번들러(Bundler)

번들러란

웹사이트를 만들기 위해선 HTML, CSS, JavaScript 이렇게 단 3가지의 재료만 있어도 가능하지만 이렇게 순수하게 3가지 재료만 가지고 웹사이트를 만드는 경우는 거의 없다.
HTML은 React나 Vue.js 등의 프레임워크로 대체되고, CSS는 Sass 같은 전처리기로 대체되곤 합니다. 그리고 JavaScript 대신 TypeScript를 사용하기도 하며 여기에 Firebase, Lodash 등 다양한 써드 파티 모듈(3rd party modules)까지 추가 사용된다.

대부분의 디펜던시(dependency)들이 CommonJS 방식을 사용하는데 ES6 모듈 문법과 맞지 않아 에러가 발생 되기도 하고 거대한 자바스크립트 파일 때문에 구동이 늦어지고, 구형 브라우저(legacy browser)에서도 코드가 돌아가도록 하기 위해 폴리필(polypill)도 추가하기도 한다.
모듈 번들러가 이러한 번거로운 과정을 도와 줄 수 있다. 모듈 번들러가 근본적으로 하는 일은 다수의 자바스크립트 파일과 그 밖의 써드 파티 모듈들을 가지고 브라우저에서 로드되는 하나의 커다란 자바스크립트 파일(즉 자바스크립트 번들)로 만드는 것입니다.

> 폴리필 : 브라우저에서 지원하지 않는 코드를 사용가능한 코드 조각이나 플러그인(추가기능)을 의미한다.

설치하기

1. 폴더 생성하기
2. 초기화하여 package.json 파일 생성하기

```
npm init -y
3. web pack 설치하기
npm i -D webpack webpack-cli webpack-dev-server@next
```

> webpack: 모듈(패키지) 번들러의 핵심 패키지
> webpack-cli: 터미널에서 Webpack 명령(CLI)을 사용할 수 있음
> webpack-dev-server: 개발용으로 Live Server를 실행(HMR)

`require() : Node.JS 에서는 require 메서드를 통해 외부 모듈을 가져올 수 있습니다. require 메서드는 node가 local object에 추가한 메서드로서 다음과 같이 파라미터로 추가할 모듈의 파일 경로값을 받습니다.`

webpack 설정하기

```
 "scripts": {
    "dev": "webpack-dev-server --mode development",
    "build": "webpack --mode production"
  },
```

기본 config 파일 설정하기

1. webpack.config.js생성하기
2. build 설정하기

```
const path = require('path')
module.exports = {
  entry : './js/main.js',
  output : {
    path: path.resolve(__dirname, 'public'),
    filename: ' main.js',
    clean: true
  },
```

HTML 파일 설정하기

1. install 하기

npm i -D html-webpack-plugin

2. config 구성하기

```
const HtmlPlugin = require('html-webpack-plugin')

plugins: [
    new HtmlPlugin({
      template : './index.html'
    })
],
```

html-webpack-plugin: 최초 실행될 HTML 파일(템플릿)을 연결

copy-webpack-plugin: 정적 파일(파비콘, 이미지 등)을 제품(dist) 폴더로 복사

서버 host 설정하기

```
  devServer: {
    host: 'localhost',
    port: 8080,
    hot: true
  }
```

외부파일 연결하기

1. install 하기

> npm i -D copy-webpack-plugin

2. 이미지 파일 준비하기 :
3. html 문서에 <img src=""> 추가하기
4. static 폴더만들기
5. static 폴더에 favicon 추가하기
6. static 폴더에 images 폴더 추가하고 이미지 파일 추가하
7. config 구성하기

```
const CopyPlugin = require('copy-webpack-plugin')

//plugins 에 추가하기

 [new CopyPlugin({
   patterns: [
     { from: 'static' }
   ]
 })
],
```

모듈 추가하기: css, scss

1. install 하기

> npm i -D css-loader style-loader
> 2. root 폴더에 css 폴더 추가하기
> 3. main.css 만들고 -> main.js에 import 하기

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

sass-loader: SCSS(Sass) 파일을 로드

postcss-loader: PostCSS(Autoprefixer)로 스타일 파일을 처리
```

```
css-loader: CSS 파일을 로드
style-loader: 로드된 스타일(CSS)을 <style>로 <head>에 삽입
```

babel 연결하기

1. install 하기

```
npm i -D @babel/core @babel/preset-env @babel/plugin-transform-runtime
```

```
npm i -D babel-loader
```

2. root 폴더에 .babelrc.js 생성하기

```
module.exports = {
  presets: ['@babel/preset-env'],
  plugins: [
    ['@babel/plugin-transform-runtime']
  ]
}
```

3. config 설정하기

```
{
  test: /\.js$/,
  exclude: /node_modules/, // 제외할 경로
  use: [
    'babel-loader'
  ]
}

babel-loader: JS 파일을 로드
@babel/core: ES6 이상의 코드를 ES5 이하 버전으로 변환
@babel/preset-env: Babel 지원 스펙을 지정
@babel/plugin-transform-runtime: Async/Await 문법 지원
```