

Browser and UI

#3 Network/Performance


ken7253


Frontend developer

ブラウザの標準化まわりを追うのが趣味

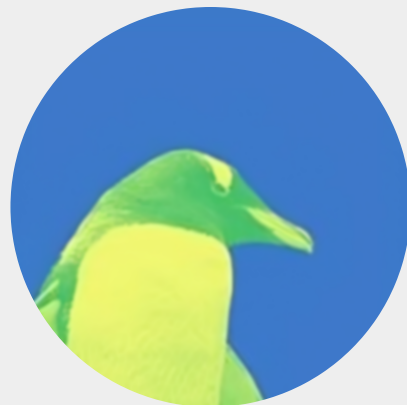
最近はReactを使ったアプリケーションを書いています。

ユーザーインターフェイスやブラウザが好き。

 <https://github.com/ken7253>

 <https://zenn.dev/ken7253>

 <https://bsky.app/profile/ken7253.bsky.social>



Browser and UI とはなにか

狭義の フロントエンドエンジニアの勉強会

狭義の フロントエンドエンジニアの勉強会

話していきたいこと

- ブラウザの仕様・標準化の話
- ブラウザの実装について
- UI・デザインの話
- UI実装を支えるツール
- フォント・画像とかのアセット系の話とかも

狭義の フロントエンドエンジニアの勉強会

積極的にはやらないこと

- ライブラリ・フレームワーク論
- （サービス全体の）設計論・アーキテクチャ
- サーバーサイドの話
- 技術以外の話

お願いしたいこと

お願いしたいこと

- 誰かを不快にさせる行動・発表はしないでください。
- ミニマムな開催にご協力をお願いします。
- 次回以降の会場提供できそうな人は教えて下さい。
- また次回やるので来てください！

バンドルサイズを半減させた話

#3 Network/Performance

プロジェクトの前提

- 組み込みブラウザ上で動作するアプリケーション
- 古いモデルでChromium 58ぐらいで、一部Safari 11ぐらいの環境も
- webpack/babelで素朴なSPAを構築

Q:サイズを削減しやすい環境とは？

A:元のバンドルサイズが大きい

やったこと

- コンパイルターゲットをES5からES6に
- Terserの設定で `mangle:false` になっていたのを解消
- 不要なコードの削除

作業としては上の2つがメイン、コードの削除は段階的に実施

コンパイラターゲットをES5からES6に

- webpackを利用しているのでwebpackの設定を変更
- 大きめのリリースに混ぜてQAが実施されるのでそれに混ぜてリリース

```
// webpack.config.js
{
  // ...
  - target: ['web', 'es5'],
  + target: ['web', 'es6'],
  // ...
}
```

この対応で **6.1MB** -> **3.74MB** にバンドルサイズを削減。

なぜES5はサイズが大きくなるか

なぜES5はサイズが大きくなるか

- ESMが使えないためTree shakingが効かない
- アロー関数から関数宣言への変換
- Generatorが存在せず非同期処理のコードが冗長になりがち

ほとんどのサービスでは意味がないので **ES5** へのトランスパイルはやめるべき

一般的なアプリケーションの指標は？

Baseline(Widely available)にあわせる

**特殊な事情がある場合は、
ES2017/ES2020を目指すとよさそう**

特殊な事情がある場合

今回のバンドルサイズ調整ではES2020を目指していた。

- Optional chaining(オプショナルチェーン `a?.b`)
- Nullish coalescing(Null 合体演算子 `a ?? b`)

上記の機能はES2020で追加されたものでよく使われている。

一方で、トランスパイル時のサイズが増えがちだった。

特殊な事情がある場合

Optional chaining(オプショナルチェーン)

```
// src: 9 Bytes
a?.b?.c
// downlevel: 100 Bytes
var _a;
(_a = a) === null || _a === void 0 || (_a = _a.b) === null || _a === void 0 ? void 0 : _a.c;
// minfy: 64 Bytes x7.1!
var l;null===(l=a)||void 0===l||null===(l=l.b)||void 0===l||l.c;
```

Nullish coalescing(Null 合体演算子)

```
// src: 7 Bytes
a ?? b;
// downlevel: 52 Bytes
var _a;
(_a = a) !== null && _a !== void 0 ? _a : b;
// minfy: 34 Bytes x4.8!
var l;null!==(l=a)&&void 0!==(l=l.b)||l;
```

Terserの設定で `mangle:false` になっていた
のを解消

Terserとmangleとは？

Terserとmangleとは？

Terser

JSのminifyを行うツール

mangle

変数名などを短くしてminifyを行う方法

なぜ `mangle:false` になっていたのか

依存ライブラリの一部がmangleを行うと壊れたため

リリースのために一時的に `mangle:false` されていたのが放置されていた。

- 調査ログなどが軽く当時のPRに書いてあったのでそれを確認
- ライブラリ側のアップデートで修正されていたのが確認
- ライブラリをアップデート
- 記述を削除してリリース

`4.11MB` -> `2.95MB` に削減

不要なコードの削除

- 開発中しか利用しないコードのバンドルを防ぐ
- 重複しているコードを共通化する

開発中しか利用しないコードのバンドルを防ぐ

```
// mock.ts
export const worker = setupWorker(...handlers);

/* ===== */

// mswのモックを有効化する処理
if (process.env.NODE_ENV === 'development') {
  if (process.env.ENABLE_MSW === 'true') {
    import("./mock.ts").then((worker) => worker.start())
  }
};

const initApp = () => {
  //...
}
```

- デットコード削除の対象になるようにダイナミックインポートに

重複しているコードを削除・共通化する

ここはほぼおまけ程度

- ダイアログやローディングなど共通化できる部分を共通化
- リファクタリングの作業を進めていくと自然とコード量が減っていく
- 型情報から不要だと分かるオプションチェーンや初期値の代入を削る

2.95MB -> 2.83MB

ざっくりとバンドルサイズを半減できた

まとめ

まとめ

ES5に変換してはいけない

- トランスパイルターゲットもBaseline(Widely available)に合わせていく
- 特殊な環境の人は頑張って自分たちで答えを出そう
- ES2017/ES2020が大きな目標かも

定期的にビルド設定の見直しを

- サポートブラウザと共に定期的に見直す
- きちんと想定したバージョンにトランスパイルされているか確認する

無駄なコードを書かない

- 静的解析とレビューでカバー

END