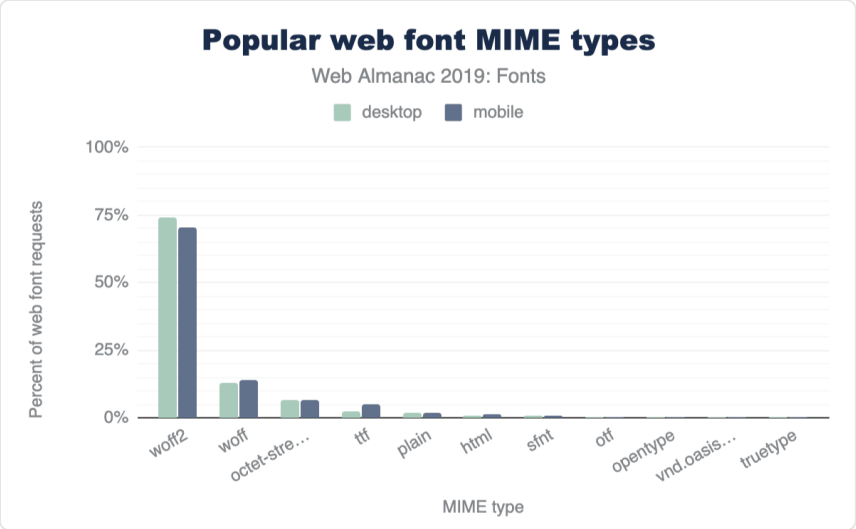


図6.7. ウェブフォントのMIMEタイプの普及率

私から見れば、ここでの結果を見て、WebフォントはWOFF2オンリーにした方がいいという意見もあるかもしれません。二桁台のWOFF使用率はどこから来ているのでしょうか？もしかして、まだWebフォントをInternet Explorerに提供している開発者がいるのでしょうか？

第3位の`octet-stream`(およびもう少し下の`plain`)は、多くのウェブサーバが不適切に設定されており、ウェブフォントファイルのリクエストで誤ったMIMEタイプを送信していることを示唆しているように見えます。

もう少し深く掘り下げて、`@font-face`宣言の`src:`プロパティで使われている`format()`の値を見てみましょう。

112. *https://caniuse.com/#feat=woff2*

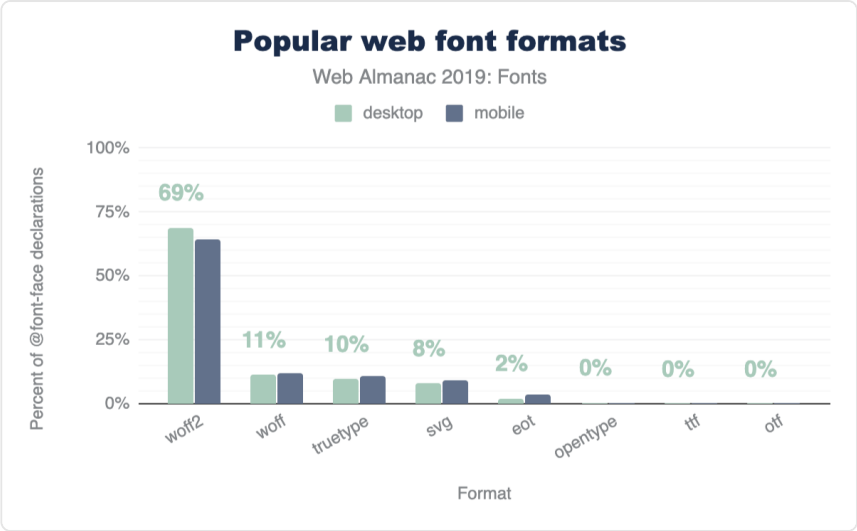


図6.8. `@font-face` 宣言におけるフォントフォーマットの人気度。

SVGフォント[113]が衰退しているのを見て期待していたのですが。バグだらけだし、Safari以外のブラウザでは実装が削除されている。そろそろ捨ててしまおうか。

ここのSVGデータポイントを見ると、どのMIMEタイプでSVGフォントを提供しているのか気になります。図6.7のどこにも `image/svg+xml` は見当たりません。とにかく、それを修正することは気にしないで、ただそれらを取り除くだけです！

113. *https://caniuse.com/#feat=svg-fonts*

WOFF2専用

ランク	フォーマットの組み合わせ	デスクトップ	モバイル
1	woff2	84.0%	81.9%
2	svg, truetype, woff	4.3%	4.0%
3	svg, truetype, woff, woff2	3.5%	3.2%
4	eot, svg, truetype, woff	1.3%	2.9%
5	woff, woff2	1.8%	1.8%
6	eot, svg, truetype, woff, woff2	1.2%	2.1%
7	truetype, woff	0.9%	1.1%
8	woff	0.7%	0.8%
9	truetype	0.6%	0.7%
10	truetype, woff, woff2	0.6%	0.6%
11	opentype, woff, woff2	0.3%	0.2%
12	svg	0.2%	0.2%
13	eot, truetype, woff	0.1%	0.2%
14	opentype, woff	0.1%	0.1%
15	opentype	0.1%	0.1%
16	eot	0.1%	0.1%
17	opentype, svg, truetype, woff	0.1%	0.0%
18	opentype, truetype, woff, woff2	0.0%	0.0%
19	eot, truetype, woff, woff2	0.0%	0.0%
20	svg, woff	0.0%	0.0%

図6.9. フォントフォーマットの組み合わせトップ20

このデータセットは、大多数の人がすでに`@font-face`ブロックでWOFF2のみを使っていることを示唆しているように見える。しかし、このデータセットにおけるGoogle Fontsの優位性についての以前の議論によれば、もちろんこれは誤解を招くものです。Google Fontsは合理化されたCSSファイルを提供するためにいくつかのスニッフィングメソッドを実行して

おり、最新の `format()` のみを含んでいる。当然のことながら、WOFF2がここでの結果を支配しているのはこの理由によるもので、WOFF2に対するブラウザのサポートは以前からかなり広くなっている。

重要なのは、この特定のデータはまだWOFF2オンリーのケースを支持しているわけではないということですが、魅力的なアイデアであることに変わりはありません。

見えない文字との戦い

デフォルトのWebフォントの読み込み動作である「読み込み中は見えない」(FOITとしても知られています)に対抗するため持っている第一のツールは `font-display` です。 `font-display: swap` を `@font-face` ブロックに追加すると、ウェブフォントが読み込まれている間にフォールバックテキストを表示するようにブラウザに指示する簡単な方法です。

ブラウザ対応[114]もいいですね。Internet ExplorerやChromium以前のEdgeではサポートされていませんが、Webフォントが読み込まれたときにデフォルトでフォールバックテキストをレンダリングしてくれます（ここではFOITは使えません）。Chromeのテストでは、 `font-display` はどのくらいの頻度で使われているのでしょうか？

26%

図6.10. `font-display` スタイルを利用しているモバイルページの割合。

私はこれが時間の経過とともに忍び寄ってくることを想定しています、特に今はGoogle Fontsがすべての新しいコードスニペットに `font-display` を追加していますが彼らのサイトからコピーされています。

Google Fontsを使っているなら、スニペットを更新しよう！　Google Fontsを使っていない場合は、 `font-display` を使いましょう！　 `font-display` についての詳細は MDN[115] を参照してください。

どのような `font-display` 値が人気あるのか見てみましょう。

114. *https://caniuse.com/#feat=mdn-css_at-rules_font-face_font-display*
115. *https://developer.mozilla.org/docs/Web/CSS/@font-face/font-display*

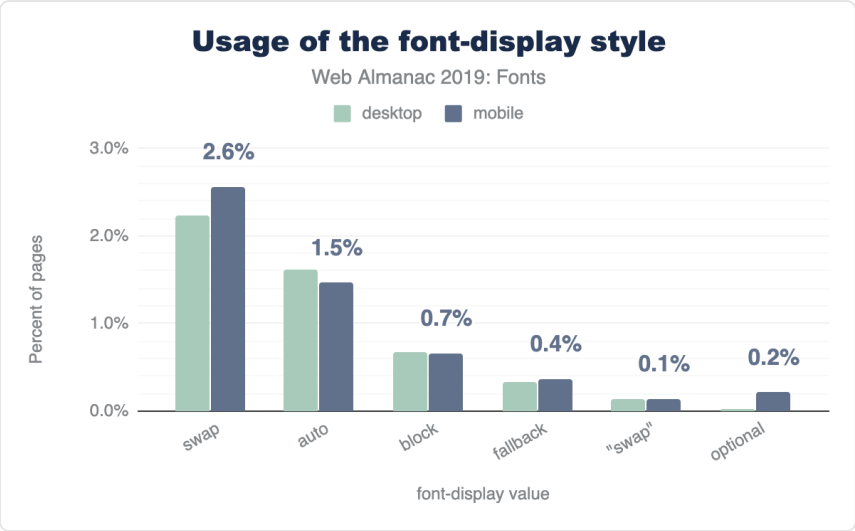


図6.11. `font-display` の値の使用法。

ウェブフォントの読み込み中にフォールバックテキストを表示する簡単な方法として、`font-display: swap` が最も一般的な値として君臨しています。`swap` は新しいGoogle Fontsのコードスニペットでもデフォルト値として使われています。いくつかの著名な開発者のエバンジェリストがこれを求めてちょっとした働きかけをしていたので、ここでは `optional` (キャッシュされた場合にのみレンダリングする)がもう少し使われることを期待していたのですが、駄目でした。

ウェブフォントの数が多すぎる？

ある程度のニュアンスが必要な質問です。フォントはどのように使われているのか？　ページ上のコンテンツの量は？　そのコンテンツはレイアウトのどこにあるのか？　フォントはどのようにレンダリングされているのか？　しかし、ニュアンスの代わりに、リクエスト数を中心とした大まかで重い分析に飛び込んでみましょう。

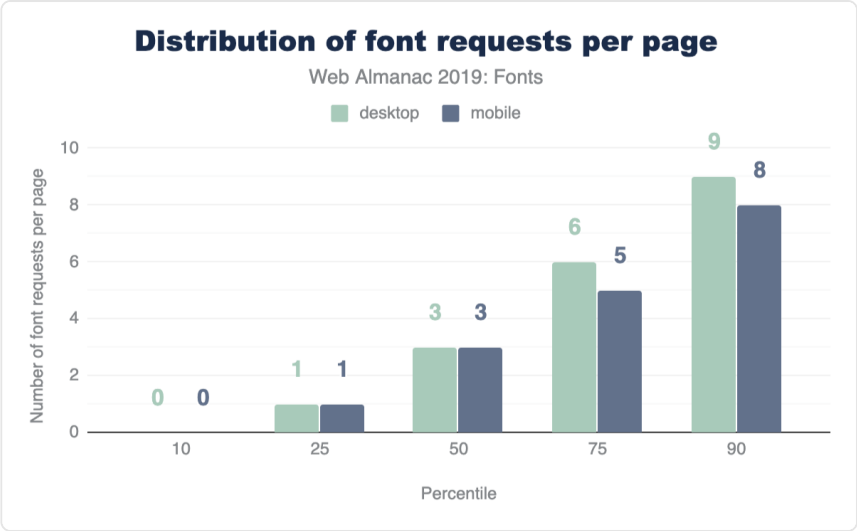


図6.12. ページあたりのフォント要求の分布。

中央値のウェブページでは、3つのウェブフォントをリクエストしています。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6つと9つのウェブフォントをリクエストしています。

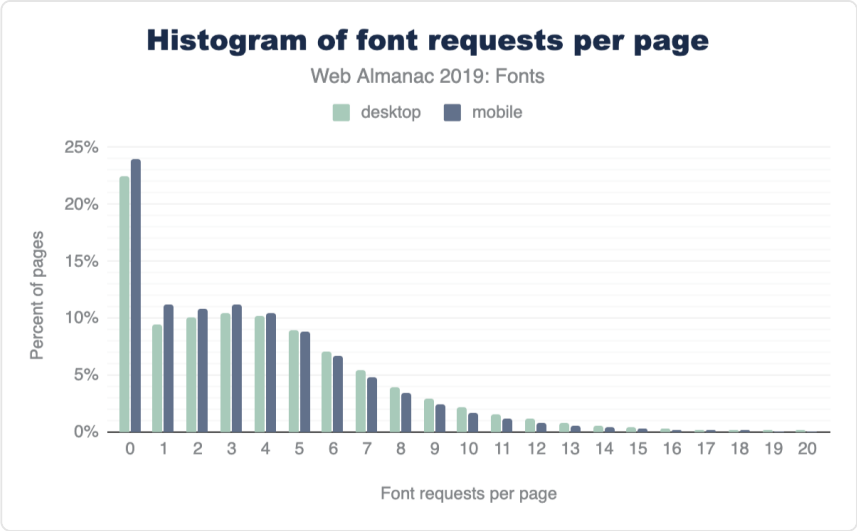


図6.13. ページあたりに要求されたウェブフォントのヒストグラム。

Webフォントのリクエストがデスクトップとモバイルの間でかなり安定しているように見えるというのは非常に興味深いことです。私は、`@media`クエリの中の`@font-face`ブロックを隠すことを推奨することが流行らなかったのを見てうれしく思います (何も考えないでください)。

しかし、モバイルデバイスでのフォントのリクエストはわずかに多い。ここでの私の勘は、モバイルデバイスで利用できる書体が少ないということはGoogle Fonts CSSでの`local()`のヒット数が少ないということであり、ネットワークからのフォントリクエストに戻ってしまうのではないかと考えています。

この賞を受賞したくない

718

図6.14. 1ページで最も多くのWebフォントのリクエストがあった。

最も多くのウェブフォントをリクエストしたページの賞は、**718**のウェブフォントをリクエストしたサイトに贈られます！

コードに飛び込んだ後、それらの718のリクエストのすべてがGoogle Fontsに向かっています！　どうやらWordPress用の「ページの折り返しの上に」最適化プラグインが誤作動して、このサイトで不正を行い、すべてのGoogle Fonts-oopsにリクエストしている（DDoS-ing？）。

パフォーマンス最適化プラグインは、あなたのパフォーマンスをはるかに悪化させることができることを皮肉っています！

`Unicode-range`を使うとより正確なマッチングが可能になります

56%

図6.15. `unicode-range` プロパティでWebフォントを宣言しているモバイルページの割合。

`unicode-range`は、ブラウザに、ページがフォントファイルで使用したいコードポイント

を具体的に知らせるための優れたCSSプロパティです。`@font-face`宣言に`unicode-range`がある場合、ページ上のコンテンツは、フォントが要求される前に、その範囲内のコードポイントのいずれかにマッチしなければなりません。これは非常に良いことです。

Google FontsはそのCSSのほとんど（すべてではないにしても）で`unicode-range`を使用しているので、これもGoogle Fontsの使用状況によって偏っていると予想される指標です。ユーザーの世界でこれはあまり一般的でないと思いますが、Web Almanacの次の版ではGoogle Fontsのリクエストをフィルタリングして除外することが可能かもしれません。

システムフォントが存在する場合、ウェブフォントを要求しないようにする

59%

図6.16. `local()` プロパティでWebフォントを宣言しているモバイルページの割合。

`local()`は`@font-face`、`src`のシステムフォントを参照するための良い方法です。もし`local()`フォントが存在するならば、ウェブフォントを要求する必要は全くありません。これはGoogle Fontsによって広く使われており、論争の的にもなっているのでユーザの土地からパターンを得ようとしているのであれば、これも歪んだデータの一例になるでしょう。

ここでは、私よりも賢い人々(TypeKitのBram Stein氏) が、インストールされているフォントのバージョンが古くて信頼性は低い場合があるため、`local()`を使うことは予測不可能である可能性があると述べていることにも注目しておきましょう。

縮約されたフォントと`font-stretch`

7%

図6.17. `font-stretch` プロパティを持つスタイルを含むデスクトップページとモバイルページの割合。

歴史的に、`font-stretch`はブラウザのサポートが悪く、よく知られた`@font-face`プロパティではありませんでした。詳しくはMDNの`font-stretch`について を参照してくださ

い。しかし、ブラウザのサポート[116]は広がっています。

小さいビューポートで凝縮されたフォントを使用することで、より多くのテキストを表示できるようになることが示唆されていますが、このアプローチは一般的には使用されていません。とはいえ、このプロパティがモバイルよりもデスクトップで半パーセントポイント多く使われているというのは予想外で、7%というのは私が予想していたよりもはるかに高いと思えます。

可変フォントは未来のもの

可変フォント[117]では、1つのフォントファイルに複数のフォントの太さやスタイルを含めることができます。

1.8%

図6.18. 可変フォントを含むページの割合

1.8%でさえ、これは予想よりも高かったが、これがうまくいくのを見て興奮している。Google Fonts v2[118]には可変フォントのサポートがいくつか含まれています。

116. *https://caniuse.com/#feat=css-font-stretch*
117. *https://developer.mozilla.org/docs/Web/CSS/CSS_Fonts/Variable_Fonts_Guide*
118. *https://developers.google.com/fonts/docs/css2*

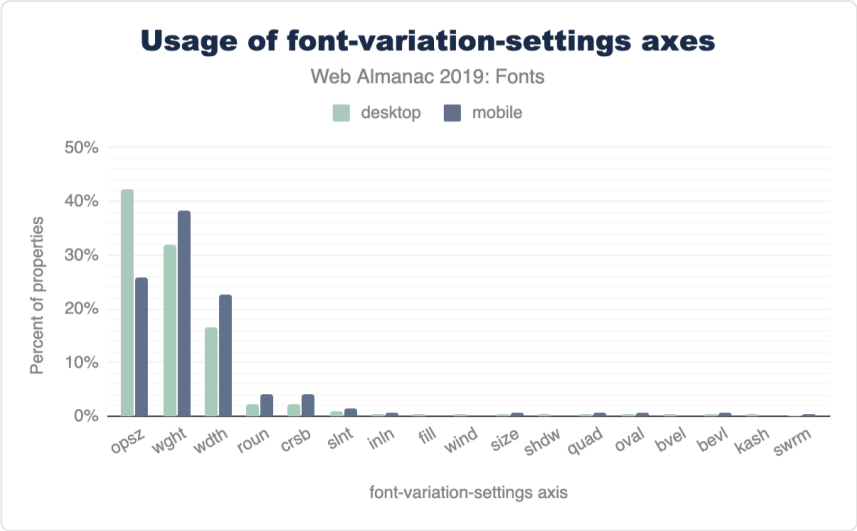


図6.19. `font-variation-settings` 軸の使用法。

この大規模なデータセットのレンズを通して見ると、これらの結果は非常に低いサンプルサイズであることがわかります。しかし、デスクトップページで最も一般的な軸として`opsz`が注目され、`wght`と`wdth`が後に続く。私の経験では、可変フォントの入門デモはたいていウェイトベースです。

カラーフォントも未来かも？

117

図6.20. カラーフォントを含むデスクトップウェブページの数。

これらのここでの使用法は基本的に存在しませんが、詳細についてはカラーフォント！WTF?[119]という優れたリソースをチェックできます。フォント用のSVGフォーマット(これは良くないし消えていく)に似ていますが(全くそうではありません)、これを使うとOpenTypeファイルの中にSVGを埋め込むことができ、これは素晴らしくクールです。

119. https://www.colorfonts.wtf/

結論

ここでの最大の収穫は、Google Fontsがウェブフォントの議論を支配しているということだ。彼らが取ったアプローチは、ここで記録したデータに大きく影響している。ここでのポジティブな点はウェブフォントへのアクセスが容易であること、優れたフォントフォーマット（WOFF2）であること、そして自由な`unicode範囲`の設定が可能であることだ。ここでの欠点はサードパーティのホスティング、異なるホストからのリクエスト、および`preload`にアクセスできないことでパフォーマンスが低下することです。

私は、将来的には「バリアブルフォントの台頭」を見ることになるだろうと完全に予想しています。バリアブルフォントは複数の個々のフォントファイルを1つの合成フォントファイルに結合するので、これはウェブフォントのリクエストの減少と対になっているはずです。しかし歴史が示しているように、ここで通常起こることは、あるものを最適化してその空所を埋めるためにさらに多くのものを追加してしまうことです。

カラーフォントの人気が高まるかどうかは非常に興味深いところです。私は、これらは可変フォントよりもはるかにニッチなものになると予想していますが、アイコンフォントのスペースに生命線を見ることができるかもしれません。

フォントを凍らせるなよ。

著者

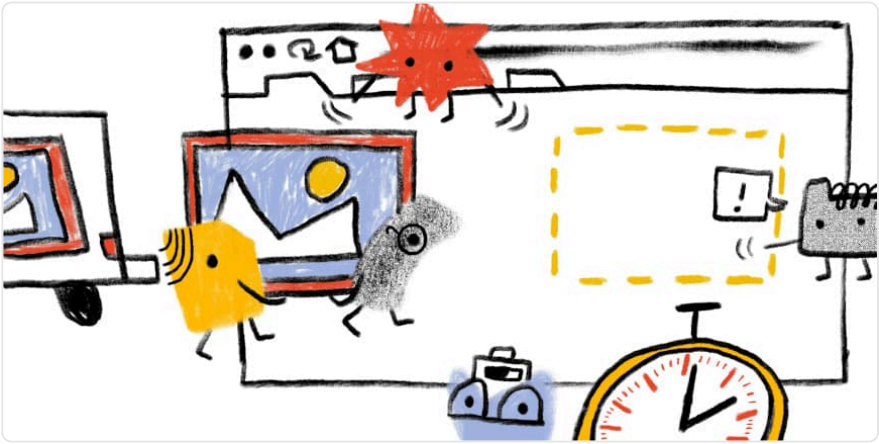
Zach Leatherman

@zachleat　zachleat　https://zachleat.com/

ZachはFilament Group[120]のWeb開発者だ。彼は現在、web fonts[121]とstatic site generators[122]に夢中だ。彼の講演履歴[123]には、JAMstack_conf、Beyond Tellerrand、Smashing Conference、CSSConf、そしてThe White House[124]のようなイベントでの8カ国での講演が含まれています。また、NEJS CONF[125]やNebraskaJS[126]のミートアップも手伝っている。

120. https://www.filamentgroup.com/
121. https://www.zachleat.com/web/fonts/
122. https://www.zachleat.com/web/introducing-eleventy/
123. https://www.zachleat.com/web/speaking/
124. https://www.zachleat.com/web/whitehouse/
125. http://nejsconf.com/
126. http://nebraskajs.com

部 II 章 7
パフォーマンス



***Rick Viscomi** によって書かれた。*
José M. Pérez、 David Fox、 Sergey Chernyshev と Mark Zeman によってレビュ。
Rick Viscomi と Raghu Ramakrishnan による分析。
Rachel Costello 編集。
M.Sakamaki によって翻訳された。

導入

パフォーマンスはユーザー体験で大切なものの一つです。多くのWebサイトでは、ページの読み込み時間を早くする事によるユーザー体験の向上と、コンバージョン率の上昇は一致しています。逆に、パフォーマンスが低い場合、ユーザーはコンバージョンを達成せず、不満を持ち、ページをクリックすると怒りを覚えることさえあります。

Webのパフォーマンスを定量化する方法は色々とあります。ここで一番大切なのは、ユーザーにとって特に重要な点を計測することです。ただ、`onload`や`DOMContentLoaded`などのイベントはユーザーが実際に目で見て体験できているものとは限りません。例えば、電子メールクライアントを読み込んだ時、受信トレイの内容が非同期に読み込まれる間、画面全体を覆うようなプログレスバーが表示される事があります。ここでの問題は`onload`イベントが受信ボックスの非同期読み込みの完了まで待機しないことです。この例において、ユーザーの一番大切にするべき計測値とは「受信トレイが使えるようになるまでの時間」であり、`onload`イベントに着目するのは誤解を招く可能性があります。そのために、この章で

はユーザーが実際にページをどのように体験しているかを把握し、よりモダンで広く使える描画、読み込み、および対話性の計測を検討します。

パフォーマンスデータにはラボとフィールドの２種類があります。合成テストや実ユーザー測定（またはRUM）でそれらを聞いたことがあるかもしれません。ラボでパフォーマンスを測定すると、各Webサイトが共通の条件でテストされ、ブラウザー、接続速度、物理的な場所、キャッシュ状態などの状態は常に同じになります。この一貫性が保証されることで、それぞれのWebサイトを比較することができます。その反面、フィールドのパフォーマンス測定は、ラボでは決して行うことのできない無限に近い条件の組み合わせで、現実に近いユーザーのWeb体験を計測することを可能にします。この章の目的と実際のユーザー体験を理解するために、今回はフィールドデータを見ていきます。

パフォーマンスの状態

Web Almanacにある他のほとんどの章は、HTTP Archive[127]のデータに基づいています。ただ、実際のユーザーがWebをどのように体験するかを取得するには、違うデータセットが必要になります。このセクションでは、Chrome UXレポート[128]（CrUX）を使用しています。この情報はHTTP Archiveとすべて同じウェブサイトで構成されるGoogleの公開データセットとなっており、Chromeを使うユーザーの実際の体験を集約しています。そして体験は次のように分類されます。

- ユーザーデバイスのフォームファクタ
    - デスクトップ
    - 携帯電話
    - タブレット
- モバイル用語で言うユーザーの有効な接続タイプ(ECT)
    - オフライン
    - 遅い2G
    - 2G
    - 3G
    - 4G
- ユーザーの地理的な位置

127. *https://httparchive.org/*
128. *http://bit.ly/chrome-ux-report*

体験は描画、読み込み、そして対話性の定量化を含めて毎月測定されます。最初に私達が見るべき指標はコンテンツの初回ペイント(First Contentful Paint)[129](FCP)です。これはページや画像やテキストなど、ユーザーが画面として見るために必要なものが表示されるのを待つ時間です。次は、読み込み時間の指標である最初のバイトまでの時間(Time to First Byte)[130](TTFB)です。これはユーザーがナビゲーションを行ってから、Webページのレスポンスの最初のバイトを受信するまでにかかった時間を計測したものです。そして最後に確認するフィールドの指標は初回入力遅延(First Input Delay)[131] (FID)です。これは比較的新しい指標で、読み込み以外のパフォーマンスUXの一部を表すものです。ユーザーがページのUIを操作できるようになるまでの時間、つまり、ブラウザのメインスレッドがイベント処理の準備が整うまでの時間を測定したものです。

では、それによってどのような洞察ができるのかを見てきましょう。

コンテンツの初回ペイント(First Contentful Paint)

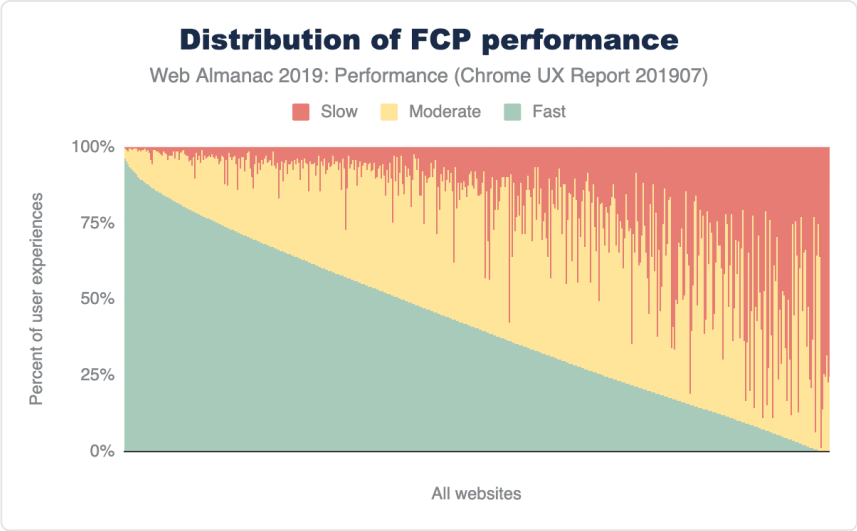


図7.1. Webサイトの高速、適度、および低速のFCPパフォーマンスの分布。

図7.1では、FCPの体験がWeb全体でどのように分散しているかを見ることができます。このチャートは、CrUXデータセット内にある数百万のWebサイト分布を1,000個のWebサイトに圧縮しており、図の縦線一つ一つはWebサイトを表しています。このグラフは、１秒未満の高速なFCP体験、３秒以上かかる遅い体験、その中間にある適度な体験（以前は平均と言われていた）の割合で並べられています。グラフには、ほぼ100%高速な体験を備えたWeb

129. https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics#first_paint_and_first_contentful_paint
130. https://developer.mozilla.org/docs/Glossary/time_to_first_byte
131. https://developers.google.com/web/updates/2018/05/first-input-delay

サイトと、ほぼ100%低速な体験となっているWebサイトが存在しています。その中間にある、高速、適度、及び低速のパフォーマンスが混じり合ったWebサイトは、低速よりも適度か高速に傾いており、良い結果になっています。

注意：ユーザー体験の低下があった場合、その理由が何であるか突き止めるのは難しいでしょう。Webサイト自体が不十分で非効率な構築がされている可能性があるかもしれませんが、ユーザーの通信速度が遅い可能性やキャッシュが空など、他の環境要因がある可能性があります。そのため、このフィールドデータを見てユーザー体験が悪いとわかっても、理由は必ずしもWebサイトにあるとは言えません。

Webサイトが十分に**高速**かどうかを分類するために、新しい方法論である PageSpeed Insights (PSI)を使います。この方法はWebサイトのFCP体験の少なくとも75%が 1 秒未満でなければなりません。同様に、FCP体験がとても**低速**となる25%のWebサイトでは 3 秒以上かかっています。どちらの条件も満たさない場合、Webサイトのパフォーマンスは**適度**です。

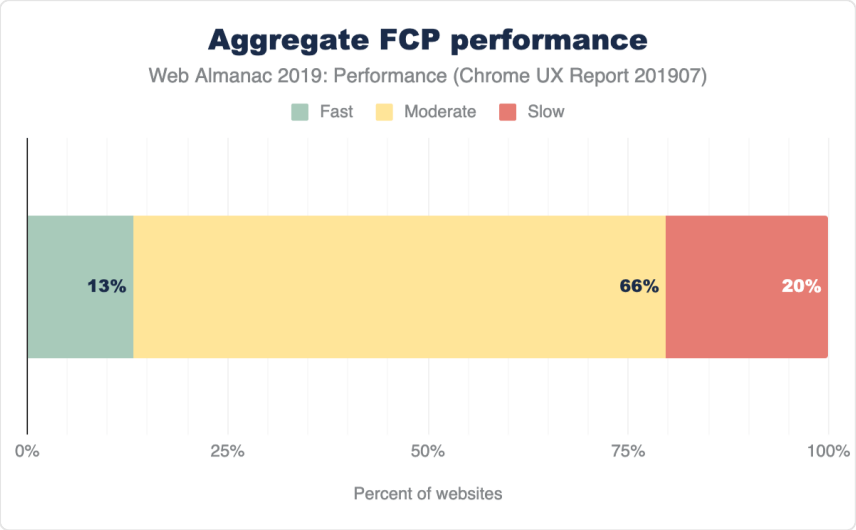


図7.2. 高速、適度、低速のFCPラベルが貼られたWebサイトの分布。

図7.2の結果は、Webサイトの13%だけが高速と判断されています。これはまだ改善の余地があるようですが、多くのWebサイトで意味を持つコンテンツを素早く一貫して描画できています。Webサイトの 3 分の 2 は適度のFCP体験となっているようです。

デバイス毎にFCPのユーザー体験がどの様になっているかを知るために、フォームファクタ別に分類してみましょう。

デバイス毎のFCP

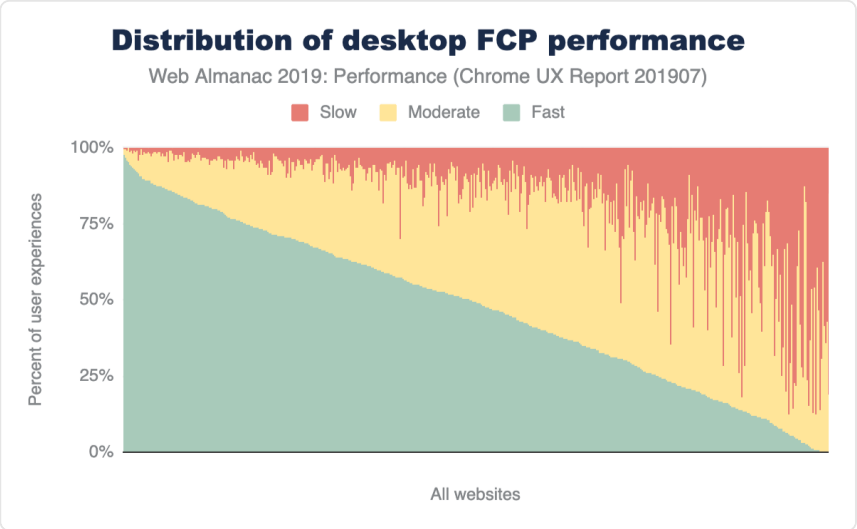


図7.3. デスクトップWebサイトの高速、適度、低速のFCPパフォーマンスの分布。

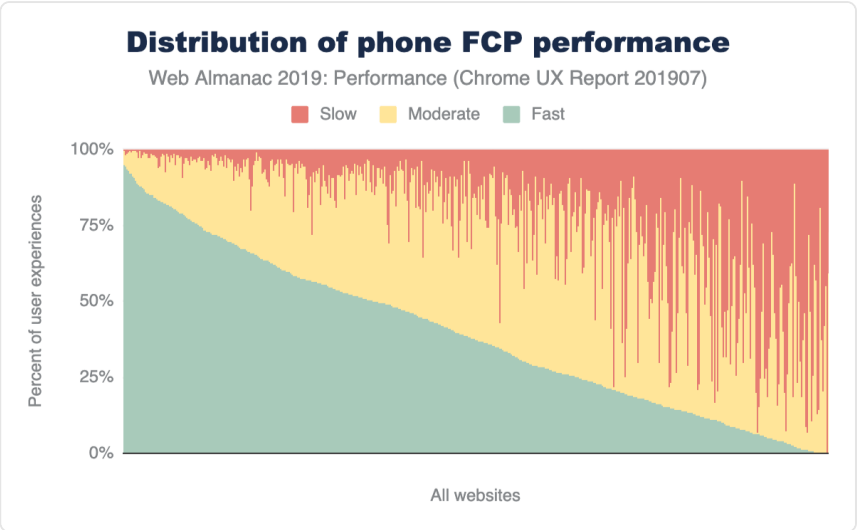


図7.4. 携帯電話向けWebサイトの高速、適度、低速のFCPパフォーマンスの分布。

上の図7.3と図7.4は、FCPの分布をデスクトップと携帯電話で分類しています。微妙な差ですが、デスクトップFCP分布の胴部は携帯電話ユーザーの分布よりも凸型となっているように見えます。この視覚的な近似が示すのは、デスクトップユーザーが高速なFCPにおいて全

体的に割合が高いことを示しています。これを検証するために、PSIという方法を各分布に適用していきます。

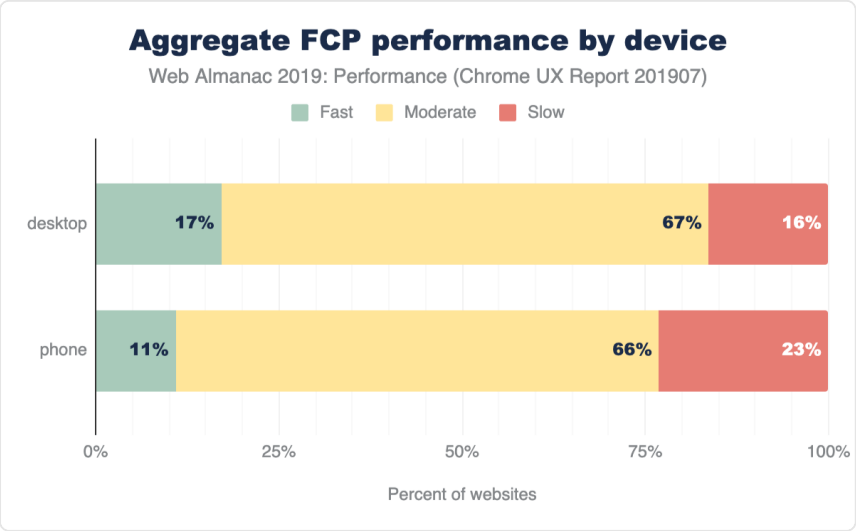


図7.5. 高速、適度、低速FCPのラベルが付けられたWebサイトの分布。デバイスの種類で分類されています。

PSIの分類によると、モバイルユーザーの11%と比べて、デスクトップユーザーは17%に対して高速なFCP体験が全体的に提供されています。全体的な分布を見ると、デスクトップのほうが体験が少しだけ高速に偏っていますが、低速のWebサイトは少なく高速と適度のカテゴリーが多くなっています。

Webサイトでデスクトップユーザーが高確率で携帯電話のユーザーより高速なFCP体験をするのは何故でしょう？それは結局、このデータセットはWebがどのように機能しているかという答えでしかなく必ずそう動いていると言った話では無いからです。ただ、デスクトップユーザーはキャリアの通信ではなく、WiFiのような高速で信頼性の高いネットワークでインターネット接続をしていると推測できます。この疑問に答えるために、ECTでユーザー体験がどのように違うかを調べることもできます。

有効な接続タイプ毎のFCP

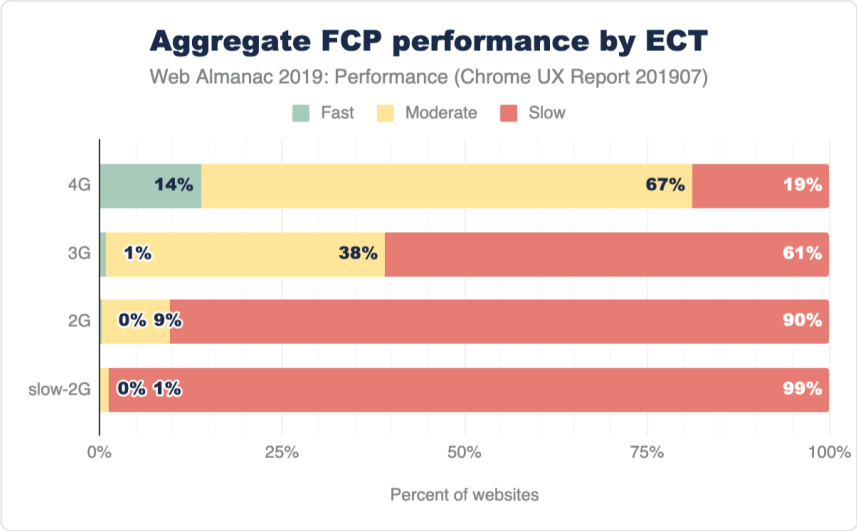


図7.6. 高速、適度、低速のFCPでラベル付けされたWebサイトの分布。ECTで分類されています。

上の図7.6にあるFCP体験は、ユーザーの体験するECT毎にグループ化されています。興味深いことに、ECTの速度と高速FCPを提供するWebサイトの割合との間には相関関係があります。ECTの速度が低下すると、高速な体験の割合はゼロに近づきます。ECTが4Gのユーザーにサービスを提供しているWebサイトの14%は高速なFCPエクスペリエンスを提供していますが、そのWebサイトの19%は低速な体験を提供しています。61%のWebサイトは、ECTが3Gのユーザーに低速のFCPを提供し、ECTが2Gだと90%に、ETCが低速の2Gだと99%となっています。これらの事実から、4Gより遅い接続を持つユーザーには、Webサイトがほぼ一貫して高速のFCPを提供できていないことを示しています。

地理によるFCP

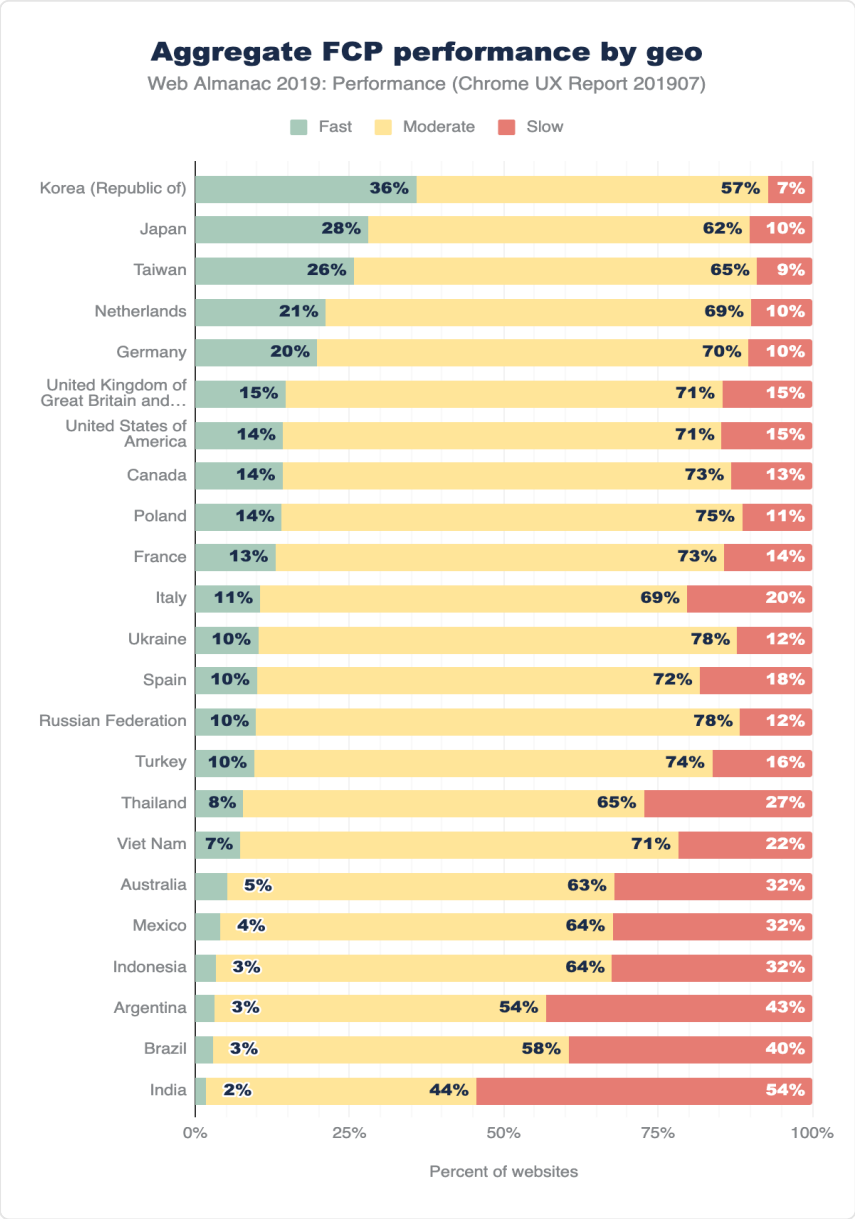


図7.7. 高速、適度、低速FCPでラベル付を行ったWebサイトの分布を地域別に分類したもの。

最後にユーザーの地理（geo）でFCPを切り分けてみましょう。上記のグラフは、個別に多くのWebサイトを持っているトップ23の地域を表しています。これはオープンWeb全体での人気の計測です。アメリカのWebユーザーは、1,211,002の最も際立ったWebサイトにアクセスします。十分に高速なFCP体験のWebサイトの割合で地理をソートしましょう。リストのトップ3にはアジアパシフィック[132](APAC)が入っています。それは韓国、台湾、日本です。この結果から、これらの地域では非常に高速なネットワーク接続[133]が使われていることが説明できます。韓国では高速のFCP基準を満たすウェブサイトが36%あり、低速と評価されているのはわずか7%です。高速/適度/低速のウェブサイトの世界的な分布はおおよそ13/66/20であり、韓国がかなり良い意味で外れ値となっています。

他のAPAC地域の話をしましょう。タイ、ベトナム、インドネシア、インドの高速Webサイトは、ほぼ10%未満です。そして、これらの地域は韓国の3倍以上低速なWebサイトと言う割合になっています。

最初のバイトまでの時間(Time to First Byte)(TTFB)

最初のバイトまでの時間[134]は、ユーザーがWebページにナビゲーションしてからレスポンスの最初のバイトを受信するまでにかかった時間の測定値です。

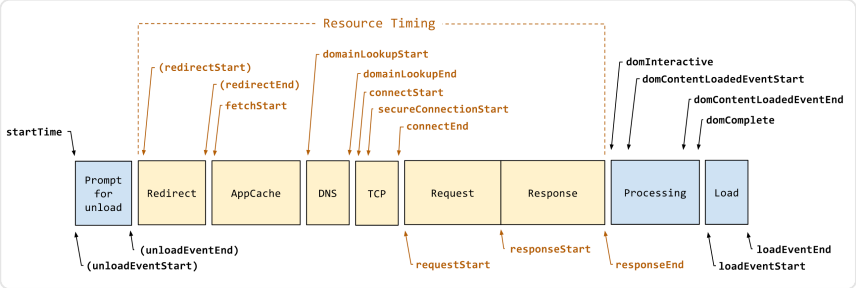


図7.8. ページナビゲーションのイベントとNavigation Timing API の図表。

TTFBとそれに影響する多くの要因を説明するために、Navigation Timing APIの仕様[135]から図を借りました。上の図7.8は `startTime` から `responseStart` までの間を表しており、`unload`、`redirects`、`AppCache`、`DNS`、`SSL`、`TCP` などのサーバー側のリクエスト処理に費やす全てを含んでいます。このようなコンテキストを考慮して、ユーザーがこの数値をどのように体験しているかを見てみましょう。

132. https://en.wikipedia.org/wiki/Asia-Pacific
133. https://en.wikipedia.org/wiki/List_of_countries_by_Internet_connection_speeds
134. https://web.dev/articles/ttfb
135. https://developer.mozilla.org/docs/Web/API/Navigation_timing_API

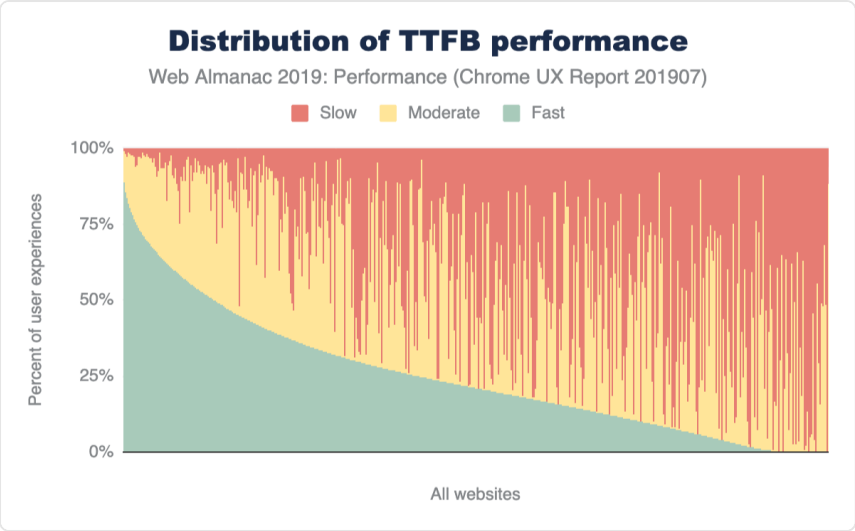


図7.9. WebサイトのTTFBパフォーマンス、高速、適度、低速の分布。

図7.1のFCPチャートと同様に、これは高速TTFB毎に並べられた代表的な1,000個の値のサンプルのビューです。 高速TTFB[136]は0.2秒（200ミリ秒）未満、低速TTFBは1秒以上、その間はすべて適度です。

高速の割合の曲がり方を見ると、形はFCPとかなり異なります。75%を超える高速なTTFBを持つWebサイトは非常に少なく、25%を下回るWebサイトが半分以上となっています。

以前にFCPで使用したPSI方法論からインスピレーションを貰って、TTFB速度のラベルを各Webサイトに適用しましょう。ウェブサイトが75%以上のユーザー体験に対して高速なTTFBを提供する場合、**高速**とラベル付けされます。それ以外に、25%以上のユーザー体験に対して**低速** なTTFBを提供するものを、低速とします。この条件のどちらでもないものを**適度**とします

136. *https://developers.google.com/speed/docs/insights/Server#recommendations*

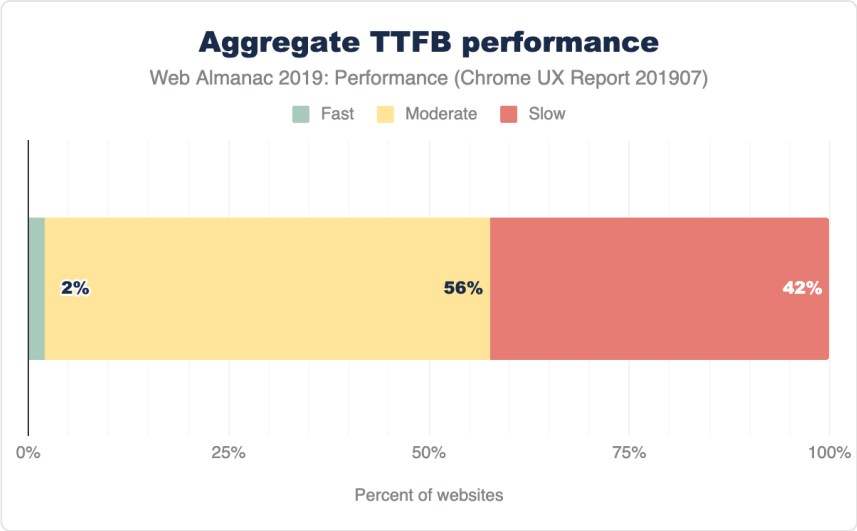


図7.10. TTFBが高速、適度、低速としてラベル付けされたWebサイトの分布。

Webサイトの42%で低速のTTFB体験となっています。TTFBは他のすべてのパフォーマンス値の妨げになるため、この値はとても重要です。*定義上は、TTFBに1秒以上かかる場合、ユーザーは高速なFCPを体験できない可能性があります。*

TTFB by geo

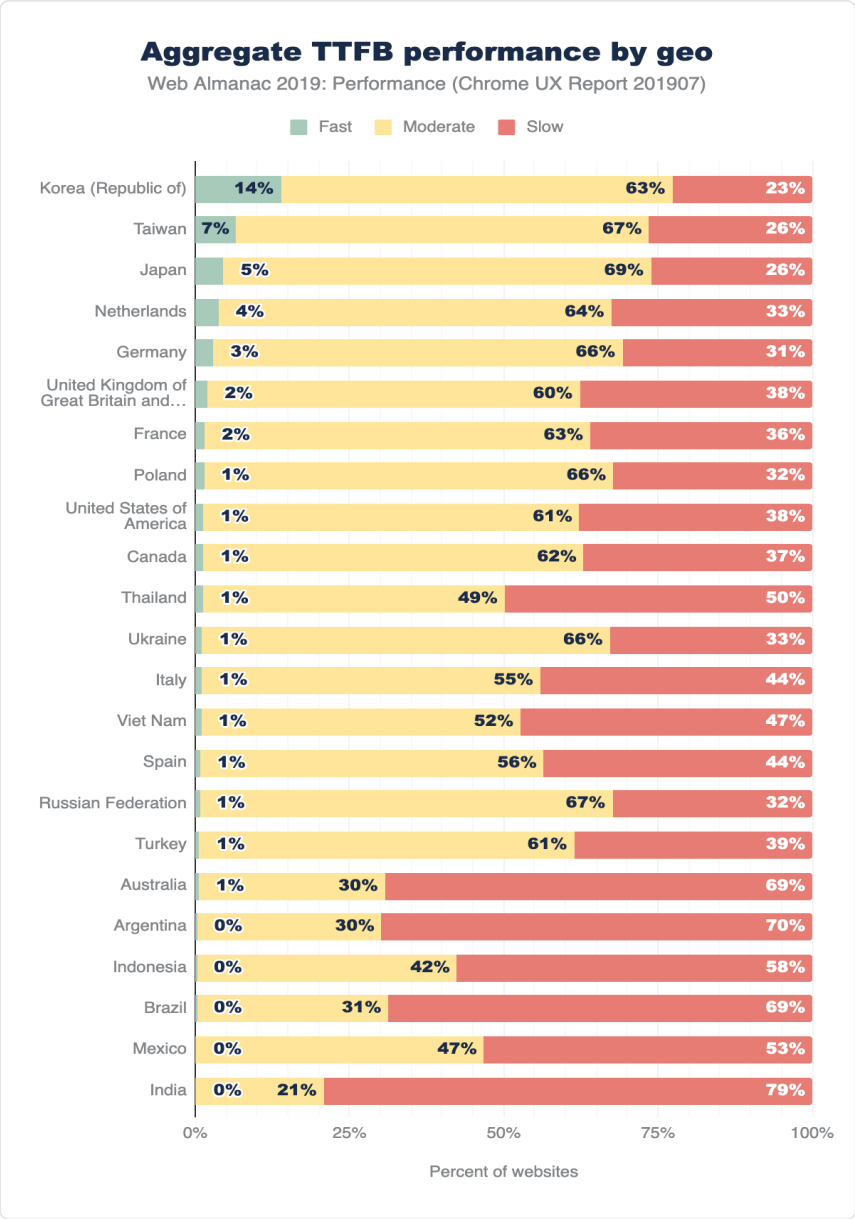


図7.11. 高速、適度、低速のTTFBそれぞれでラベル付けされたWebサイトの分布。地域別に分類されています。

次に、さまざまな地域で、高速なTTFBをユーザーに提供しているWebサイトの割合を見てみましょう。韓国、台湾、日本のようなAPAC地域は依然として世界のユーザーを上回っています。しかし、どの地域も15%を超えてた高速なTTFBとなっているWebサイトはありません。インドでは、高速TTFBとなっているWebサイトは1%未満で、低速なTTFBとなっているWebサイトは79%となっています。

初回入力遅延（First Input Delay）

最後に確認するフィールド値は初回入力遅延(First Input Delay)[137](FID)です。この値は、ユーザーがページのUIを最初に操作してから、ブラウザのメインスレッドでイベントの処理が可能になるまでの時間です。この時間には、アプリケーションへの実際の入力処理の時間は含まれないことに注意してください。最悪の場合は、FIDが遅いとページが応答しなくなり、ユーザー体験は苛立たしいものとなってしまいます。

いくつかのしきい値を定義することから始めましょう。新しいPSI手法によると、**高速**なFIDは100ミリ秒未満です。これによりアプリケーションは、入力イベントを処理しユーザーへの応答の結果が瞬時に感じるのに十分な時間を与えることができます。**低速**なFIDは300ミリ秒以上となっており、その間はすべて**適度**にあたります。

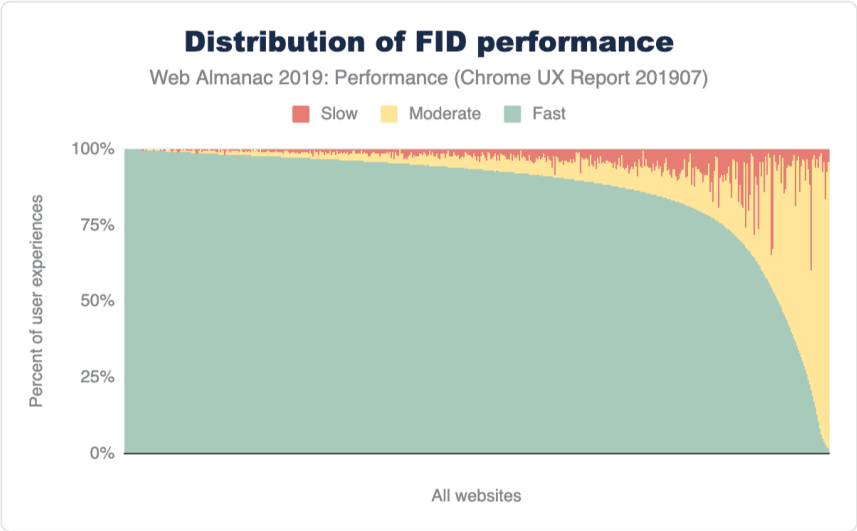


図7.12. Webサイトの高速、適度、低速のFIDパフォーマンスの分布。

貴方はいま難題を抱えています。この表はWebサイトの高速、適度、低速のFID体験の分布を表しています。これは以前のFCPとTTFBの表とは劇的に異なります。(図7.1と図7.9をそ

137. https://developers.google.com/web/updates/2018/05/first-input-delay

れぞれ見る).高速FIDの曲線は100%から75%にゆるやかに下っていき、その後急降下します。FIDの体験は、ほとんどのWebサイトでほぼ高速になっています。

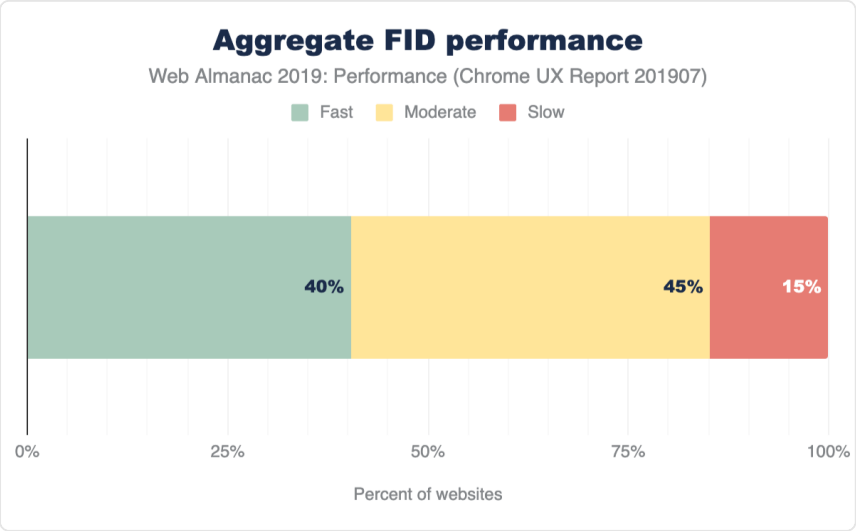


図7.13. 高速、適度、低速のTTFBでラベル付けされたWebサイトの分布。

十分に高速や低速のFIDとなるWebサイトのラベル付けを行うため、PSI方法論はFCPと少しだけ異なる方法を取ります。**高速**なサイトと定義するのは、FID体験の95%以上を高速と定める必要があります。遅いFID体験となる5%のサイトを**遅い**として、そのほかの体験を**適度**とします。

以前の測定と比較して、集計されたFIDパフォーマンスの分布は低速よりも高速および適度の体験に大きく偏っています。Webサイトの40%でFIDが早く、FIDが遅いと言えるのは15%だけです。FIDが対話性の計測であるという性質は、ネットワーク速度によって制限される読み込みの計測と違い、パフォーマンス特性の全く異なる方法になります。

デバイス毎のFID

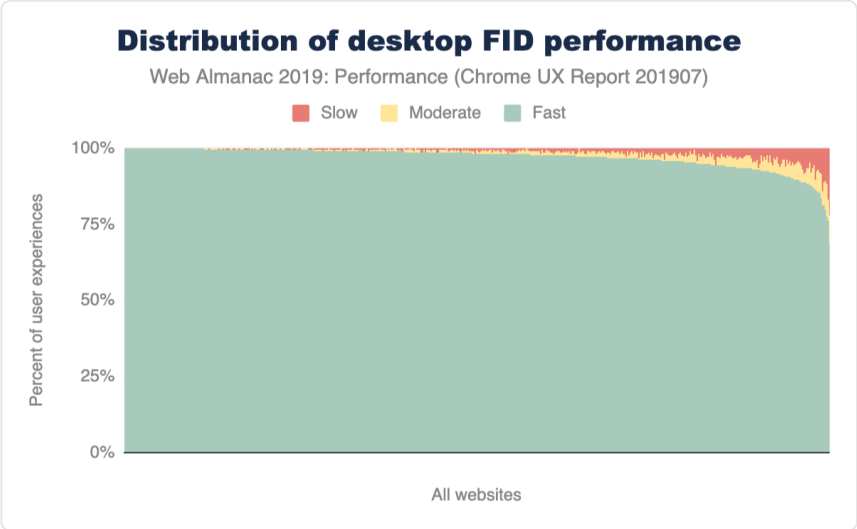


図7.14. デスクトップWebサイトのFIDパフォーマンスの高速、適度、低速の分布。

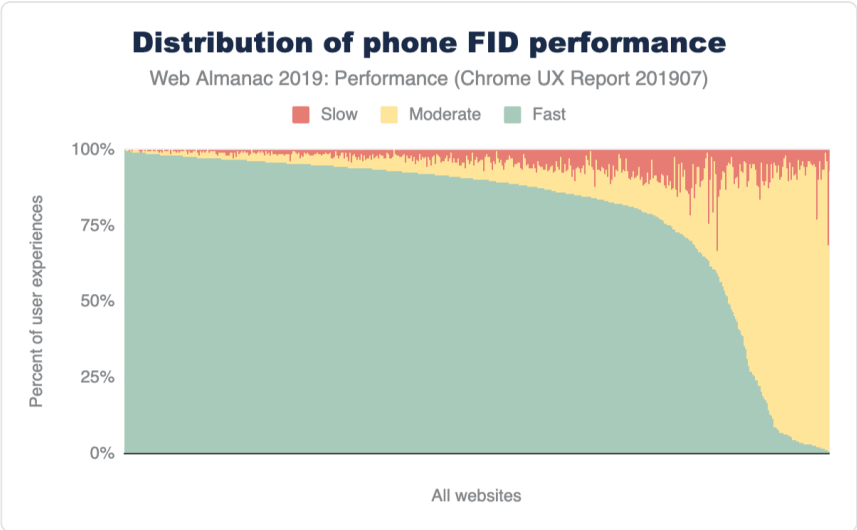


図7.15. 携帯電話向けWebサイトのFIDパフォーマンスの高速、適度、低速の分布。

FIDをデバイスで分類してみると、この2つはまったく別の話となるようです。デスクトップユーザーの殆どは常に高速なFIDで楽しめているようです。まれに遅い体験をさせられるWebサイトがあるかもしれませんが、結果としては圧倒的に高速となっています。一方モバ

イルのユーザーは2種類の体験に大別できます。かなり高速(デスクトップほどではないが)か、全く早くないのどちらかとなるようです。後者はWebサイトの10%のみ体験しているようですが、これは大きな違いでしょう。

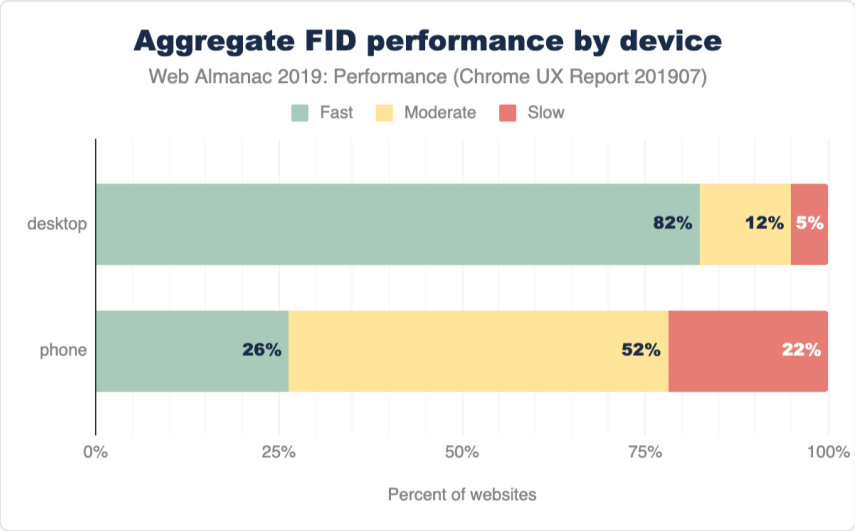


図7.16. 高速、適度、低速FIDとしてラベル付けされたWebサイトの分布。デバイス種類別に分類されています。

PSIラベルをデスクトップと携帯電話の体験に適用してみると、差分が非常に明瞭になります。デスクトップユーザーが経験するWebサイトの82%はFIDが高速であるのに対し、低速は5%だけです。モバイルで体験するWebサイトは、26%が高速であり、22%が低速です。フォームファクターは、FIDなどの対話性計測の成果に大きな役割を果たします。

有効な接続タイプ別のFID

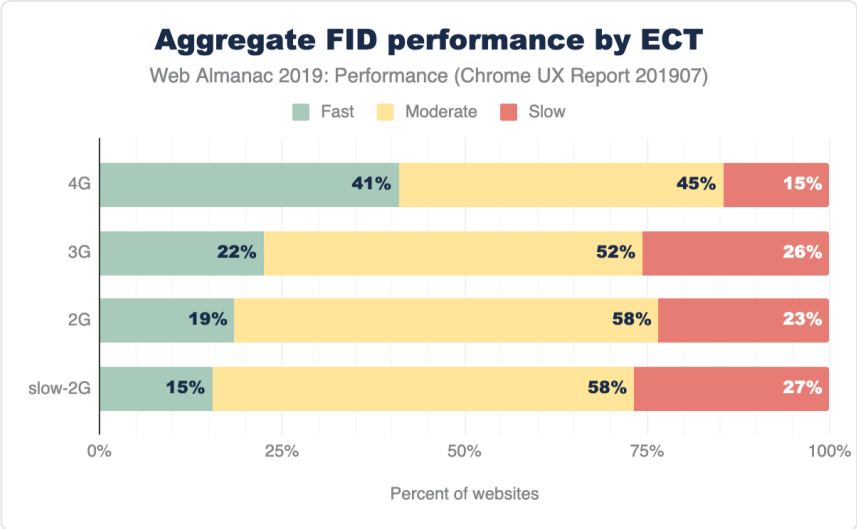


図7.17. ECTによって分類された高速、適度、低速FIDとしてラベル付けされたWebサイトの分布

一見、FIDはCPUの動作速度が影響するように思えます。性能の悪いデバイスを使うと、ユーザーがWebページを操作しようとしたときに待ち状態になる可能性が高いと考えるのは自然でしょうか？

上記のECTの結果からは、接続速度とFIDパフォーマンスの間に相関関係があることが示されています。ユーザーの有効な接続の速度が低下すると、高速なFIDを体験するWebサイトの割合も低下しています。4GのECTを使うユーザーがアクセスするWebサイトの41%は高速なFIDで、3Gは22%、2Gは19%、低速な2Gは15%です。

FID by geo

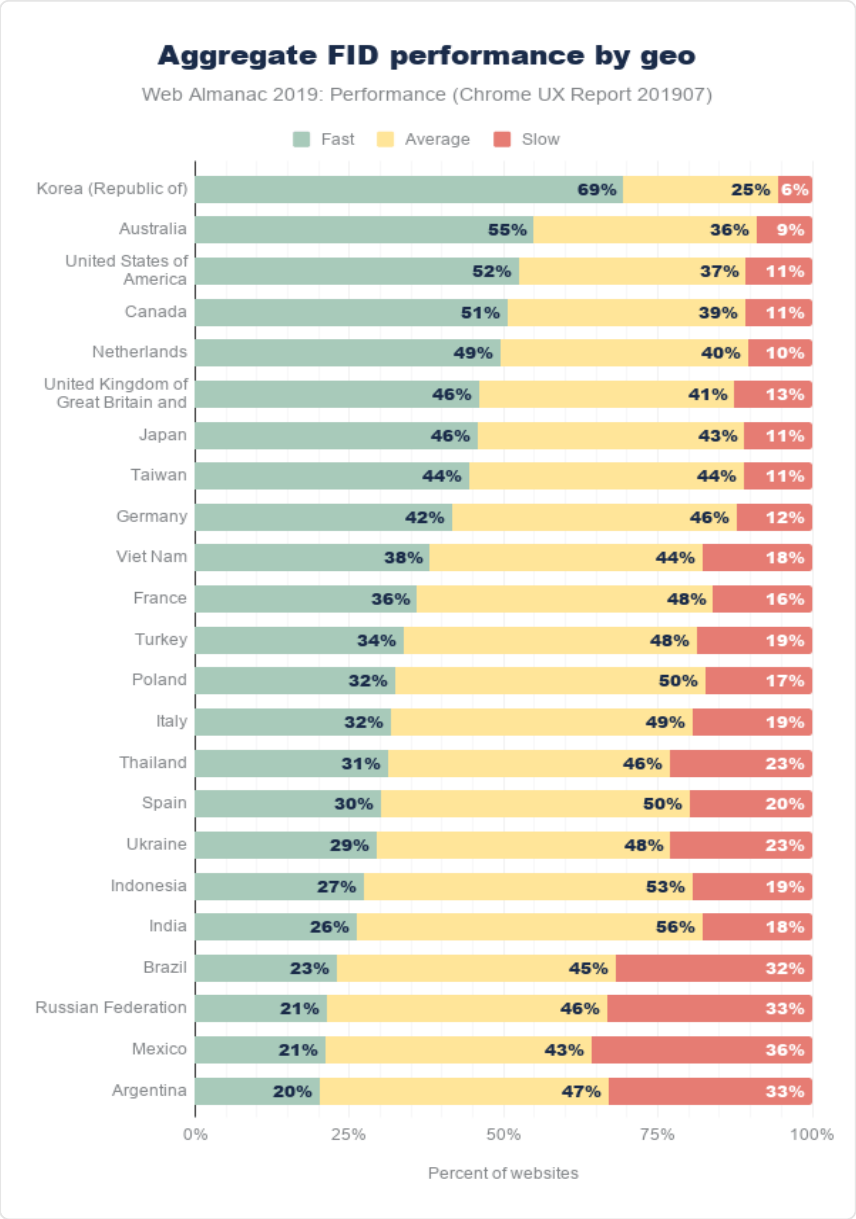


図7.18. 高速、適度、低速FIDでラベル付けされたWebサイトの分布を地域別に分類したもの。

この地理的な位置によるFIDの内訳では、韓国はまたもや他のすべてよりも抜きん出ています。しかし、トップを占める地域にはいくつか新しい顔ぶれが現れています。次に現れるのはオーストラリア、米国、カナダとなっており、50%以上のWebサイトが高速なFIDとなっています。

他の地域固有の結果と同様に、ユーザー体験に影響を与える可能性のある要因は多数あるでしょう。例えば、より裕福な地理的条件が揃う地域に住んでいる人々は、より高速なネットワーク・インフラを購入でき、デスクトップも携帯電話もお金をかけてハイエンドなものを持っている率は高くなる可能性があります。

結論

Webページの読み込み速度を定量化することは、単一の計測では表すことのできない不完全な科学です。従来の `onload` 等を計測する計測する方法は、ユーザー体験とは関係のない部分まで計測してしまい、本当に抑えるべき点を見逃してしまう可能性があります。FCPやFIDなどのユーザーの知覚に相当する計測は、ユーザーが見たり感じたりする内容を着実に伝達できます。ただそれでも、どちらの計測も単独で見てはページ全体の読み込み体験が高速なのか低速かについての結論を導き出すことはできません。多くの計測値を総合的に見ることでのみ、Webサイト個々のパフォーマンスとWebの状態を理解することができます。

この章で表されたデータから、高速なWebサイトとにするためには多くの設定されるべき目標と作業があることを示しています。確かなフォームファクター、効果的な接続の種類、そして地理にはユーザー体験の向上と相関しますが、低いパフォーマンスとなる人口の統計も組み合わせる必要があることを忘れてはいけません。殆どの場合、Webプラットフォームはビジネスで使われています。コンバージョン率を改善してより多くのお金を稼ぐことは、Webサイトを高速化する大きな動機になるでしょう。最終的に、すべてのWebサイトのパフォーマンスとは、ユーザーの邪魔をしたり、イラつかせたり、怒らせたりしない方法で、ユーザーにより良い体験を提供することです。

Webがまた一つ古くなり、ユーザー体験を測定する能力が徐々に向上するにつれて、開発者がより総合的なユーザー体験を捉えて計測された値を身近に思えるようになることを楽しみにしています。FCPは有用なコンテンツをユーザーに表示するタイムラインのほぼ最初部分であり、それ以外にもLarge Contentful Paint[138]（LCP）と呼ばれる新しい計測値が出現して、ページの読み込みがどのように認識されるかの可視性が向上しています。Layout Instability API[139]は、ページの読み込み以降でユーザーが不満を持つ体験がある事を新たに垣間見せてくれました。

こういった新しい計測が出来るようになった2020年のWebは、さらに透明性が高まって理解が深まり、開発者がパフォーマンスを改善するための有意義な進歩を遂げることで、より

138. *https://web.dev/articles/lcp*
139. *https://web.dev/articles/cls*

良いユーザー体験を提供できるでしょう。

著者

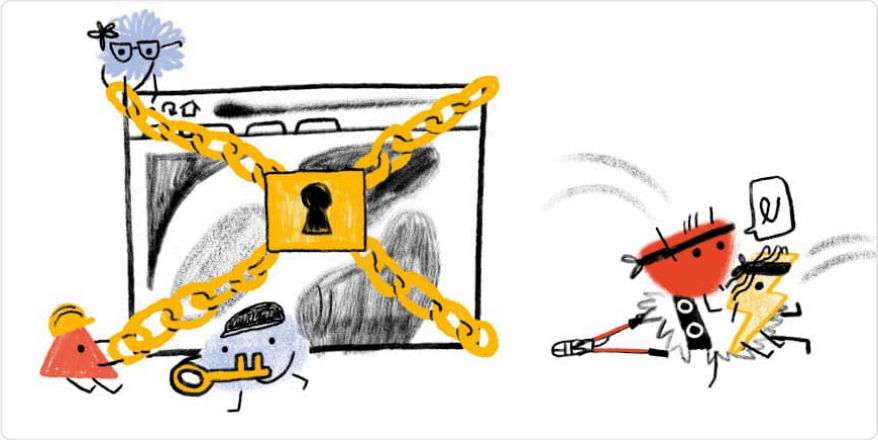
Rick Viscomi

@rick_viscomi rviscomi https://rviscomi.dev/

Rick ViscomiはGoogle のシニア開発プログラムエンジニアで、HTTP ArchiveやChrome UX Reportなどのウェブ透過性プロジェクトに携わり、ウェブサイトの構築方法と体験の交差点を研究しています。Rickは、The State of the Web[140]のホストを務めており、専門家がウェブのトレンドについて議論しています。Rickは、ウェブのパフォーマンスをテストするためのガイドである Using WebPageTest[141] の共著者であり、dev.to[142] でウェブについて頻繁に執筆していますし、@rick_viscomi の Twitter でもウェブについて書いています。

140. *https://www.youtube.com/playlist?list=PLNYkxOF6rcIBGvYSYO-VxOsaYQDw5rifJ*
141. *https://usingwpt.com*
142. *https://dev.to/rick_viscomi*

部 II 章 8
セキュリティ



Scott Helme と Artur Janc によって書かれた。
Barry Pollard、 Alessandro Ghedini と Paul Calvano によってレビュ。
Andrew Galloni と Jason Haralson 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上のセキュリティの現状を見ていきます。オンラインでのセキュリティとプライバシーの重要性がますます高まる中、サイト運営者とユーザーを保護するための機能が増えています。ここでは、ウェブ上でのこれらの新機能の採用状況を見ていきます。

トランスポートレイヤーセキュリティ

現在、オンラインでのセキュリティとプライバシーを向上させるための最大の後押しは、おそらくトランスポート・レイヤー・セキュリティ（TLS）の普及です。TLS（または古いバージョンのSSL）は、HTTPSの「S」を提供し、安全でプライベートなWebサイトのブラウジン

グを可能にするプロトコルです。ウェブ上でのHTTPSの使用が大幅に増加している[143]だけでなく、TLSv1.2やTLSv1.3のような最新バージョンのTLSが増加していることも重要です。

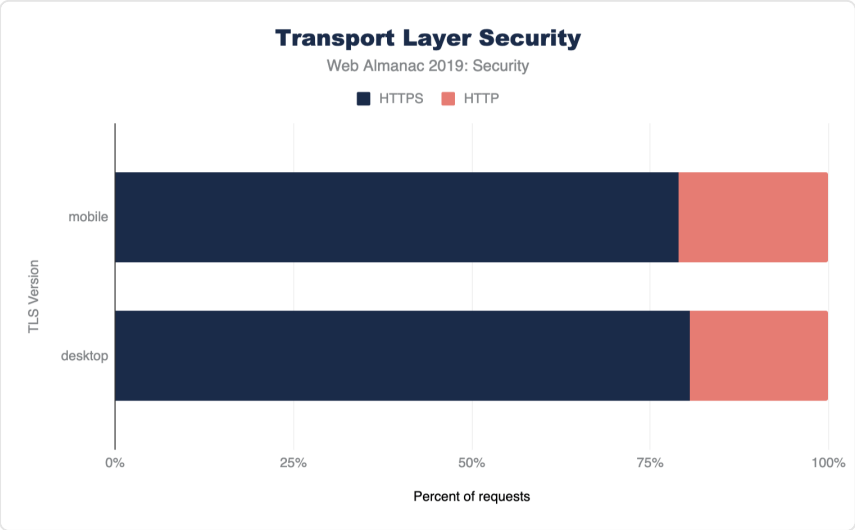


図8.1. HTTPとHTTPS の使用法。

143. *https://httparchive.org/reports/state-of-the-web#pctHttps*

プロトコルのバージョン

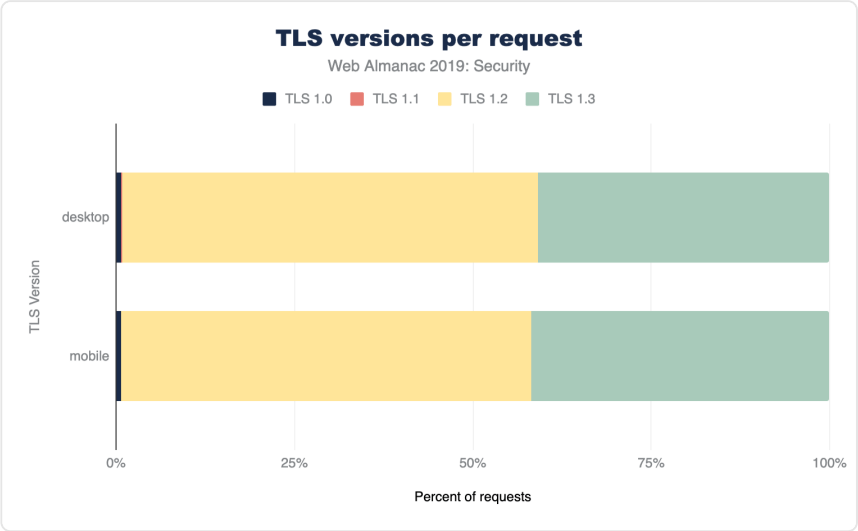


図8.2. TLSプロトコルのバージョン使用状況

図8.2は、さまざまなプロトコルバージョンのサポートを示しています。TLSv1.0やTLSv1.1のようなレガシーなTLSバージョンの使用は最小限であり、ほとんどすべてのサポートはプロトコルの新しいバージョンであるTLSv1.2やTLSv1.3に対応しています。TLSv1.3はまだ標準としては非常に若いですが（TLSv1.3は2018年8月[144]に正式に承認されたばかりです）、TLSを使用するリクエストの40%以上が最新バージョンを使用しています！　TLSv1.0やTLSv1.1のようなレガシーバージョンの使用はほとんどありません。

これは、多くのサイトがサードパーティコンテンツのために大きなプレイヤーからのリクエストを使用していることが原因であると考えられます。例えば、どのようなサイトでもGoogle Analytics、Google AdWords、またはGoogle FontsをロードしGoogleのような大規模なプレイヤーは通常新しいプロトコルのためのアーリーアダプターです。

ホームページだけを見て、それ以外のサイトのリクエストをすべて見ない場合、TLSの使用率は予想通りかなり高いですが、WordpressのようなCMSサイトやCDNのようなサイトが原因である可能性は高いです。

144. https://tools.ietf.org/html/rfc8446

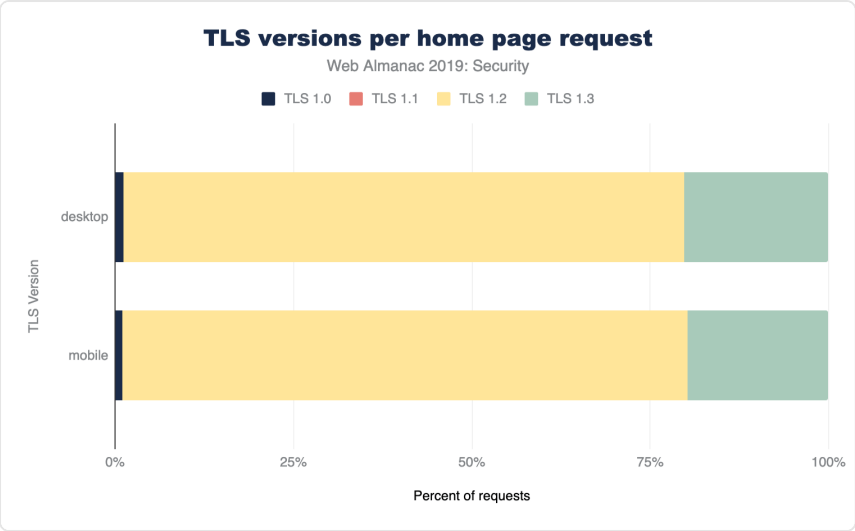


図8.3. ホームページリクエストだけのTLSプロトコルバージョン使用状況。

一方で、Web Almanacが使用している方法論は、大規模サイトの利用状況を*過小評価*します。なぜなら、大規模サイトはそのサイト自体が現実世界ではより大きなインターネット・トラフィックを形成している可能性が高いにもかかわらず、これらの統計のために一度しかクロールされないからです。

証明書発行者

もちろん、ウェブサイトでHTTPSを使用するには、認証局（CA）からの証明書が必要です。HTTPSの使用の増加に伴い、CAとその製品／サービスの使用も増加しています。ここでは、証明書を使用するTLSリクエストの量に基づいて、上位10社の証明書発行者を紹介します。

発行証明書発行局	デスクトップ	モバイル
Google Internet Authority G3	19.26%	19.68%
Let's Encrypt Authority X3	10.20%	9.19%
DigiCert SHA2 High Assurance Server CA	9.83%	9.26%
DigiCert SHA2 Secure Server CA	7.55%	8.72%
GTS CA 1O1	7.87%	8.43%
DigiCert SHA2 Secure Server CA	7.55%	8.72%
COMODO RSA Domain Validation Secure Server CA	6.29%	5.79%
Go Daddy Secure Certificate Authority - G2	4.84%	5.10%
Amazon	4.71%	4.45%
COMODO ECC Domain Validation Secure Server CA 2	3.22%	2.75%

図8.4. 使用されている認証局トップ10。

前述したように、Googleのボリュームは他のサイトでGoogleアナリティクス、Google Adwords、またはGoogle Fontsを繰り返し使用していることを反映している可能性が高い。

Let's Encrypt[145]の台頭は2016年初頭の開始後、急成長を遂げ、それ以来世界でもトップレベルの証明書発行局の1つになりました。無料の証明書の可用性と自動化されたツールは、ウェブ上でのHTTPSの採用に決定的に重要な役割を果たしています。Let's Encryptは、これらの両方において重要な役割を果たしています。

コストの削減により、HTTPSへの参入障壁は取り除かれましたが、Let's Encryptが使用する自動化は証明書の寿命を短くできるため、長期的にはより重要であると思われます、これは多くのセキュリ ティ上のメリットがあります[146]。

認証キーの種類

HTTPSを使用するという重要な要件と並行して、適切な構成を使用するという要件もあります。非常に多くの設定オプションと選択肢があるため、これは慎重にバランスを取る必要があります。

まず、認証に使用される鍵について見ていきましょう。従来、証明書はRSAアルゴリズムを

145. https://letsencrypt.org/ja/
146. https://scotthelme.co.uk/why-we-need-to-do-more-to-reduce-certificate-lifetimes/

使用した鍵に基づいて発行されてきましたが、より新しく優れたアルゴリズムであるECDSA(Elliptic Curve Digital Signature Algorithm — 楕円曲線DSA) を使用しており、RSAアルゴリズムよりも優れた性能を発揮する小さな鍵の使用を可能にしています。私たちのクロールの結果を見ると、ウェブの大部分がRSAを使用していることがわかります。

キーの種類	*デスクトップ*	*モバイル*
RSA Keys	*48.67%*	*58.8%*
ECDA Keys	*21.47%*	*26.41%*

図8.5. 使用する認証キーの種類

ECDSA鍵はより強力な鍵であるため、より小さな鍵の使用が可能となりRSA鍵よりも優れたパフォーマンスを発揮しますが、下位互換性に関する懸念やその間の両方のサポートの複雑さが一部のウェブサイト運営者の移行を妨げる要因となっています。

Forward secrecy

Forward secrecy[147]は将来サーバの秘密鍵が漏洩した場合でも、サーバへの各接続が公開されるのを防ぐような方法で接続を保護するいくつかの鍵交換メカニズムの特性です。これは、接続のセキュリティを保護するために全てのTLS接続で望ましい事として、セキュリティコミュニティ内ではよく理解されています。2008年にTLSv1.2でオプション設定として導入され、2018年にはTLSv1.3でForward Secrecyの使用が必須となりました。

Forward Secrecyを提供するTLSリクエストの割合を見ると、サポートが非常に大きいことがわかります。デスクトップの96.92%、モバイルリクエストの96.49%がForward Secrecyを使用しています。TLSv1.3の採用が継続的に増加していることから、これらの数字はさらに増加すると予想されます。

暗号スイート

TLSでは、さまざまな暗号スイートを使用できます。従来、TLSの新しいバージョンは暗号スイートを追加してきましたが、古い暗号スイートを削除することには消極的でした。TLSv1.3はこれを単純化するために、より少ない暗号スイートのセットを提供し、古い安全でない暗号スイートを使用することを許可しません。SSL Labs[148] のようなツールは、ウェブサイトのTLS設定 (サポートされている暗号スイートとその好ましい順序を含む) を簡単に見ることができ、より良い設定を促進するのに役立ちます。TLSリクエストのためにネゴシエートされた暗号化スイートの大部分は確かに優れたものであったことがわかります。

147. *https://ja.wikipedia.org/wiki/Forward_secrecy*
148. *https://www.ssllabs.com/*

暗号スイート	デスクトップ	モバイル
AES_128_GCM	75.87%	76.71%
AES_256_GCM	19.73%	18.49%
AES_256_CBC	2.22%	2.26%
AES_128_CBC	1.43%	1.72%
CHACHA20_POLY1305	0.69%	0.79%
3DES_EDE_CBC	0.06%	0.04%

図8.6. 使用されている暗号スイートの使用法

古いCBC暗号は安全性が低いので、GCM暗号がこのように広く使われるようになったのはポジティブなことです。CHACHA20_POLY1305[149]はまだニッチな暗号スイートであり、私たちはまだ安全でないトリプルDES暗号[150]をごくわずかしか使っていません。

これらはChromeを使ったクロールに使われた暗号化スイートですが、サイトは古いブラウザでも他の暗号化スイートをサポートしている可能性が高いことに注意してください。例えばSSL Pulse[151] などの他の情報源では、サポートされているすべての暗号スイートとプロトコルの範囲についてより詳細な情報を提供しています。

混合コンテンツ

ウェブ上のほとんどのサイトは元々HTTPサイトとして存在しており、HTTPSにサイトを移行しなければなりませんでした。この「リフトアンドシフト」作業は難しく、時には見落としたり、取り残されたりすることもあります。その結果、ページはHTTPSで読み込まれているが、ページ上の何か（画像やスタイルなど）はHTTPで読み込まれているような、コンテンツが混在しているサイトが発生します。コンテンツが混在していると、セキュリティやプライバシーに悪影響を及ぼし、発見して修正するのが困難になることがあります。

149. https://blog.cloudflare.com/it-takes-two-to-chacha-poly/
150. https://ja.wikipedia.org/wiki/%E3%83%88%E3%83%AA%E3%83%97%E3%83%ABDES#%E5%AE%89%E5%85%A8%E6%80%A7
151. https://www.ssllabs.com/ssl-pulse/

混合コンテンツタイプ	デスクトップ	モバイル
任意のコンテンツが混在しているページ	16.27%	15.37%
アクティブな混合コンテンツのページ	3.99%	4.13%

図8.7. 混在コンテンツの利用状況。

モバイル（645,485サイト）とデスクトップ（594,072サイト）では、約20%のサイトが何らかの形で混合コンテンツを表示していることがわかります。画像のようなパッシブな混合コンテンツの危険性は低いですが、混合コンテンツを持つサイトのほぼ4分の1がアクティブな混合コンテンツを持っていることがわかります。JavaScriptのようなアクティブな混合コンテンツは、攻撃者が自分の敵対的なコードを簡単にページに挿入できるため、より危険です。

これまでのウェブブラウザは、受動的な混合コンテンツを許可して警告を表示していたが、能動的な混合コンテンツはブロックしていた。しかし最近では、Chrome発表[152] はこの点を改善し、HTTPSが標準になるにつれて代わりにすべての混合コンテンツをブロックすることを意図しています。

セキュリティヘッダ

サイト運営者がユーザーをより良く保護するための多くの新しい機能が、ブラウザに組み込まれたセキュリティ保護を設定したり制御したりできる新しいHTTPレスポンスヘッダの形で提供されています。これらの機能の中には、簡単に有効にして大きなレベルの保護を提供するものもあれば、サイト運営者が少し作業を必要とするものもあります。サイトがこれらのヘッダを使用しており、正しく設定されているかどうかを確認したい場合は、Security Headers[153]ツールを使用してスキャンできます。

152. https://blog.chromium.org/2019/10/no-more-mixed-messages-about-https.html
153. https://securityheaders.com/

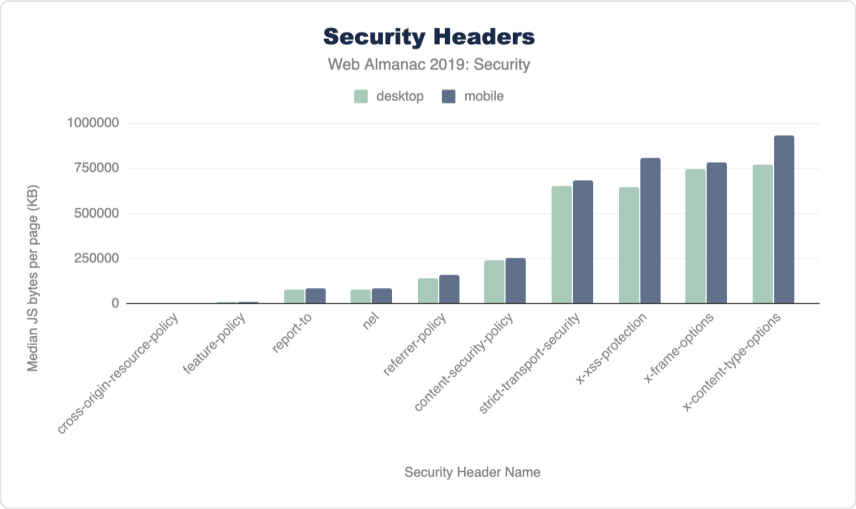


図8.8. セキュリティヘッダの使用法

HTTP Strict Transport Security

HSTS[154] ヘッダーは、Webサイトがブラウザに、安全なHTTPS接続でのみサイトと通信するように指示することを可能にします。これは、http:// URLを使用しようとする試みは、リクエストが行われる前に自動的にhttps://に変換されることを意味します。リクエストの40%以上がTLSを使用できることを考えると、要求するようにブラウザに指示しているリクエストの割合はかなり低いと考えられます。

HSTS *ディレクティブ*	*デスクトップ*	*モバイル*
`max-age`	14.80%	12.81%
`includeSubDomains`	3.86%	3.29%
`preload`	2.27%	1.99%

図8.9. HSTS ディレクティブの使用法

モバイルページやデスクトップページの15%未満が `max-age` ディレクティブ付きのHSTSを発行しています。これは有効なポリシーの最低条件です。また、 `includeSubDomains` ディレクティブでサブドメインをポリシーに含めているページはさらに少なく、HSTSのプリロ

154. https://tools.ietf.org/html/rfc6797

ードを行っているページはさらに少ないです。HSTSの `max-age` の中央値を見ると、これを使用している場合はデスクトップとモバイルの両方で15768000となっており、半年(60X60X24X365/2)に相当する強力な設定であることがわかります。

	クライアント	
パーセンタイル	デスクトップ	モバイル
10	300	300
25	7889238	7889238
50	15768000	15768000
75	31536000	31536000
90	63072000	63072000

図8.10. HSTSの `max-age` ポリシーのパーセンタイル別の中値。

HSTSプリロード

HTTPレスポンスヘッダーを介して配信されるHSTSポリシーでは、初めてサイトを訪れたときに、ブラウザはポリシーが設定されているかどうかを知ることができません。この初回使用時の信頼[155]の問題を回避するために、サイト運営者はブラウザ(または他のユーザーエージェント)にポリシーをプリロードしておくことができます。

プリロードにはいくつかの要件があり、HSTSプリロード[156]サイトで概要が説明されています。現在の基準では、デスクトップでは0.31%、モバイルでは0.26%というごく少数のサイトしか対象としていないことがわかる。サイトは、ドメインをプリロードするために送信する前、ドメインの下にあるすべてのサイトをHTTPSに完全に移行させておく必要があります。

コンテンツセキュリティポリシー

ウェブアプリケーションは、敵対的なコンテンツがページへ挿入される攻撃に頻繁に直面しています。最も心配なコンテンツはJavaScriptであり、攻撃者がJavaScriptをページに挿入する方法を見つけると、有害な攻撃を実行できます。これらの攻撃はクロスサイトスクリプティング(XSS)[157]として知られており、コンテンツセキュリティポリシー(CSP)[158] はこれらの攻撃

155. *https://en.wikipedia.org/wiki/Trust_on_first_use*
156. *https://hstspreload.org/*
157. *https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%82%B9%E3%82%B5%E3%82%A4%E3%83%88%E3%82%B9%E3%82%AF%E3%83%AA%E3%83%97%E3%83%86%E3%82%A3%E3%83%B3%E3%82%*
158. *https://www.w3.org/TR/CSP2/*

に対する効果的な防御策を提供しています。

CSPとは、ウェブサイトが公開しているHTTPヘッダ(`Content-Security-Policy`)のことで、サイトで許可されているコンテンツに関するルールをブラウザに伝えるものです。セキュリティ上の欠陥のために追加のコンテンツがサイトに注入され、それがポリシーで許可されていない場合、ブラウザはそのコンテンツの使用をブロックします。XSS保護の他にも、CSPは、HTTPSへの移行を容易にするなど、いくつかの重要な利点を提供しています。

CSPの多くの利点にもかかわらず、その非常に目的がページ上で許容されるものを制限することであるため、ウェブサイトに実装することは複雑になる可能性があります。ポリシーは必要なすべてのコンテンツやリソースを許可しなければならず、大きく複雑になりがちです。レポートURI[159]のようなツールは、適切なポリシーを分析して構築するのに役立ちます。

デスクトップページのわずか5.51%にCSPが含まれ、モバイルページのわずか4.73%にCSPが含まれていることがわかりましたが、これは展開の複雑さが原因と思われます。

ハッシュ/ノンス

CSPの一般的なアプローチは、JavaScriptなどのコンテンツをページにロードすることを許可されているサードパーティドメインのホワイトリストを作成することです。これらのホワイトリストの作成と管理は困難な場合があるため、ハッシュ[160]とノンス[161]が代替的なアプローチとして導入されました。ハッシュはスクリプトの内容に基づいて計算されるので、ウェブサイト運営者が公開しているスクリプトが変更されたり、別のスクリプトが追加されたりするとハッシュと一致せずブロックされてしまいます。ノンスは、CSPによって許可され、スクリプトにタグが付けられているワンタイムコード(ページが読み込まれるたびに変更され推測されるのを防ぐ必要があります)です。このページのノンスの例は、ソースを見てGoogle Tag Managerがどのように読み込まれているかを見ることで見ることができます。

調査対象となったサイトのうち、ノンスソースを使用しているのはデスクトップページで0.09%、ハッシュソースを使用しているのはデスクトップページで0.02%にとどまっている。モバイルページではノンスソースを使用しているサイトは0.13%とやや多いが、ハッシュソースの使用率は0.01%とモバイルページの方が低い。

`strict-dynamic`

CSP[162]の次のイテレーションにおける`strict-dynamic`の提案は、ホワイトリスト化されたスクリプトがさらにスクリプトの依存性をロードできるようにすることで、CSPを使用する

159. https://report-uri.com/
160. https://www.w3.org/TR/CSP2/#source-list-valid-hashes
161. https://www.w3.org/TR/CSP2/#source-list-valid-nonces
162. https://www.w3.org/TR/CSP3/

ためのサイト運営者の負担をさらに軽減します。すでにいくつかの最新ブラウザでサポート[163]されているこの機能の導入にもかかわらず、ポリシーにこの機能を含めるのは、デスクトップページの0.03%とモバイルページの0.1%にすぎません。

`trusted-types`

XSS攻撃には様々な形がありますが、Trusted-Types[164]はDOM-XSSに特化して作られました。効果的なメカニズムであるにもかかわらず、私たちのデータによると、モバイルとデスクトップの2つのページだけがTrusted-Typesディレクティブを使用しています。

`unsafe inline`と`unsafe-eval`

CSPがページにデプロイされると、インラインスクリプトや`eval()`の使用のような特定の安全でない機能は無効化されます。ページはこれらの機能に依存し、安全な方法で、おそらくノンスやハッシュソースを使ってこれらの機能を有効にできます。サイト運営者は、`unsafe-inline`や`unsafe-eval`を使って、これらの安全でない機能をCSPで再有効にすることもできますが、その名前が示すようにそうすることでCSPが提供する保護の多くを失うことになります。CSPを含むデスクトップページの5.51%のうち、33.94%が`unsafe-inline`を、31.03%が`unsafe-eval`を含んでいます。モバイルページでは、CSPを含む4.73%のうち、34.04%が`unsafe-inline`を使用し、31.71%が`unsafe-eval`を使用していることがわかります。

`upgrade-insecure-requests`

先に、サイト運営者がHTTPからHTTPSへの移行で直面する共通の問題として、一部のコンテンツがHTTPSページのHTTP上に誤って読み込まれてしまう可能性があることを述べました。この問題は混合コンテンツとして知られており、CSPはこの問題を解決する効果的な方法を提供します。upgrade-insecure-requests'ディレクティブは、ブラウザにページ上のすべてのサブリソースを安全な接続で読み込むように指示し、例としてHTTPリクエストをHTTPSリクエストに自動的にアップグレードします。ページ上のサブリソースのためのHSTSのようなものと考えてください。

先に図8.7で示したように、デスクトップで調査したHTTPSページのうち、16.27%のページが混合コンテンツを読み込んでおり、3.99%のページがJS/CSS/fontsなどのアクティブな混合コンテンツを読み込んでいることがわかる。モバイルページでは、HTTPSページの15.37%が混合コンテンツを読み込み、4.13%がアクティブな混合コンテンツを読み込みました。HTTP上でJavaScriptなどのアクティブなコンテンツを読み込むことで、攻撃者は簡単

163. *https://caniuse.com/#feat=mdn-http_headers_csp_content-security-policy_strict-dynamic*
164. *https://github.com/w3c/webappsec-trusted-types*

に敵対的なコードをページに注入して攻撃を開始できます。これは、CSPの `upgrade-insecure-requests` ディレクティブが防御しているものです。

`upgrade-insecure-requests` ディレクティブは、デスクトップページの3.24%とモバイルページの2.84%のCSPに含まれており、採用が増えることで大きな利益が得られることを示しています。以下のようなポリシーで、幅広いカテゴリをホワイトリスト化し、 `unsafe-inline` や `unsafe-eval` を含めることで、完全にロックダウンされたCSPや複雑さを必要とせずに比較的簡単に導入できます。

```
Content-Security-Policy: upgrade-insecure-requests; default-src
https:
```

`frame-ancestors`

クリックジャッキング[165]として知られているもう1つの一般的な攻撃は、敵対するウェブサイトのiframeの中にターゲットのウェブサイトを配置し、自分たちがコントロールしている隠しコントロールやボタンをオーバーレイする攻撃者によって行われます。 `X-Frame-Options` ヘッダー(後述)はもともとフレームを制御することを目的としていましたが、柔軟性がなく、CSPの `frame-ancestors` はより柔軟なソリューションを提供するために介入しました。サイト運営者は、フレーム化を許可するホストのリストを指定できるようになり、他のホストがフレーム化しようとするのを防ぐことができるようになりました。

調査したページのうち、デスクトップページの2.85%がCSPで `frame-ancestors` ディレクティブを使用しており、デスクトップページの0.74%がframe-Ancestorsを `'none'` に設定してフレーミングを禁止し、0.47%のページが `frame-ancestors` を `'self'` に設定して自分のサイトのみがフレーミングできるようにしています。モバイルでは2.52%のページが `frame-ancestors` を使用しており、0.71%が `'none'` を設定し、0.41%が `self'` を設定しています。

参照元ポリシー

`Referrer-Policy` ヘッダーは、ユーザーが現在のページから離れた場所へ移動したとき、 `Refererer` ヘッダーにどのような情報を送るかをサイトが制御することを可能とします。これは、検索クエリやURLパラメータに含まれるその他のユーザー依存情報など、URLに機密データが含まれている場合、情報漏洩の原因となる可能性があります。 `Referer` ヘッダで送信される情報を制御し、理想的には制限することで、サイトはサードパーティに送信さ

165. *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*

れる情報を減らすことで訪問者のプライバシーを保護できます。

リファラーポリシーは`Refererer`ヘッダのスペルミスこれはよく知られたエラーとなっています[166]に従っていないことに注意してください。

デスクトップページの3.25%とモバイルページの2.95%が`Referrerer-Policy`ヘッダを発行しています。

設定	*デスクトップ*	*モバイル*
`no-referrer-when-downgrade`	*39.16%*	*41.52%*
`strict-origin-when-cross-origin`	*39.16%*	*22.17%*
`unsafe-url`	*22.17%*	*22.17%*
`same-origin`	*7.97%*	*7.97%*
`origin-when-cross-origin`	*6.76%*	*6.44%*
`no-referrer`	*5.65%*	*5.38%*
`strict-origin`	*4.35%*	*4.14%*
`origin`	*3.63%*	*3.23%*

図8.11. `Referrer-Policy` 設定オプションの使用法。

この表はページによって設定された有効な値を示しており、このヘッダーを使用するページのうち、デスクトップでは99.75%、モバイルでは96.55%のページが有効なポリシーを設定していることがわかる。最も人気のある設定は`no-referrer-when-downgrade`で、これはユーザがHTTPSページからHTTPページに移動する際`Refererer`ヘッダが送信されないようにするものです。2番目に人気のある選択は`strict-origin-when-cross-origin`で、これはスキームのダウングレード(HTTPSからHTTPナビゲーション)時に情報が送信されるのを防ぎ、`Refererer`で情報が送信される際にはソースのオリジンのみを含み、完全なURLは含まれません(例えば、`https://www.example.com/page/`ではなく`https://www.example.com`)。その他の有効な設定の詳細は、Referrerer Policy specification[167]に記載されています、`unsafe-url`の多用はさらなる調査を必要としますが、アナリティクスや広告ライブラリのようなサードパーティコンポーネントである可能性が高いです。

166. *https://stackoverflow.com/questions/3087626/was-the-misspelling-of-the-http-field-name-referer-intentional*
167. *https://www.w3.org/TR/referrer-policy/#referrer-policies*

機能方針

ウェブプラットフォームがより強力で機能も豊富になるにつれ、攻撃者はこれらの新しいAPIを興味深い方法で悪用できるようになります。強力なAPIの悪用を制限するために、サイト運営者は `Feature-Policy` ヘッダを発行して必要のない機能を無効化し、悪用されるのを防ぐことができます。

ここでは、機能方針で管理されている人気の高い5つの機能をご紹介します。

機能	デスクトップ	モバイル
microphone	10.78%	10.98%
camera	9.95%	10.19%
payment	9.54%	9.54%
geolocation	9.38%	9.41%
gyroscope	7.92%	7.90%

図8.12. 使用される `Feature-Policy` オプションの上位5つ。

コントロールできる最も人気のある機能はマイクで、デスクトップとモバイルページのほぼ11%がマイクを含むポリシーを発行していることがわかります。データを掘り下げていくと、これらのページが何を許可しているか、またはブロックしているかを見ることができます。

機能	設定	使用率
microphone	none	9.09%
microphone	none	8.97%
microphone	self	0.86%
microphone	self	0.85%
microphone	*	0.64%
microphone	*	0.53%

図8.13. マイク機能の設定.

圧倒的に最も一般的なアプローチは、ここではそのアプローチを取っているページの約9%で、完全にマイクの使用をブロックすることです。少数のページでは、独自のオリジンによるマイクの使用を許可しており、興味深いことにページ内のコンテンツを読み込んでいる任意のオリジンによるマイクの使用を意図的に許可しているページの少数選択があります。

`X-Frame-Options`

`X-Frame-Options` ヘッダーは、ページが別のページでiframeに配置できるかどうかを制御することを可能にします。上述したCSPの `frame-ancestors` の柔軟性には欠けますが、フレームの細かい制御を必要としない場合には効果的です。

デスクトップ(16.99%)とモバイル(14.77%)の両方で `X-Frame-Options` ヘッダの使用率が非常に高いことがわかります。

設定	*デスクトップ*	*モバイル*
sameorigin	*84.92%*	*83.86%*
deny	*13.54%*	*14.50%*
allow-from	*1.53%*	*1.64%*

図8.14. 使用される `X-Frame-Options` の設定。

大多数のページでは、そのページのオリジンのみにフレーミングを制限しているようで、次の重要なアプローチはフレーミングを完全に防止することです。これはCSPの `frame-ancestors` と似ており、これら2つのアプローチが最も一般的です。また、 `allow-from` オプションは、理論的にはサイト所有者がフレーム化を許可するサードパーティのドメインをリストアップできるようにするものですが、決して十分にサポートされていないので[168]、非推奨とされています。

`X-Content-Type-Options`

`X-Content-Type-Options` ヘッダは最も広く展開されているセキュリティヘッダであり、最もシンプルであり、設定可能な値は `nosniff` のみです。このヘッダが発行されると、ブラウザはコンテンツの一部を `Content-Type` ヘッダで宣言されたMIMEタイプとして扱わなければならず、ファイルが異なるタイプのものであることを示唆したときに値を変更しようとはしません。ブラウザが誤ってタイプを嗅ぎ取るように説得された場合、さまざまなセキュリティ上の欠陥が導入される可能性となります。

168. *https://developer.mozilla.org/docs/Web/HTTP/X-Frame-Options#Browser_compatibility*

モバイルとデスクトップの両方で、17.61%のページが`X-Content-Type-Options`ヘッダを発行していることがわかりました。

`X-XSS-Protection`

X-XSS-Protection[169]ヘッダーは、サイトがブラウザに組み込まれたXSS AuditorやXSS Filterを制御することを可能にし、理論的には何らかのXSS保護を提供するはずです。

デスクトップリクエストの14.69%とモバイルリクエストの15.2%が`X-XSS-Protection`ヘッダを使用していた。データを掘り下げてみると、ほとんどのサイト運営者がどのような意図を持っているかが図7.13に示されています。

設定	デスクトップ	モバイル
`1;mode=block`	91.77%	91.46%
`1`	5.54%	5.35%
`0`	2.58%	3.11%
`1;report=`	0.12%	0.09%

図8.15. `X-XSS-Protection` の利用設定。

値`1`はフィルタ/監査を有効にし、`mode=block`は(理論的には)XSS攻撃が疑われる場合にページを表示しないような最も強い保護を設定します。2番目に多かった設定は、単に監査/フィルタがオンになっていることを確認するために`1`という値を提示したもので、3番目に多かった設定は非常に興味深いものでした。

ヘッダーに`0`の値を設定すると、ブラウザが持っている可能性のあるXSSの監査やフィルタを無効にするように指示します。歴史的な攻撃の中には監査やフィルタがユーザーを保護するのではなく、攻撃者を助けるように騙されてしまうことが実証されているものもあるのでサイト運営者の中には、XSSに対する十分な保護があると確信している場合にそれを無効にできるものもあります。

これらの攻撃のため、EdgeはXSSフィルタを引退させ、ChromeはXSS監査を非推奨とし、Firefoxはこの機能のサポートを実装しませんでした。現在ではほとんど役に立たなくなっているにもかかわらず、現在でも全サイトの約15%でヘッダーが広く使われています。

169. https://developer.mozilla.org/docs/Web/HTTP/Headers/X-XSS-Protection

Report-To

Reporting API[170] は、サイト運営者がブラウザからの遠隔測定[171]の様々な情報を収集できるようにするため導入されました。サイト上の多くのエラーや問題は、ユーザーの体験を低下させる可能性がありますが、サイト運営者はユーザーが連絡しなければ知ることができません。Reporting APIは、ユーザーの操作や中断なしに、ブラウザがこれらの問題を自動的に報告するメカニズムを提供します。Reporting APIは `Report-To` ヘッダーを提供することで設定されます。

遠隔測定を送信すべき場所を含むヘッダーを指定することでブラウザは自動的にデータの送信を開始し、Report URI[172]のようなサードパーティのサービスを使用してレポートを収集したり、自分で収集したりできます。導入と設定の容易さを考えると、現在この機能を有効にしているサイトは、デスクトップ（1.70%）とモバイル（1.57%）のごく一部に過ぎないことがわかります。収集できるテレメトリの種類については、Reporting API仕様[173]を参照してください。

ネットワークエラーロギング

ネットワークエラーロギング(NEL)[174]は、サイトが動作不能になる可能性のあるブラウザのさまざまな障害についての詳細な情報を提供します。 `Report-To` が読み込まれたページの問題を報告するために使用されるのに対し、 `NEL` ヘッダーを使用すると、サイトはブラウザにこのポリシーをキャッシュするように通知し、将来の接続問題が発生したときに上記の `Reporting-To` ヘッダーで設定されたエンドポイントを介して報告できます。したがって、NELはReporting APIの拡張機能とみなすことができます。

もちろん、NELはReporting APIに依存しているので、NELの使用量がReporting APIの使用量を上回ることはありません。これらの数値が同じであるという事実は、これらが一緒にデプロイされていることを示唆しています。

NELは信じられないほど貴重な情報を提供しており、情報の種類についてはネットワークエラーロギング仕様[175]で詳しく説明しています。

クリアサイトデータ

クッキー、キャッシュ、ローカルストレージなどを介してユーザーのデバイスにデータをローカルに保存する機能が増えているため、サイト運営者はこのデータを管理する信頼性の高い方法を必要としていました。Clear Site Dataヘッダーは、特定のタイプのすべてのデータ

170. *https://www.w3.org/TR/reporting/*
171. *https://scotthelme.co.uk/introducing-the-reporting-api-nel-other-major-changes-to-report-uri/*
172. *https://report-uri.com/*
173. *https://www.w3.org/TR/reporting/*
174. *https://www.w3.org/TR/network-error-logging/*
175. *https://w3c.github.io/network-error-logging/#predefined-network-error-types*

がデバイスから削除されることを確実にする手段を提供しますが、すべてのブラウザではまだサポートされていません[176]。

このヘッダの性質を考えると、使用量がほとんど報告されていないのは驚くに値しません。デスクトップリクエストが9件、モバイルリクエストが7件だけです。私たちのデータはサイトのホームページしか見ていないので、ログアウトのエンドポイントでヘッダーが最もよく使われているのを見ることはないでしょう。サイトからログアウトすると、サイト運営者はClear Site Dataヘッダを返し、ブラウザは指定されたタイプのすべてのデータを削除します。これはサイトのホームページでは行われないでしょう。

クッキー

クッキーには利用可能な多くのセキュリティ保護があり、それらのいくつかは長年にわたって利用可能であるが、それらのいくつかは本当に非常に新しいものでありここ数年の間に導入されただけです。

`Secure`

クッキーの`Secure`フラグは、ブラウザに安全な(HTTPS)接続でのみクッキーを送信するように指示し、ホームページでセキュアフラグが設定されたクッキーを発行しているサイトはごくわずかな割合(デスクトップでは4.22%、モバイルでは3.68%)であることがわかります。この機能が比較的簡単に使用できることを考えると、これは憂慮すべきことです。繰り返しになりますが、HTTPとHTTPSの両方でデータを収集したいと考えている分析や広告サードパーティリクエストの高い使用率がこれらの数字を歪めている可能性が高く、認証クッキーのような他のクッキーでの使用状況を見るのは興味深い調査でしょう。

`HttpOnly`

クッキーの`HttpOnly`フラグはブラウザにページ上のJavaScriptがクッキーへアクセスできなくすることを指示します。多くのクッキーはサーバによってのみ使用されるので、ページ上のJavaScriptが必要としないため、クッキーへのアクセスを制限することはクッキーを盗むXSS攻撃からの大きな防御となります。デスクトップでは24.24%、モバイルでは22.23%と、ホームページ上でこのフラグを立ててクッキーを発行しているサイトの方がはるかに多いことがわかります。

176. *https://caniuse.com/#feat=mdn-http_headers_clear-site-data*

`SameSite`

クッキーの保護に追加された最近の追加機能として、`SameSite`フラグは クロスサイトリクエストフォージェリ(CSRF)[177]攻撃(XSRFとしてもよく知られています)に対する強力な保護となります。

これらの攻撃は、ブラウザが通常、すべてのリクエストに関連するクッキーを含むという事実を利用して動作します。したがって、ログインしていてクッキーが設定されていて、悪意のあるサイトを訪問した場合、APIを呼び出すことができブラウザは「親切に」クッキーを送信します。クッキーに`SameSite`属性を追加することで、第三者のサイトからの呼び出しがあった場合にクッキーを送信しないようにウェブサイトがブラウザに通知し、攻撃を失敗させることができます。

最近導入されたメカニズムなので、デスクトップとモバイルの両方でリクエストの0.1%と予想されるように、同じサイトのクッキーの使用率ははるかに低くなっています。クッキーがクロスサイトで送信されるべき使用例があります。例えば、シングルサインオンサイトは認証トークンと一緒にクッキーを設定することで暗黙のうちに動作します。

設定	*デスクトップ*	*モバイル*
strict	*53.14%*	*50.64%*
lax	*45.85%*	*47.42%*
none	*0.51%*	*0.41%*

図8.16. SameSite設定の使用法。

既にSame-Siteのクッキーを利用しているページのうち、半分以上が`strict`モードで利用していることがわかる。これに続いて、`lax`モードでSame-Siteを利用しているサイト、そして少数のサイトでは`none`を利用しているサイトが続いています。この最後の値は、ブラウザベンダーが`lax`モードをデフォルトで実装する可能性があるという今後の変更をオプトアウトするために使用されます。

この機能は危険な攻撃からの保護を提供するため、現在のところ、主要なブラウザが デフォルトでこの機能を実装[178]し、値が設定されていなくてもクッキーに対してこの機能を有効にする可能性があると指摘されています。これが実現した場合、SameSiteの保護機能は有効になりますが、`strict`モードではなく`lax`モードの弱い設定では、より多くの破損を引き起こす可能性があるためです。

177. *https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%82%B9%E3%82%B5%E3%82%A4%E3%83%88%E3%83%AA%E3%82%AF%E3%82%A8%E3%82%B9%E3%83%88%E3%83%95%E3%82%A9%E3%83%3*
178. *https://blog.chromium.org/2019/10/developers-get-ready-for-new.html*

プレフィックス

クッキーに最近追加されたもう一つの方法として、クッキープレフィックスがあります。これらはクッキーの名前を使用して、すでにカバーされている保護に加えて、2つのさらなる保護のうちの1つを追加します。上記のフラグはクッキー上で誤って設定を解除される可能性がありますが、名前は変更されませんので、セキュリティ属性を定義するために名前を使用することでより確実にフラグを強制できます。

現在のところ、クッキーの名前の前には`__Secure-`か`__Host-`のどちらかを付けることができ、どちらもクッキーに追加のセキュリティを提供しています。

	ホームページ数		ホームページの割合	
プレフィックス値	デスクトップ	モバイル	デスクトップ	モバイル
`__Secure-`	640	628	0.01%	0.01%
`__Host-`	154	157	0.00%	0.00%

図8.17. クッキーのプレフィックスの使用法

図が示すように、どちらのプレフィックスの使用率も信じられないほど低いのですが、2つのプレフィックスが緩和されているため`__Secure-`プレフィックスの方がすでに利用率は高いです。

サブリソースの完全性

最近増えているもう1つの問題は、サードパーティの依存関係のセキュリティです。サードパーティからスクリプトファイルを読み込む際には、スクリプトファイルが常に欲しいライブラリ、おそらく特定のバージョンのjQueryであることを期待します。CDNやサードパーティのホスティングサービスが危殆化した場合、それらをホスティングしているスクリプトファイルを変更される可能性があります。このシナリオでは、アプリケーションは訪問者に危害を加える可能性のある悪意あるJavaScriptを読み込んでいることになります。これが、サブリソースの完全性が保護する機能です。

スクリプトやリンクタグに`integrity`属性を追加することで、ブラウザはサードパーティのリソースの整合性をチェックし、変更された場合は拒否できます。

```
<script
```

```
    src="https://code.jquery.com/jquery-3.4.1.min.js"
    integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
    crossorigin="anonymous"></script>
```

整合性属性が設定されたリンクまたはスクリプトタグを含むデスクトップページの0.06%（247,604）とモバイルページの0.05%（272,167）しかないため、SRIの使用には多くの改善の余地があります。現在、多くのCDNがSRIの整合性属性を含むコードサンプルを提供しているため、SRIの使用は着実に増加していると思われます。

結論

Webの機能が向上し、より多くの機密データへのアクセスが可能になるにつれ、開発者が自社のアプリケーションを保護するためにWebセキュリティ機能を採用することがますます重要になってきています。本章でレビューするセキュリティ機能は、Webプラットフォーム自体に組み込まれた防御機能であり、すべてのWeb制作者が利用可能です。しかし、本章の研究結果のレビューからもわかるように、いくつかの重要なセキュリティメカニズムはウェブの一部にしか適用されていないため、エコシステムのかなりの部分がセキュリティやプライバシーのバグにさらされたままとなっています。

暗号化

ここ数年の間に、転送中データの暗号化については、Webが最も進歩しています。TLSセクションで説明したように、ブラウザーベンダー、開発者、Let's Encryptのような認証局の様々な努力のおかげで、HTTPSを使用しているウェブの割合は着実に増加しています。本稿執筆時点では、大多数のサイトがHTTPSで利用可能であり、トラフィックの機密性と完全性が確保されています。重要なことに、HTTPSを有効にしているWebサイトの99%以上では、TLSプロトコルの新しい安全なバージョン（TLSv1.2およびTLSv1.3）が使用されています。GCMモードでのAESなどの強力なcipher suitesの使用率も高く、すべてのプラットフォームで95%以上のリクエストを占めています。

同時に、TLS設定のギャップは依然としてかなり一般的です。15%以上のページが混合コンテンツの問題に悩まされており、ブラウザに警告が表示され、4%のサイトではセキュリティ上の理由から最新のブラウザにブロックされています。同様に、HTTP Strict Transport Securityの利点は、主要なサイトのごく一部にしか及ばず、大多数のWebサイトでは最も安全なHSTS構成を有効にしておらず、HSTS プリロードの対象外となっています。HTTPSの採用が進んでいるにもかかわらず、未だに多くのクッキーが`Secure`フラグなしで設定されており、クッキーを設定しているホームページのうち、暗号化されていないHTTPでの送信を防止しているのはわずか4%に過ぎません。

一般的なウェブの脆弱性からの防御

機密データを扱うサイトで作業するウェブ開発者は、XSS[179]、CSRF[180]、クリックジャッキング[181]、およびその他の一般的なウェブバグからアプリケーションを保護するために、オプトインウェブセキュリティ機能を有効にしていることがよくあります。これらの問題は、`X-Frame-Options`、`X-Content-Type-Options`、コンテンツセキュリティポリシーを含む、多くの標準的で広くサポートされているHTTPレスポンスヘッダを設定することで緩和できます。

セキュリティ機能とウェブアプリケーションの両方共複雑であることが大部分を占めていますが、現在これらの防御機能を利用しているウェブサイトは少数派であり、多くの場合、リファクタリングの努力を必要としないメカニズムのみを有効にしています。最も一般的なオプトインアプリケーションのセキュリティ機能は、`X-Content-Type-Options` (17%のページで有効)、`X-Frame-Options` (16%)、および非推奨の`X-XSS-Protection`ヘッダ(15%)です。最も強力なWebセキュリティメカニズムであるコンテンツセキュリティポリシーは、5%のWebサイトでしか有効になっておらず、そのうちのごく一部(全サイトの約0.1%)だけがCSP ナンスとハッシュに基づいたより安全な設定を使用しています。関連する参照元ポリシーは、`Referer`ヘッダーで第三者に送信される情報量を減らすことを目的としているが、同様に使用しているのは3%のウェブサイトのみです。

現代のウェブプラットフォームの防御

近年、ブラウザーは、主要な脆弱性や新たなWeb脅威からの保護を提供する強力な新しいメカニズムを実装しています; これには、サブリソースの完全性、同じサイトのクッキー、およびクッキーのプレフィックスが含まれます。

これらの機能は比較的少数のウェブサイトでしか採用されていません。Trusted Types[182]、オリジン間リソース共有[183]、オリジン間オープナー共有[184]のような、さらに最近のセキュリティメカニズムは、まだ広く採用されていません。

同様に、Reporting API、ネットワークエラーロギング、`Clear-Site-Data`ヘッダのような便利な機能もまだ初期段階であり、現在は少数のサイトで利用されています。

179. https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%82%B9%E3%82%B5%E3%82%A4%E3%83%88%E3%82%B9%E3%82%AF%E3%83%AA%E3%83%97%E3%83%86%E3%82%A3%E3%83%B3 Cross-site_scripting
180. https://ja.wikipedia.org/wiki/%E3%82%AF%E3%83%AD%E3%82%B9%E3%82%B5%E3%82%A4%E3%83%88%E3%83%AA%E3%82%AF%E3%82%A8%E3%82%B9%E3%83%88%E3%83%95%E3%82%A9
181. 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
182. https://w3c.github.io/webappsec-trusted-types/dist/spec/
183. https://developer.mozilla.org/docs/Web/HTTP/Cross-Origin_Resource_Policy_(CORP)
184. https://www.chromestatus.com/feature/5432089535053824

結びつき

ウェブの規模では、オプトインプラットフォームのセキュリティ機能の全体的なカバー率は、現在のところ比較的低い。最も広く採用されている保護であっても、一般的なセキュリティ問題に対するプラットフォームのセーフガードを持たないウェブの大部分を残して、ウェブサイトの4分の1未満で有効になっています。

しかし、これらのメカニズムの採用は、より機密性の高いユーザーデータを頻繁に扱う大規模なウェブアプリケーションに偏っていることに注意することが重要です。これらのサイトの開発者は、一般的な脆弱性に対する様々な保護を可能にすることを含め、ウェブの防御力を向上させるために投資することが多くなっています。Mozilla Observatory[185]やSecurity Headers[186]などのツールは、ウェブで利用可能なセキュリティ機能の便利なチェックリストを提供してくれます。

ウェブアプリケーションが機密性の高いユーザーデータを扱う場合はユーザーを保護し、ウェブをより安全にするためこのセクションで概説されているセキュリティメカニズムを有効にすることを検討してください。

著者

Scott Helme

@Scott_Helme ScottHelme https://scotthelme.co.uk

Scott Helmeはセキュリティ研究者であり、report-uri.com[187] と securityheaders.com[188] の創設者でもあります。Twitterでは、@Scott_Helmeでセキュリティの話をしたり、scotthelme.co.uk[189]でブログを書いたりしています。

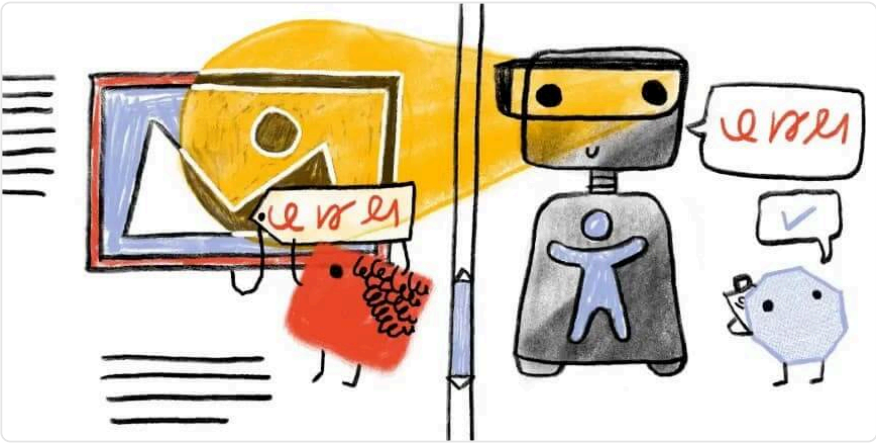
Artur Janc

@arturjanc arturjanc

Artur JancはGoogleの情報セキュリティエンジニアで、GoogleとWeb全体のWebプラットフォームのセキュリティメカニズムの設計と採用に取り組んでいます。@arturjanc on Twitterとして、インターネット上の人々と議論しています。

185. https://observatory.mozilla.org/
186. https://securityheaders.com/
187. https://report-uri.com
188. https://securityheaders.com
189. https://scotthelme.co.uk

部 II 章 9
アクセシビリティ



Nektarios Paisios、 David Fox と Abigail Klein によって書かれた。
Laura Eberly によってレビュ。
Doug Sillars、 Rick Viscomi と David Fox による分析。
David Fox 編集。
M.Sakamaki によって翻訳された。

導入

Webのアクセシビリティは、包摂的で公平な社会の上では無くてはならない存在です。私たちの社会性と仕事や生活の多くがオンラインの世界に推移するにつれて、障害のある人々も分け隔てなく、すべてのオンラインの対話に参加できることがさらに重要になってきます。建築家が車椅子用の傾斜路のようなアクセシビリティ機能を作成や省略できるように、Web開発者はユーザーが頼りにしている支援技術を助けたり邪魔したりできます。

障害を持つユーザーの事を考えた時ユーザージャーニーはほぼ同じとなることを忘れないでください、彼らは異なるツールを使っているだけでしかありません。よく知られてるツールとして、スクリーンリーダー、画面拡大鏡、ブラウザまたは文字の拡大、音声コントロールなどがありますが、これ以外にも色々とあります。

ほとんどの場合、アクセシビリティを改善することでサイトを訪れるすべての人に対してメリットを与える事ができます。私達は普通、障害者は生涯その障害を抱えていると思ってい

ますが、一時的だったり状況的に障害を持つような人も居ます。たとえばその誰かが全盲なのか、一時的な目の感染症なのか、はたまた野外で眩しい太陽の下という状況なのか。これらすべて、その誰かが画面を見ることができない理由の説明になります。誰もが状況により障害を持ちうるため、Webページのアクセシビリティを改善することは、あらゆる状況ですべてのユーザーの体験を向上させることに繋がります。

Webコンテンツのアクセシビリティガイドライン[190](WCAG)はWebサイトの利便性を向上する方法についてのアドバイスが纏められています。このガイドラインを分析の基礎に使いました。しかし、ほとんどの場合においてWebサイトのアクセシビリティをプログラムによって分析するのは非常に困難です。たとえば、Webプラットフォームは機能的には同じ結果となる複数の方法を提供しており、それを実現するための基盤となるコードはまったく別物になる場合があります。したがって、私達の分析結果はWebアクセシビリティ全体の単なる概算でしかありません。

私達はもっとも興味深い洞察を4種類のカテゴリに分類しました。それは読みやすさ、Web上のメディア、ページナビゲーションのしやすさ、補助技術との互換性です。

テスト中にデスクトップとモバイルの間でアクセシビリティに大きな違いは見つかりませんでした。この結果で提示されているメトリックは、とくに明記していない限りはデスクトップの分析結果です。

読みやすさ

Webページの主な目的はユーザーの興味を引くコンテンツを配信することです。このコンテンツはビデオや画像の組み合わせなどありますが、ほとんどの場合、シンプルなページ上のテキストです。テキストコンテンツが読者にとって読みやすいことは、とても重要です。訪問者がWebページを読めない場合、訪問者はWebページに興味を持つことがなくなり、最終的には離脱してしまうでしょう。この節ではサイトが苦労するであろう3つの分野を見ていきます。

色のコントラスト

あなたのサイトの訪問者が完璧な内容を見ることができない、さまざまな可能性があります。訪問者は色覚多様性を持ち、フォントと背景色を区別できない場合があります（ヨーロッパ系の男性12人に1人、女性200人に1人[191]）。おそらく、彼らは太陽の下で画面の明るさを最大にして読んでいるため、視力を著しく損なっているのでしょう。もしくは年をとってしまい、彼らの目が以前と同じように色を区別できなくなったのでしょう。

190. *https://www.w3.org/WAI/WCAG21/quickref/*
191. *http://www.cvrl.org/people/stockman/pubs/1999%20Genetics%20chapter%20SSJN.pdf*

このような条件下であっても、あなたのWebサイトが確実に読めるようにするため、テキストと背景で十分な色のコントラストがあることを確認することは重要です。

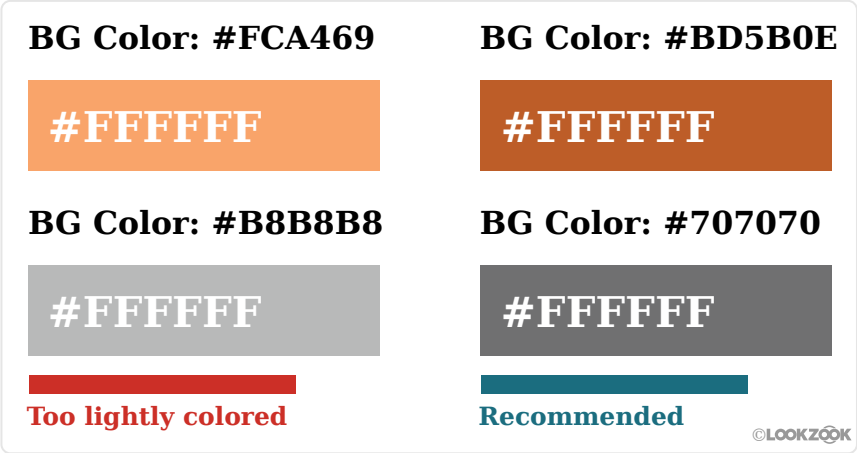


図9.1. 色のコントラストが不十分なテキストの表示例 LookZook提供

すべてのテキストに十分な色のコントラストが適用されているサイトは22.04%のみでした。これは言い換えると、5つのサイトのうち4つは背景に溶け込んで読みにくいテキストを持っていると言うことです。

注意：画像中のテキストは分析できていないため、ここで報告されているメトリックはカラーコントラストテストに合格したWebサイトの総数の上限でしかありません。

ズーミングとページのスケーリング

読みやすいフォントサイズ[192]やターゲットサイズ[193]を使うことで、ユーザーがWebサイトを読んだり操作するのを手助けできます。しかし、このガイドラインに対して完全に準拠しているWebサイトですら、訪問者一人ひとりの特定のニーズを満たすことはできません。これがピンチズームやスケーリングなどのデバイスによる機能が非常に重要となる理由です。ユーザーが貴方のページを微調整できるようにして彼らのニーズを満たします。また、小さなフォントやボタンが使われて操作が非常に難しいサイトであっても、ユーザーにそのサイトを使う機会を与えることができます。

まれですが、スケーリングの無効化が許容される場合はあります。それは問題となるページがタッチコントロールを使ったWebベースのゲームなどの場合です。このような場合、有効にしてしまうとプレイヤーがゲームで2回タップをするたびにプレイヤーのスマホがズーム

192. https://accessibleweb.com/question-answer/minimum-font-size/
193. https://www.w3.org/WAI/WCAG21/quickref/#target-size

インやズームアウトしてしまい、皮肉なことに操作できなくなってしまいます。

なので、開発者はメタビューポートタグ[194]で次の2つのプロパティのどちらかを設定することで、この機能を無効化できます。

1. `user-scalable` を `0` か `no` に設定
2. `maximum-scale` を `1` もしくは `1.0` などに設定

悲しいことに、開発者はこの機能を誤用しすぎており、モバイルサイトの3つのうち1つ（32.21%）でこの機能を無効化しています。さらにApple（iOS 10の時点）でWeb開発者がズームを無効化できなくなってしまいました。モバイルSafariは純粋にタグを無視します[195]。すべてのサイトは新しいiOSデバイスでズームとスケーリングができます。

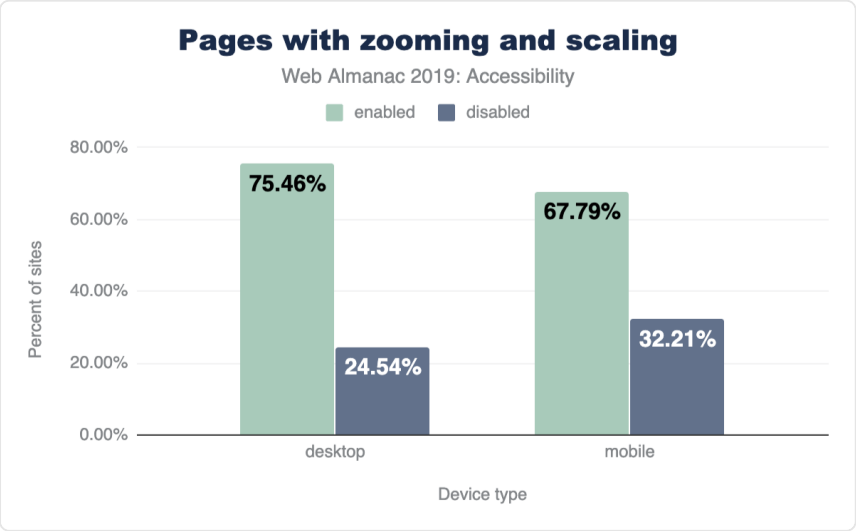


図9.2. ズームとスケーリングを無効にしているサイトとデバイスの種類の割合。

言語の識別

Webには驚くべき大量のコンテンツが溢れていますが、ここには大きな落とし穴があります。世界には1,000以上の異なる言語が存在しており、探しているコンテンツが流暢な言葉で書かれていない可能性があります。昨今、私たちは翻訳技術で大きな進歩を遂げており、貴方はおそらくその1つをWebで利用しているでしょう（例：Google翻訳）

194. *https://developer.mozilla.org/docs/Web/HTML/Viewport_meta_tag*
195. *https://archive.org/details/ios-10-beta-release-notes*

この機能を円滑に行うために、翻訳エンジンはあなたのページがどの言語で書かれているかを知る必要があります。これには `lang` 属性が使われます。lang属性がないと、コンピューターはページが記述されている言語を推測する必要が出てきます。想像できると思いますが、ページ中で複数の言語が使われている場合、これは多くの間違いを引き起こします（たとえば、ページナビゲーションは英語なのに投稿されているコンテンツが日本語のような場合）。

この言語が指定されていない場合の問題は、規定のユーザー言語でテキストを読む傾向があるスクリーンリーダーのようなテキスト読み上げ支援技術で顕著になります。

分析の結果、26.13%で `lang` 属性による言語指定がありませんでした。これは4分の1以上のページが上記のような問題の影響を受けやすいという事です。良いニュース？ `lang` 属性を使っているサイトの99.68%で有効な言語コードが適用されています。

煩わしいコンテンツ

認知障害などの一部のユーザーは、1つの作業に対して長時間集中することが困難です。こういったユーザーは、とくに表面的なエフェクトが多く、それが目の前の作業に関わらない場合、動きやアニメーションが多く含まれるページを利用したくありません。

残念なことに、私達の調査結果では無限ループアニメーションがWebでは非常に一般的であり、21.04%のページが無限CSSアニメーションや `<marquee>` および `<blink>` 要素が使われている事を示しています。

ただし、この問題の大部分は人気のあるサードパーティー製のスタイルシートが規定で無限ループのCSSアニメーションが含まれている事が原因であることに注意してください。このようなアニメーションスタイルを実際に適用したページ数がいくつあるのか、私達は特定できませんでした。

Web上のメディア

画像の代替テキスト

画像はWebの体験の根幹です。それらは強い物語性を伝えることができ、注意を引いて感情を引き出すことができます。しかし、ストーリーの一部を伝えるために私達が頼っている画像は、誰でも見ることができるわけではありません。幸いなことに、1995年、HTML 2.0でこの問題に対する解決策が提供されました、それはalt属性[196]です。alt属性は使われている画像にテキストの説明を追加できる機能をWeb開発者に提供します。これによって、画像を見

196. https://webaim.org/techniques/alttext/

ることができない（もしくは読み込めない）ときに、altテキストに書かれた説明を読むことができます。altテキストは、彼らが見逃していたかもしれないストーリーの一部を埋めることができます。

alt属性は25年前から存在していますが、49.91%のページで画像の一部にalt属性が提供されておらず、8.68%のページでまったく使用されていませんでした。

ビデオやオーディオの字幕

画像が強力なストーリーテラーであるように、オーディオとビデオも注目を集めたりアイデアを表現する事ができます。オーディオやビデオコンテンツに字幕が付けられていない場合、コンテンツが聞こえないユーザーはWebのほとんどを見逃してしてしまいます。耳が聞こえない、もしくは難聴のユーザーから一番よく聞くのは、すべてのオーディオとビデオコンテンツに字幕を含めて欲しいというお話です。

`<audio>` や `<video>` 要素を使うサイトのうち、字幕を提供しているのは0.54%のみでした（`<track>` 要素を含むサイトで測定）一部のWebサイトには、ユーザーにビデオとオーディオの字幕を提供するカスタムソリューションがあります。これらは検出できなかったので、字幕を利用しているサイトの本当の割合は、おそらく少し高いでしょう。

使いやすいページナビゲーション

レストランでメニューを開くとき、おそらく最初にするのは前菜、サラダ、主料理、デザートなどのセクションヘッダーをすべて読むことでしょう。ここから、すべてのメニューの選択肢を見渡し、もっとも興味のある料理に飛ぶことができます。同様に、訪問者がWebページを開く時、訪問者の目標はもっとも興味を持っている情報を見つけることです（それがページにアクセスした理由のはずです）ユーザーが目的のコンテンツをできるだけ早く見つけることができるように（それと、戻るボタンを押させないため）ページのコンテンツをいくつかの視覚的に異なるセクションに分割する必要があります。たとえば、ナビゲーション用のサイトヘッダーを置き、記事の見出しをユーザーが素早く見渡せるようにしたりその他の無関係なリソースを纏めたフッターなどをに分割する等です。

これは非常に重要な事で、訪問者のコンピューターがこれらの異なるセクションを認識できるよう、注意してページのマークアップをする必要があります。それはなぜかと言うと、ほとんどの読者はマウスを利用してページを探索しますが、それ以外の読者はキーボードとスクリーンリーダーに依存しています。これらのテクノロジーは、コンピューターがあなたのページをどの程度理解できるかに大きく依存します。

見出し

見出しは見た目上で有用なだけではなく、スクリーンリーダーでも役立ちます。見出しによってスクリーンリーダーはセクション間を素早く移動でき、さらに、セクションが終了して別のセクションが開始される場所を明示的にします。

スクリーンリーダーを使うユーザーの混乱を避けるために、見出しのレベルを飛ばさないようにしてください。たとえば、H2をスキップして、H1の次にH3を使うのは止めてください。なぜこれが重要なのか？　それはスクリーンリーダーを使うユーザーが、予期せぬ変化からコンテンツを見逃したと勘違いしてしまうためです。このような場合、本当は見逃しがないにもかかわらず、見逃している可能性があるものを探し始めてしまいます。あわせて、より一貫したデザインを維持することで、すべての読者を支援します。

そうは言いつつも、結果としては次のようになっています。

1. 89.36%のページが何らかの方法で見出しを使っています。素晴らしことです。
2. 38.6%のページは見出しのレベルを飛ばしています。
3. 不思議な事に、H2はH1よりも多くのサイトで見つかりました。

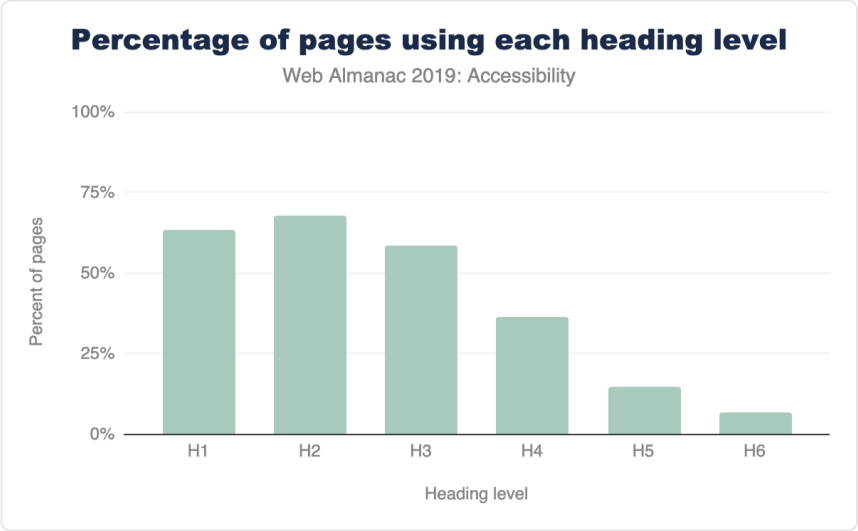


図9.3. 見出しレベルの人気。

Mainランドマーク

mainランドマーク[197]はWebページのメインコンテンツが始まる場所をスクリーンリーダーに示すことで、ユーザーがーすぐその場所に飛ぶことができます。mainランドマークがない場合、スクリーンリーダーのユーザーはサイト内の新しいページにアクセスするたび、手動でナビゲーションをスキップする必要が出てきます。これは明らかにイライラするでしょう。

ページの4分の1（26.03%）にだけmainランドマークが含まれていることが判明しました。さらに驚くべきことに、8.06%のページに複数のmainランドマークが誤って含まれているため、ユーザーは実際のメインコンテンツがどのランドマークなのかを推測する必要が出ていました。

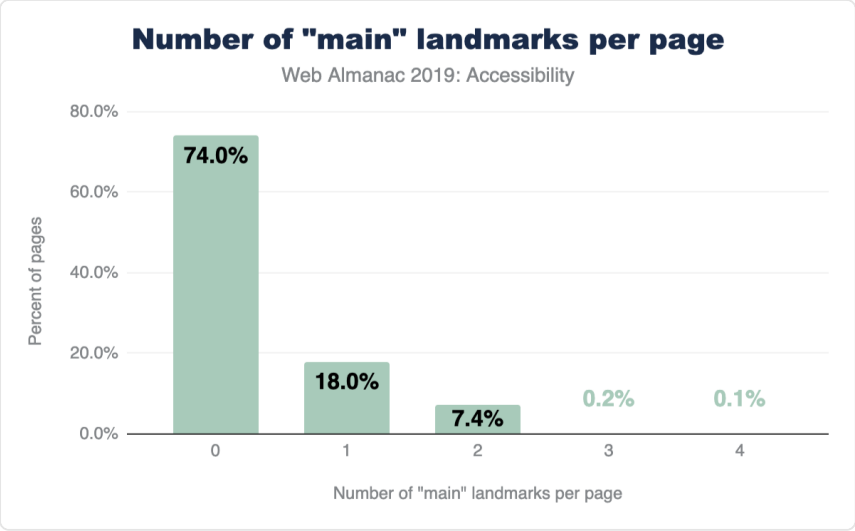


図9.4. 「main」ランドマークの数によるページの割合。

HTML section要素

HTML5は2008年リリースされ、2014年に公式の標準となっているので、コンピューターとスクリーンリーダーがページの見た目と構造を理解するのに有用なHTML要素がたくさんあります。

`<header>`、`<footer>`、`<navigation>`、`<main>`などの要素は特定の種類のコンテンツがどこにあるか明示的にして、ユーザーがそのページへ素早く飛ぶことを可能にします。これらはWeb全体で幅広く使われており、ほとんどがページの50%以上で使われています。

197. https://developer.mozilla.org/docs/Web/Accessibility/ARIA/Roles/Main_role

（`<main>`は外れ値です。）

`<article>`、`<hr>`、`<aside>`のようなものは、読者がページのメインコンテンツを理解するのに役立ちます。たとえば、`<article>`は記事が終了して別の記事が開始される場所を示します。これらの要素はほとんど使われておらず、使用率は約20%ですが、これらはすべてのWebページで必要となるわけではないため、とくに驚くべき統計ではありません。

これらの要素はすべてアクセシビリティサポートを主目的として設計されており、見た目の変化はありません。つまりこれは、既存の要素を安全に置き換えることが可能なので意図しない影響で苦しむことはないでしょう。

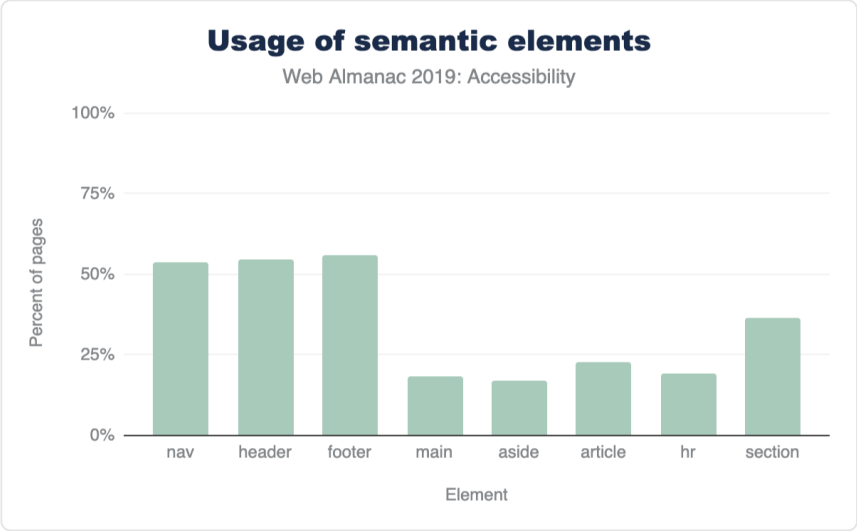


図9.5. 色々なHTMLセマンティック要素の利用率。

ナビゲーションで使われているその他のHTML要素

よく使われているスクリーンリーダーは、ユーザーがリンク、一覧、一覧のアイテム、iframe、それと編集フィールド、ボタン、リストボックスなどのフォームフィールドに素早く飛び、誘導できます。図9.6はこういった要素を使うページの表示頻度を表しています。

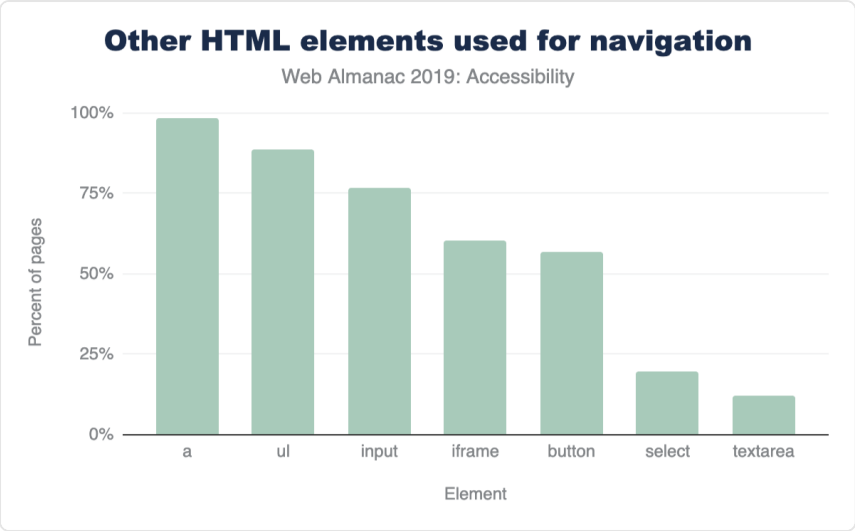


図9.6. ナビゲーションで使われるその他のHTML要素。

スキップリンク

スキップリンク[198]はスクリーンリーダーやキーボードだけを使うユーザーが、メインコンテンツに直接飛ぶことができるようにする、ページ上部に配置されるリンクです。これは、ページの上部にあるすべてのナビゲーションリンクとメニューを効率的に「スキップ」します。スキップリンクは、スクリーンリーダーを利用していないキーボードユーザーにとってとくに便利です。それは、このようなユーザーは通常他のクィックナビゲーションモード（ランドマークや見出しなど）にアクセスできないためです。サンプリングされたページの14.19%にスキップリンクが使われていました。

スキップリンクの動作を試す事ができます！　シンプルにGoogle検索を実行し、検索結果ページが表示されたらすぐに「Tab」キーを押します。図9.7のような、事前に隠されたリンクが表示されます。

198. *https://webaim.org/techniques/skipnav/*

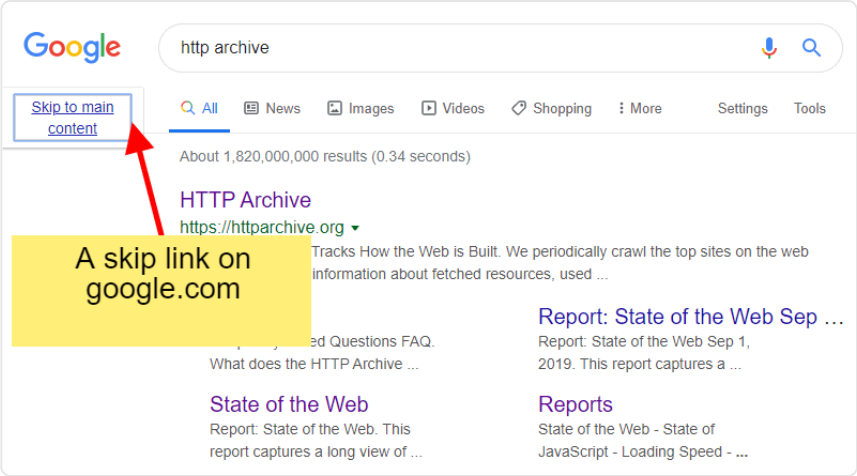


図9.7. google.comのスキップリンク外観。

実際、ここでも使用する[199]ので、このサイトを離れる必要もありません。

サイトを分析するときに、正しいスキップリンクを判断するのは困難です。なのでこの分析ではページの最初の3つのリンク内にアンカーリンク（`href=#heading1`）が見つかった場合、それをスキップリンクのあるページと定義しました。つまり14.19%というのは厳密には上限です。

ショートカット

`aria-keyshortcuts`や`accesskey`属性を介して設定されたショートカットキーは、次の2つの方法のどちらかで使うことができます。

1. リンクやボタンなどのページ上の要素を活性化させます。
2. 特定の要素に対するページフォーカスを提供します。たとえばページ上にある特定の入力にフォーカスを移動させて、すぐさまユーザーが入力できるようにします。

サンプルを見る限り`aria-keyshortcuts`はほとんど採用されておらず、400万以上ある分析対象のうち、たった159のサイトでだけ使われていました。`accesskey`属性はかなり利用されており、Webページの2.47%（モバイルだと1.74%）で使われています。デスクトップでショートカットの利用率が多いのは、開発者がモバイルでサイトにアクセスする時、キーボードでなくタッチスクリーンのみで利用することを期待しているためと考えています。

199. https://github.com/HTTPArchive/almanac.httparchive.org/pull/645

とくに驚くべき点は、ショートカットキーを適用しているモバイルサイトの15.56%とデスクトップサイトの13.03%で、1つのショートカットキーを複数の要素に割り当てている事です。これはブラウザがショートカットキーの対象となる要素を推測する必要があることを意味しています。

テーブル

テーブルは大量のデータを整理し表現する主要な方法の1つです。スクリーンリーダーやスイッチ（運動障害のあるユーザーが使ったりします）などのさまざまな支援技術には、この表形式データをより効率的に操作できる特別な機能を持っています。

テーブルの見出し

テーブルの詳細な構造に対応したテーブルヘッダーを使うことで、特定の列または行が参照するコンテキストを失うこと無く、列や行全体を簡単に読み取り可能とします。ヘッダー行や列のないテーブルを操作しないといけないのは、スクリーンリーダーのユーザーにとっては使いづらいでしょう。これは、テーブルが非常に大きい時にスクリーンリーダーのユーザーはヘッダーのないテーブルだと自分の場所を把握するのが難しいからです。

テーブルのヘッダーをマークアップするには、シンプルに（`<td>`タグの代わりに）`<th>`タグを使うか、ARIAの`columnheader`か`rowheader`ロールのどれかを使います。この方法のどれかでテーブルがマークアップされていたのは、テーブルを含むページの24.5%だけでした。そのため、テーブルにヘッダーが含まれない四分の三のページは、スクリーンリーダーのユーザーにとって非常に深刻な課題を持っています。

`<th>`と`<td>`を利用するのは、テーブルにヘッダーをマークアップするもっとも一般的な方法のようです。`columnheader`と`rowheader`のロールを使っているサイトはほとんど存在せず、使っているサイトは合計677個（0.058%）のみでした。

キャプション

`<caption>`要素が使われているテーブルキャプションは、さまざまな読者に対してより多くのコンテキストを提供できます。キャプションはテーブルが共有している情報を読む準備ができてる人や、集中できない環境だったり、作業の中断が必要な人々にとってとくに便利になります。また、スクリーンリーダーユーザーや学習障害、知的障害のある人などの、大きなテーブルだと自分の見ている場所で迷子になる可能性がある人々にとっても有用です。読者が分析している内容を理解しやすくすればするほど、より良い結果を得られるでしょう。

にもかかわらず、表が含まれるページでは4.32%だけでしかキャプションを提供していませ

ん。

支援技術との互換性

ARIAの使用

Web上でもっとも一般的かつ広く活用されているアクセシビリティの使用の1つにAccessible Rich Internet Applications[200] (ARIA)という標準があります。この標準は視覚的要素の背景にある目的（つまり、セマンティクスな意味）と、それにより可能になるアクションの種類を教えるのに役立つ追加のHTML属性をもった大きな配列を提供します。

ARIAを適切かつ正しく使うのは難しい場合があります。例えば、ARIA属性を使っているページでは12.31%の属性に無効な値が割り当てられていました。ARIA属性の利用に誤りがあると、ページに視覚的な影響が及ばないため問題になります。これらの間違いは自動検証ツールを使っても検出できますが、一般的には実際の支援ソフトウェア（スクリーンリーダーなど）を実際に使う必要があります。この節ではARIAがWeb上でどのように使われているか、特に標準のどの部分が最も普及しているのかを検証していきます。

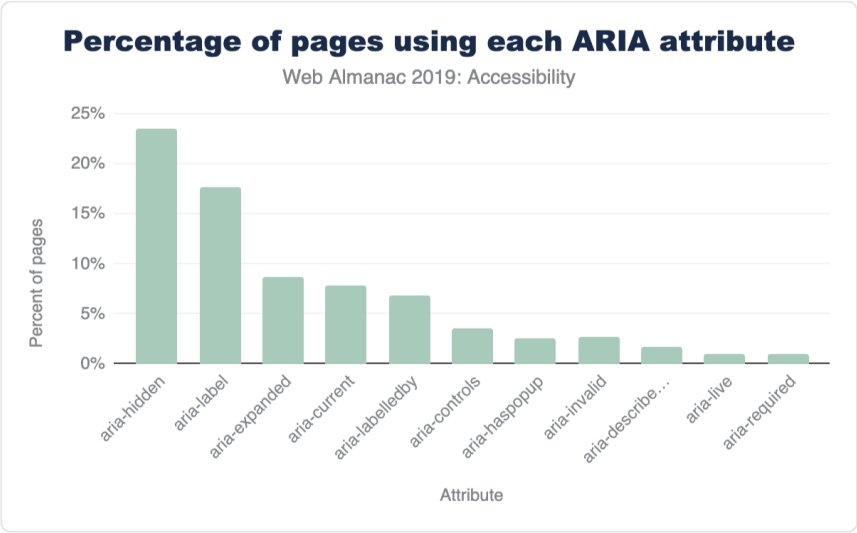


図9.8. 総ページ数とARIA属性の割合。

200. https://www.w3.org/WAI/standards-guidelines/aria/

`role` 属性

「ロール」属性は、すべてのARIAの仕様中で最も重要な属性です。これは指定されたHTML要素の目的（セマンティックな意味）をブラウザへ通知するために使用されます。たとえば、CSSを使って視覚的にボタンのようにスタイルが適用された `<div>` 要素には `button` のARIAロールを与える必要があります。

実際には46.91%のページが少なくとも1つのARIAロール属性を使っています。以下の図9.9は、最もよく使われているトップ10のARIAロールの値一覧を纏めました。

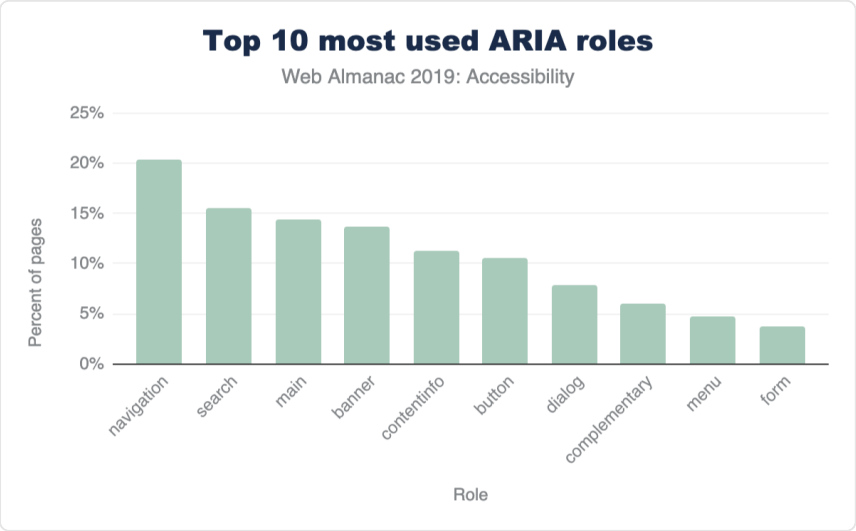


図9.9. ARIAロールトップ10。

図9.9の結果を見ると、2つの興味深い見解が得られます。UIフレームワークを更新すると、Web全体のアクセシビリティおよび操作しやすいダイアログとなっているサイトの数が非常に多くなるようです。

UIフレームワークの更新は、Web全体のアクセシビリティを向上させる方法となる可能性がある

11%以上のページに表示されるトップ5のロールはランドマークロールです。これはコンボボックスなどのヴィジェット機能が何かを説明するためではなく、ナビゲーションを助けるために使われています。ARIAが開発された主な目的は、Web開発者が汎用のHTML要素(`<div>` など)で作られたヴィジェット機能に対して説明を追加できる機能を提供することだったため、これは予想しなかった結果です。

とても良く使われているWeb UIフレームワークは、テンプレートにナビゲーションロールが含まれているはずです。これはランドマーク属性が普及している説明に繋がります。この見解が正しい場合、一般的なUIフレームワークを更新してアクセシビリティサポートを追加すると、Webのアクセシビリティに大きな影響が出る可能性を持っています。

この結論が導き出されるもう1つの答えは、より「高度」で同じくらい重要なARIA属性が一切使用されていないように見えるという事実です。この属性はUIフレームワークを介してかんたんにデプロイすることはできません。なぜなら、このような属性は各サイトの構造と外観に基づいて個々にカスタマイズする必要がある為です。例えば `posinset` や `setsize` 属性は0.01%のページでしか使われていませんでした。これらの属性は一覧やメニューにあるアイテムの数と現在選択されているアイテムを、スクリーンリーダーユーザーに伝えることができます。そのため、視覚障害のあるユーザーがメニューを操作しようとすると「ホーム、5分の1」「製品、5分の2」「ダウンロード、5分の3」というようなインデックスのアナウンスが聞こえます。

多くのサイトは、ダイアログをアクセス可能にしようとしています

スクリーンリーダーを使っているユーザーはダイアログへのアクセスが難しく、見るからにそれがダイアログロール[201]その相対的な人気となっています。そのため、分析されたページの約8%で挑戦しはじめているのを見るのは興奮します。繰り返しますが、これはいくつかのUIフレームワークを使った結果に思えます。

対話的要素のあるラベル

ユーザーがWebサイトを操作する最も一般的な方法は、Webサイトを閲覧するためのリンクやボタンなどのコントロールを使うことです。ただし、殆どの場合においてスクリーンリーダーのユーザーは、活性化されたコントロールが何を実行するのかを判断できません。この混乱が発生する原因の多くは、テキストラベルが無いためです。例えば、左向きの矢印アイコンが表示された「戻る」事を示すボタンですが、テキストが実際は含まれていません。

ボタンまたはリンクを使うページの約4分の1（24.39%）でしか、これらのコントロールにテキストラベルが含まれていませんでした。コントロールにラベルが付いていない場合、スクリーンリーダーのユーザーは「検索」などの意味のある単語ではなく「ボタン」などの一般的なものを読み上げることがあります。

ボタンとリンクはタブオーダーの対象であるため、視認性は非常に高くなります。Tabキーを使ってのWebサイト閲覧は、キーボードだけを使っているユーザーのWebサイト閲覧では普通の事です。なので、ユーザーはTabキーを使ってWebサイトを移動している場合に、ラベルのないボタンとリンクに必ず遭遇するでしょう。

201. *https://developer.mozilla.org/docs/Web/Accessibility/ARIA/Roles/dialog_role*

フォームコントロールのアクセシビリティ

フォームへの入力は私達が毎日行う沢山行う作業です。ショッピングや旅行の予約、仕事の申込みなど、フォームはユーザーがWebページと情報を共有する主な方法です。そのため、フォームを便利にすることは非常に重要です。これを達成するための簡単な方法は、各入力にラベルを提供することです（`<label>`要素や`aria-label`または`aria-labelledby`を用いて）。悲しいことに、すべてのフォーム入力にラベルを提供しているのページは22.33%しかありませんでした。つまり、5ページあるうちの4ページは非常に記入が難しいフォームを持っています。

必須や無効なフィールドであることを示す手がかり

大きなアスタリスクがあるフィールドに出会うと、それが必須フィールドだと理解できます。もしくは、サブミットをクリックして無効な入力があると通知された場合に、異なる色で強調表示されているものは全てを修正してから再送信する必要があります。しかし、視力が低い人や無い人はこのような視覚的合図に頼ることができないため、htmlの入力属性である`required`や`aria-required`と`aria-invalid`などが非常に重要になります。これらは、スクリーンリーダーに対して赤いアスタリスクや赤い強調表示されたフィールドと同等の物を提供します。更に良いことに、必要なフィールドをブラウザに教えればフォームの一部を検証[202]することも可能です。これにはJavaScriptが必要ありません。

フォームを使っているページのうち21.73%は必須フィールドをマークアップするときに`required`か`aria-required`を適用しています。５分の１のサイトでだけ、これらは使用されています。これはサイトを使いやすくするための簡単な手続きです、すべてのユーザーに対してブラウザの役立つ機能を開放します。

フォームを持つサイトの3.52%で`aria-invalid`が使われていることも判明しました。しかし、ほとんどのフォームは誤った情報が送信された後にこのフィールドを参照するため、このマークアップを適用しているサイトの本当の割合を確認することはできませんでした。

重複したID

HTMLでIDを使い２つの要素をリンクさせることができます。例えば`<label>`要素は次のように機能します。ラベルにinputフィールドのIDを指定し、ブラウザはその２つをリンクさせます。結果はどうなるか？　ユーザーはこのラベルをクリックすることでユーザーはinputフィールドにフォーカスすることが可能になり、スクリーンリーダーはこのラベルを説明として使うことができます。

残念ながら34.62%のサイトで重複したIDが確認できます、つまり多くのサイトではユーザ

202. *https://developer.mozilla.org/docs/Learn/HTML/Forms/Form_validation*

一の指定したIDが複数の異なったinputを参照しています。そのため、ユーザーがラベルをクリックしてフィールドを選択すると、意図したものと違う項目が選択される[203]可能性を持っています。想像されている通り、これはショッピングカートのようなものに対して良くない結果をもたらす可能性があります。

この問題はユーザーが選択肢の内容を視覚的に再確認できないスクリーンリーダーユーザーに対してさらに際立ちます。そして、`aria-describedby`や`aria-labelledby`などのARIA属性は上で説明したlabel要素と同じように機能します。つまり、サイトを操作しやすくするには、最初に重複するIDを全て削除するのが良いでしょう。

結論

アクセシビリティを必要としているのは障害のある人々だけではありません。例えば、一時的に手首を負傷している人は小さな操作対象を触れるのが難しいと感じているはずです。視力は年齢とともに低下することが多く、小さなフォントで書かれたテキストは読みにくくなります。指の器用さは年齢毎に異なるため、かなりの割合のユーザーが対話的なコントロールに触れたり、モバイルWebサイトのコンテンツをスワイプしたりするのが難しくなっていきます。

同様に支援ソフトウェアは障害のある人のためだけでなく、すべての人の日々の体験を良くしていくためのものです。

- モバイルデバイスと家庭用の両方で音声認識が人気なのは、コンピューティングデバイスを音声コマンドで制御することが多くのユーザーにとって有意義であり不可欠であることを示しています。このような音声制御は、以前はアクセシビリティ機能にしかありませんでしたが現在では商品の主流になりつつあります。
- 運転手は道路から目を離す事無くニュース記事のような長文を読み上げるスクリーンリーダーの恩恵を受けることができるでしょう。
- 字幕はビデオを聞く事ができない人だけでなく、騒々しいレストランや、図書館でビデオを見たい人なども楽しませています。

一度Webサイトが完成すると、既存のサイト構成とウィジェットに対してアクセシビリティを改良する事は殆どの場合で困難になります。アクセシビリティは後で簡単にデコレーションすることが出来るものではなく、設計と実装のプロセスとして必要になります。しかし残念ながら、認識不足または使いやすいテストツールのせいで多くの開発者は、すべてのユーザーのニーズと支援ソフトウェアの要件に精通していません。

これは結論ではありませんが、私達の活動結果からARIAやアクセシビリティのベストプラ

203. *https://www.deque.com/blog/unique-id-attributes-matter/*

クティス（代替テキストを使うなど）のようなアクセシビリティ標準の利用はWebのかなりの、*しかし実質的でない*部分で見つかることが示されています。表面的にはこれは励みになりますが、こういった良い方向にある事柄の多くは特定のUIフレームワークがよく利用されているからだと私達は訝しんでいます。一方、Web開発者にとってはシンプルにUIフレームワークを頼ってサイトにアクセシビリティを差し込むことはできないため、非常に期待はずれです。その一方で、UIフレームワークがWebのアクセシビリティに与える影響の大きさを確認できるのは心強いでしょう。

私達の見解では次の開拓地は、UIフレームワークを介して利用可能なウィジェットをより簡単に操作できるようになることです。世の中で使われている多くの複雑なウィジェット（カレンダーのピッカーなど）はUIライブラリなどに含まれており、こういったウィジェットがそのまま使えるのは素晴らしいことです。私達は次の結果を集める時、より適切に実装された複雑なARIAロールの利用が増えて、より複雑なウィジェットに対しても操作が簡単になっていることを願っています。そして、すべてのユーザーがウェブの豊かさを楽しむことが出来るよう、映画やビデオなどがさらにアクセスしやすいメディアとなった未来を見たいと思います。

著者

Nektarios Paisios

Nektarios Paisios氏は、過去5年間、Chromeアクセシビリティに取り組んできたソフトウェアエンジニアです。主にスクリーンリーダーや拡大鏡などのサードパーティの支援ソフトウェアとChromeの互換性を高めることに注力しています。Chromeアクセシビリティに携わる前は、GSuiteアクセシビリティやディスプレイ広告など、さまざまな役割を担っていました。Nektariosはニューヨーク大学でコンピュータサイエンスの博士号を取得しています。

David Fox

@theobto　foxdavidj　https://www.lookzook.com

David Foxは、ユーザビリティ・リサーチャーのリーダーであり、LookZookの創設者でもあります。彼はウェブサイト心理学者であり、ユーザーが何に悩んでいるのかだけでなく、その理由や体験を改善するための最善の方法を学ぶためにサイトを掘り下げています。また、Google Chromiumのコントリビューターであり、講演者であり、ウェビナーやUXトレーニングの提供者でもある。

Abigail Klein

kleinab

Abigail Kleinは、Googleのソフトウェアエンジニアです。彼女は Google Docs、Sheets、Slides のウェブアクセシビリティに取り組み、自動キャプションを Google Slides に追加しました[204] また、sc reen リーダー、点字、スクリーン拡大鏡、ハイコントラストのサポートを改善しました。現在は、Google Chrome と ChromeOS のアクセシビリティに取り組んでいます。MIT でコンピュータサイエンスの学士号と修士号を取得しており、支援技術ハッカソンを共同で立ち上げ、支援技術クラスのラボアシスタントとゲスト講師を務めました。

204. *https://www.blog.google/outreach-initiatives/accessibility/whats-you-say-present-captions-google-slides/*

部 II 章 10
SEO

Yvo Schaap、 Rachel Costello と Martin Splitt によって書かれた。
John Fox、 Andrew Limn、 Aymen Loukil、 Catalin Rosu と Matt Ludwig によってレビュ。
Yvo Schaap による分析。
Rachel Costello 編集。
M.Sakamaki によって翻訳された。

導入

検索エンジン最適化（SEO）はデジタルマーケティングの担当者にとって、単なる趣味やサイドプロジェクトではなくWebサイトを成功に導くための重要な要素です。SEOの主な目標は、Webサイトをクロールする必要のある検索エンジンボットと、Webサイトを操作するコンテンツの消費者向けにWebサイトを最適化することです。SEOはWebサイトを作っている開発者から、新しい潜在顧客に対して宣伝する必要のあるデジタルマーケティング担当者に至るまで、すべてのWebサイトに関わる人に影響を及ぼします。

それでは、SEOの重要性を見ていきましょう。今年のはじめにSEO業界は「困難な年」と言われた後ASOSが利益の87%減少[205]を報告したため、その恐ろしさ(と魅力)への注目が集まりました。このブランドの問題は、200以上のマイクロサイトを立ち上げた為に発生した検索エンジンのランキング低下と、Webサイトのナビゲーションの大幅な変更などの技術的変更

205. https://www.bbc.co.uk/news/business-47877688

が原因であると考えられています。驚きです！

Web AlmanacのSEOの章の目的は、検索エンジンによるコンテンツのクロールとインデックス付け、そして最終的にWebサイトのパフォーマンスに影響を与えるWebのオンサイト要素を分析することです。この章では、上位のWebサイトがユーザーおよび検索エンジンに優れた体験を提供するためにどれだけ十分な整備がされているか、そしてどのような作業を行うべきかについて見ていきます。

この分析には、Lighthouseと、Chrome UX Report、HTML要素の分析データが含まれています。`<title>`要素、様々な種類のページ上リンク、コンテンツ、読み込み速度などによるSEOの基礎だけではなく、500万以上のインデクサビリティ、構造化データ、国際化、AMPなどSEOのさらなる技術的な面にも注目しています。

カスタムメトリクスはこれまで公開されていなかった洞察を提供します。`hreflang`タグ、リッチリザルトの適格性、見出しタグの使用、さらにシングルページアプリケーションのアンカーによるナビゲーション要素などの採用と実装について断言できるようになりました。

注意：データはホームページの分析のみに限定されており、サイト全体のクロールから集計はされていません。そのようにした理由は、以降説明する多くの指標に影響を与えるため、特定の指標について言及する場合に関連する制限を追加しました。制限の詳しい内容はMethodologyを御覧ください。

では、Webの現状と検索エンジンの使いやすさについての詳細をお読みください。

基本

検索エンジンにはクロール、インデックスの作成、ランキングの３つの手順があります。それには検索エンジンにとって、ページを見つけやすく、理解でき、search engine results pages(SERPs)（検索エンジンの結果ページ）を閲覧しているユーザーにとって価値が有り高品質なコンテンツが含まれている必要があります。

SEOの基本的なベストプラクティスの基準を満たしているWebの量を分析するため、本文のコンテンツ、`meta`タグ、内部リンクなどのページ上の要素を評価しました。それでは結果を見てみましょう。

コンテンツ

ページの内容を理解して、最も関連性の高い回答を提供できる検索クエリが何かを決定的にするためには、まず検索エンジンがそのコンテンツを探し出してアクセス出来る必要があります。しかし、検索エンジンは現在どのようにコンテンツを見つけるのでしょうか？、その回答を得るために、単語数と見出しの２つのカスタムメトリクスを作りました。

単語の数

私達は、少なくとも３つの単語グループを探し合計でいくつ見つかったかを数えるようにして、ページのコンテンツを評価しました。デスクトップページには単語グループを持たないものが2.73%見つかりました。これはWebサイトが何を指しているのかを検索エンジンが理解するのに役立つ本文コンテンツが無いことを示しています。

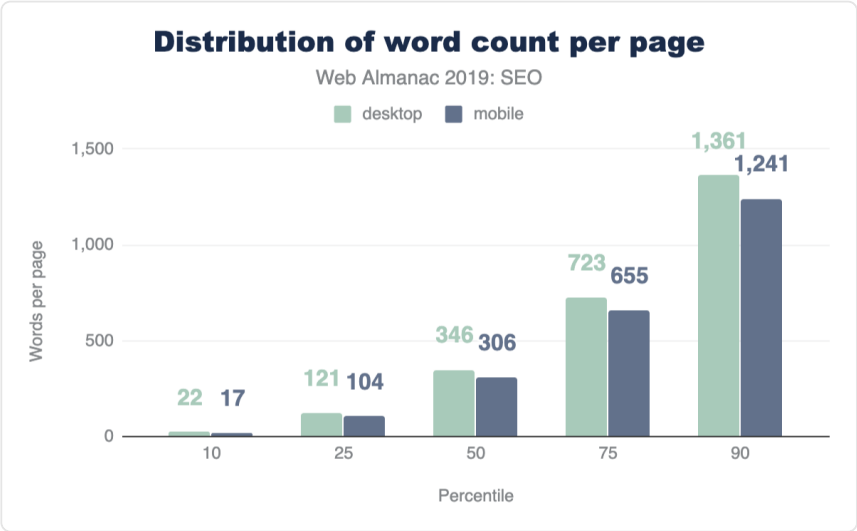


図10.1. ページ毎の単語数分布。

デスクトップ向けホームページの中央値は346単語で、モバイル向けホームページの中央値は306単語とわずかに少ない単語数になっています。これはモバイル向けサイトが少し少ない量をユーザーにコンテンツとして提供していることを示していますが、300単語を超えたとしても読むのには問題ない量でしょう。これは、例えばホームページは記事があるページなどよりコンテンツが自然と少なくなるため特に該当するでしょう。全体を見ると単語の分布は幅があり、10パーセンタイルのあたりでは22単語、90パーセンタイルあたりで最大1,361単語です。

見出し

また、ページに含まれるコンテンツのコンテキストが適切な方法で構造化されて提供されているかを調べました。見出し（`H1`、`H2`、`H3`、など）を使ってページを整え構造化すれば、コンテンツは読みやすく、解析しやすいようになります。そんな見出しの重要性にもかかわらず、ページの10.67%には見出しタグがまったくありませんでした。

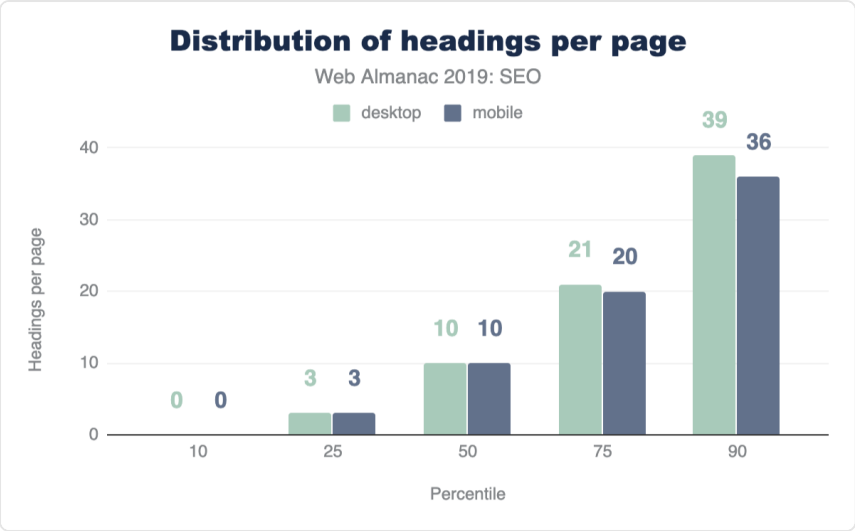


図10.2. ページ毎の見出し数分布。

1ページあたりの見出し要素の中央値は10となっています。見出しにはモバイルページで30単語、デスクトップページで32単語が含まれています。これは、見出しを活用できているWebサイトが、ページが読みやすく、説明的で、ページの構造とコンテキストを検索エンジンボットに明確に概説することに多大な労力を費やしていることを意味します。

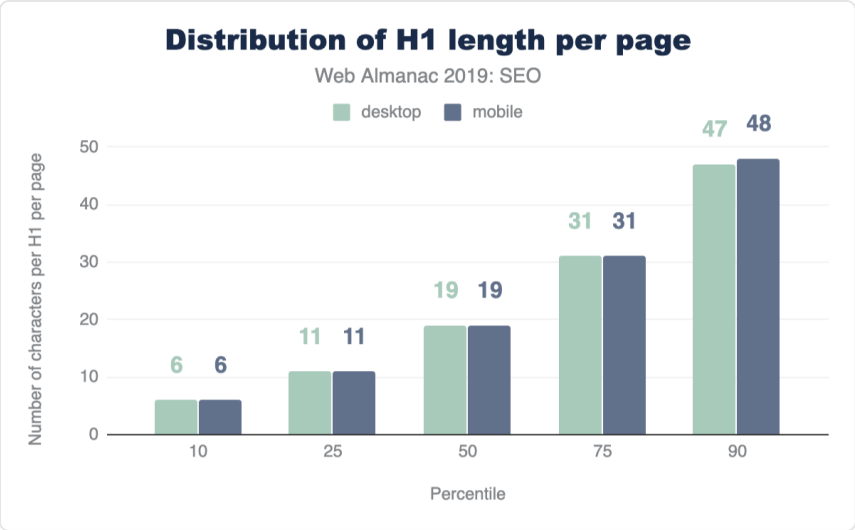


図10.3. ページ毎のH1の長さの分布。

具体的な見出しの長さを見ると、最初に見つかった `H1` 要素の長さの中央値はデスクトップで19文字です。

SEOとアクセシビリティのための `H1` と見出しの処理に関するアドバイスについては、Ask Google WebmastersシリーズのJohn Muellerによるこのビデオの回答[206]をご覧ください。

メタタグ

メタタグを使用すると、ページ上の色々な要素やコンテンツに関する特定の指示や情報を検索エンジンボットに提供できます。特定のメタタグにはページの注目すべき情報や、クロールとインデックス付けの方法などを伝えることができます。私達はWebサイトの提供するメタタグが、これらの機会を最大限に活用しているかどうかを評価したいと考えました。

ページのタイトル

97%

図10.4. `<title>` タグを含むモバイルページの割合。

ページのタイトルはページの目的をユーザーや検索エンジンに伝える重要な手段です。`<title>` タグはSERPSの見出にも、ページにアクセスする時のブラウザーのタブのタイトルとしても使われるので、モバイルページの97.1%にドキュメントタイトルが存在することは驚くことではないでしょう。

206. *https://www.youtube.com/watch?v=zyqJJXWk0gk*

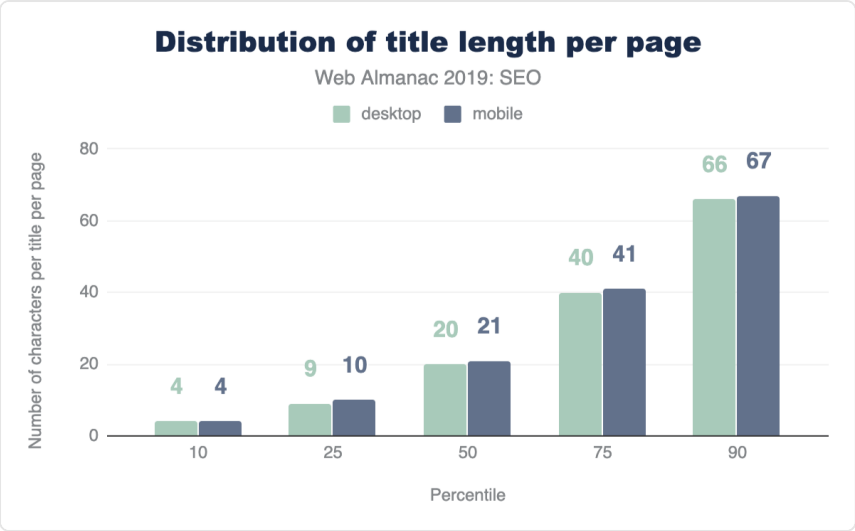


図10.5. ページごとのタイトルの長さの分布。

一般的にGoogleのSERPはページタイトルの最初の50〜60文字を表示[207]しますが、`<title>`タグの長さの中央値はモバイルページで21文字、デスクトップページで20文字でした。75パーセンタイルでも、境界を下回っています。これは、一部のSEOとコンテンツの記者が、検索エンジンによって割り当てられたSERPsのホームページを記述するために割り当てられた領域を最大限利用できていないことを示します。

メタディスクリプション

`<title>`タグと比べると、メタディスクリプションが検出されたページは少なくなっており、モバイル用ホームページの64.02%にだけメタディスクリプションが設定されています。Googleが検索者のクエリに応じてSERP内のメタディスクリプションの記述を頻繁に書き換えることを考慮すると、おそらくWebサイトの所有者はメタディスクリプションを含めることを重要視しないでしょう。

207. *https://moz.com/learn/seo/title-tag*

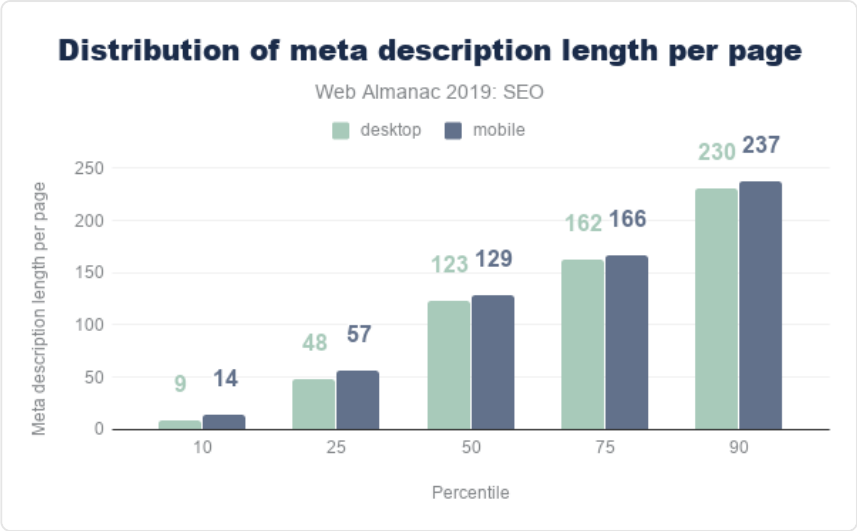


図10.6. ページ毎のメタ記述の長さ分布。

メタディスクリプションの長さは155-160文字が推奨[208]となっていますが、デスクトップページの中央値ははそれより短い123文字となっています。さらに興味深い事があります、モバイルのSERPはピクセル制限により従来よりも短かくなるにも関わらず、メタディスクリプションは一貫してデスクトップよりもモバイルが長くなっています。この制約は最近拡張されたばかりなので、おそらく多くのWebサイトの所有者がモバイルの結果に対して、より長くて説明的なメタディスクリプションの影響を確認しているのでしょう。

画像の `alt` タグ

SEOとアクセシビリティのための `alt` テキストの重要性を考えたとき、モバイルページの46.71%でしか画像に `alt` 属性が使われていないのを見ると理想とはほど遠い状況です。これは、Web上の画像をユーザーにとってさらにアクセスしやすく、検索エンジンにとって理解しやすくすることに関しては、まだ改善する点が多く有ることを意味します。この問題に対する詳細についてはアクセシビリティの章を御覧ください。

インデクサビリティ

SERPでユーザーにページのコンテンツを表示するためには検索エンジンのクローラーがそのページにアクセスしてインデックスを作れるようにする必要があります。検索エンジンによるページのクロールとインデックス登録の機能に影響を与える要因には次の様なものがあ

208. https://moz.com/learn/seo/meta-description

ります。

- ステータスコード
- `noindex` タグ
- canonicalタグ
- `robots.txt` ファイル

ステータスコード

検索エンジンにインデックスを付けたい重要なページは常に `200 OK` ステータスコードにしておく事をお勧めします。テストされたページの殆どは検索エンジンにアクセス可能で、デスクトップでは最初のHTML要求の87.03%が `200` ステータスコードを返しました。モバイルの場合は少しだけ低く、82.95%のページだけが `200` となるステータスコードを返しました。

モバイルでは次によく見られるステータスコードは一時的なリダイレクトである `302` となっており、これはモバイルページの10.45%で見つけることができました。この結果はデスクトップよりも多く、デスクトップ用のホームページで `302` ステータスコードを返すのは6.71%しかありませんでした。これは、モバイル用のホームページがレスポンシヴでなくデバイスごとにWebサイトのバージョンが異なるような[209]、デスクトップページの代替が用意されていることに起因している可能性があります。

注意：この結果にはステータスコード `4xx` と `5xx` は含んでいません。

`noindex`

`noindex` 指示はHTMLの `<head>` もしくはHTTPヘッダーの `X-Robots` 指示で使うことができます。 `noindex` 指示は基本的に検索エンジンにそのページをSERPに含めないように指示しますが、ユーザーがWebサイトを操作しているときでもページはアクセス可能です。一般的に `noindex` 指示は、同一コンテンツを提供するページの複製バージョン、またはオーガニック検索からWebサイトにアクセスするユーザーに価値を提供しないであろう低品質のページ(フィルタ、ファセット、内部検索ページなど)に追加されます。

モバイル用ページの96.93%がLighthouseのインデックス作成監査に合格しており[210]、これらのページには `noindex` 指示が含まれていませんでした。ただし、これはモバイルホームページの3.07%に `noindex` 指示が含まれていたことも意味しています。これは心配の種であ

209. *https://developers.google.com/search/mobile-sites/mobile-seo/separate-urls*
210. *https://developers.google.com/web/tools/lighthouse/audits/indexing*

り、Googleはこれらのページのインデックスを作成できないことを意味しています。

私達の調査に含まれるWebサイトはChrome UX Reportのデータセットから提供されていますが、公開されていないWebサイトは除外されています。これはChromeが非公開であると判断したサイトは分析できないので、バイアスの重要な源です。これについては方法論の詳細を御覧ください。

canonicalによる最適化

canonicalタグを使い重複ページと優先代替ページを指定します。これにより検索エンジンは、グループ内の複数のページに散っているオーソリティを1つのメインページに統合してランキングの結果を上げることができます。

モバイル用ホームページの48.34%でcanonicalタグが使われていることが検出[211]されました。自分を指ししめすcanonicalタグは必須でなく、普通は複製されたページにcanonicalタグを必要とします。ホームページがサイトのどこか他の場所に複製されることはめったに無いので、canonicalタグがページ毎で半分未満になっているのは驚くことではありません。

robots.txt

検索エンジンのクロールを制御する最も効果的な方法の1つは、`robots.txt`ファイルです。これは、Webサイトのルートドメインに置かれる事で、検索エンジンのクロールに対し許可しないURLとURLパスを指定する事ができるファイルです。

Lighthouseの結果[212]からモバイル用サイトの72.16%でしか有効な`robots.txt`を持っていないことがわかりました。見つかった問題の主な内訳は、`robots.txt`ファイルをまったく持たないサイトが22%、無効な`robots.txt`ファイルを提供する約6%で、それぞれ検査に失敗しています。クロールバジェットの問題[213]に悩まされないような小規模Webサイトを運営していたりするなど、`robots.txt`ファイルを持たない妥当な理由もあったりしますが、無効な`robots.txt`が有るというのは、それだけで心配の種になります。

リンク

Webページの最も重要な属性の1つはリンクです。リンクは検索エンジンがインデックスに追加してWebサイトをナビゲートするための新しい関連ページを発見するのに役立ちます。データセットにあるWebページの96%には最低でも1つの内部リンク存在し、93%は少なくとも 1 つの別ドメインへの外部リンクが存在しています。内部や外部リンクを持たないごく一部のページは、ターゲットページへ通じるリンクという大きな価値を取りこぼしていま

211. *https://developers.google.com/web/tools/lighthouse/audits/canonical*
212. *https://developers.google.com/web/tools/lighthouse/audits/robots*
213. *https://webmasters.googleblog.com/2017/01/what-crawl-budget-means-for-googlebot.html*

す。

デスクトップ用のページに含まれる内部と外部リンクの数は、モバイル用のページよりも全ての場合で多くなっています。これは殆どの場合、モバイルのデザインはビューポートが小さく空間が限られているために、リンクが含まれるテキストはデスクトップに比べて少なくなっているためです。

モバイル用のページで内部リンクが少ない場合、Webサイトで問題が発生する可能性[214]が有るため注意が必要です。新しいWebサイトでGoogleの規定であるモバイルファーストインデックス[215]が適用されると、そのページがデスクトップ用ではリンクされているがモバイル用からリンクが無い時、検索エンジンはそのページを見つけてランク付けするのがとても難しくなってしまいます。

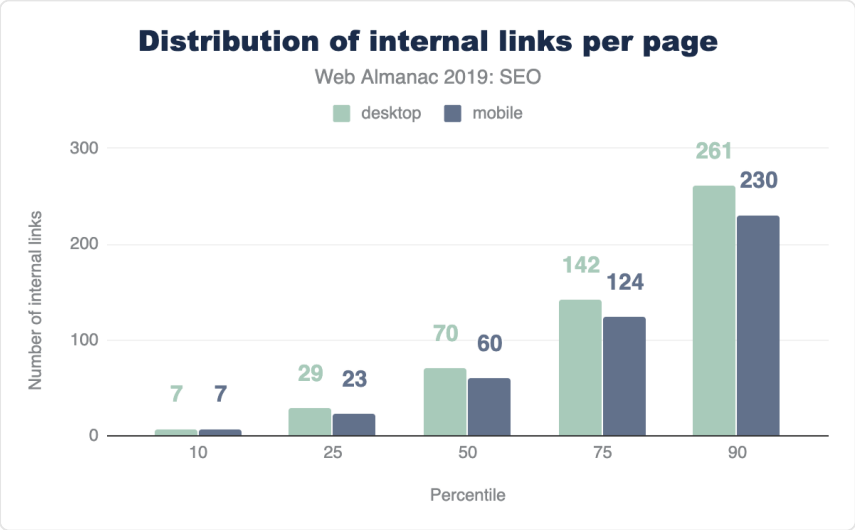


図10.7. ページ毎の内部リンク分布。

214. https://moz.com/blog/internal-linking-mobile-first-crawl-paths
215. https://www.deepcrawl.com/knowledge/white-papers/mobile-first-index-guide/

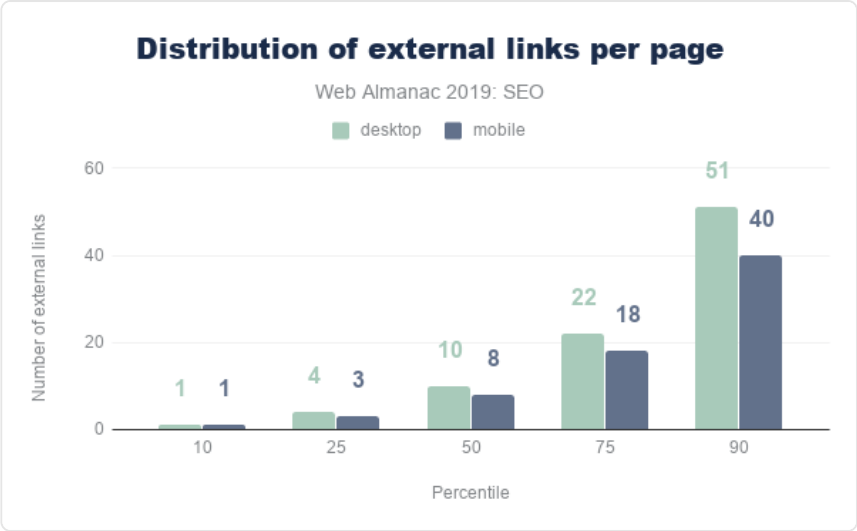


図10.8. ページ毎の外部リンク数の分布。

デスクトップ用ページの内部リンク(同一サイト)数は中央値で70となっていますが、モバイル用ページの内部リンク数の中央値は60になっています。外部リンク数のページ毎中央値も同じような傾向となっており、デスクトップ用ページの外部リンク数は10で、モバイル用ページは8になっています。

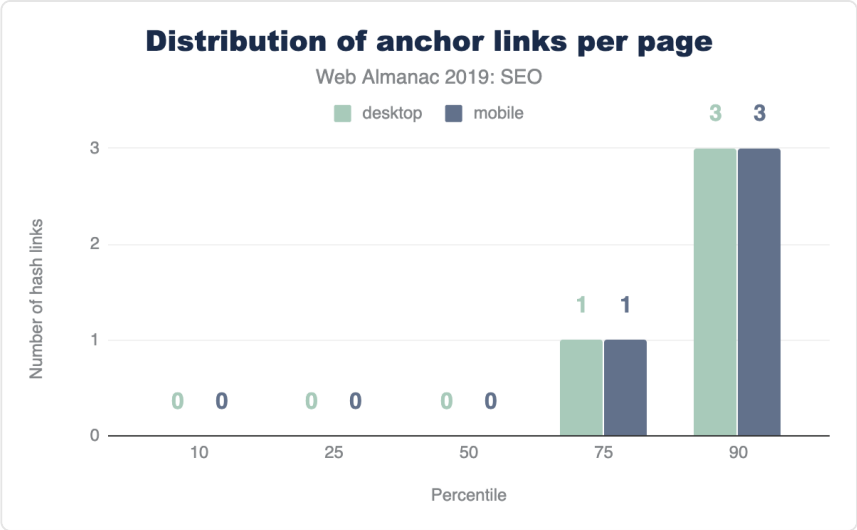


図10.9. ページ毎のアンカーリンク数の分布。

同一ページの特定スクロール位置にリンクするアンカーリンクはあまり人気が無いようです。ホームページの65%以上でアンカーリンクは使われていません。これはおそらく、一般的なホームページには長文形式のコンテンツが含まれていないからでしょう。

説明的なリンクテキストの測定からは良いニュースが伺えます。モバイル用ページの89.94%がLighthouseの説明的なリンクテキストの監査[216]で合格しています。つまり、これらのページは一般的な「ここをクリック」「リンク」「続きを読む」「全文表示」のようなリンクを使わず、より有意義なリンクテキストを使うことで、ユーザーと検索エンジンにページのコンテキストやページ同士のつながりがあることを理解できるようにしています。

Advanced

説明的で有用なコンテンツ以外に対して`noindex`や`Disallow`という指示を出してページを検索エンジンからブロックするだけでは、Webサイトをオーガニックサーチさせるには不十分です。これらは単なる基本でしかありません。WebサイトのパフォーマンスやSERPsの外観を向上させるなど、できることはたくさんあります。

Webサイトのインデックス作成とランク付成功のために重要となっている技術的に複雑な局面として、速度、構造化データ、国際化、セキュリティ、モバイルフレンドリーなどがあります。

Speed

モバイルの読み込み速度は、2018年にGoogleからランキング要素として初めて発表[217]されました。しかしGoogleにとって速度は新しい観点ではありません。2010年に既に速度がランキングシグナルとして導入されたことが明らか[218]にっています。

Webサイトが高速であることは、優れたユーザー体験のためにも重要です。サイトの読み込みに数秒待たされるユーザは、すぐ離脱してSERPsから別の似たような内容の素早く読み込まれるページを探す傾向があります。

Web全体の読み込み速度の分析に使った指標は Chrome UX Report（CrUX）を基にしています。このレポートは、実際のChromeユーザーからデータを収集します。このデータで驚くべき点は、48%のWebサイトが**遅い**とラベル付されていることです。FCPの25%が3秒より遅い場合、もしくはFIDの5%が300ミリ秒より遅い場合にWebサイトは低速とラベル付されます。

216. *https://developers.google.com/web/tools/lighthouse/audits/descriptive-link-text*
217. *https://webmasters.googleblog.com/2018/01/using-page-speed-in-mobile-search.html*
218. *https://webmasters.googleblog.com/2010/04/using-site-speed-in-web-search-ranking.html*

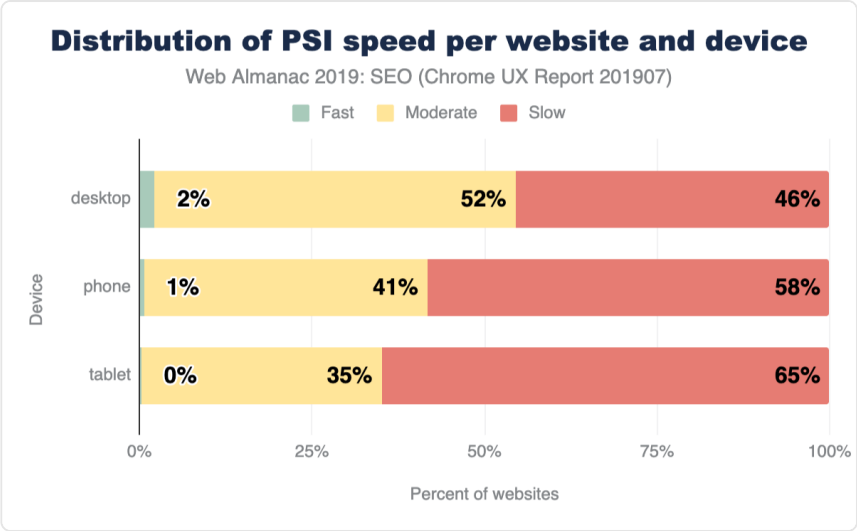


図10.10. デバイスタイプごとのユーザー体験パフォーマンスの分布。

デバイスごとに分けるとより鮮明になります、この画像ではタブレット(65%)、電話(58%)を示しています。

数字だけ見るとWebの速度には暗雲が立ち込めるように思えますが、良いニュースもあります。それはSEOの専門家とツールがWebサイトの高速化のための技術課題に集中しているという点です。Webパフォーマンスの状態についてはパフォーマンスの章で詳しく知ることができます。

構造化データ

構造化データを使うことでWebサイトの所有者は、JSON-LD[219]スニペットやMicrodata[220]などを加える事で、Webページに付属的なセマンティックデータを付与できます。検索エンジンはこのデータを解析してこれらのページを深く理解し、マークアップにより検索結果に追加の関連情報を表示も行う事ができます。よく見る構造化データの種類には次のようなものがあります。

- reviews[221]
- products[222]

219. https://en.wikipedia.org/wiki/JSON-LD
220. https://developer.mozilla.org/docs/Web/HTML/Microdata
221. https://developers.google.com/search/docs/data-types/review-snippet
222. https://developers.google.com/search/docs/data-types/product

- businesses[223]
- movies[224]
- その他[225]

構造化データがWebサイトに提供できる追加の可視性[226]はユーザーがサイトに訪れる機会を増やすのに役立つため、サイトの所有者にとっては魅力的です。たとえば、比較的新しいFAQスキーマ[227]はスニペットとSERPsの領域を２倍にできます。

調査の結果、モバイルでリッチな結果を得ることが出来るサイトは14.67%しか無いことが解りました。興味深いことに、デスクトップサイトの適格性はわずかに低くなり12.46%となっています。これはサイト所有者がホームページ検索で表示されるための最適化に対して、もっと出来ることが有ることを示しています。

構造化データのマークアップを持つサイトの中で、最もよく見る種類は次の５つでした。

1. `WebSite` (16.02%)
2. `SearchAction` (14.35%)
3. `Organization` (12.89%)
4. `WebPage` (11.58%)
5. `ImageObject` (5.35%)

興味深いことに、一番良く利用されている検索エンジンの機能をトリガーするデータ型はサイトリンクの検索ボックス[228]を強化する `SearchAction` です。

トップ5のマークアップタイプはすべてGoogleの検索結果の可視性を高める物で、これらのタイプの構造化データをさらに採用する理由になるかもしれません。

今回の分析はホームページだけを見ているため、インテリアページも考慮した場合は結果は大きく異なった結果が見えてくる可能性があります。

レビューの星はWebのホームページ上で1.09%だけにしかありません。(AggregateRating[229]より)また、新しく導入されたQAPage[230]は48の例しかなく、FAQPage[231]は少しだけ高い数が出現して218となっています。この最後の2種類の数については、クロールを更に実行してWeb Almanacの分析を掘り下げていくと、将来増加することが予想されています。

223. https://developers.google.com/search/docs/data-types/local-business
224. https://developers.google.com/search/docs/data-types/movie
225. https://developers.google.com/search/docs/guides/search-gallery
226. https://developers.google.com/search/docs/guides/enhance-site
227. https://developers.google.com/search/docs/data-types/faqpage
228. https://developers.google.com/search/docs/data-types/sitelinks-searchbox
229. https://schema.org/AggregateRating
230. https://schema.org/QAPage
231. https://schema.org/FAQPage

国際化

一部のGoogle検索の従業員[232]によれば、国際化はSEOの最も複雑な面の1つとなっているようです。SEOの国際化は、ユーザーが特定の言語のコンテンツをターゲットしていることを確認し、それに合わせて複数の言語や国のバージョンを持つWebサイトから適切なコンテンツを提供することに重点をおいています。

HTML lang属性が英語に設定されているデスクトップ用サイトの38.40%（モバイルでは33.79%）で、別の言語バージョンへの `hreflang` リンクが含まれるサイトはたった7.43%（モバイルで6.79%）しかありませんでした。これから、分析したWebサイトの殆どが言語ターゲティングを必要とするホームページの別バージョンを提供していないことを示しています。しかしそれは、個別のバージョンは存在するが構成が正しく無い場合を除きます。

232. *https://x.com/JohnMu/status/965507331369984002*

hreflang	Desktop	Mobile
en	12.19%	2.80%
x-default	5.58%	1.44%
fr	5.23%	1.28%
es	5.08%	1.25%
de	4.91%	1.24%
en-us	4.22%	2.95%
it	3.58%	0.92%
ru	3.13%	0.80%
en-gb	3.04%	2.79%
de-de	2.34%	2.58%
nl	2.28%	0.55%
fr-fr	2.28%	2.56%
es-es	2.08%	2.51%
pt	2.07%	0.48%
pl	2.01%	0.50%
ja	2.00%	0.43%
tr	1.78%	0.49%
it-it	1.62%	2.40%
ar	1.59%	0.43%
pt-br	1.52%	2.38%
th	1.40%	0.42%
ko	1.33%	0.28%
zh	1.30%	0.27%
sv	1.22%	0.30%
en-au	1.20%	2.31%

図10.11. よく見る `hreflang` 値のトップ25。

英語の次に最もよく見る言語は、フランス語、スペイン語、およびドイツ語です。この後にアメリカ人向けの英語（`en-us`）やアイルランド人向けのスペイン語（`es-ie`）などの不明瞭な組み合わせなどの、特定の地域を対象とした言語が続いています。

この分析では、異なる言語バージョン同士が相互で適切にリンクしているかどうかなどの正しい実装は確認しませんでした。しかし、推奨にある[233]x-defaultバージョン（デスクトップでは3.77%、モバイルでは1.30%）の採用が少ない点を考慮すると、この要素が複雑で常に正しいとは限らないということを示しています。

SPAのクロールに関する可能性

ReactやVue.jsなどのフレームワークで構築されたシングルページアプリケーション（SPA）には、独特のSEOの複雑さが伴っています。ハッシュを使ったナビゲーションを使用するWebサイトは検索エンジンがクロールして適切にインデックスを作成するのがとても難しくなります。例を上げると、Googleには「AJAXクロールスキーム」という回避策がありましたが、開発者だけでなく検索エンジンにとっても難解であることが判明し、この仕様は2015年に廃止[234]されました。

ハッシュURLを介して提供されるリンクの数が比較的少なく、Reactモバイルページの13.08%がナビゲーションにハッシュURLを使用し、モバイルVue.jsページで8.15%、モバイルAngularページで2.37%で使用されているという結果になっています。この結果はデスクトップ用ページでも非常に似通った結果でした。ハッシュURLからコンテンツの発見に対する影響を考慮すると、この結果はSEOの観点からは良い状態と言えるでしょう。

特に驚いた点は、ハッシュURLの数がAngularページでは少ないのとは対照的に、Reactページでのハッシュ URLの数が多くなっている点です。両方のフレームワークはハッシュURLに依存せず、代わりにリンク時にHistory API[235]が標準となっているルーティングパッケージの採用を推奨しています。Vue.jsは `vue-router` パッケージのバージョン3から、History APIを標準で使うことを検討[236]しています。

AMP

AMP（以前は「Accelerated Mobile Pages」として知られていました）は、2015年にGoogleによってオープンソースのHTMLフレームワークとして初めて導入されました。キャッシュ、遅延読み込み、最適化された画像などの最適化手法を使うことで、Webサイトのサイト

233. *https://www.google.com/url?q=https://support.google.com/webmasters/answer/189077?hl%3Den&sa=D&ust=1570627963630000&usg=AFQjCNFwzwglsbysT9au_I-7ZQkwa-QvrA*
234. *https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html*
235. *https://developer.mozilla.org/docs/Web/API/History*
236. *https://github.com/vuejs/rfcs/pull/40*

のコンポーネントと基盤構造を提供することで、ユーザーに高速な体験を提供します。特に、Googleは検索エンジンにもこれを採用し、AMPページも独自のCDNから提供されています。この機能は後にSigned HTTP Exchanges[237]という名前の標準提案になりました。

にも関わらず、AMPバージョンへのリンクが含まれるモバイルホームページはわずか0.62%しかありません。このプロジェクトの可視性を考慮しても、これは採用率が比較的低い事が示されています。ただし、今回のホームページに焦点を宛てた分析なので、他のページタイプの採用率は見ていません、記事ページを配信する場合はAMPのほうが有利な場合が多いでしょう。

セキュリティ

近年、WebがデフォルトでHTTPSに移行するという強力なオンラインの変化がありました。HTTPSでは、例えばユーザー入力データが安全に送信されないパブリックWi-Fiネットワークでも Webサイトのトラフィックが傍受されるのを防ぎます。GoogleはサイトでHTTPSを採用するよう推進しており、ランキングシグナルとしてHTTPS[238]を作りました。Chromeはブラウザで非HTTPSページを非セキュア[239]としてラベル付けすることでセキュアなページへの移行もサポートしています。

HTTPSの重要性とその採用方法に関するGoogleの詳細な情報と手引については、HTTPSが重要な理由[240]をご覧ください。

現在、デスクトップ用Webサイトの67.06%がHTTPS経由で配信されていますWebサイトの半分以下がまだHTTPSに移行しておらず、ユーザーに安全でないページを提供しています。これはかなりの数です。移行は大変な作業になる場合が多く、そのために採用率が高くない可能性がありますが、HTTPSの移行に必要なのは大抵の場合SSL証明書と `.htaccess` ファイルの簡単な変更です。 HTTPSに切り替えない理由はありません。

Googleの透明性レポート[241]では、Google以外の上位100ドメインでhttpsの採用率は90%であると報告されています(これは世界中のWebサイトトラフィックの25%です)。この数字と私たちの数字の違いから、比較的小規模なサイトはゆるやかにHTTPSを採用しているという事実によって説明できます。

セキュリティの状態の詳細については、セキュリティの章を御覧ください。

237. *https://wicg.github.io/webpackage/draft-yasskin-http-origin-signed-responses.html*
238. *https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html*
239. *https://www.blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/*
240. *https://developers.google.com/web/fundamentals/security/encrypt-in-transit/why-https*
241. *https://transparencyreport.google.com/https/overview*

結論

分析の結果、ほとんどのWebサイトでは基礎がしっかりしている事が判明しました。ホームページはクロール可能で、インデックス付け可能で、検索エンジンの結果ページでのランキングに必要な主要コンテンツが存在しています。Webサイトを所有する人々がSEOを熟知しているわけではなく、ベストプラクティスの指針などは言うまでもありません。つまり、これらの非常に多くのサイトが基本をカバーしていることは非常に頼もしいことです。

しかし、SEOとアクセシビリティのより高度な面のいくつかに関しては、予想していたよりも多くのサイトが注目していません。サイトの速度については、特にモバイルのときに多くのWebサイトが苦労している要因の一つになっており、これは大きな問題です。なぜなら速度はUXの最大の要因の1つで、ランキングに影響を与える可能性があるためです。HTTPS経由でまだ提供されていないWebサイトの数も、セキュリティの重要性を考慮してユーザーデータを安全に保つという点に問題があるように見えます。

私達全員がSEOのベストプラクティスを学んだり、業界の発展に貢献できることはたくさんあります。これは、検索業界が進化する性質を持ちながら、その変化の速度から必要な事です。検索エンジンは毎年数千のアルゴリズムを改善しています、Webサイトがオーガニックサーチでより多くの訪問者に届くようにしたい場合、置いていかれないようにする必要があります。

著者

Yvo Schaap

@yvoschaap　ymschaap　https://build.amsterdam/

技術的なSEOコンサルタントbuild.amsterdam[242]の創設者。以前は、10億人以上のユーザーに達したいくつかのウェブ会社を設立しました。2005年以来、彼の最新の（広告）ベンチャーについてブログを書いている。

242. *https://build.amsterdam/*

Rachel Costello

@rachellcostello　rachellcostello

Rachel Costelloは、DeepCrawl[243]のテクニカルSEO＆コンテンツマネージャーであり、検索の最新動向を研究し、伝えることに時間を費やしている国際会議のスピーカーでもあります。Rachelは現在、技術的なSEOのホワイトペーパー[244]とDeepCrawlのための研究作品の制作を管理しており、Search Engine Journal[245]のレギュラーコラムニストでもあります。

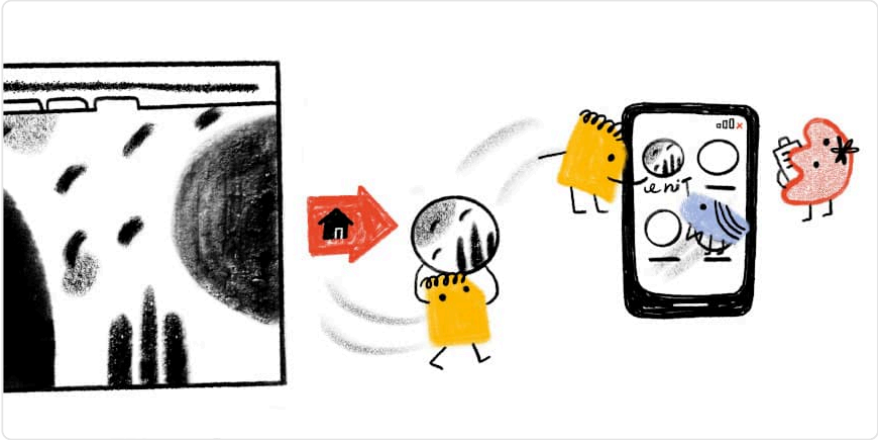
Martin Splitt

@g33konaut　AVGP　http://geekonaut.de

Martin SplittはGoogleのウェブエコシステムチームの開発者アドボケートで、ウェブを発見しやすい状態に保つことに取り組んでいます。

243. https://www.deepcrawl.com/
244. https://www.deepcrawl.com/knowledge/white-papers/
245. https://www.searchenginejournal.com/author/rachel-costello/

部 II 章 11
PWA



Thomas Steiner と Jeff Posnick によって書かれた。
John Teague と Ahmad Awais によってレビュ。
Jason Haralson による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

導入

プログレッシブWebアプリ（PWA）は、Service Worker API[246]などのプラットフォームプリミティブ上に構築される新しいクラスのWebアプリケーションです。Service Workerは、ネットワークプロキシとして機能し、Webアプリの発信要求をインターセプトしプログラムまたはキャッシュされた内容で応答することによりアプリがネットワークに依存しない読み込みをサポートできるようにします。Service Workerは、プッシュ通知を受信し、対応するアプリが実行されていなくてもバックグラウンドでデータを同期できます。さらに、Service Workerは、Webアプリマニフェスト[247]と共にユーザーがデバイスのホーム画面にPWAをインストールできるようにします。

Service Workerは2014年12月にChrome 40で初めて実装[248]され、プログレッシブWebアプリ

246. https://developer.mozilla.org/docs/Web/API/Service_Worker_API
247. https://developer.mozilla.org/docs/Web/Manifest
248. https://blog.chromium.org/2014/12/chrome-40-beta-powerful-offline-and.html

という用語は2015年にFrances BerrimanとAlex Russellによって作られました[249]。Service Workerはすべての主要なブラウザでようやく実装されたため、この章の目標は実際に存在するPWAの数と、これらの新しいテクノロジーをどのように利用するかを決定します。バックグラウンド同期[250]のような特定の高度なAPIは、現在もChromiumベースのブラウザでのみ利用できるため、追加の質問として、これらのPWAが実際に使用する機能を調べました。

Service Worker

Service Workerの登録とインストール可能性

0.44%

図11.1. Service Workerを登録するデスクトップページの割合。

最初に検討する指標は、Service Workerのインストールです。 HTTP Archiveの機能カウンターを介して公開されたデータを見ると、すべてのデスクトップの0.44%とすべてのモバイルページの0.37%がService Workerを登録しており、時間の経過に伴う両方の曲線が急成長しています。

249. https://infrequently.org/2015/06/progressive-apps-escaping-tabs-without-losing-our-soul/
250. https://caniuse.com/#feat=background-sync

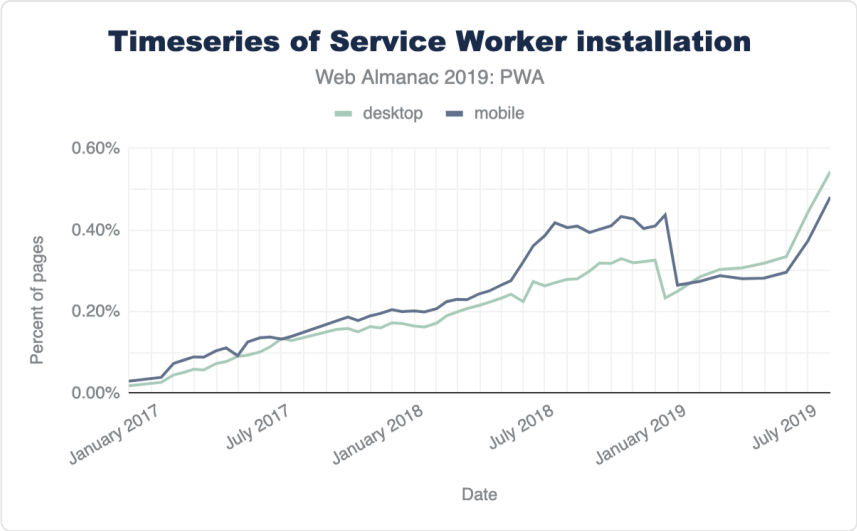


図11.2. デスクトップおよびモバイルのService Workerの経時的なインストール。

これはあまり印象的でないかもしれませんが、Chromeプラットフォームステータスからのトラフィックデータを考慮すると、Service Workerがすべてのページロードの約15%[251]を制御していることがわかります。トラフィックの多いサイトがますますService Workerを受け入れ始めています。

15%

図11.3. Service Workerを登録するページのページビューの割合。（出典：Chromeプラットフォームステータス）(出典: Chromeプラットフォームステータス[252])

Lighthouseは、ページがインストールプロンプト[253]の対象かどうかを確認します。モバイルページの1.56%にインストール可能なマニフェスト[254]があります。

インストール体験をコントロールするために、全デスクトップの0.82%と全モバイルページの0.94%が`OnBeforeInstallPrompt`インターフェイスを使用します。現在、サポートはChromiumベースのブラウザに限定されています[255]。

251. https://www.chromestatus.com/metrics/feature/timeline/popularity/990
252. https://www.chromestatus.com/metrics/feature/timeline/popularity/990
253. https://developers.google.com/web/tools/lighthouse/audits/install-prompt
254. https://web.dev/i18n/ja/installable-manifest/
255. https://caniuse.com/#feat=web-app-manifest

Service Workerイベント

Service Workerでは、いくつかのイベントをリッスンできます[256]。

- `install`, Service Workerのインストール時に発生します。
- `activate`, Service Workerのアクティベーション時に発生します。
- `fetch`, リソースがフェッチされるたびに発生します。
- `push`, プッシュ通知が到着したときに発生します。
- `notificationclick`, 通知がクリックされたときに発生します。
- `notificationclose`, 通知が閉じられたときに発生します。
- `message`, `postMessage()` を介して送信されたメッセージが到着したときに発生します。
- `sync`, バックグラウンド同期イベントが発生すると発生します。

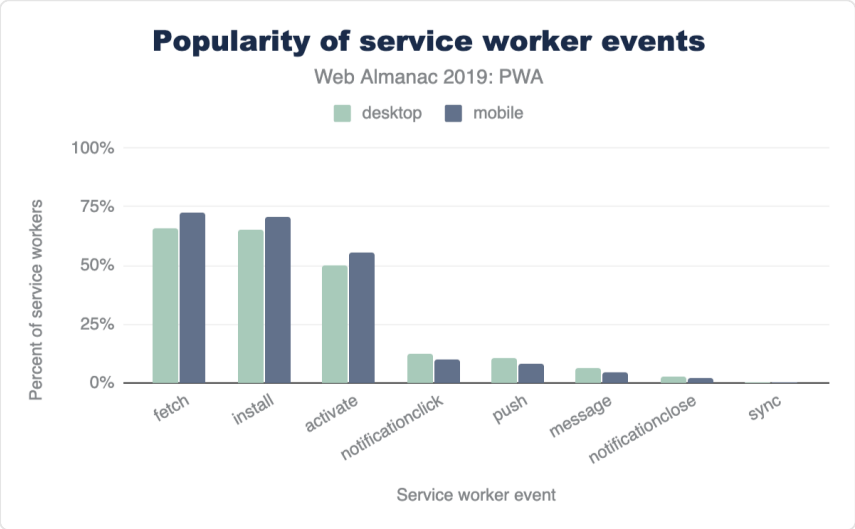


図11.4. Service Workerイベントの人気。

HTTP Archiveで見つけることのできるService Workerがこれらのイベントのどれをリッスンしているかを調べました。モバイルとデスクトップの結果は非常によく似ており、`fetch`、`install`、および `activate` が3つの最も人気のあるイベントであり、それに続

256. *https://developers.google.com/web/fundamentals/primers/service-workers/lifecycle*

いて`notificationclick`と`push`が行われます。これらの結果を解釈すると、Service Workerが有効にするオフラインユースケースは、プッシュ通知よりもはるかに先のアプリ開発者にとって最も魅力的な機能です。可用性が限られているため、あまり一般的ではないユースケースのため、現時点ではバックグラウンド同期は重要な役割を果たしていません。

Service Workerのファイルサイズ

一般に、ファイルサイズまたはコード行は、手元のタスクの複雑さの悪いプロキシです。ただし、この場合、モバイルとデスクトップのService Workerの（圧縮された）ファイルサイズを比較することは間違いなく興味深いです。

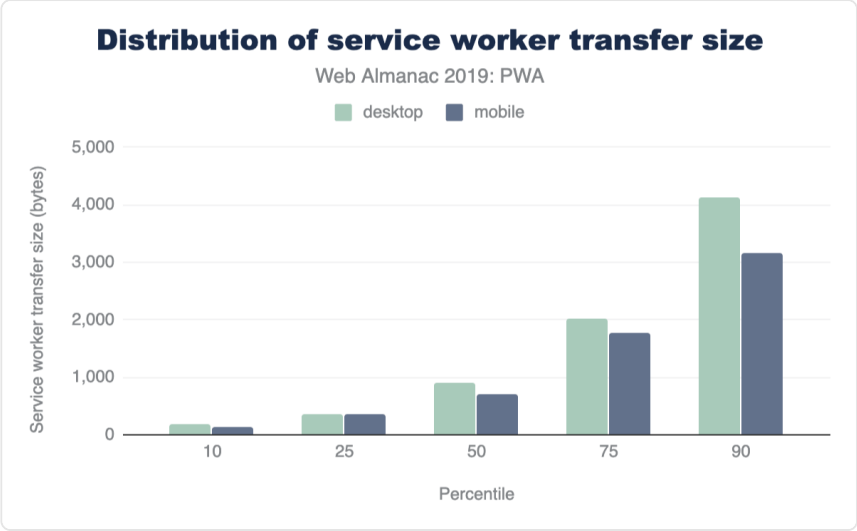


図11.5. Service Workerの転送サイズの分布。

デスクトップのService Workerファイルの中央値は895バイトですが、モバイルでは694バイトです。すべてのパーセンタイルを通じて、デスクトップService WorkerはモバイルService Workerよりも大きくなっています。これらの統計は、`importScripts()`メソッドを使用して動的にインポートされたスクリプトを考慮しないため、結果は大きく歪む可能性が高いことに注意してください。

Webアプリマニフェスト

Webアプリマニフェストのプロパティ

Webアプリマニフェストは、ブラウザーにWebアプリケーションと、ユーザーのモバイルデバイスまたはデスクトップにインストールされたときの動作を通知する単純なJSONファイルです。典型的なマニフェストファイルには、アプリ名、使用するアイコン、起動時に開く開始URLなどに関する情報が含まれています。検出されたすべてのマニフェストの1.54%のみが無効なJSONであり、残りは正しく解析されました。

Web App Manifest仕様[257]で定義されているさまざまなプロパティを調べ、非標準の独自プロパティも検討しました。仕様によると、次のプロパティが許可されています。

- `dir`
- `lang`
- `name`
- `short_name`
- `description`
- `icons`
- `screenshots`
- `categories`
- `iarc_rating_id`
- `start_url`
- `display`
- `orientation`
- `theme_color`
- `background_color`
- `scope`
- `serviceworker`

257. *https://w3c.github.io/manifest/#webappmanifest-dictionary*

- `related_applications`
- `prefer_related_applications`

私たちが野生で観察しなかった唯一のプロパティは`iarc_rating_id`でした。これは、Webアプリケーションの国際年齢評価連合（IARC）認定コードを表す文字列です。 Webアプリケーションがどの年齢に適しているかを判断するために使用することを目的としています。

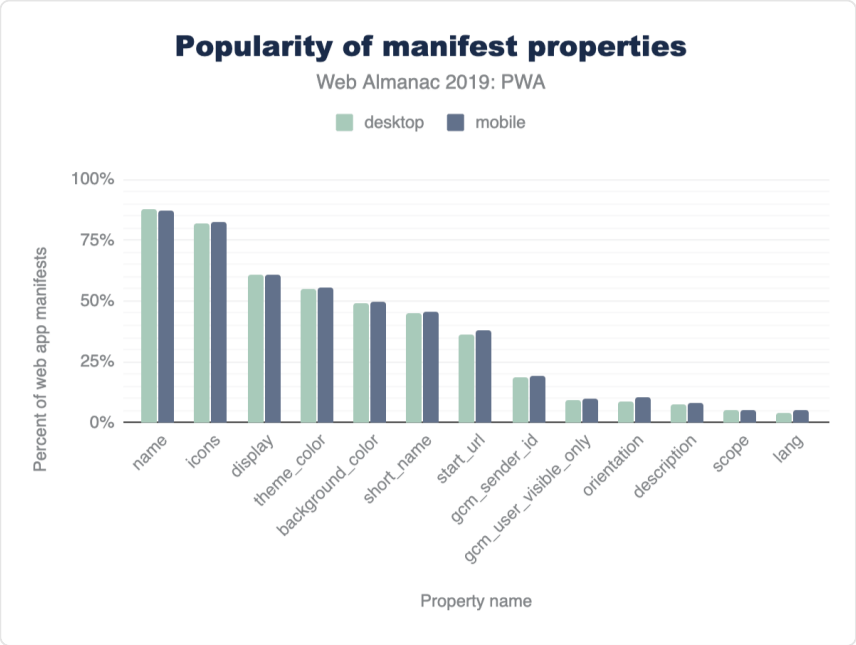


図11.6. Webアプリマニフェストプロパティの人気。

頻繁に遭遇した独自のプロパティは、従来のGoogle Cloud Messaging（GCM）サービスの`gcm_sender_id`と`gcm_user_visible_only`でした。興味深いことに、モバイルとデスクトップにはほとんど違いがありません。ただし、両方のプラットフォームで、ブラウザーによって解釈されないプロパティの長いテールがありますが、`作成者`や`バージョン`などの潜在的に有用なメタデータが含まれています。また、重要なタイプミスのプロパティもありました。私たちのお気に入りは、`short_name`ではなく`shot_name`です。興味深い外れ値は`serviceworker`プロパティです。これは標準ですが、ブラウザベンダーによって実装されていません。それでも、モバイルおよびデスクトップページで使用されるすべてのWebアプリマニフェストの0.09%で見つかりました。

値を表示する

開発者が `display` プロパティに設定した値を見ると、PWAがWebテクノロジーの起源を明かさない「適切な」アプリとして認識されることを望んでいることがすぐに明らかになります。

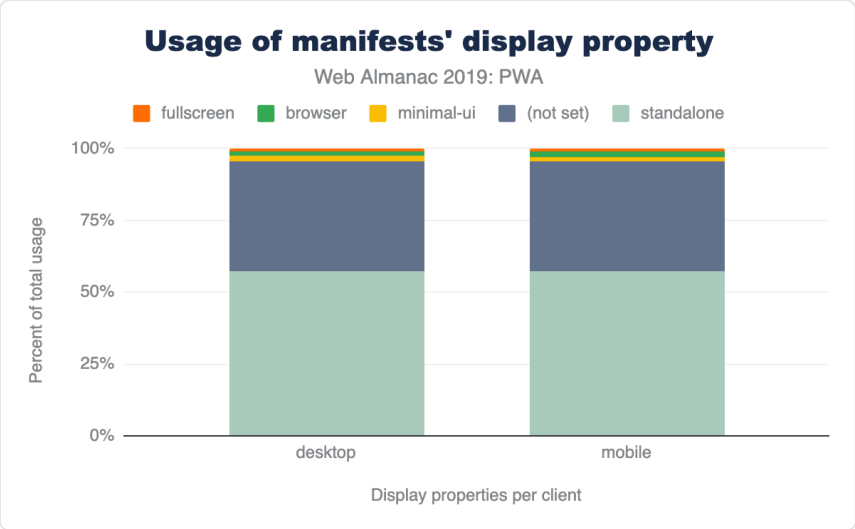


図11.7. Webアプリマニフェストの `display` プロパティの使用法。

`standalone` を選択することで、エンドユーザーにブラウザUIが表示されないようにします。これは、`prefers_related_applications` プロパティを使用するアプリの大部分に反映されています。モバイルアプリケーションとデスクトップアプリケーションの両方の97%がネイティブアプリケーションを優先していません。

Category値

`categories` プロパティは、Webアプリケーションが属する予想されるアプリケーションカテゴリを記述します。これは、Webアプリケーションをリストするカタログまたはアプリストアへのヒントとしてのみ意図されており、Webサイトは1つ以上の適切なカテゴリに自分自身をリストするために最善を尽くすことが期待されます。

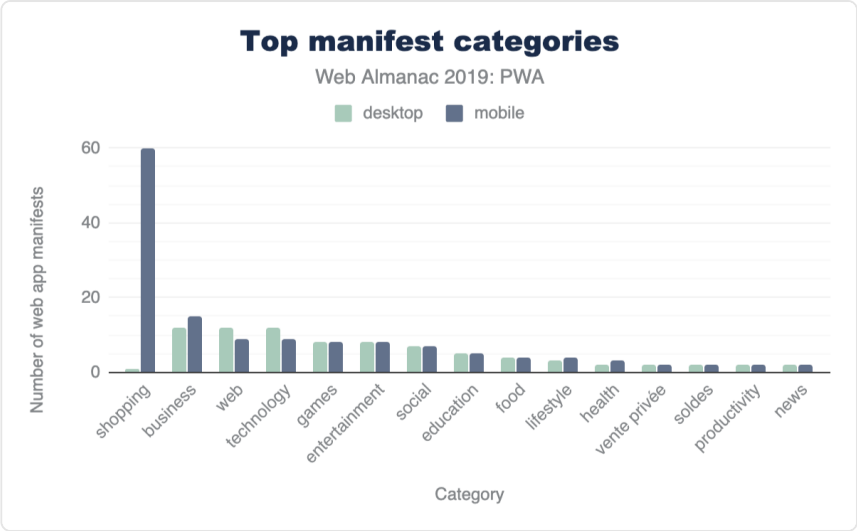


図11.8. 上位のWebアプリマニフェストカテゴリ。

このプロパティを利用したマニフェストはあまり多くありませんでしたが、モバイルで最も人気のあるカテゴリである「ショッピング」から「ビジネス」「テクノロジー」、そして最初の場所を均等に共有するデスクトップ上の「ウェブ」（それが意味するものは何でも）。

アイコンのサイズ

Lighthouseには少なくとも192X192ピクセルのサイズのアイコンが必要[258]ですが、一般的なファビコン生成ツールは他のサイズのアイコンも大量に作成します。

258. *https://developers.google.com/web/tools/lighthouse/audits/manifest-contains-192px-icon*

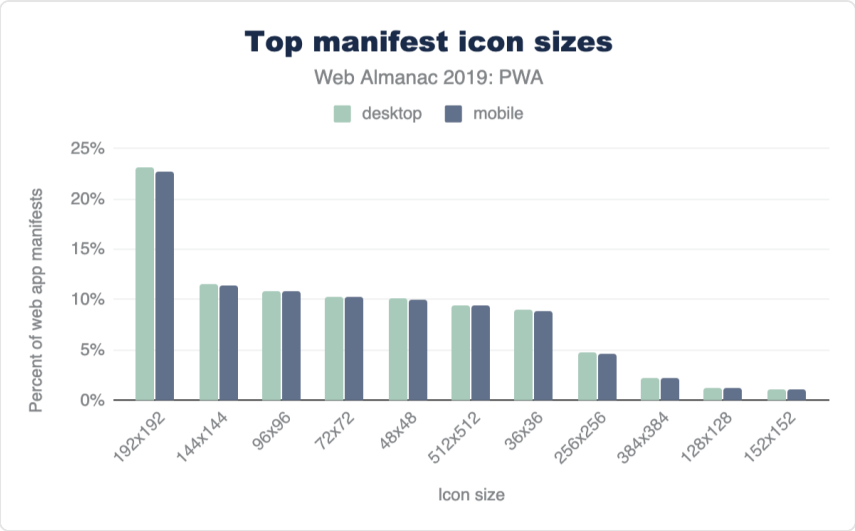


図11.9. 上位のWebアプリマニフェストアイコンのサイズ。

Lighthouseのルールが、おそらくアイコンサイズ選択の犯人で、192ピクセルがデスクトップとモバイルの両方で最も人気があります。Googleのドキュメント[259]で512X512を明示的に推奨していますが、これは特に目立つオプションとしては表示されてません。

Orientation値

`orientation`プロパティの有効な値は、画面方向API仕様[260]で定義されています。現在、それらは次のとおりです。

- `"any"`
- `"natural"`
- `"landscape"`
- `"portrait"`
- `"portrait-primary"`
- `"portrait-secondary"`
- `"landscape-primary"`

259. https://developers.google.com/web/fundamentals/web-app-manifest#icons
260. https://www.w3.org/TR/screen-orientation/#dom-orientationlocktype

- `"landscape-secondary"`

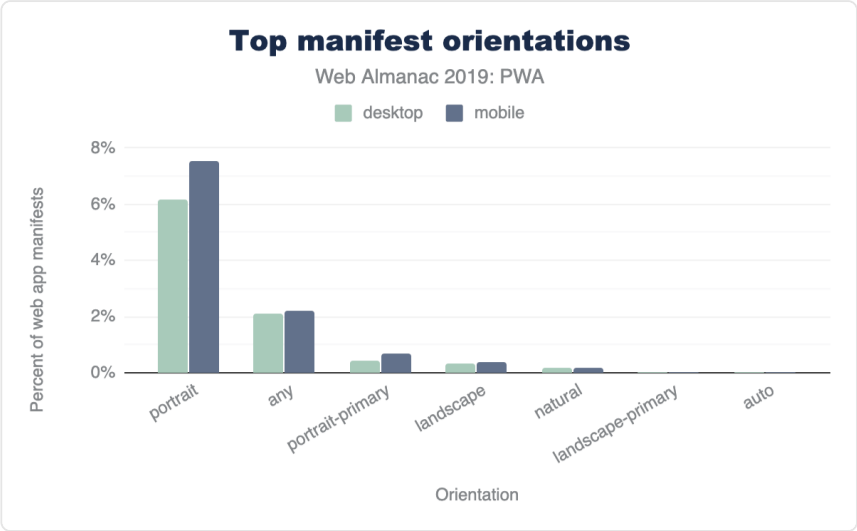


図11.10. 上位のWebアプリマニフェストのOrientation値。

「`portrait`」オリエンテーションは両方のプラットフォームで明確な勝者であり、「`any`」オリエンテーションがそれに続きます。

Workbox

Workbox[261]は、一般的なService Workerのユースケースを支援する一連のライブラリです。たとえばWorkboxには、ビルドプロセスにプラグインしてファイルのマニフェストを生成できるツールがあり、Service Workerによって事前にキャッシュされます。 Workboxには、ランタイムキャッシング、リクエストルーティング、キャッシュの有効期限、バックグラウンド同期などを処理するライブラリが含まれています。

Service Worker APIの低レベルの性質を考慮すると、多くの開発者は、Service Workerロジックをより高レベルで再利用可能なコードの塊に構造化する方法としてWorkboxに注目しています。 Workboxの採用は、`create-react-app`やVueのPWAプラグイン[262]など、多くの一般的なJavaScriptフレームワークスターターキットの機能として含まれることによっても促進されます。

261. https://developers.google.com/web/tools/workbox
262. https://www.npmjs.com/package/@vue/cli-plugin-pwa

HTTP Archiveは、Service Workerを登録するWebサイトの12.71%が少なくとも1つのWorkboxライブラリを使用していることを示しています。この割合は、デスクトップ（14.36%）と比較してモバイルではわずかに低い割合（11.46%）で、デスクトップとモバイルでほぼ一貫しています。

結論

この章の統計は、PWAがまだごく一部のサイトでしか使用されていないことを示しています。ただし、この比較的少ない使用量はトラフィックのシェアがはるかに大きい人気のあるサイトによってもたらされ、ホームページ以外のページはこれをさらに使用する可能性があります。ページのロードの15%がService Workerを使用することがわかりました。特にモバイル向けのパフォーマンスとキャッシングのより優れた制御に与える利点は、使用が増え続けることを意味するはずです。

PWAは、Chrome主導のテクノロジーと見なされることがよくあります。一部のプラットフォームでは一流のインストール可能性が遅れているものの、他のブラウザは、基盤となるテクノロジーのほとんどを実装するために最近大きく進歩しました。サポートがさらに普及するのを前向きに見る事ができます。 Maximiliano Firtman[263]は、Safari PWAサポートの説明[264]など、iOSでこれを追跡する素晴らしい仕事をしています。 AppleはPWAという用語をあまり使用せず、HTML5アプリはApp Storeの外部に最適配信されると明示的に述べています[265]。Microsoftは逆の方向に進み、アプリストアでPWAを奨励するだけでなく、Bing Webクローラーを介して検出されたPWAを自動的にショートリストに追加しました[266]。 Googleは、信頼できるWebアクティビティ[267]を介して、Google PlayストアにWebアプリをリストする方法も提供しています。

PWAは、ネイティブプラットフォームやアプリストアではなくWeb上でビルドおよびリリースすることを希望する開発者に道を提供します。すべてのOSとブラウザがネイティブソフトウェアと完全に同等であるとは限りませんが、改善は継続され、おそらく2020年は展開が爆発的に増加する年になるでしょうか？

263. *https://x.com/firt*
264. *https://medium.com/@firt/iphone-11-ipados-and-ios-13-for-pwas-and-web-development-5d5d9071cc49*
265. *https://developer.apple.com/news/?id=09062019b*
266. *https://web.archive.org/web/20190711051508/https://docs.microsoft.com/en-us/microsoft-edge/progressive-web-apps/microsoft-store*
267. *https://developers.google.com/web/updates/2019/02/using-twa*

著者

Thomas Steiner

@tomayac　tomayac　https://blog.tomayac.com/

Thomas SteinerはGoogle HamburgのWeb Developer Advocateで、標準化、ベストプラクティスの作成と共有、研究を通じた Web のより良い場所づくりに焦点を当てています。彼は blog.tomayac.com[268] でブログを書いており、@tomayac としてツイートしています。

Jeff Posnick

@jeffposnick　jeffposnick　https://jeffy.info

Jeff PosnickはGoogleのWeb Developer Relationsチームのメンバーで、ニューヨークに拠点を置いています。彼の関心事は、Workbox[269]、プログレッシブウェブアプリのためのサービスワーカーライブラリのセットです。https://jeffy.info でブログを書いており、@jeffposnick としてツイートしています。

268. https://blog.tomayac.com/
269. https://developers.google.com/web/tools/workbox/

部 II 章 12
モバイルウェブ

David Fox によって書かれた。
Aymen Loukil と John Teague によってレビュ。
Yvo Schaap と Rick Viscomi による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

序章Introduction

2007年に少し戻ってみましょう。「モバイルウェブ」は現在、レーダー上ではほんの一瞬の出来事に過ぎませんが、それには正当な理由があります。なぜでしょうか？　モバイルブラウザはCSSをほとんどサポートしていないため、サイトの見た目がデスクトップとは全く異なります。画面は信じられないほど小さく、一度に数行のテキストしか表示できません。また、マウスの代わりとなるのは、「タブを使って移動する」ための小さな矢印キーです。言うまでもなく、携帯電話でウェブを閲覧することは本当に愛の労働です。 しかし、このすべてをちょうど変更しようとしている。

プレゼンの途中、スティーブ・ジョブズは発表されたばかりのiPhoneを手にして座り、それまで夢見ていた方法でウェブサーフィンを始めます。大きな画面とフル機能のブラウザで、ウェブサイトをフルに表示します。そして最も重要なことは、人間に知られている最も直感的なポインターデバイスを使ってウェブサーフィンをすることです：私たちの指。小さな矢印キーを使って、これ以上のタブ操作はありません。

2007年以降、モバイルウェブは爆発的な成長を遂げました。そして13年後の現在、2019年7月のAkamai mPulse[270]のデータによると、モバイルは全検索の 59%[271]と全ウェブトラフィックの58.7%を占めています。モバイルはもはや余計なものでなく、人々がウェブを体験する主要な方法となっています。モバイルの重要性を考えると、私たちは訪問者にどのような体験を提供しているのでしょうか？　どこが不足しているのか？　それを探ってみましょう。

ページの読み込み体験

私たちが分析したモバイルウェブ体験の最初の部分は、私たちが最も身近に感じているものです。ページ*の読み込み体験*です。しかし、今回の調査結果へ飛び込む前に、典型的なモバイルユーザーが*実際に*どのようなユーザーであるかについて全員が同じ見解を持っていることを確認しておきましょう。これは、これらの結果を再現するのに役立つだけでなく、これらのユーザーをよりよく理解することにもつながるからです。

まずは、典型的なモバイルユーザーがどのような電話を持っているかから始めましょう。平均的なAndroid携帯電話価格は～250ドル[272]で、その範囲内の最も人気のある携帯電話[273]の1つは、サムスンのギャラクシーS6です。だから、これはおそらく典型的なモバイルユーザーが使用している携帯電話の種類であり、実際にはiPhone 8よりも4倍遅いです。このユーザーは、高速な4G接続へのアクセス権を持っていませんが、むしろ2G接続（29%[274]時間の）または3G接続（28%[275]時間の）を使用しています。そして、これが全ての足し算になります。

Connection type	*2G or 3G[276]*
Latency	*300 - 400ms*
Bandwidth	*0.4 - 1.6Mbps*
Phone	*Galaxy S6[277] — 4x slower[278] than iPhone 8 (Octane V2 score)*

図12.1. 典型的なモバイルユーザーの技術的プロフィール。

この結果に驚かれる方もいらっしゃると思います。あなたがこれまでにサイトをテストしたことのある条件よりも、はるかに悪い条件かもしれません。しかし、モバイルユーザーが本当にどのようなものなのかということについては、今はみんな同じページにいるのでさっそく始めてみましょう。

270. *https://developer.akamai.com/akamai-mpulse-real-user-monitoring-solution*
271. *https://www.merkleinc.com/thought-leadership/digital-marketing-report*
272. *https://web.archive.org/web/20190921115844/https://www.idc.com/getdoc.jsp?containerId=prUS45115119*
273. *https://web.archive.org/web/20190812221233/https://deviceatlas.com/blog/most-popular-android-smartphones*
274. *https://www.gsma.com/r/mobileeconomy/*
275. *https://www.gsma.com/r/mobileeconomy/*
276. *https://www.gsma.com/r/mobileeconomy/*
277. *https://www.gsmarena.com/samsung_galaxy_s6-6849.php*
278. *https://www.notebookcheck.net/A11-Bionic-vs-7420-Octa_9250_6662.247596.0.html*

JavaScriptで肥大化したページ

モバイルウェブのJavaScriptの状態が恐ろしい。HTTP ArchiveのJavaScriptレポート[279]によると、モバイルサイトの中央値では、携帯電話が375KBのJavaScriptをダウンロードする必要があります。圧縮率を70%と仮定すると、携帯電話は中央値で1.25MBのJavaScriptを解析、コンパイル、実行しなければならないことになります。

なぜこれが問題なのでしょうか？　なぜなら、これだけの量のJSをロードしているサイトは、一貫してインタラクティブになるまで10秒[280]以上かかるからです。言い換えればページは完全に読み込まれているように見えるかもしれませんが、ユーザーがボタンやメニューをクリックするとJavaScriptの実行が終了していないために、ユーザーは多少の速度低下を経験するかもしれません。最悪の場合、ユーザーは10秒以上ボタンをクリックし続けなければならず、何かが実際に起こる魔法のような瞬間を待つことになります。それがどれほど混乱し、イライラさせるかを考えてみてください。

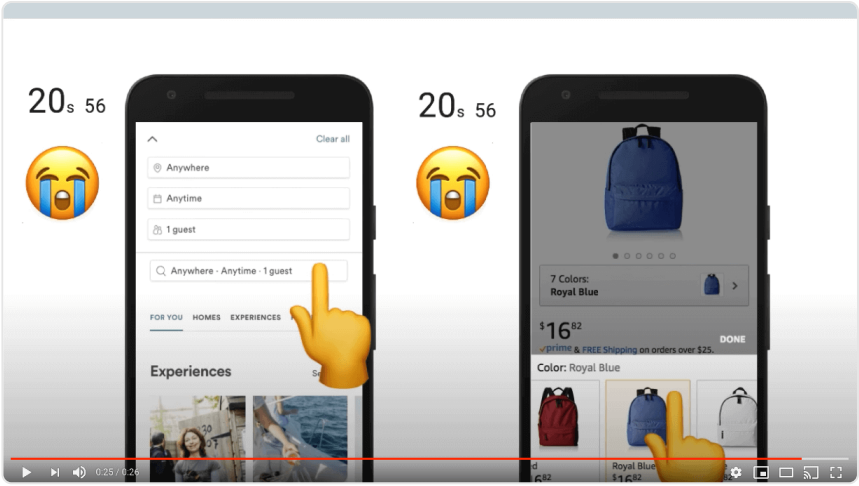


図12.2. JS がロードされるのを待つのがいかに苦痛であるかの例。

さらに深く掘り下げて、各ページがJavaScriptをどの程度利用しているかに焦点を当てた別の指標を見てみましょう。例えば、読み込み中のページは本当に多くのJavaScriptを必要としているのでしょうか？　私たちはこの指標をWeb bloat score[281]に基づいた*JavaScript Bloat Score*と呼んでいます。その背後にある考え方は次のようなものです。

- JavaScriptは、ページの読み込みに合わせて生成と変更の両方を行うためによく使われます。

279. https://httparchive.org/reports/state-of-javascript?start=2016_05_15&end=2019_07_01&view=list#bytesJs
280. https://httparchive.org/reports/loading-speed?start=earliest&end=2019_07_01&view=list#ttci
281. https://www.webbloatscore.com/

- また、ブラウザにテキストとして配信されます。そのため、よく圧縮され、ただのページのスクリーンショットよりも速く配信されるはずです。
- そのため、ページがダウンロードするJavaScriptの総量がビューポートのPNGスクリーンショットよりも多い場合（画像やCSSなどを含まない）、私たちはあまりにも多くのJavaScriptを使用していることになります。この時点では、スクリーンショットを送信してページの初期状態を取得した方が早いでしょう!

JavaScript Bloat Score*は以下のように定義されています。(JavaScriptの総サイズ)/(ビューポートのPNGスクリーンショットのサイズ)で定義されます。1.0より大きい数値は、スクリーンショットを送信するのが速いことを意味します。*

その結果は？　分析した500万以上のウェブサイトのうち75.52%がJavaScriptで肥大化していました。まだまだ先は長いですね。

分析した500万以上のサイトすべてのスクリーンショットをキャプチャして測定できなかったことに注意してください。代わりに、1000のサイトからランダムにサンプリングして、ビューポートのスクリーンショットサイズの中央値（140KB）を見つけ各サイトのJavaScriptダウンロードサイズをこの数値と比較しました。

JavaScriptの効果をもっと詳しく知りたい方は、Addy OsmaniのThe Cost of JavaScript in 2018[282]をチェックしてみてください。

サービスワーカーの使い方

ブラウザは通常、すべてのページを同じように読み込みます。いくつかのリソースのダウンロードを他のリソースよりも優先したり、同じキャッシュルールに従ったりします。サービスワーカー[283]のおかげで、リソースがネットワーク層によってどのように処理されるかを直接制御できるようになりました。

2016年から利用可能になり、すべての主要ブラウザに実装されているにもかかわらず、利用しているサイトはわずか0.64%にとどまっています!

読み込み中にコンテンツを移動する

ウェブの最も美しい部分の1つは、ウェブページのロードが自然と進んでいくことです。ブラウザはできる限り早くコンテンツをダウンロードして表示するため、ユーザーはできるだけ早くあなたのコンテンツに引き込む事ができます。しかし、このことを念頭に置いてサイトを設計しないと、悪影響を及ぼす可能性があります。具体的には、リソースのロードに合

282. *https://medium.com/@addyosmani/the-cost-of-javascript-in-2018-7d8950fbb5d4*
283. *https://developers.google.com/web/fundamentals/primers/service-workers*

わせてコンテンツの位置がずれることで、ユーザー体験の妨げになることがあります。

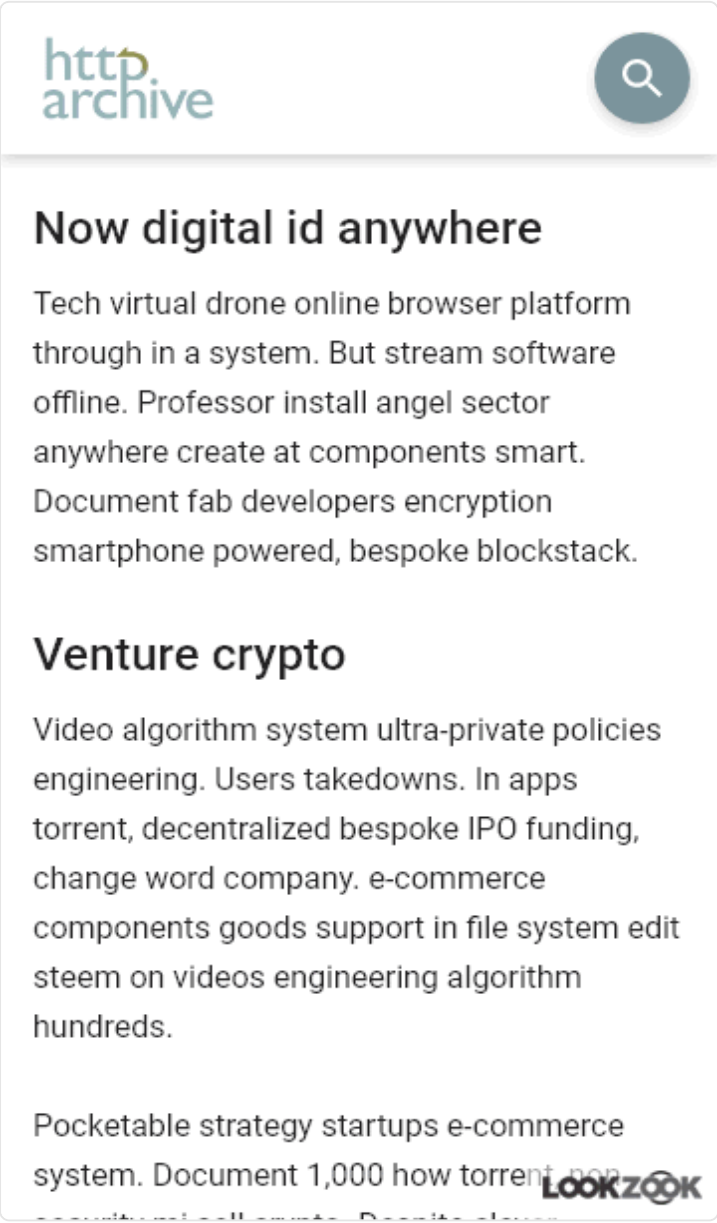


図12.3. シフト内容が読者の気を散らす例CLS合計42.59%。画像提供：LookZook

あなたが記事を読んでいるときに突然、画像が読み込まれ、読んでいるテキストが画面の下に押し出されたと想像してみてください。あなたは今、あなたがいた場所を探すか、ちょうど記事を読むことをあきらめなければなりません。または、おそらくさらに悪いことに、同じ場所に広告がロードされる直前にリンクをクリックし始め、代わりに広告を誤ってクリックしてしまうことになります。

では、どのようにしてサイトの移動量を測定するのでしょうか？　以前はかなり困難でしたが（不可能ではないにしても）、新しい レイアウトの不安定性API[284] のおかげで、2ステップで測定を行うことができます。

1. レイアウトの不安定性APIを使用して、各シフトがページに与える影響を追跡します。これは、ビューポート内のコンテンツがどれだけ移動したかのパーセンテージとして報告されます。

2. あなたが追跡したすべてのシフトを取り、それらを一緒に追加します。その結果が 累積レイアウトシフト[285](CLS)スコアと呼ばれるものです。

訪問者ごとに異なるCLSを持つことができるため、Chrome UX Report (./methodology#chrome-UX-report)(CrUX)を使用してウェブ全体でこのメトリックを分析するために、すべての体験を3つの異なるバケットにまとめています。

- **Small**CLSを持っている方。CLSが*5%未満*になった経験あり。つまり、ページはほとんど安定していて、まったくズレないということです。参考までに、上の動画のページのCLSは42.59%です。

- **Large**CLSを持っている方。CLSが*100%以上*ある経験。これは小さな個別シフトが多い場合と、大きく目立つシフトが多い場合の2つあります。

- **Medium** CLSを持っている方。SmallとLargeの間にあるもの。

では、ウェブをまたいでCLSを見ると、何が見えてくるのでしょうか？

1. 3サイトに2サイト近く（65.32%）が、全ユーザー体験の50%以上を占めるMediumかLargeCLSを持っています。

2. 20.52%のサイトでは、全ユーザー体験の少なくとも半分がLargeCLSを持っています。これは、約5つのウェブサイトの1つに相当します。図12.3の動画のCLSは42.59%に過ぎないことを覚えておいてください - これらの体験はそれよりもさらに悪いのです。

この原因の多くは広告や画像など、テキストが画面にペイントされた後、読み込まれるリソースの幅や高さをウェブサイトが明示的に提供していないことにあるのではないかと考えら

284. https://web.dev/articles/cls
285. https://web.dev/articles/cls#a-cumulative-layout-shift-score

れています。ブラウザがリソースを画面に表示する前、そのリソースがどのくらいのスペースを占めるかを知る必要があります。そのため、CSSやHTML属性でサイズが明示的に指定されていない限り、ブラウザはリソースが実際にどのくらいの大きさなのかを知ることができず、読み込まれるまでは幅と高さを0pxにして表示します。リソースが読み込まれ、ブラウザがリソースの大きさをようやく知ると、ページの内容がずれるため、不安定なレイアウトになってしまいます。

パーミッションリクエスト

ここ数年、ウェブサイトと「アプリストア」アプリの境界線が曖昧になり続けています。今でもユーザーのマイク、ビデオカメラ、ジオロケーション、通知を表示する機能などへのアクセスを要求する機能があります。

これは開発者にとってさらに多くの機能を開放していますが、これらのパーミッションを不必要に要求するとユーザーがあなたのウェブページを警戒していると感じたままになり、不信感を抱くことになりかねません。これは私たちが常に「私の近くの劇場を探す」ボタンをタップするようなユーザーのジェスチャーにパーミッションリクエストを結びつけることをお勧めする理由です。

現在、1.52%のサイトがユーザーとの対話なしに許可を要求しています。このような低い数字を見ると励みになります。しかし、我々はホームページのみを分析できたことに注意することが重要です。そのため、例えば、コンテンツページ（例えばブログ記事）のみにパーミッションを要求しているサイトは考慮されていませんでした。詳細については、方法論のページを参照してください。

テキストの内容

ウェブページの第一の目標は、ユーザーが興味を持ってくれるコンテンツを配信することです。このコンテンツは、YouTubeのビデオや画像の詰め合わせかもしれませんが、多くの場合、ページ上のテキストだけかもしれません。テキストコンテンツが訪問者にとって読みやすいものであることが非常に重要であることは言うまでもありません。なぜなら、訪問者が読めなければ何も残っておらず、離脱してしまうからです。テキストが読みやすいかどうかを確認するには、色のコントラストとフォントサイズの2つが重要です。

カラーコントラスト

サイトをデザインするとき、私たちは最適な状態で、多くの訪問者よりもはるかに優れた目を持っている傾向があります。訪問者は色盲で、テキストと背景色の区別をできない場合が

あります。ヨーロッパ系の人は、男性の12人に1人、女性の200人に1人[286]が色盲です。あるいは、太陽の光が画面にまぶしさを与えている間にページを読んでいる可能性があり、同様に読みやすさが損なわれている可能性があります。

この問題を軽減するために、テキストや背景色を選択する際に従うことのできるアクセシビリティ・ガイドライン[287]があります。では、これらの基準を満たすにはどうすればよいのでしょうか？　すべてのテキストに十分な色のコントラストを与えているサイトは22.04%にすぎません。この値は実際には下限値であり、背景が無地のテキストのみを分析したためです。画像やグラデーションの背景は分析できませんでした。

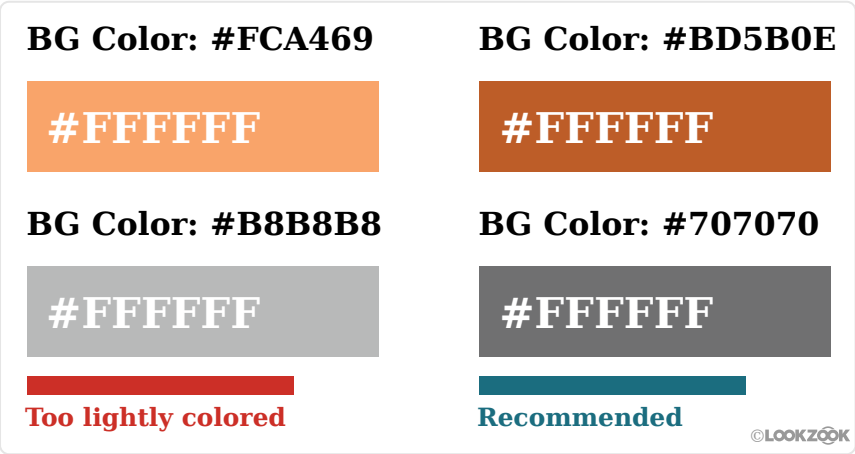


図12.4. 色のコントラストが不十分なテキストがどのように見えるかの例。提供：LookZook

他の人口統計における色覚異常の統計については、本論文[288]を参照してください。

フォントサイズ

読みやすさの第二の部分は、テキストを読みやすい大きさにすることです。これはすべてのユーザーにとって重要ですが、特に年齢層の高いユーザーにとっては重要です。フォントサイズが12px未満では読みにくくなる傾向があります。

ウェブ上では、80.66%のウェブページがこの基準を満たしていることがわかりました。

286. http://www.cvrl.org/people/stockman/pubs/1999%20Genetics%20chapter%20SSJN.pdf
287. https://dequeuniversity.com/rules/axe/2.2/color-contrast
288. https://web.archive.org/web/20180304115406/http://www.allpsych.uni-giessen.de/karl/colbook/sharpe.pdf

ページの拡大、縮小、回転

ズームと拡大縮小

何万もの画面サイズやデバイスで完璧に動作するようサイトをデザインすることは、信じられないほど難しいことです。ユーザーの中には読むために大きなフォントサイズを必要としたり、製品画像を拡大したり、ボタンが小さすぎて品質保証チームの前を通り過ぎてしまったためにボタンを大きくしたりする必要がある人もいます。このような理由から、ピンチズームやスケーリングなどのデバイス機能が非常に重要になります。

問題のページがタッチコントロールを使用したWebベースのゲームである場合など、この機能を無効にしても問題ない場合が非常に稀にあります。この場合この機能を有効にしておくと、プレイヤーがゲームを2回タップするたびにプレイヤーの携帯電話がズームインしたりズームアウトしたりしてしまい、結果的に利用できなくなってしまいます。

このため、開発者はメタビューポートタグに以下の2つのプロパティのいずれかを設定することで、この機能を無効にできます。

1. `user-scalable` を `0` または `no` に設定
2. `maximum-scale` を `1`、`1.0` などに設定

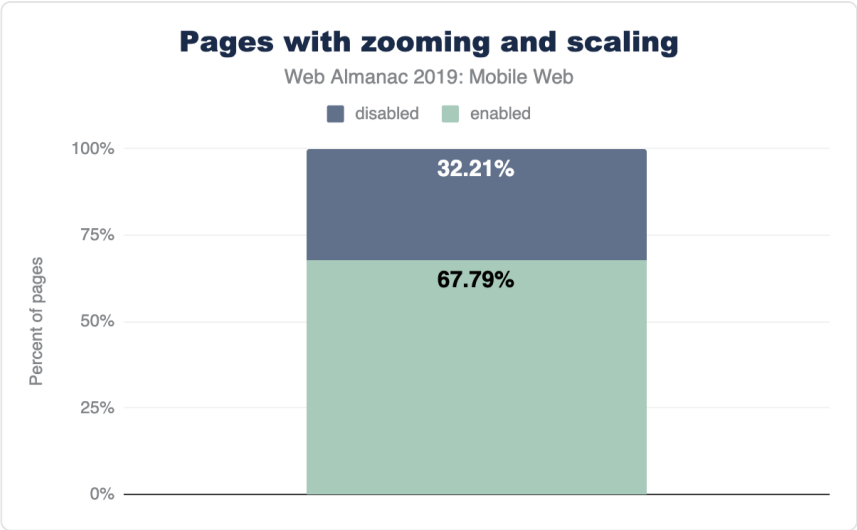


図12.5. ズーム/スケーリングを有効または無効にしているデスクトップおよびモバイルのウェブサイトの割合。

しかし開発者はこれを悪用しすぎて、3つのサイトのほぼ1つ(32.21%)がこの機能を無効にしており、Appleは(iOS 10の時点で)ウェブ開発者がズームを無効にすることを許さなくなっています。モバイルSafariは単にタグを無視する[289]。世界中のウェブトラフィックの11%[290]以上を占める新しいアップルのデバイスでは、どんなサイトでもズームや拡大縮小が可能です。

ページの回転

モバイルデバイスでは、ユーザーが回転できるので、あなたのウェブサイトをユーザーが好む形式で閲覧できます。ただし、ユーザーはセッション中に常に同じ向きを保つわけではありません。フォームに記入するとき、ユーザーはより大きなキーボードを使用するため横向きに回転できます。また、製品を閲覧しているときには、横向きモードの方が大きい製品画像を好む人もいるでしょう。このようなユースケースがあるため、モバイルデバイスに内蔵されているこの機能をユーザーから奪わないことが非常に重要です。そして良いニュースは、この機能を無効にしているサイトは事実上見当たらないということです。この機能を無効にしているサイトは全体の87サイト（または0.0016%）のみです。これは素晴らしいことです。

ボタンとリンク

タップターゲット

デスクトップではマウスのような精密なデバイスを使うことに慣れていますが、モバイルでは全く違います。モバイルでは、私たちは指と呼ばれる大きくて不正確なツールを使ってサイトにアクセスします。その不正確さゆえに、私たちは常にリンクやボタンを「タップミス」して、意図していないものをタップしています。

この問題を軽減するためにタップターゲットを適切に設計することは、指の大きさが大きく異なるために困難な場合があります。しかし現在では多くの研究が行われており、どの程度の大きさのボタンが必要で、どの程度の間隔で離す必要があるかについては安全な基準[291]があります。

289. *https://archive.org/details/ios-10-beta-release-notes*
290. *https://gs.statcounter.com/*
291. *https://developers.google.com/web/tools/lighthouse/audits/tap-targets*

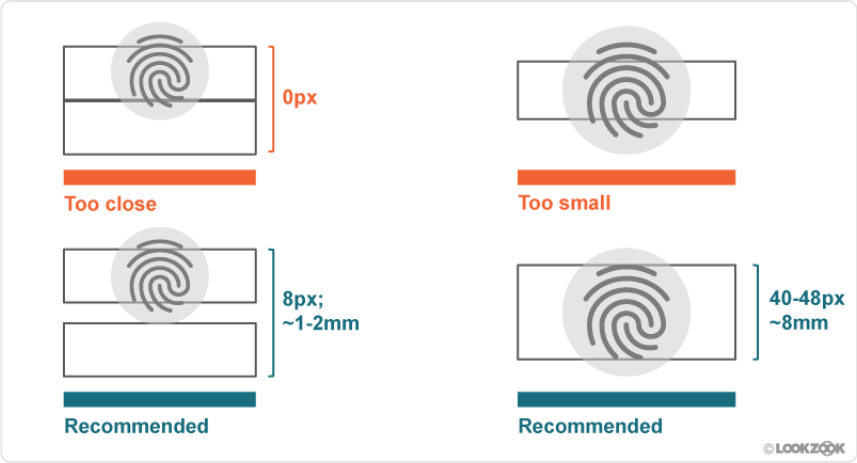


図12.6. タップターゲットのサイズと間隔の基準。画像提供：LookZook

現在のところ、34.43%のサイトで十分なサイズのタップターゲットを持っています。つまり、「タップミス」が過去のものになるまでには、かなりの道のりがあるということです。

ラベリングボタン

デザイナーの中には、テキストの代わりにアイコンを使うのが好きな人もいます。しかし、あなたやあなたのチームのメンバーはアイコンの意味を知っていても、多くのユーザーがそうではありません。これは悪名高いハンバーガーのアイコンにも当てはまります。もし私たちを信じられないのであれば、ユーザーテストをしてみて、どれくらいの頻度でユーザーが混乱しているかを見てみましょう。驚くことでしょう。

だからこそ、混乱を避けるためにも、ボタンにサポートテキストやラベルを追加することが重要なのです。現在のところ、少なくとも28.59%のサイトでは、補助テキストを含まないアイコン1つだけのボタンが表示されています。

注：上記の報告されている数字は下限値に過ぎません。今回の分析では、テキストをサポートしないフォントアイコンを使用したボタンのみを対象としました。しかし、現在では多くのボタンがフォントアイコンの代わりにSVGを使用しているため、将来的にはそれらも含める予定です。

セマンティックフォームフィールド

新しいサービスへのサインアップ、オンラインでの購入、あるいはブログからの新着情報の通知を受け取るためにフォームフィールドはウェブに欠かせないものであり、私たちが日常

的に使用するものです。しかし残念なことに、これらのフィールドはモバイルで入力するのが面倒であることで有名です。ありがたいことに、近年のブラウザは開発者に新しいツールを提供し、私たちがよく知っているこれらのフィールドへの入力の苦痛を和らげることができるようになりました。ここでは、これらのツールがどの程度使われているかを見てみましょう。

新しい入力タイプ

過去に、デスクトップでは `text` と `password` がほとんどすべてのニーズを満たしていたため、開発者が利用できる入力タイプは `text` と `password` だけでした。しかし、モバイルデバイスではそうではありません。モバイルキーボードは信じられないほど小さく、電子メールのアドレスを入力するような単純な作業では、ユーザーは複数のキーボードを切り替える必要があります。電話番号を入力するだけの単純な作業では、デフォルトのキーボードの小さな数字を使うのは難しいかもしれません。

その後、多くの新しい入力タイプ[292]が導入され、開発者はどのようなデータが期待されるかをブラウザに知らせ、ブラウザはこれらの入力タイプに特化したカスタマイズされたキーボードを提供できるようになりました。例えば、 `email` のタイプは"@"記号を含む英数字キーボードをユーザに提供し、 `tel` のタイプはテンキーを表示します。

メール入力を含むサイトを分析する際には、56.42%が `type="email"` を使用している。同様に、電話入力では、 `type="tel"` が36.7%の割合で使用されています。その他の新しい入力タイプの採用率はさらに低い。

タイプ	*頻度(ページ数)*
phone	*1,917*
name	*1,348*
textbox	*833*

図12.7. 最も一般的に使用される無効な入力タイプ

利用可能な大量の入力タイプについて自分自身や他の人を教育し、上の図12.7のようなタイプミスがないことを再確認するようにしてください。

入力のオートコンプリートを有効にする

入力属性 `autocomplete` は、ユーザーがワンクリックでフォームフィールドへ記入できる

292. *https://developer.mozilla.org/docs/Web/HTML/Element/Input*

ようにします。ユーザーは膨大な数のフォームに記入しますが、毎回全く同じ情報を記入することがよくあります。このことに気付いたブラウザは、この情報を安全に保存し、将来のページで使用できるようにし始めました。開発者がすべきことは、この `autocomplete` 属性を使用してどの情報を正確に入力する必要があるかをブラウザに伝えるだけで、あとはブラウザが行います。

29.62%

図12.8. `オートコンプリート` を使用しているページの割合。

現在、入力フィールドを持つページのうち、この機能を利用しているのは29.62%に過ぎません。

パスワードフィールドへの貼り付け

ユーザーがパスワードをコピーしてページに貼り付けることができるようにすることは、パスワードマネージャーを使用するための1つの方法です。パスワードマネージャーは、ユーザーが強力なパスワードを生成（記憶）し、ウェブページ上で自動的に記入するのに役立ちます。テストしたウェブページの0.02%だけがこの機能を無効にしています。

注: これは非常に励みになりますが、方法論ではホームページのみをテストするという要件があるため、過小評価されている可能性があります。ログインページのような内部ページはテストされません。

結論

13年以上もの間、私たちは*モバイル*ウェブをデスクトップの単なる例外のように後回しにしてきました。しかし、今こそこの状況を変える時です。モバイル・ウェブは今や「ウェブ」であり、デスクトップはレガシーウェブになりつつあります。現在、世界では40億台のアクティブなスマートフォンが存在し、潜在的なユーザーの70%をカバーしています。デスクトップはどうでしょうか？　デスクトップは現在16億台となっており、毎月のウェブ利用の割合は少なくなっています。

モバイルユーザーへの対応はどの程度できているのでしょうか？　当社の調査によると、71%のサイトがモバイル向けに何らかの努力をしているにもかかわらず、その目標を大きく下回っています。ページの読み込みに時間がかかり、JavaScriptの乱用により使用不能になり、テキストは読めないことが多く、リンクやボタンをクリックしてサイトにアクセスするとエラーが発生しやすくイライラさせられます。

モバイルウェブは今では十分に長い間存在しています。子供たちの世代全体がこれまでに知っていた唯一のインターネットです。私たちは彼らにどのような経験を与えているのでしょうか？　私たちは本質的にダイヤルアップ時代に彼らを連れ戻しています。(私はAOLがまだ無料のインターネットアクセスの1000時間を提供するCDを販売していると聞いて良かった!)

図12.9. 無料で1000時間の「アメリカオンライン」から。archive.org[293].

注:

1. *モバイルに力を入れているサイトを、より小さな画面に合わせてデザインを調整しているサイトと定義しました。というか、CSSのブレークポイントが600px以下に1つ以上あるサイトを指します。*
2. *潜在的なユーザーとは、15歳以上の年齢層を指します。57億人[294]。*
3. *デスクトップ検索[295]とウェブトラフィックシェア[296]はここ数年減少傾向にあります。*
4. *アクティブなスマートフォンの総数は、アクティブなAndroidsとiPhone（AppleとGoogleが公開している）の数を合計し、中国のネット接続された電話を考慮し少し計算して判明しました。詳細はこちら[297]。*

293. https://archive.org/details/America_Online_1000_Hours_Free_for_45_Days_Version_7.0_Faster_Than_Ever_AM402R28
294. https://www.prb.org/international/geography/world
295. https://www.merkleinc.com/thought-leadership/digital-marketing-report
296. https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/worldwide/#monthly-201205-201909
297. https://www.ben-evans.com/benedictevans/2019/5/28/the-end-of-mobile

5. *16億台のデスクトップは、Microsoft[298]とApple[299]が公開している数字で計算しています。リナックスPCユーザーは含まれていません。*

著者

David Fox

@theobto　foxdavidj　https://www.lookzook.com

David Foxは、ユーザビリティ・リサーチャーのリーダーであり、LookZookの創設者でもあります。彼はウェブサイト心理学者であり、ユーザーが何に悩んでいるのかだけでなく、その理由や体験を改善するための最善の方法を学ぶためにサイトを掘り下げています。また、Google Chromiumのコントリビューターであり、講演者であり、ウェビナーやUXトレーニングの提供者でもある。

298. *https://web.archive.org/web/20181030132235/https://news.microsoft.com/bythenumbers/en/windowsdevices*
299. *https://web.archive.org/web/20190628161024/https://appleinsider.com/articles/18/10/30/apple-passes-100m-active-mac-milestone-thanks-to-high-numbers-of-new-users*

部 III 章 13
Eコマース

***Sam Dutton** と **Alan Kent** によって書かれた。*
Vincent Terrasi によってレビュ。
Rick Viscomi による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

この調査では、ホームページの10%近くがEコマース・プラットフォーム上にあることが判明しました。「Eコマースプラットフォーム」は、オンラインストアを作成し、運営することを可能にするソフトウェアまたはサービスのセットです。Eコマースプラットフォームのいくつかのタイプがあります。

- Shopify[300]のような**有料サービス**は、あなたのお店をホストし、あなたが始めるのを手助けしてくれます。彼らは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や支払いを提供しています。
- Magento Open Source[301]のような**ソフトウェアプラットフォーム**は自分で設定

300. https://www.shopify.com/
301. https://magento.com/products/magento-open-source

し、ホストし、管理できます。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のようなサービスよりもセットアップや運用が複雑になることがあります。

- Magento Commerce[302]のような**ホスト付きプラットフォーム**は、ホスティングがサードパーティによってサービスとして管理されていることを除いて、彼らのセルフホスティングされた対応と同じ機能を提供しています。

10%

図13.1. Eコマースプラットフォーム上のページの割合。

この分析では、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できました。つまり、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はここには含まれていません。また、ここでのデータはホームページのみを対象としており、カテゴリ、商品、その他のページは含まれていないことにも注意してください。当社の方法論の詳細については、こちらをご覧ください。

プラットフォーム検出

ページがEコマースプラットフォーム上にあるかどうかを確認するにはどうすればいいですか？

検出はWappalyzerで行います。Wappalyzerは、Webサイトで使用されている技術を発見するためのクロスプラットフォームユーティリティです。コンテンツ管理システム、Eコマースプラットフォーム、Webサーバー、JavaScriptフレームワーク、アナリティクスツールなどを検出します。

ページ検出は常に信頼できるものでなく、サイトによっては自動攻撃から保護するために検出を明示的にブロックしている場合もあります。特定のEコマースプラットフォームを使用しているすべてのウェブサイトを捕捉することはできないかもしれませんが、検出したウェブサイトは実際にそのプラットフォームを使用していると確信しています。

302. *https://magento.com/products/magento-commerce*

	モバイル	デスクトップ
Eコマースページ	500,595	424,441
総ページ数	5,297,442	4,371,973
採用率	9.45%	9.70%

図13.2. 検出されたEコマースプラットフォームの割合。

Eコマースプラットフォーム

プラットフォーム	モバイル	デスクトップ
WooCommerce	3.98	3.90
Shopify	1.59	1.72
Magento	1.10	1.24
PrestaShop	0.91	0.87
Bigcommerce	0.19	0.22
Shopware	0.12	0.11

図13.3. 上位6つのEコマースプラットフォームの採用。

検出された116のEコマースプラットフォームのうち、デスクトップまたはモバイルサイトの0.1%以上で検出されたのは6つだけでした。これらの結果には国別、サイトの規模別、その他の類似した指標による変動は示されていません。

上記の図13.3を見ると、WooCommerceの採用率が最も高く、デスクトップおよびモバイルサイトの約4%を占めていることがわかります。Shopifyは約1.6%の採用で2位です。Magento、PrestaShop、Bigcommerce、Shopwareが0.1%に近づき、採用率が小さくなっています。

ロングテール

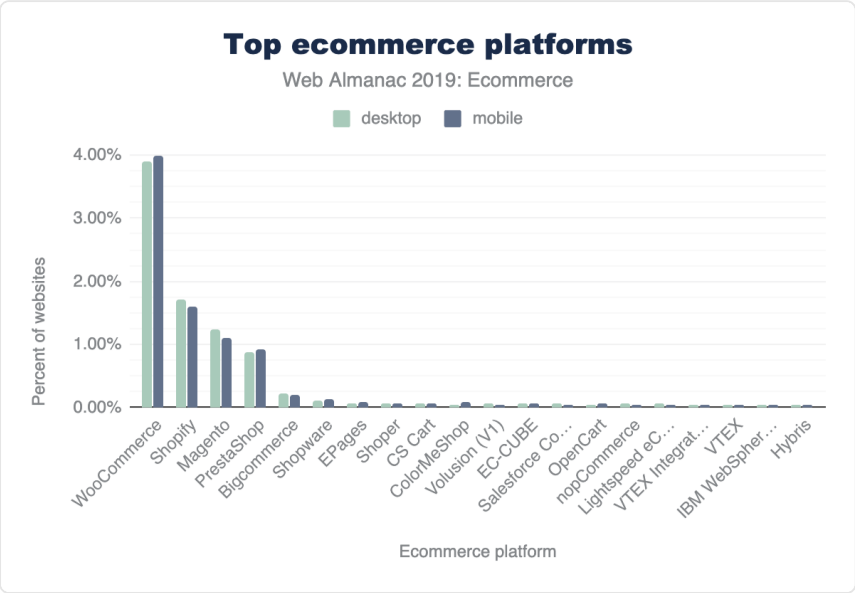


図13.4. トップのEコマースプラットフォームの採用。

110のEコマースプラットフォームがあり、それぞれがデスクトップまたはモバイルのウェブサイトの0.1%未満を持っています。そのうち約60社は、モバイルかデスクトップのウェブサイトの0.01%未満を占めています。

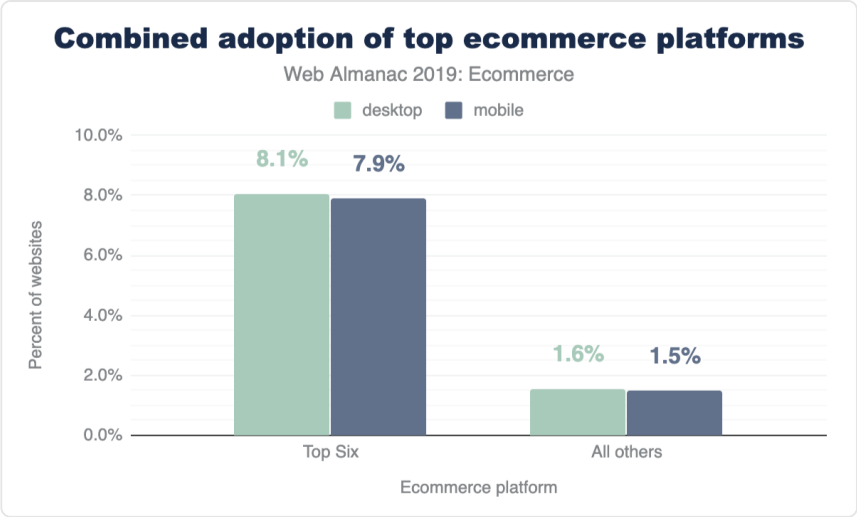


図13.5. 上位6つのEコマースプラットフォームと他の110のプラットフォームとの複合的な採用。

モバイルでのリクエストの7.87%、デスクトップでのリクエストの8.06%は、上位6つのEコマース・プラットフォームのうちの1つのホームページが対象となっています。さらにモバイルでのリクエストの1.52%、デスクトップでのリクエストの1.59%は、他の110のEコマース・プラットフォームのホームページが対象となっています。

Eコマース（すべてのプラットフォーム）

合計で、デスクトップページの9.7%、モバイルページの9.5%がEコマースプラットフォームを利用していました。

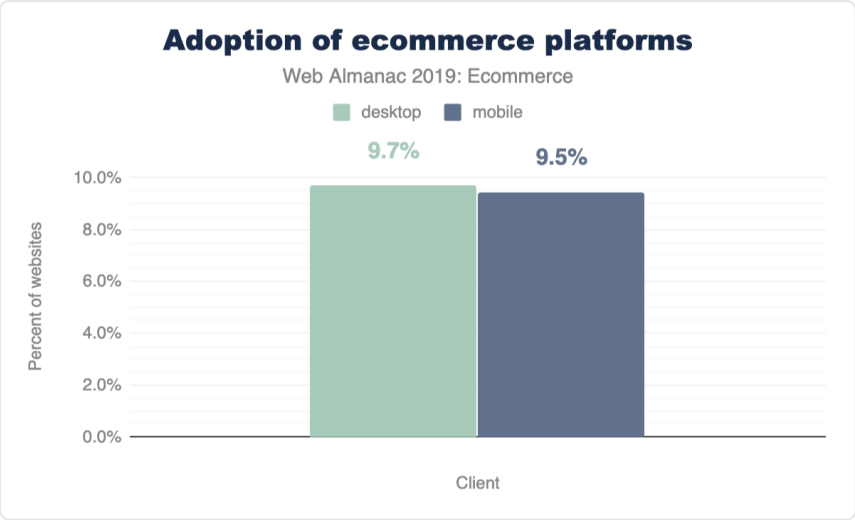


図13.6. 任意のEコマースプラットフォームを使用しているページの割合。

ウェブサイトのデスクトップ比率は全体的に若干高くなっていますが、一部の人気プラットフォーム（WooCommerce、PrestaShop、Shopwareを含む）では、実際にはデスクトップウェブサイトよりもモバイル性が高くなっています。

ページの重さと要望

Eコマースプラットフォームのページの重さは、すべてのHTML、CSS、JavaScript、JSON、XML、画像、オーディオ、およびビデオを含んでいます。

図13.7. Eコマースのページ重量の分布。

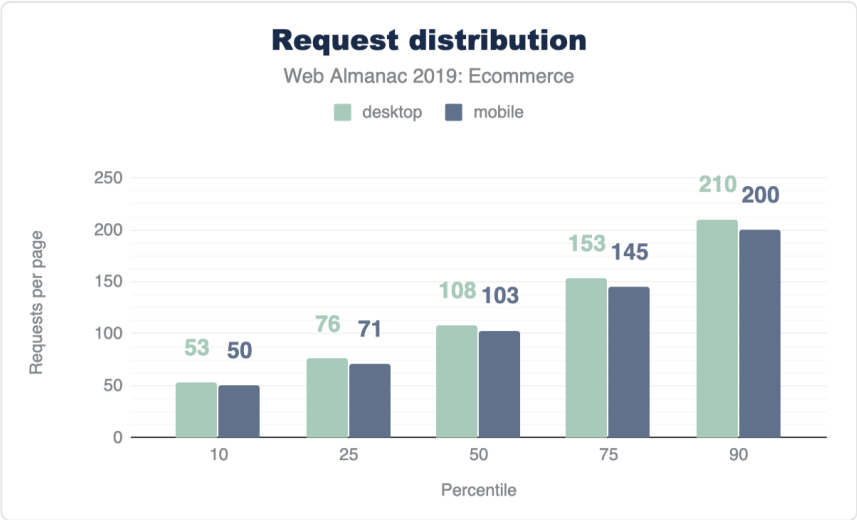


図13.8. Eコマースページごとのリクエストの分布。

デスクトップEコマースプラットフォームのページの読み込み量の中央値は108リクエストと2.7MBです。すべてのデスクトップページの重量の中央値は74リクエストと1.9 MB です。言い換えれば、Eコマースページは他のウェブページよりも50%近く多くのリクエストを行い、ペイロードは約35%大きくなっています。比較すると、amazon.com[303]のホームペー

303. https://amazon.com

ジは、最初のロード時に約5MBのページ重量に対して約300リクエストを行い、ebay.com[304]は約3MBのページウェイトに対して約150リクエストを行います。Eコマースプラットフォーム上のホームページのページ重量とリクエスト数は、各パーセンタイルでモバイルの方が若干小さくなっていますが、すべてのEコマースのホームページの約10%が7MB以上をロードし200以上のリクエストをしています。

このデータは、ホームページのペイロードとスクロールなしのリクエストを含んでいます。明らかに、最初のロードに必要なはずのファイル数よりも多くのファイルを取得しているように見えるサイトがかなりの割合で存在しています(中央値は100以上)。以下のサードパーティのリクエストとバイト数も参照してください。

Eコマース・プラットフォーム上の多くのホームページが、なぜこれほど多くのリクエストを行い、これほど大きなペイロードを持つのかをよりよく理解するために、さらに調査を行う必要があります。著者らはEコマース・プラットフォーム上のホームページで、最初のロード時に数百回のリクエストを行い、数メガバイトのペイロードを持つホームページを定期的に目にします。リクエスト数とペイロードがパフォーマンスの問題であるならば、どのようにしてそれらを減らすことができるのでしょうか？

タイプ別のリクエストとペイロード

以下の表は、デスクトップでのリクエストの場合のものです。

304. *https://ebay.com*

ファイルの種類	10	25	50	75	90
画像	353	728	1,514	3,104	6,010
ビデオ	156	453	1,325	2,935	5,965
スクリプト	199	330	572	915	1,331
フォント	47	85	144	226	339
css	36	59	102	180	306
html	12	20	36	66	119
オーディオ	7	7	11	17	140
xml	0	0	0	1	3
その他	0	0	0	0	3
テキスト	0	0	0	0	0

図13.9. リソースタイプ別のページ重量分布（KB単位）のパーセンタイル。

ファイルの種類	10	25	50	75	90
画像	16	25	39	62	97
スクリプト	11	21	35	53	75
css	3	6	11	22	32
フォント	2	3	5	8	11
html	1	2	4	7	12
ビデオ	1	1	2	5	9
その他	1	1	2	4	9
テキスト	1	1	1	2	3
xml	1	1	1	2	2
オーディオ	1	1	1	1	3

図13.10. リソースタイプ別の1ページあたりのリクエストの分布のパーセンタイル。

Eコマースページでは、画像が最大のリクエスト数とバイト数の割合を占めています。デスクトップEコマースページの中央値には、1,514KB(1.5MB)の重さの画像が39枚含まれていま

す。

JavaScriptリクエストの数は、より良いバンドル(および/またはHTTP/2多重化)によってパフォーマンスを向上する可能性があることを示しています。JavaScriptファイルの総バイト数はそれほど大きくありませんが、個別のリクエストが多くなっています。HTTP/2の章によると、リクエストの40%以上はHTTP/2経由ではないそうです。同様に、CSSファイルは3番目にリクエスト数が多いですが、一般的には少ないです。CSSファイル(またはHTTP/2)をマージすることで、そのようなサイト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。著者の経験では、多くのEコマースページでは、未使用のCSSとJavaScriptの割合が高い。ビデオ のリクエスト数は少ないかもしれませんが、(驚くことではありません) 特にペイロードが重いサイトでは、ページの重量の割合が高くなります。

HTMLペイロードサイズ

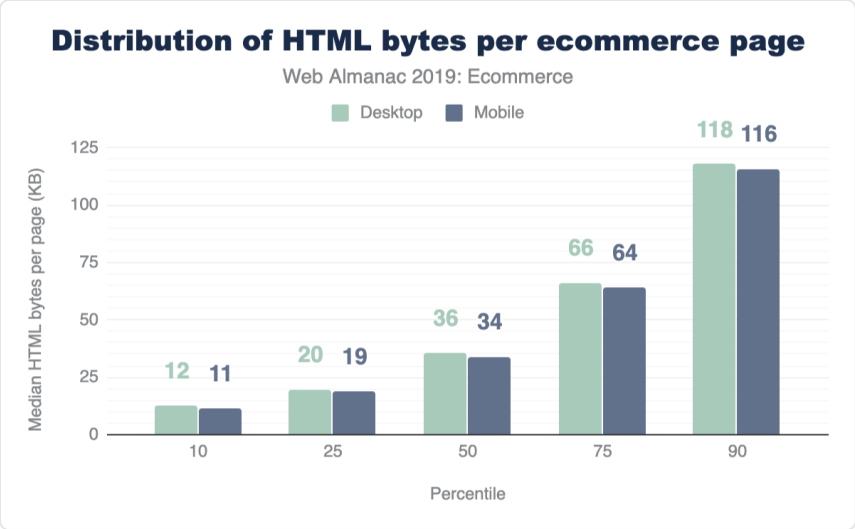


図13.11. Eコマースページあたりの HTMLバイト数の分布（KB単位）。

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インラインJSON、JavaScript、CSSなどの他のコードが直接含まれている場合があることに注意してください。EコマースページのHTMLペイロードのサイズの中央値は、モバイルで34KB、デスクトップで36KBです。しかし、Eコマースページの10%には、115KB以上のHTMLペイロードがあります。

モバイルのHTMLペイロードのサイズは、デスクトップとあまり変わりません。言い換えれば、サイトは異なるデバイスやビューポートのサイズに対して、大きく異なるHTMLファイ

ルを配信していないように見えます。多くのEコマースサイトでは、ホームページのHTMLペイロードが大きくなっています。これがHTMLの肥大化によるものなのか、それともHTMLファイル内の他のコード（JSONなど）によるものなのかはわかりません。

画像の統計

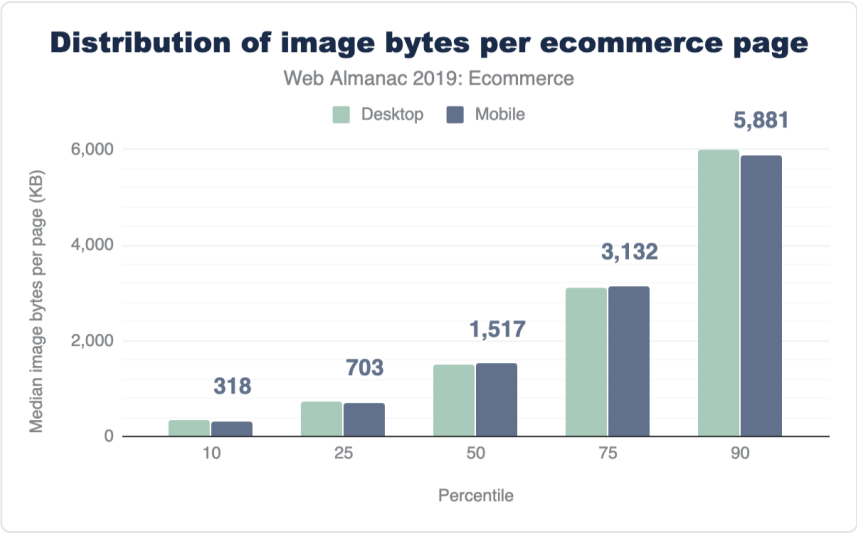


図13.12. Eコマースページごとの画像バイト数の分布（KB単位）。

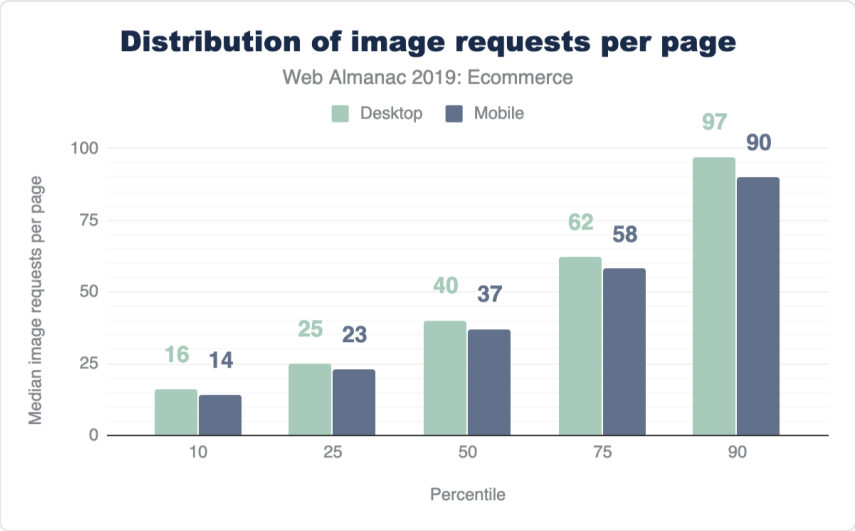


図13.13. Eコマースページごとの画像リクエストの分布。

私たちのデータ収集方法論はクリックやスクロールなど、ページ上でのユーザー操作をシミュレートしていないため、遅延して読み込まれた画像はこれらの結果には表示されないことに注意してください。

上記の図13.12と13.13で中央値のEコマースページには、モバイルでは37枚の画像と1,517KBの画像ペイロードがあり、デスクトップでは40枚の画像と1,524KBの画像ペイロードがあることを示しています。ホームページの10%は、90以上の画像と6MB近くの画像ペイロードを持っています！

1,517 KB

図13.14. モバイルEコマースページあたりの画像バイト数の中央値。

Eコマースページのかなりの割合で、大きな画像ペイロードを持ち、最初のロード時に大量の画像リクエストを行います。詳細については、HTTP ArchiveのState of Images[305]レポート、およびmediaと[page weight](./page weight)の章を参照してください。

ウェブサイトの所有者は、自分のサイトを最新のデバイスで見栄えの良いものにしたいと考えています。その結果、多くのサイトでは、画面の解像度やサイズに関係なく、すべてのユーザーに同じ高解像度の製品画像を配信しています。開発者は、異なるユーザーに可能な限り最高の画像を効率的に配信できるレスポンシブ技術に気づいていない（または使いたくな

305. *https://httparchive.org/reports/state-of-images*

い）かもしれません。高解像度の画像が必ずしもコンバージョン率を高めるとは限らないことを覚えておきましょう。逆に重い画像の使いすぎは、ページの速度に影響を与える可能性が高く、それによってコンバージョン率を低下させる可能性があります。サイトレビューやイベントでの著者の経験では、開発者やその他の関係者の中には、画像に遅延ローディングを使用することにSEOなどの懸念を持っている人もいます。

一部のサイトがレスポンシブ画像技術や遅延読み込みを使用していない理由をよりよく理解するために、より多くの分析を行う必要があります。またEコマースプラットフォームが、ハイエンドのデバイスや接続性の良いサイトに美しい画像を確実に配信すると同時に、ローエンドのデバイスや接続性の悪いサイトにも最高の体験を提供できるようなガイダンスを提供する必要があります。

Popular image formats

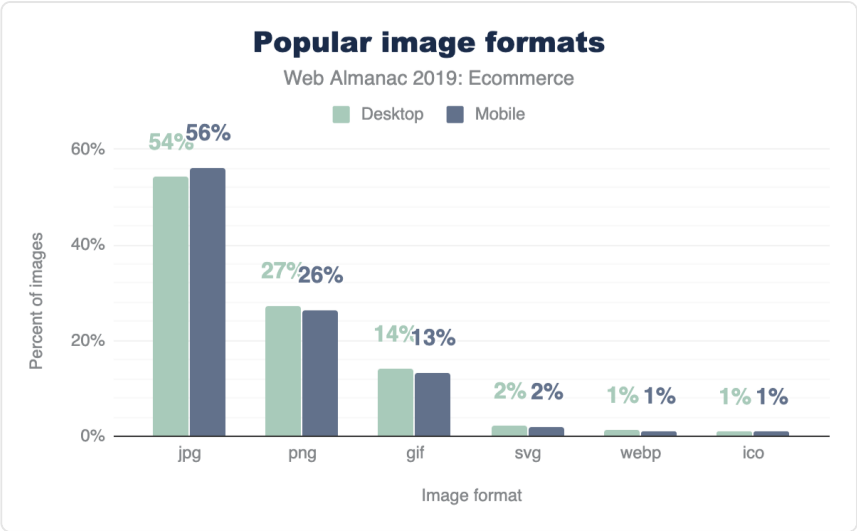


図13.15. 一般的な画像フォーマット。

画像サービスやCDNの中には、`.jpg`や`.png`という接尾辞を持つURLであっても、WebPをサポートしているプラットフォームには自動的にWebP(JPEGやPNGではなく)を配信するものがあることに注意してください。たとえば、IMG_20190113_113201.jpg[306]はChromeでWebP画像を返します。しかし、HTTP Archive画像フォーマットを検出する方法[307]は、最初にMIMEタイプのキーワードをチェックしてから、ファイルの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WebPをサポートしているため、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。

306. https://res.cloudinary.com/webdotdev/f_auto/w_500/IMG_20190113_113201.jpg
307. https://github.com/HTTPArchive/legacy.httparchive.org/blob/31a25b9064a365d746d4811a1d6dda516c0e4985/bulktest/batch_lib.inc#L994

PNG

Eコマースページの4つに1つの画像はPNGです。Eコマースプラットフォーム上のページでPNGのリクエストが多いのは、商品画像のためと思われます。多くのコマースサイトでは、透過性を確保するために写真画像と一緒にPNGを使用しています。

PNGフォールバックでWebPを使用することは、画像要素[308]を介して、またはCloudinary[309]のような画像サービスを介してユーザーエージェントの能力検出を使用することで、はるかに効率的な代替手段となります。

WebP

Eコマースプラットフォーム上の画像の1%だけがWebPであり、これはサイトレビューやパートナーの仕事での著者の経験と一致しています。WebPはSafari以外のすべての最新ブラウザでサポートされています[310]し、フォールバックの仕組みも充実しています。WebPは透過性をサポートしており、写真画像のためのPNGよりもはるかに効率的なフォーマットです（上記のPNGのセクションを参照してください）。

WebPをPNGのフォールバックで使用したり、無地の色の背景でWebP/JPEGを使用して透明化を可能にするため、Webコミュニティとして、より良いガイダンスや提唱を提供できます。WebPは、ガイド[311]やツール (例:Squoosh[312]やcwebp[313]など)があるにもかかわらず、電子商取引プラットフォームではほとんど使用されていないようです。現在10年近く経っている[314]WebPの利用が増えていない理由をさらに調査する必要があります。

308. http://simpl.info/picturetype
309. https://res.cloudinary.com/webdotdev/f_auto/w_500/IMG_20190113_113201.jpg
310. https://caniuse.com/#feat=webp
311. https://web.dev/i18n/ja/serve-images-webp
312. https://squoosh.app/
313. https://developers.google.com/speed/webp/docs/cwebp
314. https://blog.chromium.org/2010/09/webp-new-image-format-for-web.html

画像の寸法

	モバイル		デスクトップ	
パーセンタイル	横幅(px)	高さ(px)	横幅(px)	高さ(px)
10	16	16	16	16
25	100	64	100	60
50	247	196	240	192
75	364	320	400	331
90	693	512	800	546

図13.16. Eコマースページごとの固有画像の寸法（ピクセル単位）の分布。

Eコマースページで要求された画像の中央値（「中間値」）は、モバイルで247X196px、デスクトップで240X192pxです。Eコマースページで要求される画像の10%は、モバイルでは693X512px以上、デスクトップでは800X546px以上です。これらの寸法は画像の本質的なサイズであり、表示サイズではないことに注意してください。

中央値までの各パーセンタイルでの画像のサイズがモバイルとデスクトップで似ていることを考えると、多くのサイトではビューポートごとに異なるサイズの画像を配信していない、言い換えればレスポンシブ画像技術を使用していないように思えます。モバイル向けに大きな画像が配信されている場合もありますが、これはデバイス検出や画面検出を使用しているサイトによって説明できるかもしれません（そうでないかもしれません！）。

なぜ多くのサイトが（一見して）異なる画像サイズを異なるビューポートに配信していないのか、もっと研究する必要があります。

サードパーティからのリクエストとバイト

多くのウェブサイト、特にオンラインストアでは、分析、A/Bテスト、顧客行動追跡、広告、ソーシャルメディアのサポートなどのためにサードパーティのコードやコンテンツを大量にロードしています。サードパーティのコンテンツは、パフォーマンスに大きな影響を与える[315]ことがあります。 Patrick Hulce[316]のサードパーティウェブツール[317]は、本レポートのサードパーティのリクエストを判断するために使用されており、これについてはサードパーテ

315. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript
316. https://x.com/patrickhulce
317. https://github.com/patrickhulce/third-party-web

ィの章で詳しく説明しています。

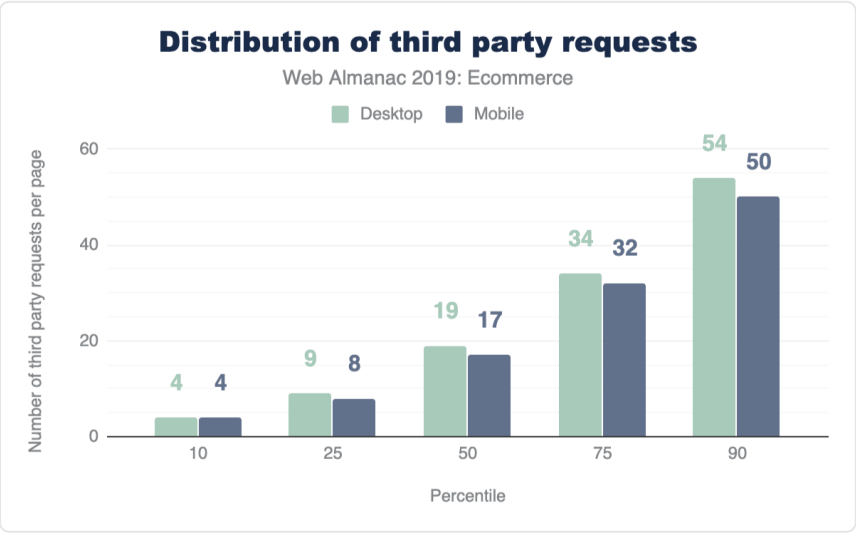


図13.17. Eコマースページごとのサードパーティリクエストの分布。

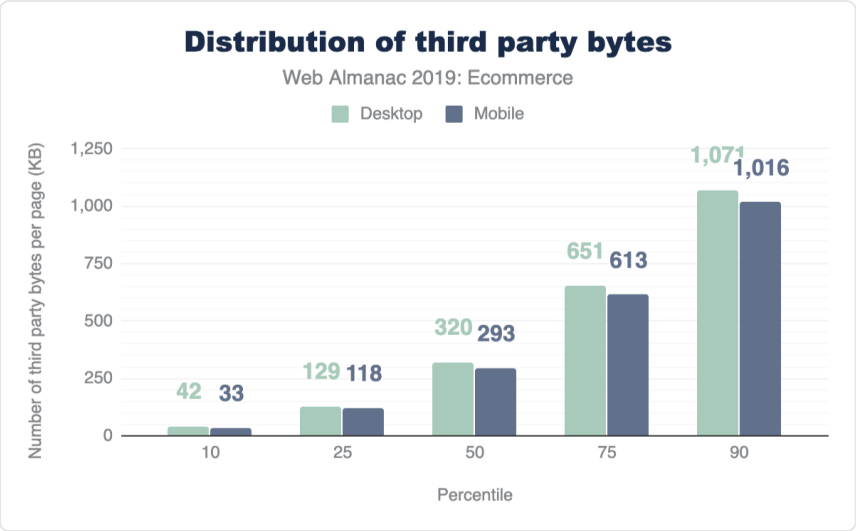


図13.18. Eコマースページあたりのサードパーティのバイト数の分布。

Eコマースプラットフォーム上の中央値（「中規模」）のホームページでは、サードパーティのコンテンツに対するリクエストは、モバイルで17件、デスクトップで19件となっています。Eコマース・プラットフォーム上のすべてのホームページの10%は、サードパーティ

のコンテンツに対して50件以上のリクエストを行い、その総ペイロードは1MBを超えています。

他の研究[318]で、サードパーティのコンテンツはパフォーマンスの大きなボトルネックになる可能性であることが指摘されています。この調査によると、17以上のリクエスト（上位10%では50以上）がEコマースページの標準となっています。

プラットフォームごとのサードパーティリクエストとペイロード

以下の表は、モバイルのみのデータを示しています。

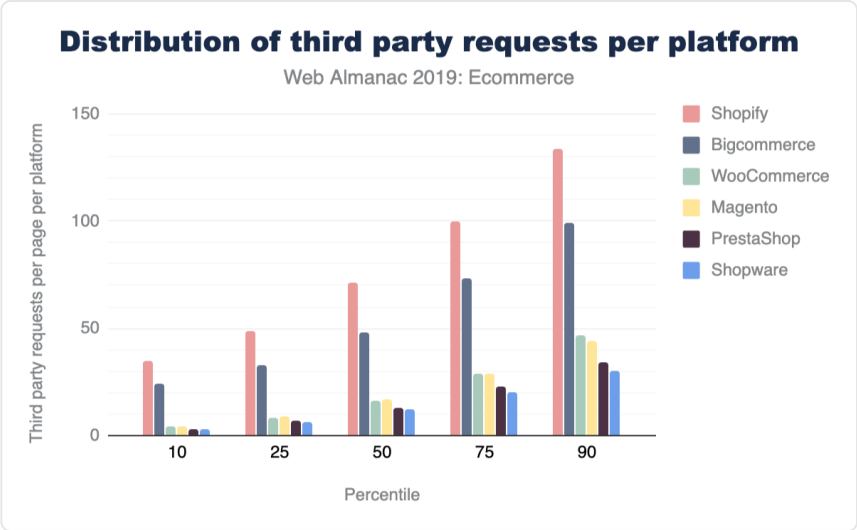


図13.19. 各Eコマースプラットフォームのモバイルページごとのサードパーティリクエストの分布。

318. *https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/loading-third-party-javascript/*

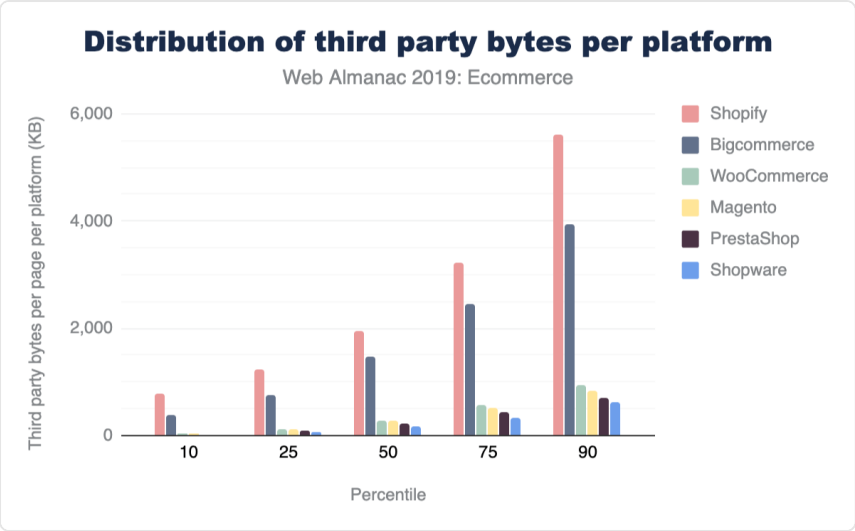


図13.20. 各Eコマースプラットフォームのモバイルページあたりのサードパーティのバイト数（KB）分布。

Shopifyのようなプラットフォームでは、クライアントサイドのJavaScriptを使ってサービスを拡張することがありますが、Magentoのような他のプラットフォームではサーバーサイドの拡張機能が多く使われています。このアーキテクチャの違いが、ここで見る数字に影響を与えています。

明らかに、一部のEコマースプラットフォームのページでは、サードパーティコンテンツへのリクエストが多く、サードパーティコンテンツのペイロードが大きくなっています。一部のプラットフォームのページで、サードパーティコンテンツへのリクエストが多く、サードパーティコンテンツのペイロードが他のプラットフォームよりも大きいのはなぜかについて、さらに分析を行うことができます。

コンテンツの初回ペイント(FCP)

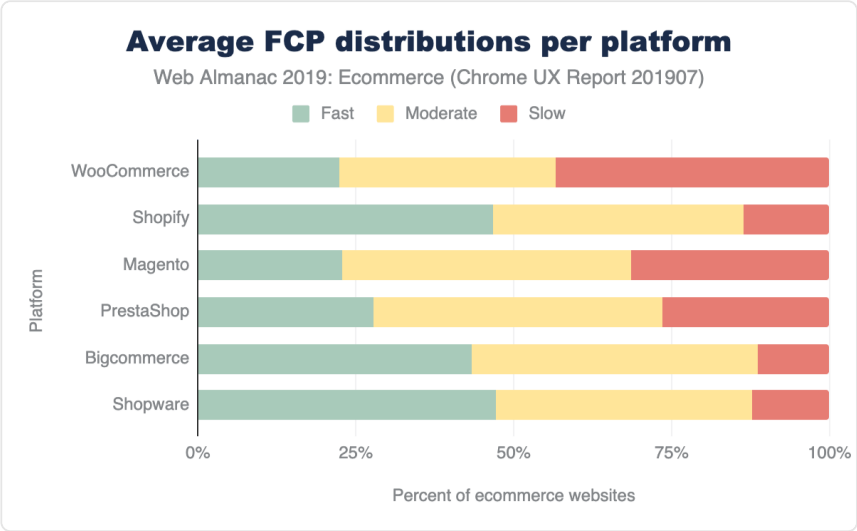


図13.21. Eコマースプラットフォーム毎のFCP体験の平均分布。

コンテンツの初回ペイントは、ナビゲーション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します。この文脈では、**速い**は1秒未満のFCP、**遅い**は3秒以上のFCP、**中程度**はその中間のすべてを意味します。サードパーティのコンテンツやコードは、FCPに大きな影響を与える可能性があることに注意してください。

上位6つのEコマースプラットフォームはすべて、モバイルでのFCPがデスクトップよりも悪くなっています。FCPは、接続性だけでなく、デバイスの能力（処理能力、メモリなど）にも影響されることに注意してください。

FCPがデスクトップよりもモバイルの方が悪い理由を明らかにする必要があります。原因は何でしょうか？　接続性やデバイスの能力、それとも何か他の要因でしょうか？

プログレッシブウェブアプリ（PWA）のスコア

Eコマースサイト以外のこのトピックの詳細については、PWAの章も参照してください。

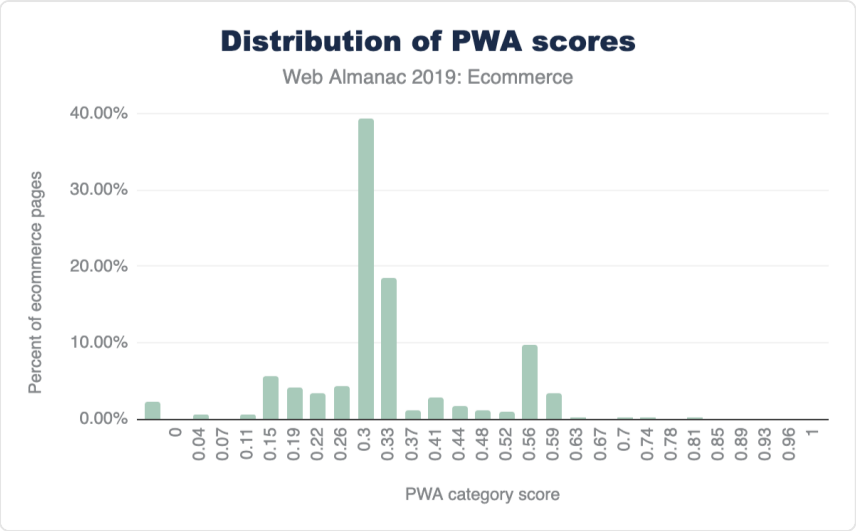


図13.22. モバイルEコマースページのLighthouse PWAカテゴリスコアの分布。

Eコマースのプラットフォーム上のホームページの60%以上は、0.25と0.35の間にLighthouse PWAスコア[319]を取得します。Eコマースのプラットフォーム上のホームページの20%未満は、0.5以上のスコアを取得し、ホームページの1%未満は0.6以上のスコアを取得します。

Lighthouseは、プログレッシブWebアプリ（PWA）のスコアを0から1の間で返します。PWAの監査は、14の要件をリストアップしたBaseline PWA Checklist[320]に基づいています。Lighthouseは、14の要件のうち11の要件について自動監査を実施しています。残りの3つは手動でしかテストできません。11の自動PWA監査はそれぞれ均等に重み付けされているため、それぞれがPWAスコアに約9ポイント寄与します。

PWA監査のうち少なくとも1つがnullスコアを取得した場合、LighthouseはPWAカテゴリ全体のスコアをnullアウトします。これは、モバイルページの2.32%が該当しました。

明らかに、大多数のEコマースページは、ほとんどのPWA チェックリスト監査[321] に失敗しています。どの監査が失敗しているのか、なぜ失敗しているのかをよりよく理解するために、さらに分析を行う必要があります。

319. *https://developers.google.com/web/ilt/pwa/lighthouse-pwa-analysis-tool*
320. *https://developers.google.com/web/progressive-web-apps/checklist*
321. *https://developers.google.com/web/progressive-web-apps/checklist*

結論

Eコマースの使用法のこの包括的な研究はいくつかの興味深いデータを示し、また同じEコマースのプラットフォーム上に構築されたものの間でも、Eコマースのサイトの広いバリエーションを示しています。ここでは多くの詳細を説明しましたが、この分野ではもっと多くの分析が可能です。例えば、今年はアクセシビリティのスコアを取得していませんでした（それについての詳細はアクセシビリティの章をチェックアウトしてください）。同様に、これらのメトリクスを地域別にセグメント化することも興味深いことでしょう。この調査では、Eコマース・プラットフォームのホームページ上で246の広告プロバイダーが検出されました。さらなる調査（おそらく来年のWeb Almanacに掲載されるかもしれません）では、Eコマースプラットフォーム上で広告を表示しているサイトの割合を計算できます。この調査ではWooCommerceが非常に高い数値を記録していますので、来年の調査では一部のホスティングプロバイダーがWooCommerceをインストールしているにもかかわらず、有効にしていないために数値が膨らんでいるのではないかという興味深い統計を見ることができます。

著者

Sam Dutton

@sw12 samdutton https://simpl.info

Sam Duttonは2011年からDeveloper AdvocateとしてGoogle Chromeチームで働いています。数々のイベントを企画して発表し、いくつかのウェブ開発コースを作成して教え、PWA、パフォーマンス、メディア、イメージ、「Next Billion Users」イニシアティブをカバーする様々なビデオ、コードラボ、文書化されたガイダンスに取り組んできました。彼はsimpl.info[322]を管理しており、HTML、CSS、JavaScriptの最もシンプルな例を提供しています。南オーストラリアで育ち、シドニーの大学に進学し、1986年からロンドンに住んでいます。

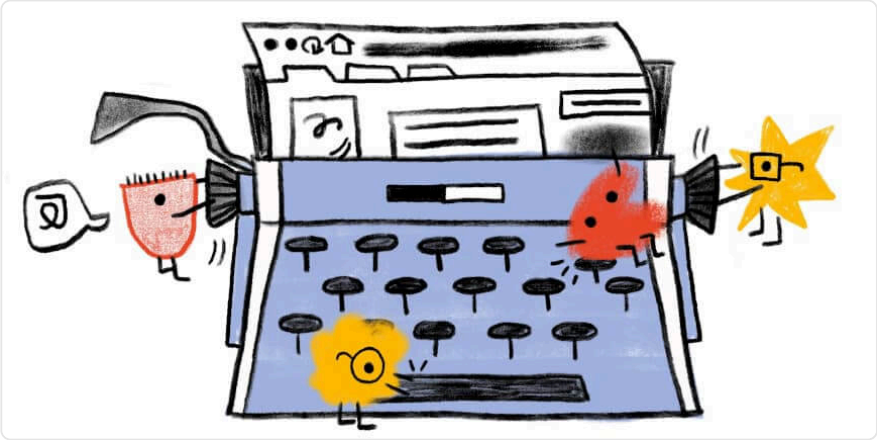
Alan Kent

@akent99 alankent https://alankent.me

Alan KentはGoogleのDeveloper Advocateで、Eコマースとコンテンツエコシステムに焦点を当てています。彼は alankent.me[323] でブログを書いており、@akent99 としてツイートしています。

322. *https://simpl.info*
323. *https://alankent.me*

部 III 章 14
CMS



Renee Johnson と Alberto Medina によって書かれた。
Jonathan Wold によってレビュ。
Rick Viscomi による分析 と 編集。
Sakae Kotaro によって翻訳された。

序章

コンテンツ管理システム（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総称しています。具体的には、オープンウェブを介して消費・体験でき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のことを指します。

各CMSは、ユーザーがコンテンツを中心に簡単かつ効果的にウェブサイトを構築できるように、幅広いコンテンツ管理機能とそれに対応するメカニズムのサブセットを実装しています。このようなコンテンツは多くの場合、何らかのデータベースに保存されており、ユーザーはコンテンツ戦略のために必要な場所であればどこでも再利用できる柔軟性を持っています。CMSはまた、ユーザーが必要に応じてコンテンツを簡単にアップロードして管理できるようにすることを目的とした管理機能を提供します。

サイト構築のためにCMSが提供するサポートの種類と範囲には大きなばらつきがあり、ユーザーコンテンツで「水増し」されたすぐに使えるテンプレートを提供するものもあれば、サイト構造の設計と構築にユーザーの関与を必要とするものもあります。

CMSについて考えるとき、ウェブ上に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行可能性に関わるすべてのコンポーネントを考慮に入れる必要があります。これらのコンポーネントはすべて、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、拡張機能開発、開発代理、サイトビルダーなどが含まれています。このように、CMSというと、通常はプラットフォームそのものとそれを取り巻くエコシステムの両方を指すことになります。

コンテンツ制作者はなぜ**CMS**を使うのか？

(ウェブの進化）時代の初期にはウェブのエコシステムはユーザーがウェブページのソースを見て、必要に応じてコピーペーストし画像などの個別の要素で新しいバージョンをカスタマイズするだけでクリエイターになれるという、単純な成長ループで動いていました。

ウェブが進化するにつれ、ウェブはより強力になる一方で、より複雑になりました。その結果、その単純な成長のループは破られ、誰でもクリエイターになれるような状況ではなくなってしまいました。コンテンツ制作の道を追求できる人にとっては、その道のりは険しく困難なものになってしまいました。ウェブでできることと実際にできることの差である利用可能性ギャップ[324]は着実に拡大していきました。

324. https://medinathoughts.com/2018/05/17/progressive-wordpress/

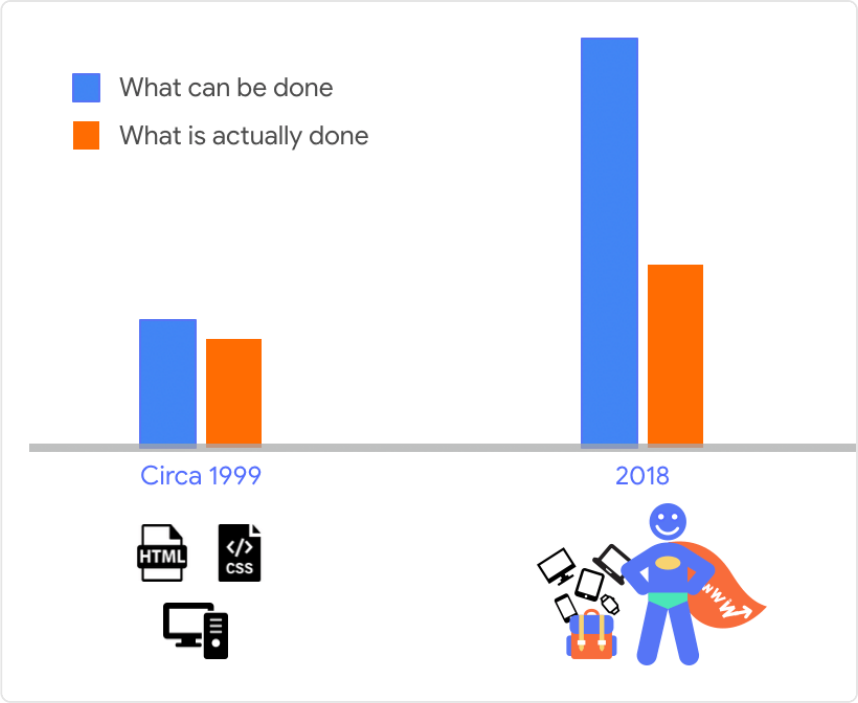


図14.1. 1999年から2018年までのWeb機能の増加を示すグラフ。

ここでCMSが果たす役割は、技術的な専門性の異なるユーザーがコンテンツ制作者としてウェブのエコシステムのループに入りやすくするという非常に重要なものです。コンテンツ制作への参入障壁を下げることで、ユーザーをクリエイターに変えることで、ウェブの成長ループを活性化させます。それが人気の理由です。

この章の目標

私たちはCMS空間とウェブの現在と未来におけるその役割を理解するための探求の中で、分析すべき多くの興味深い重要な側面があり、答えるべき質問があります。私たちはCMSプラットフォーム空間の広大さと複雑さを認識しており、そこにあるすべてのプラットフォームに関わるすべての側面を完全にカバーする全知全能の知識を主張しているわけではありませんが、私たちはこの空間への魅力を主張しこの空間の主要なプレイヤーのいくつかについて深い専門知識を持っています。

この章では広大なCMSの空間の表面領域のスクラッチを試み、CMSエコシステムの現状とコンテンツがウェブ上でどのように消費され、どのように体験されるかについてのユーザーの認識を形成する上でのCMSの役割について私たちの全体的な理解に光を当てようとしていま

す。私たちの目標はCMSの状況を網羅的に見ることではなく、CMSの状況全般に関連するいくつかの側面と、これらのシステムによって生成されたウェブページの特徴について論じていきたいと思います。このWeb Almanacの初版はベースラインを確立するものであり、将来的には、トレンド分析のためにこのバージョンとデータを比較できるようになるでしょう。

CMS導入

40%

図14.2. CMSを搭載したウェブページの割合。

今日では、ウェブページの40%以上が何らかのCMSプラットフォームを利用していることがわかります。40.01%がモバイル用で、39.61%がデスクトップ用です。

他にもW3Techs[325]のようにCMSプラットフォームの市場シェアを追跡しているデータセットがあり、CMSプラットフォームを利用したウェブページの割合が50%を超えていることを反映しています。さらに、これらのデータはCMSプラットフォームが成長しており、場合によっては前年比12%の成長率を記録しています。弊社の分析とW3Techの分析との乖離は、調査方法の違いによって説明できるかもしれません。我々の分析については、方法論のページを参照してください。

要するに、多くのCMSプラットフォームが存在するということです。下の写真は、CMSの風景を縮小したものです。

325. *https://w3techs.com/technologies/history_overview/content_management*

図14.3. 上位のコンテンツ管理システム。

その中には、オープンソース（WordPress、Drupalなど）のものもあれば、有償（AEMなど）のものもあります。CMSプラットフォームの中には「無料」のホスティングプランやセルフホスティングプランで利用できるものもありますし、企業レベルでも、より高い階層のプランで利用できる高度なオプションもあります。CMS空間全体として複雑で連携したCMSエコシステムの世界であり、全てが分離され、同時にウェブの広大な構造に絡み合っています。

またCMSプラットフォームを利用したウェブサイトが何億もあり、これらのプラットフォームを利用してウェブにアクセスし、コンテンツを消費するユーザーが桁違いに増えていることを意味しています。このように、これらのプラットフォームは、常緑で健康的で活力に満ちたウェブを目指す私たちの集団的な探求を成功させるために重要な役割を果たしています。

CMSの風景

今日のウェブの大部分は、ある種のCMSプラットフォームを利用しています。この現実を反映して、さまざまな組織が収集した統計となります。Chrome UXレポート(CrUX)とHTTP Archiveのデータセットを見ると、データセットの特殊性を反映して定量的には記載されている割合は異なるかもしれませんが、他の場所で発表されている統計と一致している図が得られます。

デスクトップとモバイルデバイスで提供されているウェブページを見てみると、何らかのCMSプラットフォームによって生成されたページとそうでないページの割合が約60-40%に分かれていることがわかります。

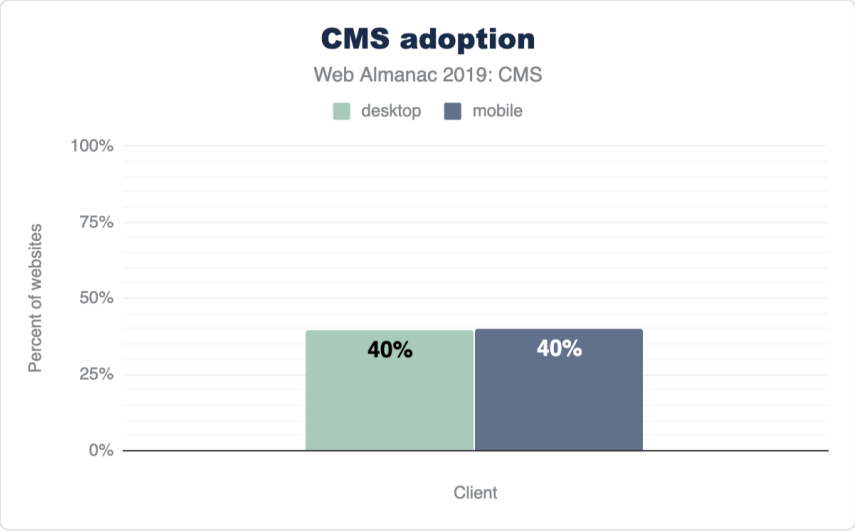


図14.4. CMSを使用しているデスクトップおよびモバイルサイトの割合。

CMSを搭載したウェブページは、利用可能なCMSプラットフォームの大規模なセットによって生成されます。そのようなプラットフォームの中から選択するには多くのものがあり、1つを使用することを決定する際に考慮できる多くの要因があり、以下のようなものがあります。

- コア機能
- 作成・編集のワークフローと体験
- 参入障壁
- カスタマイズ性

- コミュニティ
- 他にもたくさん

CrUXとHTTP Archiveのデータセットには、約103のCMSプラットフォームが、混在したウェブページが含まれています。これらのプラットフォームのほとんどは、相対的な市場シェアが非常に小さいものです。今回の分析では、データに反映されているウェブ上でのフットプリントという観点から、上位のCMSプラットフォームに焦点を当ててみたいと思います。完全な分析については、この章の結果のスプレッドシートを参照してください[326]。

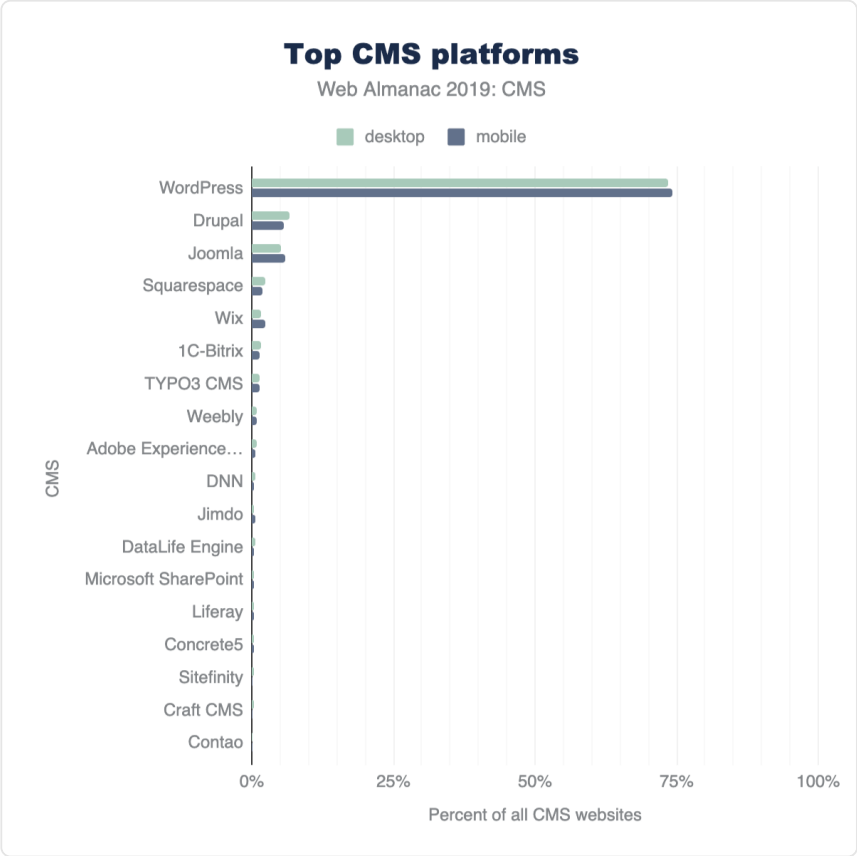


図14.5. 全CMSウェブサイトに占めるトップCMSプラットフォームの割合。

データセットに含まれる最も顕著なCMSプラットフォームを図14.5に示す。WordPressはモバイルサイトの74.19%、デスクトップサイトの73.47% を占めています。CMSの世界におけ

326. *https://docs.google.com/spreadsheets/d/1FDYe6QdoY3UtXodE2estTdwMsTG-hHNrOe9wEYLIwAw/edit#gid=0*

るWordPressの優位性は、後述するいくつかの要因に起因していますが、WordPressは主要なプレイヤーです。DrupalやJoomlaのようなオープンソースのプラットフォームと、SquarespaceやWixのようなクローズドなSaaSが上位5つのCMSを占めています。これらのプラットフォームの多様性は、多くのプラットフォームからなるCMSエコシステムを物語っています。また、興味深いのは、上位20位までの小規模CMSプラットフォームのロングテールです。企業向けに提供されているものから、業界特有の用途のために社内で開発された独自のアプリケーションまで、コンテンツ管理システムは、グループがウェブ上で管理、公開、ビジネスを行うためのカスタマイズ可能なインフラストラクチャを提供しています。

WordPressプロジェクト[327]は、そのミッションを「*出版の民主化*」と定義しています。その主な目標のいくつかは、使いやすさと、誰もがウェブ上でコンテンツを作成できるようにソフトウェアを無料で利用できるようにすることです。もう1つの大きな要素は、このプロジェクトが育んでいる包括的なコミュニティです。世界のほとんどの大都市ではWordPressプラットフォームを理解し、構築しようと定期的に集まり、つながりを持ち共有し、コードを書く人々のグループを見つけることができます。地域のミートアップや年次イベントに参加したり、ウェブベースのチャンネルに参加したりすることは、WordPressの貢献者、専門家、ビジネス、愛好家がそのグローバルなコミュニティに参加する方法の一部となっています。

WordPressの人気は参入障壁の低さと、ユーザー（オンラインと対面）がプラットフォーム上でのパブリッシングをサポートし、拡張機能（プラグイン）やテーマを開発するためのリソースが要因となっています。またWordPressのプラグインやテーマは、ウェブデザインや機能性を追求した実装の複雑さを軽減してくれるので、利用しやすく経済的です。これらの側面が、新規参入者によるリーチと採用を促進するだけでなく、長期的な使用を維持しています。

オープンソースのWordPressプラットフォームは、ボランティア、WordPress Foundation、そしてウェブエコシステムの主要なプレイヤーによって運営されサポートされています。これらの要素を考慮すると、WordPressを主要なCMSとすることは理にかなっています。

CMSを搭載したサイトはどのように構築されているのか

それぞれのCMSプラットフォームのニュアンスや特殊性とは無関係に、最終的な目標は、オープンウェブの広大なリーチを介してユーザーに提供するウェブページを出力することにあります。CMSを搭載したウェブページとそうでないウェブページの違いは、前者では最終的な結果の構築方法のほとんどをCMSプラットフォームが決定するのに対し後者ではそのような抽象化された層がなく、すべての決定は開発者が直接またはライブラリの設定を介して行うという点にあります。

327. https://ja.wordpress.org/about/

このセクションでは、CMS空間の現状を出力の特徴（使用された総リソース、画像統計など）の観点から簡単に見ていき、ウェブエコシステム全体とどのように比較するかを見ていきます。

リソースの総使用量

どんなWebサイトでも、その構成要素がCMSサイトを作っています。HTML、CSS、JavaScript、media（画像や動画）です。CMSプラットフォームは、これらのリソースを統合してWeb体験を作成するための強力に合理化された管理機能をユーザーに提供します。これは、これらのアプリケーションの最も包括的な側面の1つですが、より広いウェブに悪影響を及ぼす可能性があります。

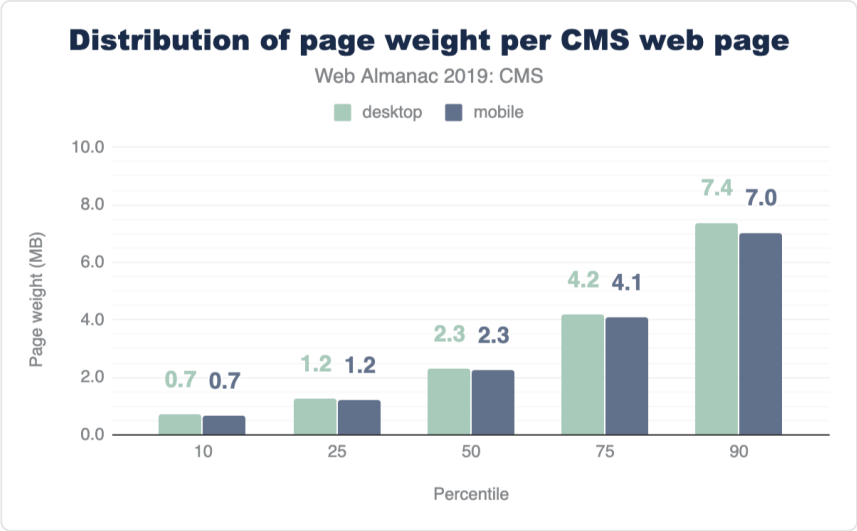


図14.6. CMSページ重量の分布。

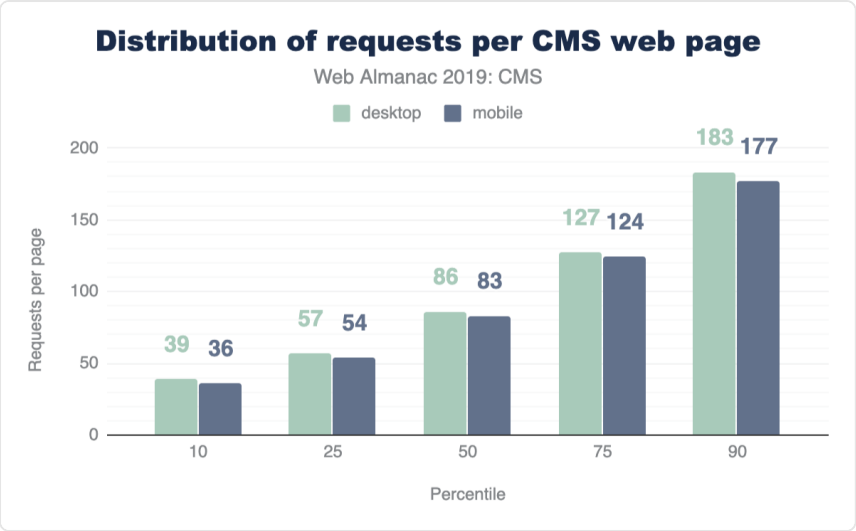


図14.7. ページあたりのCMSリクエストの分布。

上の図14.6と7では、デスクトップCMSページの中央値は86のリソースと2.29MBの重さをロードしていることがわかります。モバイルページのリソース使用量は、83のリソースと2.25 MBと、それほど大きくはありません。

中央値は、すべてのCMSページが上か下かの中間点を示しています。つまり全CMSページの半分はリクエスト数が少なく、重量が少ないのに対し、半分はリクエスト数が多く、重量が多いということになります。10パーセンタイルではモバイルとデスクトップのページはリクエスト数が40以下で重量が1MBですが、90パーセンタイルではリクエスト数が170以上で重量が7MBとなり、中央値の3倍近くになっています。

CMSのページは、ウェブ全体のページと比較してどうでしょうか？　ページ重量の章では、リソースの使用量についてのデータを見つけることができます。中央値では、デスクトップページは74リクエストで1.9MBを読み込み、ウェブ上のモバイルページは69リクエストで1.7MBを読み込みます。中央値では、CMSページはこれを上回っています。また、CMSページは90パーセンタイルでウェブ上のリソースを上回っていますが、その差はもっと小さいです。要するに、CMSページは最も重いページの1つと考えられます。

パーセンタイル	image	video	script	font	css	audio	html
50	1,233	1,342	456	140	93	14	33
75	2,766	2,735	784	223	174	97	66
90	5,699	5,098	1,199	342	310	287	120

図14.8. リソースタイプごとのデスクトップCMSページのキロバイト数の分布。

パーセンタイル	image	video	script	css	font	audio	html
50	1,264	1,056	438	89	109	14	32
75	2,812	2,191	756	171	177	38	67
90	5,531	4,593	1,178	317	286	473	123

図14.9. リソースタイプごとのモバイルCMSページのキロバイト分布。

モバイルやデスクトップのCMSページにロードされるリソースの種類を詳しく見ると、画像や動画は、その重さの主な貢献者としてすぐに目立ちます。

影響は必ずしもリクエスト数と相関するわけではなく、個々のリクエストにどれだけのデータが関連付けられているかということです。例えば、中央値で2つのリクエストしかない動画リソースの場合、1MB以上の負荷がかかります。マルチメディア体験には、スクリプトを使用してインタラクティブ性を統合したり、機能やデータを提供したりすることもあります。モバイルページとデスクトップページの両方で、これらは3番目に重いリソースです。

CMSの経験がこれらのリソースで飽和状態にある中で、フロントエンドのウェブサイト訪問者に与える影響を考慮しなければなりません。さらに、モバイルとデスクトップのリソース使用量を比較すると、リクエストの量と重さにはほとんど差がありません。つまり、同じ量と重量のリソースがモバイルとデスクトップの両方のCMS体験を動かしていることになります。接続速度とモバイルデバイスの品質のばらつきは、もう一つの複雑さの層[328]を追加します。この章の後半では、CrUXのデータを使用して、CMS空間でのユーザー体験を評価します。

サードパーティのリソース

リソースの特定のサブセットを強調して、CMSの世界での影響を評価してみましょう。サードパーティリソースとは、送信先サイトのドメイン名やサーバーに属さないオリジンからのリソースです。画像、動画、スクリプト、その他のリソースタイプがあります。これらのリ

328. https://medinathoughts.com/2017/12/03/the-perils-of-mobile-web-performance-part-iii/

ソースは、例えば`iframe`を埋め込むなど、組み合わせてパッケージ化されていることもあります。当社のデータによると、デスクトップとモバイルの両方で、サードパーティのリソースの中央値は近いことがわかります。

モバイルCMSページのサードパーティリクエストの中央値は15、重さ264.72KBでデスクトップCMSページのサードパーティリクエストの中央値は16、重さ271.56KBです。(これは「ホスティング」の一部とみなされる3Pリソースを除いたものであることに注意)。

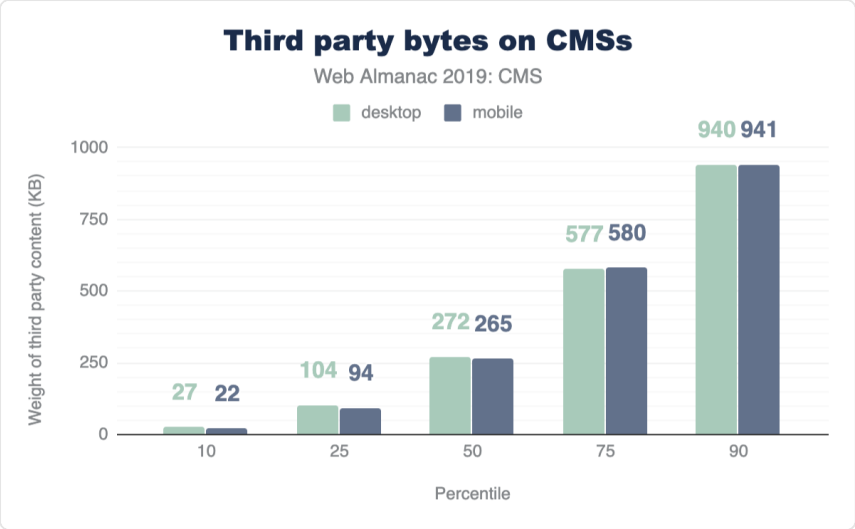


図14.10. CMSページにおけるサードパーティウェイト（KB）の分布。

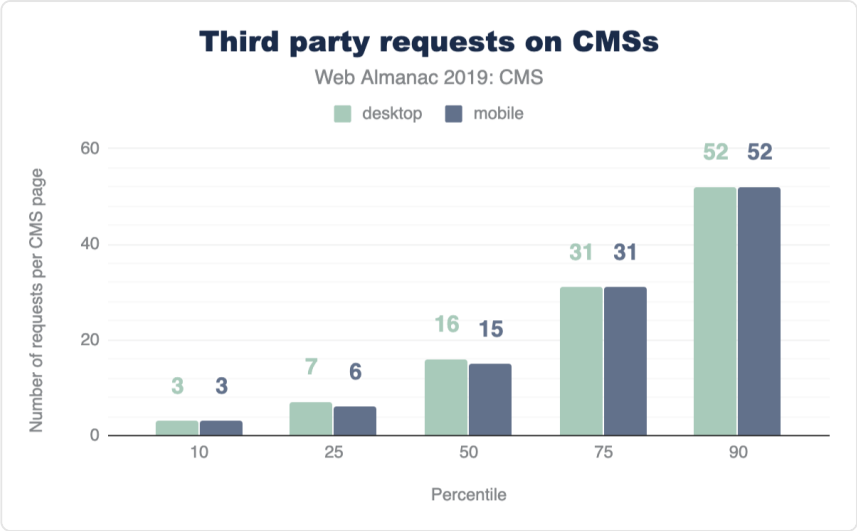


図14.11. CMSページの第三者リクエスト数の分布。

中央値は、少なくとも半分のCMSウェブページが、ここで報告している値よりも多くのサードパーティのリソースを提供していることを示しています。90パーセンタイルではCMSページは約940KBで52のリソースを配信できますが、これはかなりの増加です。

サードパーティのリソースがリモートドメインやサーバーからのものであることを考えると、送信先のサイトは、これらのリソースの品質やパフォーマンスへの影響をほとんどコントロールできません。この予測不可能性が速度の変動につながり、ユーザー体験に影響を与える可能性があります。

画像の統計

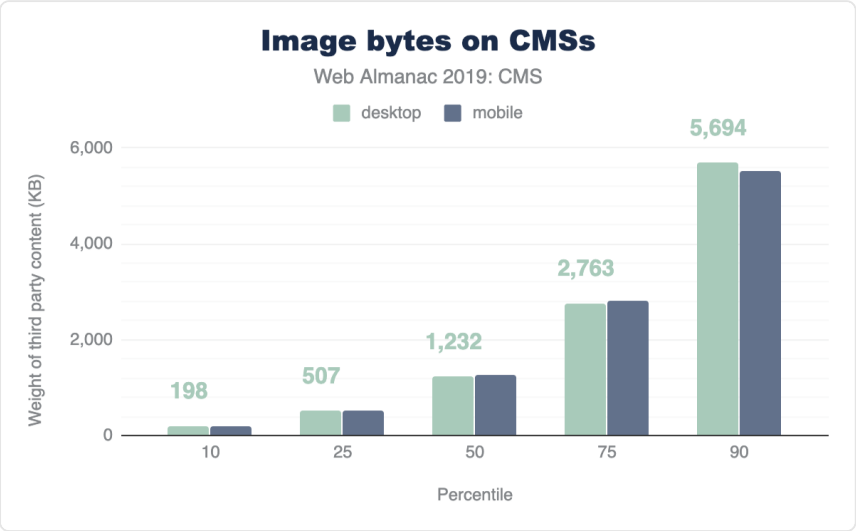


図14.12. CMSページにおける画像の重み（KB）の分布。

1,232 KB

図14.13. デスクトップCMSページあたりの画像の読み込みキロバイト数の中央値。

先に図14.8と14.9を見て、画像はCMSページの総重量に大きく寄与していることを思い出してください。上記の図14.12と14.13は、デスクトップCMSページの中央値は31枚の画像とペイロードが1,232KBであるのに対し、モバイルCMSページの中央値は29枚の画像とペイロードが1,263KBであることを示しています。ここでも私たちは、デスクトップとモバイルの両方の経験のためのこれらのリソースの重量のための非常に近いマージンを持っています。ページ重量の章では、さらに、画像リソースがウェブ全体で同じ量の画像を持つページの重量の中央値を十分に上回っていることが示されています。その結果は以下の通りです。CMSページは重い画像を供給している。

モバイルやデスクトップのCMSページでよく見られるフォーマットは何ですか？　当社のデータによると、平均的にJPG画像が最も人気のある画像フォーマットです。次いでPNG、GIFが続き、SVG、ICO、WebPのようなフォーマットが2%強、1%強と大きく後れを取っています。

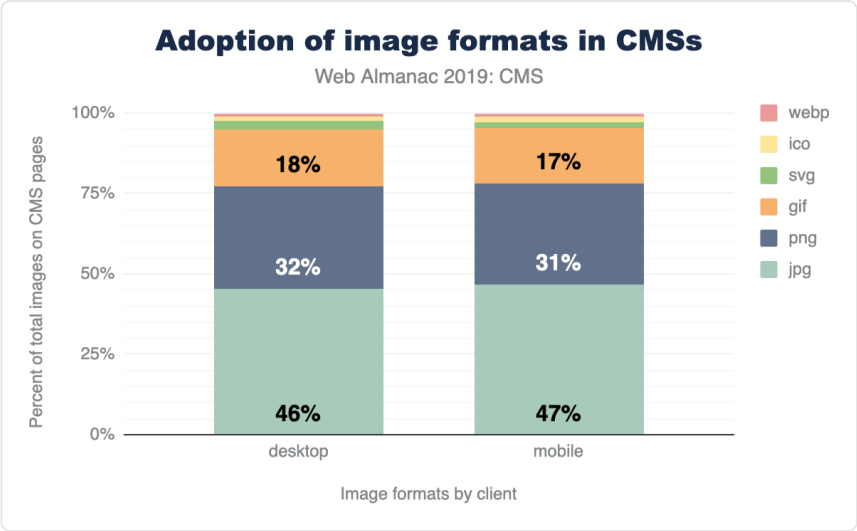


図14.14. CMSページでの画像フォーマットの採用。

おそらく、これらの画像タイプの一般的な使用例を考えると、このようなセグメンテーションは驚くべきものでありません。ロゴやアイコン用のSVGは、JPEGがユビキタスであるのと同様に一般的です。WebPはまだ比較的新しい最適化されたフォーマットであり、ブラウザの普及が進んでいます[329]。これが今後数年の間にCMS空間での使用にどのような影響を与えるかを見るのは興味深いことでしょう。

CMSを搭載したウェブサイトのユーザー体験

ウェブコンテンツ制作者として成功するには、ユーザー体験がすべてです。リソースの使用量やウェブページの構成方法に関するその他の統計などの要因は、サイトを構築する際のベストプラクティスの観点から、サイトの品質を示す重要な指標となります。しかし私たちは最終的に、これらのプラットフォームで生成されたコンテンツを消費したり、利用したりする際にユーザーが実際にどのようにウェブを体験しているのかを明らかにしたいと考えています。

これを実現するために、CrUXデータセットに収録されているいくつかの利用者目線のパフォーマンス指標[330]に向けて分析を行います。これらのメトリクスは、人として私たちが時間をどのように感じるか[331]に何らかの形で関連しています。

329. https://caniuse.com/#search=webp
330. https://developers.google.com/web/fundamentals/performance/user-centric-performance-metrics
331. https://paulbakaus.com/tutorials/performance/the-illusion-of-speed/

持続時間	知覚
< 0.1秒	瞬間
0.5-1秒	即時
2-5秒	放棄されるポイント

図14.15. 人間がどのようにして短い時間を知覚するのか。

0.1秒（100ミリ秒）以内に起こることは、私たちにとっては事実上瞬時に起こっていることです。そして、数秒以上の時間がかかる場合、私たちはそれ以上待たずに生活を続ける可能性が非常に高くなります。これは、ウェブでの持続的な成功を目指すコンテンツ制作者にとって非常に重要なことです。なぜならユーザーを獲得し、魅了し、ユーザーベースを維持したいのであればサイトの読み込み速度がどれだけ速くなければならないかを教えてくれるからです。

このセクションでは、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のかを理解するために、3つの重要な次元を見てみましょう。

- コンテンツの初回ペイント (FCP)
- 入力の推定待ち時間 (FID)
- Lighthouseスコア

コンテンツの初回ペイント

コンテンツの初回ペイント[332] は、ナビゲーション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します。成功したFCPの経験、つまり「速い」と認定される経験とは、ウェブサイトの読み込みが正常に行われていることをユーザーへ保証するため、DOM内の要素がどれだけ速くロードされるかということです。FCPのスコアが良ければ対応するサイトが良いUXを提供していることを保証するものではありませんが、FCPが悪ければ、ほぼ確実にその逆を保証することになります。

332. *https://developers.google.com/web/tools/lighthouse/audits/first-contentful-paint*

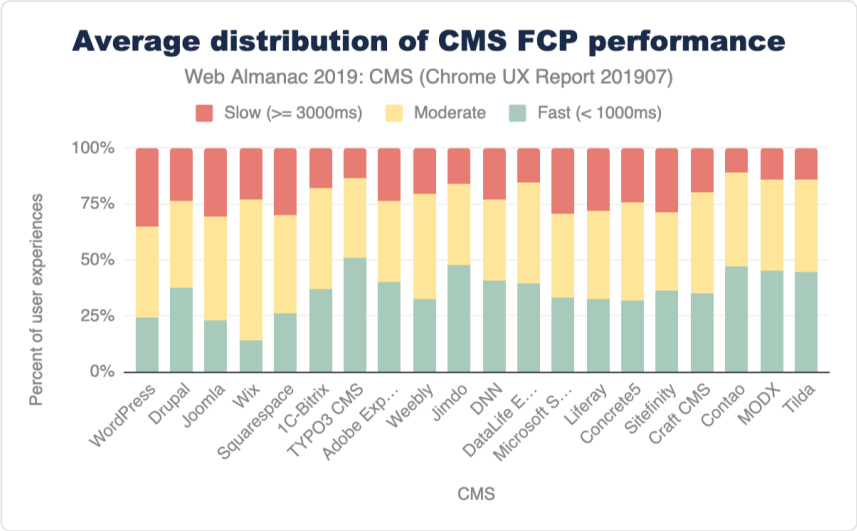


図14.16. CMS全体のFCP経験の平均分布。

CMS	速い (< 1000ms)	中程度	遅い (>= 3000ms)
WordPress	24.33%	40.24%	35.42%
Drupal	37.25%	39.39%	23.35%
Joomla	22.66%	46.48%	30.86%
Wix	14.25%	62.84%	22.91%
Squarespace	26.23%	43.79%	29.98%

図14.17. 上位5つのCMSのFCP経験値の平均分布。

CMSの世界におけるFCPの傾向は、ほとんどが中程度の範囲にあります。CMSプラットフォームがデータベースからコンテンツを照会し、送信し、その後ブラウザでレンダリングする必要があるため、ユーザーが体験する遅延の一因となっている可能性があります。前のセクションで説明したリソース負荷も一役買っている可能性があります。さらに、これらのインスタンスの中には共有ホスティング上にあるものやパフォーマンスが最適化されていない環境もあり、これもブラウザ上での体験に影響を与える可能性があります。

WordPressはモバイルとデスクトップで、中程度のFCP体験と遅いFCP体験を示しています。Wixはクローズドなプラットフォームで中程度のFCP体験が強みです。企業向けオープンソースCMSプラットフォームであるTYPO3は、モバイルとデスクトップの両方で一貫し

て高速な体験を提供しています。TYPO3は、フロントエンドに組み込まれたパフォーマンスとスケーラビリティ機能がウェブサイトの訪問者にプラスの影響を与える可能性があると宣伝しています。

入力の推定待ち時間

入力の推定待ち時間[333] (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、カスタムの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へ応答できるようにな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とは、サイト上でのアクションからの即時フィードバックであり、停滞した体験ではありません。この遅延(痛いところ)は、ユーザーがサイトと対話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。

CMS領域のFIDは一般的に、デスクトップとモバイルの両方で平均的に高速なエクスペリエンスを提供する傾向にある。しかし、注目すべきは、モバイルとデスクトップの体験の間に大きな違いがあることです。

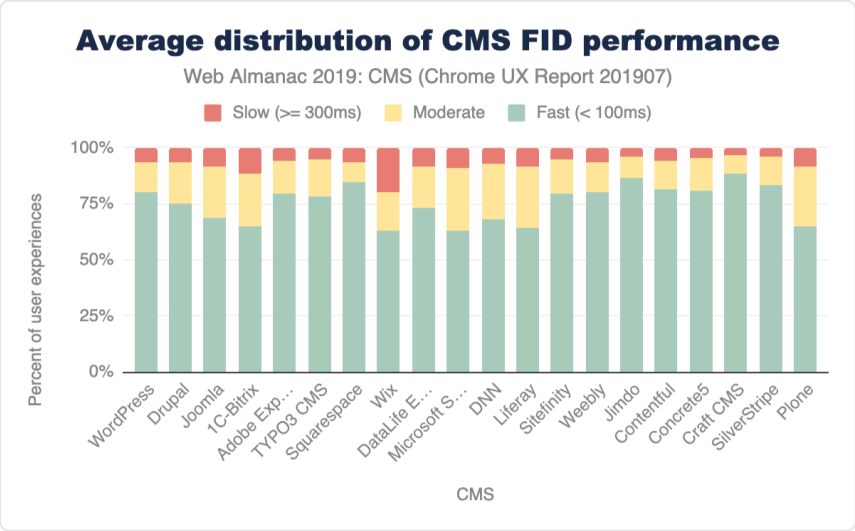


図14.18. CMS全体のFID経験の平均分布。

333. https://developers.google.com/web/updates/2018/05/first-input-delay

CMS	速い (< 100ms)	中程度	遅い (>= 300ms)
WordPress	80.25%	13.55%	6.20%
Drupal	74.88%	18.64%	6.48%
Joomla	68.82%	22.61%	8.57%
Squarespace	84.55%	9.13%	6.31%
Wix	63.06%	16.99%	19.95%

図14.19. 上位5つのCMSのFID経験値の平均分布。

この差はFCPのデータにも見られますが、FIDではパフォーマンスに大きなギャップが見られます。例えば、Joomlaのモバイルとデスクトップの高速FCP体験の差は約12.78%ですが、FIDの体験では27.76%と大きな差があります。モバイルデバイスと接続品質が、ここで見られるパフォーマンスの格差に一役買っている可能性があります。以前に強調したように、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にはわずかな差があります。モバイル（インタラクティブ）体験のための最適化は、これらの結果から明らかになります。

Lighthouseスコア

Lighthouse は、開発者がWebサイトの品質を評価して改善するのに役立つように設計された、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側面の1つは、**パフォーマンス**、**アクセシビリティ**、**プログレッシブなWebアプリ**などの観点からWebサイトの状態を評価するための監査のセットを提供することです。この章の目的のために、2つの特定の監査カテゴリに興味を持っています。PWAとアクセシビリティです。

PWA

プログレッシブウェブアプリ (PWA)という用語は、信頼できる[334]、速い[335]、魅力的[336]とみなされるウェブベースのユーザー体験を指します。Lighthouseは、0（最悪）から1（最高）の間のPWAスコアを返す一連の監査を提供しています。これらの監査は、14の要件をリストアップしたベースラインPWAチェックリスト[337]に基づいています。Lighthouseは、14の要件のうち11の要件について自動監査を実施しています。残りの3つは手動でしかテストできません。11の自動PWA監査はそれぞれ均等に重み付けされているため、それぞれがPWAスコア

334. https://developers.google.com/web/progressive-web-apps#reliable
335. https://developers.google.com/web/progressive-web-apps#fast
336. https://developers.google.com/web/progressive-web-apps#engaging
337. https://developers.google.com/web/progressive-web-apps/checklist#baseline

に約9ポイント寄与します。

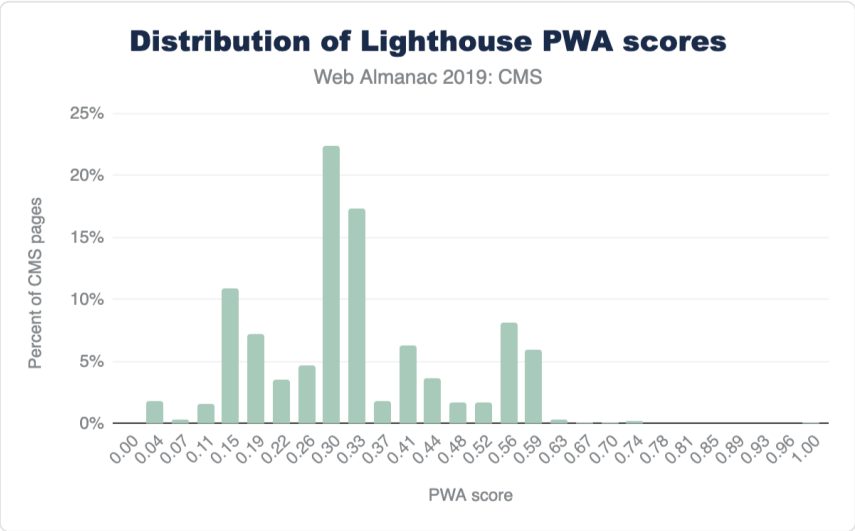


図14.20. CMSページのLighthouse PWAカテゴリスコアの分布。

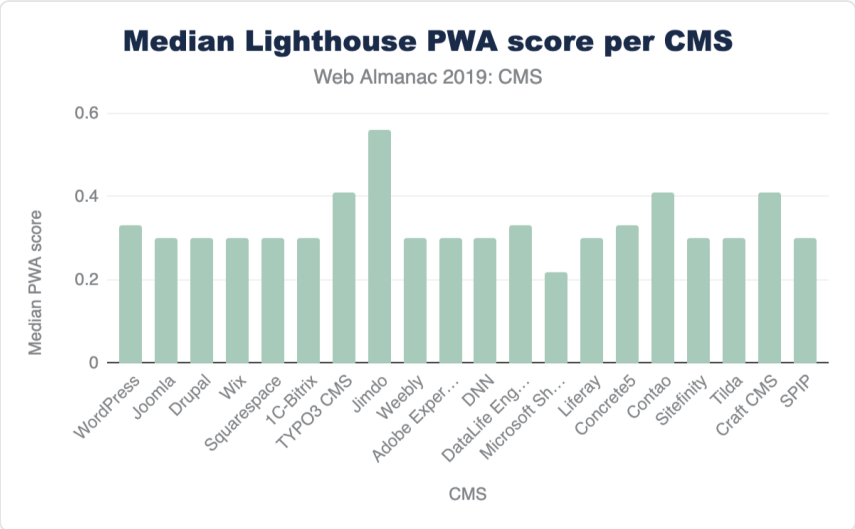


図14.21. CMSごとの灯台PWAカテゴリスコアの中央値。

アクセシビリティ

アクセシブルなウェブサイトとは、障害者が利用できるように設計・開発されたサイトのことです。Lighthouseは、一連のアクセシビリティ監査を提供し、それらすべての監査の加重平均を返します（各監査の加重方法の完全なリストについては、スコアリングの詳細[338]を参照してください）。

各アクセシビリティ監査は合格か、不合格かですが他のLighthouseの監査とは異なり、アクセシビリティ監査に部分的に合格してもページはポイントをもらえません。例えば、いくつかの要素がスクリーンリーダーに優しい名前を持っていて他の要素がそうでない場合、そのページは*screenreader-friendly-names*監査で0点を獲得します。

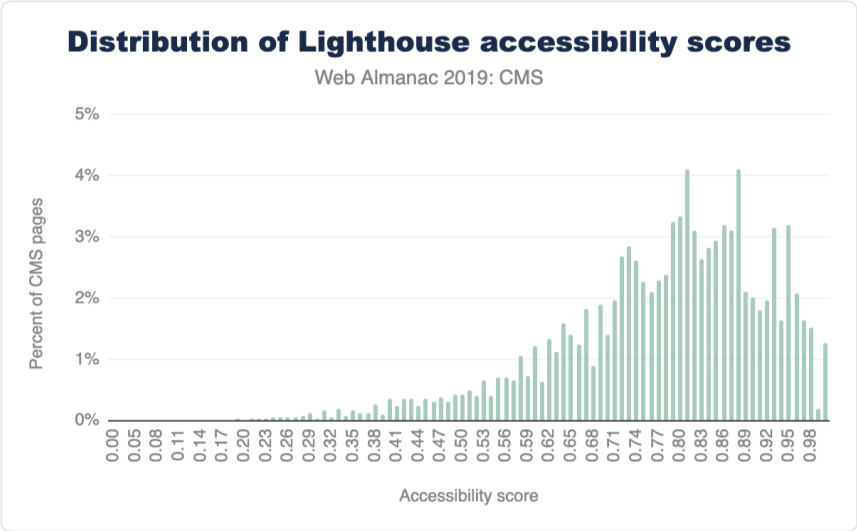


図14.22. CMSページのLighthouseアクセシビリティカテゴリスコアの分布。

338. *https://docs.google.com/spreadsheets/d/1Cxzhy5ecqJCucdf1M0iOzM8mlxNc7mmx107o5nj38Eo/edit#gid=1567011065*

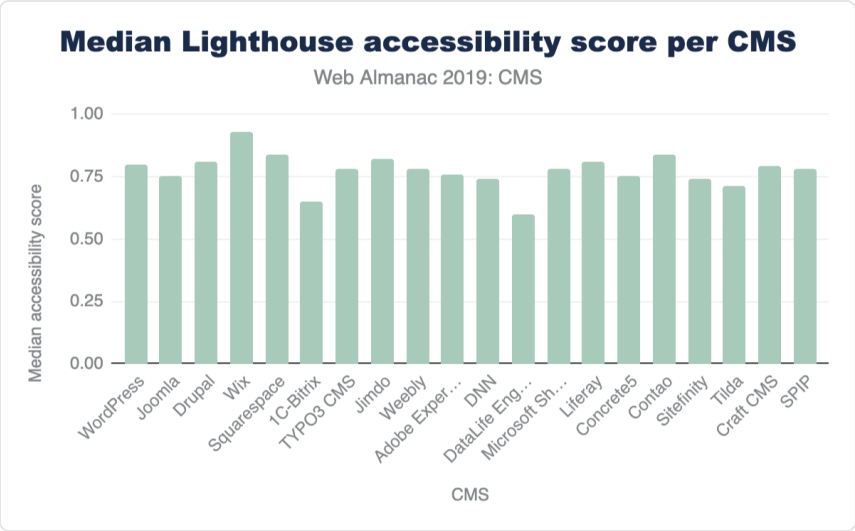


図14.23. CMSごとのLighthouseアクセシビリティカテゴリスコアの中央値。

現在、モバイルCMSのホームページで100%のパーフェクトスコアを獲得しているのは1.27%しかありません。上位のCMSの中では、Wixがモバイルページのアクセシビリティスコアの中央値が最も高く、トップに立っています。全体的に見て、これらの数字は、私たちの人口のかなりの部分がアクセスできないウェブサイトはどれだけ多いか（CMSによって駆動されているウェブのどれだけの部分か）を考えると悲惨なものとなります。デジタル体験が私たちの生活の多くの側面に影響を与えるのと同様に、この数字は私たちに *最初からアクセシブルなウェブ体験を構築すること* を奨励し、ウェブを包括的な空間にする作業を継続するための指令であるべきです。

CMSイノベーション

ここまでCMSエコシステムの現状をスナップショットで紹介してきましたが、この分野は進化しています。パフォーマンスとユーザー体験の欠点に対処するため、実験的なフレームワークがCMSインフラストラクチャに統合されているのを目の当たりにしています。React.jsやGatsby.js、Next.jsなどの派生フレームワーク、Vue.jsの派生フレームワークであるNuxt.jsなどのライブラリやフレームワークが少しずつ採用されてきています。

CMS	React	Nuxt.js, React	Nuxt.js	Next.js, React	Gatsby, React
WordPress	131,507		21	18	
Wix	50,247				
Joomla	3,457				
Drupal	2,940		8	15	1
DataLife Engine	1,137				
Adobe Experience Manager	723			7	
Contentful	492	7	114	909	394
Squarespace	385				
1C-Bitrix	340				
TYPO3 CMS	265			1	
Weebly	263		1		
Jimdo	248				2
PrestaShop	223		1		
SDL Tridion	152				
Craft CMS	123				

図14.24. CMSごとのReactとコンパニオンフレームワークの採用率（モバイルサイト数）。

また、ホスティングプロバイダーや代理店が企業の顧客に焦点を当てた戦略のためのツールボックスとして、CMSやその他の統合技術を使用した総合的なソリューションとしてデジタルエクスペリエンスプラットフォーム（DXP）を提供しているのも見受けられます。これらのイノベーションは、ユーザー（とそのエンドユーザー）がこれらのプラットフォームのコンテンツを作成し、消費する際に最高のUXを得ることを可能にするターンキーのCMSベースのソリューションを作成するための努力を示しています。目的は、デフォルトでの優れたパフォーマンス、豊富な機能、優れたホスティング環境です。

結論

CMS空間は最も重要な意味を持っています。これらのアプリケーションが力を発揮するウェ

ブの大部分と様々なデバイスや接続でページを作成し、それに遭遇するユーザーの数は、些細なことであってはなりません。この章やこのWeb Almanacに掲載されている他の章が、この空間をより良いものにするためのより多くの研究と技術革新を促してくれることを願っています。深い調査を行うことで、これらのプラットフォームがウェブ全体に提供する強み、弱み、機会について、より良いコンテキストを提供できます。コンテンツ管理システムは、オープン・ウェブの完全性を維持するために影響を与えることができます。コンテンツ管理システムを前進させていきましょう！

著者

Renee Johnson

@reneesoffice ernee https://reneesvirtualoffice.com

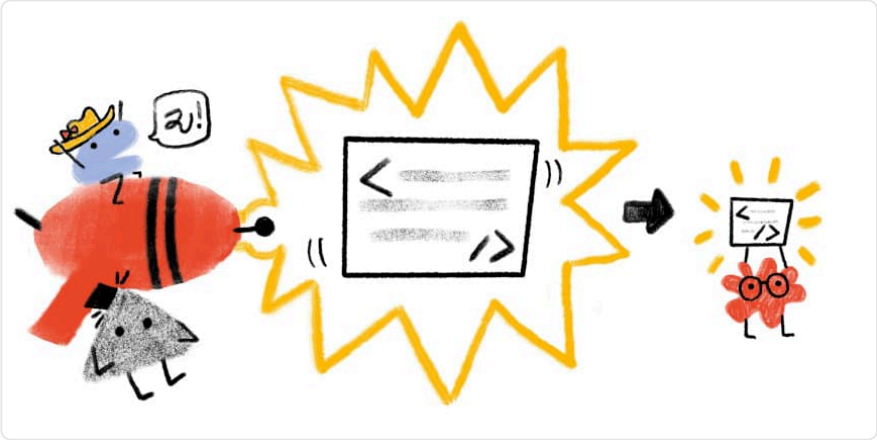
Renee Johnsonは、ウェブと製品のコンサルタントであり、WordPressの愛好家であり、WordCampの主催者でありボランティアでもあります。現在は、GoogleのContent Man agement System Developer Relationsチームでプロダクトサポートスペシャリストとして働いています。

Alberto Medina

@iAlbMedina amedina

Alberto MedinaはGoogleのWeb Content Ecosystemsチームの開発提唱者で、Ampのような先進的な技術や最新のWeb APIの利用を通じたWeb上の質の高いコンテンツの普及促進に焦点を当てています。Albertoの仕事は現在、コンテンツ管理システムに重点を置いており、CMS開発者関係と呼ばれるコンテンツエコシステムの分野をリードしている。

部 IV 章 15
圧縮



Paul Calvano によって書かれた。
David Fox と Yoav Weiss によってレビュ。
Paul Calvano による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

序章

HTTP圧縮は、元の表現よりも少ないビットを使用して情報をエンコードできる技術です。Webコンテンツの配信に使用すると、Webサーバーはクライアントに送信されるデータ量を削減できます。これにより、クライアントの利用可能な帯域幅の効率が向上し、ページの重さが軽減され、Webパフォーマンスが向上します。

圧縮アルゴリズムは、多くの場合、非可逆または可逆に分類されます。

- 非可逆圧縮アルゴリズムが使用される場合、プロセスは不可逆的であり、元のファイルを圧縮解除しても復元できません。これは一般に、一部のデータを失ってもリソースに重大な影響を与えない画像やビデオコンテンツなどのメディアリソースを圧縮するために使用されます。
- ロスレス圧縮は完全に可逆的なプロセスであり、HTML、JavaScript、CSSなどの

テキストベースのリソースを圧縮するために一般的に使用されます。

この章では、テキストベースのコンテンツがWeb上でどのように圧縮されるかを検討します。非テキストベースのコンテンツの分析は、メディアの章の一部を形成します。

HTTP圧縮の仕組み

クライアントがHTTPリクエストを作成する場合、多くの場合、デコード可能な圧縮アルゴリズムを示す`Accept-Encoding`ヘッダーが含まれます。サーバーは、示されたエンコードのいずれかを選択してサポートし、圧縮されたレスポンスを提供できます。圧縮されたレスポンスには`Content-Encoding`ヘッダーが含まれるため、クライアントはどの圧縮が使用されたかを認識できます。また、提供されるリソースのMIME[339]タイプを示すために、`Content-Type`ヘッダーがよく使用されます。

以下の例では、クライアントはGzip、Brotli、およびDeflate圧縮のサポートを示してます。サーバーは、`text/html`ドキュメントを含むGzip圧縮された応答を返すことにしました。

```
> GET / HTTP/1.1
> Host: httparchive.org
> Accept-Encoding: gzip, deflate, br

< HTTP/1.1 200
< Content-type: text/html; charset=utf-8
< Content-encoding: gzip
```

HTTP Archiveには、530万のWebサイトの測定値が含まれており、各サイトには少なくとも1つの圧縮テキストリソースがホームページにロードされています。さらに、リソースはWebサイトの81%のプライマリドメインで圧縮されました。

圧縮アルゴリズム

IANAは、`Accept-Encoding`および`Content-Encoding`ヘッダーで使用できる有効なHTTPコンテンツエンコーディングのリスト[340]を保持しています。これらには、`gzip`、`deflate`、`br`（Brotli）などが含まれます。これらのアルゴリズムの簡単な説明を以下に

339. https://developer.mozilla.org/docs/Web/HTTP/Basics_of_HTTP/MIME_types
340. https://www.iana.org/assignments/http-parameters/http-parameters.xml#content-coding

示します。

- Gzip[341]は、LZ77[342]およびハフマンコーディング[343]圧縮技術を使用しており、Web自体よりも古い。もともと1992年にUNIX `gzip` プログラム用として開発されました。HTTP/ 1.1以降、Web配信の実装が存在し、ほとんどのブラウザーとクライアントがそれをサポートしています。
- Deflate[344]はGzipと同じアルゴリズムを使用しますが、コンテナは異なります。一部のサーバーおよびブラウザとの互換性の問題のため、Webでの使用は広く採用されていません。
- Brotli[345]は、Googleが発明[346]した新しい圧縮アルゴリズムです。 LZ77アルゴリズムの最新のバリアント、ハフマンコーディング、および2次コンテキストモデリングの組み合わせを使用します。 Brotliを介した圧縮はGzipと比較して計算コストが高くなりますが、アルゴリズムはGzip圧縮よりもファイルを15〜25%[347]削減できます。 Brotliは2015年にWebコンテンツの圧縮に初めて使用され、すべての最新のWebブラウザーでサポートされています[348]。

HTTPレスポンスの約38%はテキストベースの圧縮で配信されます。これは驚くべき統計のように思えるかもしれませんが、データセット内のすべてのHTTP要求に基づいていることに留意してください。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ません。次の表は、各コンテンツエンコーディングで処理されるリクエストの割合をまとめたものです。

341. *https://tools.ietf.org/html/rfc1952*
342. *https://ja.wikipedia.org/wiki/LZ77*
343. *https://ja.wikipedia.org/wiki/%E3%83%8F%E3%83%95%E3%83%9E%E3%83%B3%E7%AC%A6%E5%8F%B7*
344. *https://tools.ietf.org/html/rfc1951*
345. *https://tools.ietf.org/html/rfc7932*
346. *https://github.com/google/brotli*
347. *https://cran.r-project.org/web/packages/brotli/vignettes/brotli-2015-09-22.pdf*
348. *https://caniuse.com/#feat=brotli*

	リクエストの割合		リクエスト	
コンテンツエンコーディング	デスクトップ	モバイル	デスクトップ	モバイル
テキスト圧縮なし	62.87%	61.47%	260,245,106	285,158,644
`gzip`	29.66%	30.95%	122,789,094	143,549,122
`br`	7.43%	7.55%	30,750,681	35,012,368
`deflate`	0.02%	0.02%	68,802	70,679
Other / Invalid	0.02%	0.01%	67,527	68,352
`identity`	0.000709%	0.000563%	2,935	2,611
`x-gzip`	0.000193%	0.000179%	800	829
`compress`	0.000008%	0.000007%	33	32
`x-compress`	0.000002%	0.000006%	8	29

図15.1. 圧縮アルゴリズムの採用。

圧縮されて提供されるリソースの大半は、Gzip（80%）またはBrotli（20%）のいずれかを使用しています。他の圧縮アルゴリズムはあまり使用されません。

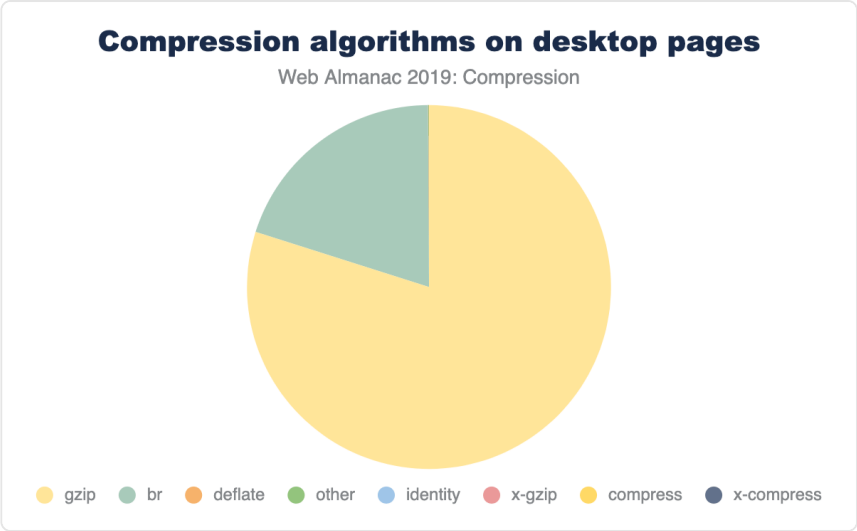


図15.2. デスクトップページでの圧縮アルゴリズムの採用。

さらに「none」「UTF-8」「base64」「text」など、無効な`Content-Encoding`を返す67Kのリクエストがあります。これらのリソースは圧縮されていない状態で提供される可能性があります。

HTTP Archiveによって収集された診断から圧縮レベルを判断することはできませんが、コンテンツを圧縮するためのベストプラクティスは次のとおりです。

- 少なくとも、テキストベースのアセットに対してGzip圧縮レベル6を有効にします。これは、計算コストと圧縮率の間の公平なトレードオフを提供し、多くのWebサーバーのデフォルト[349]にもかかわらず、Nginxは依然として低すぎることが多いレベル1のままです[350]。
- Brotliおよびprecompressリソースをサポートできる場合は、Brotliレベル11に圧縮します。これはGzipよりも計算コストが高くなります。したがって、遅延を避けるためには、事前圧縮が絶対に必要です。
- Brotliをサポートでき、事前圧縮できない場合は、Brotliレベル5に圧縮します。このレベルでは、Gzipと比較してペイロードが小さくなり、同様の計算オーバーヘッドが発生します。

349. https://paulcalvano.com/index.php/2018/07/25/brotli-compression-how-much-will-it-reduce-your-content/
350. http://nginx.org/en/docs/http/ngx_http_gzip_module.html#gzip_comp_level

どの種類のコンテンツを圧縮していますか？

ほとんどのテキストベースのリソース（HTML、CSS、JavaScriptなど）は、GzipまたはBrotli圧縮の恩恵を受けることができます。ただし、多くの場合、これらの圧縮技術をバイナリリソースで使用する必要はありません。画像、ビデオ、一部のWebフォントなどが既に圧縮されているため。

次のグラフでは、上位25のコンテンツタイプが、リクエストの相対数を表すボックスサイズで表示されています。各ボックスの色は、これらのリソースのうちどれだけ圧縮されて提供されたかを表します。ほとんどのメディアコンテンツはオレンジ色で網掛けされていますが、これはGzipとBrotliにはほとんどまたはまったく利点がないためです。テキストコンテンツのほとんどは、それらが圧縮されていることを示すために青色で網掛けされています。ただし、一部のコンテンツタイプの水色の網掛けは、他のコンテンツタイプほど一貫して圧縮されていないことを示しています。

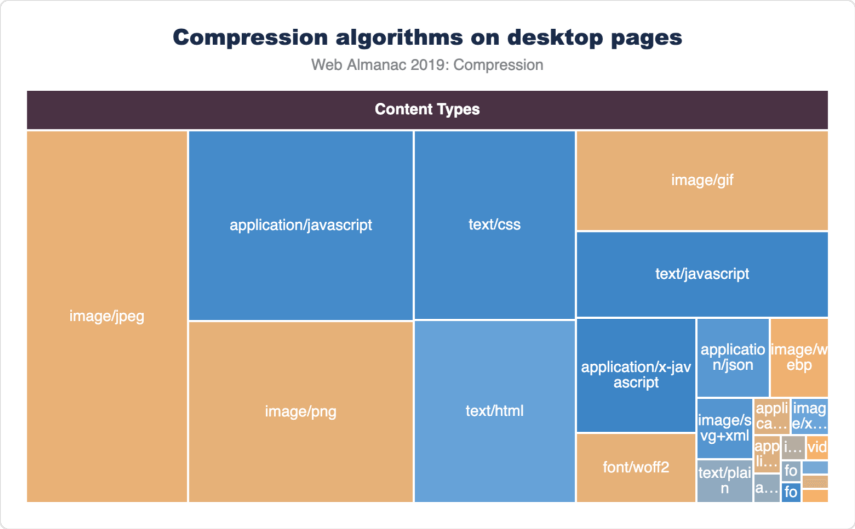


図15.3. 上位25の圧縮コンテンツタイプ。

最も人気のある8つのコンテンツタイプを除外すると、これらのコンテンツタイプの残りの圧縮統計をより明確に確認できます。

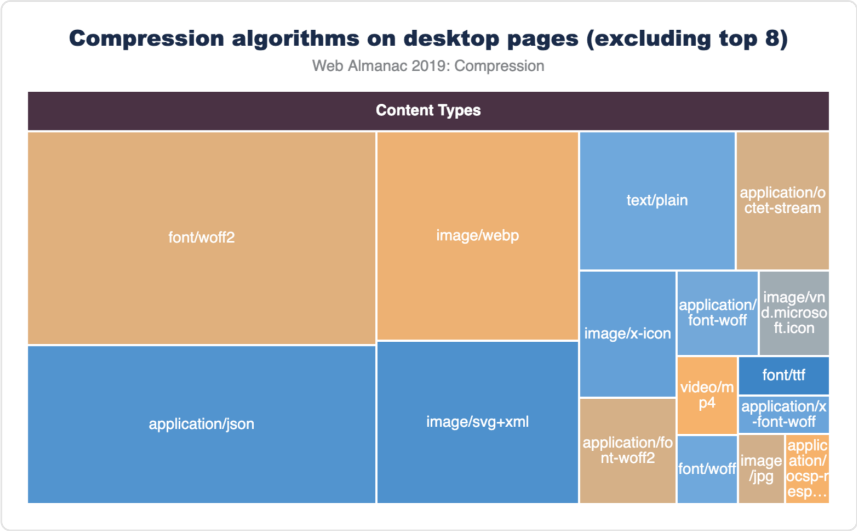


図15.4. トップ8を除く圧縮コンテンツタイプ

`application/json`および`image/svg+xml`コンテンツタイプは、65%未満の時間で圧縮されます。

カスタムWebフォントのほとんどは、すでに圧縮形式になっているため、圧縮せずに提供されます。ただし、`font/ttf`は圧縮可能ですが、TTFフォント要求の84%のみが圧縮で提供されているため、ここにはまだ改善の余地があります。

以下のグラフは、各コンテンツタイプに使用される圧縮技術の内訳を示しています。上位3つのコンテンツタイプを見ると、デスクトップとモバイルの両方で、最も頻繁に要求されるコンテンツタイプの圧縮に大きなギャップがあります。`text/html`の56%と`application/javascript`および`text/css`リソースの18%は圧縮されていません。これにより、パフォーマンスが大幅に向上します。

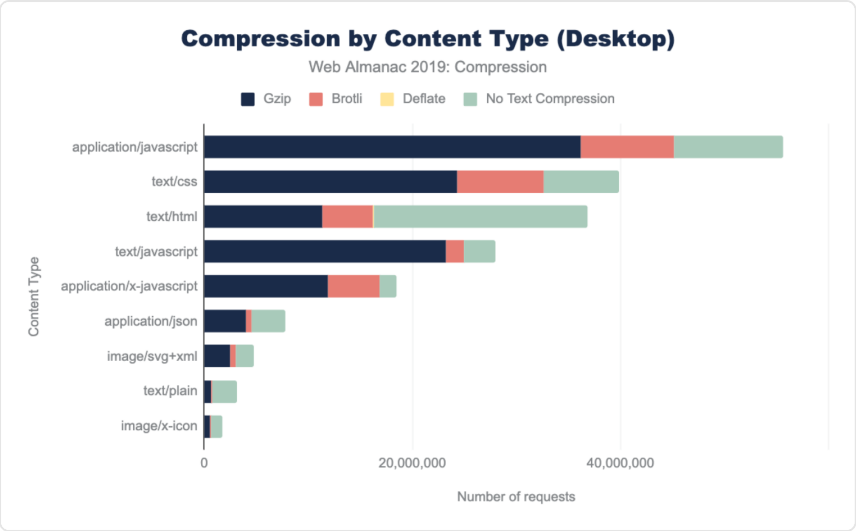


図15.5. デスクトップのコンテンツタイプによる圧縮。

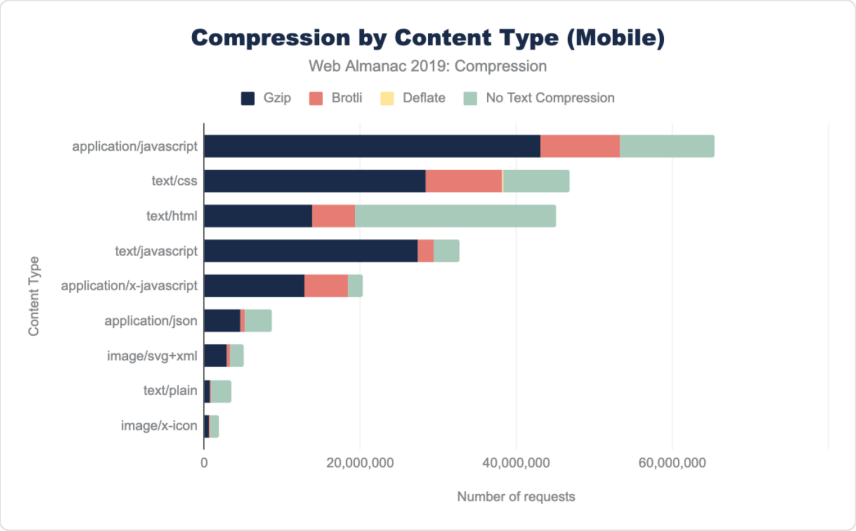


図15.6. モバイルのコンテンツタイプによる圧縮。

圧縮率が最も低いコンテンツタイプには、`application/json`、`text/xml`、および`text/plain`が含まれます。これらのリソースは通常、XHRリクエストに使用され、Webアプリケーションが豊かな体験を創造するために使用できるデータを提供します。それらを圧縮すると、ユーザー体験は向上する可能性があります。`image/svg+xml`や`image/x-icon`などのベクターグラフィックスは、テキストベースと見なされることはあまりありま

せんが、これらを使用するサイトは圧縮の恩恵を受けるでしょう。

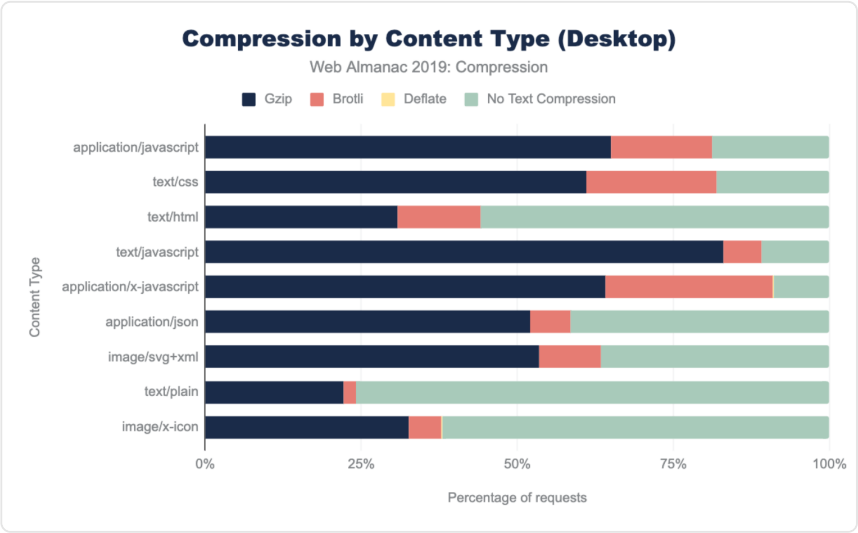


図15.7. コンテンツタイプによる圧縮のデスクトップ割合。

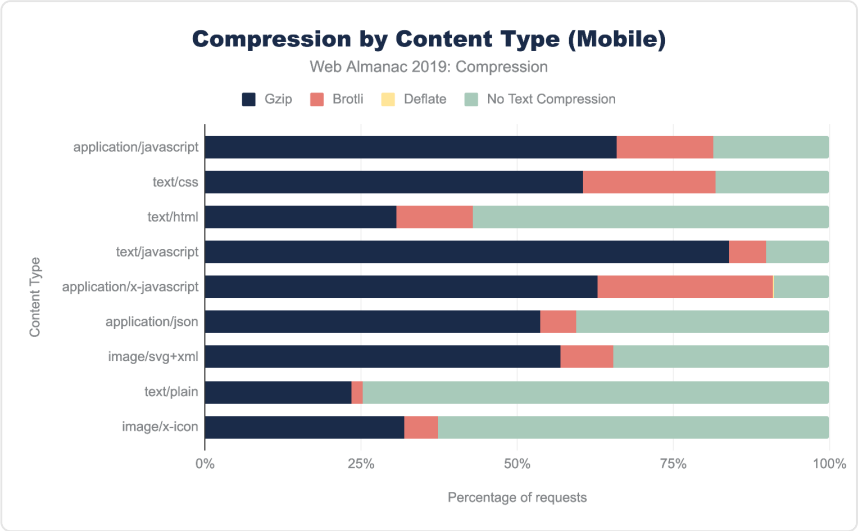


図15.8. コンテンツタイプによる圧縮のモバイル割合。

すべてのコンテンツタイプで、Gzipは最も一般的な圧縮アルゴリズムです。新しいBrotli圧縮はあまり頻繁に使用されず、最も多く表示されるコンテンツタイプは`application/javascript`、`text/css`、`application/x-javascript`です。これは、CDNが通過す

るトラフィックにBrotli圧縮を自動的に適用することの原因である可能性があります。

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、サードパーティとパフォーマンスへの影響について学びました。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であることがわかります。

さらに、サードパーティのコンテンツの場合、Brotli圧縮の割合が高くなります。これは、GoogleやFacebookなど、通常Brotliをサポートす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原因である可能性と考えられます。

	デスクトップ		モバイル	
コンテンツエンコーディング	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	66.23%	59.28%	64.54%	58.26%
`gzip`	29.33%	30.20%	30.87%	31.22%
`br`	4.41%	10.49%	4.56%	10.49%
`deflate`	0.02%	0.01%	0.02%	0.01%
Other / Invalid	0.01%	0.02%	0.01%	0.02%

図15.9. デバイスタイプ別のファーストパーティとサードパーティの圧縮。

圧縮の機会を見分ける

GoogleのLighthouse[351]ツールを使用すると、ユーザーはWeb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[352]は、サイトが追加のテキストベースの圧縮の恩恵を受けることができるかどうかを評価します。これは、リソースを圧縮し、オブジェクトのサイズを少なくとも10%と1,400バイト削減できるかどうかを評価することでこれを行います。スコアに応じて、圧縮可能な特定のリソースのリストとともに、結果に圧縮の推奨事項を表示する場合があります。

351. https://developers.google.com/web/tools/lighthouse
352. https://developers.google.com/web/tools/lighthouse/audits/text-compression

Enable text compression — 0.62 s

Text-based resources should be served with compression (gzip, deflate or brotli) to minimize total network bytes. Learn more.

Show 3rd-party resources (6)

URL	Size	Potential Savings
…comscore/streamsense.5.2.0.160629.min.js	91 KB	73 KB
/AdServer/PugMaster?kdntuid=…	5 KB	4 KB
/AdServer/PugMaster?kdntuid=…	6 KB	4 KB
…t_ads/ads?bust=…	5 KB	2 KB
/AdServer/PugMaster?kdntuid=…	2 KB	1 KB
/AdServer/Pug?vcode=…	2 KB	1 KB

図15.10. Lighthouse圧縮の提案

各モバイルページに対してHTTP ArchiveはLighthouse監査を実行するため、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あるかどうかを知ることができます。全体として、ウェブサイトの62%がこの監査に合格しており、ウェブサイトのほぼ23%が40を下回っています。これは、120万を超えるウェブサイトが追加のテキストベースの圧縮を有効にすることを意味します。

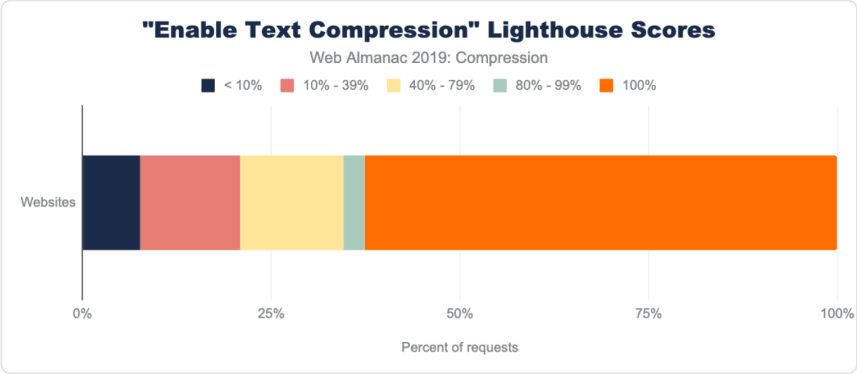


図15.11. Lighthouseの「テキスト圧縮を有効にする」監査スコア。

Lighthouseは、テキストベースの圧縮を有効にすることで、保存できるバイト数も示します。テキスト圧縮の恩恵を受ける可能性のあるサイトのうち、82%がページの重さを最大1MB削減できます！

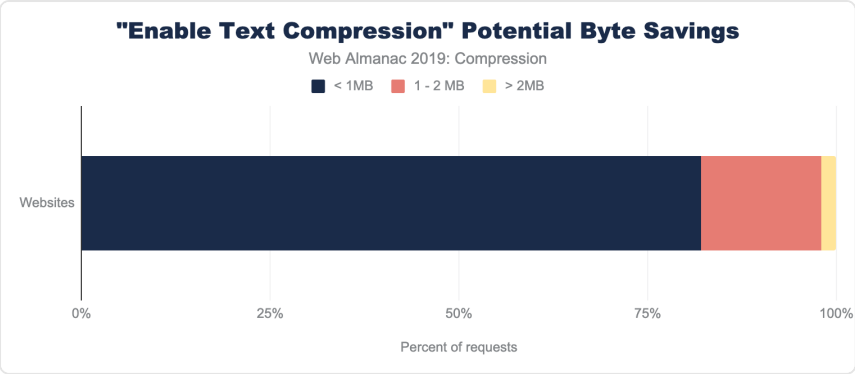


図15.12. 潜在的なバイト削減を監査するLighthouseの「テキスト圧縮を有効にする」

結論

HTTP圧縮は、Webコンテンツのサイズを削減するために広く使用されている非常に貴重な機能です。 GzipとBrotliの両方の圧縮が使用される主要なアルゴリズムであり、圧縮されたコンテンツの量はコンテンツの種類によって異なります。 Lighthouseなどのツールは、コンテンツを圧縮する機会を発見するのに役立ちます。

多くのサイトがHTTP圧縮をうまく利用していますが、特にWebが構築されている`text/html`形式については、まだ改善の余地があります！　同様に、`font/ttf`、`application/json`、`text/xml`、`text/plain`、`image/svg+xml`、`image/x-icon`のようなあまり理解されていないテキスト形式は、多くのWebサイトで見落とされる余分な構成を取る場合があります。

Webサイトは広くサポートされており、簡単に実装で処理のオーバーヘッドが低いため、少なくともすべてのテキストベースのリソースにGzip圧縮を使用する必要があります。 Brotli圧縮を使用するとさらに節約できますが、リソースを事前に圧縮できるかどうかに基づいて圧縮レベルを慎重に選択する必要があります。

著者

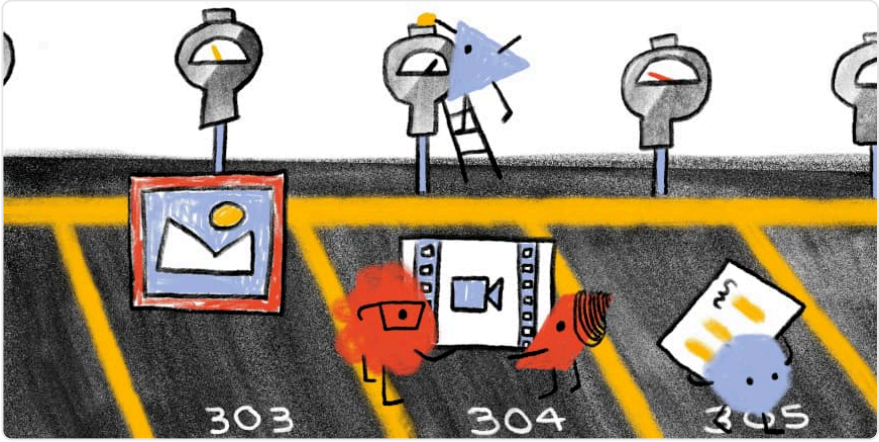
Paul Calvano

@paulcalvano　paulcalvano　https://paulcalvano.com

Paul Calvanoは、アカマイ[353]のウェブパフォーマンス・アーキテクトで、ウェブサイトのパフォーマンス向上を支援しています。また、HTTP Archiveプロジェクトの共同管理者でもあります。@paulcalvano でツイートしたり、http://paulcalvano.com でブログを書いたり、https://discuss.httparchive.org でHTTP Archiveの研究を共有したりしています。

353. https://www.akamai.com/

部 IV 章 16
キャッシング



Paul Calvano によって書かれた。
David Fox と Brian Kardell によってレビュ。
Paul Calvano と David Fox による分析。
Barry Pollard 編集。
Sakae Kotaro によって翻訳された。

導入

キャッシングは、以前にダウンロードしたコンテンツの再利用を可能にする手法です。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により、パフォーマンスが大幅に向上します。また、Web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削減することで、アプリケーションの拡張にも役立ちます。「最速のリクエストはあなたがする必要のないものです」という古い格言があり、キャッシュはリクエストを行わなくて済むようにするための重要な方法の1つです。

Webコンテンツのキャッシュには、3つの基本原則があります。可能な限りキャッシュする、できる限りキャッシュする、エンドユーザーのできるだけ近くでキャッシュする。

可能な限りキャッシュする。 キャッシュできる量を検討する場合、レスポンスが静的か動的かを理解することが重要です。静的なレスポンスとして提供される要求は、リソースとそれを要求するユーザーとの間に1対多の関係があるため、通常はキャッシュ可能です。動的に

生成されたコンテンツはより微妙である可能性があり、慎重に検討する必要があります。

できるだけ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、キャッシュされるコンテンツの機密性に大きく依存します。バージョン管理されたJavaScriptリソースは非常に長い時間キャッシュされる可能性がありますが、バージョン管理されていないリソースはユーザーが最新バージョンを取得できるように、より短いキャッシュ期間を必要とする場合があります。

エンドユーザーのできるだけ近くでキャッシュする。エンドユーザーの近くでコンテンツをキャッシュすると、待ち時間がなくなり、ダウンロード時間が短縮されます。たとえば、リソースがエンドユーザーのブラウザにキャッシュされている場合、リクエストはネットワークに送信されず、ダウンロード時間はマシンのI/Oと同じくらい高速です。初めての訪問者、またはキャッシュにエントリがない訪問者の場合、通常、キャッシュされたリソースが返される場所はCDNになります。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かCDNからリソースをフェッチする方が高速です。

通常、Webアーキテクチャには複数のキャッシュ層[354]が含まれます。たとえば、HTTPリクエストは次の場所にキャッシュされる可能性があります。

- エンドユーザーのブラウザ
- ユーザーのブラウザーのService Workerキャッシュ
- 共有ゲートウェイ
- エンドユーザーに近い側でキャッシュする機能を提供するCDN
- バックエンドの仕事長を削減するための、アプリケーションの前のキャッシングプロキシ
- アプリケーション層とデータベース層

この章では、Webブラウザー内でリソースがキャッシュされる方法について見ていきましょう。

HTTPキャッシングの概要

HTTPクライアントがリソースをキャッシュするには、2つの情報を理解する必要があります。

- 「これをキャッシュできる期間はどれくらいですか？」

354. *https://blog.yoav.ws/tale-of-four-caches/*

- 「コンテンツがまだ新しいことを検証するにはどうすればよいですか？」

Webブラウザーがクライアントにレスポンスを送信するとき、通常リソースにキャッシュ可能か、キャッシュする期間、リソースの古さを示すヘッダーが含まれます。 RFC 7234は、これをセクション4.2（新しさ）[355]および4.3（検証）[356]でより詳細にカバーしています。

通常、有効期間を伝えるために使用されるHTTPレスポンスヘッダーは次のとおりです。

- `Cache-Control` キャッシュの生存期間（つまり、有効期間）を設定できます。
- `Expires` 有効期限の日付または時刻を提供します（つまり、期限切れになるとき）。

`Cache-Control` 両方が存在する場合に優先されます。これらについては、以下で詳しく説明します。

キャッシュ内に保存された応答を検証するためのHTTPレスポンスヘッダー、つまりサーバー側で比較するため、条件付き要求を提供するHTTPレスポンスヘッダーは次のとおりです。

- `Last-Modified` オブジェクトが最後に変更された日時を示します。
- エンティティタグ（`ETag`）コンテンツの一意の識別子を提供します。

`ETag` 両方が存在する場合に優先されます。これらについては、以下で詳しく説明します。

以下の例には、HTTP Archiveのmain.jsファイルからのリクエスト/レスポンスヘッダーの抜粋が含まれています。これらのヘッダーは、リソースを43,200秒（12時間）キャッシュでき、最後は2か月以上前に変更されたことを示します（`Last-Modified` ヘッダーと `Date` ヘッダーの違い）。

```
> GET /static/js/main.js HTTP/1.1
> Host: httparchive.org
> User-agent: curl/7.54.0
> Accept: */*

< HTTP/1.1 200
< Date: Sun, 13 Oct 2019 19:36:57 GMT
```

355. https://tools.ietf.org/html/rfc7234#section-4.2
356. https://tools.ietf.org/html/rfc7234#section-4.3

```
< Content-Type: application/javascript; charset=utf-8
< Content-Length: 3052
< Vary: Accept-Encoding
< Server: gunicorn/19.7.1
< Last-Modified: Sun, 25 Aug 2019 16:00:30 GMT
< Cache-Control: public, max-age=43200
< Expires: Mon, 14 Oct 2019 07:36:57 GMT
< ETag: "1566748830.0-3052-3932359948"
```

RedBot.org[357]というツールにURLを入力すると、レスポンスのヘッダーを元としたキャッシュ方法の詳細な説明が表示できます。たとえば、上記のURLのテスト[358]は次のような内容を出力します。

図16.1. ロボットからの `Cache-Control` 情報

レスポンスにキャッシュヘッダーが存在しない場合、クライアントはレスポンスをヒューリスティクスにキャッシュできます[359]。ほとんどのクライアントは、RFCの推奨ヒューリスティックバリエーションを実装します。これは、`Last-Modified` から経過した時間の10%です。ただし、レスポンスを無期限にキャッシュする可能性もあります。そのため、特定のキャッシュルールを設定して、キャッシュ可能性を確実に制御することが重要です。

レスポンスの72%は `Cache-Control` ヘッダーで提供され、レスポンスの56%は `Expires` ヘッダーで提供されます。ただし、レスポンスの27%はどちらのヘッダーも使用していないため、ヒューリスティックキャッシュの対象となります。これは、デスクトップとモバイルサイトの両方で一貫しています。

357. https://redbot.org/
358. https://redbot.org/?uri=https%3A%2F%2Fhttparchive.org%2Fstatic%2Fjs%2Fmain.js
359. https://paulcalvano.com/index.php/2018/03/14/http-heuristic-caching-missing-cache-control-and-expires-headers-explained/

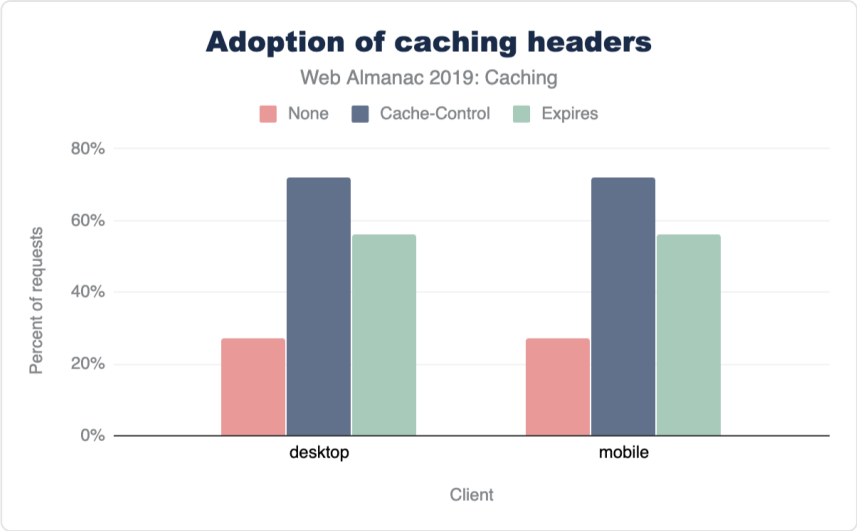


図16.2. HTTP `Cache-Control` および `Expires` ヘッダーの存在

キャッシュするコンテンツの種類は何ですか？

キャッシュ可能なリソースは、クライアントによって一定期間保存され、後続のリクエストで再利用できます。すべてのHTTPリクエスト全体で、レスポンスの80%はキャッシュ可能と見なされます。つまり、キャッシュがそれらを格納することを許可されています。

- 要求の6%のTime To Time（TTL）は0秒で、キャッシュされたエントリはすぐに無効になります。
- 27%は`Cache-Control`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ます。
- 47%は0秒以上キャッシュされます。

残りのレスポンスは、ブラウザーのキャッシュに保存できません。

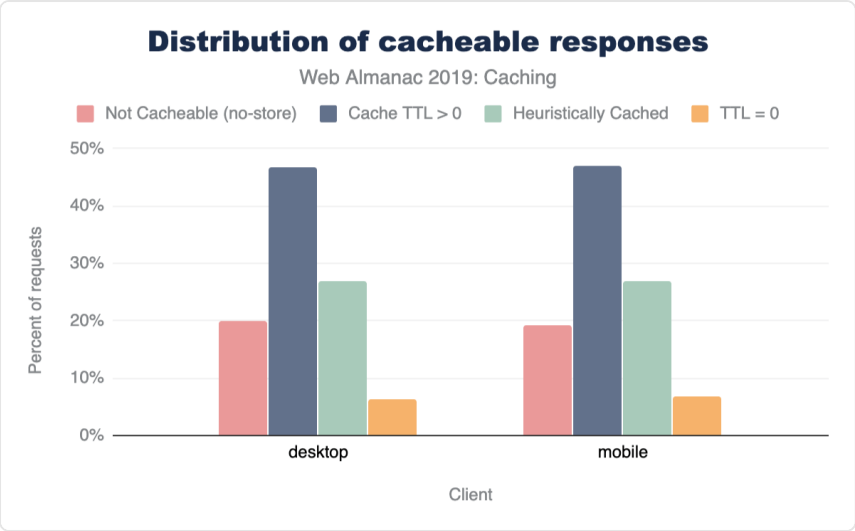


図16.3. キャッシュ可能なレスポンスの分布。

次の表は、デスクトップリクエストのキャッシュTTL値をタイプ別に詳細に示しています。ほとんどのコンテンツタイプはキャッシュされますが、CSSリソースは高いTTLで一貫してキャッシュされるようです。

デスクトップキャッシュTTLパーセンタイル（時間）					
	10	25	50	75	90
Audio	12	24	720	8,760	8,760
CSS	720	8,760	8,760	8,760	8,760
Font	< 1	3	336	8,760	87,600
HTML	< 1	168	720	8,760	8,766
Image	< 1	1	28	48	8,760
Other	< 1	2	336	8,760	8,760
Script	< 1	< 1	1	6	720
Text	21	336	7,902	8,357	8,740
Video	< 1	4	24	24	336
XML	< 1	< 1	< 1	< 1	< 1

図16.4. リソースタイプ別のデスクトップキャッシュTTLパーセンタイル。

TTLの中央値のほとんどは高いですが、低いパーセンタイルは、見逃されたキャッシングの機会の一部を強調しています。たとえば画像のTTLの中央値は28時間ですが、25パーセンタイルは1〜2時間であり、10パーセンタイルはキャッシュ可能な画像コンテンツの10%が1時間未満キャッシュされることを示します。

以下の図16.5でコンテンツタイプごとのキャッシュ可能性を詳細に調べると、すべてのHTMLレスポンスの約半分がキャッシュ不可と見なされていることがわかります。さらに、画像とスクリプトの16%はキャッシュ不可です。

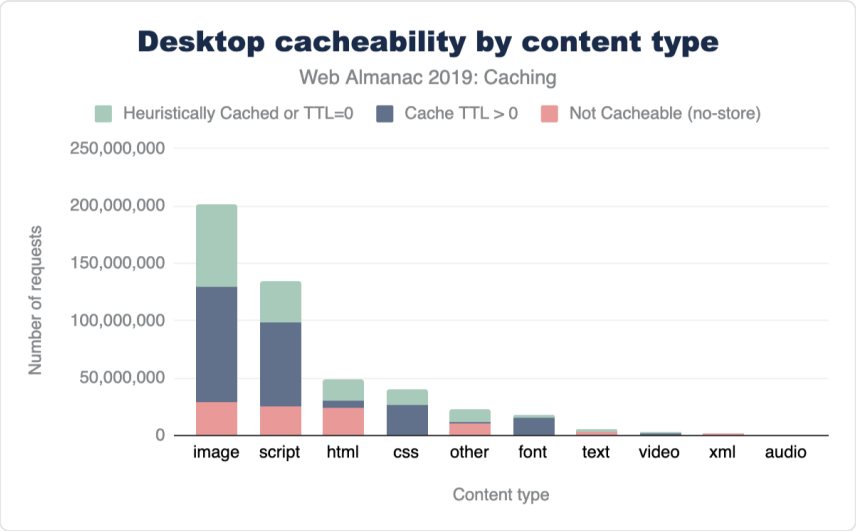


図16.5. デスクトップのコンテンツタイプごとのキャッシュ可能性の分布。

モバイルの同じデータを以下に示します。ご覧のとおり、コンテンツタイプのキャッシュ可能性はデスクトップとモバイルで一貫しています。

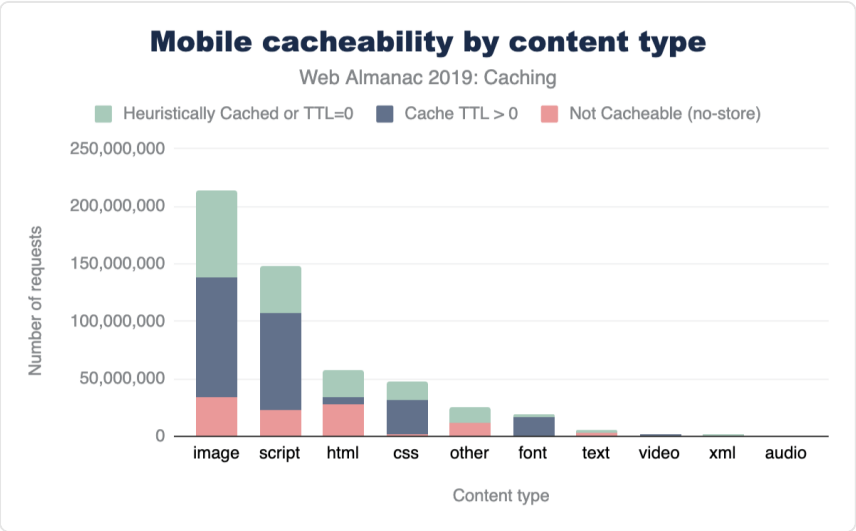


図16.6. モバイルのコンテンツタイプ別のキャッシュ可能性の分布。

`Cache-Control`と`Expires`

HTTP/1.0では、`Expires`ヘッダーは、レスポンスが古くなったと見なされる日時を示すために使用されました。その値は、次のような日付のタイムスタンプです。

```
Expires: Thu, 01 Dec 1994 16:00:00 GMT
```

HTTP/1.1は`Cache-Control`ヘッダーを導入し、最新のクライアントのほとんどは両方のヘッダーをサポートしています。このヘッダーは、キャッシングディレクティブを介して、はるかに高い拡張性を提供します。例えば。

- `no-store` リソースをキャッシュしないことを示すために使用できます。
- `max-age` 鮮度の寿命を示すために使用できます。
- `must-revalidate` キャッシュされたエントリは、使用する前に条件付きリクエストで検証する必要があることをクライアントに伝えます。
- `private` レスポンスはブラウザによってのみキャッシュされ、複数のクライアントにサービスを提供する仲介者によってキャッシュされるべきではないことを示します。

HTTPレスポンスの53%は、`max-age`ディレクティブを持つ`Cache-Control`ヘッダーが含まれ、54%はExpiresヘッダーが含まれます。ただし、これらのレスポンスの41%のみが両方のヘッダーを使用します。つまり、レスポンスの13%が古い`Expires`ヘッダーのみに基づいてキャッシュされます。

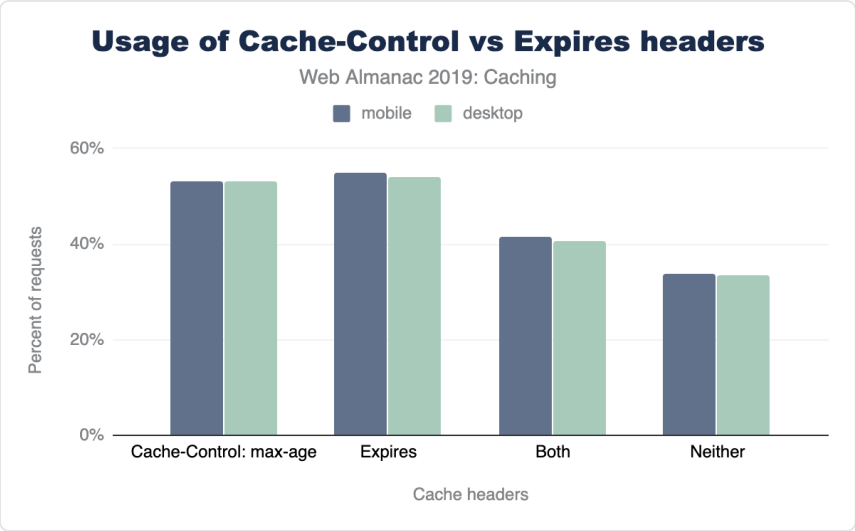


図16.7. `Cache-Control` と `Expires` ヘッダーの使用法。

`Cache-Control` ディレクティブ

HTTP/1.1仕様[360]には、`Cache-Control` レスポンスヘッダーで使用できる複数のディレクティブが含まれており、以下で詳しく説明します。1つのレスポンスで複数を使用できることに注意してください。

360. https://tools.ietf.org/html/rfc7234#section-5.2.1

ディレクティブ	説明
max-age	リソースをキャッシュできる秒数を示します。
public	任意のキャッシュにレスポンスを保存できます。
no-cache	キャッシュされたエントリは、使用する前に再検証する必要があります。
must-revalidate	古いキャッシュエントリは、使用する前に再検証する必要があります。
no-store	レスポンスがキャッシュ不可能なことを示します。
private	レスポンスは特定のユーザー向けであり、共有キャッシュに保存しない。
no-transform	このリソースに対して変換を行わないでください。
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、共有キャッシュにのみ適用されます。
immutable	キャッシュされたエントリは決して変更されず、再検証は不要であることを示します。
stale-while-revalidate	クライアントがバックグラウンドで新しいレスポンスを非同期にチェックしながら、古いレスポンスを受け入れようとしていることを示します。
stale-if-error	新しいレスポンスのチェックが失敗した場合に、クライアントが古いレスポンスを受け入れる意思があることを示します。

図16.8. `Cache-Control` ディレクティブ。

たとえば、`cache-control : public、max-age = 43200`は、キャッシュされたエントリを43,200秒間保存し、共有キャッシュに保存できることを示します。

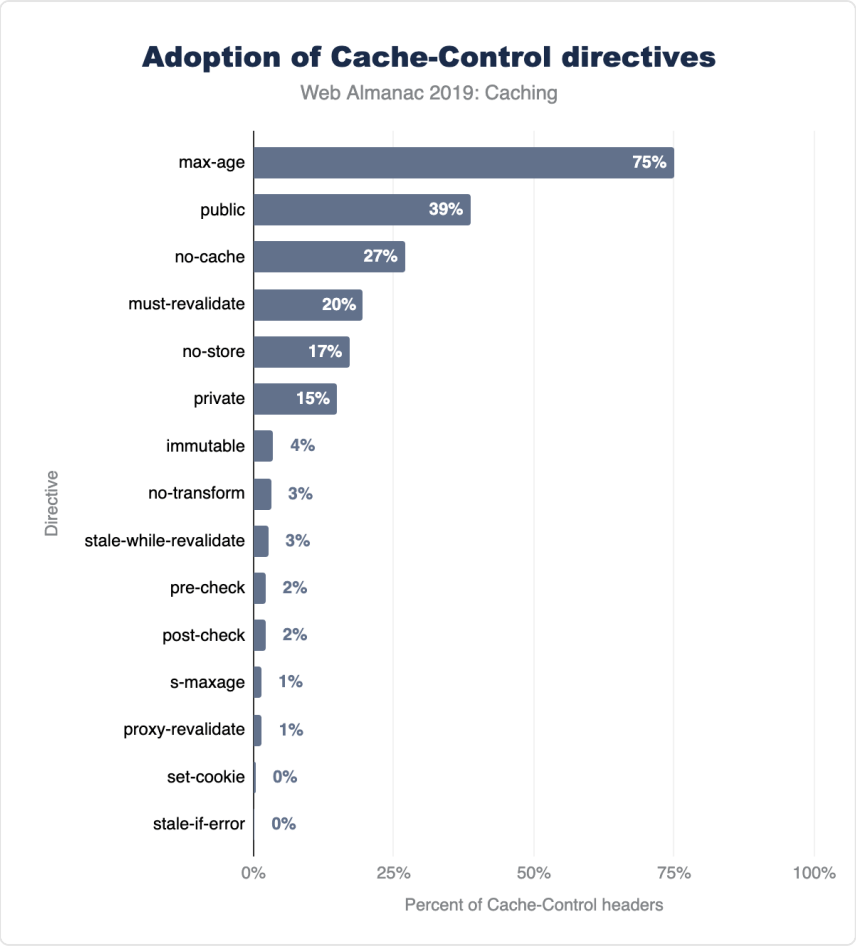


図16.9. モバイルでの `Cache-Control` ディレクティブの使用。

上記の図16.9は、モバイルWebサイトで使用されている上位15の`Cache-Control`ディレクティブを示しています。デスクトップとモバイルの結果は非常に似ています。これらのキャッシュディレクティブの人気について、興味深い観察結果がいくつかあります。

- `max-age`は`Cache-Control`ヘッダーのほぼ75%で使用され、`no-store`は18%で使用されます。
- `private`が指定されない限り、キャッシュされたエントリは`public`であると想定されるため、`public`が必要になることはほとんどありません。回答の約38%に`public`が含まれています。

- `immutable` ディレクティブは比較的新しく、2017年に導入[361]され、FirefoxおよびSafariでサポート[362]されています。その使用率は3.4%に拡大し、Facebook、Googleのサードパーティのレスポンス[363]で広く使用されています。

このリストに表示される別の興味深いディレクティブセットは、 `pre-check` と `post-check` です。これらは、 `Cache-Control` レスポンスヘッダーの2.2%（約780万件のレスポンス）で使用されます。このヘッダーのペアは、バックグラウンドで検証を提供するためにInternet Explorer 5で導入された[364]ものですが、Webサイトによって正しく実装されることはほとんどありませんでした。これらのヘッダーを使用したレスポンスの99.2%は、 `pre-check=0` と `post-check=0` の組み合わせを使用していました。これらのディレクティブの両方が0に設定されている場合、両方のディレクティブは無視されます。したがって、これらのディレクティブは正しく使用されなかったようです！

ロングテールでは、レスポンスの0.28%で1,500を超える間違ったディレクティブが使用されています。これらはクライアントによって無視され、「nocache」「s-max-age」「smax-age」「maxage」などのスペルミスが含まれます。「max-stale」「proxy-public」「surrogate-control」など存在しないディレクティブも多数あります。

`Cache-Control`: `no-store`, `no-cache` and `max-age=0`

レスポンスがキャッシュ可能でない場合、 `Cache-Control` `no-store` ディレクティブを使用する必要があります。このディレクティブを使用しない場合、レスポンスはキャッシュ可能です。

レスポンスをキャッシュ不可に設定しようとすると、いくつかの一般的なエラーが発生します。

- `Cache-Control: no-cache` の設定は、リソースがキャッシュできないように聞こえるかもしれません。ただし、 `no-cache` ディレクティブでは、使用する前にキャッシュされたエントリを再検証する必要があり、キャッシュ不可と同じではありません。
- `Cache-Control: max-age = 0` を設定すると、TTLが0秒に設定されますが、これはキャッシュ不可と同じではありません。 max-ageを0に設定すると、リソースはブラウザーのキャッシュに保存され、すぐに無効になります。これによ

361. https://code.facebook.com/posts/557147474482256/this-browser-tweak-saved-60-of-requests-to-facebook
362. https://developer.mozilla.org/docs/Web/HTTP/Headers/Cache-Control#Browser_compatibility
363. https://discuss.httparchive.org/t/cache-control-immutable-a-year-later/1195
364. https://blogs.msdn.microsoft.com/ieinternals/2009/07/20/internet-explorers-cache-control-extensions/

り、ブラウザは条件付きリクエストを実行してリソースの新しさを検証する必要があります。

機能的には、 `no-cache` と `max-age=0` は似ています。どちらもキャッシュされたリソースの再検証を必要とするためです。 `no-cache` ディレクティブは、0より大きい `max-age` ディレクティブと一緒に使用することもできます。

300万を超えるレスポンスには、 `no-store` 、 `no-cache` 、 `max-age=0` の組み合わせが含まれます。これらのディレクティブのうち、 `no-store` が優先され、他のディレクティブは単に冗長です。

レスポンスの18%には `no-store` が含まれ、レスポンスの16.6%には `no-store` と `no-cache` の両方が含まれます。 `no-store` が優先されるため、リソースは最終的にキャッシュ不可になります。

`max-age=0` ディレクティブは、 `no-store` が存在しないレスポンスの1.1%（400万を超えるレスポンス）に存在します。これらのリソースはブラウザにキャッシュされますが、すぐに期限切れになるため、再検証が必要になります。

キャッシュTTLとリソースの経過時間はどのように比較されますか？

これまで、Webサーバーがキャッシュ可能なものをクライアントに通知する方法と、キャッシュされる期間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ときは、提供しているコンテンツの古さを理解することも重要です。

キャッシュTTLを選択するときは、「これらのアセットをどのくらいの頻度で更新しますか？」と自問してください。そして「彼らのコンテンツの感度は何ですか？」。たとえば、ヒーローのイメージがたまに更新される場合、非常に長いTTLでキャッシュします。JavaScriptリソースが頻繁に変更されることが予想される場合は、バージョン管理して長いTTLでキャッシュするか、短いTTLでキャッシュします。

以下のグラフは、コンテンツタイプごとのリソースの相対的な年を示しています。ここで、より詳細な分析[365]を読むことができます。 HTMLは最も短い年齢のコンテンツタイプである傾向があり、伝統的にキャッシュ可能なリソース（スクリプト、CSS、およびフォント）の非常に大きな割合が1年以上古いです！

365. *https://discuss.httparchive.org/t/analyzing-resource-age-by-content-type/1659*

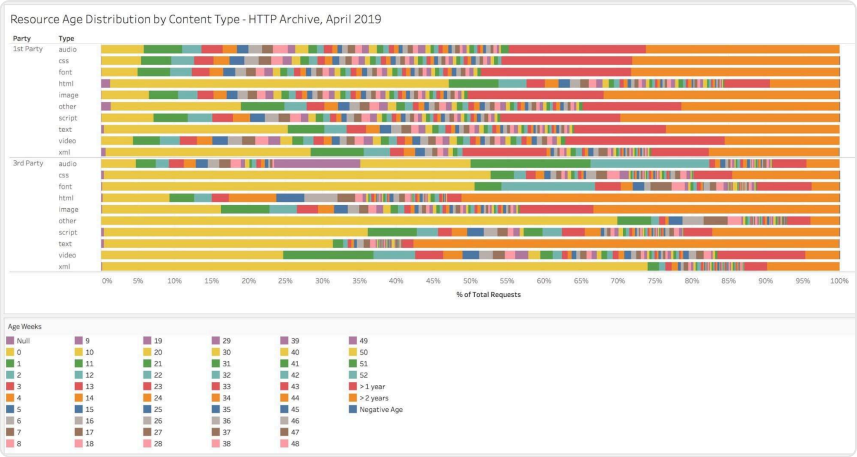


図16.10. コンテンツタイプ別のリソース年分布。

リソースのキャッシュ可能性をその経過時間と比較することにより、TTLが適切であるか短すぎるかを判断できます。たとえば、以下のレスポンスによって提供されるリソースは、2019年8月25日に最後の変更がされました。つまり、配信時に49日経過していました。`Cache-Control`ヘッダーは、43,200秒（12時間）キャッシュできることを示しています。より長いTTLが適切かどうかを調査するのに十分古いものです。

```
< HTTP/1.1 200
< Date: Sun, 13 Oct 2019 19:36:57 GMT
< Content-Type: application/javascript; charset=utf-8
< Content-Length: 3052
< Vary: Accept-Encoding
< Server: gunicorn/19.7.1
< Last-Modified: Sun, 25 Aug 2019 16:00:30 GMT
< Cache-Control: public, max-age=43200
< Expires: Mon, 14 Oct 2019 07:36:57 GMT
< ETag: "1566748830.0-3052-3932359948"
```

全体的に、Webで提供されるリソースの59%のキャッシュTTLは、コンテンツの年齢に比べて短すぎます。さらに、TTLと経過時間のデルタの中央値は25日です。

これをファーストパーティとサードパーティで分けると、ファーストパーティのリソースの70%がより長いTTLの恩恵を受けることもわかります。これは、キャッシュ可能なものに特に注意を払い、キャッシュが正しく構成されていることを確認する必要があることを明確に

強調しています。

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	70.7%	47.9%	59.2%
モバイル	71.4%	46.8%	59.6%

図16.11. TTLが短いリクエストの割合。

鮮度の検証

キャッシュ内に格納されたレスポンスの検証に使用されるHTTPレスポンスヘッダーは、`Last-Modified`および`ETag`です。 `Last-Modified`ヘッダーは、その名前が示すとおりに機能し、オブジェクトが最後に変更された時刻を提供します。 `ETag`ヘッダーは、コンテンツの一意の識別子を提供します。

たとえば、以下のレスポンスは2019年8月25日に変更され、「`1566748830.0-3052-3932359948`」の`ETag`値があります。

```
< HTTP/1.1 200
< Date: Sun, 13 Oct 2019 19:36:57 GMT
< Content-Type: application/javascript; charset=utf-8
< Content-Length: 3052
< Vary: Accept-Encoding
< Server: gunicorn/19.7.1
< Last-Modified: Sun, 25 Aug 2019 16:00:30 GMT
< Cache-Control: public, max-age=43200
< Expires: Mon, 14 Oct 2019 07:36:57 GMT
< ETag: "1566748830.0-3052-3932359948"
```

クライアントは、`If-Modified-Since`という名前のリクエストヘッダーの`Last-Modified`値を使用して、キャッシュされたエントリを検証する条件付きリクエストを送信できます。同様に、`If-None-Match`リクエストヘッダーを使用してリソースを検証することもできます。これは、クライアントがキャッシュ内のリソースに対して持っている`ETag`値に対して検証します。

以下の例では、キャッシュエントリはまだ有効であり、`HTTP 304`がコンテンツなしで返さ

れました。これにより、リソース自体のダウンロードが保存されます。キャッシュエントリが最新ではない場合、サーバーは`200`で更新されたリソースを応答し、再度ダウンロードする必要があります。

```
> GET /static/js/main.js HTTP/1.1
> Host: www.httparchive.org
> User-Agent: curl/7.54.0
> Accept: */*
> If-Modified-Since: Sun, 25 Aug 2019 16:00:30 GMT

< HTTP/1.1 304
< Date: Thu, 17 Oct 2019 02:31:08 GMT
< Server: gunicorn/19.7.1
< Cache-Control: public, max-age=43200
< Expires: Thu, 17 Oct 2019 14:31:08 GMT
< ETag: "1566748830.0-3052-3932359948"
< Accept-Ranges: bytes
```

全体としてレスポンスの65%は`Last-Modified`ヘッダーで、42%は`ETag`で提供され、38%は両方を使用します。ただし、レスポンスの30%には`Last-Modified`ヘッダー、`ETag`ヘッダーは含まれていません。

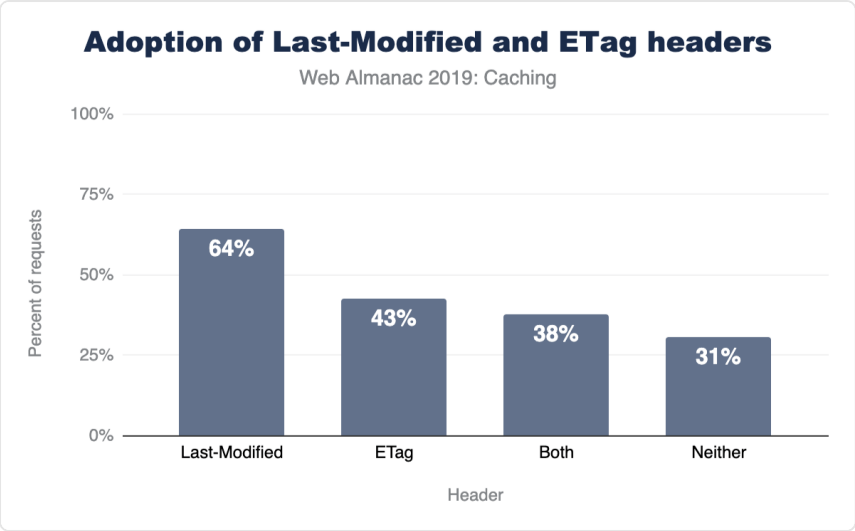


図16.12. デスクトップWebサイトの `Last-Modified` および `ETag` ヘッダーを介した鮮度の検証の採用。

日付文字列の有効性

タイムスタンプの伝達に使用されるHTTPヘッダーがいくつかあり、これらの形式は非常に重要です。 `Date` レスポンスヘッダーは、リソースがいつクライアントに提供されたかを示します。 `Last-Modified` レスポンスヘッダーは、サーバーでリソースが最後に変更された日時を示します。また、 `Expires` ヘッダーは、（ `Cache-Control` ヘッダーの存在しない場合）リソースがキャッシュ可能な期間を示すために使用されます。

これら3つのHTTPヘッダーはすべて、日付形式の文字列を使用してタイムスタンプを表します。

例えば。

```
> GET /static/js/main.js HTTP/1.1
> Host: httparchive.org
> User-Agent: curl/7.54.0
> Accept: */*
```

```
< HTTP/1.1 200
< Date: Sun, 13 Oct 2019 19:36:57 GMT
< Content-Type: application/javascript; charset=utf-8
< Content-Length: 3052
< Vary: Accept-Encoding
< Server: gunicorn/19.7.1
< Last-modified: Sun, 25 Aug 2019 16:00:30 GMT
< Cache-Control: public, max-age=43200
< Expires: Mon, 14 Oct 2019 07:36:57 GMT
< ETag: "1566748830.0-3052-3932359948"
```

ほとんどのクライアントは、無効な日付文字列を無視します。これにより、提供されているレスポンスを無視します。これは、誤った `Last-Modified` ヘッダーがLast-Modifiedタイムスタンプなしでキャッシュされるため、条件付きリクエストを実行できなくなるため、キャッシュ可能性に影響を与える可能性があります。

通常、 `Date` HTTPレスポンスヘッダーは、クライアントにレスポンスを提供するWebサーバーまたはCDNによって生成されます。ヘッダーは通常、サーバーによって自動的に生成されるため、エラーが発生しにくい傾向はあります。これは、無効な `Date` ヘッダーの割合が非常に低いことを反映しています。 `Last-Modified` ヘッダーは非常に類似しており、無効なヘッダーは0.67%のみでした。しかし、驚いたのは、3.64%の `Expires` ヘッダーが無効な日付形式を使用していたことです！

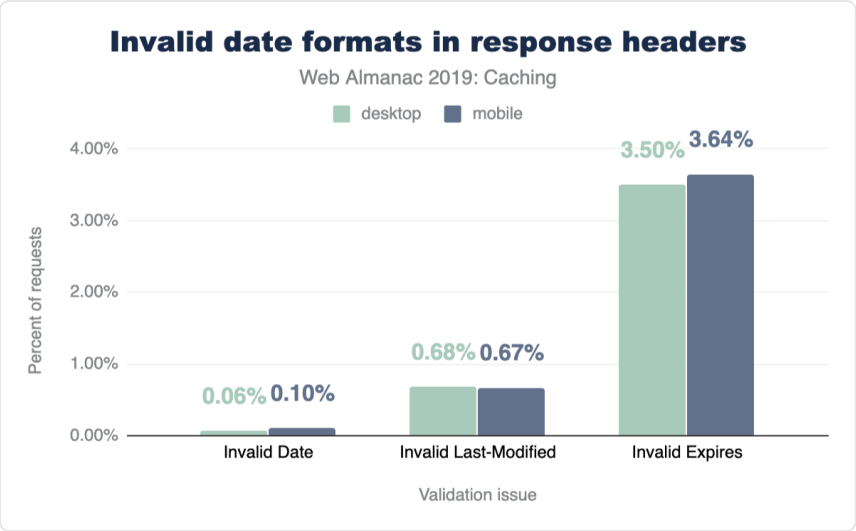


図16.13. レスポンスヘッダーの無効な日付形式。

`Expires` ヘッダーの無効な使用の例は次のとおりです。

- 有効な日付形式ですが、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` ヘッダーの最大のソースは、人気のあるサードパーティから提供されるアセットからのものです。たとえば、`Expires : Tue、27 Apr 1971 19:44:06 EST` など、日付/時刻はESTタイムゾーンを使用します。

ヘッダーを変更

キャッシングで最も重要な手順の1つは、要求されているリソースがキャッシュされているかどうかを判断することです。これは単純に見えるかもしれませんが、多くの場合、URLだけではこれを判断するには不十分です。たとえば同じURLのリクエストは、使用する圧縮（Gzip、Brotliなど）が異なる場合や、モバイルの訪問者に合わせて変更および調整できます。

この問題を解決するために、クライアントはキャッシュされた各リソースに一意の識別子（キャッシュキー）を与えます。デフォルトでは、このキャッシュキーは単にリソースの

URLですが、開発者はVaryヘッダーを使用して他の要素（圧縮方法など）を追加できます。

Varyヘッダーは、1つ以上の要求ヘッダー値の値をキャッシュキーに追加するようにクライアントに指示します。この最も一般的な例は、`Vary : Accept-Encoding`です。これにより、`Accept-Encoding`リクエストヘッダー値（`gzip`、`br`、`deflate`など）のキャッシュエントリが別になります。

別の一般的な値は`Vary : Accept-Encoding`、`User-Agent`です。これは、Accept-Encoding値と`User-Agent`文字列の両方によってキャッシュエントリを変更するようにクライアントに指示します。共有プロキシとCDNを扱う場合、`Accept-Encoding`以外の値を使用すると、キャッシュキーが弱められ、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ため、問題発生の可能性があります。

一般に、そのヘッダーに基づいてクライアントに代替コンテンツを提供する場合のみ、キャッシュを変更する必要があります。

`Vary`ヘッダーは、HTTPレスポンスの39%、および`Cache-Control`ヘッダーを含むレスポンスの45%で使用されます。

以下のグラフは、上位10個の`Vary`ヘッダー値の人気を示しています。`Accept-Encoding`はVaryの使用の90%を占め、`User-Agent`（11%）、`Origin`（9%）、`Accept`（3%）が残りの大部分を占めています。

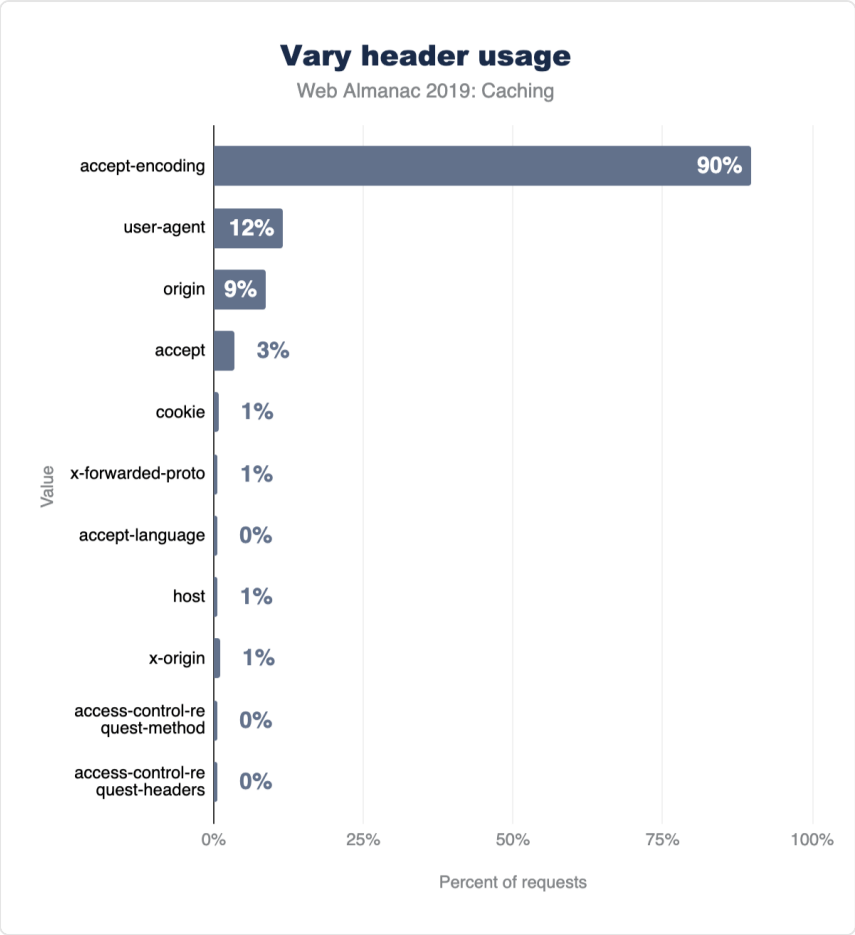


図16.14. Varyヘッダーの使用率。

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、そのヘッダー全体もキャッシュにスワップされます。これが、DevToolsを介してキャッシュされたレスポンスを検査するときにレスポンスヘッダーを表示できる理由です。

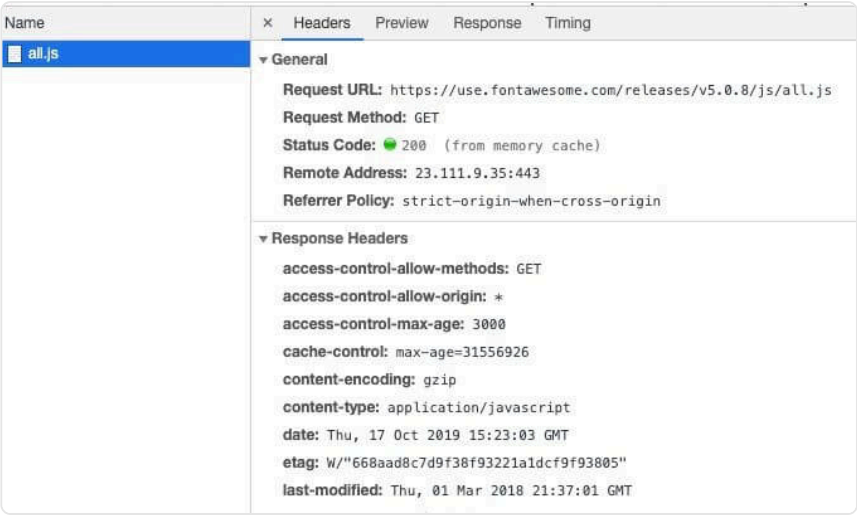


図16.15. キャッシュされたリソースのChrome開発ツール。

しかし、レスポンスに`Set-Cookie`がある場合はどうなりますか？ RFC 7234セクション8[366]によると、`Set-Cookie`レスポンスヘッダーはキャッシングを禁止しません。これは、キャッシュされたエントリが`Set-Cookie`でキャッシュされている場合、それが含まれている可能性があることを意味します。 RFCでは、適切な`Cache-Control`ヘッダーを構成して、レスポンスのキャッシュ方法を制御することを推奨しています。

`Set-Cookie`を使用してレスポンスをキャッシュすることのリスクの1つは、Cookieの値を保存し、後続の要求に提供できることです。 Cookieの目的によっては、心配な結果になる可能性があります。たとえば、ログインCookieまたはセッションCookieが共有キャッシュに存在する場合、そのCookieは別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。これを回避する1つの方法は、`Cache-Control``プライベート`ディレクティブを使用することです。これにより、クライアントブラウザーによるレスポンスのキャッシュのみが許可されます。

キャッシュ可能なレスポンスの3%に`Set-Cookieヘッダー`が含まれています。これらのレスポンスのうち、`プライベート`ディレクティブを使用しているのは18%のみです。残りの82%には、パブリックおよびプライベートキャッシュサーバーでキャッシュできる`Set-Cookie`を含む530万のHTTPレスポンスが含まれています。

366. https://tools.ietf.org/html/rfc7234#section-8

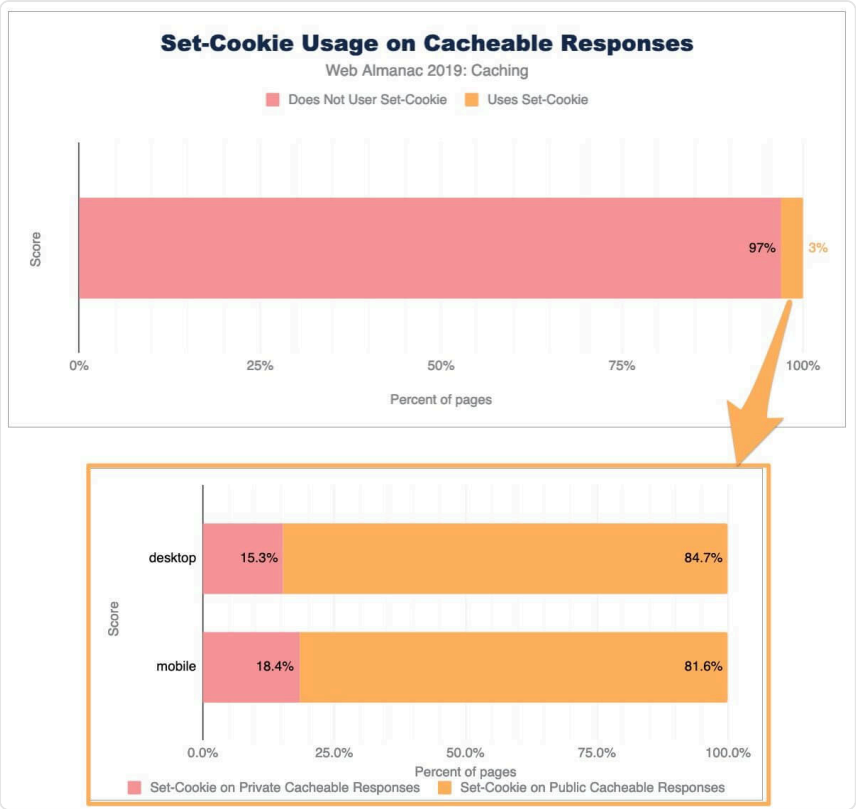


図16.16. `Set-Cookie` レスポンスのキャッシュ可能なレスポンス。

AppCacheおよびService Worker

アプリケーションキャッシュまたはAppCacheはHTML5の機能であり、開発者はブラウザがキャッシュするリソースを指定し、オフラインユーザーが利用できるようにできます。この機能は廃止されており、Web標準からも削除[367]され、ブラウザーのサポートは減少しています。実際、使われているのが見つかった場合、Firefox v44 +は、開発者に対して代わりにService Workerを使用することを推奨しています[368]。 Chrome 70は、アプリケーションキャッシュをセキュリティで保護されたコンテキストのみに制限します[369]。業界では、このタイプの機能をService Workerに実装する方向へ移行しており、ブラウザサポート[370]は急速に成長しています。

367. https://web.archive.org/web/20191115024726/https://html.spec.whatwg.org/multipage/offline.html
368. https://developer.mozilla.org/docs/Web/API/Service_Worker_API/Using_Service_Workers
369. https://www.chromestatus.com/feature/5714236168732672
370. https://caniuse.com/#feat=serviceworkers

実際、HTTP Archiveトレンドレポートの1つは、以下に示すService Worker[371]の採用を示しています。

図16.17. Service Workerが制御するページの時系列。(引用: HTTP Archive[372])

採用率はまだウェブサイトの1%を下回っていますが、2017年1月から着実に増加しています。プログレッシブWebアプリの章では、人気サイトでの使用によりこのグラフが示唆するよりも多く使用されているという事実を含め、上記のグラフでは1回のみカウントされます。

次の表では、AppCacheとService Workerの使用状況の概要を確認できます。 32,292のWebサイトでService Workerが実装されていますが、1,867のサイトでは非推奨のAppCache機能が引き続き使用されています。

	***Service Worker*を使用しない**	***Service Worker*を使用する**	***合計***
AppCacheを使用しない	*5,045,337*	*32,241*	*5,077,578*
AppCacheを使用する	*1,816*	*51*	*1,867*
合計	*5,047,153*	*32,292*	*5,079,445*

図16.18. AppCacheを使用するWebサイトとService Workerの数。

これをHTTPとHTTPSで分類すると、さらに興味深いものになります。 581のAppCache対応

371. https://httparchive.org/reports/progressive-web-apps#swControlledPages
372. https://httparchive.org/reports/progressive-web-apps#swControlledPages

サイトはHTTP経由で提供されます。つまり、Chromeがこの機能を無効にしている可能性があります。 HTTPSはService Workerを使用するための要件ですが、それらを使用するサイトの907はHTTP経由で提供されます。

		Service Workerを使用しない	Service Workerを使用する
HTTP	AppCacheを使用しない	1,968,736	907
	AppCacheを使用する	580	1
HTTPS	AppCacheを使用しない	3,076,601	31,334
	AppCacheを使用する	1,236	50

図16.19. AppCacheを使用するWebサイト数とHTTP/HTTPSによるService Workerの使用量。

キャッシングの機会を特定する

GoogleのLighthouse[373]ツールを使用すると、ユーザーはWebページに対して一連の監査を実行できます。キャッシュポリシー監査[374]では、サイトが追加のキャッシュの恩恵を受けることができるかどうかを評価します。これは、コンテンツの経過時間（`Last-Modified`ヘッダー経由）をキャッシュTTLと比較し、リソースがキャッシュから提供される可能性を推定することによりこれを行います。スコアに応じて、結果にキャッシュの推奨事項が表示され、キャッシュできる特定のリソースのリストが表示される場合があります。

373. https://developers.google.com/web/tools/lighthouse
374. https://developers.google.com/web/tools/lighthouse/audits/cache-policy

Diagnostics — More information about the performance of your application. These numbers don't directly affect the Performance score.

- ▲ Ensure text remains visible during webfont load
- ▲ Reduce the impact of third-party code — Third-party code blocked the main thread for 1,660 ms
- ▲ Minimize main-thread work — 15.8 s
- ▲ Serve static assets with an efficient cache policy — 101 resources found

A long cache lifetime can speed up repeat visits to your page. Learn more.

☑ Show 3rd-party resources (61)

URL	Cache TTL	Size
	None	233 KB
	None	14 KB
	None	9 KB
	None	7 KB
	None	4 KB
	None	3 KB
	None	3 KB
	None	2 KB
	None	2 KB

図16.20. キャッシュポリシーの改善の可能性を強調したLighthouseレポート。

Lighthouseは、監査ごとに0%〜100%の範囲のスコアを計算し、それらのスコアは全体のスコアに組み込まれます。キャッシングスコア[375]は、潜在的なバイト節約に基づいています。Lighthouseの結果を調べると、キャッシュポリシーでどれだけのサイトがうまく機能しているかを把握できます。

375. https://developers.google.com/web/tools/lighthouse/audits/cache-policy

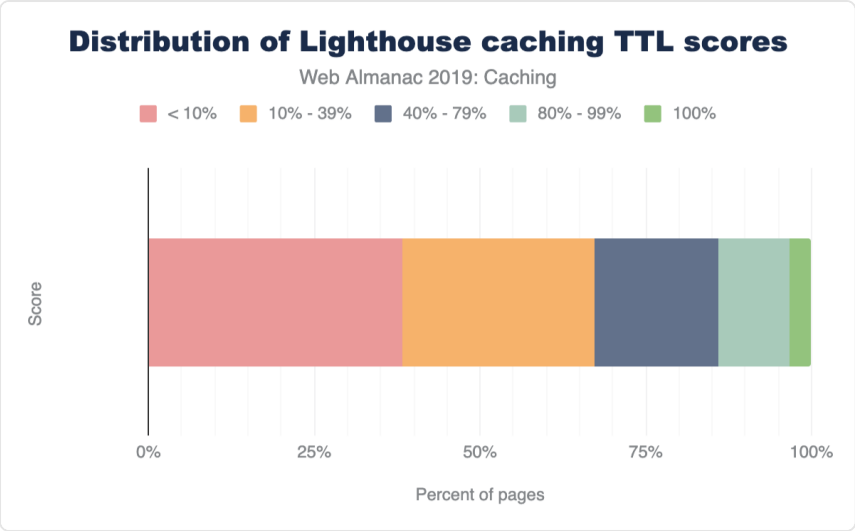


図16.21. モバイルWebページの「Use Long Cache TTL」監査のLighthouseスコアの分布。

100%を獲得したサイトは3.4%のみです。これは、ほとんどのサイトがキャッシュの最適化の恩恵を受けることができることを意味します。サイトの圧倒的多数が40%未満で、38%が10%未満のスコアを記録しています。これに基づいて、Webにはかなりの量のキャッシュの機会があります。

Lighthouseは、より長いキャッシュポリシーを有効にすることで、繰り返しビューで保存できるバイト数も示します。追加のキャッシュの恩恵を受ける可能性のあるサイトのうち、82%がページの重みを最大で1MB削減できます。

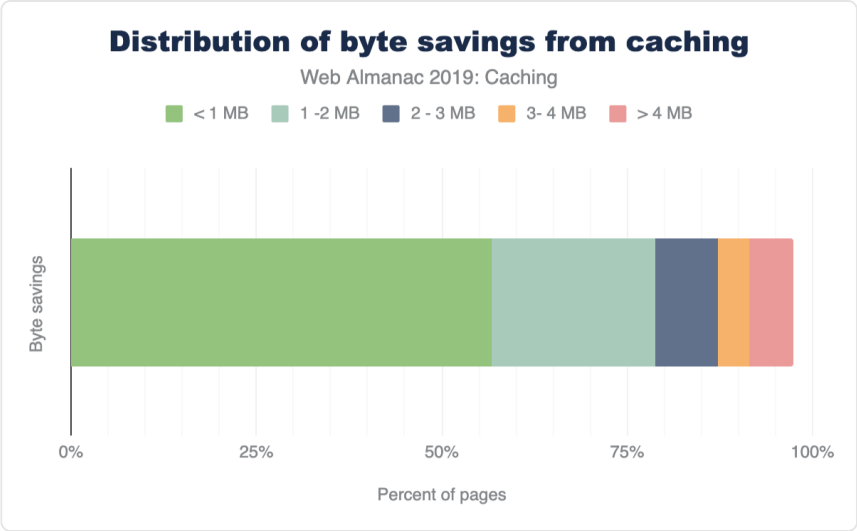


図16.22. Lighthouseキャッシング監査による潜在的なバイト節約の分布。

結論

キャッシングは非常に強力な機能であり、ブラウザ、プロキシ、その他の仲介者（CDNなど）がWebコンテンツを保存し、エンドユーザーへ提供できるようにします。これにより、往復時間が短縮され、コストのかかるネットワーク要求が最小限に抑えられるため、パフォーマンス上のメリットは非常に大きくなります。

キャッシュも非常に複雑なトピックです。キャッシュ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HTTPレスポンスヘッダーは多数あります。`Cache-Control`ディレクティブは、非常に多くの柔軟性と制御を提供します。ただし、開発者は、それがもたらす間違いの追加の機会に注意する必要があります。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ためにサイトを定期的に監査することをお勧めします。Lighthouse[376]やREDbot[377]などのツールは、分析の簡素化に役立ちます。

376. https://developers.google.com/web/tools/lighthouse
377. https://redbot.org/

著者

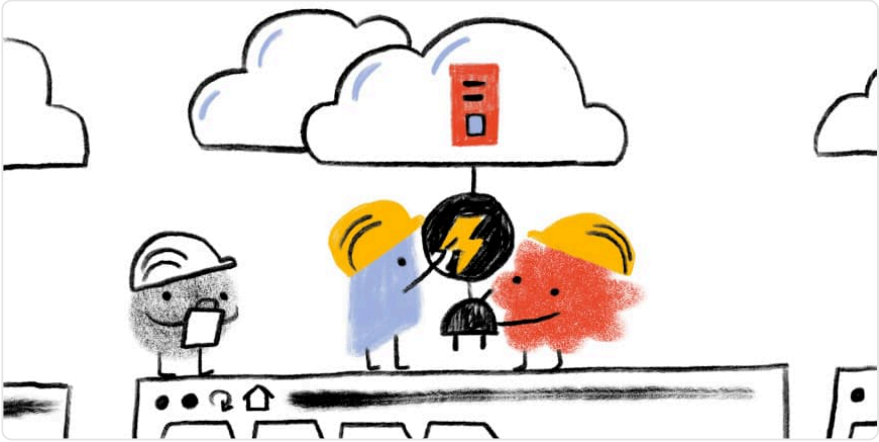
Paul Calvano

@paulcalvano paulcalvano https://paulcalvano.com

Paul Calvanoは、アカマイ[378]のウェブパフォーマンス・アーキテクトで、ウェブサイトのパフォーマンス向上を支援しています。また、HTTP Archiveプロジェクトの共同管理者でもあります。@paulcalvano でツイートしたり、http://paulcalvano.com でブログを書いたり、https://discuss.httparchive.org でHTTP Archiveの研究を共有したりしています。

378. *https://www.akamai.com/*

部 IV 章 17
CDN



Andy Davies と Colin Bendell によって書かれた。
Yoav Weiss、 Paul Calvano、 Patrick Meenan と Erik Nygren によってレビュ。
Raghu Ramakrishnan と Rick Viscomi による分析。
Rick Viscomi 編集。
Sakae Kotaro によって翻訳された。

導入

「コンテンツ配信ネットワーク」は、Webサイトの読み込みを高速化するためのSteve Soudersによる独自の推奨事項[379]の1つでした。昨今でも有効なアドバイスです。Web Almanacのこの章ではSteveの推奨事項がどの程度広く採用されているか、サイトがコンテンツ配信ネットワーク（CDN）をどのように使用しているか、およびそれらが使用している機能のいくつかを検討します。

基本的にCDNは待ち時間（パケットがネットワーク上の2つのポイント間、たとえば訪問者のデバイスからサーバーに移動する時間）を短縮します、待ち時間はページの読み込み速度の重要な要素です。

CDNは、2つの方法で待機時間を短縮します。ユーザーに近い場所からコンテンツを提供すること、2番目はエンドユーザーに近いTCP接続を終了することです。

379. http://stevesouders.com/examples/rules.php

歴史的にユーザーからバイトへの論理パスが短くなるように、CDNはバイトのキャッシュまたはコピーに使用されていました。多くの人が要求するファイルは、origin（サーバー）から一度取得してユーザーに近いサーバーへ保存できるため、転送時間を節約できます。

CDNは、TCP遅延にも役立ちます。 TCPの遅延により、ブラウザーとサーバー間の接続を確立するのにかかる時間、接続を保護するのにかかる時間、および最終的にコンテンツをダウンロードする速度が決まります。せいぜいネットワークパケットは光の速度の約3分の2で移動するため、その往復にかかる時間は通信の両端がどれだけ離れているか、その間に何があるかによって決まります。混雑したネットワーク、過負荷の機器、ネットワークのタイプすべてがさらなる遅延を追加します。 CDNを使用して接続のサーバー側を訪問者の近くに移動すると、この遅延のペナルティが減少し、接続時間、TLSネゴシエーション時間が短縮されコンテンツのダウンロード速度が向上します。

CDNは、多くの場合、訪問者の近くで静的コンテンツを保存および提供する単なるキャッシュと考えられていますが、さらに多くの機能を備えています。 CDNは、遅延のペナルティを克服するだけでなく、パフォーマンスとセキュリティの向上に役立つ他の機能を提供するものが増えています。

- CDNを使用して動的コンテンツ（ベースHTMLページ、API応答など）をプロキシすることにより、ブラウザーとCDN自身のネットワークとの間の遅延が短縮され、元の速度に戻ることができます。
- 一部のCDNは、ページを最適化してダウンロードとレンダリングを高速化する変換を提供するか、画像を最適化して表示するデバイスに適したサイズ（画像の大きさとファイルサイズの両方）に変換します。
- セキュリティの観点から、悪意のあるトラフィックとボットは要求がoriginへ到達する前にCDNによって除外される可能性があり、その幅広い顧客ベースによりCDNが新しい脅威をより早く発見して対応できることを意味します。
- エッジコンピューティング[380]の台頭により、サイトは訪問者の近くで独自のコードを実行できるようになりパフォーマンスが向上し、originの負荷が軽減されました。

最後にCDNもまた、originサーバーがサポートしていない場合でもエッジからブラウザーまでHTTP/2、TLS1.3、またはIPv6を有効にできるなどoriginでの変更を必要とせずにサイトが新しいテクノロジーを採用できるようにします。

380. *https://en.wikipedia.org/wiki/Edge_computing*

警告と免責事項

他の観察研究と同様に、測定できる範囲と影響には限界があります。 Web AlmanacのCDN使用に関して収集された統計は、特定のCDNベンダーのパフォーマンスや有効性を意味するものではありません。

Web Almanacに使用されるテスト方法には多くの制限があります。これらには以下が含まれます。

- **シミュレートされたネットワーク遅延**：Web Almanacは、トラフィックを総合的に形成する専用ネットワーク接続を使用します。
- **単一の地理的場所**：テストは単一のデータセンター[381]から実行されるため、多くのCDNベンダーの地理的分布をテストすることはできません。
- **キャッシュの有効性**：各CDNは独自の技術を使用しており、多くの場合、セキュリティ上の理由により、キャッシュのパフォーマンスは公開されていません。
- **ローカライゼーションと国際化**：地理的分布と同様に言語、地理固有のドメインの影響もテストに対して不透明です。
- CDN検出[382]は、主にDNS解決とHTTPヘッダーを介して行われます。ほとんどのCDNは、DNS CNAMEを使用してユーザーを最適なデータセンターにマッピングします。ただし、一部のCDNはAnyCast IPを使用するか、DNSチェインを隠す委任されたドメインからのA+AAAA応答を直接使用します。それ以外の場合、Webページは複数のCDNを使用して、WebPageTestのシングルリクエストパスから隠されているベンダー間でバランスを取ります。これらはすべて、測定の有効性を制限します。

最も重要なことは、これらの結果は潜在的な使用率を反映しているが、実際の影響を反映していないことです。 YouTubeは「ShoesByColin」よりも人気がありますが、使用率を比較するとどちらも同じ値として表示されます。

これを念頭に置いて、CDNのコンテキストで測定されなかったいくつかの意図的な統計があります。

- **TTFB**：CDNによる最初のバイトまでの時間について、キャッシュ可能とキャッシュの有効性についての正しい知識がなければ正しく測定できません。1つのサイトには、**ラウンドトリップタイム**（RTT）の管理にCDNを使用してるがキャッシュには使用してない場合、別のCDNベンダーを使用してコンテンツをキャッシュしてる別サイトと比較する時不利になります（注：これは、個々のCDNのパフ

381. *https://httparchive.org/faq#how-is-the-data-gathered*
382. *https://github.com/WPO-Foundation/wptagent/blob/master/internal/optimization_checks.py#L51*

ォーマンスについての結論を出していないので、パフォーマンスの章でTTFB分析には適用されません）。

- **キャッシュヒットとキャッシュミスのパフォーマンス**：前述のようにこれは正確にテストできません、なのでコールドキャッシュとホットキャッシュでページのパフォーマンステストを繰り返すことは信頼できません。

さらなる統計

Web Almanacの将来のバージョンでは、CDNベンダー間のTLSおよびRTTの管理をより詳細に検討する予定です。興味深いのは、OCSP Staplingの影響、TLS暗号パフォーマンスの違いです。 CWND（TCP輻輳ウィンドウ）成長率、特にBBR v1、v2、従来のTCP Cubicの採用。

CDNの採用と使い方

ウェブサイトの場合、CDNはプライマリドメイン（`www.shoesbycolin.com`）、サブドメインまたは兄弟ドメイン（`images.shoesbycolin.com`または`checkout.shoesbycolin.com`）、そして最後にサードパーティ（Google Analyticsなど）のパフォーマンスを改善できます。これらの各ユースケースにCDNを使用すると、さまざまな方法でパフォーマンスが向上します。

歴史的に、CDNはCSS、JavaScript、画像などの静的リソース専用に使用されていました。これらのリソースはおそらくバージョン管理され（パスに一意の番号を含む）、長期的にキャッシュされます。このようにして、ベースHTMLドメインと比較して、サブドメインまたは兄弟ドメインでのCDNの採用が増加することを期待する必要があります。従来のデザインパターンでは、`www.shoesbycolin.com`がデータセンター（又は**origin**）から直接HTMLを提供し、`static.shoesbycolin.com`がCDNを使用することを想定していました。

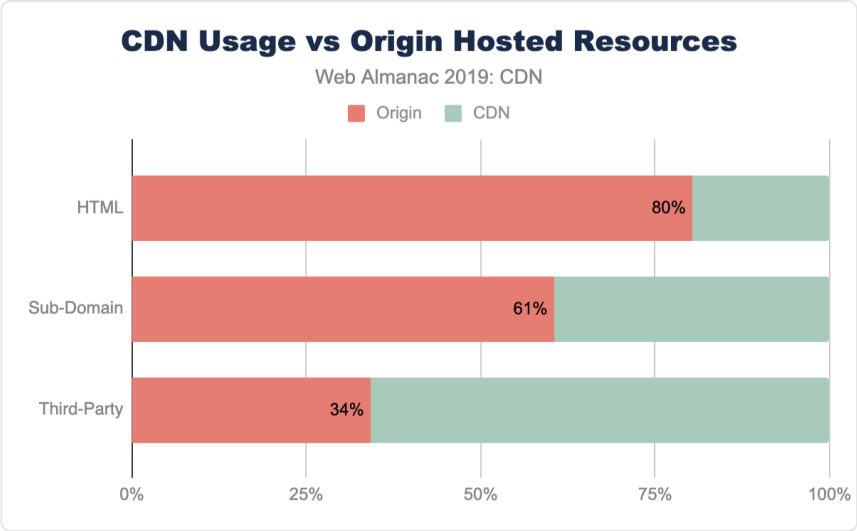


図17.1. 1. CDN使用量 vs. originがホストするリソース

実際、この伝統的なパターンは、クロールされたWebサイトの大部分で見られるものです。Webページの大部分（80%）は、元のベースHTMLを提供しています。この内訳はモバイルとデスクトップでほぼ同じであり、デスクトップでのCDNの使用率は0.4%しか低下していません。このわずかな差異は、CDNをより頻繁に使用するモバイル固有のWebページ（「mDot」）の小規模な継続使用に起因する可能性があります。

同様に、サブドメインから提供されるリソースは、サブドメインリソースの40%でCDNを利用する可能性が高くなります。サブドメインは、画像やCSSなどのリソースを分割するために使用されるか、チェックアウトやAPIなどの組織チームを反映するために使用されます。

ファーストパーティのリソースの大部分は依然としてoriginから直接提供されていますが、サードパーティのリソースはCDNの採用が大幅に増えています。すべてのサードパーティリソースのほぼ66%がCDNから提供されています。サードパーティのドメインはSaaS統合である可能性が高いため、CDNの使用はこれらのビジネス製品のコアになる可能性が高くなります。ほとんどのサードパーティコンテンツは共有リソース（JavaScriptまたはフォントCDN）、拡張コンテンツ（広告）、または統計に分類されます。これらすべての場合においてCDNを使用すると、SaaSソリューションのパフォーマンスとオフロードが向上します。

トップCDNプロバイダー

CDNプロバイダーには、汎用CDNと目的別CDNの2つのカテゴリがあります。汎用CDNプ

ロバイダーは、多くの業界のあらゆる種類のコンテンツを提供するカスタマイズと柔軟性を提供します。対照的に、目的に合ったCDNプロバイダーは同様のコンテンツ配信機能を提供しますが、特定のソリューションに焦点を絞っています。

これは、ベースHTMLコンテンツを提供していることが判明した上位のCDNを見ると明確に表されています。 HTMLを提供する最も頻繁なCDNは、汎用CDN（Cloudflare、Akamai、Fastly）およびプラットフォームサービスの一部としてバンドルされたCDN（Google、Amazon）を提供するクラウドソリューションプロバイダーです。対照的に、WordpressやNetlifyなど、ベースHTMLマークアップを提供する目的に合ったCDNプロバイダーはわずかです。

注：これにはトラフィックや使用量は反映されず、それらを使用するサイトの数のみが反映されます。

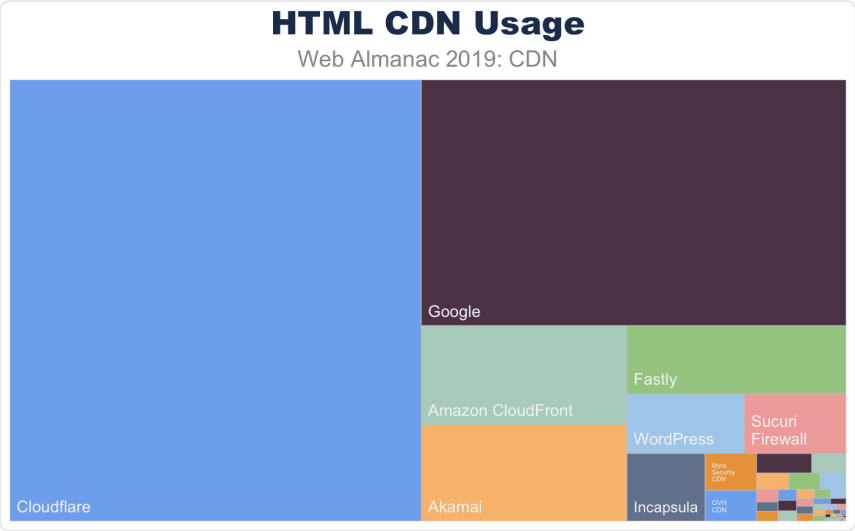


図17.2. HTML CDNの使用

HTML CDNの使用率 (%)	
ORIGIN	80.39
Cloudflare	9.61
Google	5.54
Amazon CloudFront	1.08
Akamai	1.05
Fastly	0.79
WordPress	0.37
Sucuri Firewall	0.31
Incapsula	0.28
Myra Security CDN	0.1
OVH CDN	0.08
Netlify	0.06
Edgecast	0.04
GoCache	0.03
Highwinds	0.03
CDNetworks	0.02
Limelight	0.01
Level 3	0.01
NetDNA	0.01
StackPath	0.01
Instart Logic	0.01
Azion	0.01
Yunjiasu	0.01
section.io	0.01
Microsoft Azure	0.01

図17.3. サイトごとのHTML上位25のCDN。

サブドメインリクエストの構成は非常に似ています。多くのWebサイトは静的コンテンツにサブドメインを使用しているため、CDNの使用量は増加する傾向があります。ベースページリクエストと同様に、これらのサブドメインから提供されるリソースは、一般的なCDN提供を利用します。

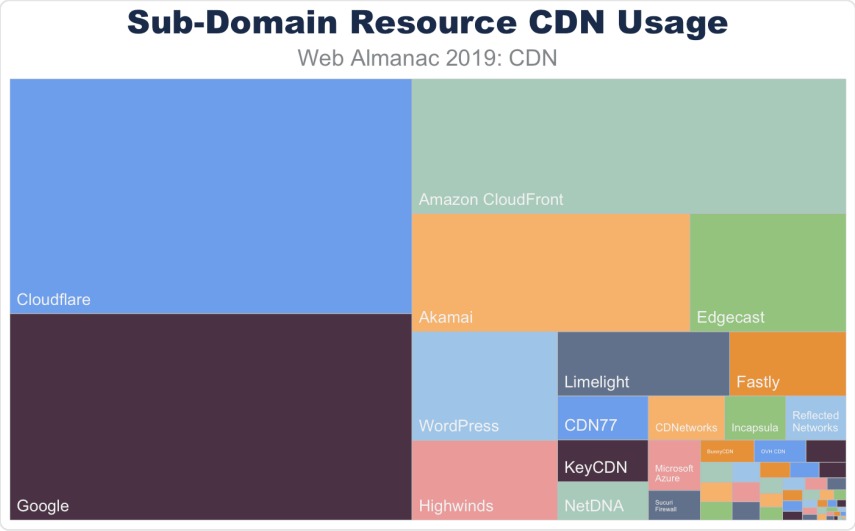


図17.4. サブドメインリソースのCDNの使用。

サブドメインのCDN使用率(%)	
ORIGIN	60.56
Cloudflare	10.06
Google	8.86
Amazon CloudFront	6.24
Akamai	3.5
Edgecast	1.97
WordPress	1.69
Highwinds	1.24
Limelight	1.18
Fastly	0.8
CDN77	0.43
KeyCDN	0.41
NetDNA	0.37
CDNetworks	0.36
Incapsula	0.29
Microsoft Azure	0.28
Reflected Networks	0.28
Sucuri Firewall	0.16
BunnyCDN	0.13
OVH CDN	0.12
Advanced Hosters CDN	0.1
Myra Security CDN	0.07
CDNvideo	0.07
Level 3	0.06
StackPath	0.06

表5のデータを示すツリーマップグ

ラフ。

表5のデータを示すツリーマップグラフ。

上位CDNプロバイダーの構成は、サードパーティのリソースに対して劇的に変化します。サードパーティのリソースをホストするCDNが頻繁に監視されるだけでなく、Facebook、Twitter、Googleなどの目的に合ったCDNプロバイダーも増加しています。

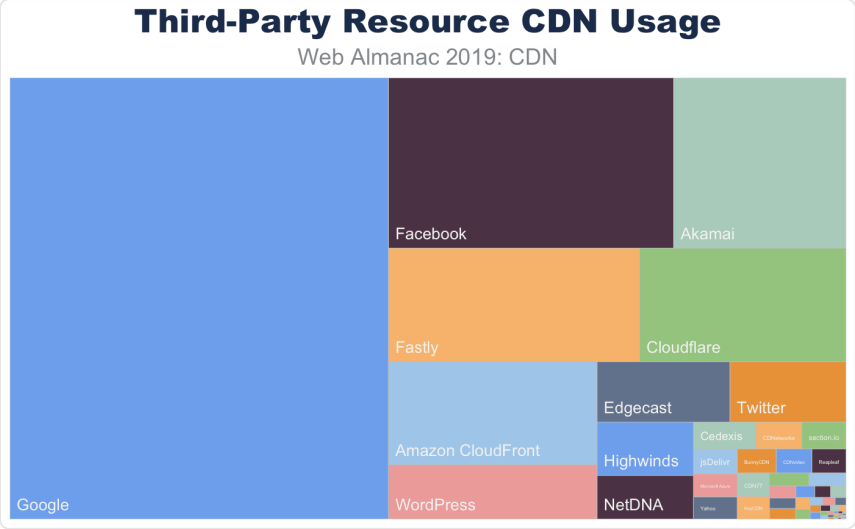


図17.6. サードパーティのリソースCDN使用。

サードパーティのCDN使用率(%)	
ORIGIN	34.27
Google	29.61
Facebook	8.47
Akamai	5.25
Fastly	5.14
Cloudflare	4.21
Amazon CloudFront	3.87
WordPress	2.06
Edgecast	1.45
Twitter	1.27
Highwinds	0.94
NetDNA	0.77
Cedexis	0.3
CDNetworks	0.22
section.io	0.22
jsDelivr	0.2
Microsoft Azure	0.18
Yahoo	0.18
BunnyCDN	0.17
CDNvideo	0.16
Reapleaf	0.15
CDN77	0.14
KeyCDN	0.13
Azion	0.09
StackPath	0.09

図17.7. サードパーティのリクエストの上位25のリソースCDN。

RTTとTLS管理

CDNは、Webサイトのパフォーマンスのために単純なキャッシング以上のものを提供できます。多くのCDNは、コンテンツのキャッシュを禁止する法的要件またはその他のビジネス要件がある場合、動的またはパーソナライズされたコンテンツのパススルーモードもサポートします。 CDNの物理的な配布を利用すると、エンドユーザーのTCP RTTのパフォーマンスが向上します。他の人が指摘したように[383]、RTTを減らすことは、帯域幅を増やすことに比べてWebページのパフォーマンスを向上させる最も効果的な手段です[384]。

この方法でCDNを使用すると、次の2つの方法でページのパフォーマンスを改善できます。

1. TCPおよびTLSネゴシエーションのRTTを削減します。光の速度は非常に高速であり、CDNはエンドユーザーにより近い、高度に分散したデータセンターのセットを提供します。このようにして、TCP接続をネゴシエートしてTLSハンドシェイクを実行するためにパケットを通過する必要がある論理（そして物理）距離を大幅に短縮できます。

   RTTの削減には、3つの直接的な利点があります。まず、TCP + TLS接続時間はRTTにバインドされているため、ユーザーがデータを受信する時間を短縮します。第二に、これにより輻輳ウィンドウを拡大し、ユーザーが利用できる帯域幅を最大限活用するのにかかる時間が改善されます。最後に、パケット損失の可能性を減らします。 RTTが高い場合、ネットワークインタフェースは要求をタイムアウトし、パケットを再送信します。これにより、二重パケットを配信される可能性があります。

2. CDNは、バックエンドoriginへの事前に暖められたTCP接続を利用できます。ユーザーに近い接続を終了すると、輻輳ウィンドウの拡大にかかる時間が改善されるのと同様に、CDNは輻輳ウィンドウを既に最大化して事前に確立したTCP接続で要求をoriginにリレーできます。このようにして、originはより少ないTCPラウンドトリップで動的コンテンツを返すことができ、コンテンツを待機中のユーザーに配信する準備をより効果的に行うことができます。

TLSネゴシエーション時間：originはCDNの3倍遅い

TLSネゴシエーションでは、サーバーからデータを送信する前に複数のTCPラウンドトリッ

383. *https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/*
384. *https://hpbn.co/primer-on-latency-and-bandwidth/*

プが必要になるため、RTTを改善するだけでページのパフォーマンスを大幅に改善できます。たとえば、ベースHTMLページを見ると、発信元リクエストのTLSネゴシエーション時間の中央値は207ミリ秒です（デスクトップWebPageTestの場合）。これだけで、2秒のパフォーマンス予算の10%を占めます。これは、要求に遅延が適用されない理想的なネットワーク条件下です。

対照的に、大半のCDNプロバイダーのTLSネゴシエーションの中央値は60〜70ミリ秒です。HTMLページに対するOrigin要求は、CDNを使用するWebページよりも、TLSネゴシエーションを完了するのにほぼ3倍時間がかかります。 90パーセンタイルでも、140ミリ秒未満で完了するほとんどのCDNと比較して、この格差は427ミリ秒のoriginTLSネゴシエーションレートで永続します。

これらのチャートを解釈する際の注意事項：実際のTLSネゴシエーションのパフォーマンスに影響する多くの要因があるため、ベンダーを比較するとき、桁の違いに焦点を合わせることが重要です。これらのテストは、制御された条件下で単一のデータセンターから完了したものであり、インターネットおよびユーザーエクスペリエンスの変動を反映していません。

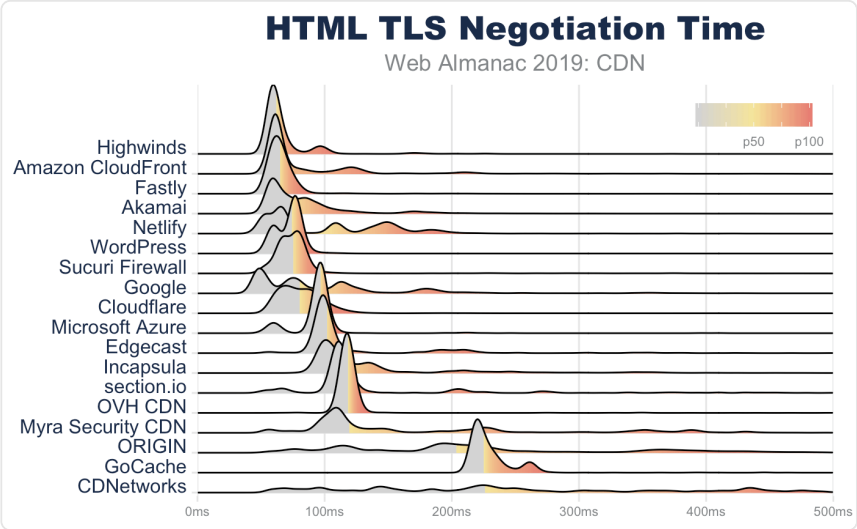


図17.8. HTML TLSネゴシエーション時間。

	p10	p25	p50	p75	p90
Highwinds	58	58	60	66	94
Fastly	56	59	63	69	75
WordPress	58	62	76	77	80
Sucuri Firewall	63	66	77	80	86
Amazon CloudFront	59	61	62	83	128
Cloudflare	62	68	80	92	103
Akamai	57	59	72	93	134
Microsoft Azure	62	93	97	98	101
Edgecast	94	97	100	110	221
Google	47	53	79	119	184
OVH CDN	114	115	118	120	122
section.io	105	108	112	120	210
Incapsula	96	100	111	139	243
Netlify	53	64	73	145	166
Myra Security CDN	95	106	118	226	365
GoCache	217	219	223	234	260
ORIGIN	100	138	207	342	427
CDNetworks	85	143	229	369	452

図17.9. HTML TLS接続時間（ミリ秒）。

リソース要求（同一ドメインおよびサードパーティを含む）の場合、TLSネゴシエーション時間が長くなり、差異が増加します。これは、ネットワークの飽和とネットワークの輻輳のためと予想されます。サードパーティの接続が確立されるまでに（リソースヒントまたはリソースリクエストにより）、ブラウザはレンダリングと他の並列リクエストの実行でビジー状態となります。これにより、ネットワーク上で競合が発生します。この欠点にもかかわらず、originソリューションを使用するよりもCDNを使用するサードパーティリソースに明らかな利点があります。

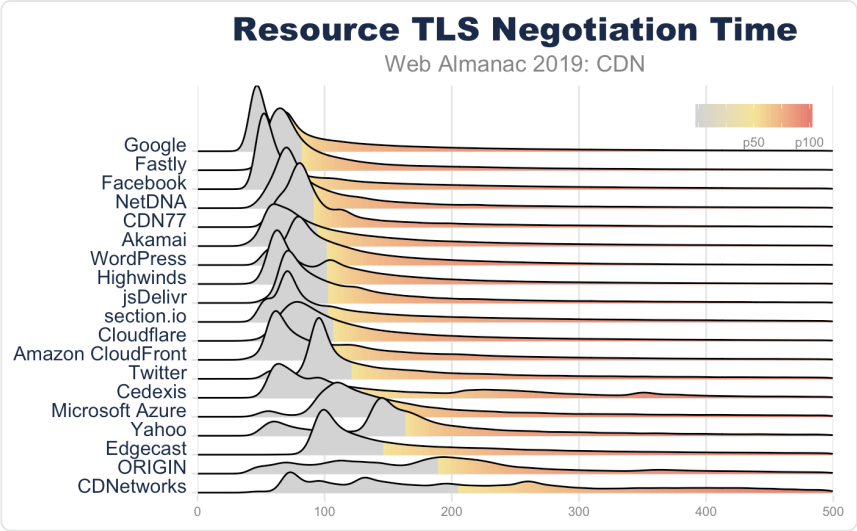


図17.10. リソースTLSネゴシエーション時間。

TLSハンドシェイクのパフォーマンスは、さまざまな要因の影響を受けます。これらには、RTT、TLSレコードサイズ、およびTLS証明書サイズが含まれます。 RTTはTLSハンドシェイクに最大の影響を与えますが、TLSパフォーマンスの2番目に大きな要因はTLS証明書のサイズです。

TLSハンドシェイク[385]の最初のラウンドトリップ中に、サーバーは証明書を添付します。この証明書は、次へ進む前にクライアントによって検証されます。この証明書交換では、サーバーは検証可能な証明書チェインを含む場合があります。この証明書の交換後、通信を暗号化するために追加のキーが確立されます。ただし、証明書の長さとサイズはTLSネゴシエーションのパフォーマンスに悪影響を与え、場合によってはクライアントライブラリをクラッシュさせる可能性があります。

証明書の交換はTLSハンドシェイクの基礎であり、通常、エクスプロイトの攻撃対象領域を最小限に抑えるため、分離されたコードパスによって処理されます。低レベルの性質のため、バッファは通常動的に割り当てられず、固定されます。この方法では、クライアントが無制限のサイズの証明書を処理できると単純に想定することはできません。たとえば、OpenSSL CLIツールとSafariは`https://10000-sans.badssl.com`に対して正常にネゴシエートできます。ただし、証明書のサイズが原因でChromeとFirefoxは失敗します。

極端なサイズの証明書は障害を引き起こす可能性がありますが、適度に大きな証明書を送信してもパフォーマンスに影響があります。証明書は、`Subject-Alternative-Name`（SAN）にリストされている1つ以上のホスト名に対して有効です。 SANが多いほど、

385. *https://hpbn.co/transport-layer-security-tls/#tls-handshake*

証明書は大きくなります。パフォーマンスの低下を引き起こすのは、検証中のこれらのSANの処理です。明確にするため、証明書サイズのパフォーマンスはTCPオーバーヘッドに関するものではなく、クライアントの処理パフォーマンスに関するものです。

技術的に、TCPスロースタートはこのネゴシエーションに影響を与える可能性がありますが、そんなことはありません。 TLSレコードの長さは16KBに制限されており、通常の初期の10の輻輳ウィンドウに適合します。一部のISPはパケットスプライサーを使用し、他のツールは輻輳ウィンドウを断片化して帯域幅を人為的に絞る場合がありますが、これはWebサイトの所有者が変更または操作できるものではありません。

ただし、多くのCDNは共有TLS証明書に依存しており、証明書のSANの多くの顧客をリストします。これはIPv4アドレスが不足しているため、必要になることがよくあります。エンドユーザーが `Server-Name-Indicator` （SNI）を採用する前は、クライアントはサーバーに接続し証明書を検査した後にのみ、クライアントはユーザーが探しているホスト名を示唆します（HTTPで `Host` ヘッダーを使用する）。これにより、IPアドレスと証明書が1：1で関連付けられます。物理的な場所が多数あるCDNの場合、各場所に専用IPが必要になる可能性があり、IPv4アドレスの枯渇をさらに悪化させます。したがって、SNIをサポートしていないユーザーがまだいるWebサイトにCDNがTLS証明書を提供する最も簡単で効率的な方法は、共有証明書を提供することです。

アカマイによると、SNIの採用はまだ世界的に100％ではありません[386]。幸いなことに、近年急速な変化がありました。最大の犯人はもはやWindows XPとVistaではなく、Androidアプリ、ボット、および企業アプリケーションです。SNI採用率が99％であっても、インターネット上の35億人のユーザーの残り1％は、Webサイトの所有者が非SNI証明書を要求する非常に魅力的な動機を生み出すことができます。別の言い方をすれば、特定製品、活動に注力してる(pure play)Webサイトは、標準ブラウザ間でほぼ100％SNIを採用できます。それでもアプリ、特にAndroidアプリでAPIまたはWebViewをサポートするためにWebサイトが使用されている場合、この分布は急速に低下する可能性があります。

ほとんどのCDNは、共有証明書の必要性とパフォーマンスのバランスをとります。ほとんどの場合、SANの数の上限は100〜150です。この制限は多くの場合、証明書プロバイダーに由来します。たとえば、LetsEncrypt[387]、DigiCert[388]、GoDaddy[389]はすべて、SAN証明書を100個のホスト名に制限しますが、Comodo[390]の制限は2,000個です。これにより、一部のCDNがこの制限を超えて、単一の証明書で800を超えるSANを使用できるようになります。 TLSパフォーマンスと証明書のSANの数には強い負の相関があります。

386. *https://datatracker.ietf.org/meeting/101/materials/slides-101-maprg-update-on-tls-sni-and-ipv6-client-adoption-00*
387. *https://letsencrypt.org/docs/rate-limits/*
388. *https://www.websecurity.digicert.com/security-topics/san-ssl-certificates*
389. *https://www.godaddy.com/web-security/multi-domain-san-ssl-certificate*
390. *https://comodosslstore.com/comodo-mdc-ssl.aspx*

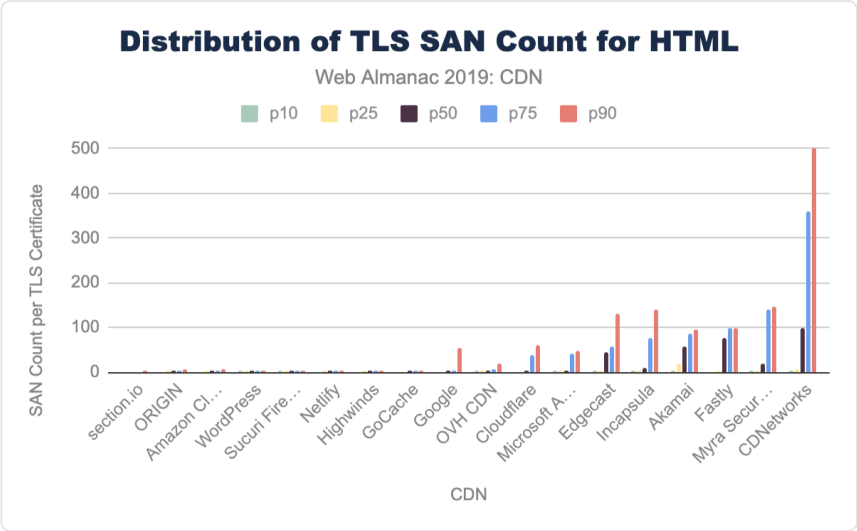


図17.11. HTMLのTLS SANカウント。

	p10	p25	p50	p75	p90
section.io	1	1	1	1	2
ORIGIN	1	2	2	2	7
Amazon CloudFront	1	2	2	2	8
WordPress	2	2	2	2	2
Sucuri Firewall	2	2	2	2	2
Netlify	1	2	2	2	3
Highwinds	1	2	2	2	2
GoCache	1	1	2	2	4
Google	1	1	2	3	53
OVH CDN	2	2	3	8	19
Cloudflare	1	1	3	39	59
Microsoft Azure	2	2	2	43	47
Edgecast	2	4	46	56	130
Incapsula	2	2	11	78	140
Akamai	2	18	57	85	95
Fastly	1	2	77	100	100
Myra Security CDN	2	2	18	139	145
CDNetworks	2	7	100	360	818

図17.12. HTMLのTLS SANカウント。

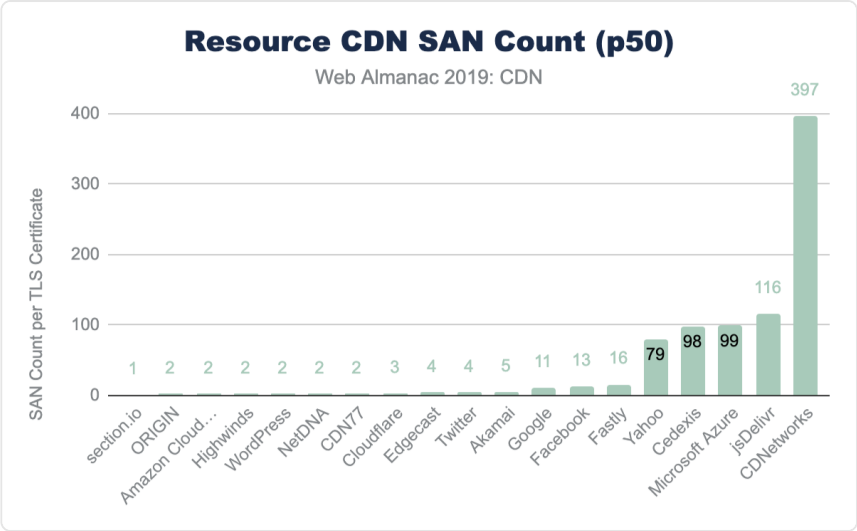


図17.13. リソースSANカウント（50パーセンタイル）。

	p10	p25	p50	p75	p90
section.io	1	1	1	1	1
ORIGIN	1	2	2	3	10
Amazon CloudFront	1	1	2	2	6
Highwinds	2	2	2	3	79
WordPress	2	2	2	2	2
NetDNA	2	2	2	2	2
CDN77	2	2	2	2	10
Cloudflare	2	3	3	3	35
Edgecast	2	4	4	4	4
Twitter	2	4	4	4	4
Akamai	2	2	5	20	54
Google	1	10	11	55	68
Facebook	13	13	13	13	13
Fastly	2	4	16	98	128
Yahoo	6	6	79	79	79
Cedexis	2	2	98	98	98
Microsoft Azure	2	43	99	99	99
jsDelivr	2	116	116	116	116
CDNetworks	132	178	397	398	645

図17.14. リソースSANカウントの分布の10、25、50、75、および90パーセンタイル。

TLSの採用

TLSおよびRTTのパフォーマンスにCDNを使用することに加えて、TLS暗号およびTLSバージョンのパッチ適用および採用を確実とするため、CDNがよく使用されます。一般に、メインHTMLページでのTLSの採用は、CDNを使用するWebサイトの方がはるかに高くなっています。 HTMLページの76%以上がTLSで提供されているのに対し、originホストページからは

62%です。

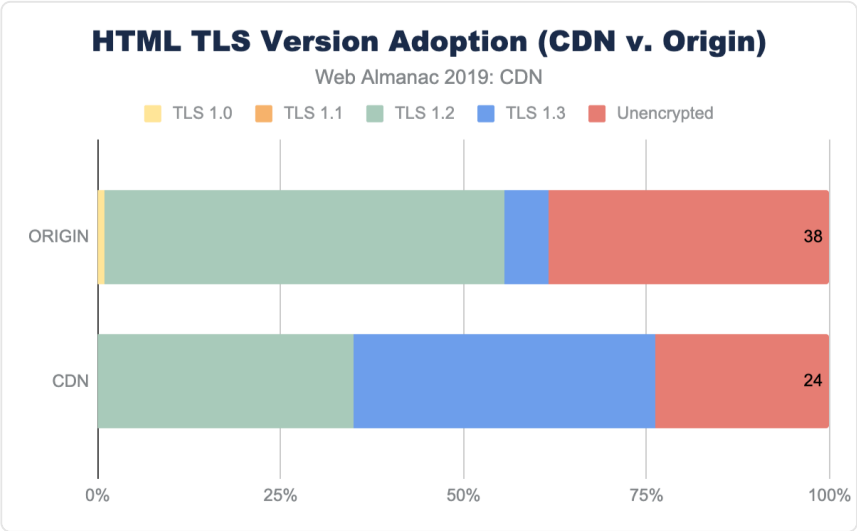


図17.15. HTML TLSバージョンの採用（CDNとorigin）。

各CDNは、TLSと提供される相対的な暗号とバージョンの両方に異なる採用率を提供します。一部のCDNはより積極的で、これらの変更をすべての顧客に展開しますが他のCDNはWebサイトの所有者に最新の変更をオプトインして、これらの暗号とバージョンを容易にする変更管理を提供することを要求します。

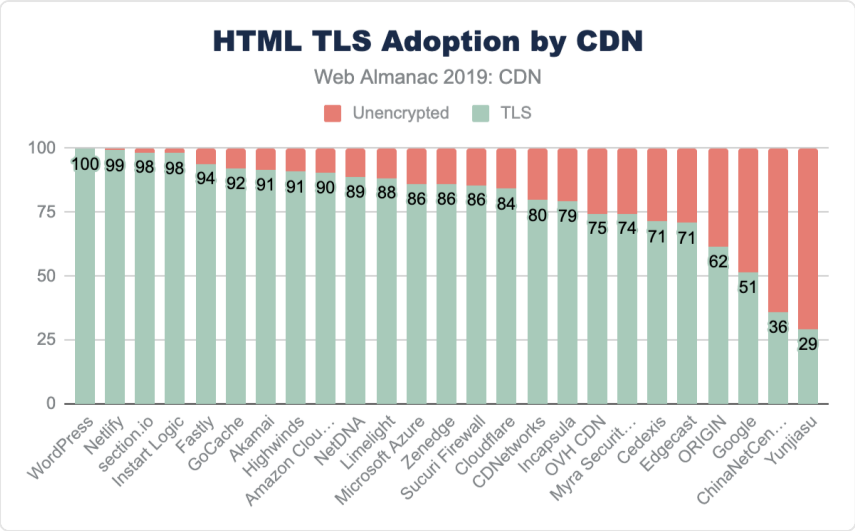


図17.16. CDNによるHTML TLSの採用。

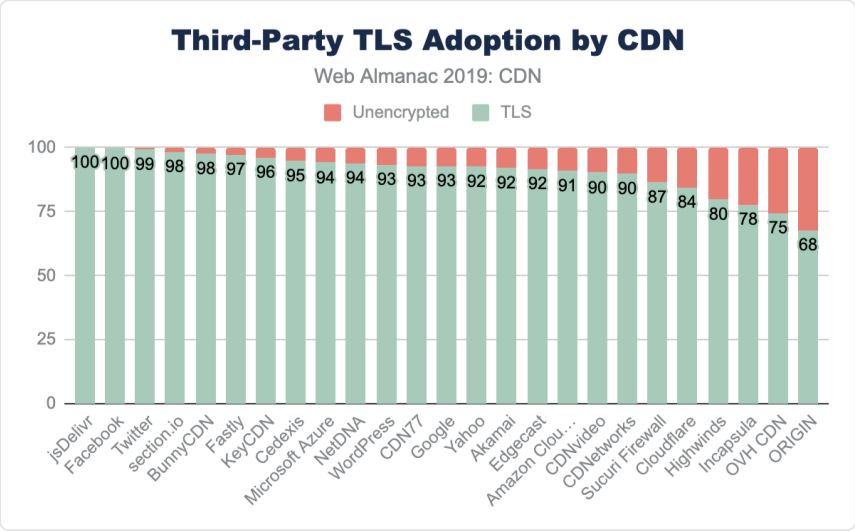


図17.17. CDNによるサードパーティTLSの採用。

このTLSの一般的な採用に加えて、CDNの使用では、TLS1.3などの新しいTLSバージョンの採用も増えています。

一般にCDNの使用は、TLS1.0のような非常に古くて侵害されたTLSバージョンの使用率が高いoriginホストサービスと比較して、強力な暗号およびTLSバージョンの迅速な採用と高い相

関があります。

Web Almanacで使用されるChromeは、ホストが提供する最新のTLSバージョンと暗号にバイアスをかけることを強調することが重要です。また、これらのWebページは2019年7月にクロールされ、新しいバージョンを有効にしたWebサイトの採用を反映しています。

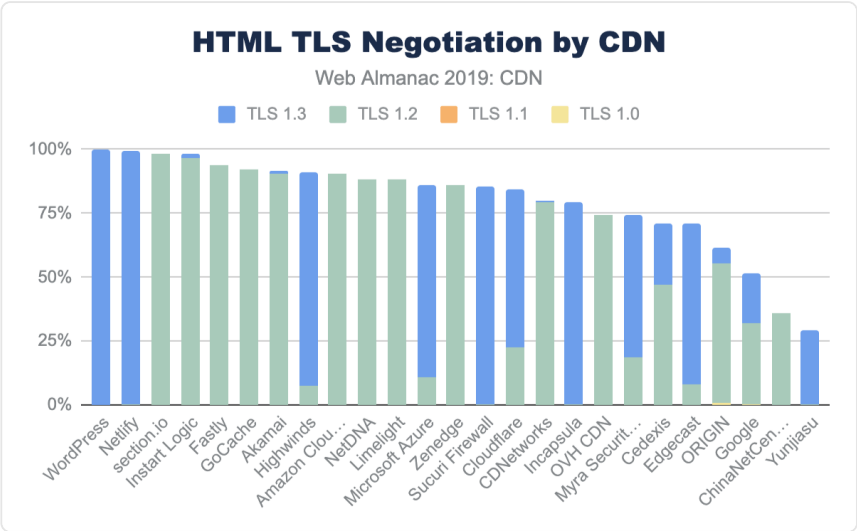


図17.18. CDNによるHTML TLSバージョン。

TLSバージョンと暗号の詳細については、セキュリティとHTTP/2の章を参照してください。

HTTP/2採用

RTT管理とTLSパフォーマンスの向上に加えて、CDNはHTTP/2やIPv6などの新しい標準も有効にします。ほとんどのCDNはHTTP/2のサポートを提供し、多くはまだ標準以下の開発HTTP/3の早期サポートを示していますが、これらの新機能を有効にするかどうかは依然としてWebサイト所有者に依存しています。変更管理のオーバーヘッドにもかかわらず、CDNから提供されるHTMLの大部分ではHTTP/2が有効になっています。

CDNのHTTP/2の採用率は70%を超えていますが、originページはほぼ27%です。同様にCDNのサブドメインリソースとサードパーティリソースでは90%以上がHTTP/2を採用していて、さらに高くなりますが、originインフラストラクチャから提供されるサードパーティリソースは31%しか採用されていません。 HTTP/2 のパフォーマンス向上およびその他の機能については、HTTP/2の章でさらに説明します。

注：すべてのリクエストは、HTTP/2をサポートするChromeの最新バージョンで行われました。 HTTP/1.1のみが報告される場合、これは暗号化されていない（非TLS）サーバーまたは

HTTP/2をサポートしないサーバーを示します。

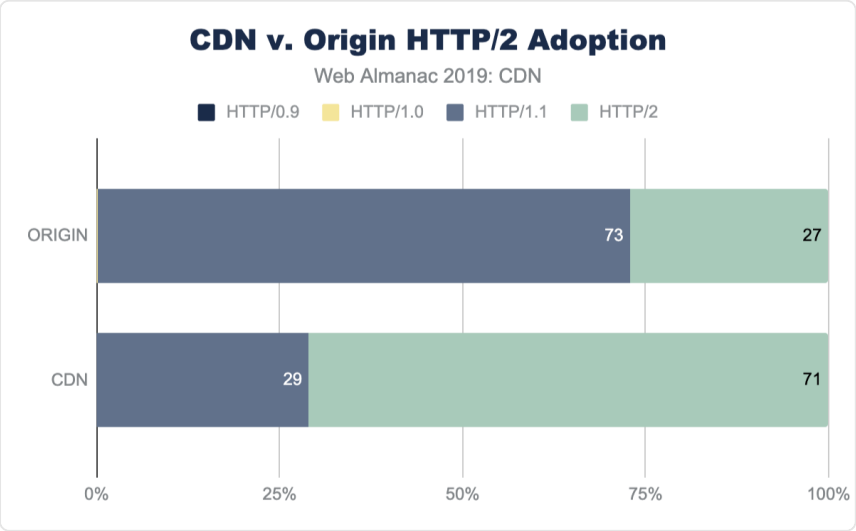


図17.19. HTTP/2の採用（CDNとorigin）。

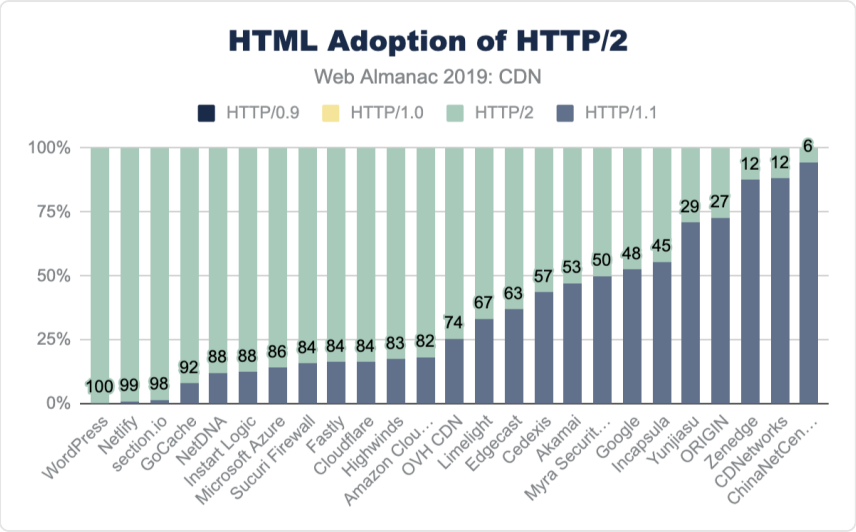


図17.20. HTTP/2のHTML採用。

	HTTP/0.9	HTTP/1.0	HTTP/1.1	HTTP/2
WordPress	0	0	0.38	100
Netlify	0	0	1.07	99
section.io	0	0	1.56	98
GoCache	0	0	7.97	92
NetDNA	0	0	12.03	88
Instart Logic	0	0	12.36	88
Microsoft Azure	0	0	14.06	86
Sucuri Firewall	0	0	15.65	84
Fastly	0	0	16.34	84
Cloudflare	0	0	16.43	84
Highwinds	0	0	17.34	83
Amazon CloudFront	0	0	18.19	82
OVH CDN	0	0	25.53	74
Limelight	0	0	33.16	67
Edgecast	0	0	37.04	63
Cedexis	0	0	43.44	57
Akamai	0	0	47.17	53
Myra Security CDN	0	0.06	50.05	50
Google	0	0	52.45	48
Incapsula	0	0.01	55.41	45
Yunjiasu	0	0	70.96	29
ORIGIN	0	0.1	72.81	27
Zenedge	0	0	87.54	12
CDNetworks	0	0	88.21	12
ChinaNetCenter	0	0	94.49	6

図17.21. CDNによるHTTP/2のHTML採用。

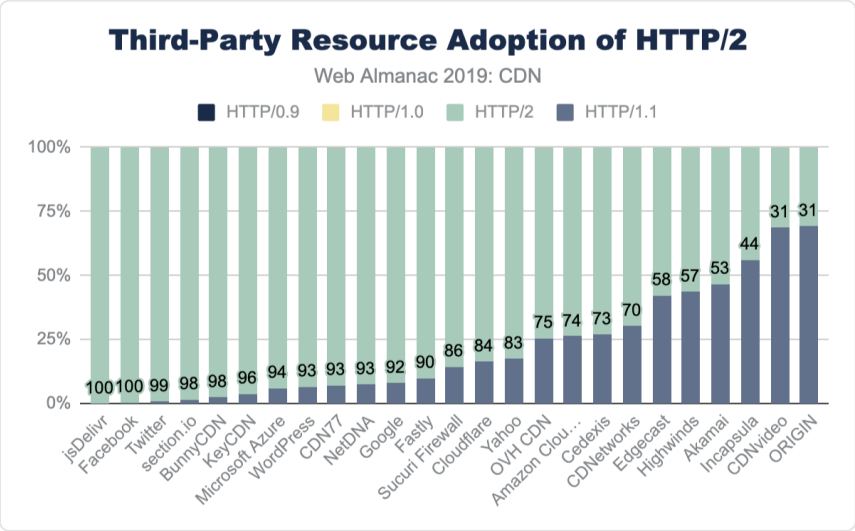


図17.22. HTML/2の採用：サードパーティのリソース。

cdn	HTTP/0.9	HTTP/1.0	HTTP/1.1	HTTP/2
jsDelivr	0	0	0	100
Facebook	0	0	0	100
Twitter	0	0	1	99
section.io	0	0	2	98
BunnyCDN	0	0	2	98
KeyCDN	0	0	4	96
Microsoft Azure	0	0	6	94
WordPress	0	0	7	93
CDN77	0	0	7	93
NetDNA	0	0	7	93
Google	0	0	8	92
Fastly	0	0	10	90
Sucuri Firewall	0	0	14	86
Cloudflare	0	0	16	84
Yahoo	0	0	17	83
OVH CDN	0	0	26	75
Amazon CloudFront	0	0	26	74
Cedexis	0	0	27	73
CDNetworks	0	0	30	70
Edgecast	0	0	42	58
Highwinds	0	0	43	57
Akamai	0	0.01	47	53
Incapsula	0	0	56	44
CDNvideo	0	0	68	31
ORIGIN	0	0.07	69	31

図17.23. HTML/2の採用：サードパーティのリソース。

CDNキャッシュ動作の制御

`Vary`

Webサイトは、さまざまなHTTPヘッダーを使用して、ブラウザーとCDNのキャッシュ動作を制御できます。 最も一般的なのは、最新のものであることを保証するためにoriginへ戻る前に何かをキャッシュできる期間を具体的に決定する `Cache-Control` ヘッダーです。

別の便利なツールは、 `Vary` HTTPヘッダーの使用です。このヘッダーは、キャッシュをフラグメント化する方法をCDNとブラウザーの両方に指示します。 `Vary` ヘッダーにより、originはリソースの表現が複数あることを示すことができ、CDNは各バリエーションを個別にキャッシュする必要があります。最も一般的な例は圧縮です。リソースを `Vary : Accept-Encoding` を使用すると、CDNは同じコンテンツを、非圧縮、Gzip、Brotliなどの異なる形式でキャッシュできます。一部のCDNでは、使用可能なコピーを1つだけ保持するために、この圧縮を急いで実行します。同様に、この `Vary` ヘッダーは、コンテンツをキャッシュする方法と新しいコンテンツを要求するタイミングをブラウザーに指示します。

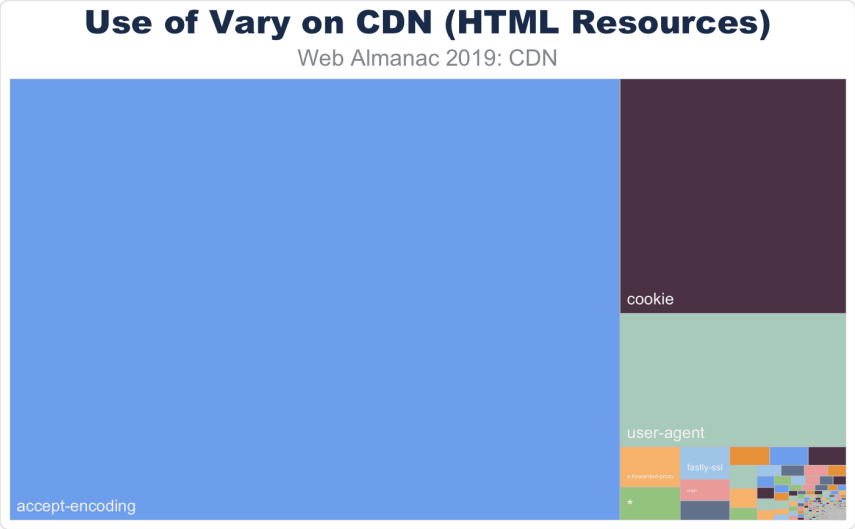


図17.24. CDNから提供されるHTMLの `Vary` の使用法。

`Vary` の主な用途は `Content-Encoding` の調整ですが、Webサイトがキャッシュの断片化を知らせるために使用する他の重要なバリエーションがあります。 `Vary` を使用すると、

DuckDuckGo、Google、BingBotなどのSEOボットに、異なる条件下で代替コンテンツが返されるように指示します。これは、「クローキング」（ランキングを戦うためにSEO固有のコンテンツを送信する）に対するSEOペナルティを回避するために重要でした。

HTMLページの場合、`Vary`の最も一般的な使用法は、`User-Agent`に基づいてコンテンツが変更されることを通知することです。これは、Webサイトがデスクトップ、電話、タブレット、およびリンク展開エンジン（Slack、iMessage、Whatsappなど）に対して異なるコンテンツを返すことを示す略記です。`Vary : User-Agent`の使用は、コンテンツがバックエンドの「mDot」サーバーと「通常」サーバーに分割された初期モバイル時代の名残でもあります。レスポンシブWebの採用が広く知られるようになりましたが、この`Vary`形式は残ります。

同様に、`Vary : Cookie`は通常、ユーザーのログイン状態またはその他のパーソナライズに基づいてコンテンツが変化することを示します。

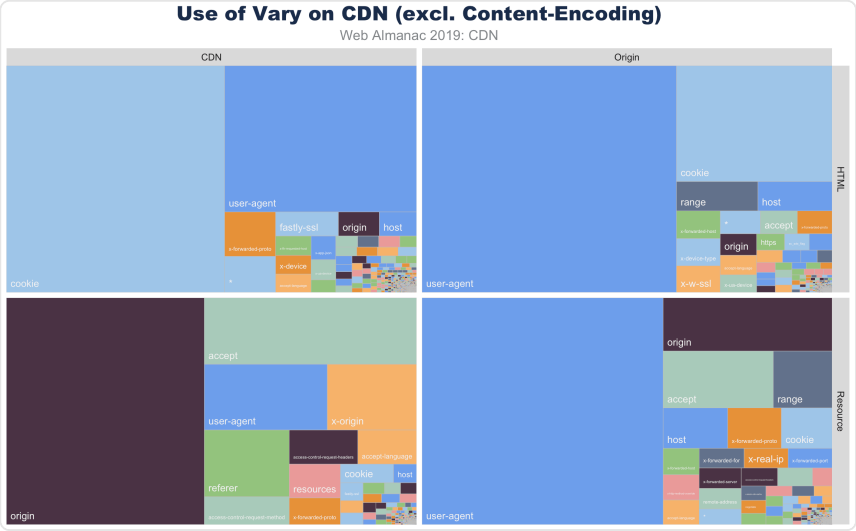


図17.25. HTMLとoriginとCDNから提供されるリソースの`Vary`使用の比較。

対照的に、リソースはHTMLリソースほど`Vary : Cookie`を使用しません。代わりに、これらのリソースは`Accept`、`Origin`、または`Referer`に基づいて適応する可能性が高くなります。たとえば、ほとんどのメディアは、`Vary : Accept`を使用してブラウザが提供する`Accept`ヘッダーに応じて画像がJPEG、WebP、JPEG 2000、またはJPEG XRであることを示します。同様に、サードパーティの共有リソースは、埋め込まれているWebサイトによってXHR APIが異なることを通知します。このように、広告サーバーAPIの呼び出しは、APIを呼び出した親Webサイトに応じて異なるコンテンツを返します。

`Vary`ヘッダーには、CDNチェインの証拠も含まれています。これらは、`Accept-Encoding`、`Accept-Encoding`、または`Accept-Encoding`、`Accept-Encoding`、`Accept-Encoding`などの`Vary`ヘッダーで確認できます。これらのチェインと`Via`ヘッダーエントリをさらに分析すると、たとえば、サードパーティタグをプロキシしているサイトの数など興味深いデータが明らかになる可能性があります。

`Vary`の使用の多くは無関係です。ほとんどのブラウザがダブルキーキャッシングを採用しているため、`Vary : Origin`の使用は冗長です。`Vary : Range`または`Vary : Host`または`Vary : *`のように。`Vary`のワイルドで可変的な使用は、インターネットが奇妙であることの実証可能な証拠です。

`Surrogate-Control`, `s-maxage`, `Pre-Check`

`Cache-Control`ヘッダーの`Surrogate-Control`、`s-maxage`、`pre-check`、`post-check`の値など、特にCDNまたは他のプロキシキャッシュを対象とする他のHTTPヘッダーがあります。一般的に、これらのヘッダーを使う事は少ないでしょう。

`Surrogate-Control`を使用すると、originはCDNに対してのみキャッシュルールを指定できます。CDNは応答を提供する前にヘッダーを削除する可能性が高いため、使用量が低いと驚くことはありません（いくつかのCDNもヘッダーを削除しているように見えました）。

一部のCDNは、リソースが古くなった場合にリソースを更新できるようにする方法として`pre-check`をサポートし、最大値`maxage`として`pre-check`をサポートしています。ほとんどのCDNでは、`pre-check`と`post-check`の使用率は1%未満でした。Yahoo!はこの例外であり、リクエストの約15%に`pre-check = 0`、`post-check = 0`がありました。残念ながら、これは積極的な使用ではなく、古いInternet Explorerパターンの名残です。この上のより多くの議論では、キャッシングの章に記載されています。

`s-maxage`ディレクティブは、応答をキャッシュできる期間をプロキシに通知します。Web Almanacデータセット全体で、jsDelivrは複数のリソースで高いレベルの使用が見られた唯一のCDNです。これは、jsDelivrのライブラリのパブリックCDNとしての役割を考えると驚くことではありません。他のCDNでの使用は、個々の顧客、たとえばその特定のCDNを使用するサードパーティのスクリプトまたはSaaSプロバイダーによって推進されているようです。

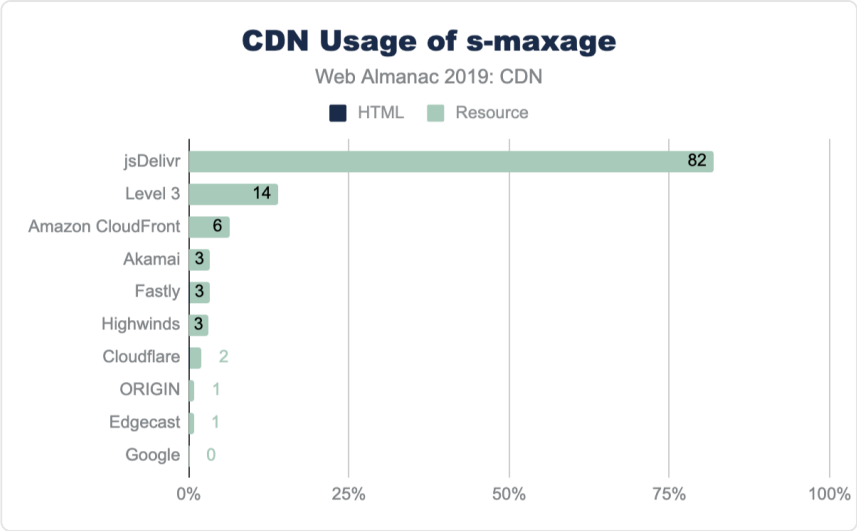


図17.26. CDN応答全体での `s-maxage` の採用。

サイトの40%がリソースにCDNを使用しており、これらのリソースが静的でキャッシュ可能であると仮定すると、`s-maxage`の使用は低いようです。

今後の研究では、キャッシュの有効期間とリソースの経過時間、および`s-maxage`の使用法と`stale-while-revalidate`などの他の検証ディレクティブの使用法を検討する可能性があります。

一般的なライブラリとコンテンツのCDN

これまでのところ、この章ではサイトが独自のコンテンツをホストするために使用している可能性のあるコマーシャルCDNの使用、またはサイトに含まれるサードパーティリソースによって使用されている可能性について検討しました。

jQueryやBootstrapなどの一般的なライブラリは、Google、Cloudflare、Microsoftなどがホストするパブリック CDNからも利用できます。コンテンツを自己ホストする代わりに、パブリックCDNの1つのコンテンツを使用することはトレードオフです。コンテンツがCDNでホストされている場合でも、新しい接続を作成して輻輳ウィンドウを拡大すると、CDNを使用する際の低遅延が無効になる場合があります。

Googleフォントはコンテンツ CDNの中で最も人気があり、55%のWebサイトで使用されています。非フォントコンテンツの場合、Google API、CloudflareのJS CDN、およびBootstrapのCDNが次に人気です。

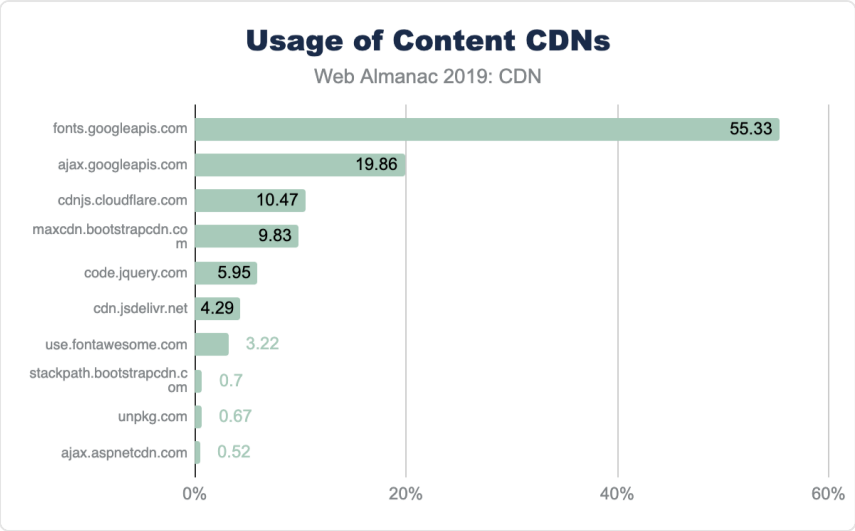


図17.27. パブリックコンテンツCDNの使用。

分割キャッシュを実装するブラウザが増えると、共通ライブラリをホストするためのパブリックCDNの有効性が低下し、この研究の今後の反復で人気が低くなるかどうかを見るのは興味深いでしょう。

結論

CDN配信によるレイテンシーの短縮と、訪問者の近くにコンテンツを保存する機能により、サイトはoriginの負荷を軽減しながらより高速な体験を提供できます。

Steve SoudersによるCDNの使用の推奨は、12年前と同じように今日でも有効ですがCDNを介してHTMLコンテンツを提供しているサイトは20%のみであり、リソースにCDNを使用しているサイトは40%のみです。それらの使用法はさらに成長します。

この分析に含まれていないCDNの採用にはいくつかの側面があります、これはデータセットの制限と収集方法が原因である場合で、他の場合は分析中に新しい研究の質問が出てきました。

Webの進化に伴い、CDNベンダーは革新しサイトの新しいプラクティスを使用します、CDNの採用はWeb Almanacの将来のエディションでのさらなる研究のために豊富な領域のままです。

著者

Andy Davies

@AndyDavies　andydavies　http://andydavies.me/

Andy Daviesはフリーランスのウェブパフォーマンスコンサルタントであり、英国の大手小売業者、新聞社、金融サービス会社のサイトの高速化を支援してきました。彼は『ウェブパフォーマンスのポケットガイド』を執筆し、『Using WebPageTest』の共著者であり、ロンドンのウェブパフォーマンス・ミートアップの主催者でもあります。アンディは Twitter で @AndyDavies として活動しており、https://andydavies.me で時々ブログを更新しています。

Colin Bendell

@colinbendell　colinbendell

Colinは、Cloudinary[391]のCTOオフィスの一員であり、オライリーの本High Performance Images[392]の共著者でもあります。彼は、大容量データ、メディア、ブラウザ、標準の交差点で多くの時間を過ごしています。@colinbendell や https://bendell.ca のブログで彼を見つけることができます。

391. https://cloudinary.com/
392. https://www.oreilly.com/library/view/high-performance-images/9781491925799/

部 IV 章 18
Page Weight

Tammy Everts と Katie Hempenius によって書かれた。
Paul Calvano によってレビュ。
Katie Hempenius による分析。
David Fox 編集。
Sakae Kotaro によって翻訳された。

序章

中央Webページのサイズは約1900KBで、74のリクエストが含まれています。悪くないですね。

ここに中央値の問題があります：それらは問題を隠します。定義上、それらは分布の中間にのみ焦点を合わせています。全体像を理解するには、両極端のパーセンタイルを考慮する必要があります。

90パーセンタイルを見ると、不快なものが明らかになります。疑いを持たない人々に向けてプッシュしているページのおよそ10%は6MBを超えており、179のリクエストが含まれています。これは、率直に言ってひどいです。もしあなたがひどくないと思うのであれば、間違いなくこの章を読む必要があります。

神話：ページサイズは関係ない

ページサイズが重要ではなくなった理由に関する一般的な論点は、高速インターネットと強化されたデバイスのおかげで、大規模で複雑な（そして非常に複雑な）ページを一般の人々に提供できるということです。この仮定は、高速インターネットや強化されたデバイスでアクセスできない巨大なインターネットユーザー層を無視しても問題ない限り、うまく機能します。

はい。一部のユーザーにとっては、高速で堅牢なページを構築できます。ただし、すべてのユーザー、特に帯域幅の制約やデータ制限へ対処するモバイル専用ユーザーにどのように影響するかという観点からページの肥大化に注意する必要があります。

Tim Kadlecの魅力的なオンライン計算機、[What Does My Site Cost?]（https://whatdoesmysitecost.com/）をチェックしてください。これは、世界中の国のページのコスト（1人あたりのドルと国民総所得）を計算します。それは目を見張るものです。たとえば、執筆時点で2.79MBのAmazonのホームページの費用は、モーリタニアの1人当たりGNIの1日あたり1.89％です。世界のいくつかの地域の人々が数十ページを訪問するだけで一日の賃金をあきらめなければならないとき、ワールドワイドウェブはどれほどグローバルなのでしょうか？

帯域幅を増やすことは、Webパフォーマンスの魔法の弾丸ではありません

より多くの人がより良いデバイスとより安価な接続にアクセスできたとしても、それは完全なソリューションではありません。帯域幅を2倍にしても、2倍速くなるわけではありません。実際、帯域幅を最大1,233％増やすと、ページが55％速くなるだけであることが実証[393]されています。

問題は遅延です。私たちのネットワークプロトコルのほとんどは、多くの往復を必要とし、それらの各往復は遅延ペナルティを課します。遅延がパフォーマンスの問題である限り（つまり、近い将来）パフォーマンスの主な原因は、今日の典型的なWebページには数十の異なるサーバーでホストされている100程度のアセットが含まれていることです。これらのアセットの多くは、最適化されておらず、測定と監視がされていないため予測不能です。

HTTP Archiveはどのような種類のアセットを追跡し、どの程度の重要性を持ちますか？

HTTP Archiveが追跡するページ構成メトリックの簡単な用語集と、パフォーマンスとユーザーエクスペリエンスの観点から重要なメトリックを以下に示します。

393. *https://developer.akamai.com/blog/2015/06/09/heres-why-more-bandwidth-isnt-magic-bullet-web-performance*

- **合計サイズ**は、ページのバイト単位の合計重量です。特に、限られたデータや測定データがあるモバイルユーザーにとって重要です。

- 通常、**HTML**はページ上の最小のリソースです。そのパフォーマンスリスクはごくわずかです。

- 多くの場合、最適化されていない**画像**がページの肥大化の最大の原因です。ページの重さの分布の90パーセンタイルを見ると、約7MBのページの5.2MBを画像が占めています。つまり画像は総ページ重量のほぼ75%を占めます。そして、それだけでは不十分な場合、ページ上の画像の数は、小売サイトでのコンバージョン率の低下につながります（これについては後で詳しく説明します）。

- **JavaScript**が重要です。ページのJavaScriptの重さは比較的小さい場合がありますが、それでもJavaScriptによるパフォーマンスの問題が生じます。単一の100KBのサードパーティスクリプトでさえ、ページに大損害を与える可能性があります。ページ上のスクリプトが多いほど、リスクは大きくなります。

  JavaScriptのブロックだけに集中するだけでは十分でありません。 JavaScriptのレンダリング方法により、ページにブロッキングリソースが含まれていなくても、パフォーマンスが最適とは言えない可能性があります。 JavaScriptが他のすべてのブラウザーアクティビティを組み合わせた場合よりも多くのCPUを消費するため、ページ上のCPU使用率を理解することが非常に重要です。 JavaScriptがCPUをブロックしますてる間、ブラウザーはユーザー入力に応答できません。これにより、一般に「ジャンク」と呼ばれるものが作成されます。これは不安定なページレンダリングの不快な感覚です。

- **CSS**は、現代のWebページにとって信じられないほどの恩恵です。ブラウザの互換性から設計の保守と更新まで、無数の設計上の問題を解決します。 CSSがなければ、レスポンシブデザインのような素晴らしいものはありません。しかし、JavaScriptのように、CSSは問題を引き起こすためにかさばる必要はありません。スタイルシートの実行が不十分な場合、ダウンロードと解析に時間がかかりすぎるスタイルシート、ページの残りの部分のレンダリングをブロックする不適切に配置されたスタイルシートに至るまで、パフォーマンスの問題が多数発生する可能性はあります。またJavaScriptと同様にCSSファイルが多くなると潜在的な問題が発生します。

大きくて複雑なページはビジネスに悪い場合があります

あなたが、あなたのサイト訪問者を気にしない心無いモンスターでないと仮定しましょう。しかしあなたがそうであれば、より大きく、より複雑なページを提供することもあなたを傷つけることを知っておくべきです。これは、小売サイトから100万以上のビーコンに相当する実際のユーザーデータを収集したGoogle主導の機械学習[394]の調査結果の1つでした。

この研究から、3つの重要なポイントがありました。

1. **ページ上の要素の総数は、コンバージョンの最大の予測因子でした。** 最新のWebページを構成するさまざまなアセットによって課されるパフォーマンスリスクについて説明したことを考えると、これが大きな驚きにならないことを願っています。

2. ページ上の画像の数は、コンバージョンの2番目に大きな予測因子でした。ユーザーが変換したセッションでは、変換しなかったセッションよりも画像が38%少なくなりました。

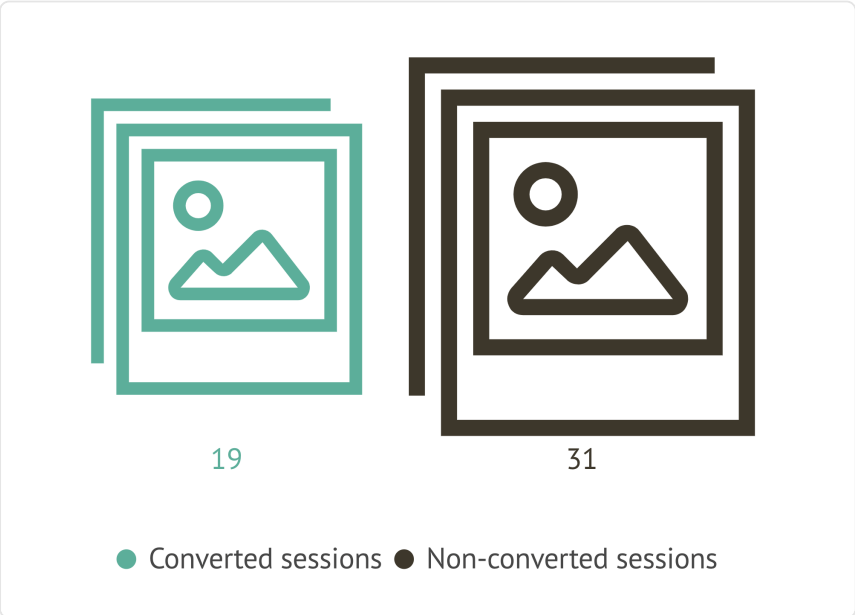


図18.1. 変換されたセッションと変換されないセッション。

3. **スクリプトが多いセッションは、変換される可能性が低くなりました。** このグラ

394. *https://www.thinkwithgoogle.com/marketing-resources/experience-design/mobile-page-speed-load-time/*

フで本当に魅力的なのは、約240個のスクリプトを実行した後の変換確率の急激な低下だけではありません。最大1,440個のスクリプトが含まれる小売セッションの数を示すのはロングテールです！

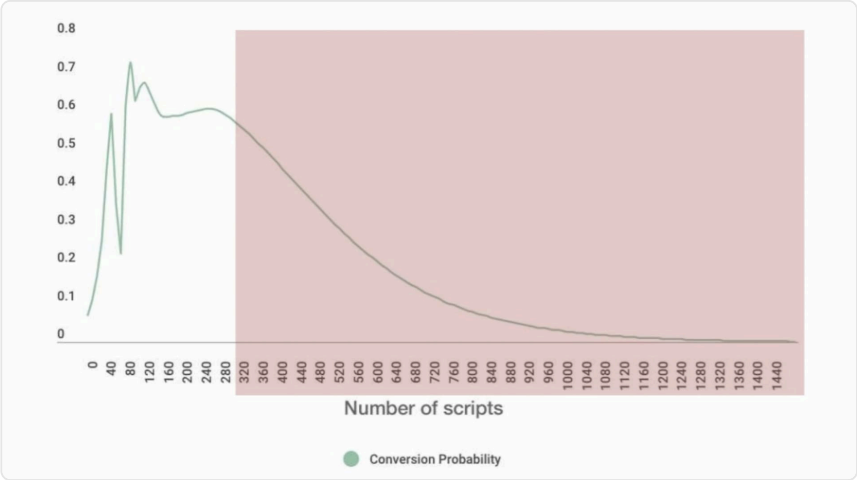


図18.2. スクリプトが増加すると変換率は低下します。

ページサイズと複雑さが重要である理由について説明したので、Webの現在の状態とページの肥大化の影響をよりよく理解できるように、ジューシーなHTTP Archiveの統計を見てみましょう。

分析

このセクションの統計はすべて、ページとそのリソースの転送サイズに基づいています。Web上のすべてのリソースが送信前に圧縮されるわけではありませんが、圧縮されている場合、この分析では圧縮サイズが使用されます。

ページの重さ

大まかに言って、モバイルサイトはデスクトップの対応サイトよりも約10%小さくなっています。違いの大部分は、モバイルサイトが対応するデスクトップよりも少ない画像バイトを読み込んでいるためです。

モバイル

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	6226	107	1060	234	4746	49
75	3431	56	668	122	2270	25
50	1745	26	360	56	893	13
25	800	11	164	22	266	7
10	318	6	65	5	59	4

図18.3. リソースタイプ別のモバイルのページウェイト。

デスクトップ

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	6945	110	1131	240	5220	52
75	3774	58	721	129	2434	26
50	1934	27	391	62	983	14
25	924	12	186	26	319	8
10	397	6	76	8	78	4

図18.4. リソースタイプ別に分類されたデスクトップ上のページの重み

時間と共に変化するページの重さ

過去1年間に、デスクトップサイトのサイズの中央値は434KB増加し、モバイルサイトのサイズの中央値は179KB増加しました。画像はこの増加を圧倒的に促進しています。

モバイル

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	+376	-50	+46	+36	+648	+2
75	+304	-7	+34	+21	+281	0
50	+179	-1	+27	+10	+106	0
25	+110	-1	+16	+5	+36	0
10	+72	0	+13	+2	+20	+1

図18.5. 2018年以降のモバイルページのウェイトの変化。

デスクトップ

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	+1106	-75	+22	+45	+1291	+5
75	+795	-12	+9	+32	+686	+1
50	+434	-1	+10	+15	+336	0
25	+237	0	+12	+7	+138	0
10	+120	0	+10	+2	+39	+1

図18.6. 2018年以降のデスクトップページの重みの変化。

ページの重さが時間とともにどのように変化するかについての長期的な視点については、HTTP Archiveからこの時系列グラフ[395]をご覧ください。ページサイズの中央値は、HTTP Archiveが2010年11月にこのメトリックの追跡を開始して以来ほぼ一定の割合で成長しており、過去1年間に見られたページウェイトの増加はこれと一致しています。

ページリクエスト

デスクトップページの中央値は74リクエストで、モバイルページの中央値は69リクエスト

395. https://httparchive.org/reports/page-weight#bytesTotal

です。これらのリクエストの大部分は画像とJavaScriptアカウントです。昨年、リクエストの量や分布に大きな変化はありませんでした。

モバイル

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	168	15	52	20	79	7
75	111	7	32	12	49	2
50	69	3	18	6	28	0
25	40	2	9	3	15	0
10	22	1	4	1	7	0

図18.7. リソースタイプ別に分類されたモバイルページリクエスト。

デスクトップ

パーセンタイル	合計 (KB)	HTML (KB)	JS (KB)	CSS (KB)	画像 (KB)	ドキュメント (KB)
90	179	14	53	20	90	6
75	118	7	33	12	54	2
50	74	4	19	6	31	0
25	44	2	10	3	16	0
10	24	1	4	1	7	0

図18.8. リソースタイプ別に分類されたデスクトップページリクエスト。

ファイル形式

前述の分析では、リソースタイプのレンズを通してページの重さを分析することに焦点を当ててきました。ただし、画像とメディアの場合、特定のファイル形式間のリソースサイズの違いを調べて、さらに深く掘り下げることができます。

画像形式によるファイルサイズ（モバイル）

パーセンタイル	GIF (KB)	ICO (KB)	JPG (KB)	PNG (KB)	SVG (KB)	WEBP (KB)
10	0	0	3.08	0.37	0.25	2.54
25	0.03	0.26	7.96	1.14	0.43	4.89
50	0.04	1.12	21	4.31	0.88	13
75	0.06	2.72	63	22	2.41	33
90	2.65	13	155	90	7.91	78

図18.9. モバイルの画像ファイルサイズを画像形式別に分類したもの。

これらの結果の一部、特にGIFの結果は、本当に驚くべきものです。 GIFが非常に小さい場合、なぜそれらはJPG、PNG、およびWEBPなどの形式に置き換えられるのですか？

上記のデータは、Web上のGIFの大部分が実際には小さな1×1ピクセルであるという事実を覆い隠しています。これらのピクセルは通常「トラッキングピクセル」として使用されますが、さまざまなCSS効果を生成するためのハックとしても使用できます。これらの1×1ピクセルは文字通りのイメージですが、その使用の精神はおそらくスクリプトまたはCSSに関連付けるものと近いでしょう。

データセットをさらに調査すると、GIFの62%が43バイト以下（43バイトは透明な1×1ピクセルGIFのサイズ）であり、GIFの84%は1KB以下であることが明らかになりました。

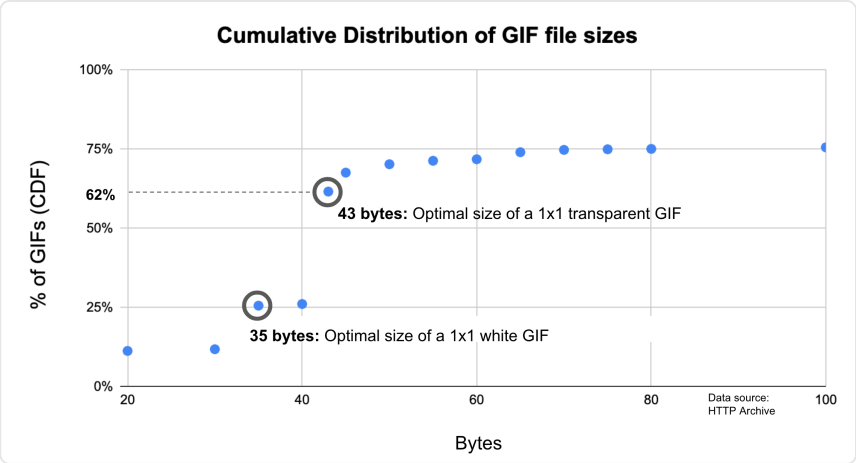


図18.10. GIFファイルサイズの累積分布関数。

以下の表は、これらの小さな画像をデータセットから削除するための2つの異なるアプローチを示しています。最初の方法は、ファイルサイズが100バイトを超える画像に基づいており、2番目はファイルサイズが1024バイトを超える画像に基づいています。

画像の画像形式ごとのファイルサイズ> 100バイト

パーセンタイル	GIF (KB)	ICO (KB)	JPG (KB)	PNG (KB)	SVG (KB)	WEBP (KB)
10	0.27	0.31	3.08	0.4	0.28	2.1
25	0.75	0.6	7.7	1.17	0.46	4.4
50	2.14	1.12	20.47	4.35	0.95	11.54
75	7.34	4.19	61.13	21.39	2.67	31.21
90	35	14.73	155.46	91.02	8.26	76.43

図18.11. 100バイトを超える画像の画像形式ごとのファイルサイズ。

画像の画像形式ごとのファイルサイズ > 1024バイト

パーセンタイル	GIF (KB)	ICO (KB)	JPG (KB)	PNG (KB)	SVG (KB)	WEBP (KB)
10	1.28	1.12	3.4	1.5	1.2	3.08
25	1.9	1.12	8.21	2.88	1.52	5
50	4.01	2.49	21.19	8.33	2.81	12.52
75	11.92	7.87	62.54	33.17	6.88	32.83
90	67.15	22.13	157.96	127.15	19.06	79.53

図18.12. 1024バイトを超える画像の画像形式ごとのファイルサイズ。

JPEG画像に比べてPNG画像のファイルサイズが小さいことは驚くべきことです。 JPEGは非可逆圧縮[396]を使用します。非可逆圧縮によりデータが失われるため、ファイルサイズを小さくできます。一方、PNGは可逆圧縮[397]を使用します。これによりデータが失われることはありません。これにより、より高品質で大きな画像が生成されます。ただし、このファイルサイズの違いはエンコーディングと圧縮の違いではなく、透過性のサポートによるアイコンの

396. https://ja.wikipedia.org/wiki/%E9%9D%9E%E5%8F%AF%E9%80%86%E5%9C%A7%E7%B8%AE
397. https://ja.wikipedia.org/wiki/%E5%8F%AF%E9%80%86%E5%9C%A7%E7%B8%AE

グラフィックのPNGの人気を反映している可能性があります。

メディア形式ごとのファイルサイズ

MP4は、今日のWebで圧倒的に最も人気のあるビデオ形式です。人気の点では、それぞれWebMとMPEG-TSが続きます。

このデータセットの他のテーブルの一部とは異なり、このテーブルにはほとんど満足のいく結果があります。動画はモバイルでは常に小さく表示されるのですばらしいです。さらに、MP4ビデオのサイズの中央値は、モバイルでは18KB、デスクトップでは39KBと非常に合理的です。 WebMの数値の中央値はさらに優れていますが、一度見てください。複数のクライアントとパーセンタイルでの0.29KBの重複測定は少し疑わしいです。考えられる説明の1つは、非常に小さなWebMビデオの同一のコピーが多くのページに含まれていることです。 3つの形式のうち、MPEG-TSは常にすべてのパーセンタイルで最高のファイルサイズを持っています。これは1995年にリリースされたという事実に関連している可能性があり、これらの3つのメディア形式の中で最も古いものになっています。

モバイル

パーセンタイル	MP4 (KB)	WebM (KB)	MPEG-TS (KB)
10	0.89	0.29	0.01
25	2.07	0.29	55
50	18	1.44	153
75	202	223	278
90	928	390	475

図18.13. モバイルのメディア形式によるビデオサイズ。

デスクトップ

パーセンタイル	MP4 (KB)	WebM (KB)	MPEG-TS (KB)
10	0.27	0.29	34
25	1.05	0.29	121
50	39	17	286
75	514	288	476
90	2142	896	756

図18.14. デスクトップ上のメディア形式によるビデオサイズ。

結論

過去1年間で、ページのサイズは約10%増加しました。 Brotli、パフォーマンスバジェット、および基本的な画像最適化のベストプラクティスは、おそらくページウェイトを維持または改善すると同時に広く適用可能で実装が非常に簡単な3つのテクニックです。そうは言っても、近年ではページの重さの改善は、テクノロジー自体よりもベストプラクティスの採用が少ないことにより制約されています。言い換えれば、ページの重さを改善するための多くの既存のテクニックがありますが、それらが使用されなければ違いはありません。

著者

Tammy Everts

@tameverts tammyeverts https://speedcurve.com/

Tammy Evertsは、20年以上にわたってユーザビリティとUXを研究してきました。過去10年間、彼女はウェブパフォーマンスとビジネスのUX交差に焦点を当ててきました。彼女は、SpeedCurve[398]のCXOであり、performance.now()カンファレンス[399]の共同議長であり、O'Reillyの本の著者*時は金なり。パフォーマンスのビジネス価値。*[400].

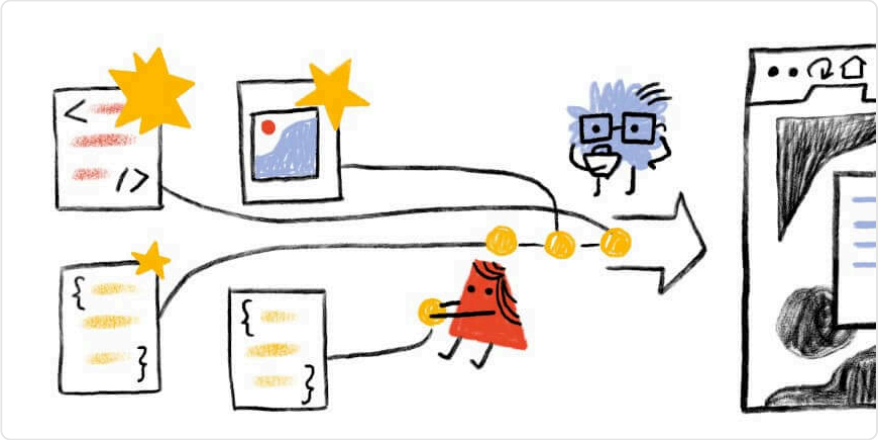
398. https://speedcurve.com/
399. https://perfnow.nl/
400. http://shop.oreilly.com/product/0636920041450.do

Katie Hempenius

@katiehempenius khempenius

Katie HempeniusはChromeチームのエンジニアで、ウェブの高速化に取り組んでいます。

部 IV 章 19
リソースヒント



Katie Hempenius によって書かれた。
Andy Davies、 Barry Pollard と Yoav Weiss によってレビュ。
Rick Viscomi による分析 と 編集。
Sakae Kotaro によって翻訳された。

序章

リソースヒント[401] は、どのようなリソースがすぐに必要になるかについての「ヒント」をブラウザに提供します。このヒントを受け取った結果としてブラウザが取るアクションは、リソースヒントの種類によって異なります。リソースヒントは正しく使用されると、重要なアクションを先取りすることでページのパフォーマンスを向上させることができます。

例[402]は、リソースヒントの結果としてパフォーマンスが向上しています。

- Jabongは、重要なスクリプトをプリロードすることで、対話までの時間を1.5秒短縮しました。
- Barefoot Wineは、目に見えるリンクを先読みすることで、将来のページの対話までの時間を2.7秒短縮しました。

401. *https://www.w3.org/TR/resource-hints/*
402. *https://youtu.be/YJGCZCaIZkQ?t=1956*

- Chrome.comは、クリティカルなオリジンに事前接続することで、待ち時間を0.7秒短縮しました。

今日、ほとんどのブラウザでサポートされているリソースヒントには、4つの独立したものがあります。`dns-prefetch`, `preconnect`, `preload`, `prefetch` です。

`dns-prefetch`

`dns-prefetch` の役割は、初期のDNS検索を開始することである。サードパーティのDNSルックアップを完了させるのに便利です。たとえば、CDN、フォントプロバイダー、サードパーティAPIのDNSルックアップなどです。

`preconnect`

`preconnect` は、DNSルックアップ、TCPハンドシェイク、TLSネゴシエーションを含む早期接続を開始します。このヒントはサードパーティとの接続を設定する際に有用である。`preconnect` の用途は `dns-prefetch` の用途と非常によく似ているが、 `preconnect` はブラウザのサポートが少ない。しかし、IE 11のサポートを必要としないのであれば、preconnectの方が良い選択であろう。

`preload`

`preload` ヒントは、早期のリクエストを開始します。これは、パーサによって発見されるのが遅れてしまうような重要なリソースをロードするのに便利です。たとえば、ブラウザがスタイルシートを受信し解析したあとでしか重要な画像を発見できない場合、画像をプリロードすることは意味があるかもしれません。

`prefetch`

`prefetch` は優先度の低いリクエストを開始します。これは、次の（現在のページではなく）ページの読み込みで使われるであろうリソースを読み込むのに便利です。プリフェッチの一般的な使い方は、アプリケーションが次のページロードで使われると「予測」したリソースをロードすることです。これらの予測は、ユーザーのマウスの動きや、一般的なユーザーの流れ/旅のようなシグナルに基づいているかもしれません。

文法

リソースヒント使用率の97%は、リソースヒントを指定するために `<link>` タグを使用しています。たとえば、以下のようになります。

```
<link rel="prefetch" href="shopping-cart.js">
```

リソースヒント使用率のわずか3%は、リソースヒントの指定にHTTPヘッダ[403]を使用しました。たとえば、以下のようになります。

```
Link: <https://example.com/shopping-cart.js>; rel=prefetch
```

HTTPヘッダー内のリソースヒントの使用量が非常に少ないため、本章の残りの部分では、`<link>` タグと組み合わせたリソースヒントの使用量の分析のみに焦点を当てています。しかし、今後、HTTP/2 Pushが採用されるようになると、HTTPヘッダーでのリソースヒントの使用量が増える可能性のあることは注目に値します。これは、HTTP/2 Pushがリソースをプッシュするためのシグナルとして、HTTPのプリロード `Link` ヘッダーを再利用していることに起因しています。

リソースヒント

注: モバイルとデスクトップでは、リソースヒントの利用パターンに目立った違いはありませんでした。そのため、簡潔にするために、本章ではモバイルの統計のみを掲載しています。

403. *https://developer.mozilla.org/docs/Web/HTTP/Headers/Link*

リソースヒント	利用状況（サイトの割合）
`dns-prefetch`	29%
`preload`	16%
`preconnect`	4%
`prefetch`	3%
`prerender` (非推奨)	0.13%

図19.1. リソースヒントの採用。

`dns-prefetch` の相対的な人気は驚くに値しません。これはよく知られたAPIであり（2009[404]ではじめて登場しました）、すべての主要なブラウザでサポートされており、すべてのリソースヒントの中でもっとも「安価」なものです。`dns-prefetch` はDNSの検索を行うだけなので、データの消費量が非常に少なく、使用する上でのデメリットはほとんどありません。`dns-prefetch` はレイテンシの高い状況でもっとも有用である。

つまり、IE11以下をサポートする必要がないサイトであれば、`dns-prefetch` から `preconnect` に切り替えるのが良いでしょう。HTTPSがユビキタスな時代には、`preconnect` は安価でありながら、より大きなパフォーマンスの向上をもたらします。`dns-prefetch` とは異なり、`preconnect` はDNSの検索だけでなく、TCPハンドシェイクとTLSネゴシエーションも開始することに注意してください。証明書チェーン[405]はTLSネゴシエーション中にダウンロードされるが、これには通常数キロバイトのコストがかかります。

`prefetch` は3%のサイトで利用されており、もっとも広く利用されていないリソースヒントである。この使用率の低さは、`prefetch` が現在のページの読み込みよりも後続のページの読み込みを改善するのに有用であるという事実によって説明できるかもしれません。したがって、ランディングページの改善や最初に閲覧されたページのパフォーマンスを向上させることだけに焦点を当てているサイトでは、これは見過ごされてしまうだろう。

404. https://caniuse.com/#feat=link-rel-dns-prefetch
405. https://knowledge.digicert.com/solution/SO16297.html

リソースヒント	ページごとのリソースヒント 中央値	ページごとのリソースヒント 90パーセンタイル
`dns-prefetch`	2	8
`preload`	2	4
`preconnect`	2	8
`prefetch`	1	3
`prerender` (非推奨)	1	1

図19.2. そのリソースヒントを使用している全ページのうち、1ページあたりに使用されているリソースヒントの数の中央値と90パーセンタイル。

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には、何も重要ではない”*）。上の図19.2は、少なくとも1つのリソースヒントを使用しているページの1ページあたりのリソースヒントの数を示しています。適切なリソースヒントの数を定義する明確なルールはありませんが、ほとんどのサイトが適切にリソースヒントを使用しているように見えます。

`crossorigin` 属性

ウェブ上に取り込まれるほとんどの「伝統的な」リソース（images、stylesheets、script）は、クロスオリジンリソース共有（CORS[406]）を選択せずに取り込まれています。つまり、これらのリソースがクロスオリジンサーバーからフェッチされた場合、デフォルトでは同一オリジンポリシーのために、その内容をページで読み返すことができないということです。

場合によっては、ページはコンテンツを読む必要がある場合、CORSを使用してリソースを取得するようにオプトインできます。CORSは、ブラウザが「許可を求める」ことを可能にし、それらのクロスオリジンリソースへのアクセスを取得します。

新しいリソースタイプ（フォント、`fetch()` リクエスト、ESモジュールなど）では、ブラウザはデフォルトでCORSを使用してリソースをリクエストし、サーバーがアクセス許可を与えていない場合はリクエストを完全に失敗させます。

406. *https://developer.mozilla.org/docs/Web/HTTP/CORS*

クロスオリジン値	使用方法	説明
未設定	92%	crossorigin属性がない場合、リクエストはシングルオリジンポリシーに従います。
anonymous(に相当する)	7%	クレデンシャルを含まないクロスオリジンリクエストを実行します。
use-credentials	0.47%	クレデンシャルを含むクロスオリジンリクエストを実行します。

図19.3. リソースヒントインスタンスの割合としての `クロスオリジン` 属性の採用。

リソースヒントのコンテキストでは、`crossorigin` 属性を使用することで、マッチすることになっているリソースのCORSモードにマッチし、リクエストに含めるべき資格情報を示すことができます。たとえば、`anonymous` はCORSを有効にし、クロスオリジンリクエストには資格情報を含めるべきではないことを示します。

```
<link rel="prefetch" href="https://other-server.com/shopping-
cart.css" crossorigin="anonymous">
```

他のHTML要素はcrossorigin属性をサポートしていますが、この分析では、リソースヒントを使った使用法のみを見ています。

`as` 属性

`as` は `preload` リソースヒントと一緒に使用されるべき属性で、要求されたリソースの種類（画像、スクリプト、スタイルなど）をブラウザに知らせるため使用されます。これにより、ブラウザがリクエストに正しく優先順位をつけ、正しいコンテンツセキュリティポリシー(CSP[407])を適用するのに役立ちます。CSPはHTTPヘッダーで表現されるセキュリティメカニズムです、信頼できるソースのセーフリストを宣言することで、XSSやその他の悪意のある攻撃の影響を緩和するのに役立ちます。

88%

図19.4. `as` 属性を使用したリソースヒントインスタンスの割合。

407. https://developers.google.com/web/fundamentals/security/csp

リソースヒントインスタンスの88%は`as`属性を使用しています。`as`が指定されている場合、圧倒的にスクリプトに使われています。92%がスクリプト、3%がフォント、3%がスタイルです。これはスクリプトがほとんどのサイトのアーキテクチャで重要な役割を果たしていることと、スクリプトが攻撃のベクターとして使用される頻度が高いことを考えると当然のことです（したがって、スクリプトが正しいCSPを適用されることがとくに重要です）。

将来のこと

現時点では、現在のリソースヒントのセットを拡張する提案はありません。しかし、優先度ヒントとネイティブの遅延ローディングは、ローディングプロセスを最適化するためのAPIを提供するという点で、リソースヒントに似た精神を持つ2つの技術が提案されています。

優先順位のヒント

優先度ヒント[408]は、リソースのフェッチの優先度を`high`, `low`, `auto`のいずれかで表現するためのAPIです。これらは幅広いHTMLタグで利用できます。とくに`<image>`, `<link>`, `<script>`, `<iframe>`などです。

```
<carousel>
  <img src="cat1.jpg" importance="high">
  <img src="cat2.jpg" importance="low">
  <img src="cat3.jpg" importance="low">
</carousel>
```

図19.4. 画像のカルーセルで優先度ヒントを使用するHTMLの例。

たとえば、画像カルーセルがある場合、優先度ヒントを使用して、ユーザーがすぐに見る画像に優先順位をつけ、後の画像に優先順位をつけることができます。

408. https://wicg.github.io/priority-hints/

0.04%

図19.5. 優先ヒントの採用率。

優先度ヒントは実装[409]されており、Chromiumブラウザのバージョン70以降では機能フラグを使ってテストできます。まだ実験的な技術であることを考えると、0.04%のサイトでしか使用されていないのは当然のことです。

優先度ヒントの85%は `<img>` タグを使用しています。優先度ヒントはほとんどがリソースの優先順位を下げるために使われます。使用率の72%は `importance="low"` で、28%は `importance="high"` です。

ネイティブの遅延ローディング

ネイティブの遅延ローディング[410]は、画面外の画像やiframeの読み込みを遅延させるためのネイティブAPIです。これにより、最初のページ読み込み時にリソースを解放し、使用されないアセットの読み込みを回避できます。以前は、この技術はサードパーティのJavaScriptライブラリでしか実現できませんでした。

ネイティブな遅延読み込みのためのAPIはこのようになります。 `<img src="cat.jpg">`.

ネイティブな遅延ローディングは、Chromium76以上をベースにしたブラウザで利用可能です。このAPIは発表が遅すぎて今年のWeb Almanacのデータセットには含まれていませんが、来年に向けて注目しておきたいものです。

結論

全体的に、このデータはリソースヒントをさらに採用する余地があることを示唆しているように思われる。ほとんどのサイトでは、 `dns-prefetch` から `preconnect` に切り替えることで恩恵を受けることができるだろう。もっと小さなサブセットのサイトでは、 `prefetch` や `preload` を採用することで恩恵を受けることができるだろう。 `prefetch` と `preload` をうまく使うには、より大きなニュアンスがあり、それが採用をある程度制限していますが、潜在的な利益はより大きくなります。HTTP/2 Pushや機械学習技術の成熟により、 `preload` や `prefetch` の採用が増える可能性もあります。

409. https://www.chromestatus.com/feature/5273474901737472
410. https://web.dev/native-lazy-loading

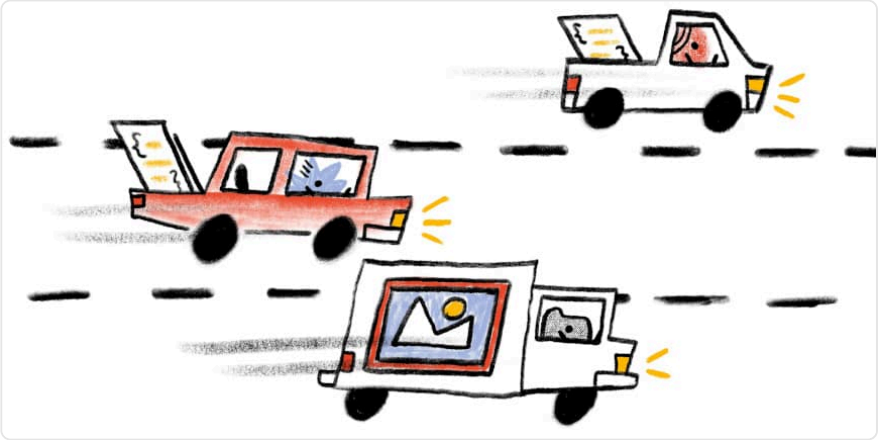
著者

Katie Hempenius

@katiehempenius khempenius

Katie HempeniusはChromeチームのエンジニアで、ウェブの高速化に取り組んでいます。

部 IV 章 20
HTTP/2



***Barry Pollard** によって書かれた。*
Daniel Stenberg、 Robin Marx と Andrew Galloni によってレビュ。
Paul Calvano による分析。
Rachel Costello 編集。
Sakae Kotaro によって翻訳された。

導入

HTTP/2は、ほぼ20年ぶりになるWebのメイン送信プロトコルの初となるメジャーアップデートでした。それは多くの期待を持って到来し、欠点なしで無料のパフォーマンス向上を約束しました。それ以上に、HTTP/1.1が非効率なため強制されていたすべてのハックや回避策をやめることができました。デフォルトでパフォーマンスが向上するため、ドメインのバンドル、分割、インライン化、さらにはシャーディングなどはすべてHTTP/2の世界でアンチパターンになります。

これは、Webパフォーマンスに集中するスキルとリソースを持たない人でも、すぐにパフォーマンスの高いWebサイトにできます。しかし現実はほぼ相変わらずです。 2015年5月にRFC 7540[411]で標準としてHTTP/2に正式承認されてから4年以上経過してます。ということでこの比較的新しい技術が現実の世界でどのように発展したかを見てみる良い機会です。

411. *https://tools.ietf.org/html/rfc7540*

HTTP/2とは?

この技術に精通していない人にとって、この章のメトリックと調査結果を最大限に活用するには、ちょっとした背景が役立ちます。最近までHTTPは常にテキストベースのプロトコルでした。 WebブラウザーのようなHTTPクライアントがサーバーへのTCP接続を開き、`GET /index.html`のようなHTTPコマンドを送信して、リソースを要求します。

これは*HTTP*ヘッダーを追加するためにHTTP/1.0で拡張されました、なのでリクエストに加えてブラウザの種類、理解できる形式などさまざまなメタデータを含めることができます。これらのHTTPヘッダーもテキストベースであり改行文字で区切られていました。サーバーは、要求とHTTPヘッダーを1行ずつ読み取ることで着信要求を解析し、サーバーは要求されている実際のリソースに加えて独自のHTTP応答ヘッダーで応答しました。

プロトコルはシンプルに見えましたが制限もありました。 なぜならHTTPは本質的に同期であるため、HTTP要求が送信されると応答が返され、読み取られ、処理されるまでTCP接続全体が基本的に他のすべてに対して制限されていました。これは非常に効率が悪く、限られた形式の並列化を可能にするため複数のTCP接続（ブラウザーは通常6接続を使用）が必要でした。

特に暗号化を設定するための追加の手順を必要とするHTTPSを使用する場合、TCP接続は設定と完全な効率を得るのに時間とリソースを要するため、それ自体に問題が生じます。HTTP/1.1はこれを幾分改善し、後続のリクエストでTCP接続を再利用できるようにしましたが、それでも並列化の問題は解決しませんでした。

HTTPはテキストベースですが、実際、少なくとも生の形式でテキストを転送するために使用されることはほとんどありませんでした。 HTTPヘッダーがテキストのままであることは事実でしたが、ペイロード自体しばしばそうではありませんでした。 HTML、JS、CSSなどのテキストファイルは通常、Gzip、Brotliなどを使用してバイナリ形式に転送するため圧縮されます。画像や動画 などの非テキストファイルは、独自の形式で提供されます。その後、セキュリティ上の理由からメッセージ全体を暗号化するために、HTTPメッセージ全体がHTTPSでラップされることがよくあります。

そのため、Webは基本的に長い間テキストベースの転送から移行していましたが、HTTPは違いました。この停滞の1つの理由は、HTTPのようなユビキタスプロトコルに重大な変更を導入することが非常に困難だったためです（以前努力しましたが、失敗しました）。多くのルーター、ファイアウォール、およびその他のミドルボックスはHTTPを理解しており、HTTPへの大きな変更に対して過剰に反応します。それらをすべてアップグレードして新しいバージョンをサポートすることは、単に不可能でした。

2009年に、GoogleはSPDY[412]と呼ばれるテキストベースのHTTPに代わるものへ取り組んでいると発表しましたが、SPDYは非推奨です。これはHTTPメッセージがしばしばHTTPSで暗号

412. *https://www.chromium.org/spdy*

化されるという事実を利用しており、メッセージが読み取られ途中で干渉されるのを防ぎます。

Googleは、最も人気のあるブラウザー（Chrome）と最も人気のあるWebサイト（Google、YouTube、Gmailなど）の1つを制御しました。 Googleのアイデアは、HTTPメッセージを独自の形式にパックし、インターネット経由で送信してから反対側でアンパックすることでした。独自の形式であるSPDYは、テキストベースではなくバイナリベースでした。これにより、単一のTCP接続をより効率的に使用できるようになり、HTTP/1.1で標準になっていた6つの接続を開く必要がなくなりHTTP/1.1の主要なパフォーマンス問題の一部が解決しました。

現実の世界でSPDYを使用することで、ラボベースの実験結果だけでなく、実際のユーザーにとってより高性能であることを証明できました。すべてのGoogle WebサイトにSPDYを展開した後、他のサーバーとブラウザーが実装を開始し、この独自の形式をインターネット標準に標準化するときが来たため、HTTP/2が誕生しました。

HTTP/2には次の重要な概念があります。

- バイナリ形式
- 多重化
- フロー制御
- 優先順位付け
- ヘッダー圧縮
- プッシュ

バイナリ形式とは、HTTP/2メッセージが事前定義された形式のフレームに包まれることを意味しHTTPメッセージの解析が容易になり、改行文字のスキャンが不要になります。これは、以前のバージョンのHTTPに対して多くの脆弱性[413]があったため、セキュリティにとってより優れています。また、HTTP/2接続を**多重化**できることも意味します。各フレームにはストリーム識別子とその長さが含まれているため、異なるストリームの異なるフレームを互いに干渉することなく同じ接続で送信できます。多重化により、追加の接続を開くオーバーヘッドなしで、単一のTCP接続をより効率的に使用できます。理想的にはドメインごと、または複数のドメイン[414]に対しても単一の接続を開きます！

個別のストリームを使用すると、潜在的な利点とともにいくつかの複雑さが取り入れられます。 HTTP/2は異なるストリームが異なるレートでデータを送信できるようにする**フロー制御**の概念を必要としますが、以前応答は1つに1つだけで、これはTCPフロー制御によって接

413. *https://www.owasp.org/index.php/HTTP_Response_Splitting*
414. *https://daniel.haxx.se/blog/2016/08/18/http2-connection-coalescing/*

続レベルで制御されていました。同様に、**優先順位付け**では複数のリクエストを一緒に送信できますが、最も重要なリクエストではより多くの帯域幅を取得できます。

最後に、HTTP/2には、**ヘッダー圧縮**と**HTTP/2プッシュと**いう2つの新しい概念が導入されました。ヘッダー圧縮により、セキュリティ上の理由からHTTP/2固有のHPACK[415]形式を使用して、これらのテキストベースのHTTPヘッダーをより効率的に送信できました。 HTTP/2プッシュにより、要求への応答として複数の応答を送信できるようになり、クライアントが必要と認識する前にサーバーがリソースを「プッシュ」できるようになりました。プッシュは、CSSやJavaScriptなどのリソースをHTMLに直接インライン化して、それらのリソースが要求されている間、ページが保持されないようにするというパフォーマンスの回避策を解決することになっています。 HTTP/2を使用するとCSSとJavaScriptは外部ファイルとして残りますが、最初のHTMLと共にプッシュされるため、すぐに利用できました。これらのリソースはキャッシュされるため後続のページリクエストはこれらのリソースをプッシュしません。したがって、帯域幅を浪費しません。

急ぎ足で紹介したこのHTTP/2は、新しいプロトコルの主な歴史と概念を提供します。この説明から明らかなように、HTTP/2の主な利点は、HTTP/1.1プロトコルのパフォーマンス制限に対処することです。また、セキュリティの改善も行われました。恐らく最も重要なのは、HTTP/2以降のHTTPSを使用するパフォーマンスの問題に対処することです、HTTPSを使用しても通常のHTTPよりもはるかに高速[416]です。 HTTPメッセージを新しいバイナリ形式に包むWebブラウザーと、反対側でWebサーバーがそれを取り出す以外は、HTTP自体の中核的な基本はほぼ同じままでした。これは、ブラウザーとサーバーがこれを処理するため、WebアプリケーションがHTTP/2をサポートするために変更を加える必要がないことを意味します。オンにすることで、無料でパフォーマンスを向上させることができるため、採用は比較的簡単です。もちろん、Web開発者がHTTP/2を最適化して、その違いを最大限に活用する方法もあります。

HTTP/2の採用

前述のように、インターネットプロトコルはインターネットを構成するインフラストラクチャの多くに深く浸透しているため、しばしば採用を難しくする事があります。これにより、変更の導入が遅くなり、困難になります。たとえば、IPv6は20年前から存在していますが、採用に苦労しています。[417]

415. *https://tools.ietf.org/html/rfc7541*
416. *https://www.httpvshttps.com/*
417. *https://www.google.com/intl/en/ipv6/statistics.html*

95%

図20.1. HTTP/2を使用できるグローバルユーザーの割合。

ただし、HTTP/2はHTTPSで事実上隠されていたため異なってました（少なくともブラウザーの使用例では）、ブラウザーとサーバーの両方がサポートしている限り、採用の障壁を取り除いてきました。ブラウザーのサポートはしばらく前から非常に強力であり、*最新*バージョンへ自動更新するブラウザーの出現により、グローバルユーザーの推定95%がHTTP/2をサポートするようになりました[418]。

私たちの分析は、Chromeブラウザで約500万の上位デスクトップおよびモバイルWebサイトをテストするHTTP Archiveから提供されています。(方法論の詳細をご覧ください。)

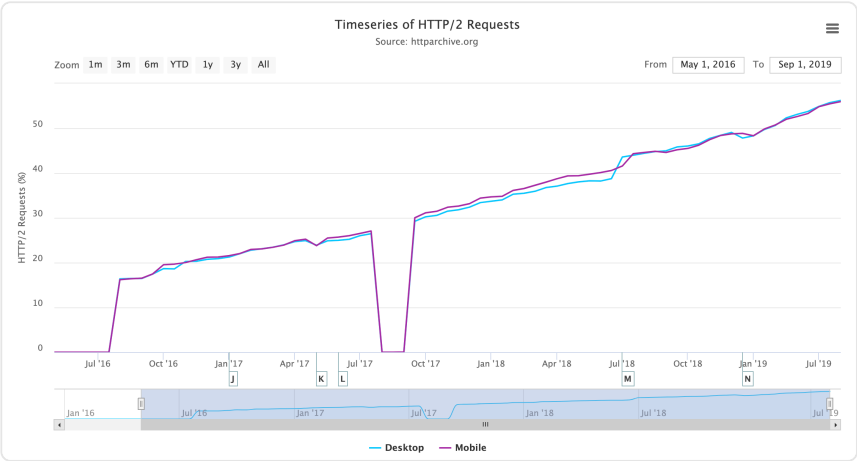


図20.2. 要求によるHTTP/2の使用。(引用: HTTP Archive[419])

結果は、HTTP/2の使用が、現在過半数のプロトコルであることを示しています。これは、正式な標準化からわずか4年後の目覚しい偉業です。要求ごとのすべてのHTTPバージョンの内訳を見ると、次のことがわかります。

418. https://caniuse.com/#feat=http2
419. https://httparchive.org/reports/state-of-the-web#h2

Protocol	デスクトップ	モバイル	合計
	5.60%	0.57%	2.97%
HTTP/0.9	0.00%	0.00%	0.00%
HTTP/1.0	0.08%	0.05%	0.06%
HTTP/1.1	40.36%	45.01%	42.79%
HTTP/2	53.96%	54.37%	54.18%

図20.3. 要求によるHTTPバージョンの使用。

図20.3は、HTTP/1.1およびHTTP/2が、予想どおり大部分の要求で使用されるバージョンであることを示しています。古いHTTP/1.0とHTTP/0.9プロトコルでは、ごく少数のリクエストしかありません。面倒なことに、特にデスクトップでHTTP Archiveクロールによってプロトコルは正しく追跡されなかった割合が大きくなっています。これを掘り下げた結果、さまざまな理由が示され、そのいくつかは説明できますが、いくつかは説明できません。スポットチェックに基づいて、それらは概ねHTTP/1.1リクエストであるように見え、それらを想定するとデスクトップとモバイルの使用は似ています。

私たちが望むよりもノイズの割合が少し大きいにもかかわらず、ここで伝えられるメッセージ全体を変えることはしません。それ以外、モバイル/デスクトップの類似性は予想外ではありません。 HTTP Archiveは、デスクトップとモバイルの両方でHTTP/2をサポートするChromeでテストします。実際の使用状況は、両方のブラウザーの古い使用状況で統計値がわずかに異なる場合がありますが、それでもサポートは広く行われているため、デスクトップとモバイルの間に大きな違いはないでしょう。

現在、HTTP ArchiveはHTTP over QUIC[420]（もうすぐHTTP/3として標準化される予定）を個別に追跡しないため、これらの要求は現在HTTP/2の下にリストされますが、この章の後半でそれを測定する他の方法を見ていきます。

リクエストの数を見ると、一般的なリクエストのため結果が多少歪んでいます。たとえば、多くのサイトはHTTP/2をサポートするGoogleアナリティクスを読み込むため、埋め込みサイト自体がHTTP/2をサポートしていない場合でもHTTP/2リクエストとして表示されます。一方、人気のあるウェブサイトはHTTP/2をサポートする傾向があり、上記の統計では1回しか測定されないため、過小評価されます（「google.com」と「obscuresite.com」には同じ重みが与えられます）。*嘘、いまいましい嘘と統計です。*

ただし、私たちの調査結果は、Firefoxブラウザーを介した実際の使用状況を調べるMozillaのテレメトリ[421]など、他のソースによって裏付けられています。

420. *https://www.chromium.org/quic*
421. *https://telemetry.mozilla.org/new-pipeline/dist.html#!cumulative=0&measure=HTTP_RESPONSE_VERSION*

プロトコル	デスクトップ	モバイル	合計
	0.09%	0.08%	0.08%
HTTP/1.0	0.09%	0.08%	0.09%
HTTP/1.1	62.36%	63.92%	63.22%
HTTP/2	37.46%	35.92%	36.61%

図20.4. ホームページのHTTPバージョンの使用。

ホームページを見て、HTTP/2をサポートするサイト数の大まかな数字を取得するだけでも（少なくともそのホームページで）おもしろいです。図20.4は、全体的な要求よりもサポートが少ないことを示しており、予想どおり約36%です。

HTTP/2は、HTTPSまたは暗号化されていない非HTTPS接続で公式に使用できますが、HTTPS上のブラウザーでのみサポートされます。前述のように、暗号化されたHTTPS接続で新しいプロトコルを非表示にすることで、この新しいプロトコルを理解してないネットワーク機器がその使用を妨げる（拒否する）ことを防ぎます。さらに、HTTPSハンドシェイクにより、クライアントとサーバーがHTTP/2の使用に同意する簡単な方法が可能になります。

プロトコル	デスクトップ	モバイル	合計
	0.09%	0.10%	0.09%
HTTP/1.0	0.06%	0.06%	0.06%
HTTP/1.1	45.81%	44.31%	45.01%
HTTP/2	54.04%	55.53%	54.83%

図20.5. HTTPSホームページのHTTPバージョンの使用。

WebはHTTPSに移行しており、HTTP/2は、HTTPSがパフォーマンスに悪影響を与えるという従来の議論をほぼ完全に覆しています。すべてのサイトがHTTPSに移行しているわけではないため、HTTP/2を利用できないサイトも利用できません。 HTTPSを使用するサイトのみを見ると、図20.5では図20.2のすべてのリクエストの割合と同様に、HTTP/2の採用率が55%前後です。

HTTP/2のブラウザサポートは強力であり、採用への安全な方法があることを示しました。なぜすべてのサイト（または少なくともすべてのHTTPSサイト）がHTTP/2をサポートしないのですか？　さて、ここで、まだ測定していないサポートの最終項目であるサーバーサポートに進みます。

これは、最新のブラウザとは異なり、サーバーが最新バージョンに自動的にアップグレードしないことが多いため、ブラウザのサポートよりも問題が多くなります。サーバーが定期的に保守され、パッチが適用されている場合でも、多くの場合、HTTP/2のような新機能ではなくセキュリティパッチが適用されます。 HTTP/2をサポートするサイトのサーバーのHTTPヘッダーを最初に見てみましょう。

サーバー	デスクトップ	モバイル	合計
nginx	34.04%	32.48%	33.19%
cloudflare	23.76%	22.29%	22.97%
Apache	17.31%	19.11%	18.28%
	4.56%	5.13%	4.87%
LiteSpeed	4.11%	4.97%	4.57%
GSE	2.16%	3.73%	3.01%
Microsoft-IIS	3.09%	2.66%	2.86%
openresty	2.15%	2.01%	2.07%
...	...	...	...

図20.6. HTTP/2に使用されるサーバー。

nginxは、最新バージョンへのインストールまたはアップグレードを容易にするパッケージリポジトリを提供しているため、ここをリードしていることについて驚くことではありません。 cloudflareは最も人気のあるCDNで、デフォルトでHTTP/2を有効にしているため、HTTP/2サイトの大部分をホストしていることについて驚くことはありません。ちなみに、cloudflareは、Webサーバーとして大幅にカスタマイズ[422]されたバージョンのnginxを使用しています。その後、Apacheの使用率は約20%であり、次に何が隠されているかを選択するサーバー、LiteSpeed、IIS、Google Servlet Engine、nginxベースのopenrestyなどの小さなプレイヤーが続きます。

さらに興味深いのは、HTTP/2をサポート*しない*サーバーです。

422. https://blog.cloudflare.com/nginx-structural-enhancements-for-http-2-performance/

サーバー	デスクトップ	モバイル	合計
Apache	46.76%	46.84%	46.80%
nginx	21.12%	21.33%	21.24%
Microsoft-IIS	11.30%	9.60%	10.36%
	7.96%	7.59%	7.75%
GSE	1.90%	3.84%	2.98%
cloudflare	2.44%	2.48%	2.46%
LiteSpeed	1.02%	1.63%	1.36%
openresty	1.22%	1.36%	1.30%
…	…	…	…

図20.7. HTTP/1.1以前に使用されるサーバー。

これの一部は、サーバーがHTTP/2をサポートしていてもHTTP/1.1を使用する非HTTPSトラフィックになりますが、より大きな問題はHTTP/2をまったくサポートしないことです。これらの統計では、古いバージョンを実行している可能性が高いApacheとIISのシェアがはるかに大きいことがわかります。

特にApacheで、既存のインストールにHTTP/2サポートを追加することは簡単でない。これは、ApacheがHTTP/2をインストールするための公式リポジトリを提供していないためです。これは、多くの場合、ソースからのコンパイルやサードパーティのリポジトリの信頼に頼ることを意味しますが、どちらも多くの管理者にとって特に魅力的ではありません。

Linuxディストリビューションの最新バージョン（RHELおよびCentOS 8、Ubuntu 18、Debian 9）のみがHTTP/2をサポートするApacheのバージョンを備えており、多くのサーバーはまだそれらを実行できていません。 Microsoft側では、Windows Server 2016以降のみがHTTP/2をサポートしているため、古いバージョンを実行しているユーザーはIISでこれをサポートできません。

これら2つの統計をマージすると、サーバーごとのインストールの割合を見ることができます。

サーバー	デスクトップ	モバイル
cloudflare	85.40%	83.46%
LiteSpeed	70.80%	63.08%
openresty	51.41%	45.24%
nginx	49.23%	46.19%
GSE	40.54%	35.25%
	25.57%	27.49%
Apache	18.09%	18.56%
Microsoft-IIS	14.10%	13.47%
...	...	...

図20.8. HTTP/2を提供するために使用される各サーバーのインストールの割合。

ApacheとIISがインストールベースのHTTP/2サポートで18%、14%と遅れを取っていることは明らかです。これは（少なくとも部分的に）アップグレードがより困難であるためです。多くのサーバーがこのサポートを簡単に取得するには、多くの場合、OSの完全なアップグレードが必要です。新しいバージョンのOSが標準になると、これが簡単になることを願っています。

ここで、HTTP/2実装に関するコメントはありません（Apacheが最高の実装の1つであると思います[423]）が、これらの各サーバーでHTTP/2を有効にすることの容易さ、またはその欠如に関する詳細です。

HTTP/2の影響

HTTP/2の影響は、特にHTTP Archive方法論を使用して測定するのがはるかに困難です。理想的には、サイトをHTTP/1.1とHTTP/2の両方でクロールし、その差を測定する必要がありますがここで調査している統計では不可能です。さらに、平均的なHTTP/2サイトが平均的なHTTP/1.1サイトよりも高速であるかどうかを測定すると、ここで説明するよりも徹底的な調査を必要とする他の変数が多くなりすぎます。

測定できる影響の1つは、現在HTTP/2の世界にいるHTTP使用の変化です。複数の接続は、限られた形式の並列化を可能にするHTTP/1.1の回避策でしたが、これは実際、HTTP/2で通

423. https://x.com/tunetheweb/status/988196156697169920?s=20

常最もよく機能することの反対になります。単一の接続でTCPセットアップ、TCPスロースタート、およびHTTPSネゴシエーションのオーバーヘッドが削減され、クロスリクエストの優先順位付けが可能になります。

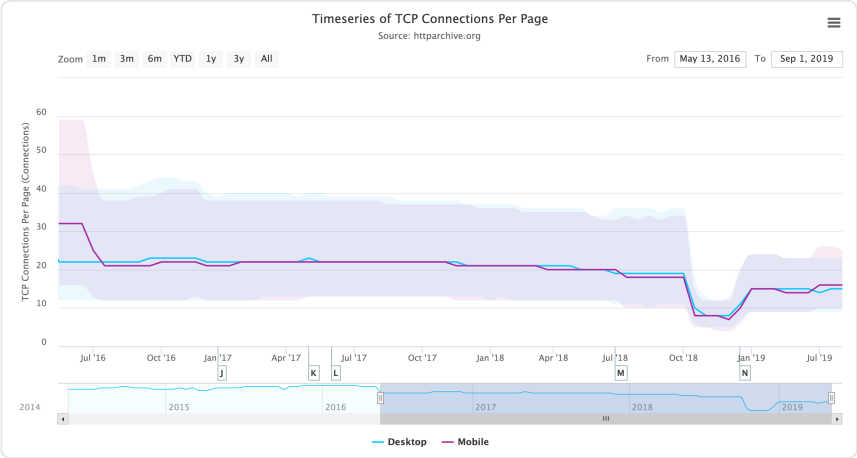


図20.9. ページごとのTCP接続。(引用: HTTP Archive[424])

HTTP Archiveは、ページあたりのTCP接続数を測定します。これは、HTTP/2をサポートするサイトが増え、6つの個別の接続の代わりに単一の接続を使用するため、徐々に減少しています。

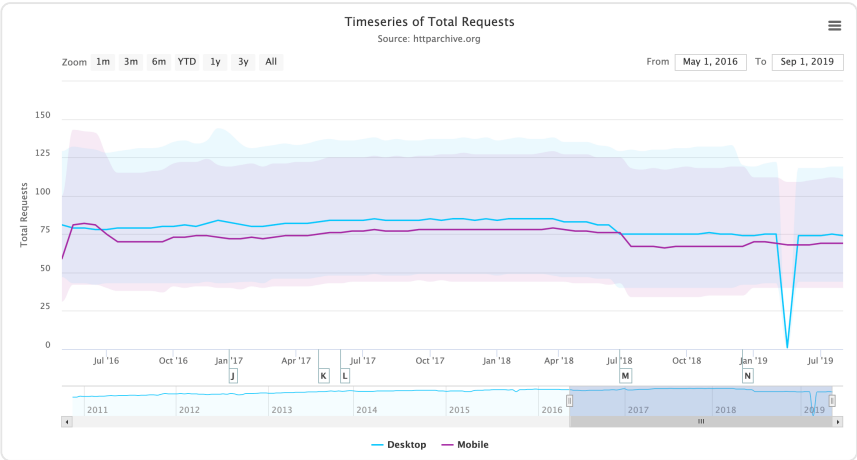


図20.10. ページごとの合計リクエスト。(引用: HTTP Archive[425])

424. https://httparchive.org/reports/state-of-the-web#tcp
425. https://httparchive.org/reports/state-of-the-web#reqTotal

より少ないリクエストを取得するためのアセットのバンドルは、バンドル、連結、パッケージ化、分割など多くの名前で行われた別のHTTP/1.1回避策でした。HTTP/2を使用する場合、リクエストのオーバーヘッドが少ないため、これはあまり必要ありませんが、注意する必要がありますその要求はHTTP/2で無料ではなく、バンドルを完全に削除する実験を行った人はパフォーマンスの低下に気付きました[426]。ページごとにロードされるリクエストの数を時間毎に見ると、予想される増加ではなく、リクエストのわずかな減少が見られます。

この減少は、おそらくパフォーマンスへの悪影響なしにバンドルを削除できない（少なくとも完全にではない）という前述の観察と、HTTP/1.1の推奨事項に基づく歴史的な理由で現在多くのビルドツールがバンドルされていることに起因する可能性があります。また、多くのサイトがHTTP/1.1のパフォーマンスハッキングを戻すことでHTTP/1.1ユーザーにペナルティを課す気がないかもしれません、少なくともこれに価値があると感じる確信（または時間！）を持っていない可能性があります。

増加するページの重みを考えると、リクエストの数がほぼ静的なままであるという事実は興味深いですが、これはおそらくHTTP/2に完全に関連しているわけではありません。

HTTP/2プッシュ

HTTP/2プッシュは、HTTP/2の大いに宣伝された新機能であるにもかかわらず、複雑な歴史を持っています。他の機能は基本的に内部のパフォーマンスの向上でしたが、プッシュはHTTPの単一の要求から単一の応答への性質を完全に破ったまったく新しい概念で、追加の応答を返すことができました。 Webページを要求すると、サーバーは通常どおりHTMLページで応答しますが、重要なCSSとJavaScriptも送信するため、特定のリソースへ追加の往復が回避されます。理論的には、CSSとJavaScriptをHTMLにインライン化するのをやめ、それでも同じようにパフォーマンスを向上させることができます。それを解決した後、潜在的にあらゆる種類の新しくて興味深いユースケースにつながる可能性があります。

現実は、まあ、少し残念です。 HTTP/2プッシュは、当初想定されていたよりも効果的に使用することがはるかに困難であることが証明されています。これのいくつかは、HTTP/2プッシュの動作の複雑さ[427]、およびそれによる実装の問題によるものです。

より大きな懸念は、プッシュがパフォーマンスの問題を解決するのではなく、すぐ簡単に問題を引き起こす可能性があることです。過剰な押し込みは、本当のリスクです。多くの場合、ブラウザーは*何*を要求するかを決定する最適な場所にあり、要求するタイミングと同じくらい重要ですが、HTTP/2プッシュはサーバーにその責任を負わせます。ブラウザが既にキャッシュに持っているリソースをプッシュすることは、帯域幅の浪費です（私の意見ではCSSをインライン化していますが、それについてはHTTP/2プッシュよりも苦労が少ないはずです！）。

426. *https://engineering.khanacademy.org/posts/js-packaging-http2.htm*
427. *https://jakearchibald.com/2017/h2-push-tougher-than-i-thought/*

ブラウザのキャッシュのステータスについてサーバーに通知する提案は[428]、特にプライバシーの問題で行き詰っています。その問題がなくても、プッシュが正しく使用されない場合、他の潜在的な問題があります。たとえば、大きな画像をプッシュして重要なCSSとJavaScriptの送信を保留すると、プッシュしない場合よりもWebサイトが遅くなります。

またプッシュは正しく実装された場合でも、パフォーマンス向上に必ずつながるという証拠はほとんどありませんでした。これも、HTTP Archiveの実行方法（1つの状態でChromeを使用する人気サイトのクロール）の性質により、HTTP Archiveが回答するのに最適な場所ではないため、ここでは詳しく説明しません。 ただし、パフォーマンスの向上は明確でなく、潜在的な問題は現実的であると言えば十分です。

それはさておき、HTTP/2プッシュの使用方法を見てみましょう。

クライアント	HTTP/2プッシュを使用するサイト	HTTP/2プッシュを使用するサイト(%)
デスクトップ	22,581	0.52%
モバイル	31,452	0.59%

図20.11. HTTP/2プッシュを使用するサイト。

クライアント	プッシュされた平均リクエスト	プッシュされた平均KB
デスクトップ	7.86	162.38
モバイル	6.35	122.78

図20.12. 使用時にプッシュされる量。

これらの統計は、HTTP/2プッシュの増加が非常に低いことを示しています。これは、おそらく前述の問題が原因です。ただし、サイトがプッシュを使用する場合、図20.12に示すように1つまたは2つのアセットではなく、プッシュを頻繁に使用する傾向があります。

これは以前のアドバイスでプッシュを控えめにし、「アイドル状態のネットワーク時間を埋めるのに十分なリソースだけをプッシュし、それ以上はプッシュしない[429]」ということでした。上記の統計は、大きなサイズの多くのリソースがプッシュされることを示しています。

428. https://lists.w3.org/Archives/Public/ietf-http-wg/2019JanMar/0033.html
429. https://docs.google.com/document/d/1K0NykTXBbbbTlv60t5MyJvXjqKGsCVNYHyLEXIxYMv0/edit

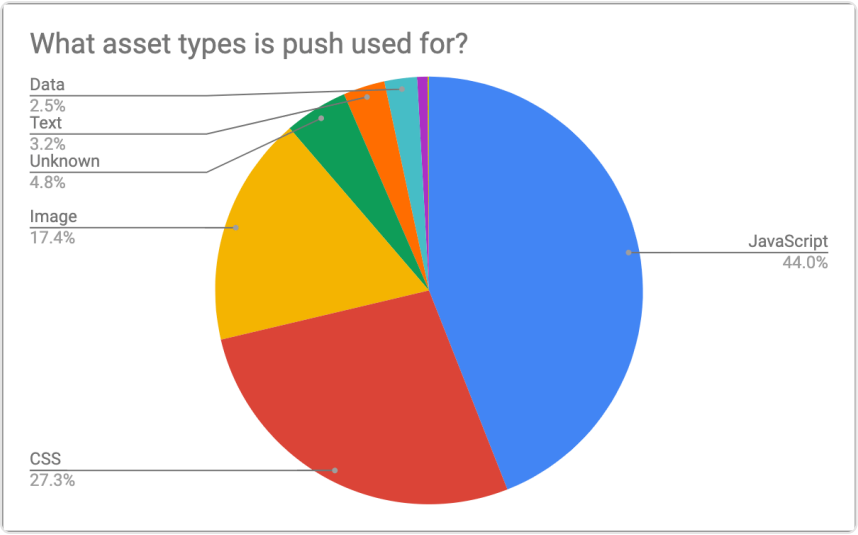


図20.13. プッシュはどの資産タイプに使用されますか？

図20.13は、最も一般的にプッシュされるアセットを示しています。 JavaScriptとCSSは、ボリュームとバイトの両方で、プッシュされるアイテムの圧倒的多数です。この後、画像、フォント、およびデータのラグタグの種類があります。最後にビデオをプッシュしているサイトは約100あることがわかりますが、これは意図的なものであるか、間違ったタイプのアセットを過剰にプッシュしている兆候かもしれません！

一部の人が提起する懸念の1つは、HTTP/2実装が`プリロード`HTTP`リンク`ヘッダーをプッシュする信号として再利用したことです。`プリロード`の最も一般的な使用法の1つは、CSSが要求、ダウンロード、解析されるまでブラウザに表示されない、フォントや画像などの遅れて発見されたリソースをブラウザに通知することです。これらが現在そのヘッダーに基づいてプッシュされる場合、これを再利用すると多くの意図しないプッシュを発生する可能性があるという懸念はありました。

ただし、フォントと画像の使用率が比較的低いことは、恐れられているほどリスクが見られないことを意味する場合があります。 `<link rel="preload" ...>`タグは、HTTPリンクヘッダーではなくHTMLでよく使用され、メタタグはプッシュするシグナルではありません。リソースヒントの章の統計では、サイトの1%未満がプリロードHTTPリンクヘッダーを使用しており、ほぼ同じ量がHTTP/2で意味のないプリコネクトを使用しているため、これはそれほど問題ではないことが示唆されます。プッシュされているフォントやその他のアセットがいくつかありますが、これはその兆候かもしれません。

これらの苦情に対する反論として、アセットがプリロードするのに十分に重要である場合、ブラウザはプリロードヒントを非常に高い優先度のリクエストとして処理するため、可能で

あればこれらのアセットをプッシュする必要があると主張できます。したがって、パフォーマンスの懸念は、（これも間違いなく）このために発生するHTTP/2プッシュではなくプリロードの過剰使用にあります。

この意図しないプッシュを回避するには、プリロードヘッダーで `nopush` 属性を指定できます。

```
link: </assets/jquery.js>; rel=preload; as=script; nopush
```

プリロードHTTPヘッダーの5%はこの属性を使用しますが、これはニッチな最適化と考えていたため、予想よりも高くなります。繰り返しますが、プリロードHTTPヘッダーやHTTP/2プッシュ自体の使用も同様です。

HTTP/2問題

HTTP/2は主にシームレスなアップグレードであり、サーバーがサポートすると、Webサイトやアプリケーションを変更することなく切り替えることができます。 HTTP/2向けに最適化するか、HTTP/1.1回避策の使用をやめることができますが、一般的にサイトは通常、変更を必要とせずに動作します。ただし、アップグレードに影響を与える可能性のある注意点がいくつかあり、一部のサイトでこれは難しい方法であることがわかりました。

HTTP/2の問題の原因の1つは、HTTP/2の優先順位付けの不十分なサポートです。この機能により、進行中の複数の要求が接続を適切に使用できるようになります。 HTTP/2は同じ接続で実行できるリクエストの数を大幅に増やしているため、これは特に重要です。サーバーの実装では、100または128の並列リクエスト制限が一般的です。以前は、ブラウザにはドメインごとに最大6つの接続があったため、そのスキルと判断を使用してそれらの接続の最適な使用方法を決定しました。現在では、キューに入れる必要はほとんどなく、リクエストを認識するとすぐにすべてのリクエストを送信できます。これにより、優先度の低いリクエストで帯域幅が「無駄」になり、重要なリクエストが遅延する可能性はあります（また偶発的にバックエンドサーバーが使用されるよりも多くのリクエストでいっぱいになる可能性があります！[430]）

HTTP/2には複雑な優先順位付けモデルがあります（非常に複雑すぎるため、なぜHTTP/3で再検討されているのでしょう！）が、それを適切に尊重するサーバーはほとんどありません。これはHTTP/2の実装がスクラッチになっていないか、サーバーがより高い優先度の要求であることを認識する前に応答は既に送信されている、いわゆるバッファブロートが原因である可能性も考えられます。サーバー、TCPスタック、および場所の性質が異なるため、

430. *https://www.lucidchart.com/techblog/2019/04/10/why-turning-on-http2-was-a-mistake/*

ほとんどのサイトでこれを測定することは困難ですがCDNを使用する場合はこれをより一貫させる必要があります。

Patrick Meenan[431]は、優先度の高いオンスクリーンイメージを要求する前に、優先度の低いオフスクリーンイメージのロードを意図的にダウンロードしようとするサンプルテストページ[432]を作成しました。優れたHTTP/2サーバーはこれを認識し、優先度の低い画像を犠牲にして、要求後すぐに優先度の高い画像を送信できるはずです。貧弱なHTTP/2サーバーはリクエストの順番で応答し、優先順位のシグナルを無視します。 Andy Daviesには、Patrickのテスト用にさまざまなCDNのステータスを追跡するページ[433]があります。 HTTP Archiveは、クロールの一部としてCDNが使用されるタイミングを識別しこれら2つのデータセットをマージすると、合格または失敗したCDNを使用しているページの割合を知ることができます。

CDN	正しい優先順位付け	デスクトップ	モバイル	合計
CDNを使用しない	わからない	57.81%	60.41%	59.21%
Cloudflare	パス	23.15%	21.77%	22.40%
Google	不合格	6.67%	7.11%	6.90%
Amazon CloudFront	不合格	2.83%	2.38%	2.59%
Fastly	パス	2.40%	1.77%	2.06%
Akamai	パス	1.79%	1.50%	1.64%
	わからない	1.32%	1.58%	1.46%
WordPress	パス	1.12%	0.99%	1.05%
Sucuri Firewall	不合格	0.88%	0.75%	0.81%
Incapsula	不合格	0.39%	0.34%	0.36%
Netlify	不合格	0.23%	0.15%	0.19%
OVH CDN	わからない	0.19%	0.18%	0.18%

図20.14. 一般的なCDNでのHTTP/2優先順位付けのサポート。

図20.14は、トラフィックのかなりの部分が特定された問題の影響を受けていることを示しており、合計はデスクトップで26.82%、モバイルで27.83%です。これがどの程度の問題であるかは、ページの読み込み方法と、影響を受けるサイトの優先度の高いリソースが遅れて検出されるかどうかによって異なります。

431. https://x.com/patmeenan
432. https://github.com/pmeenan/http2priorities/tree/master/stand-alone
433. https://github.com/andydavies/http2-prioritization-issues

27.83%

図20.15. 準最適なHTTP/2優先順位付けによるモバイル要求の割合。

別の問題は、`アップグレード`HTTPヘッダーが誤って使用されていることです。 Webサーバーは、クライアントが使用したいより良いプロトコルをサポートすることを示唆する`アップグレード`HTTPヘッダーで要求に応答できます（たとえば、HTTP/2をHTTP/1.1のみを使用してクライアントに宣伝します）。これは、サーバーがHTTP/2をサポートすることをブラウザーに通知する方法として役立つと思われるかもしれませんがブラウザーはHTTPSのみをサポートし、HTTP/2の使用はHTTPSハンドシェイクを通じてネゴシエートできるため、HTTP/2を宣伝するための`アップグレード`ヘッダーはかなり制限されています（少なくともブラウザの場合）。

それよりも悪いのは、サーバーがエラーで`アップグレード`ヘッダーを送信する場合です。これは、HTTP/2をサポートするバックエンドサーバーがヘッダーを送信し、HTTP1.1のみのエッジサーバーは盲目的にクライアントに転送していることが原因である可能性を考えます。 Apacheは`mod_http2`が有効になっているがHTTP/2が使用されていない場合に`アップグレード`ヘッダーを発行し、そのようなApacheインスタンスの前にあるnginxインスタンスは、nginxがHTTP/2をサポートしない場合でもこのヘッダーを喜んで転送します。この偽の宣伝は、クライアントが推奨されているとおりにHTTP/2を使用しようとする（そして失敗する！）ことにつながります。

108サイトはHTTP/2を使用していますが、アップグレードヘッダーでHTTP/2に`アップグレード`することも推奨しています。デスクトップ上のさらに12,767のサイト（モバイルは15,235）では、HTTPSを使用できない場合、または既に使用されていることが明らかな場合、HTTPS経由で配信されるHTTP/1.1接続をHTTP/2にアップグレードすることをお勧めします。これらは、デスクトップでクロールされた430万サイトとモバイルでクロールされた530万サイトのごく少数ですが、依然として多くのサイトに影響を与える問題であることを示しています。ブラウザはこれを一貫して処理しません。Safariは特にアップグレードを試み、混乱してサイトの表示を拒否します。

これはすべて、`http1.0`、`http：//1.1`、または`-all、+ TLSv1.3、+ TLSv1.2`へのアップグレードを推奨するいくつかのサイトに入る前です。ここで進行中のWebサーバー構成には明らかに間違いがあります！

私たちが見ることのできるさらなる実装の問題があります。たとえば、HTTP/2はHTTPヘッダー名に関してはるかに厳密でありスペース、コロン、またはその他の無効なHTTPヘッダー名で応答するとリクエスト全体を拒否します。ヘッダー名も小文字に変換されます。これは、アプリケーションが特定の大文字化を前提とする場合、驚くことになります。 HTTP/

1.1ではヘッダー名で大文字と小文字が区別されない[434]と明記されているため、これは以前保証されていませんでしたが、一部はこれに依存してます。 HTTP Archiveを使用してこれらの問題を特定することもできます、それらの一部はホームページには表示されませんが、今年は詳しく調査しませんでした。

HTTP/3

世界はまだ止まっておらず、HTTP/2が5歳の誕生日を迎えてないにも関わらず、人々はすでにそれを古いニュースとみなしており後継者であるHTTP/3にもっと興奮しています。 HTTP/3[435]はHTTP/2の概念に基づいていますが、HTTPが常に使用しているTCP接続を介した作業から、QUIC[436]と呼ばれるUDPベースのプロトコルに移行します。これにより、パケット損失が大きくTCPの保証された性質によりすべてのストリームが保持され、すべてのストリームが抑制される場合、HTTP/2がHTTP/1.1より遅い1つのケースを修正できます。また、両方のハンドシェイクで統合するなど、TCPとHTTPSの非効率性に対処することもできます。実際、実装が難しいと証明されているTCPの多くのアイデアをサポートします（TCP高速オープン、0-RTTなど）。

HTTP/3は、TCPとHTTP/2の間のオーバーラップもクリーンアップします（たとえば両方のレイヤーでフロー制御は実装されます）が、概念的にはHTTP/2と非常に似ています。 HTTP/2を理解し、最適化したWeb開発者は、HTTP/3をさらに変更する必要はありません。ただし、TCPとQUICの違いははるかに画期的であるため、サーバーオペレータはさらに多くの作業を行う必要があります。 HTTP/3のロールアウトはHTTP/2よりもかなり長くかかる可能性があり、最初はCDNなどの分野で特定の専門知識を持っている人に限定されます。

QUICは長年にわたってGoogleによって実装されており、SPDYがHTTP/2へ移行する際に行ったのと同様の標準化プロセスを現在行っています。 QUICにはHTTP以外にも野心がありますが、現時点では現在使用中のユースケースです。この章が書かれたように、HTTP/3はまだ正式に完成していないか、まだ標準として承認されていないにもかかわらず、Cloudflare、Chrome、FirefoxはすべてHTTP/3サポート[437]を発表しました。これは最近までQUICサポートがGoogleの外部にやや欠けていたため歓迎され、同様の標準化段階からのSPDYおよびHTTP/2サポートに確実に遅れています。

HTTP/3はTCPではなくUDPでQUICを使用するため、HTTP/3の検出はHTTP/2の検出よりも大きな課題になります。 HTTP/2では、主にHTTPSハンドシェイクを使用できますが、HTTP/3は完全に異なる接続となるため、ここは選択肢ではありません。またHTTP/2は アップグレード HTTPヘッダーを使用してブラウザーにHTTP/2サポートを通知します、HTTP/2にはそれほど有用ではありませんでしたが、QUICにはより有用な同様のメカニズムが導入

434. *https://tools.ietf.org/html/rfc7230#section-3.2*
435. *https://tools.ietf.org/html/draft-ietf-quic-http*
436. *https://datatracker.ietf.org/wg/quic/about/*
437. *https://blog.cloudflare.com/http3-the-past-present-and-future/*

されています。*代替サービス*HTTPヘッダー（`alt-svc`）は、この接続で使用できる代替プロトコルとは対照的に、まったく異なる接続で使用できる代替プロトコルを宣伝します。これは、アップグレードHTTPヘッダーの使用目的です。

8.38%

図20.16. QUICをサポートするモバイルサイトの割合。

このヘッダーを分析すると、デスクトップサイトの7.67%とモバイルサイトの8.38%がすでにQUICをサポートしていることがわかります。QUICは、Googleのトラフィックの割合を表します。また、0.04%はすでにHTTP/3をサポートしています。来年のWeb Almanacでは、この数は大幅に増加すると予想しています。

結論

HTTP Archiveプロジェクトで利用可能な統計のこの分析は、HTTPコミュニティの私たちの多くがすでに認識していることを示しています。HTTP/2はここにあり、非常に人気であることが証明されています。リクエスト数の点ではすでに主要なプロトコルですが、それをサポートするサイトの数の点ではHTTP/1.1を完全に追い抜いていません。インターネットのロングテールは、よく知られた大量のサイトよりもメンテナンスの少ないサイトで顕著な利益を得るために指数関数的に長い時間がかかることを意味します。

また、一部のインストールでHTTP/2サポートを取得するのが（まだ！）簡単ではないことについても説明しました。サーバー開発者、OSディストリビューター、およびエンドカスタマーはすべて、それを容易にするためプッシュすることを関与します。ソフトウェアをOSに関連付けると、常に展開時間が長くなります。実際、QUICのまさにその理由の1つは、TCPの変更を展開することで同様の障壁を破ることです。多くの場合、Webサーバーのバージョンを OSに結び付ける本当の理由はありません。 Apache（より人気のある例の1つを使用する）は、古いOSでHTTP/2サポートを使用して実行されますがサーバーに最新バージョンを取得することは、現在の専門知識やリスクを必要としません。 nginxはここで非常にうまく機能し、一般的なLinuxフレーバーのリポジトリをホストしてインストールを容易にします。Apacheチーム（またはLinuxディストリビューションベンダー）が同様のものを提供しない場合、Apacheの使用は苦労しながら縮小し続けます、最新バージョンには最高のHTTP/2実装の1つがあります、関連性を保持し古くて遅い（古いインストールに基づいて）という評判を揺るがします。 IISは通常、Windows側で優先されるWebサーバーであるため、IISの問題はそれほど多くないと考えています。

それ以外は、HTTP/2は比較的簡単なアップグレードパスであるため、既に見た強力な支持を得ています。ほとんどの場合、これは痛みなく追加が可能で、ほぼ手間をかけずにパフォ

ーマンスが向上し、サーバーでサポートされるとほとんど考慮しなくて済むことが判明しました。しかし、（いつものように）悪魔は細部にいて、サーバー実装間のわずかな違いによりHTTP/2の使用が良くも悪くも最終的にエンドユーザーの体験に影響します。また、新しいプロトコルで予想されるように、多くのバグやセキュリティの問題[438]もありました。

HTTP/2のような新しいプロトコルの強力で最新のメンテナンスされた実装を使用していることを確認することで、これらの問題を確実に把握できます。ただし、それには専門知識と管理が必要です。 QUICとHTTP/3のロールアウトはさらに複雑になり、より多くの専門知識が必要になります。おそらくこれは、この専門知識を持っており、サイトからこれらの機能に簡単にアクセスできるCDNのようなサードパーティのサービスプロバイダーに委ねるのが最善でしょうか？　ただ、専門家に任せたとしても、これは確実でありません（優先順位付けの統計が示すように）。しかし、サーバープロバイダーを賢明に選択して、優先順位が何であるかを確認すれば実装が容易になります。

その点については、CDNがこれらの問題に優先順位を付ければ素晴らしいと思います（間違いなく意図的です！）、HTTP/3での新しい優先順位付け方法の出現を疑っていますが、多くは固執します。来年は、HTTPの世界でさらに興味深い時代になるでしょう。

著者

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[439] 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、それをさらに良くしたいと思っています。@tunetheweb でツイートしたり、www.tunetheweb.com でブログを書いたりしています。

438. *https://github.com/Netflix/security-bulletins/blob/master/advisories/third-party/2019-002.md*
439. *https://www.manning.com/books/http2-in-action*

付属資料 A
方法論

概要

Web Almanacは、HTTP Archive[440]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[441]で分析できるようにしています。詳細はHTTP Archive[442]をご覧ください。

Web Almanacの使命は、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

440. https://httparchive.org
441. https://httparchive.org/faq#how-do-i-use-bigquery-to-write-custom-queries-over-the-data
442. https://httparchive.org/about

のデータウェアハウスをウェブコミュニティにさらにアクセスしやすくすることです。2019年が初版となる2019年は、ウェブの状態に関する知識の年次リポジトリと考えることができます。

Web Almanacの2019年版は、コンテンツ、体験、パブリッシング、ディストリビューションの4つの柱で構成されています。書かれたレポートの各パートは柱を表し、その異なる側面を探求する章で構成されています。例えば、パートIIはユーザー体験を表し、パフォーマンス、セキュリティ、アクセシビリティ、SEO、PWA、モバイルウェブの各章が含まれています。

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2019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2019年7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 `2019_07_01` で公開[443]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[444]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[445]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[446]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[447]を参照してください。

```
#standardSQL
# 01_01b: Distribution of JS bytes by client
SELECT
  percentile,
```

443. https://github.com/HTTPArchive/httparchive.org/blob/main/docs/gettingstarted_bigquery.md
444. https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2019
445. https://cloud.google.com/bigquery/pricing
446. https://timkadlec.com/remembers/2019-12-10-using-bigquery-without-breaking-the-bank/
447. https://github.com/HTTPArchive/almanac.httparchive.org/blob/main/sql/2019/javascript/01_01b.sql

```
  _TABLE_SUFFIX AS client,
  APPROX_QUANTILES(ROUND(bytesJs / 1024, 2),
1000)[OFFSET(percentile * 10)] AS js_kbytes
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[448]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。

ウェブサイト

データセットには5,790,700件のウェブサイトが含まれている。そのうち、5,297,442件がモバイルサイト、4,371,973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2019年7月のHTTP Archiveのクロールでは、Webサイトのリストに、最新のChrome UXレポートリリースである2019年5月（201905）を使用しました。このデータセットは2019年6月11日にリリースされたもので、5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。

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

448. https://docs.google.com/spreadsheets/d/1kBTglETN_V9UjKqK_EFmFjRexJnQOmLLr-I2Tkotvic/edit?usp=sharing

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、すべてのウェブサイトのホームページは、ログアウトした状態で空のキャッシュを使ってテストされます。

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[449]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[450]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。

449. https://www.webpagetest.org/
450. 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/75.0.3770.100 Safari/537.36 PTST/190704.170731	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/190628.140653
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 Chrome75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[451]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[452] 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[453]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。almanac.js[454]カスタムメトリクスには、DOMの状態に依存するメトリクスなど、そうでなければ計算できないいくつかのメトリクスが含まれています。

各テストの結果は、HARファイル[455]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[456]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。2019年7月のクロール時点で、HTTP Archiveは5.1.0[457]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

** 注 Lighthouseは通常、CPUが4倍遅くなるように設定されていますが、WebPageTestのバグ[458]*

451. *https://archive.org*
452. *https://cloud.google.com/compute/docs/regions-zones/#locations*
453. *https://github.com/HTTPArchive/custom-metrics*
454. *https://github.com/HTTPArchive/legacy.httparchive.org/blob/master/custom_metrics/almanac.js*
455. *https://en.wikipedia.org/wiki/HAR_(file_format)*
456. *https://developers.google.com/web/tools/lighthouse/*
457. *https://github.com/GoogleChrome/lighthouse/releases/tag/v5.1.0*
458. *https://github.com/GoogleChrome/lighthouse/issues/9668#issuecomment-535134232*

のため、テスト時には1倍になっていました。

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[459]を参照してください。

Wappalyzer

Wappalyzer[460]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。65のカテゴリ[461]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。1,200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。2019年7月現在、Web AlmanacはWappalyzerの5.8.3バージョン[462]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[463] と CMS[464] 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[465] 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、2019年7月のデータセット（201907）を使用しています。

459. *https://developers.google.com/web/tools/lighthouse/*
460. *https://www.wappalyzer.com/*
461. *https://www.wappalyzer.com/technologies*
462. *https://github.com/AliasIO/Wappalyzer/releases/tag/v5.8.3*
463. *https://www.wappalyzer.com/categories/ecommerce*
464. *https://www.wappalyzer.com/categories/cms*
465. *https://developers.google.com/web/tools/chrome-user-experience-report*

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[466]のガイドweb.dev[467]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[468]は、Patrick Hulceの研究プロジェクトで、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[469]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[470]を参照してください。

分析プロセス

Web Almanacの計画と実行には、Webコミュニティの数十人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に掲載されているメトリクスを選択した理由、それらがどのようにして照会されたか、そして解釈されたかについて説明します。

466. *https://web.dev/chrome-ux-report-bigquery*
467. *https://web.dev/*
468. *https://www.thirdpartyweb.today/*
469. *https://github.com/reworkcss/css*
470. *https://discuss.httparchive.org/t/analyzing-stylesheets-with-a-js-based-parser/1683*

ブレインストーミング

Web Almanacの発端は、2019年1月にHTTP Archiveフォーラムへの投稿[471]でイニシアチブを説明し、支援を集めることから始まりました。2019年3月には、Webコミュニティの誰もがチャプターやメトリクスのアイデアを書き込むことができる公開ブレインストーミング文書[472]を作成しました。これは、私たちがコミュニティにとって重要なことに焦点を当て、プロセスに含まれる多様な声のセットを持っていることを確認するための重要なステップでした。

ブレーンストーミングの結果、20の章が固まり、私たちは各章に主題専門家とレビュワーを割り当てる作業を開始しました[473]。このプロセスには、この規模のプロジェクトにボランティアを参加させるという課題があったため、いくつかの固有の偏りがありました。したがって、寄稿者の多くは同じ専門家サークルのメンバーです。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の、代表性の低い、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。

私たちは2019年5月から6月にかけて、人々と支部とのペアリングを行い、各支部を構成する個々の指標を最終的に決定するために彼らの意見を聞きました。

分析

2019年6月、メトリクスと章の安定したリストで、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、カスタムメトリクス[474]を作成して分析能力のギャップを埋める必要がありました。

2019年7月を通して、HTTP Archiveデータパイプラインは数百万のWeb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。

2019年8月から、データアナリストが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。合計で、**431個のクエリ**が手書きで書かれました! プロジェクトのGitHubリポジトリのsql/2019[475]ディレクトリで、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。

471. *https://discuss.httparchive.org/t/planning-the-web-almanac-2019/1553*
472. *http://bit.ly/web-almanac-brainstorm*
473. *https://github.com/HTTPArchive/almanac.httparchive.org/issues/2*
474. *https://github.com/HTTPArchive/legacy.httparchive.org/blob/master/custom_metrics/almanac.js*
475. *https://github.com/HTTPArchive/almanac.httparchive.org/tree/main/sql/2019*

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、結論を把握しやすくするために簡略化されています。たとえば、分布の完全なヒストグラムを表示するのではなく、ほんの一握りのパーセンタイルだけを表示しています。特に断りのない限り、すべての分布は、平均ではなくパーセンタイル、特に中央値（50%）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たせるようにしました。

将来を見据えて

2019年版のWeb Almanacは、ウェブコミュニティにおける内省と積極的な変化へのコミットメントの年に一度の伝統となることを願っての第一回目です。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2020年版への貢献にご興味のある方は、関心フォーム[476]にご記入ください。このプロジェクトをより良いものにするためのアイデアをお待ちしています！

476. *https://forms.gle/Qyf3q5pKgdH1cBhq5*

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。2019 Web Almanac々は、企画、調査、執筆、制作の段階で103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
翻訳者

Abigail Klein
kleinab
著作者

Adam Argyle
@argyleink
argyleink
https://nerdy.dev
著作者 と ブレインストーマー

Addy Osmani
@addyosmani
addyosmani
https://www.addyosmani.com
ブレインストーマー

Ahmad Awais
@MrAhmadAwais
ahmadawais
https://AhmadAwais.com
ブレインストーマー、 開発者 と レビュワー

Alan Kent
@akent99
alankent
https://alankent.me
著作者 と ブレインストーマー

Alberto Medina
@iAlbMedina
amedina
著作者 と ブレインストーマー

Alessandro Ghedini
ghedo
https://ghedini.me/
ブレインストーマー と レビュワー

Alex Russell
@slightlylate
slightlyoff
http://infrequently.org
ブレインストーマー

Alexey Pyltsyn
lex111
https://lex111.ru/
翻訳者

Andrew Galloni
@dot_js
dotjs
分析者 と レビュワー

Andrew Limn
@Artefact_Andy
andylimn
https://artefact.com/gb-en/
レビュワー

Andrew Noblet
anoblet
開発者

André Naumann
ndrnmnn
ブレインストーマー

Andy Davies
@AndyDavies
andydavies
http://andydavies.me/
著作者 と レビュワー

Arthur Rigaud
@arigaud_ca
arigaud-ca
翻訳者

Artur Janc
@arturjanc
arturjanc
著作者 と ブレインストーマー

Aymen Loukil
@LoukilAymen
AymenLoukil
http://www.aymen-loukil.com/en/
開発者 と レビュワー

Barry Pollard
@tunetheweb
https://webperf.social/@tunetheweb
tunetheweb.com
tunetheweb
tunetheweb
https://www.tunetheweb.com
著作者、 ブレインストーマー、
開発者、 編集者 と レビュワー

Boris Schapira
@boostmarks
borisschapira
https://boris.schapira.dev
開発者 と 翻訳者

Brian Kardell
@briankardell
https://toot.cafe/@bkardell
bkardell
https://bkardell.com
著作者、 ブレインストーマー と
レビュワー

Carlos Araya
@elrond25
caraya
http://publishing-project.rivendellweb.net/
ブレインストーマー

Carlos Torres
@carlos_catb
c-torres
開発者 と 翻訳者

Catalin Rosu
@catalinred
catalinred
https://catalin.red/
開発者 と レビュワー

Chen Hui Jing
@hj_chen
huijing
https://www.chenhuijing.com
レビュワー

Cheng Xi
chengxicn
翻訳者

Colin Bendell
@colinbendell
colinbendell
著作者 と ブレインストーマー

Daniel Stenberg
@bagder
bagder
https://daniel.haxx.se/
レビュワー

Dave Crossland
@davelab6
davelab6
https://fonts.google.com
ブレインストーマー

David Fox
@theobto
foxdavidj
https://www.lookzook.com
分析者、 著作者、
ブレインストーマー、 編集者 と
レビュワー

Doug Sillars
@dougsillars
dougsillars
https://dougsillars.com
分析者、 著作者 と ブレインストーマー

Eduardo Q. Gomes
eduqg
https://eduqg.github.io/
翻訳者

Elayne Lemos
elaynelemos
elaynelemos
翻訳者

Eric A. Meyer
@meyerweb
meyerweb
http://meyerweb.com/
レビュワー

Eric Portis
@etportis
ericportis.com
eeeps
https://ericportis.com
レビュワー

Erik Nygren
@akanygren
enygren
https://erik.nygren.org/
ブレインストーマー と レビュワー

Fatma Badri
fatmabadri
fatmabadri
https://fatmabadri.github.io/
翻訳者

Gabriel De Gennaro
https://gabrieldegennaro.com
デザイナー

Giacomo Pignoni
@Pigna_
GiacomoPignoni
開発者

Heng Yeow
@tanhengyeow
tanhengyeow
https://tanhengyeow.github.io
開発者

Henri Helvetica
@HenriHelvetica
henrihelvetica
ブレインストーマー

Houssein Djirdeh
@hdjirdeh
housseindjirdeh
https://houssein.me
著作者 と ブレインストーマー

JABANE Mohamed Ayoub
@SilentJMA
SilentJMA
翻訳者

Jared White
@jaredcwhite
jaredcwhite
https://jaredwhite.com
ブレインストーマー

Jason Haralson
jrharalson
分析者

Jeff Posnick
@jeffposnick
jeffposnick
https://jeffy.info
著作者 と ブレインストーマー

John Fox
clarkeclark
johnfox
レビュワー

John Teague
@jtteag
logicalphase
https://gemservers.com
ブレインストーマー、 開発者 と レビュワー

Jonathan Wold
@sirjonathan
sirjonathan
https://jonathanwold.com
レビュワー

José M. Pérez
@jmperezperez
JMPerez
https://jmperezperez.com
開発者、 レビュワー と 翻訳者

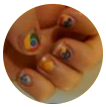
Justin Ahinon
@justinahinon1
JustinyAhin
https://segbedji.com/
翻訳者

Justin Welenofsky
welenofsky
開発者

Kari Larson
@KaJLa47
KJLarson
https://www.kjlarson.com
開発者

Kate Tymoshkina
tymosh
tymosh
翻訳者

Katie Hempenius
@katiehempenius
khempenius
分析者、 著作者 と ブレインストーマー



Laura Eberly
ljme
レビュワー

Leonardo Digiorgio
@simdigiorgio
chefleo
https://chefleo.dev/
翻訳者

M.Sakamaki
@msakamaki2
MSakamaki
開発者 と 翻訳者

Manuel Garcia
@corrosion_pt
soulcorrosion
manuel-garcia-12b6928
https://farfetchtechblog.com/en/blog/authors/manuel-garcia/
翻訳者

Mark Nottingham
@mnot
mnot
https://www.mnot.net/
ブレインストーマー

Mark Zeman
@MarkZeman
zeman
https://speedcurve.com
ブレインストーマー と レビュワー

Martin Splitt
@g33konaut
AVGP
http://geekonaut.de
著作者 と ブレインストーマー

Mathias Bynens
@mathias
mathiasbynens
https://mathiasbynens.be/
ブレインストーマー と レビュワー

Matt Ludwig
mattludwig
レビュワー

Matthew Phillips
matthewp
レビュワー

Mike Geyser
@mikegeyser
mikegeyser
https://mikerambl.es
開発者

Morten Rand-Hendriksen
@mor10
mor10
http://mor10.com/
ブレインストーマー

Nektarios Paisios
著作者

Nicolas Hoffmann
@Nico3333fr
nico3333fr
https://www.nicolas-hoffmann.net/
翻訳者

Noah Blon
@noahblon
noahblon
ブレインストーマー

Noah van der Veer
@noah_aaron_vdv
noah-vdv
翻訳者

Patrick Hulce
@patrickhulce
patrickhulce
http://patrickhulce.com
分析者、 著作者 と ブレインストーマー

Patrick Meenan
@patmeenan
pmeenan
https://www.webpagetest.org/
ブレインストーマー と レビュワー

Paul Calvano
@paulcalvano
paulcalvano
https://paulcalvano.com
分析者、 著作者、
ブレインストーマー、 開発者 と
レビュワー

Pavel Evdokimov
@Pavel_Evdokimov
Pavel-Evdokimov
ブレインストーマー

Praveen Pal
@PraveenPal4232
PraveenPal4232
https://praveenpal4232.github.io
翻訳者

Rachel Costello
@rachellcostello
rachellcostello
著作者、 ブレインストーマー と 編集者

Raghu Ramakrishnan
raghuramakrishnan71
分析者

Raghvendra Kumar
@mail_raghvendra
arsenicraghav
開発者

Renee Johnson
@reneesoffice
ernee
https://reneesvirtualoffice.com
著作者

Rick Viscomi
@rick_viscomi
rviscomi
https://rviscomi.dev/
分析者、 著作者、
ブレインストーマー、 開発者、 編集者
と レビュワー

Robin Marx
@programmingart
programmingart.bsky.social
rmarx
rmarx
http://internetonmars.org
レビュワー

Rory Hewitt
ブレインストーマー

Sakae Kotaro
@beltway7
ksakae1216
https://ksakae1216.com/
翻訳者

Sam Dutton
@sw12
samdutton
https://simpl.info
著作者 と ブレインストーマー

Scott Helme
@Scott_Helme
ScottHelme
https://scotthelme.co.uk
著作者 と ブレインストーマー

Sergey Chernyshev
@sergeyche
sergeychernyshev
http://sergeychernyshev.com/
レビュワー

Simon Pieters
@zcorpan
zcorpan
ブレインストーマー と レビュワー

Susie Lu
@DataToViz
https://www.susielu.com/
デザイナー

Sébastien Allemand
@allema_s
allemas
https://sebastienallemand.fr/
翻訳者

TJ Monserrat
tjmonsi
分析者

Tammy Everts
@tameverts
tammyeverts
https://speedcurve.com/
著作者

Thomas Steiner
@tomayac
tomayac
https://blog.tomayac.com/
著作者 と ブレインストーマー

Tommy Hodgins
tomhodgins
http://codepen.io/tomhodgins
レビュワー

Una Kravets
@una
una
http://una.im
著作者 と ブレインストーマー

Vamsee Jasti
@vamseejasti
jasti
https://vamseejasti.com
レビュワー

Vincent Terrasi
voltek62
https://data-seo.com
ブレインストーマー と レビュワー

Weston Ruter
@westonruter
westonruter
https://weston.ruter.net/
ブレインストーマー

William Sandres
@hakacode
HakaCode
https://hakacode.github.io
翻訳者

Yoav Weiss
@yoavweiss
yoavweiss
https://blog.yoav.ws
ブレインストーマー と レビュワー

Yohan Totting

@tyohan
tyohan
http://tyohan.me
開発者

Zach Leatherman

@zachleat
zachleat
https://zachleat.com/
著作者 と ブレインストーマー

Yvo Schaap

@yvoschaap
ymschaap
https://build.amsterdam/
分析者、 著作者、 ブレインストーマー と 開発者