

1. webpack이란?



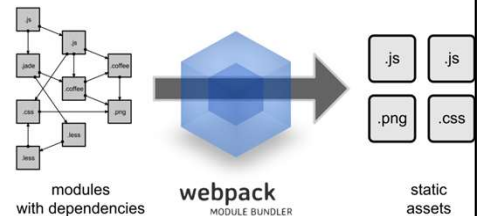
■ webpack

■ 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



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 도구 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 = [  
  { name : '홍길동', email:'gdhong@opensg.net', mobile:'010-2222-3331' },  
  { name : '이몽룡', email:'mrlee@opensg.net', mobile:'010-2222-3332' },  
  { name : '성춘향', email:'chsung@opensg.net', mobile:'010-2222-3333' },  
  { name : '박문수', email:'mspark@opensg.net', mobile:'010-2222-3334' },  
  { name : '변학도', email:'hbyun@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 1</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
 - 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 사용한 webpack 설정 파일인 경우
 - webpack myconfig.js

6

- __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"  
  },  
  "author": "",  
  "license": "ISC",  
  "devDependencies": {  
    "webpack": "^4.27.1",  
    "webpack-cli": "^3.1.2",  
  }  
}
```

3. webpack 예제 1(6)



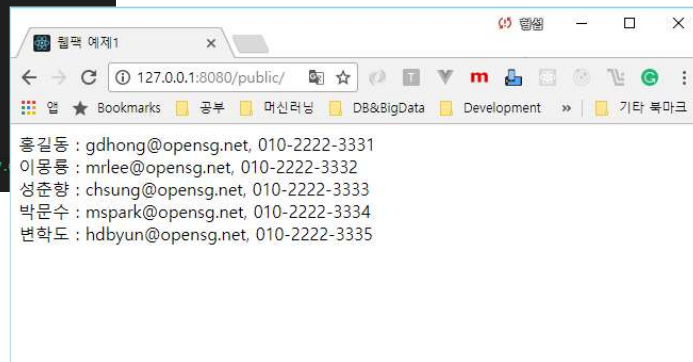
■ 이제까지의 작성 결과 확인

- 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  
  }  
};
```

9

■ 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.

4. webpack 개발 서버(2)



■ 개발 환경과 운영환경 분리를 위한 설정

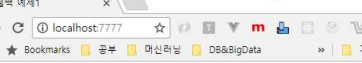
- `npm install --save-dev cross-env`
 - 개발버전의 빌드로 구동하기 위해 `NODE_ENV` 환경 변수를 `development`로 설정하고 구동한다.
 - 윈도우와 리눅스, MacOS는 환경변수를 설정하는 방법이 다르기 때문에 `cross-env`를 사용할 것을 권장한다. 다음 패키지를 먼저 설치하도록 하자.
- `package.json`에 start 태스크 러너 추가

```
{
  .....
  "scripts": {
    "build": "cross-env NODE_ENV=production webpack",
    "start": "cross-env NODE_ENV=development webpack-dev-server --hot --open"
  },
  "author": "",
  "license": "ISC",
  "devDependencies": {
    "cross-env": "^5.2.0",
    "webpack": "^4.27.1",
    "webpack-cli": "^3.1.2",
    "webpack-dev-server": "^3.1.10"
  }
}
```

■ 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

- `npm run start`

- 
- The screenshot shows a web browser window with the address bar displaying 'localhost/7777'. The browser's address bar and tabs are visible at the top. The main content area displays a list of IP addresses and domain names, each on a new line. The list includes: '홍길동: gdhong@opengs.net, 010-2222-3331', '이몽두: mrlee@opengs.net, 010-2222-3332', '성준황: chsung@opengs.net, 010-2222-3333', '박민수: mspark@opengs.net, 010-2222-3334', and '변학도: hdbyun@opengs.net, 010-2222-3335'. A mouse cursor is pointing at the third line, '성준황: chsung@opengs.net, 010-2222-3333'.

직원 정보

홍길동 : gdhong@opengsg.net, 010-2222-3331
 이몽룡 : mrllee@opengsg.net, 010-2222-3332
 성준형 : chsung@opengsg.net, 010-2222-3333
 박문수 : mspark@opengsg.net, 010-2222-3334
 변학두 : hdbyun@opengsg.net, 010-2222-3335

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
 - uglify
 - 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.config.js`에 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": [
    { "name": "홍길동", "email": "gdhong@opensg.net", "mobile": "010-2222-3331" },
    { "name": "이몽룡", "email": "mrlee@opensg.net", "mobile": "010-2222-3332" },
    .....
  ]
}
```

▪ src/app.js 변경

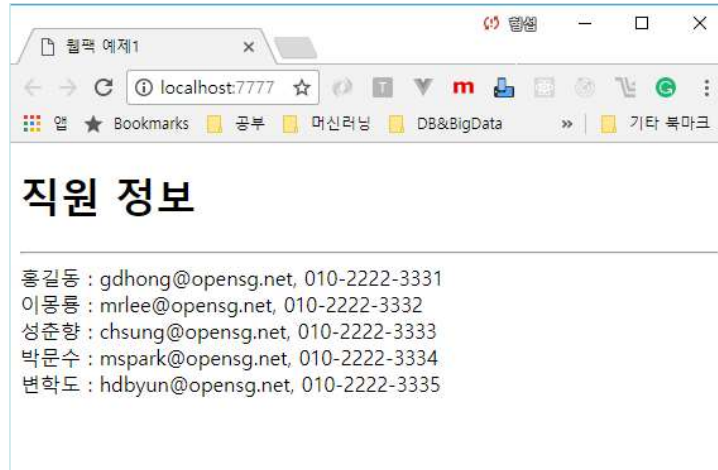
```
var data = require("./data.json");

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)

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

- 기존 코드를 react, babel-loader 기반으로 새롭게 작성

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

5. loaders(7)



- webpack.config.js의 entry를 main.js로 변경

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

- src/data.json 수정 : no 필드 추가

```
{  
  "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" },  
    { "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" }  
  ]  
}
```

5. loaders(8)



■ src/App.js 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)

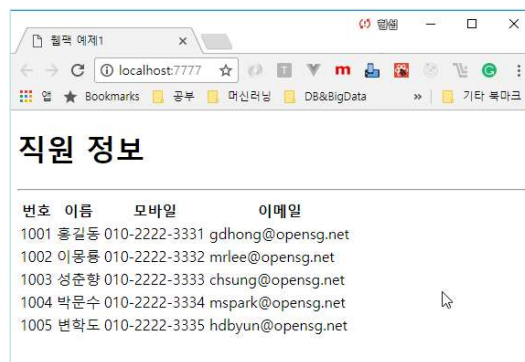


▪ src/main.js 작성

```
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 변경
 - 아래 텍스트 참조
 - 아래 설정은 전역 참조

21

■ webpack.config.js 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; }
```

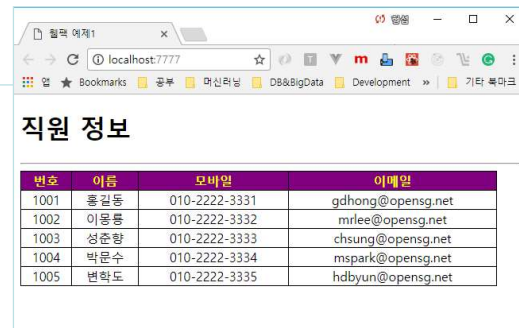
src/main.js 변경

```
import React from 'react';  
import {render} from 'react-dom';  
import App from './App';  
import './style.css';  
  
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.
 - `npm install --save-dev css-loader style-loader`

■ webpack.config.js 변경

```
{
  test: /\.css$/,
  use: [
    { loader: "style-loader" },
    {
      loader: "css-loader",
      options: {
        modules:true
      }
    }
  ]
}
```


5. loaders(13)



- src/main.js 에서...
 - import './style.css' 코드 삭제
- src/App.js 코드 변경
 - 컴포넌트 단위로 스타일을 적용함.

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

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

export default App;
```

5. loaders(14)



실행 결과

The screenshot shows a web browser window with the URL `localhost:7777`. The page title is "직원 정보" (Employee Information). The page content displays a table with the following data:

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

The browser's developer tools are open, showing the DOM structure. The `<table>` element is selected, and its CSS styles are displayed in the right-hand pane. The styles include a width of 600px, a solid black border, and a collapsed state. The table's structure is also visible in the DOM tree, showing the `<thead>` and `<tbody>` elements.

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
 - UglifyJsWebpackPlugin
 - CommonsChunkPlugin
 - I18nWebpackPlugin
 - HotModuleReplacementPlugin
 - HtmlWebpackPlugin

6. Plugin(2)



■ html webpack plugin

- html 파일 생성 기능 제공

```
npm install --save-dev html-webpack-plugin
```

- webpack.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 디렉토리의 파일을 삭제함.

27

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

6. Plugin(3)



■ 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>
```

```
...<!DOCTYPE html> == $0
<html class="gr__localhost">
  <head>
    <meta charset="UTF-8">
    <title>직원 정보 조회</title>
    <style type="text/css">...</style>
  </head>
  <body data-gr-c-s-loaded="true">
    <div id="app">
      <div>...</div>
    </div>
    <script type="text/javascript" src="bundle.js"></script>
  </body>
</html>
```

6. Plugin(4)



■ 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
- 설치 플러그인
 - UglifyJsWebpackPlugin : JS 코드 난독화 기능 제공
 - ExtractTextPlugin
 - CommonChunksPlugin
- webpack.prod.config.js
 - 기존 webpack.config.js 파일을 복사한 후 devtool, devServer 등의 옵션을 삭제함.

29

■ 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(5)



■ UglifyjsWebpackPlugin

- 코드를 난독화시키고 압축해주는 플러그인
 - npm install uglifyjs-webpack-plugin --save-dev
- webpack.prod.config.js에 내용 추가

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

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

30

- npm run build 태스크 러너를 실행하면 public/bundle.js 파일이 난독화, 압축된 것을 확인할 수 있음.
- production 빌드용으로 적합한 기능
- webpack 4.x에서는 webpack 설정 파일에서 mode 옵션을 "production" 지정하면 자동 수행됨.

6. Plugin(6)



- UglifyjsWebpackPlugin 적용후 빌드 결과
 - bundle.js 빌드 결과

```
!function(e){function t(r){if(n[r])return n[r].exports;var o=n[r]={i:r,l:!1,exports:{}};return e[r].call(o.exports,o,o.exports,t),o.l=!0,o.exports}var n={};t.m=e,t.c=n,t.d=function(e,n,r){t.o(e,n)||Object.defineProperty(e,n,{configurable:!1,enumerable:!0,get:r})},t.n=function(e){var n=e&&e.__esModule?function(){return e.default}:function(){return e};return t.d(n,"a",n),n},t.o=function(e,t){return Object.prototype.hasOwnProperty.call(e,t)},t.p="",t(t.s=14)}([function(e,t){function n(){throw new Error("setTimeout has not been defined")}function r(){throw new Error("clearTimeout has not been defined")}function o(e){if(s===setTimeout)return setTimeout(e,0);if((s===n||!s)&&setTimeout)return s=setTimeout,setTimeout(e,0);try{return s(e,0)}catch(t){try{return s.call(null,e,0)}catch(t){return s.call(this,e,0)}}}function a(h){h&&f&&(h=!1,f.length?p=f.concat(p):m=-1,p.length&&i())}function i(){if(!h){var e=o(a);h=!0;for(var t=p.length;t;){for(f=p,p=[];++m<t;)f&&f[m].run();m=-1,t=p.length}f=null,h=!1,function(e){if(c===clearTimeout)return clearTimeout(e);if((c===r||!c)&&clearTimeout)return c=clearTimeout,clearTimeout(e);try{c(e)}catch(t){try{return c.call(null,e)}catch(t){return c.call(this,e)}}}(e)}}}function l(e,t){this.fun=e,this.array=t}function u(){var s,c,d=e.exports={};!function(){try{s="function"==typeof setTimeout?setTimeout:n}catch(e){s=n}try{c="function"==typeof clearTimeout?clearTimeout:r}catch(e){c=r}}();var f,p=[],h=!1,m=-1;d.nextTick=function(e){var t=new Array(arguments.length-1);if(arguments.length>1)for(var n=1;
```


6. Plugin(6)



■ ExtractTextPlugin

- 모든 CSS에 대한 require, impor를 별도의 css 출력 파일로 옮겨서 JS에서 스타일을 인라인으로 추가할 필요가 없도록 해줌
 - CSS 텍스트 파일을 묶어서 번들링한 뒤 하나 또는 여러개의 css 파일을 생성함.
- `npm install --save-dev extract-text-webpack-plugin@next`
- `webpack.prod.config.js` 변경

```
var HtmlWebpackPlugin = require('html-webpack-plugin');
var UglifyJsPlugin = require('uglifyjs-webpack-plugin');
var ExtractTextPlugin = require("extract-text-webpack-plugin");

module.exports = {
  ....
  module: {
    rules: [
      ....
      {
        test: /\.css$/,
        use: ExtractTextPlugin.extract({
          fallback: "style-loader",
          use: [
            { loader: "css-loader", options : { modules:true } },
```

6. Plugin(7)



■ webpack.prod.config.js 에 수정

(이어서)

```
        {
          loader: 'postcss-loader',
          options: {
            plugins: (loader) => [
              require('autoprefixer')(),
              require('precss')()
            ]
          }
        }
      ]
    },
    plugins : [
      .....
      new UglifyJsPlugin(),
      new ExtractTextPlugin("[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.css';  
import style2 from './style2.css';  
  
class App extends Component {  
  render() {  
    .....  
    return (  
      <div>  
        .....  
        <div className={style2.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)



- npm run build 실행 후 번들링된 결과.

```
table._3ahFrUuYRFs3M_wu00cCr7 { width: 600px; border:1px solid black; border-collapse:collapse; }
table._3ahFrUuYRFs3M_wu00cCr7 td, table._3ahFrUuYRFs3M_wu00cCr7 th { border:1px solid black; text-align: center; }
table._3ahFrUuYRFs3M_wu00cCr7 > thead > tr { color: yellow; background-color: purple; }
._3Y9RoEYxJ9BNF-7hs9MXNm {
  background-color: aqua; border: solid 1px brown; width:578px;
  text-align: center; font-size:15pt; padding:10px 10px 10px 10px;
  font-weight:bold;
  display: -webkit-box;
  display: -ms-flexbox;
  display: flex;
}
._5GJx20GSbnoFIRv9fk-m3 {
  font-size:20pt;
  background-color: aqua;
  border:solid 1px gray;
}
```

6. Plugin(10)



Common Chunks Plugin

- 현재까지의 번들링 결과물은 단하나의 js 파일
 - 하지만 이 내부에는 변경될 일이 없는 react, react-dom 등의 라이브러리가 있음
- 변경되지 않는 부분을 분리해서 별도의 파일로 생성한다면...
 - 캐싱기능 활용
 - 로딩 속도 개선
- webpack.prod.config.js 변경
 - react, react-dom 을 vendor.js로 분리
 - main.js 모듈 사이즈를 비교할 것
 - npm run build 후 확인

index.html	2017-12-05 오전...	Chrome HTML D...	1KB
main-debad5650529f1a8eb34.css	2017-12-05 오전...	CSS 스타일시트 ...	1KB
main-debad5650529f1a8eb34.js	2017-12-05 오전...	JavaScript 파일	3KB
vendor-debad5650529f1a8eb34.js	2017-12-05 오전...	JavaScript 파일	274KB

```
.....
var webpack = require('webpack');

module.exports = {
  entry: {
    main: __dirname + '/src/main.js',
    vendor: [
      'react',
      'react-dom'
    ]
  },
  .....
  plugins: [
    .....
    new webpack.optimize.CommonsChunkPlugin({
      name: 'vendor'
    })
  ]
};
```

36

■ webpack 4.0부터는 SplitChunks Plugin 을 사용함. 자세한 내용은 아래 문서 참조

- <https://webpack.js.org/plugins/split-chunks-plugin/>
- <https://www.zerocho.com/category/Webpack/post/58ad4c9d1136440018ba44e7>

■ react-router를 적용할 경우

- 경로별로 모듈 파일을 분리시키고
- 코드 실행시에 비동기 방식으로 모듈 파일을 import 할 수 있음

■ CRA(Create-react-app) 2.x 에서는 이 설정이 기본 내장되어 있음.

7. create-react-app(1)



■ React 개발을 위한 대부분의 기본 설정을 포함하고 있음

■ 설치

- `npm install -g create-react-app`
- react-scripts에 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_modules을 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

- CSS Module 설정 등이 안되어 있음
- Split Chunks Plugin의 설정도 세밀한 설정이 필요할 수 있음.
- 다음 파일을 수정하여 변경 가능
 - config/webpack.config.dev.js
 - config/webpack.config.prod.js
 - config/webpackDevServer.config.js