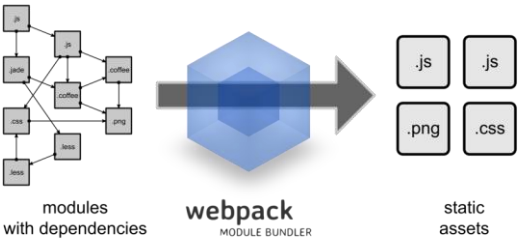
1. webpack이란?(1)

webpack

- 자바스크립트 모듈 번들러
  - 모듈들이 포함하는 정적 자원(CSS, image 등) 들을 번들링하여 모듈을 생성함
- 장점
  - 초기 로딩 타임을 줄인다.
  - 정적 자원(CSS, Image) 등까지 모듈화시킨다.
  - 모듈로 3rd party 라이브러리를 통합할 수 있다.
  - 대규모 프로젝트에 적합하다.
  - npm 패키지를 사용할 수 있다.
  - babel과의 통합성이 좋다
  - HMR(Hot Module Replacement) 지원
    - 코드가 수정될 때마다 페이지 자동갱신

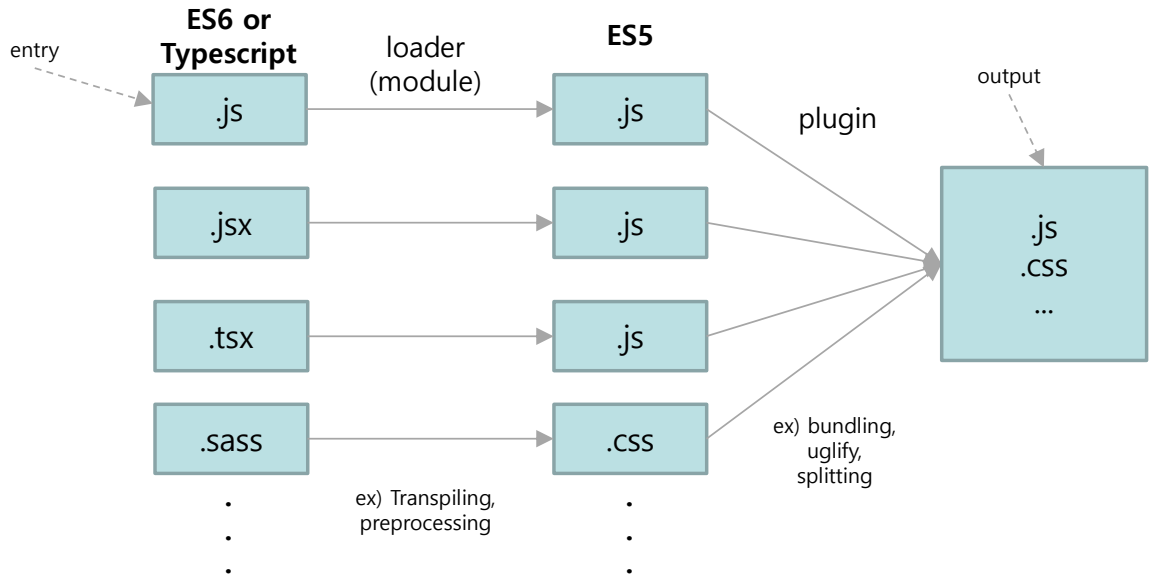


1

■ webpack의 모든 구성요소를 상세히 알아보는 것은 이 장의 목적이 아니다. 전반적인 webpack 기반의 웹앱 프로젝트 구조를 이해하고 필요하다면 설정을 추가하거나 변경할 수 있도록 하는 것이 목적이다.

1. webpack이란?(2)

webpack과 react



2

■ webpack과 같은 모듈 번들러를 학습해야 하는 이유

- 수백개의 독립된 모듈들을 몇개의 .js 파일로 묶어서 배포하도록 함.
- 전처리나 빌드과정에 개입해 여러가지 작업을 할 수 있음
- React와 같은 프론트엔드 프레임워크들이 모듈번들러를 대부분 사용하고 있음
  - CRA(create-react-app), @vue/cli 등의 도구들이 Webpack 기반으로 만들어져 있음

■ Webpack을 이해하면 얻어지는 장점

- 생성된 Vue, React 프로젝트들의 구조를 이해할 수 있음
- 프로젝트에 대한 정교한 설정을 할 수 있게됨.

2. webpack의 설치

전역 설치

- npm install -g webpack webpack-cli

프로젝트에 개발 버전으로 설치

- npm init ==> 프로젝트 생성
- npm install -D webpack webpack-cli

개발 서버 설치

- npm install -D webpack-dev-server

3

■ CRA V5 기준 Webpack 관련 주요 패키지 버전

- webpack : 5.64.x
- webpack-dev-server : 4.6.x

■ 지정된 버전으로 패키지를 설치하려면 다음과 같이 표현한다.

- npm install --save-dev webpack@5.x.x --> webpack v5의 최신버전 설치

3. webpack 예제 1(1)

프로젝트 디렉토리 생성

- mkdir webpack-test
- cd webpack-test
- npm init

webpack 및 개발용 서버 패키지 설치

- npm install -D webpack@5.x.x webpack-cli@4.x.x serve

3. webpack 예제 1(2)

번들링 테스트

- src/employees.js

```
var employees =  [
    { "no":1001, "name" : "홍길동", "email":"gdhong@opensg.net", "mobile":"010-2222-3331" },
    { "no":1002, "name" : "이몽룡", "email":"mrlee@opensg.net", "mobile":"010-2222-3332" },
    { "no":1003, "name" : "성춘향", "email":"chsung@opensg.net", "mobile":"010-2222-3333" },
    { "no":1004, "name" : "박문수", "email":"mspark@opensg.net", "mobile":"010-2222-3334" },
    { "no":1005, "name" : "변학도", "email":"hdbyun@opensg.net", "mobile":"010-2222-3335" }
];
module.exports = employees;
```

- src/App.js

```
let employees = require('./employees');
var str = "";
for (var i=0; i < employees.length; i++) {
    str += '<div>' + employees[i].name + ' : ' + employees[i].email +
            ', ' + employees[i].mobile + '</div>';
}
document.getElementById('root').innerHTML = str;
```

3. webpack 예제 1(3)

- build/index.html

```
<!DOCTYPE html>
<html lang="ko">
<head>
  <meta charset="UTF-8">
  <title>웹팩 예제</title>
</head>
<body>
  <div id="root"></div>
  <script src="bundle.js"></script>
</body>
</html>
```

3. webpack 예제 1(4)

webpack

- webpack.config.js 파일이 존재하면(기본 설정 파일)
  - webpack
  - 최소한의 webpack.config.js 파일의 예

```
module.exports = {
    mode : 'development',
    entry: __dirname + '/src/App.js',
    output: {
        path: __dirname + '/build',
        filename: 'bundle.js'
    }
};
```

- 다른 파일명을 사용하는 경우
  - webpack --config myconfig.js

7

■ __dirname은 현재 실행중인 스크립트가 포함된 디렉토리의 이름을 나타내는 node.js의 전역변수이다.

■ 작성이 되었다면 webpack 명령어만으로 실행하여 bundle.js 파일의 생성을 확인한다.

■ webpack을 Command line에서 실행할 수도 있지만 구성 파일을 작성하여 실행하는 것이 효과적이다.

- 다양한 loader, plugin을 적용해 변환, 분할, 번들링 등을 더욱 강력하게 할 수 있다.

3. webpack 예제 1(5)

디버깅을 위한 sourcemap 설정

- webpack.config.js 에 source-map 설정 추가
- task runner를 npm으로 지정

```
module.exports = {
    devtool : 'source-map',
    mode : 'development',
    entry: __dirname + '/src/App.js',
    output: {
        path: __dirname + '/public',
        filename: 'bundle.js'
    }
};
```

```
{
  "name": "webpack-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "webpack",
    "build" : "webpack --config webpack.prod.config.js"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "serve": "^13.0.2",
    "webpack": "^5.66.0",
    "webpack-cli": "^4.9.1"
  }
}
```

8

■ 웹팩의 설정 파일은 개발 버전과 운영 버전으로 나누어서 작성할 수 있다

■ 개발 버전(예: webpack.config.js)

- devtool 옵션을 'source-map'으로 부여
- mode를 development로 지정

■ 운영 버전(예: webpack.prod.config.js)

- webpack.config.js 파일의 내용을 복사한 후 다음의 내용을 수정한다.
- devtool 옵션 : 제거
- mode : 'production' 으로 지정

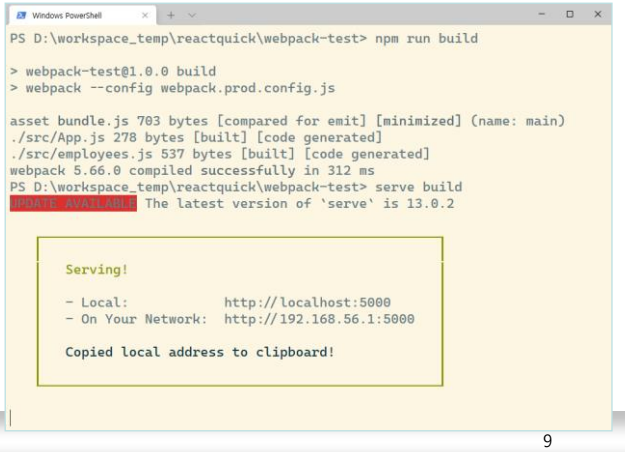
■ source-map의 기능

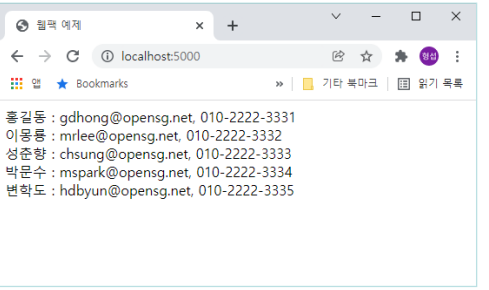
- 번들된 파일로 실행하지만 디버깅은 원본 파일로 디버깅함
- 디버깅 정보를 .js.map 파일이 지원함
- bundle.js + bundle.js.map

3. webpack 예제 1(6)

이제까지의 작성 결과 확인

- build 태스크 러너 실행 후 브라우저로 확인
- npm run build
- server build
  - build 디렉토리를 웹서버 루트로 설정해 서버 시작





9

■ npm run dev로 실행했을 때와 npm run build 로 실행했을 때의 build/bundle.js 파일을 살펴보자

- production mode : 코드 난독화 --> 번들 파일 크기 줄어듬
- development mode : 디버깅을 위한 번들

4. webpack 개발 서버(1)

webpack 개발 서버란?

- 로컬 개발을 위한 webpack 옵션
  - node.js + express 로 구성되어 있어서 별도의 http 서비스를 작성하지 않아도 됨.
- 정적 파일을 제공함.
  - 빌드한 내용을 메모리에 저장했다가 자동으로 브라우저 화면을 갱신할 수 있음.

npm 설치

- 로컬 개발 설치 : npm install -D webpack-dev-server@4.x.x
- 설치후 webpack config 파일에 devServer 옵션 추가

```
module.exports = {
    ......
   devServer : {
        static : './build',
        historyApiFallback :true,
        port : 7777,
        hot : true,
        open : true
    }
};
```

10

■ webpack config 파일에 devserver 옵션을 추가해야 한다.

- static : 프로젝트 루트가 아니라 다른 경로를 웹서버 루트 경로로 지정할 때
- port : 기본값 8080
- historyApiFallback : HTML5 history API를 이용하는 SPA를 개발할 때 유용. 이 값이 true이면 매핑되지 않은 개발 서버에 대한 요청시 /index.html로 라우팅됨.
- open : 웹서버가 시작하면 브라우저가 자동으로 열리도록 하기 위해 true로 설정
- hot : HMR(Hot Module Replacement) 지원을 위해 true로 지정

■ 이 설정은 webpack.prod.config.js에는 설정하지 않는다.

4. webpack 개발 서버(2)

개발 환경과 운영환경 분리를 위한 설정

- package.json에 start 태스크 러너 추가

```
{
  "name": "webpack-test",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "dev": "webpack",
    "build": "webpack --config webpack.prod.config.js",
    "start": "webpack-dev-server"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "serve": "^13.0.2",
    "webpack": "^5.66.0",
    "webpack-cli": "^4.9.1",
    "webpack-dev-server": "^4.7.3"
  }
}
```

11

■ webpack-dev-server의 옵션

- --hot : HMR(Hot Module Replacement) 지원. 코드 수정 후 저장하면 즉시 화면에 반영됨.
- --open : webpack 개발 서버 구동후 브라우저 자동 열림

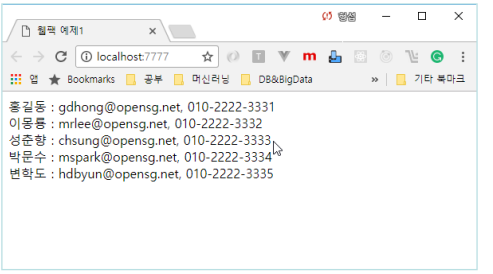
4. webpack 개발 서버(3)

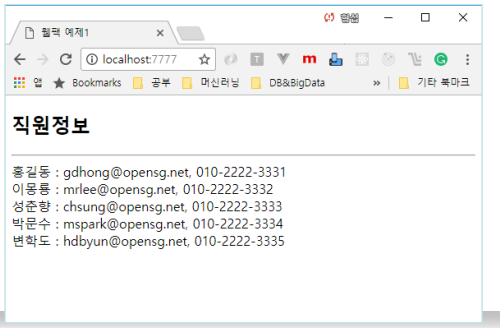
작성된 코드 실행하여 확인

- npm run start

- 코드 변경후 HMR 기능 확인
  - src/App.js

```
let employees = require('./employees');
var str = "";
str += "<h2>직원정보</h2><hr />"
for (var i=0; i < employees.length; i++) {
    str += '<div>' + employees[i].name + ' : '
            + employees[i].email + ', '
            + employees[i].mobile + '</div>';
}
document.getElementById('root').innerHTML = str;
```





12

■ webpack2 프로젝트 참조

5. loaders(1)

로더(loaders)란?

- 외부 스크립트와 도구를 이용해 소스파일, css, html, image 등에 대해 전처리, 변환 등의 작업을 적용할 수 있음
- 로더 리스트
  - https://webpack.js.org/loaders/ (공식)
  - https://github.com/webpack-contrib/awesome-webpack#loaders (third party)
- 주요 로더 : 정말 많다!
  - babel , json
  - css, file, sass, less, url
  - base64
  - coffee, coffee-jsx, coffee-redux
  - typescript

13

■ 로더가 작동하는 시점은 파일을 로딩할 때이다.

- 로딩하자마자 뭔가를 수행하고자 할 때...

5. loaders(2)

json-loader

- json 파일을 읽어와 JS 객체로 사용할 수 있도록 함.
- npm install -D json-loader
  - **webpack 2.0 부터는 추가할 필요 없음**
- webpack.config.js에 로더 등록
  - 정규식으로 파일 포맷을 정의한다.
  - .json 으로 끝나는 파일만...
  - **webpack 2.0부터는 아래 설정없이 로딩이 가능함.**

```
module.exports = {
  module: {
    loaders: [
      {
        test: /₩.json$/,
        loader: 'json-loader'
      }
    ]
  }
}
```

5. loaders(3)

- src/data.json 파일 작성

```
{
    "title": "직원 정보",
    "employees" : [
        { "no":1001, "name" : "홍길동", "email":"gdhong@opensg.net", "mobile":"010-2222-3331" },
        { "no":1002, "name" : "이몽룡", "email":"mrlee@opensg.net", "mobile":"010-2222-3332" },
        ......
    ]
}
```

- src/App.js 변경

```
var data = require("./data.json");

var str = "";
str += "<h1>" + data.title + "</h1><hr />"
for (var i=0; i < data.employees.length; i++) {
    var e = data.employees[i];
    str += '<div>' + e.name + ' : ' + e.email +
            ', ' + e.mobile + '</div>';
}
document.getElementById('app').innerHTML = str;
```

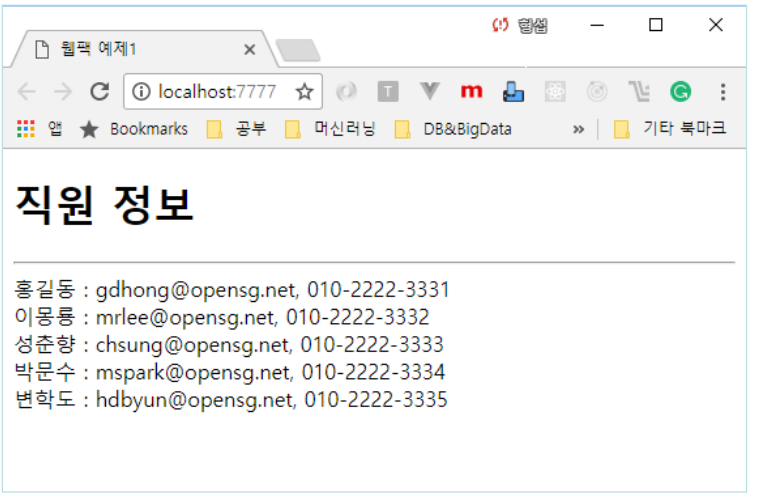
15

■ data.json 텍스트 파일을 마치 js 소스처럼 require(imort)함.

- 이것을 가능하도록 하는 것이 로더의 역할

5. loaders(4)

- json-loader 테스트



16

5. loaders(5)

babel-loader

- 기능
  - ES6 Code --> ES5로 변환
  - React의 JSX를 ES5 Code 로 변환
- Webpack과 궁합이 잘맞음
- 설치

```
//바벨 설치
npm install -D babel-loader @babel/core @babel/preset-env @babel/preset-stage-2
@babel/preset-react
npm install react react-dom
```

17

■ 설치된 버전 확인을 위해 package.json을 살펴보면 다음과 같음

```
{
  ......(중략)
  "devDependencies": {
    "@babel/core": "^7.16.7",
    "@babel/preset-env": "^7.16.8",
    "@babel/preset-react": "^7.16.7",
    "@babel/preset-stage-2": "^7.8.3",
    "babel-loader": "^8.2.3",
    "serve": "^13.0.2",
    "webpack": "^5.66.0",
    "webpack-cli": "^4.9.1",
    "webpack-dev-server": "^4.7.3"
  },
  "dependencies": {
    "react": "^17.0.2",
    "react-dom": "^17.0.2"
  }
```

```
}
```

5. loaders(6)

- babel-loader 사용을 위한 설정
  - webpack.config.js, webpack.prod.config.js 모두 추가

```
module.exports = {
    ……
  module: {
      rules: [
          {
              test: /₩.js$/,
              exclude: /(node_modules|bower_components)/,
              use: {
                  loader: 'babel-loader',
                  options: {
                      presets: ['@babel/preset-env', '@babel/preset-react'],
                      plugins: ['@babel/plugin-proposal-object-rest-spread']
                  }
              }
          }
      ]
  }
};
```

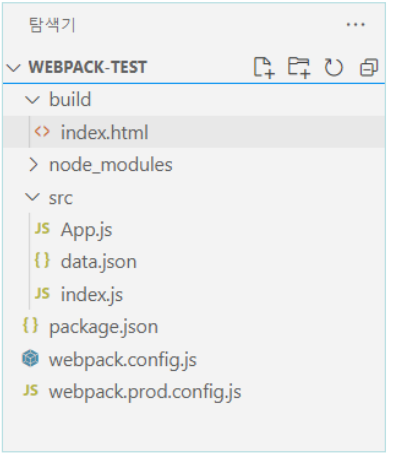
18

■ 주요 속성

- test : 정규식 패턴을 이용해 로드할 파일명 패턴을 지정. 주로 확장자로 식별함
- exclude : 로더를 적용하지 않을 패턴을 등록
- use
  - loader : 적용할 loader 지정
  - options 로더가 사용할 옵션

5. loaders(7)

- 기존 코드를 react,babel-loader 기반으로 새롭게 작성



- entry를 index.js로 변경
  - webpack.config.js, webpack.prod.config.js 모두 변경

```
module.exports = {
    ......
    entry: __dirname + '/src/index.js',
    ......
};
```

5. loaders(8)

- src/App.js 를 새롭게 작성

```
import React from 'react';
import data from './data.json';

const App = () => {
    var emplist = data.employees.map((item, index) => {
        return (
            <tr key={item.no}>
                <td>{item.no}</td>
                <td>{item.name}</td>
                <td>{item.mobile}</td>
                <td>{item.email}</td>
            </tr>
        )
    })
```

```
    return (
        <div>
            <h1>{data.title}</h1><hr/>
            <table>
                <thead>
                    <tr>
                        <th>번호</th><th>이름</th>
                        <th>모바일</th><th>이메일</th>
                    </tr>
                </thead>
                <tbody>
                    {emplist}
                </tbody>
            </table>
        </div>
    );
};

export default App;
```


■ class, import와 같은 ES6 문법을 사용할 수 있다.

■ HTML 마크업과 유사한 JSX 문법을 사용할 수 있다.

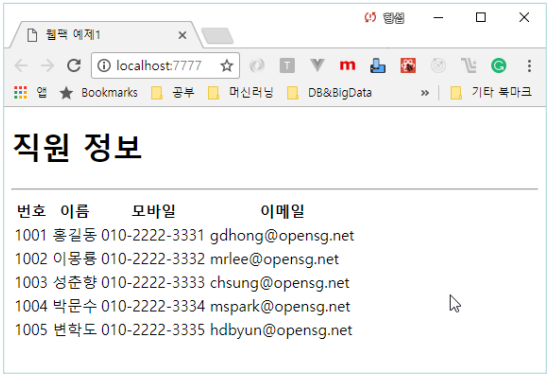
5. loaders(9)

- src/main.js 추가

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

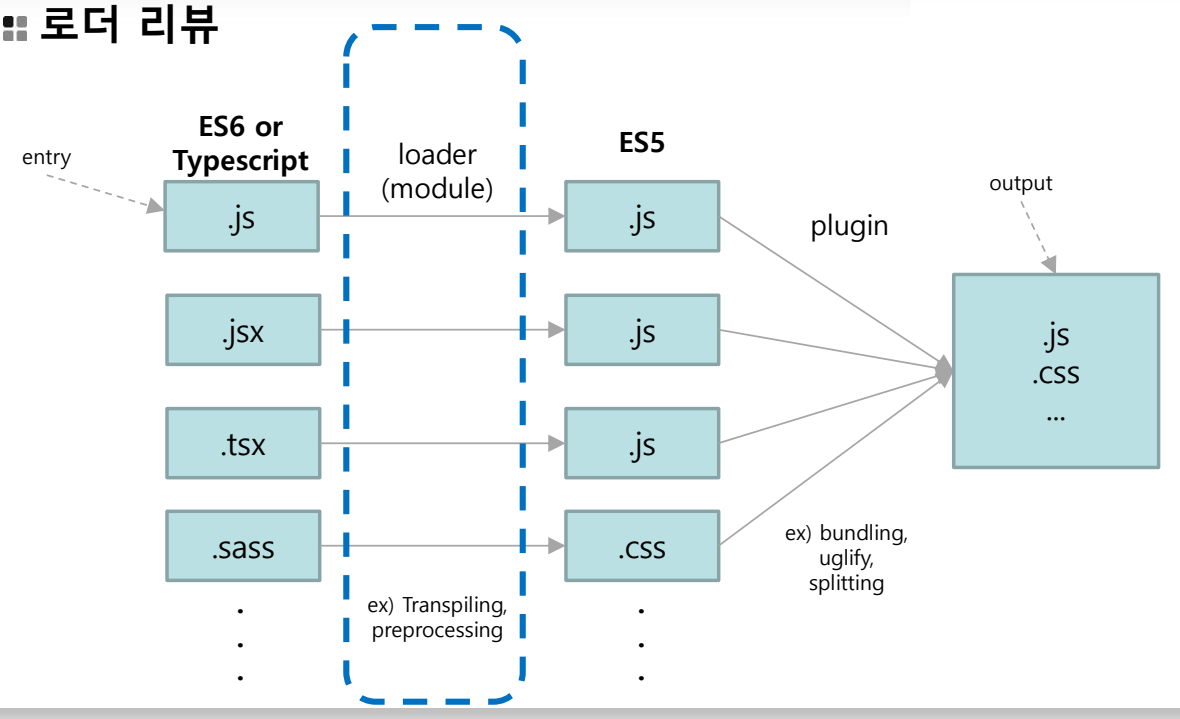
- npm run start로 실행



21

5. loaders(10)

로더 리뷰



5. loaders(11)

정적 자원 처리를 위한 로더

- webpack은 모든 파일을 모듈로 취급할 수 있으며 로더를 통해 전처리할 수 있음

css loader, style loader

- stylesheet를 전처리하는 로더
- web component를 만들때 style 정보도 포함되기 때문에...
- 로더 설치

```
npm install --save-dev style-loader css-loader
```

- webpack.config.js, webpack.prod.config.js 변경
  - 아래 텍스트 참조
  - 아래 설정은 전역 참조

■ webpack.config.js, webpack.prod.config.js 변경 내용

```
{
  module: {
    rules: [
      ......
      {
        test: /₩.css$/,
        use: [
          { loader: "style-loader" },
          { loader: "css-loader" }
        ]
      }
    ]
  }
}
```

5. loaders(12)

webpack 설정 파일 변경

- webpack.config.js, webpack.prod.config.js 모두 변경

```
{
  module: {
    rules: [
      ……
      {
        test: /₩.css$/,
        use: [
          { loader: "style-loader" },
          { loader: "css-loader" }
        ]
      },
    ]
  }
}
```

5. loaders(13)

- src/style.css 작성

```
table.list  { width: 600px; border:1px solid black; border-collapse:collapse; }
table.list td, table.list th { border:1px solid black;  text-align:center; }
table.list > thead > tr { color:yellow; background-color: purple; }
```

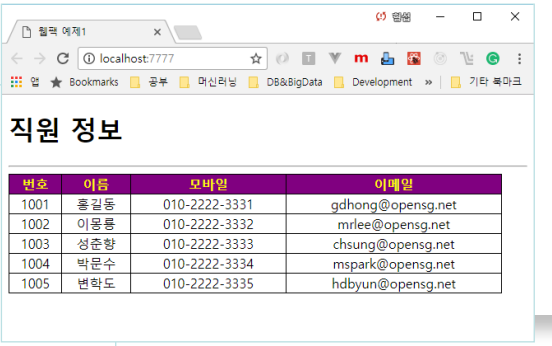
- src/index.js 변경

```
import React from 'react';
import ReactDOM from 'react-dom';
import './style.css';
import App from './App';

ReactDOM.render(<App />, document.getElementById('root'));
```

- src/App.js 변경
  - table 요소에 className 부여

```
render() {
      ......
      return (
         <div>
            <h1>{data.title}</h1><hr/>
            <table className={'list'}>
               ......
            </table>
         </div>
      );
   }
```



5. loaders(14)

css 모듈화

- 모듈화 : 코드를 명시적으로 선언된 독립적인 단위로 분할하는 작업
  - 자바스크립트 코드는 모듈화가 가능해져 왔지만 스타일시트는 대부분 전역에서 선언되고 작성되어 모듈화가 쉽지 않았음.
  - css module은 css 클래스명, 애니메이션명을 모두 로컬에서의 명칭으로 변경하여 독립적인 모듈화가 가능하도록 함.
  - 여러 컴포넌트에서 같은 이름의 클래스명을 사용했어도 로컬라이즈함.
- webpack.config.js, webpack.prod.config.js를 다음과 같이 변경

```
module.exports = {
    ......
    module: {
        rules: [
            ......
            {
                test: /₩.css$/,
                exclude : /₩.module₩.css$/,
                use: [
                  { loader: "style-loader" },
                  { loader: "css-loader" }
                ]
            },
            {
                test: /₩.module₩.css$/,
                use: [
                  { loader: "style-loader" },
                  { loader: "css-loader", options : { modules: true } }
                ]
            }
        ]
    },
    ......
};
```


■ 파일의 확장자가 .module.css로 끝나는 경우

- css 모듈 기능 적용

■ 파일의 확장자가 .css로 끝나지만 .module.css로 끝나지 않는 경우

- css 모듈 기능 적용하지 않음.

5. loaders(15)

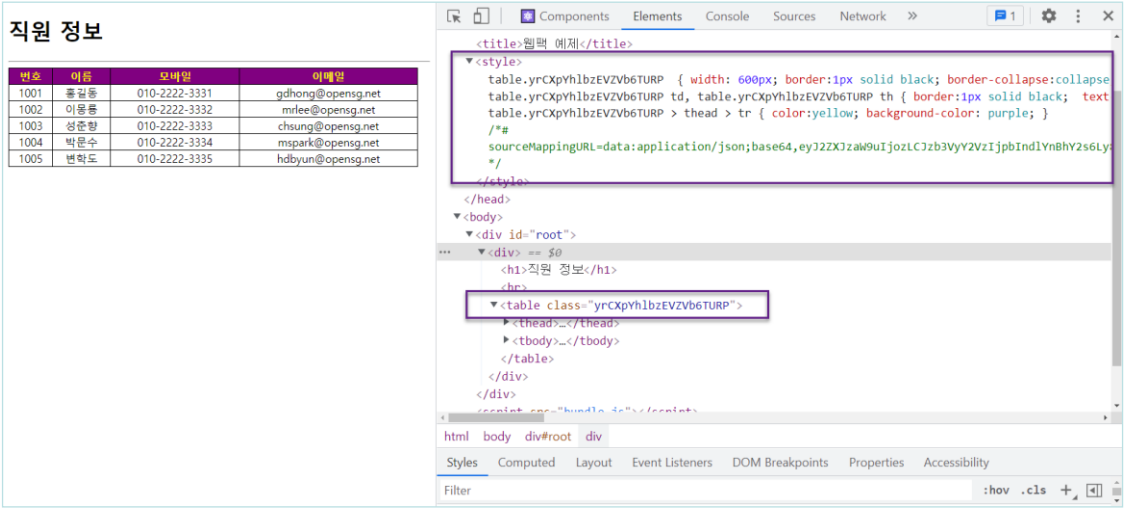
- src/index.js 에서...
  - import './style.css' 코드 삭제
- src/style.css를 style.module.css 파일명 변경
- src/App.js 코드 변경
  - 컴포넌트 단위로 스타일을 적용함.

```
import React, { Component } from 'react';
import data from './data.json';
import style from './style.module.css';

const App = () => {
  ......
  return (
    <div>
      <h1>{data.title}</h1><hr/>
      <table className={style.list}>
        ......
      </table>
    </div>
  );
}
export default App;
```

5. loaders(16)

실행 결과



28

6. Plugin(1)

plugin이란?

- webpack에서 사용가능한 추가기능 제공
- 빌드프로세스 과정에 플러그인을 주입시켜 Custom 동작이 가능하게 함
- loader VS plugin
  - loader는 리소스 파일(js, css, image, html등)을 로딩할 때 동작
  - plugin은 빌드 프로세스 과정에서 동작
- plugin 목록
  - https://webpack.js.org/plugins/
    - BannerPlugin
    - I18nWebpackPlugin
    - HtmlWebpackPlugin
    - mini-css-extract-plugin

29

6. Plugin(2)

html webpack plugin

- html 파일 생성 기능 제공

```
npm install -D html-webpack-plugin
```

- webpack.config.js, webpack.prod.config.js 수정

```
var HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {
    ......
    plugins : [
        new HtmlWebpackPlugin({
            title : '직원 정보 조회',
            template : __dirname + '/public/index.html',
            filename : 'index.html'
        }),
        ......
    ],
    ......
};
```

- build 디렉토리 내의 파일을 삭제함.

30

■ public/index.html 파일을 로드하여 가공하여 만든 index.html을 output에 저장함

6. Plugin(3)

webpack 설정 파일의 entry, output 변경

- entry 파일에 name 부여
- output filename을 entry의 name과 hash 값을 이용해 파일 생성

```
.......
module.exports = {
    ......
    entry: {
        index: __dirname + '/src/index.js'
    },
    output: {
        path: __dirname + '/build',
        filename: '[name]-[contenthash].js'
    },
    ......
};
```


31

6. Plugin(4)

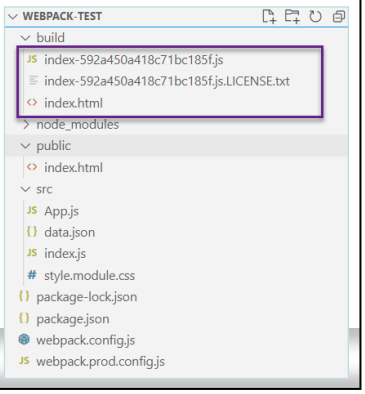
html webpack plugin(이어서)

- public/index.html 작성
  - 템플릿 페이지. <%= %> 문법 사용

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title><%=htmlWebpackPlugin.options.title %></title>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

- npm run build 후 확인

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4     <meta charset="UTF-8">
5     <title>직원 정보 조회</title>
6  <script defer src="index-7732d4bca6dad918fa37.js"></script></head>
7  <body>
8     <div id="root"></div>
9  </body>
10 </html>
11
```



■ 번들링할 때 해시값을 이용해 파일명을 생성했고 이 파일명으로 <script /> 태그를 동적으로 만들어서 추가해줌

- [name]-[contenthash].js로 번들링된 파일명 설정
- public/index.html 을 로딩하고 script 태그 추가한 후 output 경로(build)에 저장
  - 이 기능을 html-webpack-plugin이 지원

6. Plugin(5)

Production Build

- 개발용 빌드와의 차이점
  - HMR(Hot Module Replacement) 기능 등을 사용하지 않음.
  - webpack devtool, webpack dev server 사용하지 않음
  - Production용 빌드에는 최적화, 난독화, 캐싱, CSS JS 파일 분리 등의 기능이 적용됨.
    - webpack.prod.config.js 와 같이 별도의 설정 파일
- 제공할 기능
  - 코드 난독화 : webpack 설정 파일의 mode 값을 production으로 지정함.
  - 코드 스플릿팅 : webpack chunk로 해결(import() 활용)
  - mini-css-extract-plugin : 각 컴포넌트의 css 파일들을 모아서 번들링
- webpack.prod.config.js
  - devtool,devServer 등의 옵션 필요 없음.
  - 난독화 기능을 이용해 빌드 파일의 사이즈를 줄임.

33

6. Plugin(6)

mini-css-extract-plugin

- 모든 CSS에 대한 require, impor를 별도의 css 출력 파일로 옮겨서 JS에서 스타일을 인라인으로 추가할 필요가 없도록 해줌
  - npm install -D mini-css-extract-plugin
  - CSS 텍스트 파일을 묶어서 번들링한 뒤 하나 또는 몇 개의 css 파일을 생성함.
  - mini-css-extract-plugin을 사용하도록 webpack.prod.config.js 변경

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
var MiniCssExtractPlugin = require('mini-css-extract-plugin');

module.exports = {
    ......
    module: {
        rules: [
            ......
            {
                test: /₩.css$/,
                exclude : /₩.module₩.css$/,
                use: [
                    { loader: MiniCssExtractPlugin.loader  },
                    { loader: "css-loader" }
                ]
            },
```

6. Plugin(7)

webpack.prod.config.js 에 수정

```
(이어서)
        {
            test: /₩.module₩.css$/,
            use: [
               { loader: MiniCssExtractPlugin.loader },
               { loader: "css-loader", options : { modules: true } }
            ]
        }
    ]
  },
  plugins : [
    .......
    new MiniCssExtractPlugin({
        filename: '[name]-[contenthash].css'
    })
  ]
};
```


35

6. Plugin(8)

- src/style2.css

```
.mystyle {
   font-size:20pt;
   background-color:aqua;
   border:solid 1px gray;
}
```

- src/App.js

```
……
import style from './style.module.css'
import './style2.css';

class App extends Component {
    render() {
        ……
        return (
            <div>
                ……
                <div className="mystyle">contact : 010-222-3331</div>
            </div>
        );
    }
}

export default App;
```

36

■ style.css와 style2.css를 import하여 CSS 렌더링함.

■ 번들링한 결과는 하나의 css 파일

6. Plugin(9)

rimraf 를 이용한 기존 빌드 파일 삭제

- 파일 디렉토리 삭제 기능 제공
- npm install -D rimraf
- package.json 변경

```
{
  "name": "webpacktest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
 "scripts": {
    "dev": "webpack",
    "clean": "rimraf build",
    "build": "npm run clean && webpack --config webpack.prod.config.js",
    "start": "webpack-dev-server"
  },
  ……
}
```

37

■ clean 태스크 러너를 추가하고 build 태스크 러너에서 output 디렉토리(build)의 아티팩트를 삭제하고 다시 빌드하도록 함.

6. Plugin(10)

- npm run build 실행 후 번들링된 결과.
  - style2.css와 style.module.css의 내용이 하나로 번들링되었음

webpacktest8 > public > # main-85b091aa2bd322cce147.css > ...

```
1  table._3NlIeP4YRJBGqMemAF-qWI  { width: 600px; border:1px solid black; border-collapse:collapse; }
2  table._3NlIeP4YRJBGqMemAF-qWI td, table._3NlIeP4YRJBGqMemAF-qWI th { border:1px solid black;  text-align:center; }
3  table._3NlIeP4YRJBGqMemAF-qWI > thead > tr { color:yellow; background-color:  purple; }
4
5  .mystyle {
6      font-size:20pt;
7      background-color:aqua;
8      border:solid 1px gray;
9  }
10
```

정리

- webpack 기반 프로젝트의 기본 구조를 이해
- 훨씬 더 많은 로더나 플러그인
  - 모든 것을 알고 있을 필요는 없음. 전체적인 구조를 이해!!
  - https://webpack.js.org/loaders
  - https://webpack.js.org/plugins/

■ 이

7. create-react-app(1)

- **React 개발을 위한 대부분의 기본 설정을 포함하고 있음**
  - react-scripts에 대부분의 설정 포함
  - 구체적인 설정을 보려면 npm run eject 실행 후 파일, 디렉토리 구조 확인
  - entry : src/index.js
  - output : build/*
- **create-react-app 설치시 자동으로 node_modules을 내려받음**
  - yarn packager 사용을 권장함.
  - npm 보다 빠른 속도를 제공함.
- **webpack 기반**

39

■ **npm과 yarn 비교**

▪ **npm init**	**yarn init**
▪ **npm install**	**yarn**
▪ **npm install --save react**	**yarn add react**
▪ **npm uninstall --save react**	**yarn remove react**
▪ **npm install --save-dev cross-env**	**yarn add --dev cross-env**
▪ **npm update --save**	**yarn upgrade**

7. create-react-app(2)

npm run eject 후의 변화

- package.json
  - 많은 수의 dependencies
  - 태스크 러너(script)

```
"start": "node scripts/start.js",
"build": "node scripts/build.js",
"test": "node scripts/test.js"
```

- scripts 폴더
  - config 폴더의 설정들을 조합하여 각 환경에 맞게 실행해주는 스크립트들
- config 폴더 : 대부분의 중요한 설정
  - 다음 두가지를 주로 변경하게 됨.
    - config/webpack.config.js
    - config/webpackDevServer.config.js
  - 프로젝트 내부의 디렉토리를 변경하고 싶다면?
    - config/path.js 변경
    - 예) output 경로, Entry 파일 경로 등

■ 처음부터 직접 webpack 기반의 프로젝트를 구성할 수도 있으나 CRA 기반의 프로젝트를 eject 한 후 조금씩 수정,변경하여 사용하는 것이 바람직하다.

7. create-react-app(3)

npm과 yarn 비교

npm init	yarn init
npm install	yarn
npm install react	yarn add react
npm uninstall react	yarn remove react
npm install -D cross-env	yarn add -D cross-env
npm update --save	yarn upgrade

yarn을 권장함.

41