フロントエンド パフォーマンス チューニングのすゝめ

若林 奨太

はじめに

社内ではAngularをメインに使っているためAngularも絡めて話していますが、Angularだけに限った話ではないです

パフォーマンスチューニングすべき場合とは？
～ 症状例と対策例 ～

症状例	対策例
初期表示が遅い	サーバーサイドレンダリングで高速化
一覧表示が遅い	仮想DOMを使って高速化
ファイルのアップロードが遅い	APIを経由せず署名付きURLでS3に直接アップロード
アニメーションがカクつく	GPUを使うようにチューニング

使い勝手やユーザー体験に悪影響を与えている！

文脈によってはその対策は正しくないかも？

文脈付き症状例	本当に欲しかったもの
(YouTubeのようなサービスで) 初期表示が遅い	メインコンテンツ (動画など) が閲覧可能になるまでが早くならないと意味がなかった
(何万件ものデータを一覧表示していて) 一覧表示が遅い	まずページネーションが必要だった
(Google Driveのようなサービスで) ファイルのアップロードが遅い	アップロード中、他の操作はダイアログによってブロッキングされていた ブロッキングされなければ、遅さもそんなに気にならなかった
(業務システムで) アニメーションがカクつく	そもそもアニメーションなんかいらん

文脈によってはその対策は正しくないかも？

文脈付き症状例	本当に欲しかったもの
(YouTubeのようなサービスで) 初期表示が遅い	メインコンテンツ (動画など) が閲覧可能にな ても意味がなかった
(何万件ものデータを一覧表示していて) 一覧表示が遅い	ますスケルトンアニメーションが必要だった
(Google Driveのようなサービスで) ファイルのアップロードが遅い	アップロード中、他の操作はダイアログによってブロッキングされていた ブロッキングされなければ、遅さもそんなに気にならなかった
(業務システムで) アニメーションがカクつく	そもそもアニメーションなんかいらん

(この例はかなりこじつけ感があるが)
結局ケースバイケース

パフォーマンスの定義

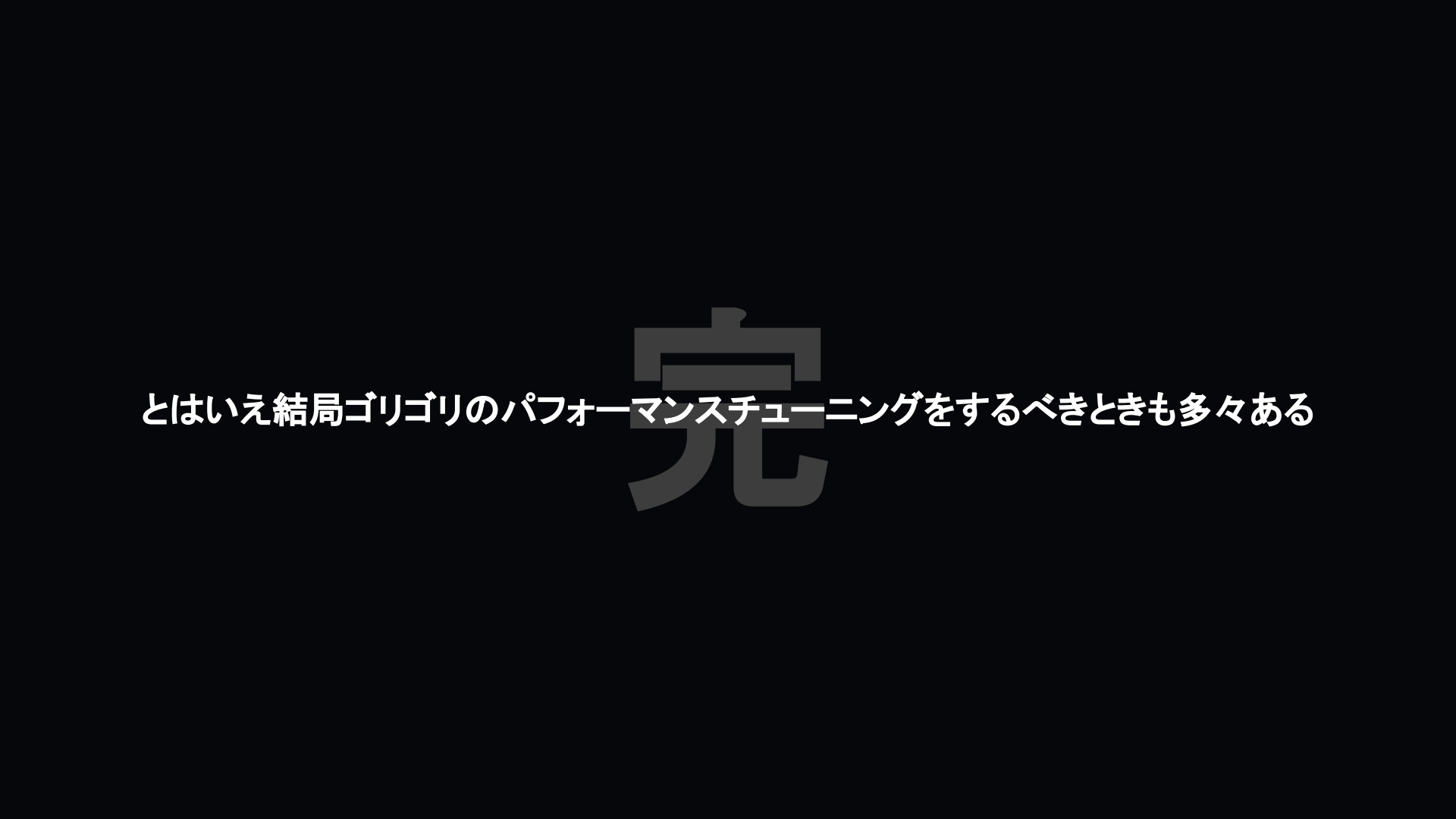
エンジニアにとっての**パフォーマンスが悪い**と、ユーザーにとっての**パフォーマンスが悪い**は違うかもしれない

パフォーマンス改善は下手をすると**1機能余裕で実装できたね**、というくらい工数がかかることもある、お客さんにも開発者にとっても大変な作業

レスポンスまでの時間、描画時間などを短くすることだけが、ユーザーにとってのパフォーマンス改善につながるとは限らない

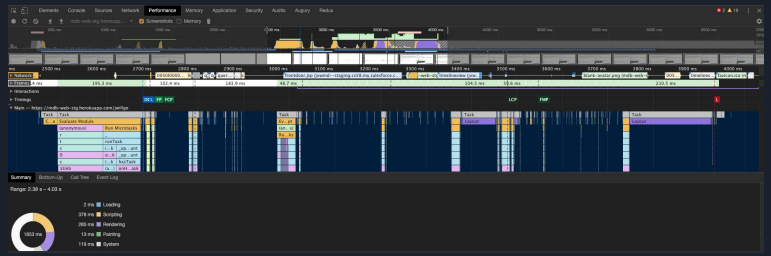
ユーザーにとって**なにが「使い勝手」「ユーザー体験」に影響しているか**とアリングや観察をすることが大切

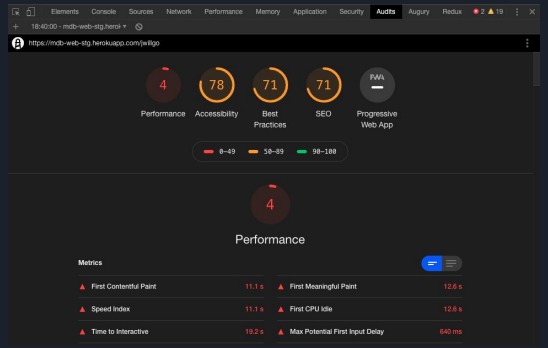
完


完
とはいえ結局ゴリゴリのパフォーマンスチューニングをするべきときも多々ある

パフォチューのすゝめ
～まず敵を知るべし～

どこがボトルネックなのかを**ちゃんと計測してから対策する**
→　当てずっぽうで対策しない
　　例）Angularだし多分変更検知のせいでしょ～(計測してないけど)
　　例）検索画面だしAPIが遅いせいでしょ～(計測してないけど)
Chromeなら**Performance**や**Lighthouse**を使えば計測できる



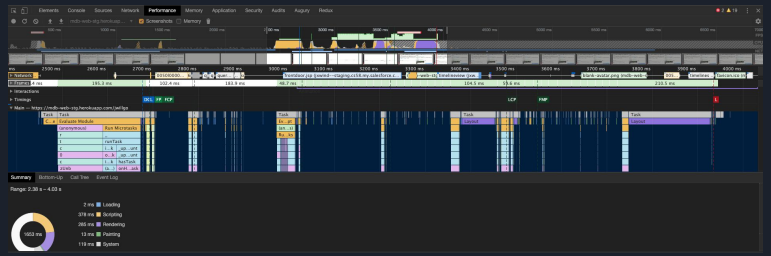


Tips: Performance VS Lighthouse

Performance

パフォーマンス計測全般に使える

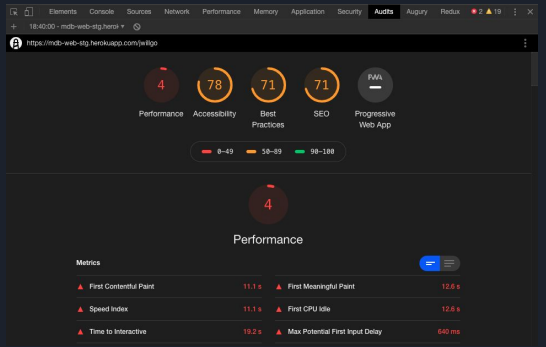
処理ごとにかかっている時間も見れる



Lighthouse

主にSEO対策で使う

どう改善したらいいかも教えてくれる



Performanceツールの使い方

見せます

パフォチューのすゝめ
～まだまだ敵を知るべし～

計測によってボトルネックとなる処理に当たりをつけたからと言って、
対策を行うのはまだ早い

パフォーマンスチューニングしたあと待つのは
　デグレ起こしているかもしれないのでちゃんとテスト
　ちゃんと早くなっているか計測 ← **早くなってなかったら時間が無駄になる**

例えばボトルネックとなっていそうな**処理をしなかったら**期待したとおりに早くなるのか
→ **これなら該当部分をコメントアウトするだけで効果をある程度計測できる**

本格的に対応を行う前に、効果を計測しておくのがとても大事！

パフォチューのすゝめ
～よくある問題への対策～

- DOM操作はめちゃくちゃ高コスト
  - よく「**変更検知で遅くなった**」という言い方を（僕含めて）することがあるが、
    正確にはほとんどの場合、変更を検知する処理自体に時間がかかっているわけではなく、
    変更が検知されたことによって DOMが頻繁に操作される から重たくなる
  - Angularのテンプレートに書いているものも最終的には **DOM操作するコードに変換される** ことを意識すること
  - 例えば *ngIf="false" だとDOMを生成しないが、display: none; はDOMを生成している
    見た目的には同じ結果になるかも知れないが、要素数が膨大な場合はもろに影響を受ける
  - ブラウザがレンダリングする流れを軽くでも理解しておくとよい
    参考　→　ブラウザレンダリングの仕組み

パフォチューのすゝめ
～よくある問題への対策～

- 変更検知
  - もちろん変更検知自体がパフォーマンスに影響を与えることもありうる
    DOMが頻繁に操作されるのを対処するには結局変更検知の回数を減らさなければいけないこともよくある
  - **ChangeDetectionStrategy.OnPush**を使うようにする
  - 参考　→　Angularでイベントから無駄に Change Detectionを走らせないためにすべきこと
  - 変更検知自体のしくみを理解するとより良い
    変更検知のしくみについては　日本語訳 : Angular 2 Change Detection Explained

パフォチューのすゝめ
～よくある問題への対策～

- 短時間に大量に発生するイベント
  - スクロールやウィンドウのリサイズ、キー入力など
  - debounceするのが基本
  - RxJSでは debounceTime を利用つかうとよい
- 大量のリスト
  - 対策しないと大量の DOMを生成することになり、画面が固まるなどする
  - 仮想スクロールを行うことで、見えている範囲以外は DOM生成を行わないようにできる
  - Angularの場合は Angular CDK の *cdkVirtualFor を使うと簡単に実装可能
    Scrolling | Angular Material
  - また、初回表示には効果はないが *ngFor に trackBy を設定することで、
    コンポーネントの再生成を抑えることができる
    無限スクロールなど、リストに項目が追加されていくようなものに有効

パフォチューのすゝめ
～よくある問題への対策～

- アニメーション
  - position: absolute; して x, y, width, height を動かすより transform を使ったほうが速い
  - transform でも遅い場合は will-change を設定すると速くなるかも
    ただし**注意点は多数あるので will-change は最終手段**
    - Chrome じゃないと効かない
    - メモリを食う
    - 拡大するとジャギる
  - 参考 → will-change - CSS: カスケーディングスタイルシート | MDN

パフォチューのすゝめ
～よくある問題への対策～

- ページロード
  - バンドルサイズを減らす
    Angularならモジュールの遅延ロードや、moment.jsなど巨大なライブラリのダイエットが有効
  - 最初に読み込まれるCSSを必要最低限にする
    ダウンロード時間という意味でも効果があるが、ブラウザがレンダリングを行うためには　CSSが必要となるため重要
  - サーバーサイドレンダリング（ SSR）を使用する
    通常なら初回のページはスクリプトを読み込んでから DOMが構築されるが、SSRを行うと初回のページが構築された状態の HTMLが吐き出されるため、ファーストビューが速くなる仕組み
    今どきのフロントエンドフレームワークなら大体ある
    Javascriptの実行などは当然ロードされてからとなるため、ボタンアクションなどについては
    早くなるとは限らない
  - PWA対応を行う
    初回アクセス時に画像などのアセットも含めてキャッシュできる
    繰り返し利用されるWebアプリなどで特に有効

完