TECHFEED CONFERENCE 2022

SPA/MPA 議論の俯瞰と現代における設計のポイント

Web フロントエンドの設計に銀の弾丸はありませんでした!!

@ahomu（Ayumu Sato）| 2022.05.14 TECHFEED CONFERENCE 2022

#tfcon

@ahomu

- TechFeed Official Expert No.25
- Web Frontend Engineer, VPoE, HR Officer
  - Offers | overflow, inc.
- Author of #超速本
  - w/@1000ch

#tfcon

“Webフロントエンドの設計”のスコープ

- 本資料におけるスコープを定義します
- 基本的にはブラウザに HTML とサブリソースを配信して UI を提供する部分の実装群が対象
- いわゆる Next/Nuxt のような BFF サーバーや、トラディショナル MVC サーバーにおける View も含まれる
- サーバーが生成した HTML をキャッシュする何らかの仕組みも含む
- 実装上のコードデザインやビルド戦略の詳細などは、すべて大前提のアーキテクチャに従属するものとしてスコープ外。各種観測範囲の偏りはご容赦を。

ではやっていきましょう

SPA vs MPA

っていうノリの議論を目にしたことはあるでしょうか？

#tfcon

用語整理

- SPA（Single Page Application）
  - 画面遷移時にブラウザが HTML をフルロードする代わりに JavaScript で制御する
  - だいたい CSR（Client Side Rendering）に強く依存する
- MPA（Multi Page Application）
  - 画面遷移時にブラウザが新しい URL から取得した HTML をフルロードする
  - だいたい SSR（Server Side Rendering）に強く依存する or ただの静的 HTML 群

定期的に浮上する議論

- SPA はコストが高い？ ref
  - なにをもってコストが高いというのか次第なのだけど･･･
  - 個人的には SPA が適さない環境、用途は存在するという点は同意できる
  - 唯一絶対的なアンサーの追求は際限がないので、みなさん好きにやりましょう
- MPA でどうやって作るの？ ref
  - CSRF 対策とかブラウザバックしても壊れないフォームとか
  - トラディショナル MVC 世代と Next/Nuxt ネイティブ世代の決定的な隔絶がありそう
  - jQuery も個人的にはもう触りたくないけど、細々と残るのも理解できる

SPA/MPA 象徴的な分類ではあるが

- MPA の中に局所的な SPA が埋め込まれることもあるし
  - 検索画面の一部だけとか、サーバー絡めた状態管理が面倒なフォーム周りだけとか
- Next/Nuxt も SPA だけでなく SSG (Static Site Generation) で MPA できる
- MPA だって CSR するし、SPA だって SSR する
- 強いていうなら SPA はレガシーとはあまり言われない技術群な傾向

SPA or MPA で対比して
議論するは既に無理筋っぽい

提供物の特性とユーザー体験をふまえて考えたら「結果的に “SPA or MPA っぽい設計” になった」くらいの温度感が適切そう

※従来的にも Micro Frontends を突き詰めると HTML 断片の Composition 大会になるので今更感はある

現代 Web の設計における関心事のおさらい

#tfcon

Web の設計において何を重要視しているのか

- **性能（パフォーマンス）はソフトウェアの重要な価値である**
- 提供物の特性 → 実現したいユーザー体験からアーキテクチャを逆算
  - 並行して、コードのメンテナンス性や習熟度、既存技術に合わせた方針が練られる
  - アーキテクチャの前提/制約条件として組織都合をすり合わせる部分は大胆に割愛
  - トラディショナル MVC のほうが手慣れた組織もあれば、Next/Nuxt じゃないと作りようが無い組織だってある
  - 実行時の性能はそこそこに、圧倒的メンテナンス性や運用能力が必要な場面もある
  - 金銭コスト等、本来考えられるその他の観点は脳内補完お願いします！

大きい関心事はHTMLの前後を速くすること

- HTML を読み込むまで
  - ブラウザが HTML をロードするまで、典型的にはキャッシュによる返却速度で解決
  - ユーザー依存情報がたくさん含まれるとキャッシュできないので困ることはある
  - 全般的に早いに越したことはない、設計をミスらなければそうそう遅くならない
- HTML を読み込んだ後
  - ブラウザがサブリソースをロードして、ユーザーに表示と操作を提供するまで
  - レンダリングパスの最適化はもちろん、近年だと JS バンドルの肥大化が槍玉の対象
  - メディアサイトで余計なことして遅いと悲しいが、SaaS だと許容しやすさはある

どこで HTML を生成するか
どこのキャッシュを効かせるか
どれくらい JavaScript ヘビーにするか

どこでHTMLを生成するか

- サーバーサイド
  - ようは SSR、Node.js 一族でも WordPress でも Ruby on Rails でもいい
- クライアントサイド
  - ようは CSR、おれたちの JavaScript と DOM API
  - CSR オンリーなら必要に応じてクローラー向けの静的 HTML 生成サービスが付随
- エッジサイド
  - 近年、CDN〜クラウド業者を中心にエッジで処理できる環境が整いつつある
  - Remix のようにエッジ上での実行を念頭においたフレームワークもでてる

どこのキャッシュを効かせるか

- 前提、色々なところでキャッシュはできる
  - サーバーアプリケーション（データベースぶち込みとかオンメモリとか）
  - ミドルウェア（nginx proxy cache とか、Varnish とか）
  - CDN（キャッシュだいすきコンテンツデリバリーネットワークさん）
  - ブラウザ（ローカルキャッシュ、ServiceWorker CacheStorage とか）
- HTML および材料のキャッシュは HTML の前工程を速くする
- サブリソースのキャッシュは HTML の後工程を速くする

どれくらい JavaScript ヘビーにするか

- JavaScript のバンドルサイズは、クライアントサイドの仕事量と比例
  - 極端な話、JavaScript ゼロで HTML 最速返却しまくるだけで優勝可能
  - 現実にはそうもいかないのでバンドル分割や減量に励むわけである
- ヘビーになりうるパラメータ
  - ビジュアル表現の有無（CSS で解決可能なものも増えているはずなんだが…）
  - インタラクションが必要な UI の有無、およびその複雑性
  - CSR の有無
  - フルスタック SPA フレームワークの(予防的)導入の有無

脱線: サバクラ間の状態引き継ぎをどうするか

- JS 界で定番化したサバクラ処理共通化で必要になる状態の引き継ぎ
- React から Hydration 界（とはいったい…）の最強を奪取したそうな各位
  - Qwik、Marko、Astro、ほかにもあったけど論壇の一部が血の気多い
- Hydration 論壇の概略について
  - Replayable vs Resumable - サーバーでやった処理をやり直すか、続きからやれるか
  - All stage vs Partial state - サーバーから全てを引き継ぐのか、一部を選択できるか
  - Initially vs Progressive（Lazy）- 初期化扱いになるのか、必要に応じて遅延できるか

提供物の理想状態にあわせて
これらのポイントを総合的に判断する

ちょうど良い力加減を見つける旅

#tfcon

Hotwire（Ruby on Rails）※参考リソース

- Turbo: 動的な画面更新においても HTML 生成をサーバーに託す
  - 古の pjax が仕組みとして全体的に賢くなったようなイメージ
- Stimulus:トラディショナル MVC へのトッピングとしては十分そう
  - Rails Way において抽象化されている分フロント複雑案件は限界あるかも（エアプ
- いわゆる “Web フロントエンド”でキャリア築いている人には…
  - React/Next.js とか Vue/Nuxt.js を前提とした習熟とはさすがに世界観の距離がある

メルカリ Web ※参考リソース

- Gatsby の SSG で静的な SPA リソースを出力している模様
- 静的なので Node.js サーバーに依存せずスケール可能になる選択
- SSR しないのは商品点数と更新頻度を鑑みるにNode.js サーバーや HTML の繊細なキャッシュの管理をするコストによる意思決定ぽい
- 軽量な SPA を維持する腕力と知性を兼ね備えた人材群が必要
- SPA で EC をやるべきでないという議論もあるがケースバイケース

Shopify Hydrogen（React Server Components）※参考リソース

- React で数年来続いている SSR+SPA の流れを最適化する試み
- HTML（シリアライズされた仮想 DOM の素を含む）はサーバーでもクライアントでも作られる
  - クライアントサイドに状態を持たないサーバーのみで処理するコンポーネントを明示することで、ブラウザに配信する JS の量を減らす
  - HTML の配信前後に SSR 処理と SPA 初期化が直列だったのを、とりあえず HTML をストリーミングで返しながら SSR 処理と SPA 初期化を並列化させたようなイメージ
- これが本当に福音をもたらすのかはまだ判別がつかない所感
  - サバクラ間の通信量やキャッシュなどの面で目に見える運用知見に乏しい故…

まとめ！

所感

- まだ SPA/MPA いってんのかい、っていう感覚の方は正常！
- HTML、キャッシュ、JS比重という勘所をおさえた上で、どうアレンジしているのかを俯瞰すると理解が楽になると思われます
- ところで、時代はエッジサイドエンジニア！っていう話が次枠にくるとヤマ張っていたのですが、どうやら外れました

ありがとう
ございました！

複業転職マッチングプラットフォーム Offers もよろしくね！

#tfcon