1. Module

❖ 모듈이란?

- 노드 애플리케이션의 기본 단위로 자바스크립트 파일 하나가 하나의 모듈을 이룸
- 데이터, 템플릿 등의 파일이나 다른 모듈이 포함된 폴더 단위의 모듈도 가능
- 노드 어플리케이션은 하나 이상의 모듈로 구성되며 require() 함수를 이용해서 외부 모듈 사용 가능
  - ES6에서는 import!!

❖ 모듈의 종류

- 코어 모듈
  - 자바스크립트에서는 제공하지 않는 파일 입출력, 네트워크 통신 등 노드에서 기본으로 제공하는 모듈
  - OS, Process, File System, Net, HTTP 등
  - 주로 low-level API 이므로 코어 모듈만 가지고 개발하기에는 처리할 작업이 많음
- 확장 모듈
  - 개인이나 서드파티에서 개발해서 배포하는 모듈(라이브러리)
  - 코어 모듈의 부족한 부분을 보완하여 확장한 모듈(high-level API)
  - npm으로 설치 -> node_modules 폴더 하위에 폴더단위로 저장됨
- 사용자 모듈
  - 프로젝트를 구성하는 내부 모듈
  - node_modules 폴더 이외의 위치에 저장된 모듈

2. Module System

❖ Node 모듈 시스템

- 코어모듈은 최대한 간결함을 유지
- CommonJS 모듈 시스템을 준수

❖ CommonJS

- 자바스크립트를 브라우저에서뿐만 아니라 서버, 데스크크탑 환경에서도 사용하려고 정의한 자바스크립트 스펙
- 자바스크립트 표준(ECMAScript)은 웹브라우저에서 동작하는 API 위주로 정의
- CommonJS는 브라우저 밖에서 동작하기 위한 API 정의(서버사이드, 데스크탑 GUI 프로그램 등)
- binary, encodings, io, fs, sockets 등의 스펙을 정의

❖ CommonJS 모듈 시스템 요구사항

- 모듈 식별자를 받아서 노출된 API를 반환하는 require 함수에 대한 지원
- 모듈명은 캐릭터 문자열이며, 경로 식별을 위해 /를 포함할 수 있음
- 모듈은 모듈 외부로 노출될 항목을 명시적으로 내보내기 해야함
- 변수는 모듈에 대해 private임

3. NPM

❖ NPM(Node Package Manager)

- 노드 모듈 관리 시스템
- 노드 설치시 기본으로 설치됨(0.6.3 이후)
- 확장 모듈 검색, 설치, 업데이트, 삭제 등의 기능 제공
- http://npmjs.com/

❖ NPM 주요 명령어

- npm install [모듈명[@버전]]
- npm uninstall 모듈명
- npm update [모듈명[@버전]]
- npm ls [모듈명]
- npm view 모듈명[@버전] [필드]
- npm init

3. NPM

❖ 모듈 설치

- npm install [모듈명[@버전]]
- 지정한 버전의 확장모듈을 설치한다.
- 버전 생략 시 최신버전 설치
- 모듈명 생략 시 package.json의 dependencies 항목의 확장 모듈 전체 설치
- 이미 설치된 모듈일 경우 삭제하고 지정한 버전으로 재설치
- npm install connect

❖ 모듈 삭제

- npm uninstall 모듈명
- 지정한 모듈을 삭제한다.
- npm uninstall connect

❖ 모듈 업데이트

- npm update [모듈명]
- 지정한 모듈을 업데이트 한다.
- package.json의 dependencies에 정의된 모듈 버전 정보에 따라 업데이트 한다.
- npm update connect

3. NPM

❖ 설치된 모듈 확인

- npm ls [모듈명]
- 설치된 확장 모듈 중 지정한 확장모듈의 정보를 보여준다.
- 모듈명 생략 시 설치된 모든 확장 모듈의 정보를 보여준다.
- 상세 정보는 npm ll로 확인
- npm ls connect

❖ 레퍼지토리에서 모듈 확인

- npm view 모듈명[@버전] [필드]
- npm 레퍼지토리에 등록된 모듈 중 지정한 확장모듈의 정보를 보여준다.
- npm view connect versions

❖ 모듈 패키징

- npm init
- package.json 파일을 생성한다.

4. package.json

❖ package.json
- 모듈을 배포할때 필요한 정보를 기술한 설정파일
- 노드 어플리케이션의 확장모듈 의존성 관리에 필요
- 텍스트 파일로 직접 작성하거나 npm init 명령어로 작성
- CommonJS 스펙 기반

❖ 주요 필드
- name(필수) - 패키지 이름, 중앙저장소(http://www.npmjs.com/)에 url로 사용되고 설치할 디렉토리명이 되므로 url이나 디렉토리명으로 사용할 수 있는 이름
- version(필수) - 패키지 버전, 1.0.3 같은 3단계 버전명
- dependencies - 패키지가 의존하는 모듈 목록
- devDependencies - 개발 의존성 모듈 목록
- description - 패키지 설명
- main - require()에서 로딩할 파일
- repository - 소스코드 저장소
- keywords - 검색어 배열
- author - 개발자정보,
- licenses - 라이선스 배열

5. 모듈 로딩과 경로 지정

❖ 모듈 로딩
- 웹브라우저에서 실행되는 자바스크립트는 <script> 태그를 이용해서 외부 라이브러리를 로딩하지만 노드는 require() 함수를 이용해서 외부 모듈을 로딩
- require('모듈명');

❖ 코어 모듈 로딩
- require('http');
- require('fs');

❖ 확장 모듈 로딩 : node_modules 폴더 밑에서 찾음
- require('underscore');
- require('express');

❖ 사용자 모듈 로딩
- require('절대경로나 상대경로로 시작하는 모듈명');
- require('./m1');
- require('../../m1');
- require('../d1/m1');
- require('/d1/d2/m1');

6. 모듈 로딩 우선 순위

❖ 코어 모듈 > 확장 모듈
- 코어 모듈과 동일한 이름의 확장 모듈이 있을 경우 코어 모듈이 우선한다.

❖ 확장 모듈 로딩
- require()를 호출한 모듈의 하위 폴더 node_modules 에서 찾는다.
- 못찾으면 상위 폴더의 node_modules 폴더에서 찾는다.
- 루트까지 계속 찾는다.
- 루트에도 없을 경우 로딩 실패.

❖ 사용자 모듈을 확장자 없이 기술할 경우
- require('./some');
- some > some.js > some.json > some.node 순서로 로딩
- some 폴더 내에 package.json 파일이 있고 package.json 파일의 main 속성에 지정한 파일이 존재할 경우 해당 파일을 로딩
- some 폴더 내에서 index > index.js > index.json > index.node 순서로 로딩

7. 전역 객체와 모듈 객체

❖ 전역 개체
- 모든 모듈에서 접근할 수 있는 개체
- global 변수로 접근(생략 가능)
- global scope
  - global
  - console
  - process
  - Buffer
  - setTimeout(), clearTimeout(), setInterval(), clearInterval()

❖ 모듈 개체
- 모듈 내부에서만 접근할 수 있는 개체
- module scope
  - __dirname
  - __filename
  - module
  - exports
  - require()

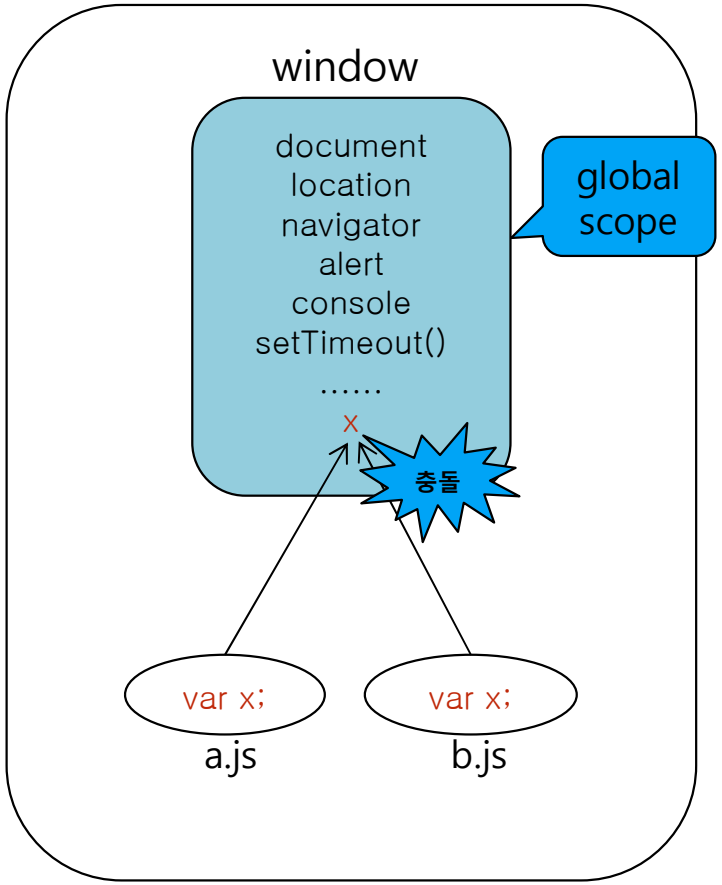
8. 전역 변수, 개체

❖ 전역 변수

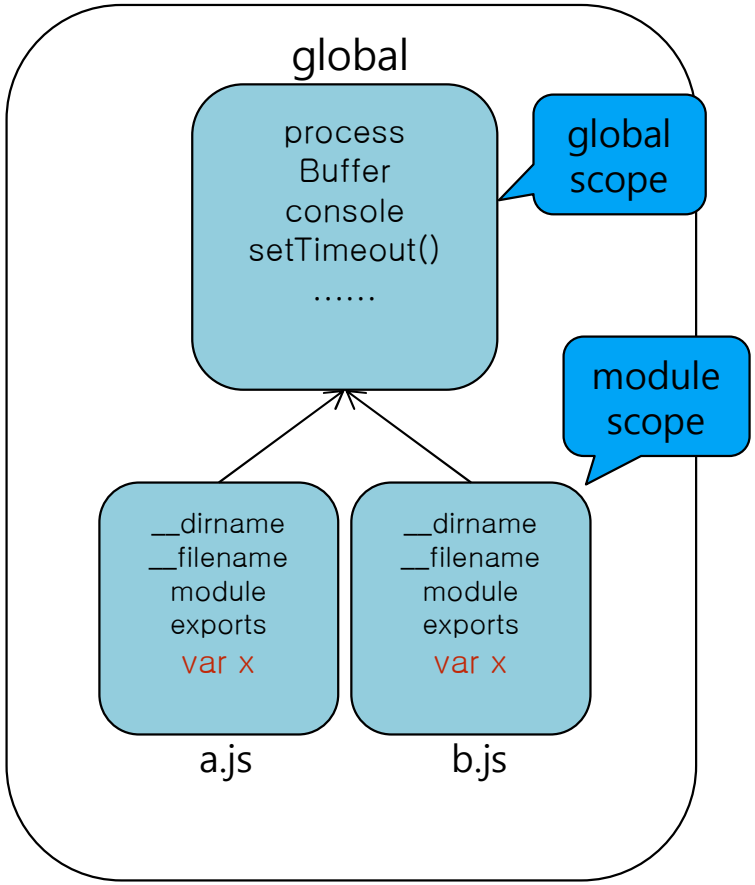
- 웹 브라우저에서 실행되는 자바스크립트의 최상위 변수는 전역 변수가 되지만 노드에서의 최상위 변수는 해당 모듈 내에서만 접근할 수 있는 모듈 변수가 됨
  - var sum = 0;
- 전역 변수를 정의할때는 명시적으로 global 객체의 속성으로 지정하거나 global을 생략하고 지정
  - global.sum = 0; (전역 변수)
  - sum = 0; (전역 변수)
  - var sum = 0; (모듈 변수)

9. Client-side javascript vs. Node

client-side javascript



node



10. 모듈 시스템

- __dirname
  - 현재 실행중인 스크립트의 경로(절대경로)
- __filename
  - 현재 실행중인 스크립트의 파일명(절대경로)
- require(id)
  - 지정한 모듈을 로딩한다.
  - resolve(id)
    - 지정한 모듈의 __filename 반환
  - main
    - node.exe 명령어로 실행한 모듈에 대한 참조
    - 어떤 모듈이 node.exe 명령어로 실행되었는지 여부를 확인하기 위해서는 require.main == module 결과를 확인하면 된다.
  - cache
    - 한번 로딩한 모듈은 require.cache 속성에 저장되고 동일 모듈을 여러번 로딩하면 해당 모듈의 캐시가 반환됨
    - 로딩한 모듈에 대한 캐시정보가 저장된 Object
    - 속성명으로 모듈의 __filename, 값으로 로딩한 모듈객체가 지정됨
    - 속성을 삭제하면 해당 모듈에 대한 캐시가 삭제됨

10. 모듈 시스템

❖ 모듈 내보내기

- exports 또는 module.exports 를 사용하여 내보내기
- ~/module1.js

```
let base = 100;
const add = (x) => base+x;
const multiply = (x) => base*x;

exports.add = add;
exports.multiply = multiply;
```

~/module2.js

```
const hello = "hello ";
const greeting = (msg) => {
    return hello + msg;
}

module.exports = greeting;
```

- module.exports 로는 단 하나의 객체, 변수만 내보낼 수 있음
- exports 로는 여러개의 객체, 변수, 함수를 내보낼 수 있음

10. 모듈 시스템

❖ 모듈 임포트

- require() 를 이용함
  - 상대 경로 지정
- module.exports 한 것은 단일 객체, 변수, 함수로 임포트
- exports 한 것은 객체의 속성 형태로 임포트
- ~/main.js

```
//객체를 받아낸 뒤 객체의 속성 이용
const m1 = require('./module1');
const greeting = require('./module2');

console.log(m1.add(3))
console.log(m1.multiply(3))

console.log(greeting("world"));
```

```
//객체내의 속성을 구조분해 할당으로 받아냄
const { add, multiply } = require('./module1');
const greeting = require('./module2');

console.log(add(3))
console.log(multiply(3))

console.log(greeting("world"));
```

11. 자주 사용하는 기본 모듈

[표 3-2] 자주 사용하는 기본 모듈

종류	메서드
'global' : 전역 객체(Global Variable)	global.exports : 모듈을 내보낼 때 사용합니다. global.require : 모듈을 불러들일 때 사용합니다. global.module : node.js 모듈을 다룰 때 사용합니다. global.__filename : 현재 경로를 포함한 파일 이름입니다. global.__dirname : 현재 실행 중인 파일의 경로입니다.
'process' : 프로세스 객체(Process Object)	process.argv : 실행 파라미터를 출력합니다. process.execPath : 실행 파일 경로입니다. process.cwd : 현재 애플리케이션 경로입니다. process.version : node 버전입니다. process.memoryUsage : 현재 서버의 메모리 상태입니다. process.env : 여러 환경 설정 정보입니다.
'os' : 운영체제 모듈(OS Module)	os.type, os.platform : 운영체제를 확인합니다. os.arch : 서버의 아키텍처를 구분합니다. os.networkInterface : 서버의 지역 IP를 확인합니다.
'fs' : 파일 시스템 모듈(File System Module)	fs.readFile : 파일을 읽습니다. fs.exists : 파일을 확인합니다. fs.writeFile : 파일을 씁니다.
'util' : 유틸 모듈(Util Module)	util.format : 파라미터로 입력한 문자열을 조합합니다.
'url' : URI 모듈(URI Module)	url.parse : url을 객체화합니다. url.format : url을 직렬화합니다.