1. 소개
  1)http://expressjs.com/
  2)초경량의 유연한 Node.js 웹 어플리케이션 프레임워크
  3)Developed by TJ Holowaychuk
  4)Initial release November 16, 2010.
  5)Fast, unopinionated, minimalist web framework for Node.js
  6)Single, Multiple page and Hybrid Web Application을 작성하기 위한 강력한 기능들을 제공
  7)강하고 유연한 라우터 기능 제공
  8)Connect가 HTTP Server 모듈을 확장했듯이, Express.js는 Connect 미들웨어를 확장한 것
  9)Template Engine
  10)TJ Holowaychuk은 express 소유권을 2014년 7월 node startup 회사인 StrongLoop에 넘겼다.
  11)StrongLoop는 현재 계속 express를 오픈소스를 유지하면서 개발하고 있다.
  12)마이스페이스, Apiary.io, 고스트, 페르소나, 모질라 기반 로그인 시스템 등에서 사용중.
  13)Paypal의 오픈소스인 크라켄JS(KrakenJS, http://krakenjs.com) 프레임워크 기반으로 사용되었다.

2. Install
    $ npm install express
        |
        |---express@4.15.2

  1)in Ubuntu with ENIDE
    -새 프로젝트 생성
      --[New] > [Node.js Project]
      --Project name: 프로젝트 이름
      --Template to use: [none/empty] 선택 > Finish
    -package.json 생성
      --새 프로젝트 우클릭 > [New] > [Other...]
      --[JavaScript] > [npm Init] > Next
      --Uncheck [Use default configuration]
      --Version : 1.0.0
      --Main : app.js
      --Author : instructor
      --Finish
    -package.json에서
      "license": "ISC", 에 아래 추가
      "dependencies":{
        "express" : "*"
      }
    -Save
    -프로젝트 우클릭 > [StartExplorer] > [Start Shell Here]
    -터미널에서 아래 명령 수행
      #npm install express --save
    -터미널 창 닫고 프로젝트 refresh하면 express 버전이 최신 버전으로 변경된다.

3. 서버 생성 및 시작
```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Hello World!');
```

```
  })

  app.listen(3000, function () {
    console.log('Example app listening on port 3000!');
  })


4. express 의 구성요소
  1)express server는 라우터, 라우트, 미들웨어라는 3 가지 구성 요소로 나눌 수 있다.
  2)Router
    -모든 웹 서버의 핵심은 튼튼한 요청 라우팅이다.
    -일반적인 서버-클라이언트 생명 주기
      --클라이언트가 자원을 요청하고
      --서버는 자원의 위치를 찾으려 시도하고,
      --자원을 찾으면 클라이언트가 기대하는 방식으로 응답한다.
    -즉, 요청을 처리하고 라우트하는 구조적인 방법이 없다면 웹 서버는 거의 무용지물이 될 것이다.
    -문제는 http모듈에는 자원 라우팅이라는 개념이 없다는 것이다.
    -개별 라우트를 위한 정규표현식을 작성해야만 했다.
    -요청이 http 서버에 들어오면 req.path를 점검해서 if else 블록 내부에서 다양한 정규표현식과 비교해야 했다.
    -또한 CSS나 HTML 정적 파일을 찾기 위한 포괄적인 논리를 추가해야 했다.
    -반면 express는 사용하기 쉽고 표현력이 강한 라우팅 인터페이스를 제공한다.

  3)Route 탐색
    -라우트를 라우터에 추가하는 순서는 아주 중요하다.
    -요청이 웹 서버로 들어오면 URI는 라우팅 테이블을 거치게 된다.
    -이 테이블에서 처음으로 일치하는 곳이 수행될 코드이다.
    -라우트 테이블 아래 두 번째로 일치하는 라우트는 내부적으로 다시 라우팅하지 하지 않는 이상 결코 수행되지 않을 것이다.
    -즉, 두번째 라우트가 기술적으로 더 명확하더라도 항상 처음에 일치한 항목만 수행된다.

  4)정적 파일
    -웹 서버에는 정적 파일을 서비스하는 단순한 수단이 존재해야 한다.
    -평균적인 웹사이트에는 웹서버에서 서비스되는 수백 개가 넘는 정적 파일이 존재한다.
    -특정 디렉토리에 들어있는 여러 파일을 라우팅 테이블에 개별적으로 추가할 필요없이 라우터에 위치를 알려주어 요청이 들어올 때 파일 시스템에서 검색할 수 있는 수단이 반드시 필요하다.
    -express에서는 express.static(directory)로 정적 파일을 서비스할 수 있다.
    -이는 기본적으로 directory에 속한 파일을 찾기 위한 라우팅 테이블에 규칙을 생성하며, 규칙을 발견하면 파일을 서비스한다.
    -규칙을 발견하지 못하면 라우팅 테이블을 계속해서 따라 내려가며 또 다른 라우트에 일치하는 파일이 있는지 탐색한다.


5. HTTP 모듈 연동
    var http = require('http');
    var express =  require('express');
    var app = express();
    app.set('port', process.env.PORT || 3000);
    http.createServer(app).listen(app.get('port'), function(){
      console.log('Running : ', app.get('port'));
    });
```

6. 주요 메소드
  1)set(name, value)
    -서버 설정을 위한 속성 지정.
    -이 메소드로 지정한 속성은 get() 메소드로 확인 가능

  2)get(name)
    -서버 설정을 위해 지정한 속성을 확인한다.

  3)use([path,] function[,function...])
    -미들웨어 함수를 사용한다.

  4)get([path,] function)
    -특정 패스로 요청된 정보를 처리한다.

  5)Lab : server.js
```
var http = require('http');
var express =  require('express');
var app = express();
app.set('port', process.env.PORT || 3000);
app.use(function(req, res, next){
  console.log("수신완료");
  res.writeHead(200, {'Content-Type':'text/html;charset=utf8'});
  res.end('<h1>Express 서버에서 응답</h1>');
});

http.createServer(app).listen(app.get('port'), function(){
  console.log('Running : ', app.get('port'));
});
```

7. Express Middlewares
  1)express 어플리케이션은 다수의 미들웨어의 조합으로 동작한다.
  2)각 미들웨어는 요청과 응답 개체를 파라미터로 하는 함수로 작성
    -function(req, res, next) 형태의 자바스크립트 함수이다.
    -req에는 accepts(), get(), is()와 같은 새로운 함수가 있다.
    -또한 path, host, xhr, cookies와 같은 여러 새로운 속성을 외부에 공개한다.
    -res 객체는 응답객체(http.ServerResponse) 이다.
    -cookie(), redirect(), send()와 많은 다른 유용한 응답 함수를 추가했다.
    -마지막 인자인 next는 express 프레임워크가 제공하는 콜백함수이다.
    -이것은 비동기식으로 활동하는 미들웨어 함수를 위해 유용하다.
    -이것은 이 미들웨어 함수의 실행이 끝났음을 express에 알리기 위해 코드가 완료될 때 호출하기만 하면 된다.
    -미들웨어가 res.send()로 응답을 전송할 책임이 있다면 next를 호출할 필요가 없다.
    -다른 종료되지 않는 모든 미들웨어 함수를 위해 next를 반드시 한 번 호출해야 한다.

  3)이 함수의 파라미터는 IncomingMessage 클래스인 요청 객체(request)와 클라이언트에게 보내는 ServerResponse 개체(response)이다.
  4)미들웨어로 전달된 요청 객체에는 요청 URL을 구성하는 요소가 다음과 같이 분석되어 전달되기 때문에 모듈을 이용해서 파싱할 필요 없다.
    -request.method
    -request.url
    -request.path

```
147     -request.query
148       --app.use(function(req, res, next){
149         console.log('test');
150
151         var userAgent = req.header('User-Agent');
152         var username = req.query.username;
153
154         res.writeHead(200, {'Content-Type':'text/html;charset=utf8'});
155         res.write('<h1>Express 서버에서 응답</h1>');
156         res.write('<div>User-Agent : ' + userAgent + '</div>');
157         res.write('<div>Name : ' + username + '</div>');
158         res.end();
159       });
160       http://localhost:3000/?username=Michael
161
162   5)미들웨어 사용
163     -use()함수를 이용해서 등록
164       app.use([미들웨어]);
165       app.use(function(req, res){
166         res.send('Hello Express!');
167       });
168
169   6)Basic routing
170     -HTTP 요청 메소드와 요청 URL 경로마다 서로 다른 미들웨어를 동작시키는 라우팅 기능 제공
171     -Refers to determining how an application responds to a client request to a particular
        endpoint, which is a URI (or path) and a specific HTTP request method (GET, POST, and
        so on).
172     -Each route can have one or more handler functions, which are executed when the
        route is matched.
173     -app.METHOD(PATH, Handler);
174       --METHOD : HTTP 메소드(get, post, put, delete 등)
175       --PATH : 요청 URI 경로
176       --Handler : 라우팅 조건에 맞으면 동작하는 핸들러 함수
177     -HTTP 요청 메소드별로 처리
178       //GET 요청
179       app.get('/', function(req, res){
180         res.send('Hello Express');
181       });
182       //POST 요청
183       app.post('/', function(req, res){
184         res.end('POST request');
185       });
186     -특정 라우팅 경로별로 처리
187       //  /user 경로에 GET 요청
188       app.get('/user', function(req, res){
189         res.send('GET request, /user');
190       });
191       //  /items 경로에 GET 요청
192       app.get('/items', function(req, res){});
193       //  /user 경로에 DELETE 요청
194       app.delete('/user', function (req, res) {
```

```
        res.send('Got a DELETE request at /user')
      })

7)정적 파일
http://localhost:3000/images/avatar.jpg
  -이미지 파일이나 내용이 고정된 HTML과 같은 정적 파일 요청은 express.static 미들웨어를 사용한다.
  -정적 파일을 처리하는 미들웨어는 별도의 서버 로직을 동작시킬 필요가 없으므로, 정적 파일 요청 여부를 먼저 확인한다.
  -다음은 public 폴더에서 정적 파일을 찾아서 응답하도록 작성한 코드이다.
    app.use(express.static(path.join(__dirname, 'public')));
  -다음과 같은 경로의 요청이 들어오면 정적 파일 미들웨어는 public/images/kitty.jpg 파일을 찾아서 클라이언트에게 응답 메시지로 전송한다
    http://localhost:3000/images/kitty.jpg
  -만일 public 폴더 하위에 index.html이 있을 경우에도 가능하다.
    http://localhost:3000/
  -정적 파일 미들웨어를 사용하면 정적 파일 요청을 상대적 경로로 다룰 수 있다는 장점이 있다.
  -정적 파일 미들웨어는 다음과 같이 여러 개를 설정할 수 있다.
    app.use(express.static(path.join(__dirname, 'public'));
    app.use(express.static(path.join(__dirname, 'files'));
  -정적 파일 기능을 작성하면서 가상 경로를 사용할 수 있다.
  -다음은 static이라는 가상 경로를 설정한 것이다.
    app.use('/static', express.static(path.join(__dirname, 'public')));
  -하지만 실제로 서버에는 static이라는 폴더는 없다. 현재 모든 정적 파일은 public 폴더 하위에 있다고 가정하자.
  -그렇다면 이렇게 요청할 것이다.
    http://localhost:3000/static/index.html
  -실제로 static 폴더도 없고 public 폴더 하위에 index.html이 있기 때문에 가상 경로를 가지고 클라이언트에게 실제 경로를 숨길 수 있는 것이다.
  -다음 요청 URL에는 static 경로가 포함되어 있지만, 실제로 서버의 public 폴더 하위로 매핑한다.
    http://localhost:3000/static/hello.html --> public/hello.html로 연결
    http://localhost:3000/static/images/kitty.jpg --> public/images/kitty.jpg로 매핑
  -다음과 같은 정적 파일 요청도 가능하다.

    var express = require('express');
    var app = express();
    app.use(express.static(__dirname, {'index' : ['index.html', 'index.htm']}));
    app.listen(3000);

8)다양한 응답 메소드
  -res.json() : JSON 응답 메세지 전송
  -res.redirect([status,] path) : 리다이렉션 응답 전송
    --app.use(function(req, res, next){
      console.log('test');
      res.redirect('http://www.google.co.kr');
    });
  -res.render(view[,locales][,callback]) : 템플릿으로 렌더링
  -res.send([body]) : client에게 JSON, HTML, Buffer 전송, 메세지 헤더에 Content-Type 자동 설정
    --app.use(function(req, res, next){
      console.log('test');
      res.send({name:'한지민', age:24});
    });
```

```
242     -res.sendStatus() : 상태 코드와 상태 메세지 전송
243     -res.status(code) : 상태 코드 설정, 응답 메소드 종료 안함
244     -res.download() : 파일 다운로드
245
246
247 7. express에서  다양한 미들웨어 사용하기
248   1)미들웨어 설정(Mount)
249     app.use([path,] function[, function...])
250     path : 경로, 생략시 루트(/)
251
252   2)미들웨어 구현 분리
253     -express 어플리케이션은 다음과 같이 미들웨어 함수를 설정한다.
254       app.use(function(req, res){
255         res.send('Hello Express!');
256       });
257     -다음과 같이 미들웨어의 구현 코드를 분리해서 작성할 수도 있다.
258       app.use(sayHello);
259
260       function sayHello(req, res){
261         res.send('Hello Express!');
262       };
263     -미들웨어의 구현을 별도로 분리해서 작성하면 다음과 같이 라우팅 로직만 간결하게 작성할 수 있다는 장점이 있
      다. -->Express v3.x
264       app.get('/movies', showMoviesList);
265       app.post('/movies', addMovieInfo);
266       app.put('/movies/:id', updateMovieInfo);
267
268   3)연속된 미들웨어
269     -다수의 미들웨어를 사용하는 경우 미들웨어 함수의 3번째 파라미터로 next를 사용한다.
270     -next를 호출하면 다음 미들웨어가 실행되고, next를 호출하지 않으면 다음 미들웨어가 동작하지 않는다.
271     app.use(function(req, res, next){
272       console.log('Hello Express!');
273       //다음 미들웨어가 동작하도록 next 호출
274       next();
275     });
276
277   4)Lab : basic.js
278     var express = require('express');
279     var app = express();
280
281     app.use(function(req, res) {
282         res.send('Hello Express');
283     });
284
285     app.listen(8124);
286
287   5)Lab : next.js
288     var express = require('express');
289     var app = express();
290
291     app.use(function(req, res, next) {
```

```
292         var now = new Date();
293         console.log(now.toDateString() + ' - url : ' + req.url);
294         next();   //첫번째 미들웨어는 next()를 호출해서 다음 미들웨어가 실행된다.
295       });
296
297       app.use(function(req, res) {
298         res.send('Hello Express!!');  //두번째 미들웨어는 next()가 없어서 다음 미들웨어가 실행되지 않는
            다.
299       });
300
301       app.listen(8124);
302       ---------------------------------------
303       Wed May 03 2017 - url : /
304       Wed May 03 2017 - url : /favicon.ico
305
306     6)미들웨어는 다음과 같이 Stack 방식으로 작성할 수도 있다.
307       -하나의 요청에 다수 미들웨어
308       //두 미들웨어를 스택 형태로 설정
309       app.use(logger, sayHello);
310
311       //next()를 호출해야 다음 미들웨어 실행
312       function logger(req, res, next){
313         next();
314       }
315       //다음에 실행될 미들웨어가 없으면 next()를 사용하지 않아도 된다.
316       function sayHello(req, res) {
317           //응답하기
318       }
319
320
321   8. 미들웨어의 종류
322     1)어플리케이션 수준의 미들웨어 : express객체(app)에 사용하는 미들웨어
323     2)라우터 수준의 미들웨어
324     3)에러 처리 미들웨어
325     4)내장 미들웨어
326     5)서드파티 미들웨어
327
328
329   9. 미들웨어 동작 순서 중요
330     1)파비콘 처리 미들웨어
331     2)로깅
332     3)정적 파일
333     4)서비스 미들웨어
334
335
336   10. 내장 미들웨어와 서드파티 미들웨어
337     1)express 버전이 4.x로 변경되면서 Connect에 더 이상 의존하지 않고, 기존의 내장(built-in) 미들웨어들이
        별도의 저장소로 독립됐다.
338     2)기존의 내장 미들웨어 중 유일하게 남은 미들웨어는 정적 파일처리를 위한 express.static 미들웨어이다.
339     3)서드파티 미들웨어는 별도로 설치해서 사용하는 미들웨어이다.
340     4)설치 방법은 npm을 사용한다.
```

```
    $ npm install cookie-parser
    var cookieParser = require('cookie-parser');

    //쿠키 분석용 미들웨어
    app.use(cookieParser());


11. 정적 파일 요청 처리 미들웨어
  1)정적파일 처리 미들웨어
    -express.static

  2)미들웨어 생성
    -express.static(root, [options])

  3)options
    -etag : etag 사용 여부, 기본은 true
      --ETag(Entity Tag) : 브라우저의 캐시에 저장되어 있는 구성요소와 원본 웹 서버의 구성요소가 일치하는
      지 판단하는 방법.
    -lastModified : last-modified 헤더 사용. 기본 true
    -maxAge : max-age 메세지 헤더. 기본값 0
    -index : 경로 요청시 기본 제공 파일. 기본값 index.html

  4)정적 파일 서비스
    -// 경로 정보 파라미터 필수
    -app.use(express.static('images'));

  5)정적 파일 요청
    -SERVER-ADDRESS/cute1.jpg --> ./images/cute1.jpg
    -SERVER-ADDRESS/images/cute1.jpg -> ./images/images/cute1.jpg

  6)정적 파일 위치 설정 - 다수 가능
    -첫번째 미들웨어 서비스 실패 -> 다음 미들웨어 실행
    -app.use(express.static('public'));
    -app.use(express.static('files'));
    -가상 경로 설정
      app.use('/static', express.static('files'));
      SERVER-ADDRESS/static/image.png -> ./files/image.png

  7)Lab : static.js
    var express = require('express');
    var app = express();

    app.use(express.static(__dirname + '/images'));

    app.use(function(req, res) {
      res.send('Hello Express');
    });

    app.listen(8124);
    ---------------------------------------
    http://localhost:8124/
```

```
391      결과 : Hello Express
392      http://localhost:8124/kitty.jpg
393
394    8)Lab : demo.js
395        var express = require('express');
396
397        var app = express();
398
399        app.use('/static', express.static(__dirname));
400        app.get('/', function (req, res) {
401           //console.log("dirname : " + __dirname);
402           res.writeHead('200', {'Content-Type' : 'text/html;charset=utf-8'});
403           res.write('<h1>Hello World!</h1>');
404           res.end();
405        })
406
407        app.listen(3000, function () {
408           console.log('Example app listening on port 3000!');
409        });
410        ------------------------------------------------------------------
411        http://localhost:3000/avatar.jpg --> X
412        http://localhost:3000/static/avatar.jpg  --> O
413
414  12. Routing
415    1)개요
416      -클라이언트의 요청을 미들웨어로 분배
417      -특정 경로와 메소드에 반응하도록 작성
418      -서로 다은 요청들을 각각에 맞는 핸들러와 매핑해주는 기능
419      -HTTP 모듈을 이용할 때에는 여러개의 if문을 사용했었음.
420      -라우팅은 HTTP 메소드와 결합할 수 있지만, 경로와 함께 작성해야
421      app.post(handler);   //error
422      app.post('/', handler);
423
424    2)라우팅 종류
425      -요청 경로
426      -요청 메소드
427      -요청 경로 + 요청 메소드
428
429    3)메소드별 라우팅 함수
430      app.all(path, callback[, callback...])
431      app.get(path, callback[, callback...])
432      app.post(path, callback[, callback...])
433      app.put(path, callback[, callback...])
434      app.delete(path, callback[, callback...])
435
436    4)파라미터
437      -path : 요청 경로
438      -callbak : 요청 담당 미들웨어
439
440    5)사용 예제
441      app.get('/', function(req, res){
```

```
    res.send('GET request, /');
  });
  app.delete('/', function(req, res){
    res.send('DELETE method');
  });
  app.put('/item/1', function(req, res){
    res.send('PUT method, /item/1');
  });
  app.all('/', function(req, res){
    res.send('모든 HTTP 메소드, /all');
  });
```

6)Lab : route.js

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

  app.all('*', function(request, response, next){
      response.writeHead(200, {'Content-Type' : 'text/plain'});
      next();
  });

  app.get('/', function(request, response){
      response.end('Welcome to the homepage!');
  });

  app.get('/about', function(request, response){
      response.end('Welcome to the ABOUT page!');
  });

  app.get('/hi/:user', function(request, response){
      response.end('Hi, ' + request.params.user + '.');
  });

  app.get('*', function(request, response){
      response.end('404!');
  });


  http.createServer(app).listen(1337);
  console.log('Server running at http://localhost:1337');
  ----------------------------------------
  http://localhost:1337/  --> Welcome to the homepage!
  http://localhost:1337/about --> Welcome to the ABOUT page!
  http://localhost:1337/hi/jimin --> Hi, jimin
```

7)동적 파라미터

-리소스의 경로를 파라미터로 사용하려면 [:PARAMETER]로 작성한다.

-다음은 /user의 하위 리소스 경로를 id 파라미터로 라우팅하도록 작성한 코드

```
  app.get('/user/:id', handler);
```

-이렇게 하면 다음과 같은 경로 모두 포함한다.

```
493         /user/1234
494         /user/abcd
495       -경로 파라미터의 값은 req.params에서 얻을 수 있다.
496       -다수의 동적 파라미터 사용 가능
497         //파라미터 1개
498         app.get('/user/:id', function(req, res){
499           var userid = req.params.id;
500           res.send('User ID ' + userid);
501         });
502         //파라미터 2개
503         app.get('/movies/:movieid/:actor', function(req, res){
504           var movieid = req.params.movieid;
505           var actor = req.params.actor;
506
507           res.send('Actor : ' + actor + ' - Movie ID : ' + movieid);
508         });
509
510     8)동적 파라미터 사용시 주의사항
511       -동적 파라미터를 사용하는 경우 라우팅 경로가 겹치게 돼서 설정 순서에 따라 동작하지 않는 경우가 있다.
512       -다음 코드에서 라우팅 코드가 user/:id보다 이후에 설정돼있으면 동작하지 않는다.
513         app.get('/user/sample', function(req, res){
514           res.send('GET /user/sample');
515         });
516       -라우팅 경로가 겹치면
517         --앞 순위 미들웨어가 담당
518         --순서 조절 필요
519
520     9)경로에 정규식 사용하기
521       -? : 문자 존재하거나 생략
522       -+ : 1번 이상 반복
523       -* : 임의의 문자
524
525     10)예제
526       app.get('/ab?cd', function(req, res){});  /abcd, /acd
527       app.get('/ab+cd', function(req, res){});  /abcd, /abbcd, /abbbcd
528       app.get('/ab*cd', function(req, res){});  /abcd, /abxcd, abFABDFDcd, /ab123cd
529       app.get('/ab(cd)?e', function(req, res){});  /abe, /abcde
530
531     11)app.route()
532       -하나의 경로(path)에 메소드 별로 라우팅 로직을 작성할 수 있다.
533       app.route('/book')
534         .get(function(req, res){
535           res.send('Get a random book');
536         })
537         .post(function(req, res){
538           res.send('Add a book');
539         })
540         .put(function(req, res){
541           res.send('Update the book');
542         });
543
```

12)라우터 수준의 미들웨어
-express의 라우터는 Express 객체 대신 라우터 객체에 설정가능
-라우터 객체는 다음과 같이 얻는다.
```
var router = express.Router();
```
-라우터 객체를 사용하면 Express의 라우팅 코드를 별도 파일로 작성할 수 있다.
-라우팅 코드를 분리하면 express 어플리케이션의 설정이 복잡해지는 것을 피할 수 있는 장점이 있다.
-라우터 객체를 이용한 라우팅 코드
```
<router.js>
var express = require('express');
var router = express.Router();

//라우팅 로직 작성
router.get('/hello', sayHello);
router.get('/howAreYou/:who', sayThankYou);

//모듈 exports
module.exports = router;
```
-이제 별도의 파일로 작성한 라우팅 모듈을 express 어플리케이션에 설정하는 코드는 다음과 같다
```
var express = require('express');
//라우팅 모듈 로드
var router = require('./router');

var app = express();
//라우터 사용 설정
app.use(router);
```
-라우팅 모듈을 별도로 작성하면, 요청의 종류별로 라우팅 모듈을 분리할 수 있다는 장점이 있다.
-다음은 greeting과 eat이라는 요청 경로를 서로 다른 라우터가 다루도록 설정하는 코드이다.
-각 라우터는 하위 경로부터 작성한다.
```
//요청 경로 /greeting/hello
app.use('/greeting', require('./greetingRouter'));

//greetingRouter의 라우팅 코드
router.get('/hello', sayHello);

//요청 경로 /eat/cooking
app.use('/eat', require(''./eatingRouter'));

//eatingRouter의 라우팅 코드
router.get('/cooking', cook);
```

13)Lab
```
<greetingRouter.js>
var express = require('express');
var router = express.Router();

// /hello, /howAreYou/[who]
router.get('/hello', sayHello);
router.get('/howAreYou/:who', sayThankYou);
```

```
    function sayHello(req, res) {
       res.send('Hello Router');
    }

    function sayThankYou(req, res) {
       var who = req.params.who;
       res.send('Fine Thank You ' + who + " And  you?");
    }

    module.exports = router;

    <app.js>
    var express = require('express');
    var app = express();

    app.use(require('./greetingRouter'));

    app.listen(8194);
```

14)Lab : route1.js

```
    var express = require('express');
    var app = express();

    app.get('/:value', work);
    app.listen(8194);

    function work(req, res, next) {
       var val = parseInt(req.params.value);

       // 입력 파라미터 체크
       if ( ! val ) {
           res.status(400).send('입력값이 숫자가 아닙니다.');
           return;
       }

       res.send('Result : ' + val);
    }
    ---------------------------------------
    http://localhost:8194/abc --> 입력값이 숫자가 아닙니다.
    http://localhost:8194/123 --> Result : 123
```

13. 에러 처리

1)각 미들웨어마다 발생하는 에러를 미들웨어 내에서 처리하지 않고 에러만 전문으로 담당하는 미들웨어를 작성할 수 있다.

2)에러를 담당하는 미들웨어는 다음과 같이 에러를 첫번째 파라미터로 선언한다.

```
  -app.use(function(err, req, res, next){
       res.status(500).send('에러 발생');
     });
```

3)에러를 처리하는 미들웨어는 사용자의 요청을 처리하는 미들웨어보다 나중에 동작하도록 설정한다.

4)사용자의 요청을 처리하는 미들웨어에서 클라이언트의 요청 메세지에 오류가 있으면 에러 처리를 담당하는 미들웨어에게 에러 정보를 전달한다.
5)에러 정보는 Error를 이용해서 생성한다.

```
var error = new Error('에러 메세지');
```

6)에러 처리 미들웨어로 에러 전달

```
app.use(function(req, res, next){
  var error = new Error('에러 메세지');
  error.code = 100;
  return next(error);
});
```

7)환경별 에러 처리
-에러 정보 출력
-개발 중 에러 처리

```
Error : Error!
   at /...../env.js:9:9
   at ..........
   at .....
   at .....
```

-서비스 중 에러 처리
--사용자 친화적 메세지 필요 : '잠시 후 다시 시도해 주세요'

8)Lab : error.js

```
var express = require('express');
var app = express();
app.get('/:value', work);
app.listen(3000);

function work(req, res, next) {
    var val = parseInt(req.params.value);
    // 입력 파라미터 체크
    if ( ! val ) {
        res.writeHead('200', {'Content-Type' : 'text/plain;charset=utf-8'});
        res.write('입력값이 숫자가 아닙니다.');
        res.end();
        return;
    }
    res.send('Result : ' + val);
}
```

---

http://localhost:3000/1234 --> Result : 1234
http://localhost:3000/abcd --> 입력값이 숫자가 아닙니다.

14. 파비콘 미들웨어

1)웹 브라우저에서 사이트의 주소창에 출력되는 작은 이미지
2)웹 브라우저로 express 서버에 접속하면 이 파비콘을 요청한다.
3)파비콘 담당
-serve-favicon

-https://github.com/expressjs/serve-favicon

4)파비콘의 요청은 로그로 남기기에 중요하지 않으므로, 많은 경우 파비콘 미들웨어를 로그 미들웨어보다 먼저 동작하도록 설정한다.
5)Install
$ npm install serve-favicon
|
|---serve-favicon@2.4.2

6)미들웨어 생성
favicon(path, options);
-path : 파비콘 경로

7)설정예제
```
var express = require('express');
var morgan = require('morgan');
var favicon = require('serve-favicon');
var app = express();

//파비콘 처리 - 파일 없으면 런타임 에러
app.use(favicon(__dirname + '/public/favicon.ico'));
//로그 남기기
app.use(morgan('combined'));
//응답 미들웨어
app.get('/', function(req, res){
  res.send('hello, world');
});
app.listen(8194);
```

15. 로그 남기기
1)https://www.npmjs.com/package/morgan
2)log
-개발용, 버그 분석
-사용자 행위 기록
-운영 기록

3)로그 기록 매체
-콘솔에 출력
-파일, 데이터베이스에 기록
-이메일, SMS 등...

4)로그 남기기 - Console
```
-console.info('Info Message');
-console.log('Log Message');
-console.warn('Warn Message');
-console.error('Error Message');
```

5)로그 미들웨어 - morgan
-간단한 설정
-요청과 응답을 자동으로 로그 남기기

```
    -로그 설치
      $ npm install morgan
            |
            |---morgan@1.8.1

  6)morgan 설정
    morgan(format, options)

  7)로그 포맷
    -combined
    -common : addr, user, method, url, status, res
    -dev : method, url, status, response-time, res
    -short
    -tiny

  8)사용예
    var morgan = require('morgan');
    app.use(morgan('dev'));

  9)요청과 응답 자동으로 기록
    -GET / 200 0.442 ms -7
    -GET / 400 0.229 ms - 11

  10)미들웨어 중 앞 순위로 설정할 것

  11)Lab : log.js
    var express = require('express');
    var morgan = require('morgan');
    var app = express();

    app.use(morgan('dev'));

    app.get('/hello', function (req, res) {
       res.send('GET request, /');
    });

    app.get('/movies', function(req, res) {
       res.send('GET request, /movies');
    });

    app.listen(3000);
    ------------------------------------------------
    GET / 404 19.001 ms - 139
    GET /hello 200 9.086 ms - 14


16. express-winston
  1)로그 미들웨어
  2)express-base에서 다양한 형태로 로그 남기기
  3)https://github.com/bithavoc/express-winston
  4)https://www.datawire.io/distributed-logging-express-js-request-winston/
```

```
5)파일로 남기기, 데이터베이스에 저장, Email, sms, 알림 서비스 사용
6)서비스 운영시 콘솔을 항상 볼 수 없다.
7)Install
  $ npm install winston express-winston

8)모듈 로딩
  var winston = require('winston')
    , expressWinston = require('express-winston');

9)로그 함수
  -winston.log(level, Message);
  -winston.info(Message);
  -winston.warn(Message);
  -winston.error(Message);

10)로그 기록 매체 : Transport
  -기본 제공 : Transport(Core Transports)
  -Console
  -File
  -Http

11)Transport 추가/삭제
  -winston.add(winston.transports.File, {filename: 'service.log'});
  -winston.remove(winston.transports.Console);

12)사용예제
  var logger = new winston.Logger({
    transports : [
      new winston.transports.Console(),
      new winston.transports.File({
        name : 'error-logger',
        filename : 'service-error.log',
        level : 'error'
      })
    ]
  });
  logger.error('Error Message');
  logger.info('Info Message');
  logger.warn('Warn Message');

13)별도 설치
  -DailyRotateFile
  -CouchDB, Redis, MongoDB
  -Mail transport
  -Notification Service(Amazon SNS)

14)검색
  -Winston Transports
  $ npm install winston

15)날자별로 로그 파일 남기기
```

```
847      -DailyRotateFile
848      -https://github.com/winstonjs/winston-daily-rotate-file
849      -Install
850        $ npm install winston-daily-rotate-file
851      -사용방법
852        winston.add(require('winston-daily-file'), options)
853
854    16)예제코드
855      var winston = require('winston');
856      winston.add(require('winston-daily-rotate-file'),
857                      {datePattern:'yyyyMMdd', filename:'service.log'});
858
859      winston.info('Info Message');
860      winston.warn('Warning Message');
861      winston.error('Error Message');
862
863    17)Lab
864      <logdemo.js>
865          var express = require('express');
866          var expressWinston = require('express-winston');
867          var winston = require('winston');
868
869          var app = express();
870
871          app.use(expressWinston.logger({
872            transports: [
873              new winston.transports.Console({
874                json: true,
875                colorize: true
876              }),
877              new winston.transports.File({
878                level:           'info',
879                filename:         './service-error.log',
880                handleExceptions: true,
881                json:            true,
882                maxsize:          5242880, //5MB
883                maxFiles:         5,
884                colorize:         false
885              })
886            ]
887          }));
888
889          app.listen(3000);
890
891
892  17. body-parser
893    1)HTTP GET 요청은 URL을 통해서 요청 내용이 전달되지만, POST와 같은 메소드는 요청 메세지의 바디에 요청
       메세지가 전달된다.
894    2)body-parser는 요청 바디로 전달되는 데이터를 분석하는 기능을 제공하는 미들웨어
895    3)https://github.com/expressjs/body-parser
896    4)Install
```

```
  $ npm install body-parser
       |
       |-------body-parser@1.17.1

5)body-parser는 다음과 같은 타입으로 작성된 바디를 파싱하는 기능을 제공
  -JSON body parser
  -Raw body parser
  -Text body parser
  -URL-encoded form body parser

6)body-parser를 사용하면 요청 메세지를 분석해서 req.body로 파싱된 내용이 전달된다.
7)물론 개별 요청을 처리하는 미들웨어보다 먼저 동작하도록 설정해야 한다.
8)멀티파트 지원안됨
  -바디파서는 멀티파트 메세지 파싱 불가능
  -formidable, multer 등 서드파티 미들웨어 사용

9)URL-Encoded
  -컨텐츠 타입이 application/x-www-form-urlencoded인 요청 바디를 파싱한다.
  -파싱한 내용은 req.body를 통해서 얻을 수 있다.
  -다음은 urlencoded 바디 파서를 사용하도록 설정하는 코드이다.
    app.use(bodyParser.urlencoded({extedded:false});
  -Options
    --inflate
      ---압축된 메세지 바디 다루기. 기본 true
    --extended
      ---false : 기본 모듈인 querystring을 사용해서 파싱
      ---true : qs 라이브러리를 사용
      ---기본값이 없어서 바디 파서를 설정하면서 옵션을 작성한다.
    --limit
      ---바디로 전달되는 메시지의 최대 크기로 크기가 큰 메세지를 전송하는 서버공격을 막는다
      ---기본값은 100KB
    --parameterLimit
      ---메시지 바디내 파라미터 갯수 제한. 기본 1000
  -다음 코드는 urlcoded 방식으로 요청 메세지 바디로 전달된 title, message 값을 얻는 코드이다.
    app.use(bodyParser.urlencoded({extended : false});
    app.post('/', function(req, res){
      var title = req.body.title;
      var message = req.body.message;
    });
    --키에 해당하는 값이 없으면 undefined가 된다.

  -Lab: Postman으로 테스트할 것
    Body > x-www-form-urlencoded
    title : 아바타
    message : 제임스 카메런

    <bodyparser.js>
    var express = require('express');
    var querystring =  require('querystring');
    var app = express();
    app.listen(3000);
```

```
948
949         app.post('/', function (req, res) {
950             var buffer = '';
951             req.on('data', function(chunk){
952                 buffer += chunk;
953             });
954             req.on('end', function(){
955                 var parsed = querystring.parse(buffer);
956                 var title = parsed.title;
957                 var message = parsed.message;
958                 console.log('title = ' + title + ', message = ' + message);
959             });
960         });
961         -위의 코드는 아래와 같이 변경이 된다.
962         var express = require('express');
963         var bodyParser = require('body-parser');
964
965         var app = express();
966         app.listen(3000);
967
968         app.use(bodyParser.urlencoded({extended:false}));
969
970         app.post('/', function (req, res) {
971             var title = req.body.title;
972             var message = req.body.message;
973
974             res.send('title : ' + title + ' message : ' + message);
975
976         });
977         -하지만 위의 코드는 JSON 방식 처리는 하지 못한다.
978           Postman에서
979           Body > raw > JSON(application/json)
980           {
981             "title" : "타이타닉",
982             "message" : "제임스 카메론"
983           }
984           결과 : title : undefined message : undefined
985         -그래서 JSON 파서도 같이 넣어준다.
986           app.use(bodyParser.urlencoded({extended:false}));
987           app.use(bodyParser.json());
988           결과 : title : 타이타닉 message : 제임스 카메론
989
990     10)JSON body parsing
991       -클라이언트 요청이 JSON 형태로 전달되면 JSON 파서를 사용한다.
992       -이 때의 요청 메세지의 컨텐츠 타입은 application/json이다.
993       -JSON 바디로 파싱한 데이터는 req.body로 얻을 수 있다.
994       -bodyParser.json(options)
995       -options
996         --inflate : 압축된 메세지 바디 다루기. 기본 true
997         --limit : 바디 메시지 크기. 기본 100KB
998         --strict : JSON의 루트 항목이 배열이나 객체만 접수. 기본 true
```

```
  -다음은 JSON 바디 파서를 설정하는 코드이다.
    app.use(bodyParser.json());

11)Lab
  <app.js>
      var express = require('express');
      var bodyParser = require('body-parser');

      var app = express();

      app.use(express.static(__dirname));
      app.use(bodyParser.json());
      app.use(bodyParser.urlencoded({ extended: false }));

      app.post('/login', function(req, res){
         var id = req.body.id;
         var passwd = req.body.passwd;
         res.writeHead('200',  {'Content-Type' : 'text/html;charset=utf-8'});
         res.write("<h1>로그인 성공</h1>");
         res.write("아이디 : " + id + "<br />");
         res.write("비밀번호 : " + passwd);
         res.end();
      });

      app.listen(3000);

<login.html>
      <!doctype html>
      <html>
         <head>
            <meta charset='utf-8' />
            <title>로그인 페이지</title>
         </head>
         <body>
            <h1>로그인</h1>
            <form method='post' action='/login'>
               <p>
                  <label for='txtId'>아이디</label>
                  <input type='text' id='txtId' name='id' />
               </p>
               <p>
                  <label for='txtPasswd'>패스워드</label>
                  <input type='password' id='txtPasswd' name='passwd' />
               </p>
               <p>
                  <input type='submit' value='전송' />
               </p>
            </form>
         </body>
      </html>
```

18. 메소드 오버라이드 미들웨어
1)form에서는 method를 GET이나 POST가 아닌 DELETE같은 것을 사용해도 결국 GET 메소드로 인식함
2)그래서 GET이나 POST가 아닌 다른 메소드를 인식하게 하는 것
3)메소드 오버라이드 미들웨어
-HTML폼에서 GET/POST외 다른 메소드 사용가능
-form의 method(GET/POST)를 다른 메소드로 덮어쓰기
-쿼리 문자열로 덮어쓸 메소드 전달
POST /?resource_method=DELETE

4)https://ewiggin.gitbooks.io/expressjs-middleware/content/method-override.html

5)https://github.com/expressjs/method-override

6)Install
$ npm install method-override
|
|-----method-override@2.3.8

7)미들웨어 설정
app.use(methdoOverride('_method'));

8)PUT/DELETE 처리
```
<form method='post' action='/?_method=delete'>
  <input type='text' name='title'>
  <input type='hidden' name='_method' value='delete'>
  <input type='submit' value='Delete'>
</form>
```

19. 렌더링과 템플릿 엔진
1)express는 웹 브라우저를 위한 HTML 형태로 응답을 하기도 한다.
2)HTML은 view 구조와 contents가 섞여 있는 방식이다
3)그렇다면 단순 문자열 연산을 위한 HTML 생성은 효율적이지 않다.
-HTML 응답 메시지 작성 예
```
res.write('<!doctype html>');
res.write('<html>');
res.write('<body>');
res.write('<h3>Favorite Movie</h3>');
res.write('<div>');
data.forEach(function(item){
    res.write(item.value);
});
res.write('</div>');
res.write('</body>');
res.end('</html>');
```
-Code와 HTML 결합 --> 코드 작성/수정/협업(각자 역할분담) 어려움 발생

4)템플릿을 사용하면, 화면에 출력될 View 구조와 데이터를 간단히 결합해서 HTML을 얻을 수 있다.
5)템플릿
-데이터를 인터프리트하고 view를 렌더링하기 위한 일련의 규칙과 언어를 사용하는 라이브러리 또는 프레임워크

```
1101      -뷰와 제어 코드(데이터)분리
1102      -결과물 : HTML
1103
1104    6)템플릿 코드
1105      <% var item = {value: 'Template Example' } %>
1106      <li><%=item.value %></li>
1107
1108    7)템플릿 엔진
1109      -ejs
1110      -jade
1111      -handlebars
1112
1113    8)템플릿 엔진 설정
1114      -express에서 사용하는 템플릿 엔진과 템플릿 파일 경로를 설정한다.
1115      -views : 템플릿 파일의 위치를 설정
1116      -view engine : 템플릿 엔진을 설정
1117      -express객체에 set 함수로 설정
1118        app.set('views', '[템플릿 폴더]);
1119        app.set('views', './views');
1120
1121        app.set('view engine', [템플릿 엔진]);
1122        app.set('view engine', 'ejs');
1123
1124    9)템플릿에 적용 : 렌더링
1125      -응답 메시지의 내용을 템플릿 파일을 이용해서 렌더링하는 메소드는 다음과 같다.
1126        res.render(view[, locals][,callback])
1127      -view : 템플릿 파일
1128      -locals : 템플릿의 지역변수로 설정될 데이터
1129      -callback : 에러와 렌더링 결과 html을 다루는 콜백 함수
1130      -응답 종료
1131
1132    10)렌더링 예
1133      //index 템플릿 파일을 렌더링한 결과로 응답
1134      res.render('index');
1135
1136      //index 템플릿 파일에 name이름으로 데이터를 제공
1137      res.render('index', {name : 'Node.js'});
1138
1139      //user 템플릿에 name이라는 이름으로 데이터를 제공. 렌더링한 결과를 다루는 콜백함수 정의
1140      res.render('user', {name:'Node.js'},function(err, html){
1141      ...
1142      });
1143
1144    11)렌더링 에러
1145      -템플릿 페이지가 없을 경우
1146        Error : Failed to lookup view 'notexist' in views directory '/Users/...'
1147      -템플릿 엔진 설정 없이 rendering 한 경우
1148        Error : No default engine was specified and no extension was provided.
1149      -템플릿 내 객체 정의 에러
1150        <%=undefinedVar %>
1151
```

20. EJS Template Engine
  1)http://www.embeddedjs.com/
  2)html에 특수 태그 형태로 데이터를 작성하는 형태이다.
  3)특징
    -HTML 코드 그대로 사용
    -코드 실행 : <% %>
    -결과 출력 : <%= %>
  4)Install
    $ npm install ejs
        |
        |------ejs@2.5.6

  5)다음은 title이름의 데이터를 h1태그로 출력되도록 작성한 코드다.
    <h1><%=title %> </h1>

  6)실행코드 작성
    -한 줄 작성
      <% var value='hello' %>
    -여러 줄 작성
      <%
        var i = 0;
        var j = i + 1;
      %>
    -HTML과 혼합
      <% if(value) { %>

      <% } %>

  7)HTML로 결과 출력
    -값으로 출력
      <%=value %>
    -태그 내 어트리뷰트
      <img src='<%=data.image %>'>
    -HTML과 혼합
      <% var tag = 'h1' %>
      <<%=tag %>>Tag 만들기</<%=tag %>>

        <% for(var i = 0 ; i < 10 ; i++){ %>
          <%=i %>
        <% } %>

  8)Lab
    <views/sports.ejs>
      <html>
        <head>
            <title>EJS Example</title>
        </head>

```
        <body>
            <h1><%=title %></h1>
            <ul>
                <% sports.forEach(function(item){ %>
                    <li><img src='images/<%=item.image %>'
                    height='50px'><%=item.title %></li>
                <% }) %>
            </ul>
        </body>
    </html>

<ejsServer.js>
    var express = require('express');
    var app = express();

    app.set('views', __dirname + '/views');
    app.set('view engine', 'ejs');

    var data = [
        {title:'야구', image:'baseball.png'},
        {title:'농구', image:'basketball.png'},
        {title:'축구', image:'football.png'}
    ];

    app.use(express.static('./'));
    app.get('/', function(req, res){
        //ejs 템플릿으로 렌더링
        res.render('sports', {title:'구기 종목', sports:data});
    });

    app.listen(3000);


21. Jade Template Engine
  1)https://www.npmjs.com/package/jade
  http://learnjade.com/tour/intro/
  2)특징
    -ejs보다 간결한 문서 구조 표현
    -들여쓰기를 이용한 문서 구조

  3)들여쓰기는 탭 문자나 공백 문자를 사용할 수 있으며 섞어쓸 수 없다.
    -단, 탭이면 모두 탭, 공백문자이면 모두 공백문자로 통일해야 한다.
    -기본은 두칸이다.

  4)Install
    $ npm install jade
        |
        |------jade@1.11.0

  5)HTML 태그
    -{} 없이 태그만 사용
```

```
    -들여쓰기로 문서 구조 표현
    -시작 태그 - 종료 태그 구조 사용 안함

    <JADE code : views/test.jade>
      doctype html
      html
          head
              meta(charset='utf-8')
              title= title  <--res.render('test', { title: 'Jade Example' });
          body
              h1= title
              div
                  p Welcome to Jade Homepage.

    <JavaScript code : testjade.js>
      var express = require('express');
      var app = express();

      app.set('views', __dirname + '/views');
      app.set('view engine', 'jade');

      app.use(express.static('./'));
      app.get('/', function(req, res){
              res.render('test', { title: 'Jade Example' });
      });

      app.listen(3000);

    <결과 페이지>
    HTML
      <!DOCTYPE html>
      <html>
        <head>
          <meta charset="utf-8">
          <title>Jade Example</title>
        </head>
        <body>
          <h1>Jade Example</h1>
          <div>
            <p>Welcome to Jade Homepage.</p>
          </div>
        </body>
      </html>

  6)어트리뷰트 작성
    img(src='image.jpg' height='50px')
    a(href='http://google.com') google
    link(rel='stylesheet', href='/stylesheets/style.css')
    div#contents --> <div id='contents'></div>
    span.box --> <span class='box'></span>
    div#contents.box  --> <div id='contents' class='box'></div>
```

```
1304
1305    7)변수와 로컬
1306        h1= title
1307        p= body
1308
1309        app.get('/', function(req, res){
1310            res.render('test', {
1311                    title : 'Express.js Guide',
1312                        body : 'The Comprehensive Book on Express.js'
1313            });
1314        });
1315
1316        <h1>Express.js Guide</h1>
1317        <p>The Comprehensive Book on Express.js</p>
1318
1319    8)속성
1320      -'|'기호는 새로운 라인에 HTML 노드 내용 추가
1321      -이 파이프 기호는 내부 텍스트가 된다.
1322        a(href=url, data-active=isActive) google.com
1323            br
1324            label
1325                input(type='checkbox', checked=isChecked)
1326                |  yes / no
1327
1328        res.render('test', {
1329            url : 'http://www.google.com',
1330            isActive : true,
1331            isChecked : false
1332      });
1333
1334        <a href="http://www.google.com" data-active>google.com</a>
1335        <br>
1336        <label><input type="checkbox"> yes / no</label>
1337
1338    9)텍스트
1339      -텍스트 자체를 출력하기 위해서 파이프 기호('|') 사용
1340        div
1341          | Jade is a template engine.
1342          | Node.js에서 사용될 수도 있고 브라우저 자바스크립트에서도 사용된다.
1343
1344        res.render('test');
1345
1346        <div>Jade is a template engine.
1347        Node.js에서 사용될 수도 있고 브라우저 자바스크립트에서도 사용된다.</div>
1348
1349    10)Script & Style Block
1350      -HTML에 script 또는 style 태그용인 대량의 콘텐츠들을 넣고자 할 때는 .(dot)를 사용
1351        script.
1352            var sum = 0;
1353            for(var i = 0 ; i <= 10 ; i++){
1354                sum += i;
```

```
1355            }
1356            document.write('sum = ' + sum);
1357
1358        <script>
1359          var sum = 0;
1360          for(var i = 0 ; i <= 10 ; i++){
1361            sum += i;
1362          }
1363          document.write('sum = ' + sum);
1364        </script>
1365
1366    11)JavaScript 코드 작성
1367      -'-'기호, =, != 사용
1368
1369      -var array = ['Orange', 'Apple', 'Lime']
1370      ul
1371        -for (var i = 0 ; i < array.length ; i++){
1372        li
1373          span=i
1374          span!="unescaped: " + array[i] + " vs. "
1375          span="escaped: " + array[i]
1376        -}
1377
1378        <ul>
1379          <li><span>0</span><span>unescaped: Orange vs. </span><span>escaped:
            Orange</span></li>
1380          <li><span>1</span><span>unescaped: Apple vs. </span><span>escaped:
            Apple</span></li>
1381          <li><span>2</span><span>unescaped: Lime vs. </span><span>escaped:
            Lime</span></li>
1382        </ul>
1383
1384    12)HTML로 출력
1385      -'=' 기호와 문자열 출력. escaped
1386      -'!=' 기호와 문자열 출력. unescaped  <--tag 해석됨
1387
1388      //Hello <strong>World</strong>
1389      div='Hello <strong>World</strong>'
1390
1391      //Unescaped String, <---tag가 해석됨.
1392      div!='Unescaped <strong>String</strong>'
1393
1394    13)Comments
1395      -주석 처리시 //를 사용
1396      //To-Do : 여기는 주석부
1397
1398      <!-- To-Do : 여기는 주석부 -->
1399
1400    14)HTML 출력 : Interpolation
1401      -Escaped : #{val}
1402      -Unescaped : !{val}
```

```
1403
1404      - var strVal = 'Hello <strong>World</strong>'
1405      div escaped. : #{strVal}
1406        --> escaped. : Hello <strong>World</strong>
1407
1408      div unescaped. : !{strVal}
1409        --> unescaped. : Hello World  <--bold 체로 렌더링됨
1410
1411
1412      -var title = '<u>Express.js</u> Guide'
1413     p Read the #{title} in PDF, MOBI and EPUB
1414
1415      <p>Read the <u>Express.js</u> Guide in PDF, MOBI and EPUB</p>
1416
1417      -var title = '<u>Express.js</u> Guide'
1418     p Read the !{title} in PDF, MOBI and EPUB
1419
1420      <p>Read the Express.js<---underline 처리  Guide in PDF, MOBI and EPUB</p>
1421
1422    15)if 조건문
1423      -'-'기호 없이 사용
1424      -들여쓰기로 if 구문 종료
1425
1426      app.get('/', function(req, res){
1427        res.render('test', { title: 'Jade Example',  val:true});
1428      });
1429
1430      if val
1431        div val is True
1432      else
1433        div val is False
1434
1435      <div>val is True</div>
1436
1437    16)Case
1438      <test.jade>
1439        h1=title
1440        -var coins = Math.round(Math.random() * 10)
1441        case coins
1442             when 0
1443                  p You have no money
1444             when 1
1445                  p You have a coin
1446             default
1447                  p You have #{coins} coins!
1448
1449      <html>
1450           <h1>Jade Example</h1>
1451           <p>You have a coin</p>
1452
1453    17)반복문
```

```
<jadetest.js>에서
  var data = ['야구', '농구', '축구'];
  app.use(express.static('./'));
  app.get('/', function(req, res){
          res.render('test', { title: 'Jade Example',  items:data});
  });

<test.jade>에서
  body
          h1=title
          each item, i in items
               li #{i} : #{item}


  <li>0 : 야구</li>
  <li>1 : 농구</li>
  <li>2 : 축구</li>

  -var languages = ['cgi', 'php', 'asp', 'jsp']
  div
       each value, index in languages
            p= index + '. ' + value

  <div>
    <p>0. cgi</p>
    <p>1. php</p>
    <p>2. asp</p>
    <p>3. jsp</p>
  </div>

  -var fruites = {'Apple' : 10, 'Mango' : 50, 'Lime' : 100, 'Melon' : 780}
  div
       each value, key in fruites
            p= key + '. ' + value

  <div>
    <p>Apple. 10</p>
    <p>Mango. 50</p>
    <p>Lime. 100</p>
    <p>Melon. 780</p>
  </div>

18)include
  -jade는 뷰의 재사용을 위해 include 기능 지원
  <head.jade>
    head
          title=title
          meta(charset='utf-8')

  <test.jade>
    head
```

```
1505          include head
1506
1507      -또는 파일을 include 할 수 있다. 이때 형식은 /path/filename 이다.
1508      include ./includes/header
1509
1510    19)extends
1511      -상속은 기존의 템플릿을 그대로 내려받는 것을 의미한다.
1512      -include와 다른 점은 include는 그대로 복사하는 의미이고
1513      -extends는 코드를 복사하되, 아래의 예처럼 부모 코드의 block을 통해 서로 이어주는 의미를 갖는다.
1514      <layout.jade>
1515        html
1516            head
1517                block scripts
1518                    script(src='/jquery.js')
1519            body
1520                block contents
1521
1522      <index.jade>
1523        extends layout
1524        block contents    <---<부모의 [block contents]에 이어서> 라는 의미, 이때 부모의 block의 이름과
          동일해야 함
1525          h1=title
1526
1527    20)mixin 지원
1528      -반복되는 HTML을 간단히 템플리화
1529        <test.jade>
1530          mixin textInput(name, label)
1531              p
1532                  label(for='#{name}')=label
1533              input(type='text', name='#{name}')
1534
1535        mixin textInput('name', '이름')
1536        mixin textInput('email', '이메일')
1537
1538        <p>
1539          <label for="name">이름</label>
1540          <input type="text" name="name">
1541        </p>
1542        <p>
1543          <label for="email">이메일</label>]
1544          <input type="text" name="email">
1545        </p>
1546
1547    21)express에서 Jade를 템플릿 엔진으로 사용하기 위한 설정
1548      app.set('views', [템플릿 폴더]);
1549      app.set('view engine', 'jade');
1550
1551    22)Lab
1552      <sports.jade>
1553        doctype html
1554        html
```

```
      head
        meta(charset='utf-8')
        title Jade Example
      body
        h1=title
        ul
          for item in sports
            li
              img(src='images/' + item.image height='50px')
              span=item.title
```

<jadeServer.js>

```
var express = require('express');
var app = express();

app.set('views', __dirname + '/views');
app.set('view engine', 'jade');

var data = [
    {title:'야구', image:'baseball.png'},
    {title:'농구', image:'basketball.png'},
    {title:'축구', image:'football.png'}
];

app.use(express.static('./'));
app.get('/', function(req, res){
    //ejs 템플릿으로 렌더링
    res.render('sports', {title:'구기 종목', sports:data});
});

app.listen(3000);
```

22. Cookie 처리하기

1)Cookie는 클라이언트 웹 브라우저에 저장되는 정보

2)Express에서는 Cookie를 처리하기 위해 cookie-parser 미들웨어를 사용한다.
-http://expressjs.com/en/resources/middleware/cookie-parser.html

3)Installation

```
$ npm install cookie-parser --save
`-- cookie-parser@1.4.3
```

4)API

```
var express = require('express');
var cookieParser = require('cookie-parser');
var app = express();
app.use(cookieParser());
```

5)Methods
-cookieParser(secret, options)

```
1606        --secret : a string or array used for signing cookies.
1607        --This is optional and if not specified, will not parse signed cookies.
1608        --If a string is provided, this is used as the secret.
1609        --If an array is provided, an attempt will be made to unsign the cookie with each
            secret in order.
1610        --options : an object that is passed to cookie.parse as the second option.
1611        --decode a function to decode the value of the cookie
1612      -cookieParser.JSONCookie(str)
1613      -cookieParser.JSONCookies(cookies)
1614      -cookieParser.signedCookie(str, secret)
1615      -cookieParser.signedCookie(cookies, secret)
1616
1617    6)Cookie 다루기
1618      -Cookie 쓰기/삭제 : Express 기본사항
1619        --res.cookie(name, value[, options])
1620        --res.clearCookie(name[,options])
1621      -Options
1622        --domain : 쿠키가 적용되는 서버
1623        --path : 쿠키가 적용되는 경로
1624        --expire : 쿠키 유효 날짜와 시간
1625        --maxAge : 쿠키 유효 기간(ms)
1626        --httpOnly : HTTP 프로토콜에서만 사용
1627        --secure : HTTPS 에서만 사용 여부. Boolean형
1628        --signed : 서명여부. Boolean형
1629
1630    7)서명 쿠키
1631      -쿠키 서명하기(Signed cookie)
1632        --쿠키 변조 방지
1633        signed=s%3AOriginalValue.XWrx3v1RQNVBNTN60QVaOnMiARB3T7BZNs81dd7wz4A;
1634
1635      -서명된 쿠키 사용하기
1636        --쿠키 파서 설정
1637          app.use(cookieParser('SECRET_KEY'));
1638        --쿠키 쓰기
1639          res.cookie'signed', 'OriginalValue', {signed:true})
1640        --쿠키 읽기
1641          res.signedCookies.singed
1642
1643    8)쿠키의 문제
1644      -메시지 크기가 커진다 --> 느려진다
1645      -다른 웹브라우저로 접속하면?
1646      -보안에 취약
1647
1648    9)Examples
1649      var express = require('express');
1650      var cookieParser = require('cookie-parser');
1651
1652      var app = express();
1653      app.use(cookieParser());
1654
1655      app.get('/', function (req, res) {
```

```
    // Cookies that have not been signed
    console.log('Cookies: ', req.cookies);

    // Cookies that have been signed
    console.log('Signed Cookies: ', req.signedCookies);
})

app.listen(3000);
-------------------------------------------------
$ node cookiedemo.js
Cookies:  {}
Signed Cookies:  {}

9)Lab : cookiedemo.js
var express = require('express');
var cookieParser = require('cookie-parser');

var app = express();
app.use(cookieParser());

app.get('/getCookie', function (req, res) {
    res.send(req.cookies);
});

app.get('/setCookie', function(req, res){
    res.cookie('user', {
        id:'devexpert',
        name:'한지민',
        authorized : true
    });
    res.redirect('/getCookie');
});

app.listen(3000);
-------------------------------------------------
-http://localhost:3000/getCookie
{}
-http://localhost:3000/setCookie
-http://localhost:3000/getCookie
{"user":{"id":"devexpert","name":"한지민","authorized":true}}


23. Session 처리하기
 -http://expressjs.com/en/resources/middleware/session.html
 1)Installation
  $ npm install express-session --save
  `-- express-session@1.15.3
    +-- crc@3.4.4
    +-- debug@2.6.7
    | `-- ms@2.0.0
    +-- on-headers@1.0.1
```

```
1707        `-- uid-safe@2.1.4
1708          `-- random-bytes@1.0.0
1709
1710    2)API
1711      var session = require('express-session');
1712
1713    3)session(options)
1714      -Create a session middleware with the given options.
1715      -Session data is not saved in the cookie itself, just the session ID.
1716      -Session data is stored server-side.
1717      -Since version 1.5.0, the cookie-parser middleware no longer needs to be used for this
          module to work.
1718      -This module now directly reads and writes cookies on req/res.
1719      -Using cookie-parser may result in issues if the secret is not the same between this
          module and cookie-parser.
1720
1721    4)options
1722      -express-session accepts these properties in the options object.
1723      -cookie
1724        --Settings object for the session ID cookie.
1725        --The default value is { path: '/', httpOnly: true, secure: false, maxAge: null }.
1726
1727      -cookie.domain
1728        --Specifies the value for the Domain Set-Cookie attribute.
1729        --By default, no domain is set, and most clients will consider the cookie to apply to
            only the current domain.
1730
1731      -cookie.expires
1732        --Specifies the Date object to be the value for the Expires Set-Cookie attribute.
1733        --By default, no expiration is set, and most clients will consider this a "non-persistent
            cookie" and will delete it on a condition like exiting a web browser application.
1734        --If both expires and maxAge are set in the options, then the last one defined in the
            object is what is used.
1735        --The expires option should not be set directly; instead only use the maxAge option.
1736
1737      -cookie.httpOnly
1738        --Specifies the boolean value for the HttpOnly Set-Cookie attribute.
1739        --When truthy, the HttpOnly attribute is set, otherwise it is not.
1740        --By default, the HttpOnly attribute is set.
1741        --Be careful when setting this to true, as compliant clients will not allow client-side
            JavaScript to see the cookie in  document.cookie.
1742
1743      -cookie.maxAge
1744        --Specifies the number (in milliseconds) to use when calculating the Expires
            Set-Cookie attribute.
1745        --This is done by taking the current server time and adding maxAge milliseconds to
            the value to calculate an Expires datetime.
1746        --By default, no maximum age is set.
1747        --If both expires and maxAge are set in the options, then the last one defined in the
            object is what is used.
1748
```

-cookie.path
  --Specifies the value for the Path Set-Cookie.
  --By default, this is set to '/', which is the root path of the domain.

-cookie.sameSite
  --Specifies the boolean or string to be the value for the SameSite Set-Cookie attribute.
  --<true> will set the SameSite attribute to Strict for strict same site enforcement.
  --<false> will not set the SameSite attribute.
  --<'lax'> will set the SameSite attribute to Lax for lax same site enforcement.
  --<'strict'> will set the SameSite attribute to Strict for strict same site enforcement.

-cookie.secure
  --Specifies the boolean value for the Secure Set-Cookie attribute.
  --When truthy, the Secure attribute is set, otherwise it is not.
  --By default, the Secure attribute is not set.
  --Be careful when setting this to true, as compliant clients will not send the cookie back to the server in the future if the browser does not have an HTTPS connection.
  --Please note that secure: true is a recommended option.
  --However, it requires an https-enabled website, i.e., HTTPS is necessary for secure cookies.
  --If secure is set, and you access your site over HTTP, the cookie will not be set.

-genid
  --Function to call to generate a new session ID.
  --Provide a function that returns a string that will be used as a session ID.
  --The function is given req as the first argument if you want to use some value attached to req when generating the ID.
  --The default value is a function which uses the uid-safe library to generate IDs.
  --Be careful to generate unique IDs so your sessions do not conflict.
```
app.use(session({
  genid: function(req) {
    return genuuid() // use UUIDs for session IDs
  },
  secret: 'keyboard cat'
}))
```

-name
  --The name of the session ID cookie to set in the response (and read from in the request).
  --The default value is 'connect.sid'.

-resave
  --변경이 없어도 저장
  --Forces the session to be saved back to the session store, even if the session was never modified during the request.
  --Depending on your store this may be necessary, but it can also create race conditions where a client makes two parallel requests to your server and changes made to the session in one request may get overwritten when the other request ends, even if it made no changes (this behavior also depends on what store you’re using).
  --The default value is true, but using the default has been deprecated, as the default

will change in the future.
--Please research into this setting and choose what is appropriate to your use-case.
--Typically, you'll want false.
--How do I know if this is necessary for my store?
---The best way to know is to check with your store if it implements the touch method.
---If it does, then you can safely set resave: false.
---If it does not implement the touch method and your store sets an expiration date on stored sessions, then you likely need resave: true.

-saveUninitialized
--session 초기화 전에도 저장
--Forces a session that is "uninitialized" to be saved to the store.
--A session is uninitialized when it is new but not modified.
--Choosing false is useful for implementing login sessions, reducing server storage usage, or complying with laws that require permission before setting a cookie.
--Choosing false will also help with race conditions where a client makes multiple parallel requests without a session.
--The default value is true, but using the default has been deprecated, as the default will change in the future.
--Please research into this setting and choose what is appropriate to your use-case.

-secret
--Required option
--session ID 서명
--This is the secret used to sign the session ID cookie.
--This can be either a string for a single secret, or an array of multiple secrets.
--If an array of secrets is provided, only the first element will be used to sign the session ID cookie, while all the elements will be considered when verifying the signature in requests.

-store
--session 저장소
--The session store instance, defaults to a new MemoryStore instance.

-unset
--Control the result of unsetting req.session (through delete, setting to null, etc.).
--The default value is 'keep'.
--'destroy' The session will be destroyed (deleted) when the response ends.
--'keep' The session in the store will be kept, but modifications made during the request are ignored and not saved.

5)properties
-req.session
--To store or access session data, simply use the request property req.session, which is (generally) serialized as JSON by the store, so nested objects are typically fine.

```
// Use the session middleware
app.use(session({ secret: 'keyboard cat', cookie: { maxAge: 60000 }}))

// Access the session as req.session
```

```
        app.get('/', function(req, res, next) {
          var sess = req.session;
          if (sess.views) {
            sess.views++;
            res.setHeader('Content-Type', 'text/html');
            res.write('<p>views: ' + sess.views + '</p>');
            res.write('<p>expires in: ' + (sess.cookie.maxAge / 1000) + 's</p>');
            res.end();
          } else {
            sess.views = 1;
            res.end('welcome to the session demo. refresh!');
          }
        });

    -req.session.id
      --Each session has a unique ID associated with it.
      --This property will contain the session ID and cannot be modified.

    -req.session.cookie
      --Each session has a unique cookie object accompany it.
      --This allows you to alter the session cookie per visitor.
      --For example we can set req.session.cookie.expires to false to enable the cookie to
      remain for only the duration of the user-agent.

    -Cookie.maxAge
      --Alternatively req.session.cookie.maxAge will return the time remaining in
      milliseconds, which we may also re-assign a new value to adjust the .expires property
      appropriately.
      The following are essentially equivalent

        var hour = 3600000;
        req.session.cookie.expires = new Date(Date.now() + hour);
        req.session.cookie.maxAge = hour;

      --For example when maxAge is set to 60000 (one minute), and 30 seconds has
      elapsed it will return 30000 until the current request has completed, at which time
      req.session.touch() is called to reset req.session.maxAge to its original value.
      --req.session.cookie.maxAge // => 30000

    -req.sessionID
      --To get the ID of the loaded session, access the request property req.sessionID.
      --This is simply a read-only value set when a session is loaded/created.

  6)Methods
    -Session.regenerate(callback)
      --To regenerate the session simply invoke the method.
      --Once complete, a new SID and Session instance will be initialized at req.session and
      the callback will be invoked.

        req.session.regenerate(function(err) {
          // will have a new session here
```

```
1877          });
1878
1879      -Session.destroy(callback)
1880        --Destroys the session and will unset the req.session property.
1881        --Once complete, the callback will be invoked.
1882
1883          req.session.destroy(function(err) {
1884            // cannot access session here
1885          });
1886
1887      -Session.reload(callback)
1888        --Reloads the session data from the store and re-populates the req.session object.
1889        --Once complete, the callback will be invoked.
1890
1891          req.session.reload(function(err) {
1892            // session updated
1893          });
1894
1895      -Session.save(callback)
1896        --Save the session back to the store, replacing the contents on the store with the
            contents in memory (though a store may do something else–consult the store's
            documentation for exact behavior).
1897        --This method is automatically called at the end of the HTTP response if the session
            data has been altered (though this behavior can be altered with various options in the
            middleware constructor).
1898        --Because of this, typically this method does not need to be called.
1899        --There are some cases where it is useful to call this method, for example, long- lived
            requests or in WebSockets.
1900
1901          req.session.save(function(err) {
1902            // session saved
1903          })
1904
1905      -Session.touch()
1906        --Updates the .maxAge property.
1907        --Typically this is not necessary to call, as the session middleware does this for you.
1908
1909    7)Example
1910        var express = require('express');
1911        var parseurl = require('parseurl');
1912        var session = require('express-session');
1913
1914        var app = express();
1915
1916        app.use(session({
1917          secret: 'keyboard cat',
1918          resave: false,
1919          saveUninitialized: true
1920        }));
1921
1922        app.use(function (req, res, next) {
```

```
1923          var views = req.session.views;
1924
1925          if (!views) {
1926            views = req.session.views = {};
1927          }
1928
1929          // get the url pathname
1930          var pathname = parseurl(req).pathname;
1931
1932          // count the views
1933          views[pathname] = (views[pathname] || 0) + 1;
1934
1935          next();
1936        });
1937
1938        app.get('/foo', function (req, res, next) {
1939          res.send('you viewed this page ' + req.session.views['/foo'] + ' times');
1940        });
1941
1942        app.get('/bar', function (req, res, next) {
1943          res.send('you viewed this page ' + req.session.views['/bar'] + ' times');
1944        });
1945
1946        app.listen(3000);
1947        ------------------------------------------------
1948        http://localhost:3000/foo
1949        you viewed this page 3 times
1950
1951    8)Lab
1952      <sessiondemo.js>
1953        var express = require('express');
1954        var session = require('express-session');
1955        var bodyParser = require('body-parser');
1956
1957        var app = express();
1958
1959        app.use(express.static(__dirname));
1960        app.use(bodyParser.json());
1961        app.use(bodyParser.urlencoded({ extended: false }));
1962
1963        app.use(session({
1964          secret: 'my key',
1965          resave: true,
1966          saveUninitialized: true
1967        }));
1968
1969        app.get('/product', function (req, res) {
1970          if(req.session.user){
1971               res.redirect('/product.html');
1972          }else{
1973               res.redirect('/login.html');
```

```
1974          }
1975      });
1976
1977      app.post('/login', function(req, res){
1978          var paramId = req.body.id;
1979          var paramPassword = req.body.passwd;
1980
1981          if(req.session.user){
1982              //이미 로그인된 상태
1983              res.redirect('/product.html');
1984          }else{
1985              //세션 저장
1986              req.session.user = {
1987                  id : paramId,
1988                  name : '한지민',
1989                  authorized : true
1990              }
1991              res.writeHead('200', {'Content-Type' : 'text/html;charset=utf-8'});
1992              res.write('<h1>로그인 성공</h1>');
1993              res.write('<div>Param Id : ' + paramId + "</div>");
1994              res.write('<div>Param password : ' + paramPassword + "</div>");
1995              res.write('<br /><br /><a href="/product">상품페이지로 이동하기</a>');
1996              res.end();
1997          }
1998      });
1999
2000      app.get('/logout', function(req, res){
2001          if(req.session.user){
2002              //로그인된 상태
2003              req.session.destroy(function(err){
2004                  if(err) throw err;
2005                  res.redirect('/login.html');
2006              });
2007          }else{
2008              //로그인 안된상태
2009              res.redirect('/login.html');
2010          }
2011      });
2012
2013      app.listen(3000);
2014
2015    <login.html>
2016      <!doctype html>
2017      <html>
2018          <head>
2019              <meta charset='utf-8' />
2020              <title>로그인 페이지</title>
2021          </head>
2022          <body>
2023              <h1>로그인</h1>
2024              <form method='post' action='/login'>
```

```
                    <p>
                        <label for='txtId'>아이디</label>
                        <input type='text' id='txtId' name='id' />
                    </p>
                    <p>
                        <label for='txtPasswd'>패스워드</label>
                        <input type='password' id='txtPasswd' name='passwd' />
                    </p>
                    <p>
                        <input type='submit' value='전송' />
                    </p>
                </form>
            </body>
        </html>

    <product.html>
        <!doctype html>
        <html>
            <head>
                <meta charset='utf-8' />
                <title>상품 페이지</title>
            </head>
            <body>
                <h3>상품정보 페이지</h3>
                <hr />
                <p>로그인 후 볼 수 있는 상품 정보 페이지</p>
                <a href="logout">로그 아웃</a>
            </body>
        </html>
    ------------------------------------------------------------
    $ node sessiondemo.js
    1)http://localhost:3000/product.html
    만일 세션이 있으면 볼 수 있고 세션이 없다면 login.html로 redirect
    2)http://localhost:3000/login.html
    아이디와 패스워드 입력하면 라우팅은 app.post('/login')으로 이동
    3)로그인 성공 메시지 창에서 <상품페이지로 이동하기>을 클릭하면 비로소 product.html로 이동함
    4)상품 페이지에서 <로그 아웃> 링크를 클릭하면 모든 세션 정보를 지우고 login.html로 이동함.


24. 파일 업로드 기능 만들기
  1)Express에서 파일 업로드 기능을 통해 사진이나 파일을 업로드 할 수 있다.
  2)파일을 업로드할 때는 멀티 파트(multipart) 포맷으로 된 파일 업로드 기능을 사용하며 파일 업로드 상태 등을
  확인할 수 있다.
  3)Express에서 파일 업로드를 하기 위한 미들웨어는 보통 multer를 사용한다.
  4)Multer is a node.js middleware for handling multipart/form-data.
  5)Multer is primarily used fo uploading files.
  6)http://expressjs.com/en/resources/middleware/multer.html
  7)Multer will not process any form which is not multipart (multipart/form-data).
  8)Installation
    $ npm install --save multer
      `-- multer@1.3.0
```

```
        +-- append-field@0.1.0
        +-- busboy@0.2.14
        | +-- dicer@0.2.5
        | | +-- readable-stream@1.1.14
        | | | +-- isarray@0.0.1
        | | | `-- string_decoder@0.10.31
        | | `-- streamsearch@0.1.2
        | `-- readable-stream@1.1.14
        |   +-- isarray@0.0.1
        |   `-- string_decoder@0.10.31
        +-- concat-stream@1.6.0
        | `-- typedarray@0.0.6
        +-- mkdirp@0.5.1
        | `-- minimist@0.0.8
        +-- object-assign@3.0.0
        `-- xtend@4.0.1

9)Usage
  var express = require('express');
  var multer  = require('multer');

10)API
  -File information
    --fieldname  : Field name specified in the form
    --originalname: Name of the file on the user's computer
    --encoding : Encoding type of the file
    --mimetype : Mime type of the file
    --size  : Size of the file in bytes
    --destination   : The folder to which the file has been saved
    --filename : The name of the file within the destination
    --path: The full path to the uploaded file
    --buffer :  A Buffer of the entire file

  -multer(options)
    --Multer accepts an options object
    --The most basic of which is the dest property.
    --In case you omit the options object, the files will be kept in memory and never
    written to disk.
    --By default, Multer will rename the files so as to avoid naming conflicts.
    --The renaming function can be customized according to your needs.
    --options
      ---dest or storage  : Where to store the files
      ---fileFilter   : Function to control which files are accepted
      ---limits : Limits of the uploaded data
      --preservePath   : Keep the full path of files instead of just the base name
        var upload = multer({ dest: 'uploads/' })

    --limits
      ---An object specifying the size limits of the following optional properties.
      ---fieldNameSize: Max field name size(default:100 bytes)
      ---fieldSize   : Max field value size(default:1MB)
```

```
        ---fields : Max number of non-file fields(default: Infinity)
        ---fileSize : For multipart forms, the max file size (in bytes)  (default:Infinity)
        ---files  : For multipart forms, the max number of file fields  (default:Infinity)
        ---parts : For multipart forms, the max number of parts (fields + files)
        (default:Infinity)
        ---headerPairs : For multipart forms, the max number of header key=>value pairs
        to parse(default:2000)


11)Lab
  <upload.jade>
    doctype html
    html
      head
        meta(charset='utf-8')
        title File Upload Demo
      body
        form(action='/upload', method='post', enctype='multipart/form-data')
          input(type='file', name='userfile')
          input(type='submit')

  <multerdemo.js>
    var express = require('express');
    var multer = require('multer');
    var app = express();
    app.set('views', __dirname);
    app.set('view engine', 'jade');

    //multer
    var storage = multer.diskStorage({
        destination : function(req, file, callback){
            callback(null, './uploads');
        },
        filename : function(req, file, callback){
            callback(null, file.originalname + '-' + Date.now());
        }
    });
    var upload = multer({storage : storage});

    app.use(express.static(__dirname));

    app.get('/upload', function(req, res){
        res.render('upload');
    });
    app.post('/upload', upload.single('userfile'), function(req, res){
        res.send('Upload Success : ' + req.file.originalname);
        console.log(req.file);
    });

    app.listen(3000);
```

25. 환경 설정
1)node.js의 동작 환경을 개발용(development)와 제품용(production)으로 설정가능
2)개발과정에서 동작 과정과 성능 등을 측정할 수 있는 다양한 정보와 에러 로그를 많이 남기도록 작성할 수 있고
3)제품용으로 동작 중일 때에는 개발용 로그가 사용자에게 노출되지 않도록 하는 것이 좋다.
4)기본값은 개발용(development)이다.
5)다음은 제품으로 환경 설정하고, 어플리케이션을 동작시키는 명령이다.

```
$ NODE_ENV=production node app
```

6)Windows 에서는 set을 이용해서 환경 설정한다.

```
set NODE_ENV=production
$ node myapp.js
```

7)In Linux

```
$ NODE_ENV=production node myapp.js
$ NODE_ENV=development node myapp.js
```

8)환경 설정값 읽기

```
app.get('env');
```

9)환경별 에러 처리 코드

```
if(app.get('env') === 'development'){
  app.use(function(err, req, res, next){
    res.end(err.stack);
  });
}else{
  app.use(function(err, req, res, next){
    res.status(err.code || 500);
    res.end('잠시 후 다시 시도해 주세요');
  });
}
```

10)Lab

```
In Windows,
C:\NodeHome>set NODE_ENV=developement
C:\NodeHome>node test.js

In Linux,
$ NODE_ENV=development node test.js

<test.js>
var express = require('express');
var app = express();

app.get('/', function(req, res){
     res.end(5678);
});

if(app.get('env') === 'development'){
     app.use(function(err, req, res, next){
          res.end(err.stack);
```

```
        });
    }else{
        app.use(function(err, req, res, next){
            res.status(err.code || 500);
            res.end('Try again later.');
        });
    }
    app.listen(8194);
    ----------------------------------------------------
    TypeError: First argument must be a string or Buffer
  at ServerResponse.OutgoingMessage.end (_http_outgoing.js:558:11)
  at C:\NodeHome\test.js:5:8
  at Layer.handle [as handle_request]
  (C:\NodeHome\node_modules\express\lib\router\layer.js:95:5)
  at next (C:\NodeHome\node_modules\express\lib\router\route.js:137:13)
  at Route.dispatch (C:\NodeHome\node_modules\express\lib\router\route.js:112:3)
  at Layer.handle [as handle_request]
  (C:\NodeHome\node_modules\express\lib\router\layer.js:95:5)
  at C:\NodeHome\node_modules\express\lib\router\index.js:281:22
  at Function.process_params
  (C:\NodeHome\node_modules\express\lib\router\index.js:335:12)
  at next (C:\NodeHome\node_modules\express\lib\router\index.js:275:10)
  at expressInit (C:\NodeHome\node_modules\express\lib\middleware\init.js:40:5)

    In Windows,
    C:\NodeHome>set NODE_ENV=production
    C:\NodeHome>node test.js

    In Linux,
    $ NODE_ENV=development node test.js
    ---------------------------------------------------
    Try again later.


26. Eclipse + Express로 Web Server 만들기(with Express v3.x)
  1)Eclipse와 Express를 사용하여 새로운 프로젝트를 만들면 프로젝트 안에 여러 가지 파일들이 자동으로 만들어
  진다.
  2)Eclipse로 새로운 Express Project 생성하기
    -Eclipse > Project Explorer > New > Node.js Express Project
    -Project name : ExpressDemo

  3)app.js
    -자동으로 만들어진 파일의 시작점 역할
      var express = require('express')
        , routes = require('./routes')
        , user = require('./routes/user')
        , http = require('http')
        , path = require('path');

    -먼저 상대경로를 지정하기 위해 각 모듈은 './'가 붙어있다.
    -즉, 내장 모듈이거나 npm으로 설치한 모듈은 경로 없이 이름으로만 지정하도록 설정되어 있다.
```

-직접 이 프로젝트 안에 만들어 넣은 모듈은 반드시 상대경로를 넣는다.
-그 아래에는 express() 반환하여 사용하려고 하는 app 즉, Express 서버 객체가 있다.
var app = express();

-이 Express Server 객체가 갖고 있는 주요 메소드들은 아래와 같다.
--set(name, value) : 서버 설정을 위한 속성을 지정, set()으로 지정하고 get()으로 뽑아서 확인한다.
--get(name) : 서버 설정을 위해 지정한 속성을 뽑는다.
--use([path,] function [, function...]) : 미들웨어 함수를 사용한다.
--get([path,] function) : 특정 패스로 요청된 정보를 처리한다.

app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'jade');

-use() 메소드를 사용한 미들웨어 설정 확인
--1. Client가 요청한다.
--2. app 객체(Express Server 객체)가 미들웨어를 use()로 설정한다.
--3. 첫 미들웨어를 실행한다.
--4. next()를 실행한다.
--5. 다음 미들웨어가 실행된다.
--6. next()를 실행한다.
--7. 마지막 미들웨어가 실행된다.
--8. next()를 실행한다.
--9. 라우터부분에서 요청패턴('/'나 '/users')이 맞으면 app 객체에서 미리 설정한 라우터 설정을 보고 라우팅한 다음, 클라이언트에게 응답한다.

4)app 실행하기
-In app.js > right-click > Run As > Node Application

In Console
Express server listening on port 3000

In Browser
http://localhost:3000

Express

Welcome to Express

In Browser
http://localhost:3000/users
respond with a resource

5)현재 Eclipse에 Plug-in한 ENIDE는 Express 버전이 3.x 이다.
6)따라서 최신 버전의 Express를 사용하기 위해서는 다른 툴을 사용하거나 다른 방법을 모색해야 한다.

27. Express 3으로 간단한 폼 전송 웹 사이트 만들기
1)Eclipse > Project Explorer > New > Node.js Express Project
2)Project name : joinin
In Windows

```
  Project properties > Resource > Text file encoding > Other : UTF-8
3)package.json
  "dependencies" : {
      "express" : "*",
      "jade" : "*"
  }

4)package.json > right-click > Run As >  npm install

5)routes/index.js
  -아래 코드 추가
  exports.form = function(req, res){
    res.render('join-form', {title : '회원가입'});
  };

6)app.js
  -아래 코드 추가
  app.get('/', routes.index);
  app.get('/join', routes.form);
  app.get('/users', user.list);

7)views/layout.jade
  -아래 코드 추가
    title= title
    link(rel='stylesheet', href='/stylesheets/style.css')
    script(src='/javascripts/jquery-3.2.1.min.js')

8)views/join-form.jade
  extends layout
  block content
    h1 회원가입
    p 회원에 가입해주세요.
    form(id='joinForm', method='POST', action='/join')
      p
        label(for='name') Name :
        input(type='text', name='name', id='name')
      p
        label(for='email') Email :
        input(type='text', name='email', id='email')
      input(type='button', value='전송')
    script.
      $(document).ready(function(){
        $("input[type='button']").click(function(){
          if($('#name').val() === '' || $('#email').val() === ''){
            alert('이름과 이메일을 입력해 주세요.');
            return  false;
          }
          $('#joinForm').submit();
        });
      });
```

9)http://localhost:3000/join으로 확인

10)routes/index.js
-아래 코드 추가
```
exports.join = function(req, res){
res.render('join-result', {
  username: req.body.name,
  useremail : req.body.email,
  title : 'Express'
});
};
```

11)app.js
-아래 코드 추가
```
app.get('/', routes.index);
app.get('/join', routes.form);
app.post('/join', routes.join);
app.get('/users', user.list);
```

12)views/join-result.jade
```
extends layout
block content
  h2 회원가입완료
  p 가입한 회원 정보는 아래와 같습니다.
  div
    ul
      li 이름 : #{username}
      li 이메일 : #{useremail}
```

28. Express Generator

1)express 어플리케이션이 기본 코드를 작성해주는 유틸리티
- Express 4.x 이상

2)https://expressjs.com/en/starter/generator.html

3)Install
```
$ npm install express-generator -g
     |
     |-----express-generator@4.15.0
```

4)Display the command options with the -h option.
```
$ express -h

-Options:
  -h, --help          output usage information
--version       output the version number
  -e, --ejs           add ejs engine supportex
  --pug           add pug engine support
--hbs           add handlebars engine support
  -H, --hogan         add hogan.js engine support
  -v, --view <engine>  add view <engine> support
```

```
    (dust|ejs|hbs|hjs|jade|pug|tw|vash) (defaults to jade)
    -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sa)
    (defaults to plain css)
  --git          add .gitignore
    -f, --force        force on non-empty directory

5)다음 예제는 myapp이라는 express 어플리케이션을 생성한다.
  -어플리케이션은 myapp이라는 폴더안에 생성된다.

  $ express myapp
    warning: the default view engine will not be jade in future releases
    warning: use `--view=jade' or `--help' for additional options


     create : myapp
     create : myapp/package.json
     create : myapp/app.js
     create : myapp/public
     create : myapp/routes
     create : myapp/routes/index.js
     create : myapp/routes/users.js
     create : myapp/views
     create : myapp/views/index.jade
     create : myapp/views/layout.jade
     create : myapp/views/error.jade
     create : myapp/bin
     create : myapp/bin/www
     create : myapp/public/javascripts
     create : myapp/public/images
     create : myapp/public/stylesheets
     create : myapp/public/stylesheets/style.css

     install dependencies:
       > cd myapp && npm install

     run the app:
       > SET DEBUG=myapp:* & npm start

6)myapp폴더로 이동하고 npm install을 이용해서 필요한 모듈을 설치한다.
  $ cd myapp
  $ npm install --save
    npm WARN deprecated jade@1.11.0: Jade has been renamed to pug, please install
    the latest version of pug instead of jade
    npm WARN deprecated transformers@2.1.0: Deprecated, use jstransformer
    myapp@0.0.0 C:\NodeHome\myapp
    +-- body-parser@1.17.2
    | +-- bytes@2.4.0
    | +-- content-type@1.0.2
    | +-- debug@2.6.7
    | +-- depd@1.1.0
    | +-- http-errors@1.6.1
```

```
2472        | | `-- inherits@2.0.3
2473        | +-- iconv-lite@0.4.15
2474        | +-- on-finished@2.3.0
2475        | | `-- ee-first@1.1.1
2476        | +-- qs@6.4.0
2477        | +-- raw-body@2.2.0
2478        | | `-- unpipe@1.0.0
2479        | `-- type-is@1.6.15
2480        |   +-- media-typer@0.3.0
2481        |   `-- mime-types@2.1.15
2482        |     `-- mime-db@1.27.0
2483        +-- cookie-parser@1.4.3
2484        | +-- cookie@0.3.1
2485        | `-- cookie-signature@1.0.6
2486        +-- debug@2.6.8
2487        | `-- ms@2.0.0
2488        +-- express@4.15.3
2489        | +-- accepts@1.3.3
2490        | | `-- negotiator@0.6.1
2491        | +-- array-flatten@1.1.1
2492        | +-- content-disposition@0.5.2
2493        | +-- debug@2.6.7
2494        | +-- encodeurl@1.0.1
2495        | +-- escape-html@1.0.3
2496        | +-- etag@1.8.0
2497        | +-- finalhandler@1.0.3
2498        | | `-- debug@2.6.7
2499        | +-- fresh@0.5.0
2500        | +-- merge-descriptors@1.0.1
2501        | +-- methods@1.1.2
2502        | +-- parseurl@1.3.1
2503        | +-- path-to-regexp@0.1.7
2504        | +-- proxy-addr@1.1.4
2505        | | +-- forwarded@0.1.0
2506        | | `-- ipaddr.js@1.3.0
2507        | +-- range-parser@1.2.0
2508        | +-- send@0.15.3
2509        | | +-- debug@2.6.7
2510        | | +-- destroy@1.0.4
2511        | | `-- mime@1.3.4
2512        | +-- serve-static@1.12.3
2513        | +-- setprototypeof@1.0.3
2514        | +-- statuses@1.3.1
2515        | +-- utils-merge@1.0.0
2516        | `-- vary@1.1.1
2517        +-- jade@1.11.0
2518        | +-- character-parser@1.2.1
2519        | +-- clean-css@3.4.26
2520        | | +-- commander@2.8.1
2521        | | | `-- graceful-readlink@1.0.1
2522        | | `-- source-map@0.4.4
```

```
2523        | |   `-- amdefine@1.0.1
2524        | +-- commander@2.6.0
2525        | +-- constantinople@3.0.2
2526        | | `-- acorn@2.7.0
2527        | +-- jstransformer@0.0.2
2528        | | +-- is-promise@2.1.0
2529        | | `-- promise@6.1.0
2530        | |   `-- asap@1.0.0
2531        | +-- mkdirp@0.5.1
2532        | | `-- minimist@0.0.8
2533        | +-- transformers@2.1.0
2534        | | +-- css@1.0.8
2535        | | | +-- css-parse@1.0.4
2536        | | | `-- css-stringify@1.0.5
2537        | | +-- promise@2.0.0
2538        | | | `-- is-promise@1.0.1
2539        | | `-- uglify-js@2.2.5
2540        | |   +-- optimist@0.3.7
2541        | |   | `-- wordwrap@0.0.3
2542        | |   `-- source-map@0.1.43
2543        | +-- uglify-js@2.8.28
2544        | | +-- source-map@0.5.6
2545        | | +-- uglify-to-browserify@1.0.2
2546        | | `-- yargs@3.10.0
2547        | |   +-- camelcase@1.2.1
2548        | |   +-- cliui@2.1.0
2549        | |   | +-- center-align@0.1.3
2550        | |   | | +-- align-text@0.1.4
2551        | |   | | | +-- kind-of@3.2.2
2552        | |   | | | | `-- is-buffer@1.1.5
2553        | |   | | | +-- longest@1.0.1
2554        | |   | | | `-- repeat-string@1.6.1
2555        | |   | | `-- lazy-cache@1.0.4
2556        | |   | +-- right-align@0.1.3
2557        | |   | `-- wordwrap@0.0.2
2558        | |   +-- decamelize@1.2.0
2559        | |   `-- window-size@0.1.0
2560        | +-- void-elements@2.0.1
2561        | `-- with@4.0.3
2562        |   +-- acorn@1.2.2
2563        |   `-- acorn-globals@1.0.9
2564        +-- morgan@1.8.2
2565        | +-- basic-auth@1.1.0
2566        | `-- on-headers@1.0.1
2567        `-- serve-favicon@2.4.3
2568          `-- safe-buffer@5.0.1
2569
2570    7)Eclipse project 만들기
2571        --Project Explorer > right-click > New > Node.js Express Project
2572        --Project name : myapp > Finish
2573
```

```
2574        --/bin/www,
2575          ---right-click > Run As > Node Application
2576        --In Browser,
2577          ---http://localhost:3000
2578        Express
2579        Welcome to Express
2580
2581    8)myapp에 생성된 폴더와 파일들 목록
2582      -app.js : express 설정, 일종의 express의 main 역할하는 파일, 라우팅 등.
2583      -bin/www : HTTP 서버와 환경 설정
2584      -public folder : 이미지, 자바스트립트, 스타일과 같은 웹 페이지용 폴더
2585      -routes folder : express 의 라우터 코드가 생성된 폴더
2586      -views folder : 템플릿 파일이 설정된 폴더
2587
2588    9)middleware 직접 만들어 보기
2589      -myapp > right-click > New > Javascript Source File
2590      -File name : app2.js
2591
2592        var express = require('express')
2593          , http = require('http')
2594          , path = require('path');
2595
2596        var app = express();
2597
2598        app.use(function(req, res, next){
2599          console.log(req.method + " : " + req.url);
2600
2601          res.writeHead('200', {'Content-Type' : 'text/html;charset=utf-8'});
2602          res.end("<h1>Hello, World Express 4</h1>");
2603        });
2604
2605        module.exports = app;
2606
2607      -/bin/www
2608        var app = require('../app');  --> var app = require('../app2');
2609
2610      -Test
2611        http://localhost:3000
2612
2613    10)간단한 로그인페이지 처리하기
2614      -/app2.js
2615        app.use(express.static(path.join(__dirname, 'public')));
2616      -/public/login.html
2617        <!DOCTYPE html>
2618        <html>
2619        <head>
2620        <meta charset="UTF-8">
2621        <title>Login Page</title>
2622        </head>
2623        <body>
2624          <h1>Login Page</h1>
```

```
        <form method="post" action="/login">
          <p>ID : <input type="text" name="id" /></p>
          <p>PWD : <input type="password" name="pwd" /></p>
          <input type="submit" value="전송" />
        </form>
      </body>
      </html>

    -post 방식으로 /login을 처리하기 위해 app2.js에 코드 추가
      var express = require('express')
        , http = require('http')
        , path = require('path');
      var bodyParser = require('body-parser');

      var app = express();
      app.use(express.static(path.join(__dirname, 'public')));
      app.use(bodyParser.urlencoded({ extended: false }));

      app.get('/', function(req, res, next){
        console.log(req.method + " : " + req.url);

        res.writeHead('200', {'Content-Type' : 'text/html;charset=utf-8'});
        res.end("<h1>Hello, World Express 4</h1>");
      });

      app.post('/login', function(req, res, next){
        var id = req.body.id;
        var pwd = req.body.pwd;
        res.writeHead('200', {'Content-Type' : 'text/html;charset=utf-8'});
        res.write('<div>아이디 : ' + id + '</div>');
        res.write('<div>Passwod : ' + pwd + '</div>');
        res.end();
      });
      module.exports = app;


29. Express 4를 이용한 간단한 웹 페이지 만들기
  1)/views/layout.jade에 jquery 코드 추가하기
    doctype html
    html
      head
        title= title
        link(rel='stylesheet', href='/stylesheets/style.css')
        script(src='/javascripts/jquery-3.2.1.min.js')                    //코드 추가
      body
        block content

  2)/views/join-form.jade 파일 생성하기
    -/views > right-click > New > File
    -File name : join-form.jade > Finish
```

```
2676        extends layout
2677
2678        block content
2679          h3 회원가입
2680          p 회원에 가입해주세요.
2681          form(id='joinForm', method='POST', action='/join')
2682            p
2683              label(for='txtName') 이름 :
2684              =" "
2685              input(type='text', name='txtName', id='txtName')
2686            p
2687              label(for='txtEmail') Email :
2688              =" "
2689              input(type='text', name='txtEmail', id='txtEmail')
2690            p
2691              input(type='submit', value='전송')
2692
2693          script.
2694            $(document).ready(function(){
2695              $('#joinForm').submit(function(){
2696                if($.trim($('#txtName').val()) === '' || $.trim($('#txtEmail').val()) === ''){
2697                  alert('이름과 Email을 입력해 주세요.');
2698                  return false;
2699                }
2700              });
2701            });
2702
2703    3)/routes/index.js 수정하기
2704        var express = require('express');
2705        var router = express.Router();
2706
2707        /* GET home page. */
2708        router.get('/join', function(req, res, next) {
2709          res.render('join-form', { title: 'Express' });
2710        });
2711
2712        module.exports = router;
2713
2714    4)http://localhost:3000/join
2715      -/bin/www
2716        right-click > Run As > Node Application
2717     -입력폼
2718
2719    5)이름과 이메일을 입력 후 전송하면 /join Post가 발생
2720      -/routes/index.js 코드 추가
2721        router.post('/join', function(req, res, next){
2722          res.render('join-result', {
2723           username : req.body.txtName
2724           , useremail : req.body.txtEmail
2725           , title : 'Express'
2726          });
```

```
2727        })
2728
2729    6)post로 넘어온 값 처리하기
2730      -/views/join-result.jade 파일 생성하기
2731
2732      extends layout
2733
2734      block content
2735        h3 회원 가입 완료
2736        p 가입한 회원 정보는 아래와 같습니다.
2737        div
2738          ul
2739            li 이름 : #{username}
2740            li Email : #{useremail}
2741
2742    7)Test
2743      -http://localhost:3000/join
```