

Webpackで作る Vueコンポーネント開発環境

バーチャー / GMO Pepabo, Inc.

2018.03.16 フロントエンドテックミーティング#1



バーチャー

@hypermkt  

ソフトウェアエンジニア

PHPer

<http://blog.hypermkt.jp>



はじめに3つ

対象者

- ・ Vue.jsの基礎文法が分かる
- ・ Vue.jsで次のレベルに行きたい



ゴール

- ・ Vue.jsでコンポーネント開発が始められるようになる



今日お話すること

- ・ 現状の課題
- ・ 課題解決
- ・ Webpackと仲良くなるう
- ・ Vue.jsのコンポーネント開発環境を作る
- ・ まとめ



話さないこと

- ・コンポーネントの設計



現状の課題

課題1.

Vue.jsの文法の基礎は分かった
次のレベルに行くために何をすべきか

Vue.js部で上がる声

- 👂 どうやったらVue.jsの技術が向上できるのか
- 👂 基礎文法の次に何をすべきか

※Vue.js部とはペパボ社内のVue.js勉強会



課題2.

既存のHTML, CSS, JavaScript
ファイルが複雑

既存のHTML, CSS, JavaScriptファイルが複雑



HTMLの行数が長すぎて読解困難



CSSクラスの使用箇所が分からない。

削除したらデザイン崩れが発生



JavaScriptの関数の使用箇所が分からない。

削除したら誤動作した

header2.tpl

javascript.js



課題解決



Vue.jsの機能で解決できないか？

コンポーネントなら解決できる!

コンポーネント

- ・HTML要素を再利用可能にカプセル化しカスタムタグとして利用できる

```
var Header = {  
  template: '<div class="title"><p>A custom component!</p></div>'  
}  
  
new Vue({  
  // ...  
  components: {  
    'my-header': Header  
  }  
})
```

```
<div id="example">  
  <my-header></my-header>  
</div>
```

A custom component!

コンポーネントのメリット

- 👍 HTMLタグの固まりをコンポーネントに集約できる
- 👍 可読性があがる
- 👍 コンポーネントを再利用できる
- 👍 コンポーネント内に関連するJavaScript処理もまとめられる



コンポーネントのデメリット

😱 JavaScriptとHTMLコードが混ざって見づらい…

😱 HTMLの修正困難…

😱 多数のコンポーネントを宣言すると管理しづらい…

```
var PreviewButton = {
  template: '<div>' +
    '<a :href="theme.preview_url" class="common__button customize__download__template_list__item__button__preview js-ti' +
    '</div>',
  props: ['theme']
}

var DownloadButton = {
  template: '<div :class="{ complete: isStep1, end: isStep2 }">' +
    '<a :href="url" class="common__button customize__download__template_list__item__button__download js-customize__dow' +
    ' :id="\`btn_\`' + theme.id" :class="{ customize__download__template_list__item__button__buy_already: isPurchased()' +
    '</div>',
}

var ActionButtons = {
  template: '<div>' +
    '<div class="customize__download__template_list__item__action clearfix" :class="isHover">\n' +
    '<template v-if="theme.charge_flg">\n' +
    '<template v-if="theme.ordered == \'1\'">\n' +
```

あれ・・・

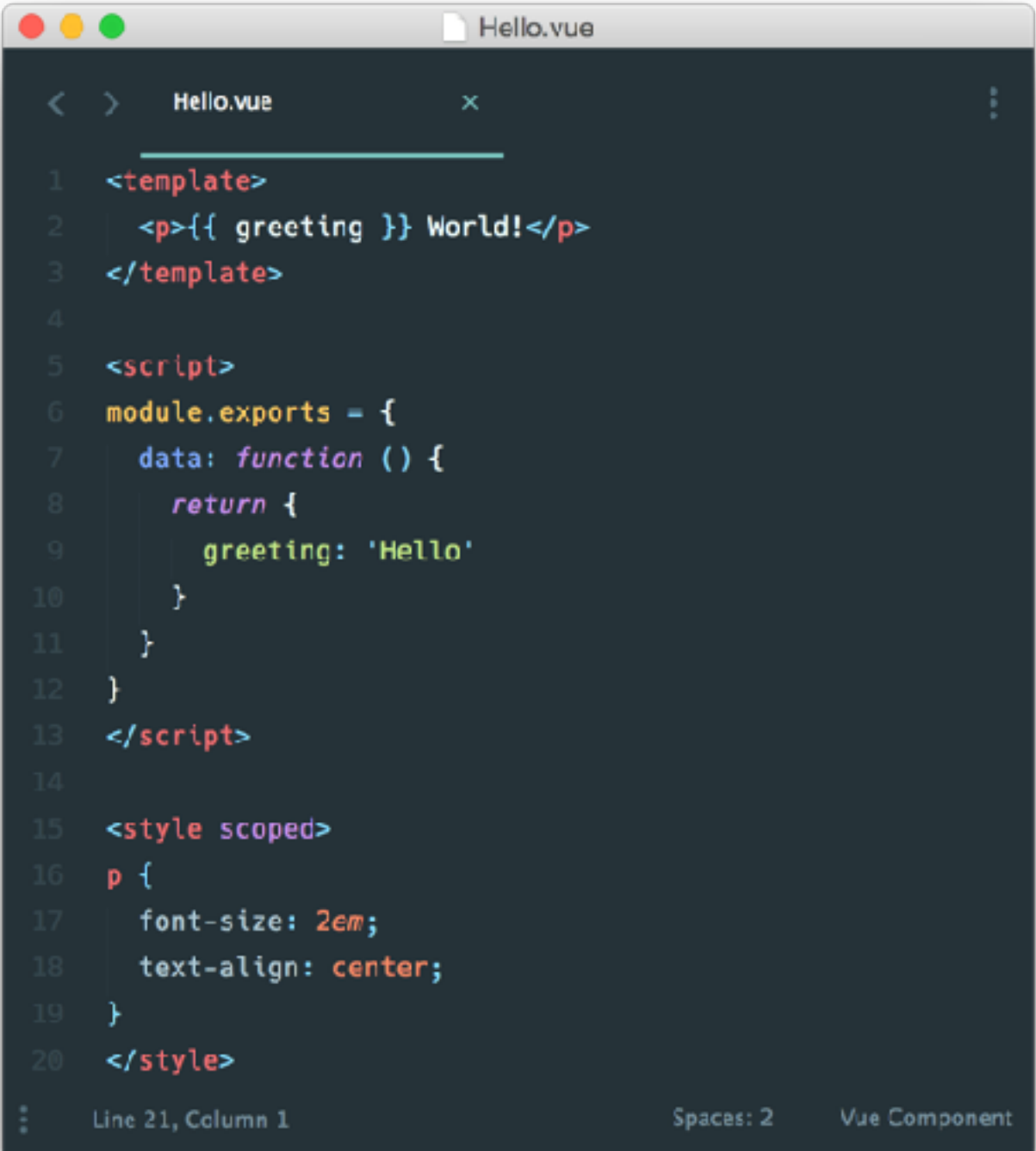
楽しようとしたはずが、
苦痛へ逆戻り 🤯

そこで
単一ファイルコンポーネント！！



単一ファイルコンポーネント

- ・コンポーネント毎にHTML, JavaScript, CSSコードを1つのファイルに分けることができる
- ・拡張子は.vue



```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6 module.exports = {
7   data: function () {
8     return {
9       greeting: 'Hello'
10    }
11  }
12 }
13 </script>
14
15 <style scoped>
16 p {
17   font-size: 2em;
18   text-align: center;
19 }
20 </style>
```

単一ファイルコンポーネント無し



```
<body id="home" class=' is__responsive'>
```

```
<header>
```

```
<div id="header" class="common__header__area js-common__header__area">
```

```
<div class="common__header__wrap clearfix">
```

```
<h1 class="common__header__logo__wrap">
```

```
<a href="/" title="ホームページ作成サービスなら「グーペ」" class="common__header__logo__link" id="logo">
```

```

```

```
</a>
```

```
</h1>
```

```
<div class="common__header__menu__list__signup__sp__wrap js-common__header__menu__list__signup__sp__wrap">
```

```
<a href="/signup/" class="common__button common__button__signup header__menu__list__signup__sp"
id="header_start_btn_free_sp">はじめる</a>
```

```
</div>
```

```
<nav>
```

```
<span class="fa fa-menu common__header__menu__icon js-common__header__menu__icon"></span>
```

```
<span class="fa fa-cross common__header__menu__close__icon js-common__header__menu__close__icon"></span>
```

```
<div class="common__header__menu__wrap js-common__header__menu__wrap">
```

```
<ul class="common__header__menu__list__wrap clearfix">
```

```
<li class="hidden__pc__block visible__tb__block visible__sp__block">
```

```
<a href="/" class="header__menu__list__title__home header__menu__sp__visible" id="menu_0_home">ホーム</a>
```

```
</li>
```


単一ファイルコンポーネント有り



```
<body>  
  <header-menu></header-menu>  
  <header-slide></header-slide>  
  <signup-button></signup-button>  
</body>
```

コンポーネント単位で
ファイル分割できる

HeaderMenu.vue

HeaderSlide.vue

SignupButton.vue

単一ファイルコンポーネントを使うには

- ・Webpack, Broserifyなどのモジュールバンドラーが必要

Webpackを使おう

- ・ 主流はWebpack
- ・ WebpackはCSS/画像などもまとめることができる
- ・ BroserifyはJavaScriptファイルのみ



vue-cliというのがあります

vue-cli とは

- Vue.js公式のコマンドラインインターフェース
- コマンドラインで高速かつ簡単に開発環境が構築できる
- <https://github.com/vuejs/vue-cli>



Welcome to Your Vue.js App

For guide and recipes on how to configure / customize this project,
check out the [vue-cli documentation](#).



最初から vue-cli を使用するのはオススメしない

- ・理由として
 - ・ Webpack, Babelの設定方法の理解につながらない
 - ・ 初期で生成される設定ファイルが複雑なので修正困難
- ・ 自分の考えとして
 - ・ 自分が理解・管理できるものを使ってほしい



まとめ

- ・ Webpack + Vue.jsで単一ファイルコンポーネントを使用して、コンポーネント開発を始めよう



Webpackと仲良くなるう

前提

- ・ webpack v4のお話です
 - ・ 2018/02/25にリリースされたばかり
- ・ npm 5.6 はインストール済みとする



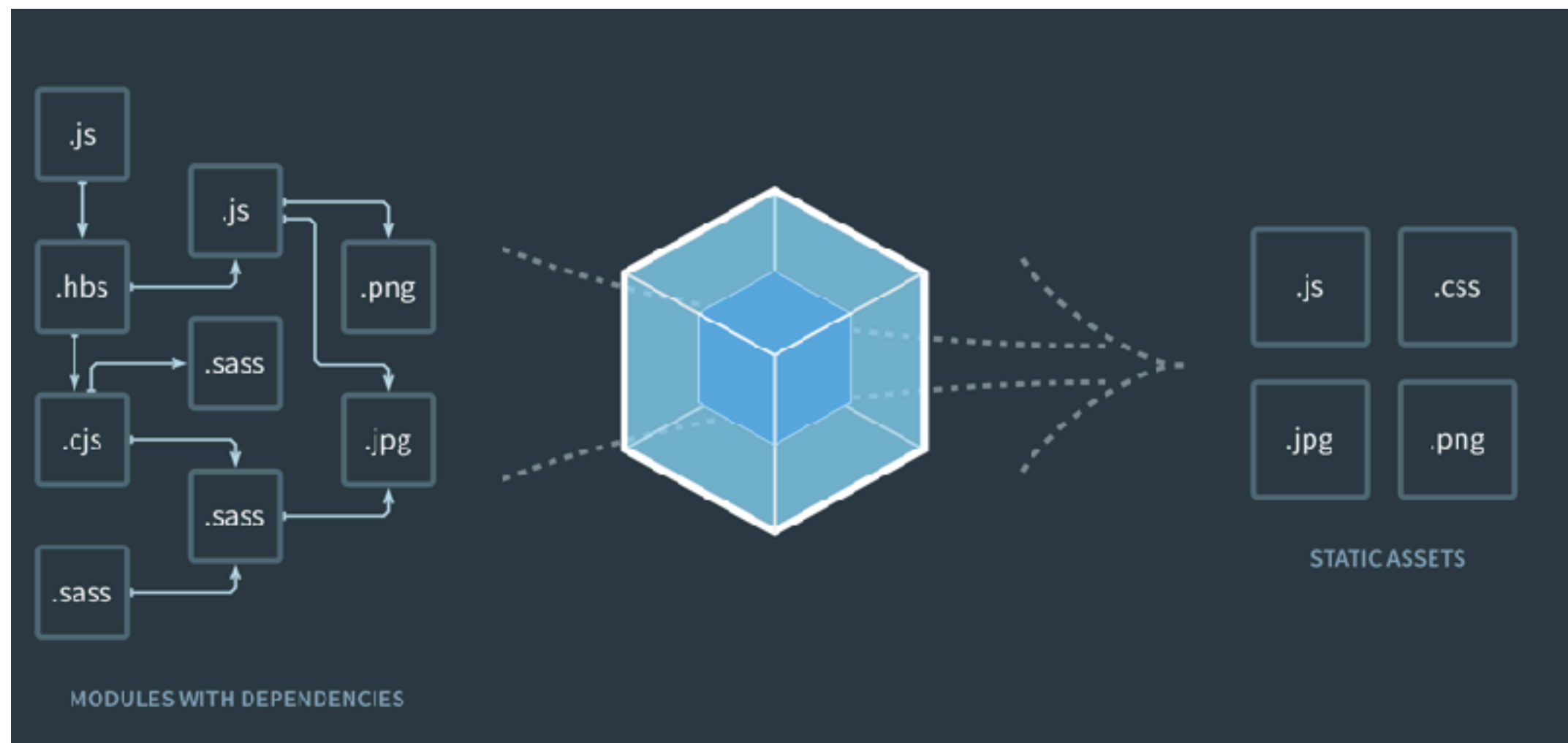
Webpackとは何か？



**モダンなJavaScriptアプリケーション向けの
モジュールバンドラーである**

モジュールバンドラーとは

- ・複数のモジュール(.js, .sassなど)を一つのファイルにバンドルして(まとめて)くれるもの

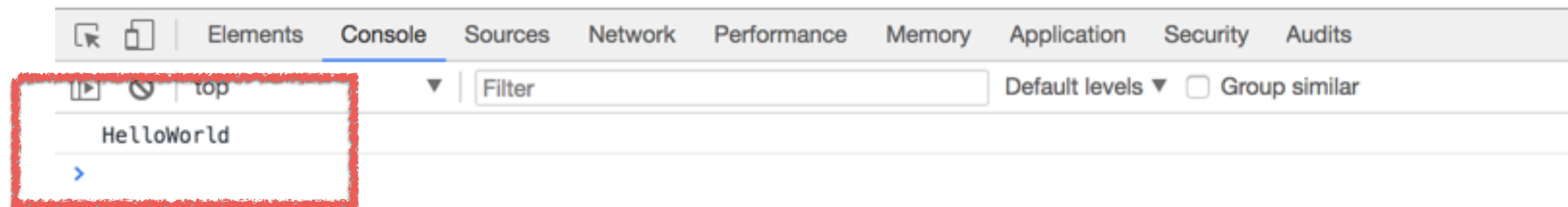
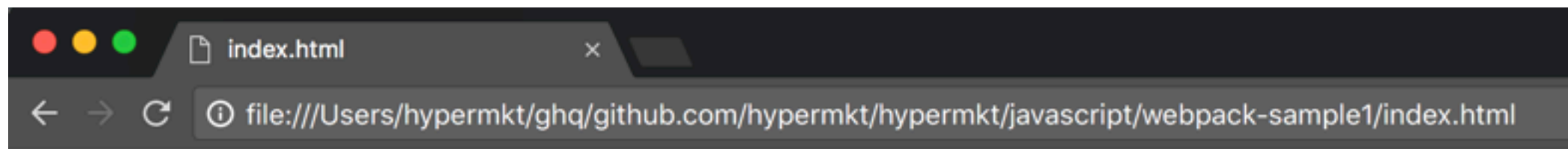


出典: <https://webpack.js.org/>

まずは簡単に触ってみよう📖

やってみること①

- ・ 2つのJavaScriptモジュールファイルを結合してみる
- ・ ブラウザのConsoleタブにHello Worldと出力しよう



ファイルとディレクトリ構成

```
$ tree .
```

```
├── index.html
├── package-lock.json
├── package.json
├── src
│   ├── app.js
│   ├── hello.js
│   └── world.js
└── webpack.config.js
```

```
1 directory, 7 files
```



Webpackのインストール

```
$ npm init -y  
$ npm install --save-dev webpack  
webpack-cli
```

- ・v4からコマンドラインインターフェースが分割されたので webpack-cliのインストールも必要



Webpack設定ファイル

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

webpack.config.js

```
module.exports = {
```

```
  // エントリーポイントの設定
```

```
  entry: './src/app.js',
```

```
  output: {
```

```
    // 出力ファイル名
```

```
    filename: 'bundle.js',
```

```
    // 出力先ディレクトリ
```

```
    path: path.join(__dirname, 'dist/')
```

```
  }
```

```
}
```

src/app.js

// 外部モジュールからエクスポートされた関数をインポートする

```
import hello from './hello.js';
import world from './world.js';

console.log(hello() + world());
```

src/hello.js

// export defaultで関数をモジュールとして公開する

```
export default function hello() {
  return 'Hello';
}
```

src/world.js

```
export default function world() {
  return 'World';
}
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
  </head>
  <body>
    <script src="dist/bundle.js"></script>
  </body>
</html>
```


webpackでビルドしてバンドルファイルを生成する

```
$ npx webpack
```

```
Hash: e7d757560a1bd755bcca
```

```
Version: webpack 4.1.1
```

```
Time: 265ms
```

```
Built at: 2018-3-15 23:17:15
```

Asset	Size	Chunks	Chunk
-------	------	--------	-------

Names

bundle.js	592 bytes	0	[emitted] main
-----------	-----------	---	----------------

```
Entrypoint main = bundle.js
```

```
  [0] ./src/app.js + 2 modules 205 bytes {0}
```

```
[built]
```

```
  | ./src/app.js 97 bytes [built]
```

```
  | ./src/hello.js 54 bytes [built]
```

```
  | ./src/world.js 54 bytes [built]
```

```
WARNING in configuration
```

```
The 'mode' option has not been set. Set 'mode'
option to 'development' or 'production' to
enable defaults for this environment.
```

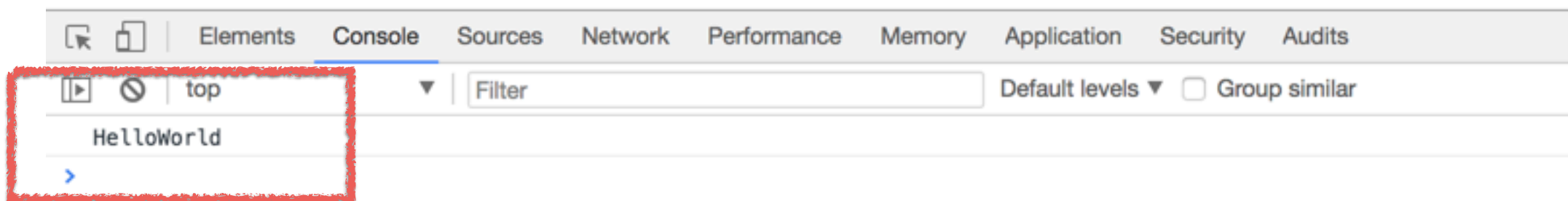
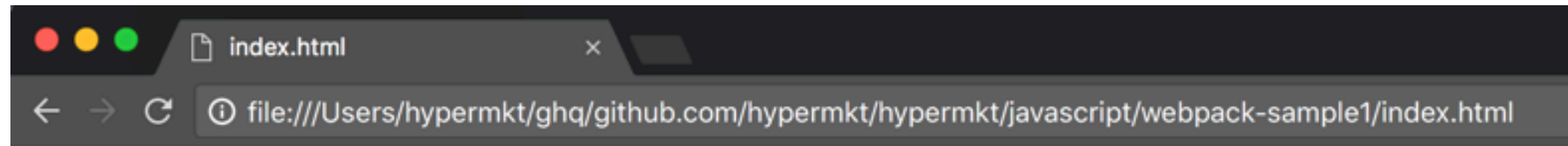
```
$ tree .
```

```
.
├── dist
│   └── bundle.js
├── index.html
├── src
│   ├── app.js
│   ├── hello.js
│   └── world.js
└── webpack.config.js
```

増えてる!!

```
2 directories, 6 files
```

ブラウザでバンドルファイルを読み込む



表示された!!

やってみること②

- ・ ES6が使えるようにしよう
- ・ 手順
 - ・ Webpackのローダー設定でES6形式のJavaScriptファイルが読み込めるように設定する



Babelのインストール

- babel-loader: BabelのWebpack用プラグイン
- Webpackに設定すればJavaScriptファイルをES6変換してくれる

```
$ npm install --save babel-loader  
babel-preset-es2015
```



ローダー設定

- ・ローダー設定とは、CSSなどJavaScript以外のファイルを読み込む設定

```
... // 省略
```

webpack.config.js

```
module: {  
  rules: [  
    {  
      test: /\.js$/, // ローダー対象の拡張子  
      use: [{  
        loader: 'babel-loader', // 利用するローダー  
        options: {  
          presets: ['es2015']  
        }  
      }]  
    }  
  ]  
}
```

実行してみよう

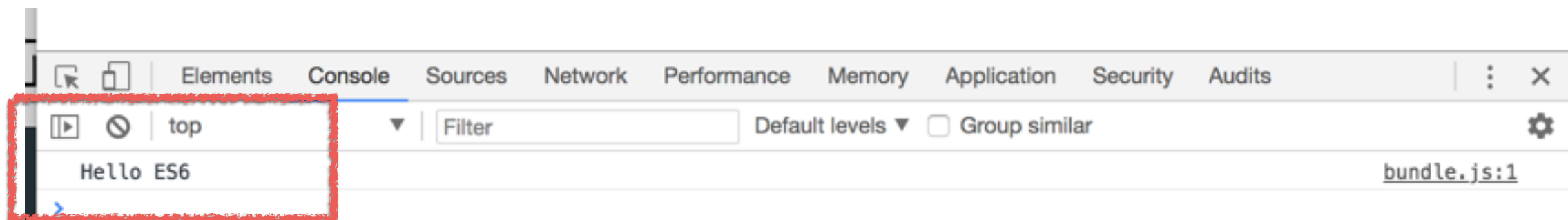
- ・ src/app.jsにES6記法の修正をしてnpx webpackを実行

```
import hello from './hello.js';  
import world from './world.js';
```

src/app.js

```
const message = hello() + ' ES6' // ES6の変数宣言を使用  
console.log(message);
```

- ・ ブラウザのConsoleログに出力される



表示された!!

ざっくりと一連の流れは
分かりましたでしょうか

開発用サーバー webpack-dev-server

- ・ ローカルサーバーを起動できる
- ・ ファイルの変更を検知して自動ビルドして、ブラウザ側も自動的にリロード
- ・ 少ない設定ですぐに利用できる



webpack-dev-serverの設定手順

① webpack-dev-serverをインストール

```
$ npm install --save-dev webpack-dev-server
```

② 配信元ディレクトリを指定する

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.join(__dirname, 'dist')
  },
  devServer: {
    contentBase: path.join(__dirname, 'dist')
  },
}
```

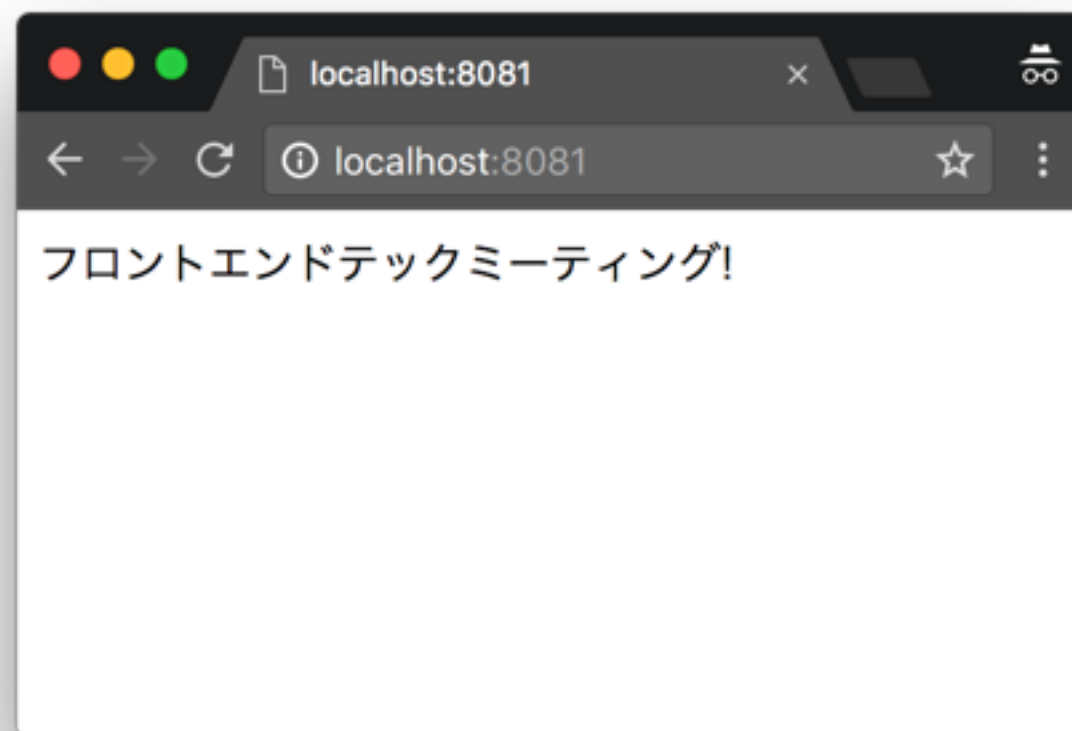
webpack.config.js



webpack-dev-serverの設定手順

③ webpack-dev-serverを起動

```
$ npx webpack-dev-server
```



後半

Vue.jsのコンポーネント開発環境を
作ろう

手順

- 1.環境構築
- 2.Webpackのインストール
- 3.Webpack開発サーバーの設定
- 4..vueが読めるように設定する
- 5.ローダー設定
- 6.単一ファイルコンポーネントの表示設定
- 7.コンポーネントファイルの作成



環境構築

- ・まずはプロジェクトのディレクトリの中で
package.jsonを作成する

```
$ mkdir webpack-with-vue  
$ cd webpack-with-vue  
$ npm init -y
```



Webpackのインストール

- ・次にWebpackと開発用サーバーをインストール
- ・ここまでは簡単ですね

```
$ npm install --save-dev webpack webpack-cli  
webpack-dev-server
```



Webpack開発サーバーの設定

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.join(__dirname, 'dist')
  },
  devServer: {
    contentBase: path.join(__dirname, 'dist')
  },
}
```

webpack.config.js

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title></title>
</head>
<body>
  Hello World
</body>
</html>
```

dist/index.html

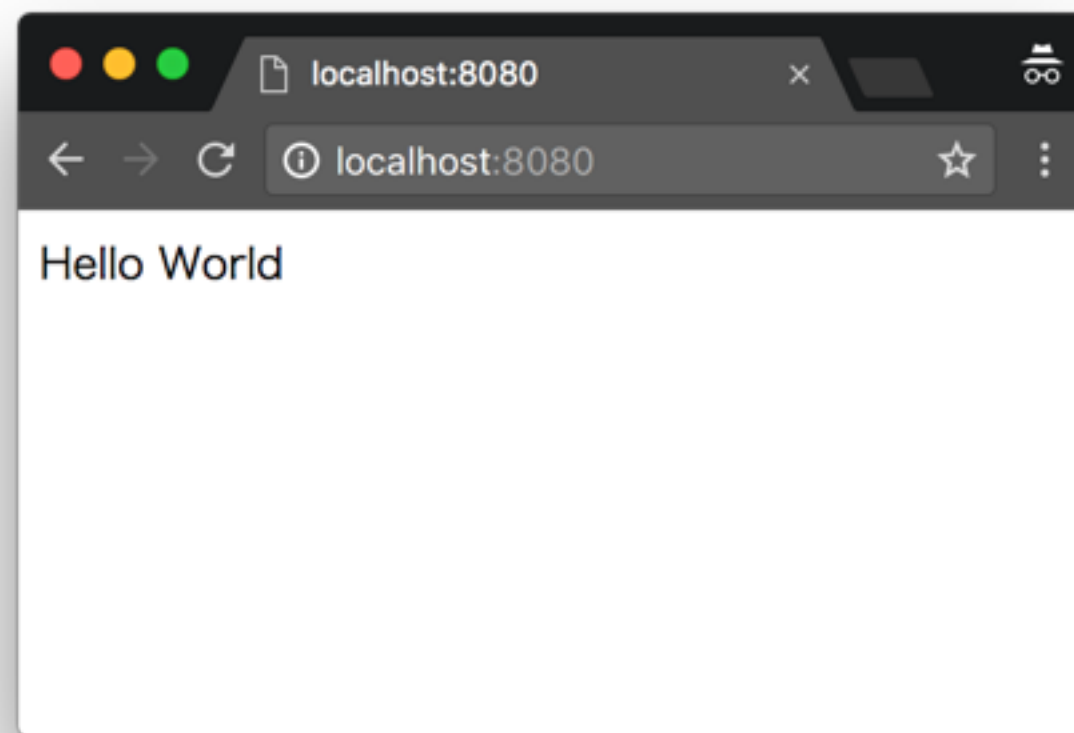
中身は空

src/index.js

開発環境サーバーを確認しよう

```
$ npx webpack-dev-server
```

コンソール上



.vueが読めるように設定する

Vueをインストールする

```
$ npm install --save vue
```

Vue.jsの単一ファイルコンポーネント(.vueファイル)を扱うために以下をインストール

```
$ npm install --save-dev vue-loader vue-template-compiler css-loader
```



ローダー設定

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

module.exports = {
  entry: './src/index.js',
  output: {
    filename: 'bundle.js',
    path: path.join(__dirname, 'dist')
  },
  module: {
    rules: [
      {
        test: /\.vue$/,
        loader: 'vue-loader',
      }
    ]
  },
  devServer: {
    contentBase: path.join(__dirname, 'dist')
  },
}
```

webpack.config.js



単一ファイルコンポーネントの表示設定

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <script src="./bundle.js"></script>
  <title></title>
</head>
<body>
  <div id="app"></div>
</body>
</html>
```

dist/index.html

- WebpackでビルドしたJSを読み込む
- Vueでマウントする箇所を指定

```
import Vue from 'vue';
import App from './components/App.vue';

window.onload = function() {
  new Vue(App).$mount('#app');
}
```

src/index.js

- 画面の読み込みが完了したら
#appにApp.vueをマウント(置き換え)



コンポーネントファイルの作成

```
<template>
  <div>
    <header-component></header-component>
    <p class="blue">This is App component.</p>
  </div>
</template>
```

```
<script>
import Header from './Header.vue';
```

```
export default {
  components: {
    'header-component': Header
  }
}
</script>
```

```
<style>
.blue {
  color: blue;
}
</style>
```

src/components/App.vue

```
<template>
  <div>
    <p class="title">{{ title }}</p>
  </div>
</template>
```

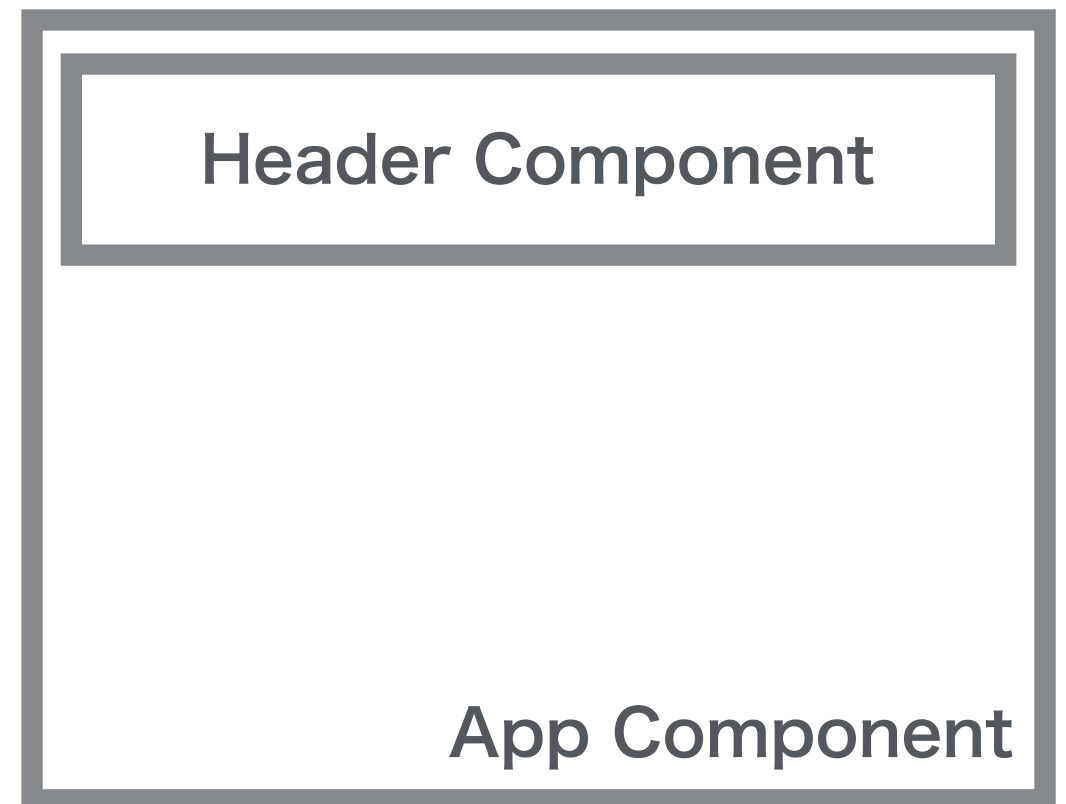
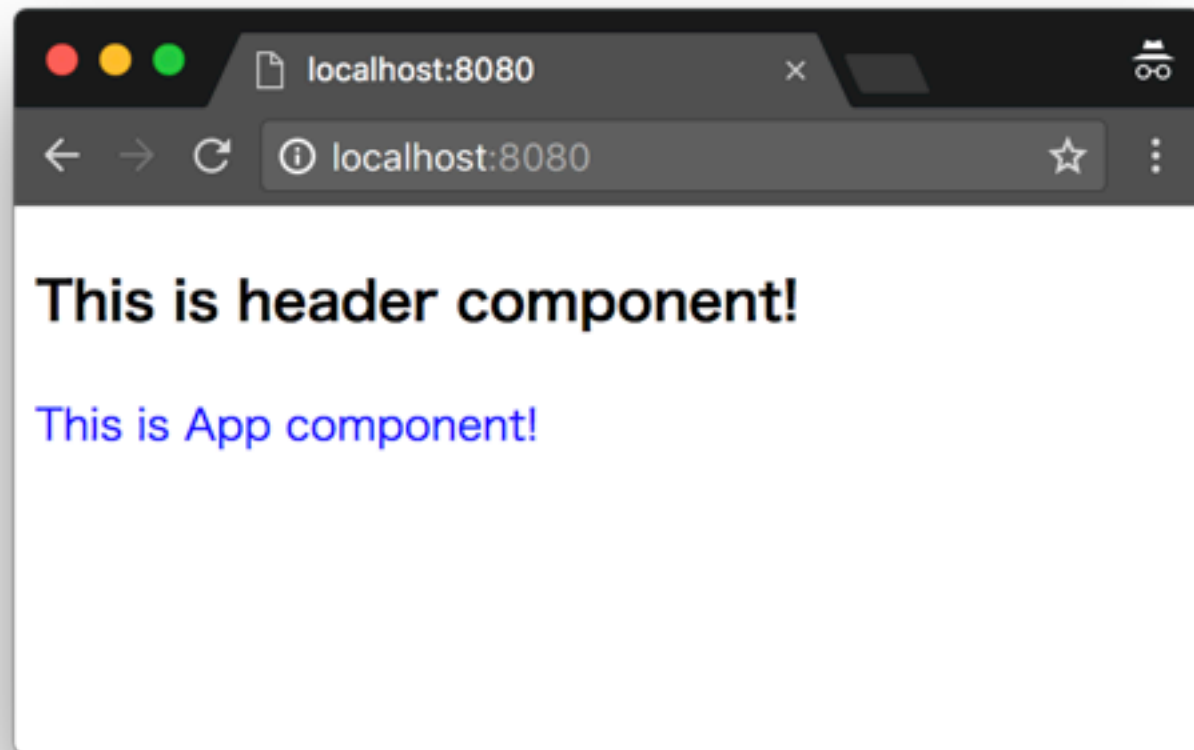
```
<script>
export default {
  data: function() {
    return {
      title: 'This is Header component'
    }
  }
}
</script>
```

```
<style>
.title {
  font-size: 20px;
  font-weight: bold;
}
</style>
```

src/components/Header.vue



ブラウザで表示すると



ぜひ日報アプリを
単一ファイルコンポーネント化
してみてください!

まとめ

まとめ

- ・ Webpackの概要、簡単な使い方が分かった
- ・ Vue.jsの単一ファイルコンポーネントのビルド環境の構築方法が分かった
- ・ **Vue.jsの次のレベルへ行こう！！**

