適当に教える
最近のフロントエンド開発
第一歩👍

自己紹介

@pvcresin

- 明治大学 宮下研究室 M1（修士1年）
- 想隆社でWEBのフロントエンドを担当
- 本職はKotlinでAndroid書くマン

はじめに

- 最近のフロントエンド開発の第一歩をめちゃくちゃ雑に紹介します **（2017/10 現在）**
- 目標は **「へぇ～こういうのもあるんだ～」**
- フロントエンドの開発環境は，毎年のように移り変わっていくので，このスライドも 2018/10 にはほぼ使いものにならないでしょう（泣

今回使うもの

- Visual Studio Code
  - https://code.visualstudio.com/
- Node.js
  - https://nodejs.org/ja/
- Yarn
  - https://yarnpkg.com/en/docs/install

最新版をインストールしておきましょう
あとブラウザはChrome使います

ちなみに

- このスライドは Marp というMarkdownからスライドを生成するツールで作成しました
- 今回，最終的にできるファイルはここ⬇
  https://github.com/pvcresin/testMarp

Menu

- Visual Studio Code
- Node.js
  - npm / Yarn
- live reload
- Pug
- PostCSS
- JavaScript (es6)
  - webpack + Babel

エディタの有名どころ

- Sublime Text
  - 「恋に落ちるエディタ」として名高い
  - 有料だが，無料でもフル機能を使える
- Atom （by Github）
  - Electron製（JSでデスクトップアプリ作る君）
  - 拡張が多く公開されている
- Brackets（by Adobe）
  - ライブプレビューが標準搭載

Visual Studio Code

- Microsoftが作ったWeb開発に特価したエディタ
- OSSで無料
- Win / Mac / Linux 対応
- Git連携機能やターミナルが標準搭載
- Electron製

`Ctrl + ``でターミナルが開き，コマンドが使える！

自分の場合: Sublime ➜ Atom ➜ VSCode
Bracketsは使ったこと無い

Node.js

- サーバサイドで動くJavaScript
- Chrome に搭載されている V8 エンジンで動作
- フロントエンド開発に無くてはならない
- 偶数バージョンが長期サポート（LTS）になる

`node -v` で動くか確認

npm

- **Node** の **Package** の **Manager** 的なやつ
  - 要はライブラリなどを入れるためのツール
- Node入れたら，だいたいデフォルトで入ってる
- YarnというFacebookが作った上位互換に押されつつある（適当）
  - npmより速い（ときもある）

`npm -v` で動くか確認

npmの使い方

- `npm init`で`package.json`（大事なやつ）を作成
  - `npm init -y`でとりあえず空のを作ることも可
- `npm run xxx`で`package.json`の`scripts`に定義したコマンドを起動可能（ = **npm script** ）

```
{
  "name": "testMarp",
  "version": "1.0.0",
  "main": "index.js",
  "license": "MIT",
  "scripts": {
    "test": "echo hello"
  }
}
```

npmの使い方

- `npm install --save xxx` = `npm i -S xxx`
  - `package.json`の`dependencies`に依存しているモジュールを追記し，`node_modules`にダウンロード
- `npm install --dev xxx` = `npm i -D xxx`
  - 上の`devDependencies`バージョン
- `npm install` = `npm i`
  - `package.json`の依存モジュールを一気に入れる
  - 例：`git clone`してきたNodeのプロジェクトなど

Yarnの使い方

- `yarn init` = `npm init`
- `yarn` = `yarn install` = `npm i`
- `yarn add` = `npm i -S xxx`
- `yarn add -D xxx` = `npm i -D xxx`
- `yarn xxx` = `yarn run xxx` = `npm run xxx`

`yarn -v`で動くか確認

live reload

- ファイルを変更して保存したら，ブラウザを自動でリロードする技術
- これを実現する色々なパッケージが存在
  - `live-server`, `browser-sync`など
- ブラウザをリロードせずに変更した要素だけを入れ替える`Hot Module Replacement`という技術もある
  - が今回は難しいので割愛

live-serverを使ってみる

1. `yarn add -D live-server`でモジュールを追加
2. `dist`フォルダを作成し，その中に`index.html`を作成
3. `scripts`に
   ```
   "watch:browser": "live-server dist --browser=chrome --watch=/"
   ```
   を追加
4. `yarn watch:browser`したら，`dist/index.html`を編集して保存するとブラウザがリロードする

出力フォルダ名をよく`dist`にするけど，**distribution（配布物）**や**district（特定の場所）**という説がある

Pug（旧Jade）

- HTMLを楽に書くための定番プリプロセッサ
- インデントで記述

```
doctype html
html(lang="ja")
  head
    meta(charset="UTF-8")
    title Pug
  body
    h1 Hello
```

- HTML2Jadeってサイトが便利

pug-cliを使ってみる

1. `yarn add -D pug-cli`して追加
2. `src/pug/index.pug`を作成
3. `scripts`はこんなの

```
"build:pug": "pug src/pug/index.pug -o dist/ -P",
"watch:pug": "npm run build:pug -- -w",
```

- `yarn build:pug`で１度だけビルド
- `yarn watch:pug`でファイルの変更を監視（`watch`）`src/pug/index.pug`の更新する度に，`dist/index.html`に出力

npm-run-allを使って並列化

- npm scriptを複数指定し，順番または並列に処理できる
- 例 `run-p build:*`
  - `build:*`にマッチするnpm scriptを**parallel** に **run** するという意味
- `yarn add -D npm-run-all`で追加

npm-run-allを使って並列化

`watch:browser`とさっきの`watch:pug`を組み合わせると

```
"build": "run-p build:*",
"build:pug": "pug src/pug/index.pug -o dist/ -P",
"watch": "run-p watch:*",
"watch:pug": "npm run build:pug -- -w",
"watch:browser":
        "live-server dist --browser=chrome --watch=/"
```

- `yarn build` → `build:*` → `build:pug`
- `yarn watch` → `watch:*`
  → `watch:browser`
  → `watch:pug` → `build:pug`

休憩

- これで**npm script**の基礎は完成
- あとは`build:xxx`と`watch:xxx`を同じように増やしていくことで並列処理を増やしていくことが出来る

PostCSS

- CSSを楽に書くための新しめのプリプロセッサ
  - 他にもStylus, LESS, SASS(SCSS)などがある
  - SASSがよく用いられていたが，高機能のため，変換に時間がかかるのが難点だった
- 欲しい機能をプラグインとして個別に導入が可能
  - 変数が使いたい, ネストしたい...などなど
- PostCSS.partsでプラグインの検索が可能（便利）

個人的によく使うプラグイン

- postcss-cssnext
  - まだ導入が進んでいない次世代のCSS記法を先取りして使える
  - 様々なプラグインの集合体でもある
    - autoprefixer: ブラウザでの表示差を埋める
    - postcss-nesting: ネストしてCSSをかける
- postcss-simple-vars
  - SASSのスタイルでCSS内に変数を宣言できる

postcss-cliを使ってみる

1. `yarn add -D postcss-cli postcss-cssnext postcss-simple-vars`
2. `src/postcss/style.css`を作成
3. `scripts`に
   `"build:postcss": "postcss src/postcss/*.css -d dist/css/ --no-map -u postcss-simple-vars postcss-cssnext",`
   と
   `"watch:postcss": "npm run build:postcss -- -w",`
   を追記

PostCSSを触ってみる

- `yarn build`すると，`dist/css/style.css`が出力される
- `src/pug/index.pug`の`head`タグの最後に`link(rel="stylesheet", href="css/style.css")`を追記
- `yarn watch`し，`src/postcss/style.css`を編集すると自動でCSSに変換し，ブラウザをリロード！

PostCSSのコード例

```
body {  // 変換前のpostcss
  $baseColor: cyan;
  background: $baseColor;
  & h1 {
    color: $baseColor;
    background: red;
  }
}
```

```
body {  /* 変換後のcss */
  background: cyan;
}
body h1 {
  color: cyan;
  background: red;
}
```

JavaScript（es6）

- JSの新しい記法 **es6** (= es2015)
  - 正式名称: **ECMA Script6**
- イメージ
  - **レガシー: es5, モダン: es6, 次世代: es7**
- es6で書いて，es5に変換するのが主流
  - Babelというツールが有名
    - ブラウザ間の差を埋める機能もついている
    - 類似品：Bubble

es6 で変わったところ

- 変数宣言 `const`, `let` の導入
  - `const`（再代入不可）, `let`（再代入可）
- アロー関数

```
// 従来の関数
var plus = function(x, y) {
  return x + y;
};

// アロー関数
const plus = (x, y) => {
  return x + y;
};
```

es6 で変わったところ

- クラス構文

```
class Person {
  constructor(name) {
    this.name = name;
  };
  hello() {
    console.log(`My name is ${this.name}.`);
  };
}

const p = new Person('es6');
p.hello();      //=> "My name is es6."
```

es6 で変わったところ

- `import` / `export` の導入
  - 他のJSファイルの関数やクラスを読み込める
  - 例
    - **person.js** `export default class Person { }`
    - **index.js** `import Person from './person';`
  - 機能を **モジュール** ごとに分割し，自由に組み合せることが可能に

webpack

- 複数のモジュールを1つのファイルにする **バンドラ**
- HTMLにおけるJSの**読み込み順の悩みから開放**
- CSSや画像もJSファイルにバンドルすることができ，リクエスト数の削減につながる
- 代替品：Rollup, Browserify, Fuseboxなど

Grunt, Gulpといったツールは **タスクランナー** と呼ばれ，先程 npm script で行ったようなことを中心にサポートするツール（最近使わない）

`webpack`と`Babel`を使ってみる

- `Babel`でes6のJSをes5にし，`webpack`でバンドル！
- モジュールの説明
  - babel-core: `Babel`本体
  - babel-preset-es2015: 変換に使うプリセット
  - babel-loader: `Babel`を`webpack`上で扱う君
  - webpack: `webpack`本体

```
yarn add -D babel-core babel-loader babel-preset-es2015 webpack
```

webpack + babel

- `webpack.config.js`をルートに作成

```
const webpack = require('webpack')
const path = require('path')

module.exports = {
  context: path.resolve(__dirname, './src/js'),
  entry: {
    index: './index.js'
  },
  output: {
    path: path.join(__dirname, 'dist/js/'),
    filename: '[name].js'
  },
```

```
  module: {
    loaders: [{
      test: /\.js$/,
      exclude: /node_modules/,
      loader: 'babel-loader',
      query: {
        presets: ['es2015']
      }
    }]
  },
  resolve: {
    extensions: ['.js']
  }
}
```

少し癖がある＋バージョンによっても書き方が変わるので注意

JSファイル

- `src/js/`に`person.js`と`index.js`を作成

`person.js`

```
export default class Person {
  constructor(name) {
    this.name = name;
  };
  hello() {
    console.log(`My name is ${this.name}.`);
  };
}
```

JSファイル

`index.js`

```
import Person from './person';

const p = new Person('es6');
p.hello();
```

- `src/pug/index.pug`の`body`タグの最後に`script(src="js/index.js")`を追加

JSを変換してみる

- `scripts`に下記2つを追加

```
"build:js": "webpack",
"watch:js": "npm run build:js -- -w",
```

- `yarn watch`すると，ブラウザのデベロッパーツールのコンソールに`My name is es6.`が表示される
- webpackのせいで，生成されたJSファイルは読めたもんじゃないが，よく見るとちゃんと2ファイルが1つにバンドルされているのが確認できる

完成

- これでメタな言語や新しい記法のファイルを，HTMLとCSSとJSのファイルに変換し，ライブリロードする環境が整いました
- あとはそれぞれをいじって学ぶだけです
- `node_modules`の中身は**git ignore**しましょう
  （`package.json`さえあれば環境構築はできるので）

お疲れ様でした！

- Next Step
  `Hot Module Replacement`機能
  Viewフレームワーク: `React`, `Vue`, `Riot`
  ルーター: `React Router`, `vue-router`, `Riot Router`
  アーキテクチャ: `Flux`, `Redux`
  CSS: `CSS modules`
  JSのメタ言語: `TypeScript`
- ここまで来てなんなんですが，HTML5 や CSS3，JS の **Promise** や **fetch** ら辺がちゃんと出来たほうが良いと思います！
- おわり！