Ask Company

리액트와 함께 장고 시작하기 / 리액트

babel과 webpack 그리고 create-react-app

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

본 챕터의 목표

~~babel/webpack에 대한 깊은 이해 X~~

babek/webpack의 역할에 대한 이해 O

우리는 본 코스에서 create-react-app 유틸리티를 통해 미리 세팅된 babek/webpack을 활용할 것입니다.

다양한 babel preset

babel-preset-es2015 : ES6

babel-preset-es2016 : ES7

babel-preset-es2017 : ES8

babel-preset-env

- 디폴트로 동작으로 ES6 이상의 preset을 적용하여, ES5로 transpiling
- 개별 지정보다 본 preset을 권장
- https://babeljs.io/env

webpack (module bundler)

javascript, jsx, css, sass, less, es6, 이미지, HTML, 폰트 등 거의 모든 것이 모듈이 될 수 있으며, 하나의 파일 (bundle)로 묶을 수 있다.

모듈성과 네트워크 성능 향상

Features

- 코드를 필요할 때, 로딩 가능
- Minifying : 불필요한 코드, 공백/줄바꿈, 긴 이름 등을 줄여, 파일 크기 줄이기
- HMR (Hot module replacement) : 개발모드에서 원본 소스코드 변경을 감지하여, 변경된 모듈만 즉시 갱신

지원 Loaders

- babel-loader : ES6나 리액트 코드를 transpiling
- css-loader : 설정에 따라 postcss-loader, sass-loader를 추가로 설정. css를 HTML내에서 <link /> 엘리먼트로 포함시킬 필요없이 JS/JSX단에서 임포트하여 React 컴포넌트에 즉시 적용 가능

webpack 빌드 (babel-loader)

필요한 유틸리티 및 라이브러리 설치

```
# 유틸리티 설치
npm install --global yarn
yarn global add webpack webpack-cli
```

상황에 따라 관리자/루트 권한이 필요할 수 있습니다.

webpack 버전 4.38.0 에서 확인하였습니다.

```
# 개발 시에 필요한 라이브러리 설치
yarn add --dev @babel/core babel-loader @babel/preset-env @babel/preset-react
```

현재 디렉토리의 ./node_modules/ 에 설치

```
# 프로덕션에서 필요한 라이브러리 설치
yarn add react react-dom
```

설정파일없이 webpack-cli 명령행 옵션 만으로 빌드하기

프로젝트경로/src/index.js

```
import React from "react";
import { render } from "react-dom";

const HomeComponent = ({ fruits }) =>
  <div>
    <h1>좋아하는 과일</h1>
    <ul>
      {fruits.map((name, idx) => (
        <li key={idx}>{name}</li>
      ))}
    </ul>
  </div>

const fruits = ['사과', '바나나'];

render(
  <HomeComponent fruits={fruits} />,
  document.getElementById('container')
)
```

프로젝트경로/.babelrc 설정파일 생성

```
{
  "presets": [
    "@babel/preset-env",
    "@babel/preset-react",
  ]
}
```

다음 쉘 명령으로 프로젝트경로/dist/main.js 경로에 bundle 생성

```
프로젝트경로> webpack --mode=development --module-bind js=babel-loader
```

프로젝트경로/index.html 에서 dist/main.js를 활용해봅시다.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>Sample HTML</title>
</head>
<body>
  <div id="container"></div>
  <script src="./dist/main.js"></script>
</body>
</html>
```

설정파일(최소설정)과 함께 webpack-cli 빌드하기

프로젝트경로/webpack.config.js

```
module.exports = {
  mode: 'development',
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /(node_modules)/,
        loader: 'babel-loader',
        query: {
          presets: [
            '@babel/preset-env',
            '@babel/preset-react'
          ]
        }
      }
    ]
  }
};
```

프로젝트경로/.babelrc 설정파일 생성

```
{
  "presets": [
    "@babel/preset-env",
    "@babel/preset-react",
  ]
}
```

다음 쉘 명령으로 프로젝트경로/dist/main.js 경로에 bundle 생성

```
프로젝트경로> webpack
```

프로젝트경로/index.html 에서 dist/main.js를 활용해봅시다.

sourceMap을 통해, 브라우저에서 원본 소스코드와 함께 디버깅 설정

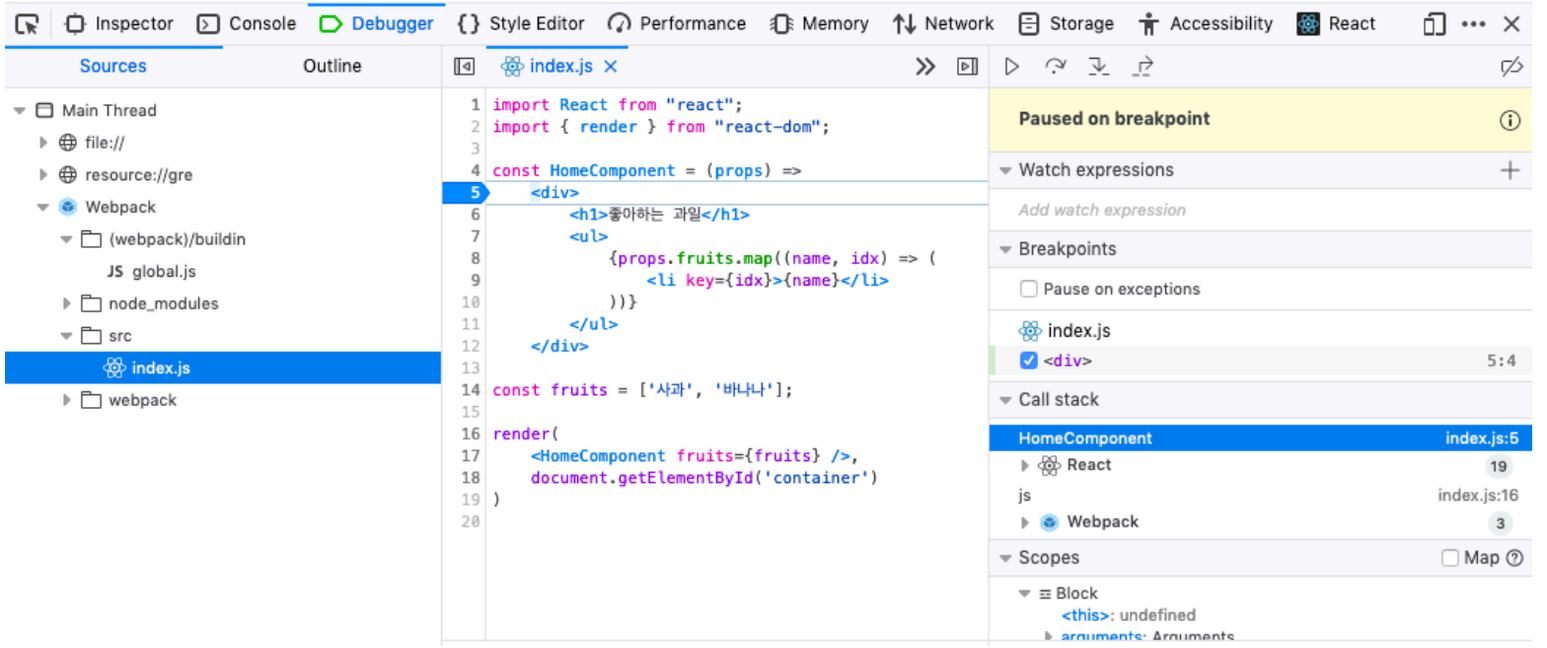
프로젝트경로/webpack.config.js

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

module.exports = {
  mode: 'development',
  output: {
    path: path.resolve(__dirname, './dist'),
    filename: 'main.js',
    sourceMapFilename: 'main.map',
  },
  devtool: '#source-map',
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /(node_modules)/,
        loader: 'babel-loader',
        query: {
          presets: [
            '@babel/preset-env',
            '@babel/preset-react'
          ]
        }
      }
    ]
  }
};
```

최근에 절대경로로 지정하는 것으로 변경

아래 Firefox에서는 Debugger 탭, Chrome에서는 Sources 탭



webpack 빌드 (css-loader)

관련 라이브러리 설치

css-loader

```
쉘> yarn add --dev style-loader css-loader postcss-loader autoprefixer
```

- CSS 파일 읽기

style-loader

- head 엘리먼트에 CSS 주입. css-loader와 함께 쓰기를 권장

postcss-loader

- PostCSS : 자바스크립트 플러그인을 통해 CSS를 변환하는 툴 → 즉, 플러그인 베이스의 자동화 툴
- 앞으로 표준이 될 CSS문법을 확장/지원하는 CSS계의 Babel

autoprefixer

- CSS의 vendor prefix 자동 처리 플러그인
- ex) -webkit-, -moz-, -ms-

https://webpack.js.org/loaders/css-loader/
https://webpack.js.org/loaders/style-loader/
https://webpack.js.org/loaders/postcss-loader/
https://github.com/michael-ciniawsky/postcss-load-config

예시

프로젝트경로/webpack.config.js

```
module.exports = {
  module: {
    rules: [
      {
        test: /\.jsx?$/i,
        중략 ...
      },
      {
        test: /\.css$/i,
        use: [
          'style-loader',
          'css-loader',
          {
            loader: 'postcss-loader',
            options: {
              plugins: () => [
                require('autoprefixer')
              ]
            }
          }
        ],
      }
    ]
  }
};
```

3개의 loader 설정

프로젝트경로/src/home.css

```
.fruit {
    color: red;
}
```

프로젝트경로/src/home.js

```
import React from "react";
import "./home.css";

const HomeComponent = ({ fruits }) =>
  <div>
    <h1>좋아하는 과일</h1>
    <ul>
      {fruits.map((name, idx) => (
        <li key={idx} className="fruit">
          {name}
        </li>
      ))}
    </ul>
  </div>

export default HomeComponent;
```

프로젝트경로/src/index.js

```
import React from "react";
import { render } from "react-dom";
import HomeComponent from "./home";

const fruits = ['사과', '바나나'];

render(
  <HomeComponent fruits={fruits} />,
  document.getElementById('container')
)
```

다양한 PostCSS 플러그인

postcss-color-function: color modifier 지원

postcss-simple-vars: Sass 형식 변수 지원

postcss-nesting: Nesting 문법 지원

postcss-for: for문 지원

postcss-assets: url내에 들어가는 파일의 경로 계산이나 이미지의 width 측정

precss: Sass처럼 사용하기위한 플러그인 모음

cssnext: 아직 지원하지 않는 새로운 구문 지원

이외에도 수백개의 플러그인들이 있습니다. ;)

https://github.com/postcss/postcss/blob/master/docs/plugins.md

create-react-app

react 프로젝트를 생성해주는 명령행 도구

create-react-app

버전마다 생성되는 프로젝트 구성이 달라질 수 있습니다.
기존 프로젝트에 지속적으로 버전 업데이트가 필요합니다.

webpack, babel, eslilnt 등의 기본 설정이 된 리액트 프로젝트 생성

설치> yarn global add create-react-app

버전: 3.0.1 기준

프로젝트 생성

쉘> create-react-app --help

쉘> create-react-app <프로젝트-디렉토리>

```
Success! Created webpack03 at /Users/allieus/FunnyClass/test-nodejs/webpack03
Inside that directory, you can run several commands:

  yarn start
    Starts the development server.

  yarn build
    Bundles the app into static files for production.

  yarn test
    Starts the test runner.

  yarn eject
    Removes this tool and copies build dependencies, configuration files
    and scripts into the app directory. If you do this, you can't go back!

We suggest that you begin by typing:

  cd webpack03
  yarn start

Happy hacking!
```

참고) 또 다른 CRA 프로젝트 생성 명령

yarn create react-app <프로젝트-디렉토리>
npx create-react-app <프로젝트-디렉토리>

참고) TypeScript 언어 지원이 들어간 프로젝트

yarn create react-app <프로젝트-디렉토리> --template typescript
npx create-react-app <프로젝트-디렉토리> --template typescript

개발서버 구동

쉘> cd <디렉토리>

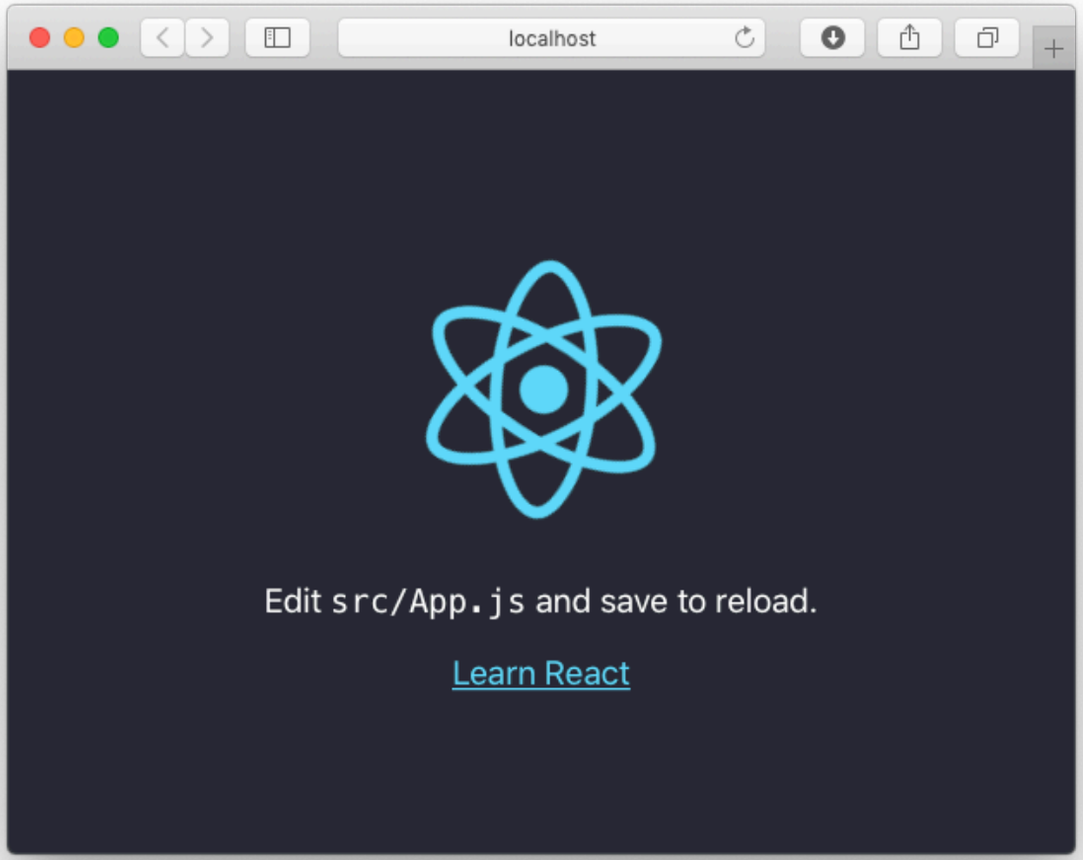
쉘> yarn start

```
Compiled successfully!

You can now view webpack02 in the browser.

  Local:            http://localhost:3000/
  On Your Network:  http://172.30.1.57:3000/

Note that the development build is not optimized.
To create a production build, use yarn build.
```



src/App.js 파일을 수정하고, 저장해보세요.
새로고침없이도 자동으로 UI가 업데이트됩니다.

상대경로 import를 절대경로로 지정하기

다음 내용을 작성 : jsconfig.json

Visual Studio Code에서도 절대경로 인지

```
{
    "compilerOptions": {
        "baseUrl": "src"
    },
    "include": ["src"]
}
```

WebStorm/PyCharm/IntelliJ 등을 사용할 경우

src 디렉토리를 Resource root로 지정이 필요 ← Mark directory as / Resource root

Life is short.
You need Python and Django.

I will be your pacemaker.