1. Middleware

1)미들웨어란?

-분산 컴퓨팅 환경에서

-서로 다른 기종의 하드웨어나 프로토콜, 통신 환경 등을 연결하여 원만한 통신이 이루어질 수 있게 해주는 소프트웨어

-일종의 중개자

-클라이언트의 Request를 제어하는 함수들의 집합

-개발자와 기간 시스템 간에 존재하는 소프트웨어

-웹 서버에서 정적 파일을 제공하기 위해 필요한 모든 기능들을 직접 제공하는 대신, 미들웨어를 사용해서 대부분 처리 가능

-미들웨어에서 지루한 작업들을 대부분 감당해주기 때문에 자신이 필요하거나 요구사항과 관련된 항목에 집중할 수 있게 해줌.

-권한관리 구성요소, 프록시, 라우터, 쿠키 및 세션 관리, 기타 웹 기술에 필요한 항목들 제공

2)Node.js의 HTTP Server

```
var http = require('http');

http.createServer(function(request, response){
    //Main page
    if(request.url == '/'){
        response.writeHead(200, {'Content-Type' : 'text/html'});
        response.end('Welcome to the Main page...');
    }
    //About page
    else if(request.url == '/about'){
        response.writeHead(200, {'Content-Type' : 'text/html'});
        response.end('Welcome to the about page!! I will show you something...');
    }
    //File Not found page
    else {
        response.writeHead(404, {'Content-Type':'text/plain'});
        response.end('Ooops, 404 error! File Not Found.');
    }
}).listen(8080, 'Server running on #8080');
```

3)단점

-만일 구축하려는 웹 어플리케이션이 10개의 페이지만 갖고 있어도, 위의 예에서 처럼 10개의 if 문이 필요하다.

-동작은 하겠지만, 유지보수하기에는 지저분한 코드이다.

2. Connect middleware

1)https://stephensugden.com/middleware_guide/

2)Is an extensible HTTP server framework for node, providing high performance "plugins" known as middleware.

3)Node.js의 기본모듈인 HTTP 모듈의 Server, ServerRequest, ServerReponse 객체를 포장하여 유용한 기능들을 제공하는 모듈

4)Wraps the Server, ServerRequest, and ServerResponse objects of node.js' standard http module, giving them a few nice extra features, one of which is allowing the Server object to use a stack of middleware.

5)Install

```
$ npm install connect
/WebHome
    |
    |--connect@3.6.1
    ...
```

or

In Eclipse,

```
   package.json에 모듈 의존성 기록
      --추후 구축한 프로젝트에서 어떤 모듈을 사용하고 있는지 쉽게 확인 가능하고
      --프로젝트간 독립성을 유지하면서
      --다른 프로젝트 혹은 글로벌하게 설치된 모듈에 영향을 받지 않게 하려함
   <package.json>
      "dependencies" : {
            "connect" : "*"  <--현존하는 버전 중 최신 릴리즈 버전을 가져오라는 의미
      },
   -package.json 파일 > 오른 마우스 > Run As > npm install

6)Create an app
   var app = connect();

7)Use middleware
   -미들웨어 사용 개수에는 제한이 없으며, connect에 내장된 것이든 서드파티에서 제공된 것이든
   use()을 사용해서 추가하기만 하면 된다.
   -서버가 생성되면, 미리 명시한 미들웨어 함수들을 호출
   -각 미들웨어는 next()를 통해 다음 계층으로 클라이언트 요청을 보낼지 말지를 결정
      app.use(function(request, response, next){
         console.log('In comes a ' + request.method + ' to ' + request.url);
         next();
      }

8)Create a server from the app
   var server = app.listen(port);
   var server = http.createServer(app);

9)Lab
   <connect.js>
      var connect = require('connect');

      function logger(req, res, next){
            console.log('%s %s', req.method, req.url);
            next();
      };
      function hello(req, res){
                  res.writeHead(200, {'Content-Type':'text/html'});
                  res.end("<h1>Hello Connect</h1>");
      };

      connect()
            .use(logger)
            .use(hello)
            .listen(3000);
      ------------------------------------------------
      GET /
      GET /favicon.ico

   <connect1.js>
      var connect = require('connect');
      var http = require('http');
      var app = connect();

      app.use(function(request, response, next){
         console.log('In comes a ' + request.method + ' to ' + request.url);
         next();
      });
      app.use(function(request, response){
            response.writeHead(200, {'Content-Type':'text/html'});
            response.end("<h1>Hello Connect</h1>");
```

```
});

http.createServer(app).listen(8124);
```

10)미들웨어 순서의 중요성

-어플리케이션에서 미들웨어의 순서를 정해서 동작에 크게 영향을 미치는 경우

--next()를 넣지 않아 남은 미들웨어의 실행을 멈춘다.

--강력한 미들웨어 순서를 사용해 편의성을 얻는다.

--미들웨어를 투입해서 인증을 수행한다.

-미들웨어가 명시적으로 next()를 수행하지 않을 때

```
var connect = require('connect');

function logger(req, res, next){
      console.log('%s %s', req.method, req.url);
      next();
};
function hello(req, res){
      res.writeHead(200, {'Content-Type':'text/html'});
      res.end("<h1>Hello Connect</h1>");
};

connect()
      .use(hello)   //hello 함수에서 next()를 사용하지 않아 logger 함수를 호출하지 않음.
      .use(logger)
      .listen(3000);
```

-미들웨어 순서를 사용해서 인증하기

--인증 같은 경우에는 미들웨어 순서를 정하는 것이 좋다.

--인증은 거의 모든 종류의 어플리케이션과 연관이 있다.

--사용자는 로그인을 해야 하고 개발자는 로그인하지 않은 사람이 컨텐츠에 접근하는 것을 막아야 한다.

--다음 코드를 보자

```
var connect = require('connect');
connect()
   .use(logger);
   .use(restrictFileAccess)
   .use(serveStaticFiles)
   .use(hello);
```

--권한이 있는 사용자에게만 파일에 접근할 수 있는 restrictFileAccess라는 미들웨어 컴포넌트를 만든다고 하자.

--권한이 있는 사용자는 다음 미들웨어 컴포넌트로 이동할 수 있지만, 권한이 없으면 next()를 호출하지 않는다.

11)미들웨어와 서버 마운팅

-connect에는 미들웨어나 전체 어플리케이션에 대해 기본 경로를 지정할 수 있는 간단하지만 강력한 도구인 마운팅이라는 개념이 있다.

-마운팅은 미들웨어를 루트 레벨에서('/'를 기반으로 하는 req.url) 사용하듯이 작성하게 해주고 코드를 변경하지 않고도 어떤 기본 경로에서도 사용할 수 있게 해준다.

```
var connect = require('connect');
connect()
   .use(logger)
   .use('/admin', restrict) //use()의 첫번째 인자로 문자가 오면 커넥트는 기본 경로 url이 맞을 때만 미들웨어를 실행함.
   .use('/admin', admin)
   .use(hello)
   .listen(3000);
```

12)인증 미들웨어

-어플리케이션은 /admin으로 시작하는 요청이 왔을 때 인증 컴포넌트가 시작.

-아래는 간단한 Base64 방식의 인증체계이다.

-한번 인증서가 미들웨어 컴포넌트를 지나가면 사용자 이름과 암호가 정확한지 점검한다.

-인증이 되면 컴포넌트는 next()를 실행하게 되고 요청이 정상적으로 진행되고 인증되지 않으면 에러가 발생한다.

```
function restrict(req, res, next){
    var authorization = req.headers.authorization;
    if(!authorization) return next(new Error('Unauthorized'));

    var parts = authorization.split(' ');
    var scheme = parts[0];
    var auth =  new Buffer(parts[1], 'base64').toString().split(':');
    var user = auth[0];
    var pass = auth[1];

    next();
}
```

13)관리영역을 보여주는 미들웨어 컴포넌트

```
function admin(req, res, next){
    switch(req.url){
        case '/':
            res.end('try /users');
            break;
        case '/users' :
            res.setHeader('Content-Type', 'application/json');
            res.write(JSON.stringify(['tobi', 'loki', 'jane']));
            res.end('\n');
            break;
    }
}
```

14)Lab

<connect2.js>

```
var connect = require('connect');
connect()
        .use(logger)
        .use('/admin', restrict)
        .use('/admin', admin)
        .use(hello)
        .listen(3000);

function logger(req, res, next){
        console.log('%s %s', req.method, req.url);
        next();
};
function restrict(req, res, next){
        var authorization = req.headers.authorization;
        if(!authorization) return next(new Error('Unauthorized'));

     var parts = authorization.split(' ');
        var scheme = parts[0];
        var auth =  new Buffer(parts[1], 'base64').toString().split(':');
        var user = auth[0];
        var pass = auth[1];

        next();
}
```

```
function admin(req, res, next){
      switch(req.url){
            case '/':
                  res.end('try /users');
                  break;
            case '/users' :
                  res.setHeader('Content-Type', 'application/json');
                  res.write(JSON.stringify(['tobi', 'loki', 'jane']));
                  res.end('\n');
                  break;
      }
}
function hello(req, res){
      res.writeHead(200, {'Content-Type':'text/plain'});
      res.end("Hello Connect\n");
};
-------------------------------------------------------------
서버 실행은 Ubuntu Server에서 할 것
실행조건
   1)이미 vsftpd 서비스가 설치되고 실행되어 있을 것
   2)ftpuser/ftpuser의 $HOME이 /WebHome이어야 함.
   2)Filezilla를 통해 /WebHome/connect2.js를 전송할 것
   3)ftpuser@ubuntu1604:~$ node connect2.js

클라이언트는 Putty.exe로 할 것
   instructor@ubuntu1604:~$ curl http://localhost:3000
   Hello Connect
   instructor@ubuntu1604:~$ curl http://localhost:3000/foo
   Hello Connect

   instructor@ubuntu1604:~$ curl http://localhost:3000/admin/users
   <!DOCTYPE html>
   <html lang="en">
   <head>
   <meta charset="utf-8">
   <title>Error</title>
   </head>
   <body>
   <pre>Error: Unauthorized<br>    at restrict
   (/WebHome/connect.js:15:36)<br>    at call
   (/WebHome/node_modules/connect/index.js:239:7)<br>    at next
   (/WebHome/node_modules/connect/index.js:183:5)<br>    at
   logger (/WebHome/connect.js:11:5)<br>    at call
   (/WebHome/node_modules/connect/index.js:239:7)<br>    at next
   (/WebHome/node_modules/connect/index.js:183:5)<br>    at
   Function.handle (/WebHome/node_modules/connect/index.js:186:3)<br>
      at Server.app
   (/WebHome/node_modules/connect/index.js:51:37)<br>    at
   emitTwo (events.js:106:13)<br>    at Server.emit
   (events.js:191:7)</pre>
   </body>
   </html>

   instructor@ubuntu1604:~$ curl --user jane:ferret
   http://localhost:3000/admin/users
   ["tobi","loki","jane"]

   instructor@ubuntu1604:~$ curl --user tobi:ferret
   http://localhost:3000/admin/users
```

```
273             ["tobi","loki","jane"]
274
275     15)Middlewares
276        -https://www.npmjs.com/package/connect#middleware
277        -These middleware and libraries are officially supported by the Connect/Express team
278           --body-parser : previously <bodyParser>, <json> and <urlencoded>
279           --compression : previously <compress>
280           --connect-timeout : previously <timeout>
281           --cookie-parser : previously <cookieParser>
282           --cookie-session : previously <cookieSession>
283           --csurf : previously <csrf>
284           --errorhandler : previously <error-handler>
285           --express-session : previously <session>
286           --method-override
287           --morgan : previously <logger>
288           --response-time
289           --serve-favicon : previously <favicon>
290           --serve-index : previously <directory>
291           --serve-static : previously <static>
292           --vhost
293
294     11)serve-static
295        -https://www.npmjs.com/package/serve-static
296        -정적 파일 서버 구축
297        -Install
298           $ npm install serve-static
299                    |
300                    |------serve-static@1.12.2
301
302        -Lab : connect-static.js
303           var connect = require('connect');
304           var serveStatic = require('serve-static');
305
306           var app = connect();
307           app.use(serveStatic(__dirname, {'index' : ['index.html', 'index.htm']}));
308           app.listen(3000);
309
310     12)morgan
311        -https://www.npmjs.com/package/morgan
312        -들어오는 요청들에 대한 로그를 스트림(기본적으로 stdout으로 설정)에 기록
313        -Install
314              $ npm install morgan
315                  |
316                     |-----morgan@1.8.1
317
318        -Predefined Formats
319           --combined : Standard Apache combined log output.
320              :remote-addr - :remote-user [:date[clf]] ":method :url HTTP/:http-version"
                 :status :res[content-length] ":referrer" ":user-agent"
321           --common : Standard Apache common log output.
322              :remote-addr - :remote-user [:date[clf]] ":method :url HTTP/:http-version"
                 :status :res[content-length]
323           --dev : Concise output colored by response status for development use.
324              :method :url :status :response-time ms - :res[content-length]
325           --short : Shorter than default, also including response time.
326              :remote-addr :remote-user :method :url HTTP/:http-version :status
                 :res[content-length] - :response-time ms
327           --tiny : The minimal output.
328              :method :url :status :res[content-length] - :response-time ms
329        -Tokens
```

--:date[format]
The current date and time in UTC. The available formats are:
---[clf] for the common log format ("10/Oct/2000:13:55:36 +0000")
---[iso] for the common ISO 8601 date time format (2000-10-10T13:55:36.000Z)
---[web] for the common RFC 1123 date time format (Tue, 10 Oct 2000 13:55:36 GMT)
If no format is given, then the default is [web].
--:http-version
The HTTP version of the request.
--:method
The HTTP method of the request.
--:referrer
The Referrer header of the request. This will use the standard mis-spelled Referer header if exists, otherwise Referrer.
--:remote-addr
The remote address of the request. This will use req.ip, otherwise the standard req.connection.remoteAddress value (socket address).
--:remote-user
The user authenticated as part of Basic auth for the request.
--:req[header]
The given header of the request.
--:res[header]
The given header of the response.
--:response-time[digits]
The time between the request coming into morgan and when the response headers are written, in milliseconds.
The digits argument is a number that specifies the number of digits to include on the number, defaulting to 3, which provides microsecond precision.
--:status
The status code of the response.
If the request/response cycle completes before a response was sent to the client (for example, the TCP socket closed prematurely by a client aborting the request), then the status will be empty (displayed as "-" in the log).
--:url
The URL of the request. This will use req.originalUrl if exists, otherwise req.url.
--:user-agent
The contents of the User-Agent header of the request.

13)Lab : connect-morgan.js

```
var connect = require('connect');
var morgan = require('morgan');
var app = connect();

app.use(morgan('combined'));
app.use('/', function (req, res) {
   res.end('hello, world!')
});
app.listen(3000);
```

14)Lab : connect-morgan1.js

```
var connect = require('connect');
var fs = require('fs');
var morgan = require('morgan');
var path = require('path');

var app = connect();

// create a write stream (in append mode)
var accessLogStream = fs.createWriteStream(path.join(__dirname, 'access.log'),
{flags: 'a'});
```

```
// setup the logger
app.use(morgan('combined', {stream: accessLogStream}));

app.use('/', function (req, res) {
   res.end('hello, world!');
});
app.listen(3000);
```

3. Router

1)한 소스로부터 무엇인가를 받아서 다른 것으로 전달한다.

2)보통 데이터 패킷이 전달되지만, 어플리케이션 수준에서는 리소스 요청이 될 수 있다.

3)http://yourplace.org/article/your-title 보다는 http://yourplace.org/node/174를 더 선호할 것이다.

4)다른 형태는 http://yourplace.org/user/3 이런 형식도 있다.

5)Node.js에서 라우터를 사용하는 가장 큰 이유는 URL로부터 필요한 정보를 추출하기 위해

6)추출된 정보를 프로세스에 전달하여 올바른 프로세스가 실행되게 하는 것이 목적

7)Crossroads router module

-http://millermedeiros.github.io/crossroads.js/

-Is a powerful and flexible routing system.

8)Install

```
$ npm install crossroads
      |
      |----crossroads@0.12.2
```

9)addRoute(pattern, [handler], [priority]) method

-Creates a new route pattern listener and add it to crossroads routes collection.

-라우팅 경로는 중괄호({})와 라우트 처리기 함수에 전달될 명명된 변수를 가진 정규식을 사용하여 정의한다.

{type}/{id}와 node/{id}

--다음에 모두 대응된다.

http://something.org/node/174

--하지만 첫번째 패턴에서는 type 매개변수가 라우팅 처리기에 전달되지만, 두번째는 전달되지 않는다.

-선택적인 부분을 표시하기 위해서는 콜론(:)을 사용할 수도 있다.

category/:type:/:id:

--이것은 다음에 대응된다.

category/

category/tech/

category/histroy/143

10)Lab : addroute.js

```
var crossroads = require('crossroads');
var http = require('http');

crossroads.addRoute('/category/{type}/:pub:/:id:', function(type,pub, id){
      if(!id && !pub){
            console.log('Accessing all entries of category ' + type);
            return ;
      }else if(!id){
            console.log('Accessing all entries of category ' + type + ' and pub ' + pub);
            return ;
      }else{
            console.log('Accessing item ' + id + ' of pub ' + pub + ' of category ' +
            type);
      }
});

http.createServer(function(req, res){
```

```
        crossroads.parse(req.url);
        res.end(' and that\'s all\n');
    }).listen(8124);
    ------------------------------------------------
    http://localhost:8124/category/history -->
    Accessing all entries of category history
    http://localhost:8124/category/history/journal -->
    Accessing all entries of category history and pub journal
    http://localhost:8124/category/history/journal/174 -->
    Accessing item 174 of pub journal of category history


4. Proxy
  1)여러 개의 다른 위치에서 요청된 내역을 어떠한 이유(캐시, 보안, 심지어 원래의 요청자를 감추려는 목적)때문에 하나의 서버로 라우팅하는 방법이다.
  2)종류
    -정방향 프록시(forward proxy)
      --공개적으로 접근이 가능한 프록시는 실제 위치대신 다른 위치로부터 요청이 보내진 것으로 보이게 해서 특정 웹 컨텐츠에 대한 어떤 이들의 접근을 제한하는 데 이용
    -역방향 프록시(reverse proxy)
      --요청이 서버로 전송되는 방법을 제어하는 방법
      --5대의 서버가 있는데, 이 중 4대에는 사람들이 접속하지 않았으면 한다고 가정하자.
      --이 경우 모든 트래픽을 5번째 서버로 보낸 다음, 이 서버가 대신 다른 서버들에게 요청을 보내게 된다.
      --이 프록시는 로드 밸런싱(load balancing) 용도와 요청을 캐시하여 시스템의 전반적인 성능을 높이는 목적으로 사용
    -특정 포트로 들어오는 요청들을 수신 대기하면서 다른 포트를 수신 대기 중인 웹 서버로 중계하는 것

  3)node-http-proxy module
    -Node.js에서 가장 인기있는 프록시 모듈
    -https://github.com/nodejitsu/node-http-proxy
    -Is an HTTP programmable proxying library that supports websockets.
    -It is suitable for implementing components such as reverse proxies and load balancers.

  4)Install
    $ npm install http-proxy
          |
          |----http-proxy@1.16.2

  5)Lab : proxy.js
    var http = require('http');
    var httpProxy = require('http-proxy');

    httpProxy.createProxyServer({target:'http://localhost:9000'}).listen(8124);

    http.createServer(function (req, res) {
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.write('request successfully proxied!' + '\n' + JSON.stringify(req.headers, true, 2));
      res.end();
    }).listen(9000);
```