**Webpack - Module**

**Module 정의**

프로그램을 구성하는 내부의 코드가 기능별로 나눠져 있는 형태

**Module 표준**

Module을 사용하기 위해서는 Module을 인식하는 Module system과 Module을 다루는 키워드가 제공되어야 함

* CommonJS (Node.js)
* ESM(ECMAScript 2015~)

**내보내기 가져오기 CommonJS 예**

// index.js

/\*\*

1. 원의 넓이를 구하는 공식

2. PI 정의

3. 공식을 통해 결과값 얻기

\*\*/

const PI = 3.14;

const getCircleArea = r => r\*r\*PI;

const result = getCircleArea(2);

console.log(result);

// mathUtil.js

const PI = 3.14;

const getCircleArea = r => r\*r\*PI;

module.exports = {

PI,

getCircleArea

}

// 또는

// exports.PI = PI;

// exports.getCircleArea = getCircleArea;

// 한가지 방식으로 통일해서 사용할 것

// index. js

const mathUtil = require('./mathUtil');

// 또는

// const { getCircleArea } = require('./mathUtil');

const result = getCircleArea(2);

console.log(result);

**내보내기 가져오기 ESM 예**

외부 모듈 설치 필요  
$ npm install esm  
$ node -r esm index.js

// CommonJS 예시와 동일

// import 모듈\_이름 from 모듈\_위치

// export 또는 export default

// index.js

import { getCircleArea } from './mathUtil';

// mathUtil.js

export {

PI,

getCircleArea

}

// 또는

// export default { PI, getCircleArea }

// index.js에서

// import mathUtil from './mathUtil';

// const result = mathUtil.getCircleArea(2);

// console.log(result);

**Module의 종류**

1. Built-in Core Module (예: Node.js module)
2. Community-based Module (예: NPM)  
   npm CLI를 사용해야 함
3. Local Module ( 특정 프로젝트에 정의된 모듈)

**Module 예**

**도형의 넓이를 구하는 코드를 기능별로 모듈화**

// readline.js

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

// .question(문자열, 콜백함수);

rl.question('원하는 도형을 작성해주세요: ', input => {

console.log(input);

rl.close();

})

// $ node readline.js

// $ 원하는 도형을 작성해주세요 : 원 + Enter

// $ 원

// index.js

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

rl.question(

"넓이를 구하고자 하는 도형의 종류를 입력해주세요.(정사각형, 원): ",

figure => {

console.log(`선택된 도형: ${figure}`);

switch(figure){

case "정사각형" :

rl.question("변의 길이를 입력해주세요: ", input=>{

console.log(`입력받은 값: ${input}`);

console.log(`정사각형의 넓이는 : ${input\*input}입니다`);

rl.colse();

break;

case "원" :

rl.question("반지름의 길이를 입력해주세요: ", input=>{

console.log(`입력받은 값: ${input}`);

console.log(`원의 넓이는 : ${input\*input\*3.14}입니다`);

rl.colse();

break;

default :

console.log(

"지원되지 않는 도형입니다. \n커맨드라인을 종료합니다."

);

rl.close();

}

}

}):

// mathUtil.js

const PI = 3.14;

const getCircleArea = r => r\*r\*PI;

const getSquareArea = d => d\*d;

module.exports = {

PI,

getCircleArea,

getSquareArea

}

// log.js

const loginput = `입력받은 값: ${input}`;

const logResult = (figure,result)=> `${figure}의 넓이는 : ${result}입니다.`;

const logFigureError = "지원되지 않는 도형입니다. \n커맨드라인을 종료합니다."

module.exports = {

loginput,

logResult,

logFigureError

}

// index.js를 다시 수정하면

const readline = require("readline");

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

const { getCircleArea, getSquareArea } = require('./mathUtil');

const { logFigureError, loginput, logResult } = require('./log');

rl.question(

"넓이를 구하고자 하는 도형의 종류를 입력해주세요.(정사각형, 원): ",

figure => {

console.log(`선택된 도형: ${figure}`);

switch(figure){

case "정사각형" :

rl.question("변의 길이를 입력해주세요: ", input=>{

console.log(loginput(input));

console.log(logResult(figure, getSquareArea(input)));

rl.colse();

break;

case "원" :

rl.question("반지름의 길이를 입력해주세요: ", input=>{

console.log(loginput(input));

console.log(logResult(figure, getCircleArea(input)));

rl.colse();

break;

default :

console.log(logFigureError);

rl.close();

}

}

}):

# Webpack - Bundle

### Bundle

webpack을 사용하면 웹에 사용되는 다양한 파일 .js, .sass, .jpg, .png, .hbs... 을 모듈로 사용할 수 있게 됨

bundle을 사용하면 확장자별로 하나의 파일이 만들어짐

### Bundle의 중요성

bundle.js  
1. 모든 모듈을 로드하기 위해 검색하는 시간이 단축  
2. 사용하지 않는 코드는 제거  
3. 파일 크기 감소 → 린더 트리 파싱 속도 증가

# Webpack - Handlebars, Caching, Minification, Mangling

### Handlebars

.hbs

// $ npm i handlebars -D

// $ npm i handlebars-loader -D

// template.hbs에 <meta> 태그 삭제

// template.hbs

<title>

Webpack {{htmlWebpackPlugin.options.title}}

</title>

// webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

extry : './index.js',

output : {

filename: 'bundle.js',

path : path.resolve(\_\_dirname, 'dist')

},

modules : {

rules : [

{

test : /\.css$/i,

use : [

{

loader : 'style-loader',

options : {

injectType : 'singletonStyleTag'

}

},

loader : 'css-loader',

options : {

modules : true

}

}

]

},

{

test : /\.hbs$/,

use : ['handlebars-loader']

}

]

},

// template.html을 template.hbs로 수정

plugins : [

title : 'Webpack',

template : './template.html',

meta : {

viewport: 'width=device-width, initial-scale=1.0'

})

],

mode : 'none'

}

// $ npm run build

// dist/index.html title 수정되었는지 확인

### Caching

시간, 비용 최소화시키기 위해 캐싱 사용(리소스 복사본)  
서버보다 가까운 위치에 데이터를 놓고 클라이언트에 제공

#### hash의 종류

hash, contenthash, chunckhash

// webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

extry : './index.js',

output : {

// filename: 'bundle.js',

filename : 'bundle..[hash].js',

path : path.resolve(\_\_dirname, 'dist')

},

modules : {

rules : [

{

test : /\.css$/i,

use : [

{

loader : 'style-loader',

options : {

injectType : 'singletonStyleTag'

}

},

loader : 'css-loader',

options : {

modules : true

}

}

]

},

{

test : /\.hbs$/,

use : ['handlebars-loader']

}

]

},

// template.html을 template.hbs로 수정

plugins : [

title : 'Webpack',

template : './template.html',

meta : {

viewport: 'width=device-width, initial-scale=1.0'

})

],

mode : 'none'

}

// $ npm run build

// 바뀐 해쉬값이 적용된 파일이 계속 누적되어 지저분해진다.

// $ npm install clean-webpack-plugin -D

// webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {

extry : './index.js',

output : {

filename: 'bundle.js',

path : path.resolve(\_\_dirname, 'dist')

},

modules : {

rules : [

{

test : /\.css$/i,

use : [

{

loader : 'style-loader',

options : {

injectType : 'singletonStyleTag'

}

},

loader : 'css-loader',

options : {

modules : true

}

}

]

},

{

test : /\.hbs$/,

use : ['handlebars-loader']

}

]

},

// template.html을 template.hbs로 수정

plugins : [

title : 'Webpack',

template : './template.html',

meta : {

viewport: 'width=device-width, initial-scale=1.0'

}),

new CleanWebpackPlugin()

],

mode : 'none'

}

// $ npm run build

// dist 파일이 깔끔하게 정리되었는지 확인

### Minification & Mangling

Module을 사용하지 않고 직접 최적화 시키기  
Minification : 소스코드 리소스 최적화 시키기, 주석, console.log 제거. 어플리케이션이 동작하기 위해 필요한 소스만 남긴다.  
Mangling : 난독화 과정

// webpack github - option 참고

// webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

module.exports = {

extry : './index.js',

output : {

// filename: 'bundle.js',

filename : 'bundle..[hash].js',

path : path.resolve(\_\_dirname, 'dist')

},

modules : {

rules : [

{

test : /\.css$/i,

use : [

{

loader : 'style-loader',

options : {

injectType : 'singletonStyleTag'

}

},

loader : 'css-loader',

options : {

modules : true

}

}

]

},

{

test : /\.hbs$/,

use : ['handlebars-loader']

}

]

},

// template.html을 template.hbs로 수정

plugins : [

title : 'Webpack',

template : './template.html',

meta : {

viewport: 'width=device-width, initial-scale=1.0'

})

],

mode : 'none'

}

// $ npm run build

// 바뀐 해쉬값이 적용된 파일이 계속 누적되어 지저분해진다.

// $ npm install clean-webpack-plugin -D

// webpack.config.js

const path = require('path');

const HtmlWebpackPlugin = require('html-webpack-plugin');

const { CleanWebpackPlugin } = require('clean-webpack-plugin');

module.exports = {

extry : './index.js',

output : {

filename: 'bundle.js',

path : path.resolve(\_\_dirname, 'dist')

},

modules : {

rules : [

{

test : /\.css$/i,

use : [

{

loader : 'style-loader',

options : {

injectType : 'singletonStyleTag'

}

},

loader : 'css-loader',

options : {

modules : true

}

}

]

},

{

test : /\.hbs$/,

use : ['handlebars-loader']

}

]

},

// template.html을 template.hbs로 수정

plugins : [

title : 'Webpack',

template : './template.html',

meta : {

viewport: 'width=device-width, initial-scale=1.0'

},

minify : {

callapseWhiteSpace : true,

useShortDoctype : true,

removeScriptTypeAttribute : true

}

}),

new CleanWebpackPlugin()

],

mode : 'none'

}

// $ npm run build

// dist/index.html 파일 확인

// CSS 최적화 - cssnano

// $ npm i cssnano optimize-css-assets-webpack-plugin -D

// Github Example 참고

// JS 최적화

// $ npm i terser-webpack-plugin -D