

Bundling Web Application

with Webpack

Setting up node.js environment

Installing NVM on Mac

- <https://github.com/nvm-sh/nvm>

Installing NVM on Windows

- <https://github.com/coreybutler/nvm-windows>
 - Releases > nvm-setup.zip 다운로드 후 설치

PowerShell Security Setting

```
Set-ExecutionPolicy Unrestricted -Scope CurrentUser
```

Setting up node.js environment

Node.js 설치 on NVM

```
nvm list # 사용가능 node 버전 확인  
nvm list available # 설치 가능한 node 버전 확인  
nvm install 14.17.0 # 목록에 표시된 node 버전 설치  
nvm use 14.17.0 # 사용 설정
```

Node.js 설치 확인

```
node --version # 노드 설치 확인  
npm --version # 패키지 매니저 설치 확인
```

Using Node Package Manager(NPM)

> Node Package Initialize

```
npm init # 프로젝트 관련 정보 입력
```

> package.json 구성요소

- name, version
 - 이 두 정보로 패키지의 고유성을 식별
- main
 - 프로그램의 시작점이 되는 모듈의 ID

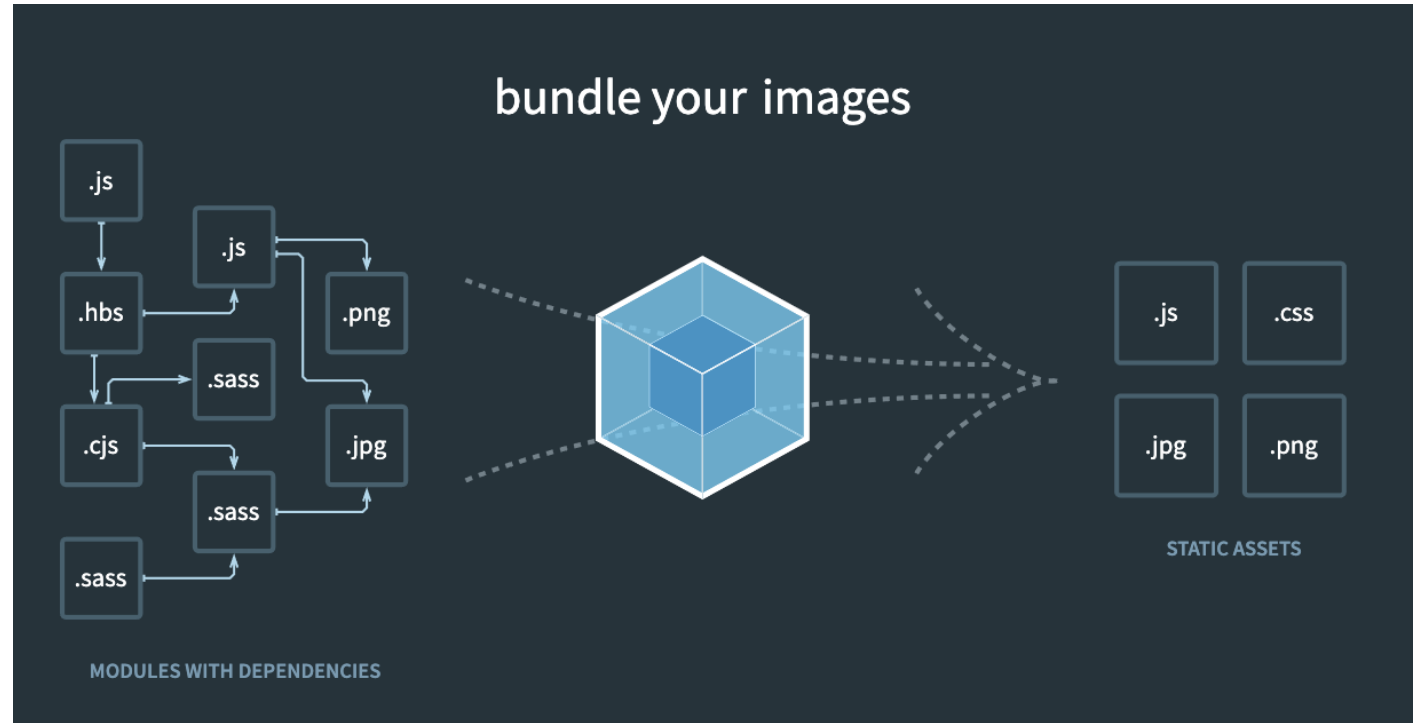
Using Node Package Manager(NPM)

> package.json 구성요소

- scripts
 - 패키지의 생명주기 중 다양한 타이밍에서 실행되는 script 명령들을 포함하고 있는 사전
- dependencies
 - 의존성 모듈 정보
- devDependencies
 - 테스트 관련 모듈, 개발 단계에서 필요한 모듈 의존성 정보

Webpack

- <https://webpack.js.org/>
- 최신 프론트엔드 프레임워크에서 가장 널리 사용되는 모듈 번들러
- 웹 어플리케이션을 구성하는 수많은 자원들을 하나의 파일로 병합 및 압축



Webpack

> 4대 주요 속성

- entry
 - webpack이 모듈의 의존 관계를 해석하는 시작점
- output
 - 번들된 내용을 출력할 파일 경로
- loader
 - 자바스크립트 이외의 정적 파일을 번들링하기 위한 변환 도구
 - 파일을 해석하고 변환하는 과정에 관여
- plugin
 - 웹팩의 기본 동작에 추가 기능을 제공하는 속성
 - 결과물의 형태를 바꾸는 역할을 수행

Webpack

> Project 생성 및 Webpack 설정

```
npm init -y  
npm install -D webpack webpack-cli
```

- 설치 후, package.json을 확인

> 사용법 확인

```
node_modules/.bin/webpack --help
```


Webpack

> Webpack 실행 주요 옵션

- `--mode` : 웹팩 실행 모드
- `--entry` : 번들링 시작점 경로 지정
- `--output` : 번들링 결과물 경로 지정
- `--config` : 웹팩 설정파일 경로 지정
 - default: `webpack.config.js`

Webpack

> 번들링 테스트

준비할 파일들

- src/math.js
- src/app.js
- src/index.html

Bundling

```
node_modules/.bin/webpack --mode development --entry ./src/app.js --output-path dist/main.js
```

- mode를 production으로 변경하여 번들링을 해 봅시다.

Webpack

> 설정파일 이용 번들링

```
const path = require("path")

module.exports = {
  mode: "development",
  entry: {
    main: "./src/app.js", // 번들링 진입점
  },
  output: {
    filename: "[name].js", // entry.main이 문자열로 들어옴
    path: path.resolve('./dist'),
  },
}
```

Webpack

> NPM 커스텀 명령어 추가

- package.json의 scripts 요소에 다음 내용 추가

```
// in package.json
// ...
{
  "scripts": {
    "build": "./node_modules/.bin/webpack"
  }
}
// ...
```

Webpack

> NPM 커스텀 명령어 추가

- npm을 이용한 스크립트 수행

```
npm run build
```

- 설정을 바꿔 가면서 번들링 결과를 확인해 봅시다.

Webpack: Loader

준비할 파일

- myloader.js

```
module.exports = function myloader(content) {  
  console.log("myloader call");  
  return content.replace("console.log(", "alert(");  
}
```

Webpack: Loader

설정 파일에 로더 추가

```
// ...  
module: {  
  rules: [  
    {  
      test: /\.js$/, // .js 확장자로 끝나는 모든 파일  
      use: [path.resolve("./myloader.js")] // 사용자 정의 로더 적용  
    }  
  ]  
}  
// ...
```

Webpack: Loader

css-loader

준비할 파일

- src/styles.css

변경할 파일

- src/app.js

```
npm install -D css-loader # css-loader 설치
```


Webpack: Loader

css-loader

설정파일에 로더 추가

```
// ...  
module: {  
  rules: [  
    // ...  
    {  
      test: /\.css$/, // .css 확장자로 끝나는 모든 파일  
      use: ['css-loader'], // css-loader 적용  
    },  
  ],  
}  
// ...
```

Webpack: Loader

style-loader

```
npm install -D style-loader # style-loader 설치
```

설정파일에 로더 추가

```
module: {  
  rules: [  
    {  
      test: /\.css$/, // .css 확장자를 가진 파일들  
      use: ['style-loader', 'css-loader'], // style-loader를 앞에 추가  
    }  
  ]  
}
```

Webpack: Loader

file-loader

준비사항

- src/style.css 변경 (이미지 사용)
- src 디렉터리에 이미지 복사

```
npm install -D file-loader # 로더 설치
```