ソフトウェア開発の環境を考える

すでに開始しているフロントエンドの開発に加わる場合、まず文法ぐらいが分かって入れば開発に入ることができるでしょう。 しかし、立ち上げる場合は最低1人は環境構築ができるメンバーが必要になります。 本章以降はその最低1人を育てるための内容になります。

近年ではJavaScriptは変換をしてデプロイするのが当たり前になってきている話は紹介しました。フロントエンド用途の場合、規模が大きくなってきているため数多くのツールのサポートが必要になってきています。

- コンパイラ

  TypeScriptで書かれたコードをJavaScriptに変換するのがコンパイラです。基本的にはTypeScript純正のものを使うことになるでしょう。 一部のBabelを前提としたシステムではBabelプラグインが使われることがあります。これは型定義部分を除外するだけで、変換はBabel本体で行います。 ネイティブコードにしてからWebAssemblyに変換するAssemblyScriptというものもあったりします。

  - TypeScript
  - @babel/plugin-transform-typescript
  - AssemblyScript

- テスティングフレームワーク

  ソフトウェアを、入力と出力を持つ小さい単位に分けて、一つずつ検証するために使います。ブラウザの画面の操作をエミュレートし、結果が正しくなるかを検証するend-to-endテストもあります。

  - Jest
  - Ava
  - Mocha
  - Jasmine

- 静的チェックツール

  近年のプログラミング言語は、他の言語の良いところを積極的に取り入れたりして機能拡張を積極的に行なっています。それにより、古い書き方と、より良い書き方のいくつかの選択肢がある場合があります。間違いやすい古い書き方をしている箇所を見つけて、警告を出すのが静的チェックツールです。TSLintが今まで多く使われていましたが、Microsoftが、TSLintでは構造的にパフォーマンスが出しにくいとのことで、ESLintをバックアップしていくという発表しました。そのため、選択肢としては現在はこれ一択です。

  - eslint

- コードフォーマッター

JavaScriptの世界では、以前は静的チェックツールの1機能として行われることが多かったのですが、最近では、役割を分けて別のツールとなっています。

  - Prettier
  - TypeScript Formatter
  - gts

- タスクランナー

  ソフトウェア開発ではたくさんのツールを組み合わせる必要がありますが、その組み合わせを定義したり、効率よく実行するのがタスクランナーです。色々なツールがでてきましたが、現在は実行自体はnpmで行い、変更があったファイルだけを効率よくビルドするのはバンドラー側で行うと、役割分担が変わってきています。 npm scriptsは標準機能ですが、少し機能が弱いため、 `npm-run-all` を組み合わせて少し強化して使うことがあります。 本ドキュメントでもそのようにします。

  - npm scripts
  - gulp
  - grunt

- バンドラー

  コンパイラが変換したファイルを最終的に結合したり、動的ロードを有効にするのがバンドラーです。次のようなものがあります。 なお、ライブラリなど、バンドラーは行わず、コンパイラだけ実行した状態で配布することもできます。

  - Webpack
  - parcel
  - rollup

さらに規模が大きくなり、ビルドしてリロードして起動、というステップに時間がかかるようになってくると、ビルドサーバーを前面にたてて開発にかかる待ち時間を減らすことも当たり前のように行われます。

ビルドサーバーはコンパイル済みのファイルを結果をメモリにキャッシュします。ローカルのファイルを監視し、変更があったらそのファイルだけを更新し、JavaScriptにコンパイルします。ビルドサーバーによっては、開発ビルドではサーバーに変更があったかを問い合わせるコードを埋め込んでおくものもあります。その機能を使うと、エディタでファイルを保存すると、コンパイルが自動で走り、ブラウザが開発サーバーから変更された情報を受け取り、ブラウザに変更されたコードをロードし直すまで自動で行われたりします。

環境構築できるメンバーがいないときはどうするか？

一時的に構築できるメンバーにヘルプに入ってもらって環境整備をやってもらう、ということも方法としてはありますが、可能であれば常に質問できる人が望ましいです。もちろん、WebPackやBabelを自分で設定する覚悟があるのであれば問題はありませんが、一時的に誰かに高度な環境

を作ってもらうと、やはりそこがタコツボ化してしまい、メンテナンスができなくなりがちです。あまり背伸びをしてしまうと、そこが負債になりがちです。

環境構築をしても、一度きりでは終わりません。ライブラリが定期的にバージョンアップしていく作業も発生しますし、何かしらの新しいツールやライブラリを付け加えるという作業も発生します。環境周りの面倒を見れるメンバーが一人はプロジェクトにはいることが望ましいです。

もしも誰もいないかつReactであればNext.jsを使い、てっとり速く作れる環境を利用すべきです。VueやAngularはデフォルトのプロジェクト作成である程度揃うので、それを活用しましょう。そして、環境設定はなるべくがんばらず、デフォルトのままで頑張れるだけ頑張るのが良いでしょう。バージョンアップ時は、新しくプロジェクトを作り、そこに既存のコードを持ってきて式年遷宮をする、というのがトラブルが少ないと思われます。

Next.jsであれば、ある程度のベストプラクティスに従って設定はされているし、そこからの機能追加も情報が得やすいので、無難な落としどころとなります。