9.2 Node.js & npm

모듈화와 npm(node package manager)

#1. 모듈화와 CommonJS

모듈이란 애플리케이션을 구성하는 개별적 요소를 말한다. 일반적으로 파일 단위로 분리되어 있으며 필요에 따라 애플리케이션은 명시적으로 모듈을 로드한다. 모듈은 애플리케이션에 분리되어 개별적으로 존재하다가 애플리케이션의 로드에 의해 비로소 애플리케이션의 일원이 된다. 모듈은 기능별로 분리되어 작성되므로 개발효율성과 유지보수성의 향상을 기대할 수 있다.

자바스크립트는 웹페이지에 있어서 보조적인 기능을 수행하기 위해 한정적인 용도로 만들어진 태생적 한계로 다른 언어에 비해 부족한(나쁜) 부분이 있는 것이 사실이다. 그 대표적인 것이 모듈 기능이 없는 것이다.

C언어는 #include, Java는 import 등 대부분의 언어는 모듈 기능을 가지고 있다. 하지만 Client-side JavaScript의 경우, script 태그를 사용하여 외부의 스크립트 파일을 가져올 수는 있지만 파일마다 독립적인 파일 Scope를 갖지 않고 하나의 전역 객체(Global Object)에 바인딩되기 때문에 전역변수가 중복되는 등의 문제가 발생할 수 있다. 이것으로는 모듈화를 구현할 수 없다.

JavaScript를 Client-side에 국한하지 않고 범용적으로 사용하고자 하는 움직임이 생기면서 모듈 기능은 반드시 해결해야하는 핵심 과제가 되었고 이런 상황에서 제안된 것이 CommonJS와 AMD(Asynchronous Module Definition)이다.

CommonJS과 AMD는 사양(spec)으로 CommonJS 또는 AMD라는 라이브러리가 존재하는 것은 아니다.

CommonJS 방식은 AMD에 비해 문법이 간단하며 동기 방식(synchronous loading)으로 동작한다.

AMD 방식은 CommonJS에 비해 문법이 다소 까다로우며 CommonJS와는 달리 비동기 방식(asynchronous loading)으로 동작한다. AMD 방식을 지원하는 대표적인 모듈 로더는 RequireJS이다.

Node.js는 사실상 모듈 시스템의 사실상 표준(de facto standard)인 CommonJS를 채택하였고 현재는 독자적인 진화를 거쳐 CommonJS 사양과 100% 동일하지는 않지만 기본적으로 CommonJS 방식을 따르고 있다. Node.js에서 모듈의 사용 방법에 대해서는 Node.js module을 참고하기 바란다.

브라우저에서의 모듈 사용은 대부분의 브라우저가 ES6의 모듈을 지원하지 않고 있으므로 Browserify 또는 webpack과 같은 모듈 번들러를 사용하여야 한다.

모듈화에 대한 자세한 사항은 JavaScript 표준을 위한 움직임: CommonJS와 AMD에 잘 정리되어 있다.

2. npm

npm(node package manager)은 자바스크립트 패키지 매니저이다. Node.js에서 사용할 수 있는 모듈들을 패키지화하여 모아둔 저장소 역할과 패키지 설치 및 관리를 위한 CLI(Command line interface)를 제공한다. 자신이 작성한 패키지를 공개할 수도 있고 필요한 패키지를 검색하여 재사용할 수도 있다.

2.1 패키지 설치

Node.js에서 사용할 수 있는 모듈인 패키지를 설치할 때에는 `npm install` 명령어 뒤에 설치할 패키지 이름을 지정한다.

BASH

```
$ npm install <package>
```

Node.js 환경에서 emoji를 지원하는 node-emoji를 설치해 보자. 먼저 적당한 위치에 프로젝트 디렉터리를 생성하고 프로젝트 디렉터리로 이동한다.

BASH

```
$ mkdir emoji && cd emoji
```

npm install 명령어로 node-emoji 패키지를 설치한다.

BASH

```
$ npm install node-emoji
npm WARN saveError ENOENT: no such file or directory, open '/Users/leeungmo/
Desktop/emoji/package.json'
npm notice created a lockfile as package-lock.json. You should commit this f
ile.
npm WARN enoent ENOENT: no such file or directory, open '/Users/leeungmo/Des
ktop/emoji/package.json'
npm WARN emoji No description
npm WARN emoji No repository field.
npm WARN emoji No README data
npm WARN emoji No license field.

+ node-emoji@1.10.0
added 2 packages from 4 contributors and audited 2 packages in 0.566s
found 0 vulnerabilities
```

경고가 발생하였으나 node_modules 폴더가 생성되고 그 내부에 node-emoji 패키지가 설치되었다. 경고에 대해서는 잠시 후에 알아보도록 하자.

2.2 지역 설치와 전역 설치

npm install 명령어에는 지역(local) 설치와 전역(global) 설치 옵션이 있다. 옵션을 별도로 지정하지 않으면 지역으로 설치되며, 프로젝트 루트 디렉터리에 `node_modules` 디렉터리가 자동 생성되고 그 안에 패키지가 설치된다. 지역으로 설치된 패키지는 해당 프로젝트 내에서만 사용할 수 있다.

BASH

```
# 지역 설치
$ npm install <package>
```

전역에 패키지를 설치하려면 npm install 명령어에 `-g` 옵션을 지정한다. 전역으로 설치된 패키지는 전역에서 참조할 수 있다. 모든 프로젝트가 공통 사용하는 패키지는 지역으로 설치하지 않고 전역에 설치한다.

BASH

```
# 전역 설치
$ npm install -g <package>
```

전역에 설치된 패키지는 OS에 따라 설치 장소가 다르다.

- macOS의 경우
  /usr/local/lib/node_modules
- 윈도우의 경우
  c:\Users\%USERNAME%\AppData\Roaming\npm\node_modules

node 명령어로 Node.js REPL을 실행하고 node-emoji를 로드한 후 emoji를 출력해 보자.

BASH

```
$ node
Welcome to Node.js v12.3.1.
Type ".help" for more information.
> const emoji = require('node-emoji').emoji;
undefined
> console.log(emoji.heart);
❤
undefined
```

2.3 package.json과 의존성 관리

앞에서 npm install 명령어로 node-emoji 패키지를 설치할 때 `package.json` 을 찾을 수 없다는 경고가 발생하였다.

BASH

```
npm WARN saveError ENOENT: no such file or directory, open '/Users/leeungmo/Desktop/emoji/package.json'
...
```

Node.js 프로젝트에서는 많은 패키지를 사용하게 되고 패키지의 버전도 빈번하게 업데이트되므로 프로젝트가 의존하고 있는 패키지를 일괄 관리할 필요가 있다. npm은 `package.json` 파일을 통해서 프로젝트 정보와 패키지의 의존성(dependency)을 관리한다. 이미 작성된 package.json이 있다면 팀 내에 배포하여 동일한 개발 환경을 빠르게 구축할 수 있는 장점이 있다. package.json은 Java의 maven에서 pom.xml과 비슷한 역할을 한다.

package.json을 생성하려면 프로젝트 루트에서 `npm init` 명령어를 실행한다.

BASH

```
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (emoji)
```

npm init 명령어를 사용하면 프로젝트에 대한 여러 가지 정보를 입력하도록 요구받는다. 이때 입력된 정보를 바탕으로 npm은 package.json 파일을 생성한다. 일단 기본 설정값으로 생성된 package.json 파

일을 수정하는 방법이 더 편리할 수 있으므로 npm init 명령어에 `--yes` 또는 `-y` 옵션을 추가한다. 그러면 기본 설정값으로 package.json 파일을 생성한다.

BASH

```
$ npm init -y
Wrote to /Users/leeungmo/Desktop/emoji/package.json:

{
  "name": "emoji",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "dependencies": {
    "node-emoji": "^1.10.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

package.json에서 가장 중요한 항목은 `name` 과 `version` 이다. 이것으로 패키지의 고유성을 판단하므로 생략할 수 없다. 그리고 `dependencies` 항목에는 해당 프로젝트가 의존하는 패키지들의 이름과 버전을 명시한다. 여기서 의존하는 패키지란 해당 프로젝트에서 참조하는 모듈을 의미한다. 프로젝트를 진행할 때는 이미 만들어진 여러 패키지를 참조해서 사용하는데, package.json 파일의 dependencies 항목에 해당 패키지의 이름과 버전을 명시함으로써 의존성을 설정한다.

npm install 명령어에 `--save` 옵션을 사용하면 패키지 설치와 함께 package.json의 dependencies에 설치된 패키지 이름과 버전이 기록된다.

BASH

```
$ npm install --save node-emoji
```

> npm@5부터 `--save` 는 기본 옵션이 되었다. `--save` 옵션을 사용하지 않더라도 모든 install 명령은 package.json의 dependencies에 설치된 패키지와 버전을 기록한다. 기존의 `--save-dev` 은 변경되지 않았다.

`devDependencies` 에는 개발 시에만 사용하는 개발용 의존 패키지를 명시한다. 예를 들어 TypeScript와 같은 트랜스파일러는 개발 단계에서만 필요하고 배포할 필요는 없으므로 devDependencies에 포함시킨다. npm install 명령어에 `--save-dev` (축약형 -D) 옵션을 사용하면 패키지 설치와 함께 package.json의 devDependencies에 설치된 패키지와 버전이 기록된다.

BASH

```
$ npm install --save-dev <package>
```

npm install 명령어를 사용하면 package.json에 명시된 모든 의존 패키지를 한번에 설치할 수 있다.

BASH

```
$ npm install
```

package.json에 관한 더 자세한 설명은 https://docs.npmjs.com/files/package.json을 참조하기 바란다.

2.4 Semantic versioning(유의적 버전)

npm install 명령어의 패키명 뒤에 @버전을 추가하면 패키지 버전을 지정하여 설치할 수 있다.

BASH

```
$ npm install node-emoji@1.5.0
```

JSON

```
...
  "dependencies": {
```

```
    "node-emoji": "^1.5.0"
  },
...
```

이때 package.json의 `dependencies` 에 새롭게 추가한 패키지가 추가되고 버전 앞에 `^` (캐럿)이 추가된 것을 확인할 수 있다. 이것은 패키지 버전을 지정하였을 때 뿐만이 아니라 `--save-exact` 옵션을 지정하지 않으면 기본적으로 추가되는 것이다. 이 `^` (캐럿)은 이후 해당 패키지의 버전이 업데이트되었을 경우, 마이너 버전 범위 내에서 업데이트를 허용한다는 의미이다.

> npm install 명령어에 `--save-exact` 옵션을 지정하면 설치된 버전을 범위 지정없이 기록한다.

다시 `npm install node-emoji` 을 실행하면 최신 버전 1.10.0로 자동 업데이트된다.

JSON

```
...
  "dependencies": {
    "node-emoji": "^1.10.0"
  },
...
```

npm은 semantic versioning(유의적 버전)을 지원한다. 버전 정보는 메이저 버전 번호, 마이너 버전 번호, 패치 버전 번호로 구성된다.

semantic versioning(유의적 버전)

버전 정보 앞에는 기호를 부여하여 업데이트 범위를 지정할 수 있다. 기술 방식은 아래와 같다.

표기법	Description
version	명시된 version과 일치
>version	명시된 version보다 높은 버전
>=version	명시된 version과 같거나 높은 버전
<version	명시된 version보다 낮은 버전
<=version	명시된 version과 같거나 낮은 버전
~version	명시된 version과 근사한 버전
^version	명시된 version과 호환되는 버전

~(틸트)와 ^(캐럿)의 차이는 아래와 같다.

~(틸트)는 패치 버전 범위 내에서 업데이트한다. :

- ~0.0.1 : 0.0.1 <= version < 0.1.0
- ~0.1.1 : 0.1.1 <= version < 0.2.0

^(캐럿)는 마이너 버전 범위 내에서 업데이트한다. :

- ^1.0.2 : 1.0.2 <= version < 2.0

npm semver calculator에 방문하면 패키지 별로 버전 표기법을 사용하여 업데이트 버전 범위를 확인할 수 있다.

버전에 대한 보다 자세한 사항은 semver : The semantic versioner for npm를 참조하기 바란다.

2.5 자주 사용하는 npm 명령어

package.json 생성

BASH

```
$ npm init
# 기본 설정
$ npm init -y
```

패키지 설치

BASH

```
# 로컬 설치
$ npm install <package-name>
# 전역 설치
$ npm install -g <package-name>
# 개발 설치
$ npm install --save-dev <package-name>
# package.json의 모든 패키지 설치
$ npm install
```

패키지 제거

BASH

```
# 로컬/개발 패키지 제거
$ npm uninstall <package-name>
# 전역 패키지 제거
$ npm uninstall -g <package-name>
```

패키지 업데이트

BASH

```
$ npm update <package-name>
```

전역 설치 패키지 확인

BASH

```
$ npm ls -g --depth=0
```

package.json scripts 프로퍼티의 start 실행

BASH

```
$ npm start
```

package.json scripts 프로퍼티의 start 이외의 scripts 실행

BASH

```
$ npm run <script-name>
```

전역 패키지 설치 폴더 확인

BASH

```
$ npm root -g
/usr/local/lib/node_modules
# 파인더 오픈
$ open /usr/local/lib/node_modules
```

패키지 정보 참조

BASH

```
$ npm view <package-name>
```

BASH

```
# react-create-app 패키지 정보 확인
$ npm view react-create-app

react-create-app@2.0.6 | MIT | deps: 19 | versions: 19
This helper exports a function returning a ready-to-use React App component
with the following: - react-router-redux - history - redux-persist

dist
.tarball: https://registry.npmjs.org/react-create-app/-/react-create-app-2.0
.6.tgz
.shasum: 88b10bc9a53e58a8b08b7a4cdf8cd6b21f8113fe
.integrity: sha512-YT8WiWx9wuJFM35EyW/Z9R/75MYj3hW9DsT7f1Dj6M3bdTRnWOocOh9r4
```

```
HkM5OKFQDzZwTvyQrzITTEn6JxCqg==

dependencies:
babel-eslint: 7.2.3
babel-jest: 20.0.3
babel-loader: 7.1.2
babel-plugin-transform-class-properties: 6.24.1
babel-plugin-transform-decorators-legacy: 1.3.4
babel-preset-env: next
babel-preset-react: 6.24.1
babel-preset-stage-0: 6.24.1
eslint-loader: 1.9.0
eslint-plugin-import: 2.7.0
eslint-plugin-jest: 20.0.3
eslint-plugin-react: 7.3.0
eslint: 4.5.0
prop-types: 15.5.10
react-redux: 5.0.6
react: 15.6.1
redux-persist-crosstab: 3.6.0
redux-persist: 4.9.1
redux: 3.7.2

maintainers:
- damianobarbati <damiano@mvpbld.com>

dist-tags:
latest: 2.0.6

published over a year ago by damianobarbati <damiano@mvpbld.com>
```

BASH

```
# eslint-config-airbnb와 함께 설치해야 하는 다른 패키지 확인
$ npm view eslint-config-airbnb@latest peerDependencies

{
  eslint: '^5.16.0 || ^6.1.0',
  'eslint-plugin-import': '^2.18.2',
  'eslint-plugin-jsx-a11y': '^6.2.3',
  'eslint-plugin-react': '^7.14.3',
```

```
    'eslint-plugin-react-hooks': '^1.7.0'
  }
```

버전 확인

BASH

```
$ npm -v
```

npm 명령어 설명 참조

BASH

```
$ npm help <command>
```

npm 명령어에 대한 자세한 설명은 npm CLI 명령어을 참조하기 바란다.

Reference

- CommonJS
- npm
- package.json
- npm CLI 명령어
- ES Modules와 Node.js: 쉽지 않은 선택