ウェブサイトやウェブアプリケーションを構築するうえで大切なものの１つに、パフォーマンスがあげられます。

遅いサイトのユーザー体験のひどさは誰もが経験のあることでしょう。ウェブサイトならコンテンツを読んでもらえない、広告クリック率の低下、離脱率の高さを招きます。近年増えているウェブアプリケーションなら利用率の低下などにつながるでしょう。コンテンツやサービスを提供する側が本質的な価値があると思っている部分を正しく受け取ってもらうためにはパフォーマンスは欠かせない要素です。

1.1　パフォーマンスを定義する

このような文脈で語られる、パフォーマンスとはそもそもなんでしょう。本書では、パフォーマンス（ウェブパフォーマンス）とは**ユーザーの様々な振る舞いに対してウェブページが応答を返す速さ**と定義します。

ユーザーは、ブラウザを操作して、様々な**振る舞い**をします。最も典型的な一連の振る舞いは、ブラウザのアドレスバーからURLを入力してウェブページを表示させることです。他にもボタンをクリックしたり、スクロールしたり、フォームに文字を入力したり、リンクをたどって別のウェブページに遷移したりといった振る舞いがあります。

ユーザーが引き起こす、これらの振る舞いによってウェブページは何らかの**応答**を返します。たとえば、リンクをクリックすればブラウザは別のURLのウェブページを読み込んで表示します。このやりとりのことを**インタラクション**といいます。インタラクションを繰り返すことでユーザーはブラウザ上で行いたい作業を完遂します。

ユーザーがなんらかの情報を探すときには、次のような振る舞いが考えられます。まず検索エンジンのウェブサイトに「アクセス」して、調べたい情報に関連するキーワードを「フォームに入力」し、その結果のリストを「スクロール」しながら、重要そうな情報が載っているリンクを「クリック」します。

ブラウザでレンダリングされるウェブページは、これらのひとつひとつの振る舞いに対して応答を返します。ユーザーがフォームを入力してEnterを押せば新しいページに遷移して検索結果のウェブページを表示します。スクロールすれば、ウェブページの描画が下にスライドしていきます。リンクをクリックすれば、別のウェブページが表示されます。

このインタラクションがうまくいかないと、ユーザーはいらつきを感じます。インタラクションしようとするたびに振る舞いに対する応答を待たされてひっかかりを感じます。ややもするとユーザーは目的を達成しようとすることをやめてしまいます。ウェブページのリンクをクリックしても十秒以上何の反応がなくアクセスするのを諦めてしまった経験は誰にでもあることでしょう。

反対に、この応答が適切であればあるほど、ユーザーは効率よくインタラクションを繰り返すことができ、結果スムーズにそのときの目的を達成できます。目的をスムーズに達成すればするほど、ユーザーは余った時間を別のことに費やすことができます。

つまり、ウェブパフォーマンスを改善することは、ユーザーが目的の達成のために費やす時間やリソースを節約させることであり、その節約した分ユーザーをより豊かにしているわけです。

本書は、このウェブパフォーマンスについて、特にフロントエンドに注目して解説していきます†1。

1.2　パフォーマンスの重要性

ウェブパフォーマンスの重要性は上がることはあっても、下がることはありません。

10年前に比べれば、デスクトップマシン†2のスペックは上がり、インターネットの回線速度も改善しました。しかし、スマートフォンやタブレットなどのモバイル機器の興隆により、ユーザーのマシンスペックや回線環境が向上したと言い切れない変化も、また同時に起きています。

モバイル機器のネットワーク環境は必ずしも速度や安定性の面で優れたものではなく、場所によってはインターネットへの接続が切れてしまう場合さえあります。また、モバイル機器は小さく持ち運びに適すよう設計されているため、多くの場合デスクトップマシンに比べるとスペック面でも劣っています。

こういった環境では、デスクトップマシン向けのウェブサイトよりも読み込むリソースを最適化したり、貧弱なスペックでもすばやく描画されるようにチューニングする必要が出てくる場合があります。

---

†1　HTML、CSS、JavaScript、画像に代表される主にクライアント側で処理されるものとサーバー・クライアント間の通信までを本書ではフロントエンドの領域とします。

†2　本書では、デスクトップとはWindows、macOSなどのいわゆるデスクトップOSを搭載する機器、環境を指します。

1.3　新たに重要になった描画パフォーマンス

これまでもウェブパフォーマンスについて記述している書籍はいくつかありました。その多くがウェブページの初期読み込み時のパフォーマンスの最適化に重点をおいています。

本書では、初期読み込み時のパフォーマンスだけではなく、**ウェブページ内でのインタラクションのパフォーマンスの最適化**にも重点を置いて解説します。

このような構成をとっているのは、HTML によって表現されるものが文書だけではなくなり、**複雑なページ内のインタラクションを持つウェブアプリケーション**が増えてきたからです。

ウェブアプリケーション以前の従来のウェブページでは、ページ内のインタラクションを持っていないのが普通でした。HTML はもともと文書を表現するためのマークアップ言語だったため、そこで行われるインタラクションの多くはハイパーリンクをたどり、あるページから別のページに遷移すること（ページ間のインタラクション）でした。

従来は、インタラクションの多くがウェブページ間での遷移だったので、必然的に初期読み込み時のパフォーマンスが非常に重要になりました。

しかし、現代のウェブでは事情が変わってきます。JavaScript の進化や Ajax の登場によりウェブページで表現できるものがよりリッチになり、**ページ内のインタラクションを持つウェブアプリケーション**が登場しました。Google マップや Gmail などがその代表例で、高い操作性を持ちながらブラウザ内で少ないページ遷移で完結しているのが特徴です。これらは HTML、CSS、JavaScript のいわゆるフロントエンド側のウェブ技術を駆使して実装されています。

さらにウェブアプリケーションの表現をサポートする HTML5 の出現によって、HTML の仕様の面からもウェブアプリケーションの存在が認められるようになりました。今まで文書を表現するものだった HTML でアプリケーションも表現できるようにするために、様々な API の仕様の策定が進んでいます。結果としてウェブ上で実装されるアプリケーションの数はますます増加しています。

近年では、ページ内のインタラクションをつきつめてすべての処理が 1 つのページで完結する SPA（Single Page Application: シングルページアプリケーション、第 9 章コラム「Single Page Application」参照）という形のウェブアプリケーションが増えてきました。SPA では、すべてのインタラクションが他のウェブページに遷移することなしに完結します。したがって、ウェブページ遷移時の初

この章で紹介する最適化の手法は、執筆時点で支配的に用いられている TCP/IP 接続 +HTTP/1.1 のプロトコル上での通信を前提とします[†3]。

4.2 HTML/CSS/JavaScript を最小化する

HTML、CSS、JavaScript などのファイルは、通常改行やタブや余計なスペースなどの不要なバイト（文字）を含みます。これらの余計なバイト列は、専用のツールを利用することで取り除くことができます。

これによって、リソースのファイルサイズやウェブサーバーからのダウンロード時間を減らすことができます。ツールを確認してみましょう。

- HTML
  - html-minifier[†4]
- CSS
  - clean-css[†5]
  - csso[†6]
- JavaScript
  - uglify-js[†7]

リソースファイルを最小化するツールは、ここで紹介している以外にも数多く存在します。ツールの中には単に余計なバイトを取り除いてくれるだけではなく、CSS のセレクタを短くしたり、JavaScript の変数名を短くしたりするといった最適化をかけてくれるものもあります。開発者はユースケースに合うツールを選択して使うと良いでしょう[†8]。

これらの最小化は、手動でツールを起動して都度行うのではなく、ウェブサイトをデプロイする前に自動的に行うとウェブサイトの運用が楽になります。自動

† 3　HTTP/2 でも同様に使えるテクニックも紹介しています。
† 4　https://www.npmjs.com/package/html-minifier
† 5　https://www.npmjs.com/package/clean-css
† 6　https://www.npmjs.com/package/csso
† 7　https://www.npmjs.com/package/uglify-js
† 8　CSS や JavaScript を結合してファイル数を削減することも有効です。これらのファイルの結合は、最小化と同じように、gulp.js や webpack で行うことが多いでしょう。

化は、gulp.js[†9] などのタスクランナーや webpack[†10] などのビルドツールを使って、ウェブサイトをデプロイする前に対象のリソースファイルを最小化するというやり方でよく行われます。

JavaScript ライブラリでは、あらかじめ最小化されたファイル[†11] を配布している場合が多くあります。巨大なライブラリは比較的最小化の恩恵が大きいので、特別な事情がなければ最小化されたものを使ってもいいでしょう。

4.3　適切な画像形式を選択する

画像ファイルはその形式ごとに適した用途があります。それぞれの用途に最適な画像を用いることでファイルサイズを削減できる可能性があります。

一般的には写真は JPEG、単純なアニメーションは GIF、それ以外は PNG という使い分けがされます。

主な画像形式の特徴をまとめると下表のようになります。PNG は利用できる色数に応じて PNG-8、PNG-24、PNG-32[†12] という形式をそれぞれさらに持ちます。いずれも拡張子は PNG でブラウザので表示するのに大きな差異はありませんが、それぞれに異なる特徴があるので分けて説明しています[†13]。

形式	可逆	色数	透過	用途と特徴
JPEG	不可逆	約 1670 万	不可	写真。
PNG-8	可逆	256	可	色数の少ないアイコンなど。
PNG-24	可逆	約 1670 万	不可	透過を必要としないイラストなど。
PNG-32	可逆	約 1670 万	可	透過を必要とするイラストなど。半透明表示ができる。
GIF	可逆	256	可	アニメーションが可能。
WEBP	可逆・不可逆	約 1670 万	可	ファイルサイズが小さく、透過が利用できる。現状 Google Chrome 以外のブラウザで表示できない。

† 9　http://gulpjs.com/
† 10　https://webpack.github.io/
† 11　min.js などと拡張子をつけて表示します。
† 12　Photoshop CC では PNG-32 ではなく、透明部分ありの PNG-24 として表記しますが同じものです。
† 13　アニメーション向けの PNG 形式として Animated PNG（APNG）があります。モダンブラウザでは Microsoft Edge 以外はサポートしています（Google Chrome は開発版でのみ）。ここでは利用実績を考慮して、アニメーション向けとして GIF を紹介しています。

4.4　画像ファイルを最適化する

JPEG、PNG、GIF に代表される画像ファイルにもファイルサイズ削減のためのツールがそろっています。

HTML、CSS、JavaScript で不要な改行や空白文字を削除していたのに対し、画像の最適化においては不要なデータの削減やデータの整理、減色などで画像容量を削減します。それぞれのツールの利用方法や削減の方法は、各ドキュメントを参照してください。

- JPEG
  - jpegtran[†14]
  - mozjpeg[†15]
- PNG
  - ZopfliPNG[†16]
  - pngquant[†17]
  - Pngcrush[†18]
- GIF
  - GIFsicle[†19]

これらのツールは直接利用するというよりも、imagemin[†20] などのラッパーツールから使ったり、タスクランナーやビルドツールから使ったりすることが多いでしょう。ImageOptim[†21] などの GUI ツールから使われることもあります。

† 14　http://jpegclub.org/jpegtran/
† 15　https://github.com/mozilla/mozjpeg
† 16　https://github.com/google/zopfli
† 17　https://pngquant.org/
† 18　https://pmt.sourceforge.io/pngcrush/
† 19　https://www.lcdf.org/gifsicle/
† 20　https://github.com/imagemin/imagemin
† 21　https://imageoptim.com/mac

COLUMN

Web フォントの最適化

近年利用が広がる Web フォントも Loading への影響を考慮する必要があります。日本語フォントや Unicode 対応のフォントは数千以上の字体が含まれ、ファイルサイズは肥大化しがちです。この問題に対処するためには Web ページ中で必要なフォントだけを抜き出して配信するフォントのサブセット化や、適切なフォント形式の選択などが必要です。フォントのサブセット化にはツール[†22]を用いた指定や、Web フォントを提供するサービス側でサブセット化[†23]する方法があります。

4.5 CSS の import を避ける

CSS では、外部 CSS ファイルを読み込むのに @import 文[†24]を利用できます。読み込みのパフォーマンスを最適化する場合には、この @import 文は避けるのが賢明です。

@import 文は次のように CSS 内で用います。style.css で、another-style.css の読み込みを @import 文で宣言しています。

```
/* style.css */

@import url("./another-style.css");

body {
  background-color: white;
}
```

HTTP リクエストを減らすという原則に反するだけでなく、レンダリングエンジンは style.css を HTTP で取得し、パースしてから初めて @import 文を解釈します。@import 文を解釈するとさらに指定した URL の CSS ファイルを HTTP で取得します。@import 文で指定した CSS ファイルは、並列で取得されるのではなく、一旦 @import 文を記述した CSS ファイルが取得・読み込まれることを待

† 22 https://github.com/ecomfe/fontmin
† 23 https://webfont.fontplus.jp/faq/864
† 24 CSS @ ルールや、CSS @ 規則と呼ばれることもあります。

これまで本書ではブラウザのレンダリングの仕組みを解説しつつ、それぞれの問題に対するチューニングテクニックを紹介してきました。

様々なテクニックがあるとはいえ、開発者が行うことができるパフォーマンスチューニングには限界があります。デベロッパーツールを駆使してパフォーマンスを測定して、ボトルネックを特定し、チューニングを施したあと、これ以上チューニングすべき箇所が見つからないことがあります。

チューニングできる箇所があったとしても、チューニングの労力に見合わない程度のオーバーヘッドでしかない場合もあります。もし、チューニングし尽くしても時間がかかる部分が残っているとすれば、それは高速化の余地のない本質的に時間のかかる処理でしょう。

こんなとき、開発者にできることはもうないのでしょうか？ パフォーマンスを改善するためにエンジニアにできることはまだあります。

既存の動作を変えずにチューニングするのではなく、振る舞いやUIを変更することで、ユーザーの体感速度を速くできます。

多くの有名なウェブサービスでもパフォーマンス計測上のスコアには出ない、UIや体感速度という視点での最適化が施されています。この章ではそういった認知的なチューニングの手法について解説します。

9.1　インジケータを用いる

アプリケーションで何らかのデータの読込中や処理中の場面で、画面に処理中を意味するインジケータを表示することはごくごく一般的なUI設計です。ここでは後述する手法の前提として簡単に記述します。

▼インジケータを表示する