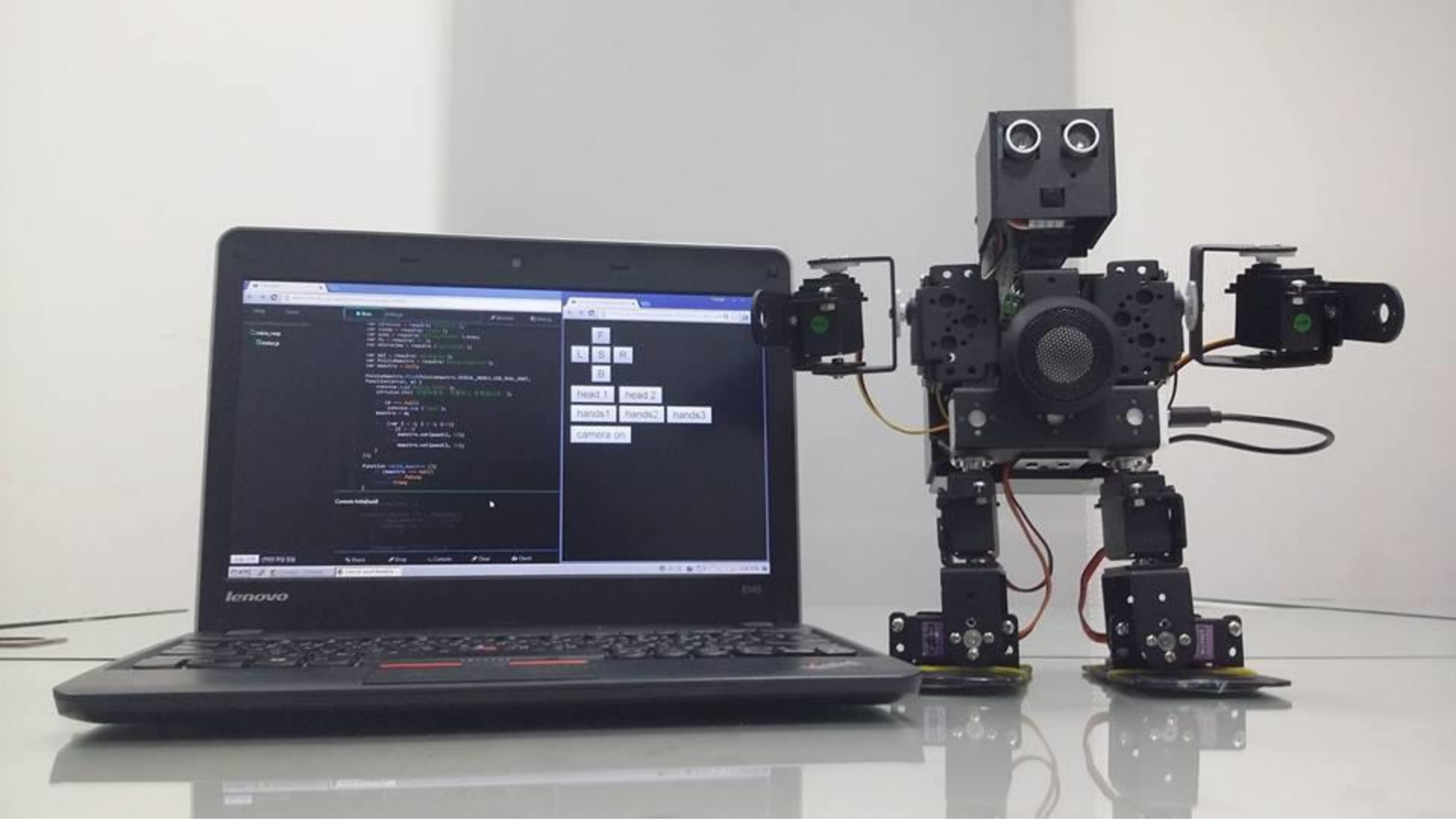
Expand your dimension

Opensource H/W + Node JS IoT 따라잡기

Lesson 3



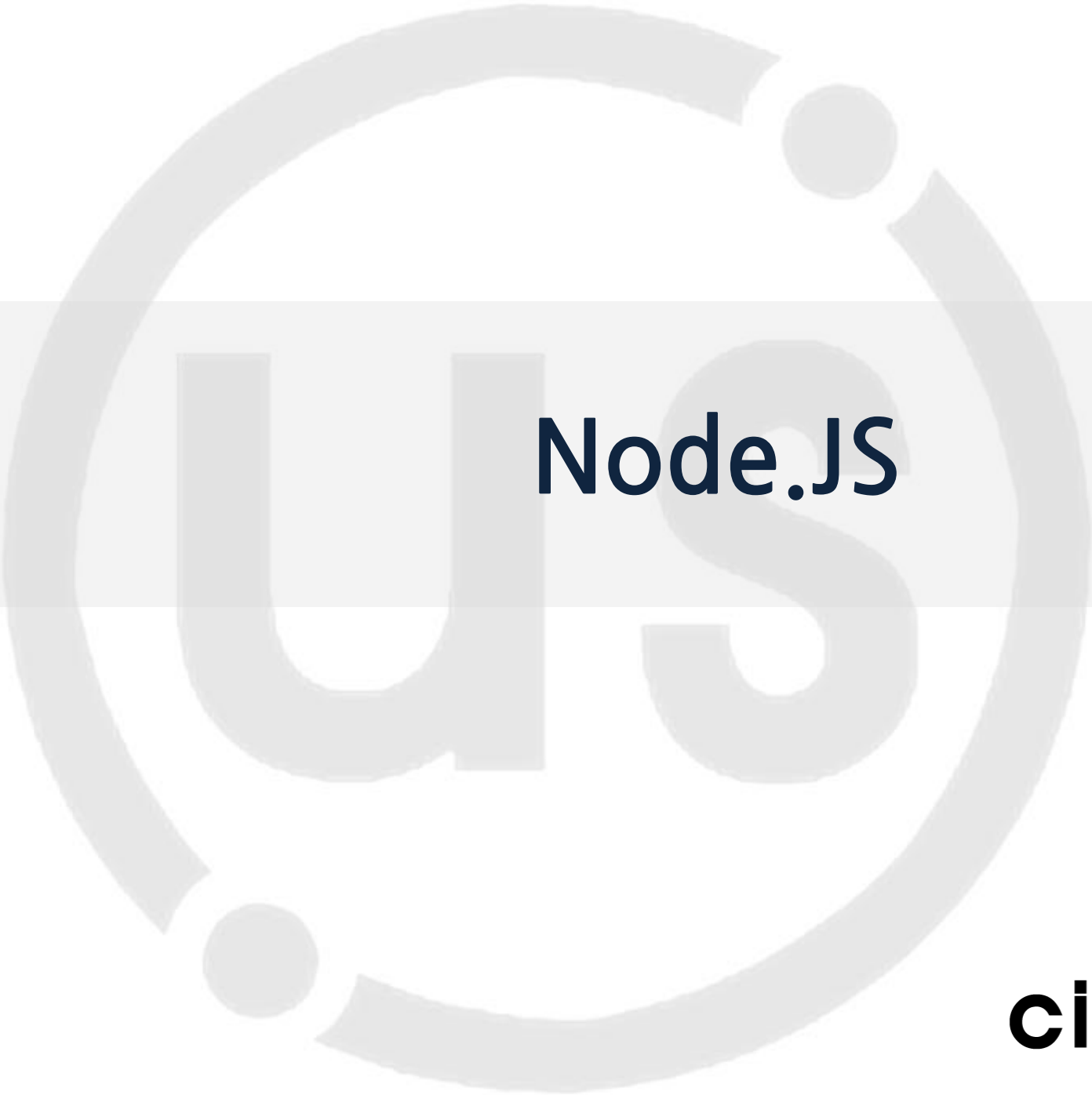
Circulus Robot ***CULU***

누구나 로봇을 만드는 그날까지!

circulus

circul(us)

Node.JS



circul(us)

NodeJS

Node.js는 확장성 있는 네트워크 애플리케이션(특히 서버 사이드) 개발에 사용되는 소프트웨어 플랫폼이다. Node.js는 작성언어로 자바스크립트를 활용하며 Non-blocking I/O와 단일 스레드 이벤트 루프를 통한 높은 처리성능을 가지고 있다.

Node.js는 내장 HTTP 서버 라이브러리를 포함하고 있어 웹서버에서 아파치 등의 별도의 소프트웨어 없이 동작하는 것이 가능하며 이를 통해 웹서버의 동작에 있어 더 많은 통제를 가능케 한다.

Why NodeJS

웹, 서버, 하드웨어 세가지를 한번에

NodeJS 는 Javascript 를 이용한 프로그래밍 언어로써, 웹앱 개발을 위해서 필수적으로 사용하는 Javascript 를 이용하여 클라이언트, 서버, 하드웨어 개발을 한번에 할 수 있는 가장 효율적인 언어이다.

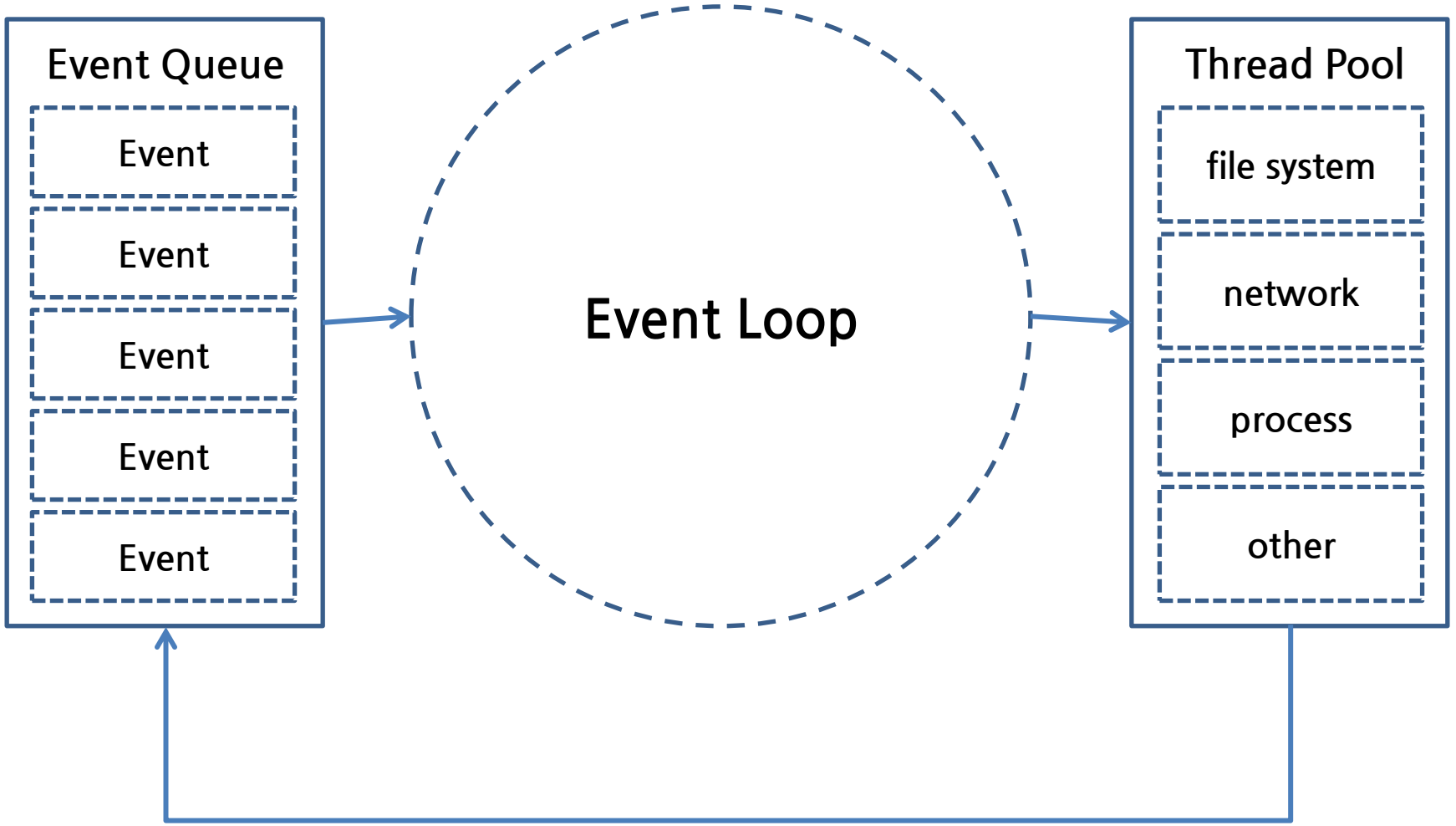
H2H, H2M, M2M 을 한번에

NodeJS 가 부각된 이유중 하나는 Socket.IO 를 이용한 손쉬운 실시간 통신 개발이였다.이 SocketIO 를 이용하여 기존 H2H(Human to Human) 뿐만 아니라, H2M (Human to Machine), M2M (Machine to Machine)을 손쉽게 구현할 수 있다.

Javascript, NodeJS 의 강력한 지원

Github 에서 가장 많은 프로젝트가 Javascript 로 이루어진 라이브러이며, NodeJS 의 30,000 여개가 넘는 강력한 라이브러리를 활용하여 다양한 프로그램을 손쉽게 구현 가능하다

NodeJS


Event Queue
Event
Event
Event
Event
Event
Event Loop
Thread Pool
file system
network
process
other

ExpressJS + Socket.IO

Express

Fast, unopinionated, minimalist web framework for Node.js

socket.io

NodeJS Download

짝수 버전은 stable, 홀수 버전은 unstable

case 1. 직접 다운로드 및 컴파일

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install build-essential libssl-dev pkg-confi
$ wget http://nodejs.org/dist/v0.10.35/node-v0.10.35.tar.gz
$ tar -zxf node-v.10.24.tar.gz
$ ./configure
$ make && sudo make install
```

case 2. 배포판 설치

```
$ wget http://node-arm.herokuapp.com/node_latest_armhf.deb
$ sudo dpkg -i node_latest_armhf.deb
```

Hello world

매우 간단하게 Hello World 를 구축할 수 있음

```
1  var http = require('http');
2
3  http.createServer(function (req, res) {
4      res.writeHead(200, {'Content-Type': 'text/plain'});
5      res.end('Hello World on Raspberry Pi!\n');
6  }).listen(80, '127.0.0.1');
7
8  console.log('Server running at http://127.0.0.1:80/');
```

패키지 설치

전역 설치

/usr/local/node-modules 에 설치되어 어디에서든지 참조 가능하다

$ npm install -g <패키지명>

지역 모드

실행한 위치에 설치되며, 해당 위치 내에서만 참조가 가능하다

$ npm install <패키지명>

버전 설치

$ npm install <패키지명>@<버전>

$ npm install jshint@1.1.0 // 1.1.0 설치

$ npm install jshint@1.1.x // 해당 Branch의 최신 배포판

$ npm install jshint@"<2.0" // 하위 버전

$ npm install jshint@">=0.1.0<2.1" // 0.1과 2.1사이의 최신 배포판

패키지 삭제/업데이트

전역 설치 제거

/usr/local/node-modules 에 설치되어 있는 모듈을 제거한다.

$ npm uninstall -g <패키지명>

지역 설치 제거

현재 폴더 하위 영역에 설치된 패키지를 제거한다.

$ npm uninstall <패키지명>

업데이트

특정 패키지의 업데이트 버전이 있으면, 업데이트 하는 명령이다.

$ npm update <패키지명>
$ npm -g update <패키지명>

모듈 만들기

```
var sum = function(a,b){
        return a+b;
}

exports.sum = sum;
```

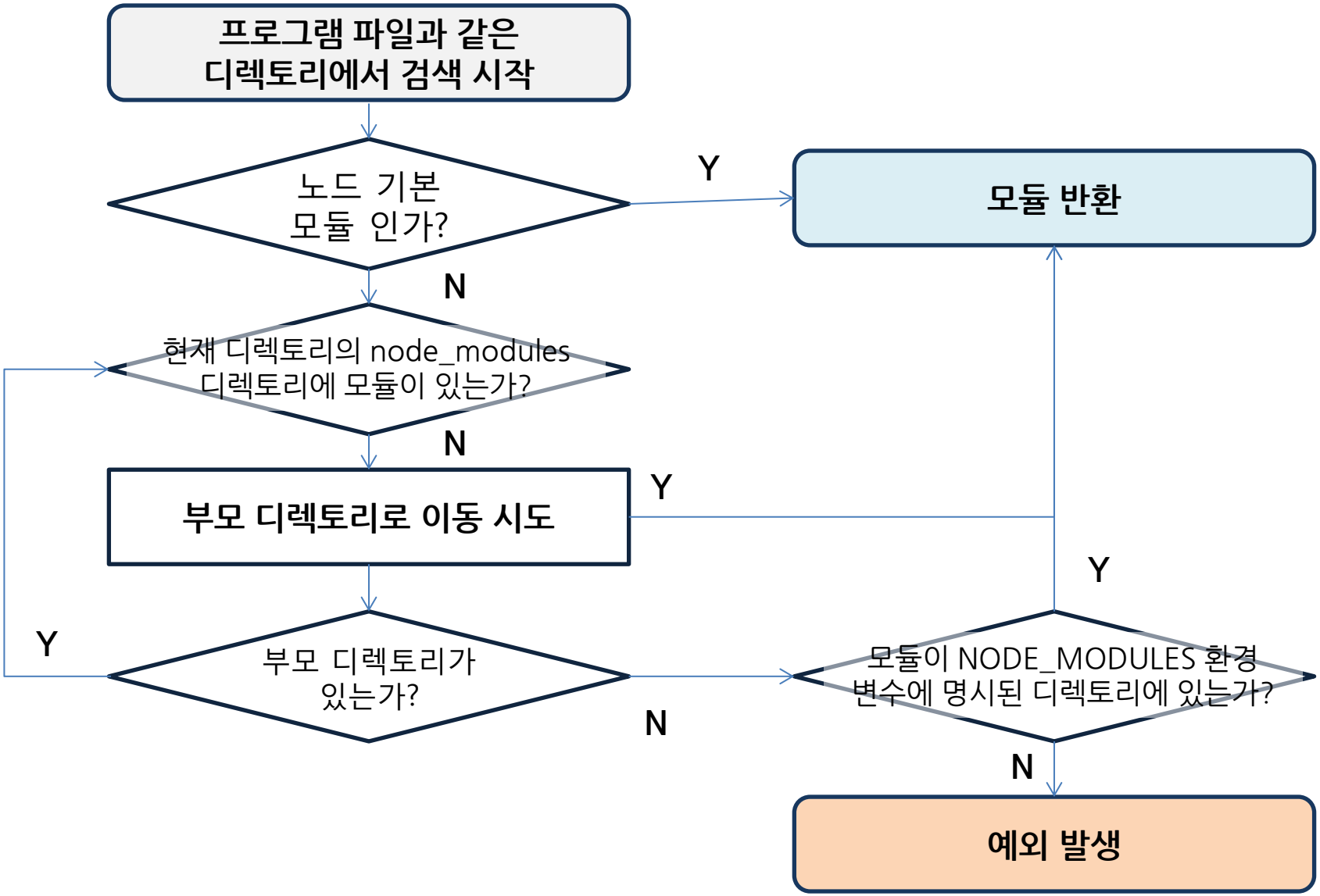
혹은,

```
exports.sum = function(a,b){
        return a+b;
}
```

식으로 직접 선언할 수 있음. 기본적으로 불러오는 법은 다음과 같음

```
var sum = require('./sum.js');
console.log(sum.sum(1,2));
```

모듈 로딩


프로그램 파일과 같은
디렉토리에서 검색 시작
노드 기본
모듈 인가?
Y
모듈 반환
N
현재 디렉토리의 node_modules
디렉토리에 모듈이 있는가?
N
부모 디렉토리로 이동 시도
Y
Y
부모 디렉토리가
있는가?
Y
모듈이 NODE_MODULES 환경
변수에 명시된 디렉토리에 있는가?
N
N
예외 발생

모듈 로딩

코어 모듈

nodejs 배포판에 미리 컴파일 된 모듈

var http = require('http');

모듈 명 만으로 참조함. 같은 이름의 서드파티 모듈이 존재하더라도 우선적으로 로딩 됨

파일 모듈 로딩

절대/상대 경로로 로딩하면 로딩 가능

/home/a/b.js

var module = require('/home/a/b');
var module = require('./b.js');
var module = require('./b');

모듈 로딩

폴더 모듈 로딩

var module = require('./moduleDir');

해당 폴더 내에서 모듈을 찾음
패키지라고 가정하며, package.json 을 찾는다.
package.json 의 main 속성을 분석해 상대경로를 찾는다.
package.json 파일이 없으면 index.js 를 찾는다

node_modules 폴더를 이용한 로딩

var module = require('myModule.js');

위와 같이 작성하면 해당 경로의 ./node_modules/myModule.js 를 찾음.
없으면 상위의 node_modules 를 찾음. 위의 방식으로 root 폴더까지 탐색 후 없으면 오류 반환

내장 객체 process

processs

프로그램과 관련된 정보를 나타내는 객체
브라우저에는 존재하지 않음

메소드/변수 명	설명
argv	실행 매개변수, 어떤 파일에서 실행 된 것인지 알려줌
env	컴퓨터 환경 관련 정보
version	NodeJS 정보 반환 (ex 0.12.01)
arch	프로그램 아키텍쳐 (ex 32bit, 64bit)
platform	플랫폼
memoryUsage()	메모리 사용량
uptime()	프로그램이 실행된 시간
exit()	프로그램 종료

내장 모듈 os

os

운영 체제와 관련된 유틸리티 함수들. require('os') 로 정의하여 사용.

메소드 명	설명
tmpdir()	시스템의 기본 임시 폴더를 반환
hostname()	운영 체제의 호스트 이름
type()	운영 체제의 이름
platform()	운영체제
arch()	CPU 아키텍쳐
uptime()	운영체제가 실행된 시간
loadavg()	운영체제 Load Average
totalmem()	메모리
freemem()	가용 메모리
cpus()	cpu
getNetworkInterfaces()	네트워크 환경

암호화 모듈

crypto 모듈은 해시 생성과 암호화를 생성하는 모듈. id/password 를 저장하는 서버에서 매우 중요함. 해시화 한 결과 값만 저장하기 때문에 해커가 결과 값을 가지고 있어도 정보를 보호할 수 있음

```
1  var crypto = require('crypto');
2  var key = 'websecretkey';
3  var input = '12345678';
4
5  var cipher = crypto.createCipher('aes192',key);
6  cipher.update(input, 'utf-8', 'base64');
7
8  var cipherOutput = cipher.final('base64');
9
10 var decipher = crypto.createDecipher('aes192',key);
11 decipher.update(cipherOutput, 'base64', 'utf-8');
12
13 var decipherOutput = decipher.final('utf-8');
14
15 console.log('input : ' + input);
16 console.log('cipherOutput : ' + cipherOutput);
17 console.log('decipherOutput : ' + decipherOutput);
```

파일 관리

간단하게 파일을 읽고 쓸수 있음. 동기식 방법과 비동기식 방법 모두를 지원하며 상황에 따라 맞는 방식을 선택하여 사용해야 함

```
1  var fs = require('fs');
2
3  // 동기식 파일 읽기
4  var data = fs.readFileSync('/index.html','utf8');
5  console.log(data);
6
7  // 비동기식 파일 읽기
8  fs.readFile('/index.html','utf8', function(err, data){
9      if(err){ throw err; }
10     console.log(data);
11 });
12
13 // 동기식 파일 쓰기
14 fs.writeFileSync('/index.html','Hello World');
15
16 // 비동기식 파일 쓰기
17 fs.writeFile('/index.html','Hello World', function(err){
18     if(err) { throw err; }
19 });
```

디렉토리 관리

디렉토리를 읽고 생성 및 삭제하는 다양한 동기/비동기 라이브러리를 제공함.
fs.stat() 함수를 이용하여 파일에 대한 정보를 획득할 수 있음.

```
1  var fs = require('fs');
2  var path = '/';
3
4  fs.readdir(path, function(err, files){
5      if(err) throw err;
6      files.forEach(function(file){
7          console.log(path + file);
8          fs.stat(path + file, function(err, stats){
9              console.log(stats);
10         });
11     });
12 });
13
14 fs.mkdir('/test', 0666, function(err){
15     if(err) throw err;
16     fs.rmdir('/test', function(err){
17         if(err) throw err;
18     })
19 });
```

파일 실행

파일 실행을 위해 exec 와 spawn을 이용할 수 있다. spawn 은 실행중인 정보(스트림)를 리턴하고 exec 는 실행 결과(버퍼)를 리턴한다.

```
1  var exec = require('child_process').exec;
2  var spawn = require('child_process').spawn;
3
4  exec('cat ./index.html', function(err, stdout, stderr){
5      if(err) trhow err;
6      console.log(stdout);
7  });
8
9  var p = spawn('netstat', ['http://www.facebook.com']]);
10
11 p.on('exit', function(){ /* todo after exit program */ });
12
13 p.stdout.on('data', function(data){
14     console.log(data);  // 정상 동작 stream 데이터 반환
15 });
16
17 p.stderr.on('data', function(data){
18     console.log(data);  // 오류 동작 stream 데이터 반환
19 });
```

유지 보수

개발 후 재 실행이나 잘못된 프로그램 종료후 실행을 위하여 매번 종료 후 재 실행하거나 수동으로 다시 실행하는 과정이 필요함. 하지만 관련 툴로 극복.

forever

재 실행을 위한 훌륭한 툴이기도 하지만, 무 정지 어플리케이션(프로그램이 오류로 종료되도 다시 시작되는 어플리케이션) 관리용으로도 사용 가능. 코드 변경후에 알아서 재 실행 하려면 -w 옵션 이용

```
$ sudo npm install -g forever
$ forever -w [실행할 어플리케이션]
```

nodemon

nodemon 을 이용하여 파일 변경시 재 시작되도록 구성할 수 있음

```
$ sudo npm install -g nodemon
$ nodemon [실행할 어플리케이션]
$ nodemon -e “.coffeel.jsl.ejs” [실행할 어플리케이션]
```

비동기 처리

circul(us)

Async

Callback Hell

File 이나 Socket 등이 연관된 기능을 사용할 때 Callback 으로 결과가 반환되는데, 이러한 프로세스 흐름에서 작업을 하다보면 중첩된 Callback function 으로 인하여 소스코드는 점차 복잡해지고 가독성이 떨어지며, 유지보수어 어려움을 겪게 됨

Async

복잡해 지는 프로세스흐름을 정리해 주는 라이브러리로 Control Flow 기능을 이용하여 Callback Hell 문제를 극복할 수 있음.

parallel : 병렬 처리한 결과값을 최종적으로 함께 받을 수 있음
waterfall : 앞 단계의 결과가 다음으로 전달됨, 오류 발생시 중지 함
series : 앞 단계의 결과가 다음으로 전달되며, 오류 발생시 중지

$ npm install async

Parallel

실행 할 1개 이상의 함수를 포함하는 Array, Object 로 구성되며, 설정한 함수는 callback 인자가 전달되어 이것을 실행 시켜줘야 다음 task 로 넘어감

```
1  var async = require('async');
2
3  async.parallel([
4      function(callback){
5          setTimeout(function(){
6              console.log('async step1');
7              callback(null, 'one');
8          }, 200);
9      },
10     function(callback){
11         setTimeout(function(){
12             console.log('async step2');
13             callback(null, 'two');
14         }, 200);
15     }
16 ], function(err, results){
17     console.log('["one","two"]');
18     console.log(results);
19 });
```

Waterfall

실행 될 1개 이상의 함수를 포함하는 배열 형식으로 구성되며, 설정한 함수에 callback 인자가 전달되며 이것이 실행되어야 다음 task 로 넘어 감

```
1  var async = require('async');
2
3  async.waterfall([
4      function(callback){
5          setTimeout(function(){
6              console.log('async step');
7              callback(null, 'one','two');
8          }, 200);
9      },
10     function(arg1, arg2, callback){
11         setTimeout(function(){
12             console.log(arguments);
13             callback(null, 'done');
14         }, 200);
15     }
16 ], function(err, results){
17     console.log('["done"]');
18     console.log(results);
19 });
```

Series

실행 할 1개 이상의 함수를 포함하는 Array, Ojbect 로 구성되며, 설정한 함수는 callback 인자가 전달되어 이것을 실행 시켜줘야 다음 task 로 넘어감

```
1  var async = require('async');
2
3  async.series([
4      one : function(callback){
5          setTimeout(function(){
6              console.log('async step');
7              callback(null, ['one','two']);
8          }, 200);
9      },
10     two : function(arg1, arg2, callback){
11         setTimeout(function(){
12             console.log(arguments);
13             callback(null, 'done');
14         }, 200);
15     }
16 ], function(err, results){
17     console.log('{ one : ["one","two"], two : "done"}');
18     console.log(results);
19 });
```

Express JS

circul(us)

ExpressJS

nodejs 를 위한 웹 애플리케이션 프레임워크로, 하나 혹인 여러 웹 페이지를 만들거나 하이브리드 웹 어플리케이션을 위한 기능들을 제공함

다양한 API

수많은 HTTP 유틸리티 메소드와 Connect 미들웨어를 원하는 대로 사용할 수 있습니다. 쉽고 간편하게 API를 생성할 수 있음

손쉬운 개발

Express는 웹 애플리케이션을 개발하는데에 있어서 필수적인 기능들을 제공하기 때문에, node.js의 어려운 기능들을 모두 알 필요가 없습니다.

ExpressJS 설치

ExpressJS 를 이용하여, 웹 어플리케이션을 손쉽게 개발할 수 있음.
nodeJS 의 express 모듈을 설치하여 사용 가능 함

express 설치

$ npm install express -g

```
1  var express = require('express');
2  var app = express();
3
4  // 기본 경로로 접속하면 hello world 를 출력함
5  app.get('/', function(req, res){
6      res.send('hello world');
7  });
8
9  // 서버를 80번 포트(http)로 구동 시킴
10 var server = app.listen(80, function(){
11     var host = server.address().address;
12     var port = server.address().port;
13
14     console.log('Express at http://$s:$s', host, port);
15 });
```

기본 라우팅

라우팅은 어플리케이션이 클라이언트의 요청에 어떻게 응답할지와 접근 경로(URIs)에 대한 정의를 나타낸다.

POST : 할일 목록에 항목 추가
GET : 현재 등록된 항목 전체를 표시하거나 특정 하목에 대한 세부 내용 표시
DELETE : 할일 목록에서 항목 삭제
PUT : 등록된 항목을 수정

기본 라우팅

HTTP 요청 메소드 GET, POST, PUT, DELETE 에 대해 기본적인 기능으로 간단하게 구현 가능

```
1  // respond with "Hello World!" on the homepage
2  app.get('/', function (req, res) {
3      res.send('Hello World!');
4  });
5
6  // accept POST request on the homepage
7  app.post('/', function (req, res) {
8      res.send('Got a POST request');
9  });
10
11 // accept PUT request at /user
12 app.put('/user', function (req, res) {
13     res.send('Got a PUT request at /user');
14 });
15
16 // accept DELETE request at /user
17 app.delete('/user', function (req, res) {
18     res.send('Got a DELETE request at /user');
19 });
```

기본 라우팅

HTTP 요청 메소드 GET, POST, PUT, DELETE 에 대해 기본적인 기능으로 간단하게 구현 가능

```
1  // a middleware sub-stack which handles GET requests to /user/:id
2  app.get('/user/:id', function (req, res, next) {
3      // if user id is 0, skip to the next route
4      if (req.params.id == 0) {
5          next('route');
6      } else {
7          // else pass  to the next middleware in this stack
8          next();
9      }
10 }, function (req, res, next) {
11     // render a regular page
12     res.render('regular');
13 });
14
15 // handler for /user/:id which renders a special page
16 app.get('/user/:id', function (req, res, next) {
17     res.render('special');
18 });
```

고급 라우팅

4.x 부터 새롭게 추가된 router 기능을 활용할 수 있음

```
1  var app = express();
2  var router = express.Router();
3
4  // a middleware with no mount path,
5  // gets executed for every request to the router
6  router.use(function (req, res, next) {
7      console.log('Time:', Date.now()); next();
8  });
9
10 // a middleware sub-stack shows request info
11 // for any type of HTTP request to /user/:id
12 router.use('/user/:id', function(req, res, next) {
13     console.log('Request URL:', req.originalUrl);
14     next();
15 }, function (req, res, next) {
16     console.log('Request Type:', req.method);
17     next();
18 });
```

고급 라우팅

router 를 이용하여 get, put, post 에 대한 요소를 활용할 수 있음

```
1  // mount the router on the app
2  app.use('/', router);
3
4  // a middleware sub-stack which handles GET requests to /user/:id
5  router.get('/user/:id', function (req, res, next) {
6      if (req.params.id == 0){
7          next('route');
8      } else {
9          else next();
10     }
11 }, function (req, res, next) {
12     res.render('regular');
13 });
14
15 // handler for /user/:id which renders a special page
16 router.get('/user/:id', function (req, res, next) {
17     console.log(req.params.id);
18     res.render('special');
19 });
```

오류 처리

이미지나 CSS, Javascript 와 같은고정된 파일은 static 기능을 이용하여 제공할 수 있음

```
1  // Error Handling
2  var bodyParser = require('body-parser');
3  var methodOverride = require('method-override');
4
5  app.use(bodyParser());
6  app.use(methodOverride());
7  app.use(function(err, req, res, next){
8      console.error(err.stack);
9      res.status(500).send('Something broken!');
10 });
```

오류 처리

오류 케이스에 따라서 일반 로그, 클라이언트 로그, 에러 핸들러 등으로 세부적으로 구현하여 활용할 수 있음

```
1  // 오류를 표시하는 일반적인 케이스
2  function logErrors(err, req, res, next){
3      console.error(err.stack);
4      next(err);
5  }
6
7  // 특정 오류가 있는 경우 검출함
8  function clientErrorHandler(err, req, res, next){
9      if(req.xhr){
10         res.status(500).send({ error : 'Something blew up!'});
11     } else {
12         next(err);
13     }
14 }
15
16 // 모든 오류를 catch 함
17 function errorHandler(err, req, res, next){
18     res.status(500);
19     res.render('error', { error: err });
20 }
```

미들웨어

Express 4 부터 미들웨어가 별도의 패키지로 존재하고 있으므로, NPM 을 이용하여 설치해 주어야 함

미들웨어 설치

```
$ npm install path -g
$ npm install serve-favicon -g
$ npm install morgan -g
$ npm install method-override -g
$ npm install express-session -g
$ npm install body-parser -g
$ npm install multer -g
$ npm install errorHandler -g
$ npm install jade -g
```

기본 구성

Express 4 부터 내부 라이브러리 보다는 외부 라이브러리를 이용하여 웹 어플리케이션을 구성할 수 있도록 변경됨. 다음과 같은 라이브러리를 활용

```
1  var express = require('express');
2  var routes = require('./routes');
3  var user = require('./routes/user');
4  var path = require('path');
5
6  var favicon = require('serve-favicon');
7  var logger = require('morgan');
8  var methodOverride = require('method-override');
9  var session = require('express-session');
10 var bodyParser = require('body-parser');
11 var multer = require('multer');
12 var errorHandler = require('errorHandler');
13
14 var app = express();
```

기본 구성

라이브러리 선언 이후 해당 메소드 이용 및 호출 부를 작성 함

```
16  app.set('port', process.env.PORT || 80);
17  app.set('views', path.join(__dirname, 'views'));
18  app.set('view engine', 'jade');
19  app.use(favicon(__dirname + '/public/favicon.ico'));
20  app.use(logger('dev'));
21  app.use(methodOverride());
22  app.use(sesson({resave : true,saveUninitialized : true,secret:'uwotm8'}));
23  app.use(multer());
24  app.use(express.static(path.join(__dirname, 'public')));
25
26  app.get('/', routes.index);
27  app.get('/users', user.list);
28
29  if('development' == app.get('env')){
30      app.use(errorHandler());
31  }
32
33  app.listen(app.get('port'), function(){
34      console.log('Express server lisening on the port ' + app.get('port'));
35  });
```

템플릿 엔진 ELS

Embeded JavaScript 템플릿 엔진은 매우 간단하게 동작하며, JSP 의 템플릿 Smarty(PHP), ERB(Ruby) 등의 템플릿 엔진과 유사하게 동작한다.

```
1  // express 호출 시에 view engine 을 선택 해야 함
2  app.set('view engine', 'ejs');
3
4  app.get('/', function(req, res){
5      // 템플릿 엔진을 통해 렌더링된 결과가 표시 됨
6      res.render('index', { title : 'Hey,', message : 'Hello there!'});
7  });
```

HTML 안에 데이터를 담을 EJS 태그를 넣어 페이지를 구성할 수 있으며, if 또는 for 구문등을 작업하는 원시 자바스크립트 로직을 수행할 수 있음

```
1  <html>
2      <head>
3          <title><%= title %></title>
4      </head>
5      <body>
6          <%= message %>
7      </body>
8  </html>
```

템플릿 엔진 Jade

html 을 그대로 활용할 수 있지만, 템플릿 엔진인 jade 를 이용하여 화면 내용을 간단하게 표현하고, 동적인 데이터를 포함할 수 있음

```
1  // express 호출 시에 view engine 을 선택 해야 함
2  app.set('view engine', 'jade');
3
4  app.get('/', function(req, res){
5      // 템플릿 엔진을 통해 렌더링된 결과가 표시 됨
6      res.render('index', { title : 'Hey,', message : 'Hello there!'});
7  });
```

jade 문법으로 작성된 웹 화면이 데이터 값과 함께 html 로 변환되어 사용자의 웹 페이지로 제공된다

```
1  html
2      head
3          title!= title
4      body
5          h1 != message
```

응답 타입

요청에 대한 응답 으로 다음과 같은 함수를 사용할 수 있음.

메소드 명	설명
res.download()	파일을 다운로드 하는 동작을 수행 시킴
res.end()	응답 프로세스를 종료함
res.json()	JSON 응답을 보냄
res.jsonp()	JSONP를 지원하는 JSON 응답을 보냄
res.redirect()	요청을 Redirect 함
res.render()	View Template 를 렌더링 함
res.send()	다양한 타입의 응답을 보냄
res.sendFile()	Octet Sream 의 파일을 보냄
res.sendStatus()	응답 상태 코드를 설정하고, 응답 바디에 문자열로 표시하여 보냄

Express JS 자동 프로젝트

자동 생성 툴을 활용하면, 샘플 프로젝트를 자동으로 생성해 줌. 샘플 프로젝트를 이용하여, 개발을 수행할 수 있음

$ npm install express-generator -g

샘플 프로젝트 생성

$ express myapp

Socket.IO

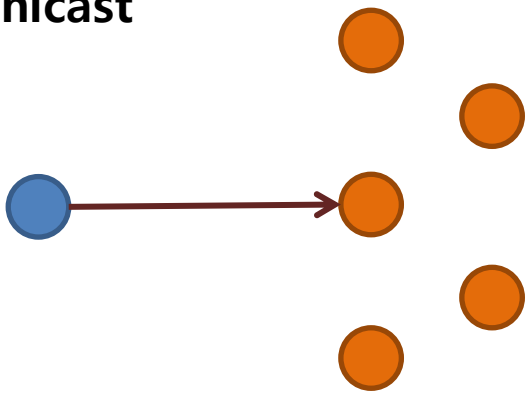
SocketIO

웹을 통하여 Realtime 서비스를 구현하기 위해서는 HTML5 의 WebSocket 서비스를 이용할 수 있으나, 브라우저나 시스템 상황에 따라서 이용하지 못하는 경우가 발생할 수 있다.
WebSocket 과 유사한 기능을 제공하는 서비스로는 FlashSocket, AJAX Long Polling, AJAX Multipart Straming, iframe, JSONP Polling 등의 기술이 있는데, socket.io 는 이러한 기술들을 하나의 API 로 추상화 하여, 브라우저와 웹 서버의 종류와 버전을 파악하여 가장 적합한 기술을 선택하여 사용할 수 있게 해 준다.

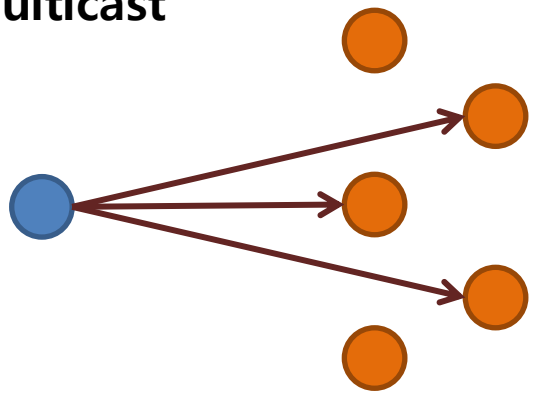
socket.io

통신 종류

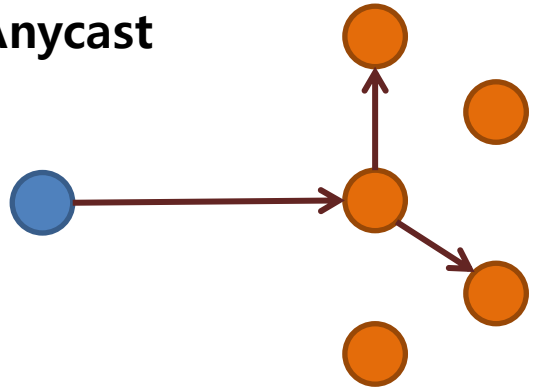
Unicast



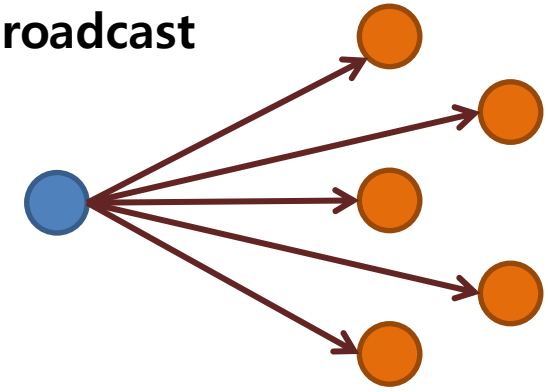
Multicast



Anycast



Broadcast



socket.io 서버

socket.io 를 호출하며 포트를 입력하면, 해당 포트로 socket.io 서버가 생성된다.

```
1  var io = require('socket.io')(80);
2
3  io.on('connection', function(socket){
4      // 모든 접속 클라이언트에 전달 함
5      io.emit('this', { will : 'be received by eveyone'});
6
7      socket.on('broadcast', function(data){
8          // 나를 제외한 모든 접속 클라이언트에 전달함
9          socket.broadcast.emit('send',{ except : 'me'});
10     });
11
12     socket.on('unicast', function(data){
13         // 나한테 메세지를 보낸 사람한테만 전달함
14         socket.emit('send', { to : 'me'});
15     });
16
17     socket.on('private', function(from, msg){
18         console.log('Received message by', from, 'saying', msg );
19     });
20 });
```

socket.io 클라이언트

웹 파일에서는 socket.io 클라이언트 라이브러리를 호출하여 사용한다.
nodejs 서버를 통해 제공되는 html 파일에서 손쉽게 구현이 가능하다.

```
<script src='/socket.io/socket.io.js'></script>
<script>
    var socket = io();

    socket.on('this', function(data){
        console.log('Received from server, event "this"');
        console.log(data);
    });

    socket.on('send', function(data){
        console.log('Received from server, event "send"');
        console.log(data);
    });

    socket.emit('broadcast', { send : 'to server'});
    socket.emit('unicast', { send : 'to server'});
    socket.emit('private')
</script>
```

ExpressJS + Socket.IO 서버

Socket.IO 단독으로 사용할 수도 있지만, ExpressJS 로 생성된 서버를 이용하여, Socket.IO 를 결합하여 구성할 수 있음.

```
33  // ExpressJS 기본 템플릿과 동일하나, server 로 리턴 받아 활용
34  var server = app.listen(app.get('port'), function(){
35      console.log('Express server lisening on the port ' + app.get('port'));
36  });
37
38  var io = require('socket.io').listen(server);
39
40  io.sockets.on('connection', function(socket){
41      socket.emit('news', { hello : 'world'});
42
43      // 사용자 지정 event
44      socket.on('user event', function(data){
45          console.log(data);
46      });
47
48      // disconnect 발생시 호출됨
49      socket.on('disconnect', function(){
50          // to do
51      });
52  });
```

ExpressJS + Socket.IO 클라이언트

웹 파일에서는 socket.io 클라이언트 라이브러리를 호출하여 사용한다.
socket.io 가 설치된 nodejs 서버를 통해 제공되는 html 파일에서는
socket.io.js 파일을 참조할 수 있도록 구성되어 진다.

```
1  <script src='/socket.io/socket.io.js'></script>
2  <script>
3      var socket = io.connect();
4
5      socket.on('news', function(data){
6          console.log(data);
7          socket.emit('user event', {my : 'data'});
8      });
9  </script>
```

namespace 구성 - Server

메세지 구성을 나누어서 하고자 하는 경우, namesace 를 이용하여 구성하면 각 namespace 영역에서만 데이터를 주고받을 수 있다.

```
1  var io = require('socket.io')(80);
2
3  var chat = io.of('/chat').on('connection', function(socket){
4      // chat 채널의 모든 접속 클라이언트에 전달 함
5      chat.emit('this', { will : 'be received by eveyone in chat'});
6
7      // chat 채널 내에서 접속한 자신에게만 전달 함
8      socket.emit('this', { will : 'be received by one in chat'});
9  });
10
11 var news = io.of('/news');
12
13 news.on('connection', function(socket){
14     // news 채널의 모든 접속 클라이언트에 전달 함
15     news.emit('this', { will : 'be received by eveyone in news'});
16
17     // news 채널 내에서 접속한 자신에게만 전달 함
18     socket.emit('this', { will : 'be received by one in news'});
19 });
```

namespace 구성 - Client

클라이언트 측에도 서버에 구성한 namespace 에 따른 접속 경로를 분화하여 구성할 수 있다.

```
<script src='/socket.io/socket.io.js'></script>
<script>
    // server에 chat이라는 이름으로 namespace 구성
    var chat = io.connect('http://localhost/chat');
    // server에 news이라는 이름으로 namespace 구성
    var news = io.connect('http://localhost/news');

    chat.on('connect', function(){
        chat.emit('hi');
    });

    news.on('news', function(){
        news.emit('woot');
    });
</script>
```

room 구성

채팅방과 같이 room 을 구성할 수 있다. join 과 leave 함수를 이용하며, 고유한 chatting room 이름을 지정하여 해당 이름으로 사용자가 접속 가능하다.

```
1  var io = require('socket.io')(80);
2
3  io.on('connection', function(socket){
4      // 1. test room 이 없으면 생성하고 들어간다.
5      // 2. test room 이 있으면 그대로 들어간다.
6      socket.join('test room');
7
8      // test room 에 존재하는 모든 사람에게 메세지를 보냄
9      io.to('test room').emit('some event');
10
11     socket.on('say to someone', function(id, data){
12         // 내가 포함되어 있는 방의 사람들에게 메세지 보냄
13         socket.broadcast.to(id).emit('my msg', data);
14     });
15
16     socket.on('leave room', function(){
17         // test room 에서 나옴
18         socket.leave('test room');
19     });
20 });
```

Team Circulus Park Jonggun, Lee Yunjai, Ki Yeonah, Lee Jonghee

Since 2013

circul(us)

Expand your dimension

Circulus Official www.circul.us

Circulus Group group.circul.us