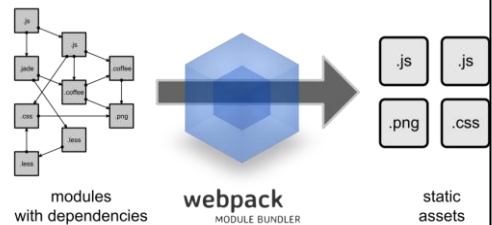


1. webpack이란?(1)



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

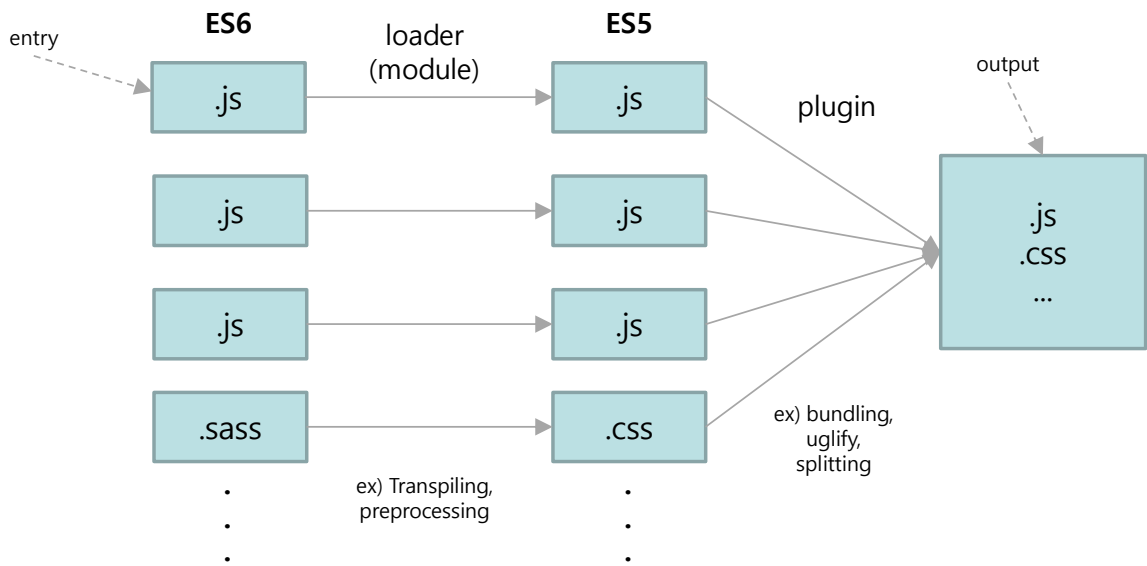


- webpack의 모든 구성요소를 상세히 알아보는 것은 이 장의 목적이 아니다. 전반적인 webpack 기반의 웹앱 프로젝트 구조를 이해하고 필요하다면 설정을 추가하거나 변경할 수 있도록 하는 것이 목적이다.

1. webpack이란?(2)



webpack과 react



2. webpack의 설치



▣ 전역 설치

- `npm install webpack webpack-cli -g`

▣ 프로젝트에 개발 버전으로 설치

- `npm init ==>` 프로젝트 생성
- `npm install --save-dev webpack webpack-cli`

▣ 개발 서버 설치

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

3. webpack 예제 1(1)



❏ 프로젝트 디렉토리 생성

- `mkdir webpacktest`
- `cd webpacktest`
- `npm init`

❏ webpack 패키지 설치

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

❏ 로컬 테스트 서버 설치

- `npm install -g live-server`

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('app').innerHTML = str;
```

3. webpack 예제 1(3)



▪ public/index.html

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

```
└─ WEBPACKTEST
   └─ node_modules
      └─ public
         └─ index.html
            └─ src
               ├── app.js
               ├── employees.js
               └─ package.json
```

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 + '/public',  
    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": "webpacktest",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {  
    "test": "echo W\"Error: no test specifiedW\" && exit 1",  
    "build": "webpack --config webpack.prod.config.js"  
  },  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "webpack": "^4.41.6",  
    "webpack-cli": "^3.3.11",  
  }  
}
```

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

3. webpack 예제 1(6)



■ 이제까지의 작성 결과 확인

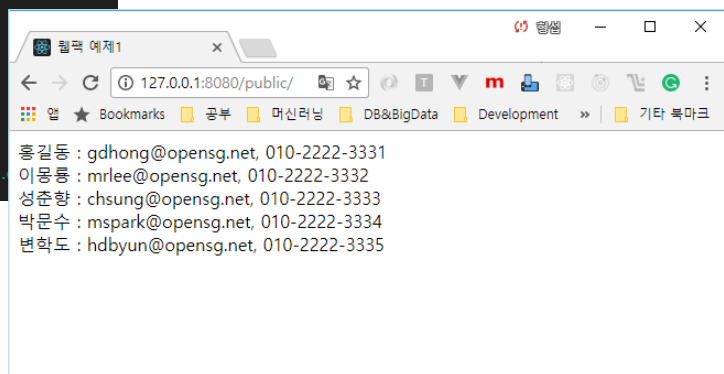
- build 태스크 러너 실행 후 브라우저로 확인
 - npm run build
 - live-server

```
PS D:\workspace\react2\ch02\webpacktest1> npm run build

> webpacktest@1.0.0 build D:\workspace\react2\ch02\webpacktest1
> webpack

Hash: 057c488d6c8bd38601d0
Version: webpack 3.8.1
Time: 62ms

   Asset      Size  Chunks             Chunk Names
bundle.js    3.28 kB       0  [emitted]  main
bundle.js.map 3.62 kB       0  [emitted]  main
    [0] ./src/app.js 277 bytes {0} [built]
    [1] ./src/employees.js 421 bytes {0} [built]
PS D:\workspace\react2\ch02\webpacktest1> live-server
Serving "D:\workspace\react2\ch02\webpacktest1" at http://127.0.0.1:8080/
GET /favicon.ico 404 7.701 ms - 24
```



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 webpack-dev-server --save-dev`
- 전역 설치 : `npm install webpack-dev-server --g`
- 설치후 webpack config 파일에 devServer 옵션 추가

```
module.exports = {  
  .....  
  devServer: {  
    contentBase: './public',  
    inline: true,  
    historyApiFallback: true,  
    port: 7777  
  }  
};
```

10

■ webpack config 파일에 devserver 옵션을 추가해야 한다.

- contentBase : 프로젝트 루트가 아니라 다른 경로를 contentBase로 지정할 때
- port : 기본값 8080
- inline : true로 설정하면 클라이언트 엔트리를 번들에 삽입해 페이지가 변경되면 새로 고침
- historyApiFallback : HTML5 history API를 이용하는 SPA를 개발할 때 유용. 이 값이 true이면 매핑되지 않은 개발 서버에 대한 요청시 /index.html로 라우팅됨.

■ 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": "webpacktest",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "build": "webpack --config webpack.prod.config.js",
    "start": "webpack-dev-server --hot --open"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "webpack": "^4.41.6",
    "webpack-cli": "^3.3.11",
    "webpack-dev-server": "^3.10.3"
  }
}
```

11

■ webpack-dev-server의 옵션

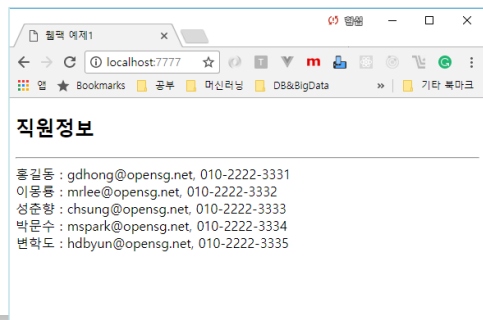
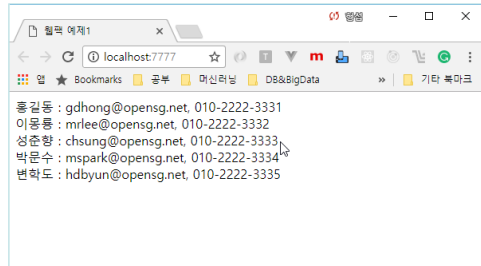
- --hot : HMR(Hot Module Replacement) 지원. 코드 수정 후 저장하면 즉시 화면에 반영됨.
- --open : webpack 개발 서버 구동후 브라우저 자동 열림

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('app').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

5. loaders(2)



■ json-loader

- json 파일을 읽어와 JS 객체로 사용할 수 있도록 함.
- npm install --save-dev 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

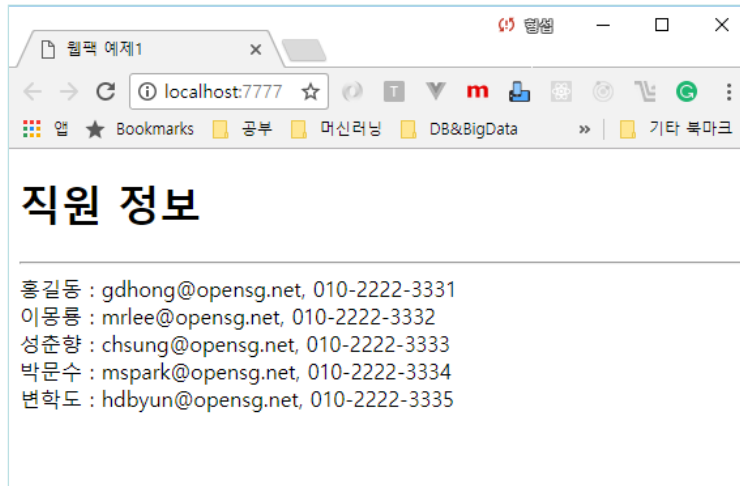
```
var data = require("./data.json");

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;
```

5. loaders(4)



▪ json-loader 테스트



5. loaders(5)



■ babel-loader

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

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

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']
          }
        }
      }
    ]
  }
};
```

5. loaders(7)



- 기존 코드를 react,babel-loader 기반으로 새롭게 작성
- entry를 main.js로 변경
 - webpack.config.js, webpack.prod.config.js 모두 변경

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

- app.js 파일명을 App.js로 변경함.

```
└─ WEBPACKTEST3  
  └─ node_modules  
    └─ public  
      └─ src  
        App.js  
        data.json  
        main.js  
        package.json  
        webpack.config.js
```

5. loaders(8)



■ src/App.js 를 새롭게 작성

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

class App extends Component {
  render() {
    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)

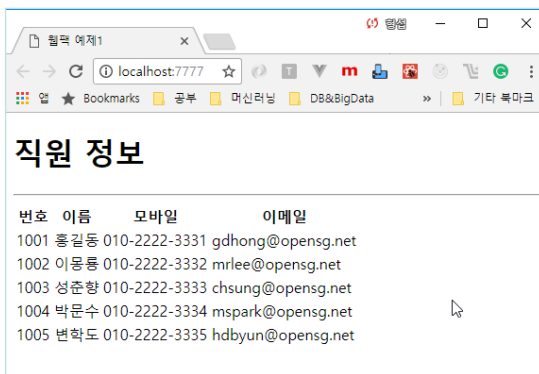


■ src/main.js 추가

```
import React from 'react';
import {render} from 'react-dom';
import App from './App';

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

■ npm run start로 실행



5. loaders(10)



▣ 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

22

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

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

}

5. loaders(11)

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/main.js 변경

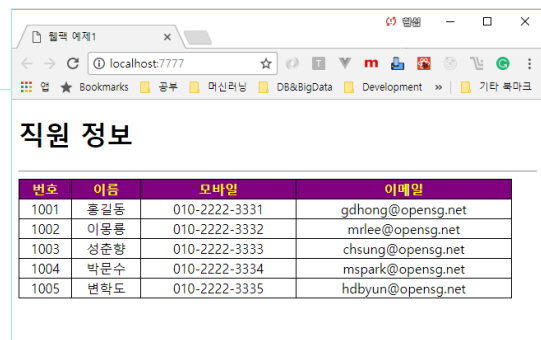
```
import React from 'react';
import {render} from 'react-dom';
import './style.css';
import App from './App';

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

src/App.js 변경

- table 요소에 className 부여

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



5. loaders(12)



■ 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 } }
  ]
},
.....
};
```

24

■ 파일의 확장자가 .module.css로 끝나는 경우

- css 모듈 기능 적용

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

- css 모듈 기능 적용하지 않음.

5. loaders(13)



- src/main.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';

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

25

5. loaders(14)

실행 결과

The screenshot shows a web browser window with the address bar at `localhost:7777`. The page title is "직원 정보" (Employee Information). The page content displays a table with employee data. The browser's developer tools are open, showing the `Elements` panel with the following HTML structure:

```
<title>예제1</title>
<style type="text/css">
  table._3ahFrUuYRFs3M_wu0OcCr7 { width: 600px; border:1px solid black; border-collapse:collapse; }
  table._3ahFrUuYRFs3M_wu0OcCr7 td, table._3ahFrUuYRFs3M_wu0OcCr7 th { border:1px solid black; text-align:center; }
  table._3ahFrUuYRFs3M_wu0OcCr7 > thead > tr { color:yellow; background-color:purple; }
</style>
</head>
<body data-gr-c-s-loaded="true">
  <div id="app">
    <div>
      <h1>직원 정보</h1>
      <hr>
      <table class="_3ahFrUuYRFs3M_wu0OcCr7">
        <thead>...</thead>
      </table>
    </div>
  </div>
</body>
```

The `Styles` panel shows the default styles for the selected table element:

```
element.style {
  margin: -
  border: -
}
```

번호	이름	모바일
1001	홍길동	010-2222-3331
1002	이몽룡	010-2222-3332
1003	성준황	010-2222-3333
1004	박문수	010-2222-3334
1005	변학도	010-2222-3335

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

6. Plugin(2)



■ html webpack plugin

- html 파일 생성 기능 제공

```
npm install --save-dev 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 + '/assets/index.html',
      filename: 'index.html'
    }),
    .....
  ],
  .....
};
```

- public 디렉토리 내의 파일을 삭제함.

28

- assets/index.html 파일을 로드하여 가공하여 만든 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: {  
    main: __dirname + '/src/main.js'  
  },  
  output: {  
    path: __dirname + '/public',  
    filename: '[name]-[hash].js'  
  },  
  .....  
};
```

6. Plugin(4)



■ html webpack plugin(이어서)

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

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

- npm run build 후 확인

```
webpacktest7 > public > <> index.html > ...
1  <!DOCTYPE html>
2  <html>
3  <head>
4    <meta charset="UTF-8">
5    <title>직원 정보 조회</title>
6  </head>
7  <body>
8    <div id="app"></div>
9  <script type="text/javascript" src="main-7cc80e025ca2f3f6c08a.js"></script></body>
10 </html>
11
```

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 스플리팅 : webpack chunk로 해결
- 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.

31

■ package.json에 새로운 태스크 러너 추가

```
{  
  .....  
  "scripts": {  
    "build": "cross-env NODE_ENV=production webpack --config  
    webpack.prod.config.js",  
    "start": "cross-env NODE_ENV=development webpack-dev-server --hot --open"  
  },  
  .....  
}
```


6. Plugin(6)



■ mini-css-extract-plugin

- 모든 CSS에 대한 require, import를 별도의 css 출력 파일로 옮겨서 JS에서 스타일을 인라인으로 추가할 필요가 없도록 해줌
 - CSS 텍스트 파일을 묶어서 번들링한 뒤 하나 또는 여러개의 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: /W.moduleW.css$/,
      use: [
        { loader: MiniCssExtractPlugin.loader },
        { loader: "css-loader", options : { modules: true } }
      ]
    }
  ],
  plugins : [
    .....
    new MiniCssExtractPlugin({
      filename: '[name]-[hash].css'
    })
  ]
};
```

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;
```

34

- 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 --save-dev rimraf`
- `package.json` 변경

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

6. Plugin(10)



- npm run build 실행 후 번들링된 결과.

```
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
```

7. create-react-app(1)



❑ React 개발을 위한 대부분의 기본 설정을 포함하고 있음

❑ 설치

- `npm install -g create-react-app`
- `react-scripts`에 대부분의 설정 포함
- 구체적인 설정을 보려면 `npm run eject` 실행후 파일, 디렉토리 구조 확인
- entry : `src/index.js`
- output : `build/*`

❑ `create-react-app` 설치시 자동으로 `node_modules`을 내려받음

- yarn packager 사용
- npm 보다 빠른 속도를 제공함.

37

■ <https://infinite.red/files/yarn.pdf>

- | | |
|---|---------------------------------------|
| ▪ <code>npm init</code> | <code>yarn init</code> |
| ▪ <code>npm install</code> | <code>yarn</code> |
| ▪ <code>npm install --save react</code> | <code>yarn add react</code> |
| ▪ <code>npm uninstall --save react</code> | <code>yarn remove react</code> |
| ▪ <code>npm install --save-dev cross-env</code> | <code>yarn add --dev cross-env</code> |
| ▪ <code>npm update --save</code> | <code>yarn upgrade</code> |

7. create-react-app(2)



▣ 수동 설정을 원할 경우 npm run eject 실행 후 상세 설정

- 다음 파일을 수정하여 변경 가능
 - config/webpack.config.js
 - config/webpackDevServer.config.js