1. 개요
  1)오늘날 대부분 네트워크를 이용한 통신은 웹(World wide web)을 기반으로 하는 콘텐츠를 주고 받는다.
  2)웹은 HTTP(Hypertext Transfer Protocol)을 기반으로 Hypertext를 주고 받는다.
  3)Hypertext는 hyperlink를 이용해서 다른 문서와 논리적으로 연결된 문서를 사용한다.

2. HTTP 통신의 특징
  1)리소스의 위치 : URL 기본 모듈 참조
  2)HTTP 프로토콜은 데이터 송수신을 위하여 메세지의 구조를 정의하고 request와 response 프로토콜로 구성되어 있다.
  3)GET : 웹서버로부터 원하는 웹 문서를 요청
  4)POST : 클라이언트가 웹 서버에 데이터를 전달하는 방식
  5)HEAD : 웹 문서의 본문을 제외한 정보를 요청
  6)https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
  7)구조를 알기 위한 필요한 tools
    -Paros
    -Fiddler2

3. Paros
  1)Paros 설치하기
    -https://sourceforge.net/projects/paros/
    -64bit OS라 하더라도 32-bit JRE(https://www.java.com/ko/download/manual.jsp)를 설치해야 함
    -이미 JAVA_HOME이 설정되어 있으면 지우고 JRE_HOME을 C:\Program Files (x86)\Java\jre1.8.0_131로 설정
    -바탕 화면의 paros실행파일 속성창에서 실행 경로 변경하기
    "C:\Program Files (x86)\Java\jre1.8.0_131\bin\javaw.exe" -jar paros.jar

  2)Paros 소개
    - Web Proxy Tool로서 web site의 취약점 분석과 웹 해킹 도구로 사용되는 툴
    - 내 컴퓨터의 가상의 proxy를 사용해서 웹서버와 클라이언트 사이에서 서로 오고가는 패킷, 즉 데이터와 쿠키 등의 값들을 가운데서 일정 시간 정지시켜 값을 변경해 웹서버로 보낼 수 있다.
    - 인터넷 할 때 내가 보내는 정보와 내가 받는 정보를 수정할 수 있게 해 주는 툴
    - Java기반 프로그램이기 때문에 미리 Java가 설치되어 있어야 한다.

  3)Paros 환경설정
    -In IE, [Internet Options] > Connections Tag > [LAN settings]
    -Local Area Network (LAN) Settings > Proxy server section
    -Check [Use a proxy server...] Address : 127.0.0.1 Port : 8080 > Click [OK] button.
    -In Paros, Tools > Options > Local proxy > Address : 127.0.0.1 Port : 8080

4. Fiddler
  1)Visit http://www.telerik.com/fiddler
  2)Download
  3)Installation

5. HTTP Request
  1)request 메세지는 다음과 같은 요소로 구성되며, 각 요소는 CRLF로 구분한다. Message body는 Entity라

고도 한다.

2)구성 요소

-Request Line
-Request Header fields
-Empty Line
-Message Body

3)Request Packet

GET https://www.webtime.co.kr/favicon.ico HTTP/1.1 <--request line
---below Header Fields----------
Host: www.webtime.co.kr
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36 Paros/3.2.13
Accept: image/webp,image/*,*/*;q=0.8
Referer: https://www.webtime.co.kr/
Accept-Encoding: sdch, br
Accept-Language: ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4
Cookie: ASP.NET_SessionId=114oi20y55vqppq5tubpps0j
---------CRLF(Empty line)----------
Mesage Body

4)Request Line

GET https://www.webtime.co.kr/favicon.ico HTTP/1.1
-Request method
-Request URL
-HTTP version

5)Request Method

-GET : Request URI를 이용하여 서버에 자료를 전송하는 방식으로 URI만 해석하기 때문에 속도가 빠르다. 주소 서버에 데이터를 전송하기 보다는 리소스를 요청하는 용도로 사용한다. 별도의 요청 메소드를 작성하지 않으면 GET 방식으로 요청한다.
-POST : Request body에 자료를 전송하는 방식으로 서버가 요청 바디를 분석해야 하므로 GET에 비해 속도가 느리다. 대신 서버로 보내는 데이터의 양의 제한이 없다.
-PUT : 서버에 저장하겠다는 요청이다. 서버에 이미 존재하는 내용이면 수정한다.
-DELETE : 요청 URI를 사용하여 서버에 존재하는 리소소를 삭제
-HEAD : GET 요청 메소드와 같으나, 요청 내용으로 헤더만 포함되어 전달한다. 주로 클라이언트에 보관되어 있는 캐시 엔티티를 갱신하는데 사용한다.
-TRACE : 바로 전에 요청한 내용의 추적을 요청한다.

6)Request Headers

-헤더는 키 : 값 방식으로 작성
-Accept : 클라이언트가 받을 수 있는 contents 형태
-Accept-Encoding, Accept-Language, accept-Charset : 응답을 해석하는데 사용하는 인코딩과 언어와 문자셋
-Authorization : 사용자 인증 정보를 보내는 용도
-Cache-Control : 캐쉬를 제어하기 위한 필드로, 변경된 내용이 없으면 저장된 캐쉬를 사용한다. 변경 벼부를 판단하기 위해서는 최종 변경일, 컨텐츠 길이 등을 이용한다.
-Connection : 요청/응답 이후 연결 유지 여부. Keep alive는 연결을 유지하고, Close는 연결을 끊는다.
-Cookie : 서버의 응답 메세지 헤더에 Set-Cookie 헤더 필드로 설정한 내용이다.
-Content-Type : 메세지 바디의 종류

-Content-Length : 메세지 바디의 길이(바이트)
-Content-MD5 : 요청 메세지 바디의 MD5 체크섬
-Date : 요청 메세지가 발송된 시각
-Expect :
-From : 요청을 보낸 곳의 이메일 주소
-Host : 서버의 도메인 이름과 TCP 포트, HTTP 표준 포트를 사용하면 포트? 생략 가능
-If-Match, If-Modified-Since, If-None-Match : 컨텐츠가 변경되지 않았을 때의 처리다. 시각을 이용하거나 ETag라는 리소스의 일련 번호를 사용한다.
-Max-Forwards : 프록시나 게이트웨이로 전송되는 최대 횟수
-Origin :
-Praga :
-Proxy-Authorization :
-Range :
-Referer :
-TE :
-User-Agent : 클라이언트의 운영체제, 브라우저 정보
-Upgrade : 통신 중 프로토콜을 변경
-Via : 프록시나 게이트웨이를 거칠 때 사용
-Warning : 요청/응답 시 경고를 보내는 용도

7)Request 정보 전달
-Message body를 사용하지 않는 방식
--URL을 이용해서 request 정보 전달
GET 메소드, TRACE 메소드
--경로와 쿼리 스트링 사용
http://example.com/sample/test.png
http://example.com/sample?keyword=nodejs&date=2017
-Message body를 사용하는 방식1
--URLEncoded 방식
--메세지 헤더(Contents-Type)
application/x-www-form-urlencoded
--메세지 바디 : 쿼리 문자열
--예
Content-Type : application/x-www-form-urlencoded
keyword=nodejs&date=2017
-Message body를 사용하는 방식2
--JSON/XML
--메세지 헤더(Contents-Type)
application/json
--예
Content-Type : application/json
{
"name" : "nodejs", "date" : 2017
}
-Multipart를 이용한 요청 정보 전달
--binary 파일을 올리는 경우에 주로 사용
--하나의 메세지 바디에 여러 데이터를 전송하기 위해 사용
--메세지에 작성하는 데이터를 파트라고 한다.
--각 파트를 구별하기 위한 바운더리를 구분자로 사용한다.
--메세지 헤더
---Content-Type : 파트 구분자(boundary)

```
        ---multipart/form-data :boundary=frontier
      --각 파트별 컨텐츠 타입
      --파트로 구성된 바디
        --frontier
        Content-Type : text/plain

        This is the body of the message.
        --frontier
        Content-Type : application/octet-stream
        Content-Transfer-Encoding : base64

        PGh0bWw+CiAgPGHlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcy
        Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==
        --frontier
    -정리
      --request정보 전달시 메세지 바디를 사용여부
      --바디의 인코딩 방식
      --URL로 request 정보 전달시 바디 분석은 불필요
      --Query string을 이용해서 다수의 정보를 구조화해서 전달할 수 있다.
        ---이름1=값1&이름2=값2... 각 구별은 &로 한다.
        ---각 브라우저마다 다를 수 있지만 대체로 1KB를 넘지 않아야 하고, URL에 정보가 노출된다는 단점이 있다.
        ---하지만, Link를 통해서, 그리고 URL을 통해서 서버에 전송할 때 가장 좋은 방법이다.
        ---이 방법을 GET 방식이라고 한다.
      --메세지 헤더를 통해 서버에 정보를 전달하는 방법은 Query string의 단점 즉 용량 제한과 데이터 노출을 해결 할 수 있다.
        ---URL이나 메세지의 헤더의 경우에는 길이의 제한이 있기 때문에 길이가 긴 정보를 전달하기에 적당하지 않다.
        ---메세지의 바디에 정보를 기록하면 데이터가 노출되지 않도록 암호화하기에 유리하다.
        ---대신 메세지의 바디까지 분석해야 하므로 URL를 이용해서 전달하는 것보다 느리다.
        ---이 방식은 POST 방식으로 전송시 사용한다. 반드시 form의 submit을 해야 한다는 단점이 있다.


6. HTTP Response
  1)HTTP Message 구성요소
    -Response line
    -Response header
    -Empty Line
    -Response body

  2)Response Packet
      HTTP/1.1 200 OK                <----Status Line(Reponse Line)
      ----Header Fields--------------------
      Content-Length: 1009
      Content-Type: application/xml; charset=utf-8
      Last-Modified: Mon, 01 May 2017 01:45:47 GMT
      ETag: "1121413750"
      Server: Microsoft-IIS/8.5
      X-WNS-Tag: 0
      X-CMS-ACSElapsedTimeInMilliseconds: 101
      X-Diagnostic-S: AppContentService_IN_2
```

```
        X-Diagnostic-DC: northeu
        AppEx-Activity-Id: d68be07a-b1b7-46f6-9967-3db994bfee13
        X-Trace-Context: {"ActivityId":"d68be07a-b1b7-46f6-9967-3db994bfee13"}
        X-AspNet-Version: 4.0.30319
        X-Powered-By: ASP.NET
        Cache-Control: public, max-age=209
        Date: Mon, 01 May 2017 03:50:26 GMT
        Connection: keep-alive
                                          <----CRLF
        <!doctype html>
        <html>                             <--Message body
        ....
```

3)Reponse line
  HTTP/1.1 200 OK
  -Version
  -Status code
  -Status mesage

4)Status Code
  리소스의 요청에 대한 서버의 Response을 간단한 상태 코드로 전달하는데, 상태 코드는 크게 100단위의 의미가 있다.
  -1xx : 동작에 대한 코드
  -2xx : 요청 성공
  -3xx : 리다이렉션에 관련된다. 300번대의 상태 코드를 받으면 클라이언트는 이후 연결된 동작을 해야 한다.
  -4xx : 클라이언트의 요청이 잘못됐을 때 사용
  -5xx : 서버 에러에 관련된 상태 코드
  -주요 상태 코드들
    --100 : Continue 요청을 진행하고 있는 상황
    --101 : Switching Protocols : 프로토콜 변환
    --200 : HTTP_OK 요청 성공
    --201 : Created. 생성 요청 성공
    --202 : Accepted. 요청 수락, 요청 처리는 보장 안됨
    --203 : Non-authoritative Information
    --204 : Non Content...
    --301 : 영구 이동
    --302 : 임시 이동
    --400 : Bad Request. 잘못된 요청으로 포맷에 맞지 않는 요청을 할 때 주로 발생
    --401 : 권한이 없는 리소스에 접근할 때 발생
    --404 : Not Found. 리소스 없음.
    --500 : Internal Server Error. 서버 오류
    --503 : Service Unavailable. 서버 점검 등의 이유로 서버 사용이 불가능한 상태
  -https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

5)Response Header
  -Access-Control-Allow-Origin : CORS(Cross Origin Resource Sharing, 다른 도메인에 있는 자원 접근 허용)인 웹 사이트
  -Accept-Patch :
  -Accept-Ranges :
  -Age : 프록시 서버가 사용될 때 원본서버가 문서를 만든 시간을 초단위로 표시한다.
  -Allow : 서버가 허용하는 요청 메소드(GET, HEAD)

-Cache-Control : Cache 매커니즘을 보낸다(max-age=3600)
-Connection : 연결 옵션
-Content-Disposition : 클라이언트에게 파일 다운로드 다이얼로그를 나타나게 하거나, 파일 이름을 알린다.
-Content-Encoding : 본문이 압축/암호화 되었을 때의 헤더 설정 값(gzip등)
-Content-Language : 컨텐츠의 언어
-Content-Location : 컨텐츠가 속한 문서의 위치
-Content-Length, Content-MD5, Content-Range, Content-Type : 요청 헤더와 동일
-Date : 서버와 동일
-ETag : 리소스의 일련 번호
-Expires : 유효 시간
-Last-Modified : 요청한 리소스가 마지막으로 수정된 시각
-Link : 연관도니 리소스와의 관계
-Location : redirect된 절대 URI 주소나 문서의 리소스를 나타낸다.
-P3P
-Pragma
-Proxy-Authenticate : 프록시
-Public-Key-Pins
-Refresh : 새로운 리소스가 생기는 것을 말한다.
-Retry-After : 일정 시간이 지난 후 문서 요청을 다시 한다.
-Server : 원본 서버가 사용하는 웹 서버의 이름을 나타낸다.
-Set-Cookie : 쿠키를 설정한다.
-Status : 요청에 대한 상태코드
-Strict-Transport-Security
-Trailer
-Transfer-Encoding : 데이터 인코딩 방식
-Upgrade : 다른 프로토콜로 업그레이드 여부를 묻는다.
-Vary
-Via : 메세지가 거쳐간 프록시에 대한 정보
-Warning : 경고
-WWW-Authenticate : 응답코드 401(Unauthorized)을 받을 때 인증사항에 대한 응답 메세지로 전달한다. 보통 웹 서버에서 클라이언트에게 사용자 인증을 보낼 때 사용한다.

6)컨텐츠 타입
-메세지 헤더에 기록
-필드 이름 : Content-Type
-대분류/소분류
-주요 컨텐츠 타입
--text/plain, text/html
--application/xml, application/json
--image/png, image/jpg
--audio/mp3, video/mp4

7)Response 메세지 예1
-HTML Response 메세지 헤더
--Content-Type : text/html
-HTML Response 메세지 바디
<!doctype html>
<html>
...
8)Response 메세지 예2
-HTML Response 메세지 헤더

```
288         --Content-Type : image/jpeg
289         --Content-Length : 73228
290       -HTML Response 메세지 바디
291         ffd8ffe000100dffdf0erfd0tetegg0dfgde0g0eww0esfdgfhfh0rett5g8fghddfg...
292
293     9)Content-Type이 맞지 않으면 제대로 표현하지 못한다.
294       ex)Response 메세지 헤더 : Content-Type : application/json, response 매시지 바디 : 이미지...
295       그러면 일치하지 않아 화면에 깨진 글자들만 나타난다.
296
297
298 7. HTTP
299     1)Node.js의 기본모듈
300       var http = require('http');
301
302     2)HTTP Server
303       -클라이언트의 request 메시지 수신
304       -클라이언트에게 response 메시지 전송
305
306     3)HTTP Client
307       -서버로 request 메시지 전송
308       -서버의 response 메시지 수신
309
310     4)서버용 클래스
311       -http.Server : HTTP Server
312       -http.IncomingMessage : HTTP 서버의 request 메시지, Readable Stream
313       -http.ServerResponse : HTTP 서버의 response 클래스
314
315     5)클라이언트 클래스
316       -http.Client : HTTP Client
317       -http.clientRequest : HTTP Client request 메시지
318       -http.IncomingMessage : HTTP 서버의 response 메시지, Readable Stream
319
320     6)Client-Server Message Exchange
321         [Client]                    [Server]
322       ClientRequest -->           -->IncomingMessage
323                                         |
324                                                       |
325                                                       |
326                                                       |
327      IncomingMessage  <--       <--- ServerResponse
328
329     7)HTTP Server
330       -http.Server
331
332       -주요 이벤트
333         --request : 클라이언트의 request 도착
334         --connection : 소켓 연결
335         --close : 서버 종료
336
337       -주요 메소드
338         --server.listen()
```

```
339       --server.close()
340       --server.setTimeout()
341
342     -Server 생성
343       var server = http.createServer([requestListener]);
344       --requestListener
345         ---2개의 파라미터
346         ---http.IncomingMessage : 클라이언트 request
347           ----request 라고 명명
348         ---http.ServerResponse : 클라이언트에게 전송하는 response 개체
349           ----response
350         ---cient에게 Request을 받으면 request 이벤트가 발생하고, 이 리스너가 동작한다.
351         ---클라이언트에게 전송할 response 메시지를 작성해서 보내도록 작성한다.
352         ---클라이언트 response 개체에 데이터를 쓰는 함수는 write이고 response을 완료하는 메소드는 end
            이다.
353
354       var server = http.createServer(function(request, response){
355         response.write('Hello, World');
356         response.end();
357       });
358
359     -HTTP 서버 동작시키기
360       --HTTP Server 생성
361       --Client 접속 대기(listening)
362         var server = http.createServer(function(request, response){
363           response.write('Hello, World');
364           response.end();
365         });
366
367         server.listen(3000);
368
369       http://localhost:3000
370
371     -PORT
372       --0 ~ 1023 : Well-known port. 미리 정의된 포트, 관리자 권한 필요
373       --1024 ~ 49151 : registered port
374       --49152 ~ 65535 : dynamic port
375       --port binding faiure
376         ---이미 사용 중
377         ---권한 없음.
378
379     -서버와 이벤트 리스너
380       var http = require('http');
381       var server = http.createServer();
382       server.on('request', function(request, response){
383       });
384
385       server.on('connection', function(socket){
386         console.log('connection event');
387       });
388       server.on('close', function(){
```

```
389         console.log('close');
390       });
391
392       server.listen(3000);
393
394   8)HTTP Client
395     http.createClient([port][, host]);   //deprecated
396     -client request
397       http.request(options[, callback])
398     -body 없이 request 보내기(GET)
399       http.get(options[, callback])
400
401   9)Lab1:helloworld.js
402     var http = require('http');
403     var server = http.createServer();
404
405     server.on('request', function(req, res){
406       res.write('Hello World');
407       res.end();
408     });
409
410     server.listen(3000);
411
412   10)HTTP request
413     -HTTP 서버에 클라이언트 request가 도착하면 request 이벤트 리스너가 동작하고, 이벤트 리스너 함수의 파
        라미터로 response 객체가 전달된다.
414     -이 응답 개체의 클래스가 http.IncomingMessage이다.
415     -IncomingMessage
416       --message.url:request url, 경로와 query string
417       --message.method : request 메소드
418       --message.headers: request 메세시지의 헤더
419       --message(streamable):request 메세지의 바디
420       --message.httpVersion : http 버전
421
422   11)Lab2 : request.js
423     var http = require('http');
424
425     var server = http.createServer(function(request, response){
426         console.log('version : ' + request.httpVersion);
427         console.log('method : ' + request.method);
428         console.log('url : ' + request.url);
429         console.log('== headers ==');
430         console.log(request.headers);
431         response.end("Hello, World");
432     });
433
434     server.listen(80);
435     ----------------------------------------------------------------------
436     version : 1.1
437     method : GET
438     url : /
```

```
439     == headers ==
440     { host: '192.168.136.5',
441       connection: 'keep-alive',
442       'cache-control': 'max-age=0',
443       'upgrade-insecure-requests': '1',
444       'user-agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36
          (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36',
445       accept:
          'text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8',
446       'accept-encoding': 'gzip, deflate, sdch',
447       'accept-language': 'ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4' }
448     version : 1.1
449     method : GET
450     url : /favicon.ico
451     == headers ==
452     { host: '192.168.136.5',
453       connection: 'keep-alive',
454       pragma: 'no-cache',
455       'cache-control': 'no-cache',
456       'user-agent': 'Mozilla/5.0 (Windows NT 6.3; Win64; x64) AppleWebKit/537.36
          (KHTML, like Gecko) Chrome/58.0.3029.81 Safari/537.36',
457       accept: 'image/webp,image/*,*/*;q=0.8',
458       referer: 'http://192.168.136.5/',
459       'accept-encoding': 'gzip, deflate, sdch',
460       'accept-language': 'ko-KR,ko;q=0.8,en-US;q=0.6,en;q=0.4' }
461
462   12)request 헤더 정보 : request.headers
463     var headers = request.headers;
464     headers.host;
465     headers.content-type;
466     headers.user-agent;
467
468   13)request 바디 정보
469     -IncomingMessage의 data 이벤트 사용
470     -POST 요청에서 다루겠음.
471
472   14)Response 메시지
473     -HTTP Server의 request 이벤트 리스너 함수 중 두번째 파라미터
474     -response 개체인 response는 http.ServerResponse 클래스이다.
475     -메시지 상태
476       --response.statusCode  <--response 상태 코드
477       --response.statusMessage  <--response 상태 메시지
478       --response 상태 코드는 response 메세지의 바디를 작성하기 전까지 입력할 수 있다.
479       --response.statusCode = 200;
480       --response.statusMessage = 'OK';
481       --response.statusCode = 404;
482       --response.statusMessage = 'Not Found';
483     -메세지 헤더
484       --setHeader()로 작성한다.
485       --HTTP 메소드의 헤더는 다수의 헤더를 작성할 수 있다.
486       --메세지 헤더는 이름:값 방식으로 작성하되, 이름은 유일해야
```

```
487         --메세지 헤더는 메세지 바디를 작성하기 전까지 헤더의 내용을 읽고 수정할 수 있다.
488         --response 메세지 바디를 작성하고 나면, 클라이언트에게 헤더를 발송해버리기 때문에 변경할 수 없다.
489           ---response.setHeader(name, value);
490           response.setHeader('Content-Type', 'text/html');
491           response.setHeader('Content-Length', body.length);
492
493         --respons.writeHead(statusCode[, statusMessage][, headers])
494           ---상태 코드와 메세지, 헤더 필드를 한번에 작성 가능
495           var body = 'hello, world';
496
497           response.writeHead(200, {'Content-Length' : body.length,
498                                    'Content-Type' : 'text/plain'});
499         --response.getHeader(name);
500           //헤더 항목 읽기(전송 전)
501           var contentType = response.getHeader('content-type');
502         --response.removeHeader(name);
503           //헤더 항목 삭제(전송 전)
504           response.removeHeader('Cotent-Encoding');
505
506       -메세지 바디
507         --response 메세지 바디에 내용을 작성하려면 write()를 사용한다.
508         --response 메세지의 바디에 write()를 여러번 작성할 수 있다.
509         --response.write(chunk[, encoding][, callback]);
510         --response 메시지 작성을 종료하려면 end()를 사용한다.
511         --end()에도 메세지 바디의 내용을 작성할 수 있다.
512         --end()를 호출하지 않으면 클라이언트는 계속 응답 메세지가 끝나기를 기다린다.
513         --response.end([data][,encoding][, callback])
514           http.createServer(function(request, response){
515             response.end('Hello Node.js');
516           });
517
518     15)Lab3 : response.js
519         var http = require('http');
520
521         var server = http.createServer(function(request, response){
522             var body = '<!doctype html>' +
523                         '<html><head><title>HTTP Example</title>' +
524                         '<meta charset="utf-8" />' +
525                         '<style type="text/css">' +
526                         ' body {background-color:yellow}' +
527                         ' h1 { color : red}' +
528                         '</style></head>' +
529                         '<body><h1>Hello Node.js</h1>' +
530                         '</body></html>';
531
532             response.statusCode = 200;
533             response.statusMessage = 'OKOK';
534             response.setHeader('Content-Type', 'text/html;charset=utf-8');
535             //response.setHeader('Content-Type', 'text/plain');
536             response.setHeader('Content-Length', body.length);
537             response.write(body);
```

```
            response.end();
        });

        server.listen(80);
        //Chrome Browser 개발자도구에서 확인할 것

16)정적 파일 request
  -미리 작성된 HTML이나 이미지, 동영상과 같이 서버의 별도의 동작이 필요없는 파일을 request
  -정적 파일 형태의 리소스는 fs 모듈을 이용해서 접근하고 response 메시지에 담아서 클라이언트에게 전달
  -request 예
    --http://myServer.com/image/cat.jpg
    --http://myService.net/music/sing.mp3
  -정적파일 request : 응답
    --정적 파일 찾기
    --파일 로딩 후 응답
  -fs 모듈
    --fs.readFile(FILEPATH, CALLBACK);
  -response 메세지에 파일 내용 쓰기
    -response.write()
    -response.end();
  -파일 상태 - 존재 확인
    fs.access(path, fs.F_OK, function(exist){
      fs.readFile(path, function(err, data){
        response.end(data);
      });
    });
  -파일 상태 - 존재 확인 --> 만일 파일이 없으면?
    fs.access(path, fs.R_OK, function(err){
      //접근 불가능시 404 에러
      if(err){
        response.statusCode = 404;
        res.end('Not Found');
        return ;
      }
    });
  -사용 예
    fs.access(path, fs.F_OK, function(err){
      if(err){
        response.statusCode = 404;
        res.end('Not Found');
        return ;
      }
      fs.readFile(path, function(err, data){
        response.statusCode = 200;
        response.setHeader('Content-Type', 'image/jpg');
        response.end(data);
      });
    });
  -Stream Pipe 이용하는 방법
    --입력스트림 : fs.createReadStream()
    --출력스트림 : response
```

```
589         fs.createReadStream(path).pipe(response);
590     -파비콘 처리
591       --웹 사이트의 아이콘 이미지
592       --Favorites + Icon
593       --HTML에 웹 사이트의 아이콘 설정에 따라서 서버로 파비콘을 요청하기도 한다.
594       --request은 GET 메소드로 경로는 /favicon.ico이다.
595         if(request.url == '/favicon.ico'){
596           //파비콘 처리
597           return;
598         }
599     -사용 예
600       var server = http.createServer(function(request, response){
601         if(request.url == '/favicon.ico'){
602         }else if(request.url == '/image.png'){
603           res.writeHead(200, {'Content-Type' : 'image/png'});
604           fs.read...
605         }else if(request.url == '/music.mp3'){
606           res.writeHead(200, {'Content-Type' : 'audio/mp3'});
607           fs.createReadStream...
608         }else if(request.url == '/movie.mp4'){
609           res.writeHead(200, {'Content-Type' : 'video/mp4'});
610           fs.createReadStream...
611         }
612       });
613     -요청 URL의 경로를 실제 파일 경로로 매핑
614       myServer.com/resource/image.png --> ./resources/image.png
615       myServer.com/resource/audio.mp3 --> ./resources/audio.mp3
616       -요청 URL에서 경로 생성
617         var pathUtil = require('path');
618         var path = __dirname + pathUtil.sep + 'resources' + req.url;
619
620   17)Web Server 만들기
621     -이미지를 비롯한 리소스로 구성된 웹 페이지를 서비스하는 웹서버를 만든다.
622     -Request
623       --Index request
624       --파비콘 request
625       --js와 css request
626       --이미지 request
627
628   18)Lab : img.js
629     var http = require('http');
630     var fs = require('fs');
631
632     var server = http.createServer(function(request, response) {
633       fs.access('images/nodejs.png', function(err) {
634         if ( err ) {
635           response.statusCode = 404;
636           response.end();
637           return;
638         }
639         fs.readFile('images/nodejs.png', function(err, data) {
```

```
            response.end(data);
        });

    });
}).listen(80);

-index.html
    <!doctype html>
    <html>
        <head>
            <title>Welcome to Homepage</title>
            <link rel="stylesheet" type="text/css" href="css/style.css">
            <link rel="icon" type="image/x-icon" href="/favicon.ico">
        </head>
        <body>
            <h1>Hello Node.js</h1>
            <img src="images/nodejs.png">
        </body>
    </html>

-css/style.css
    body { background-color:yellow}
    h1 { color : blue}
    img { width:200px;height:100px}

-server.js
    var http = require('http');
    var fs = require('fs');
    var server = http.createServer(function(request, response){
        console.log('request : ' + request.url);

        //index
        if(request.url == '/'){
            response.writeHead(200, {'Content-Type':'text/html;charset=utf-8'});
            //Stream과 Pipe 조합
            fs.createReadStream('index.html').pipe(response);
            return;
        }else if(request.url == '/favicon.icon'){
            //파비콘 처리
            response.writeHead(200, {'Content-Type':'image/x-icon'});
            fs.createReadStream('favicon.ico').pipe(response);
            return;
        }
        //파일 존재 검사
        var path = __dirname + request.url;
        console.log('path : ' + path);
        fs.access(path, fs.F_OK, function(err){
            if(err){
                response.statusCode = 404;
                response.end('Not Found');
                return ;
```

```
            }
            fs.readFile(path, function(err, data){
                if(path.substring(path.lastIndexOf('.')) == '.css'){
                    response.writeHead(200, {'Content-Type' : 'text/css'});
                }else if(path.substring(path.lastIndexOf('.')) == 'png'){
                    response.writeHead(200, {'Content-Type' : 'images/*'});
                }
                fs.createReadStream(path).pipe(response);
            });
        });
    });
    server.listen(80);

19)Lab : httpdemo.js
  var http = require('http');
  var fs = require('fs');

  var server = http.createServer(function(request, response){
    if(request.url == '/'){
      fs.readFile('./index.html', function(err, data){
        if(err){
          console.log("Server Error");
          response.writeHead(200);
          response.write("Sorry^^.");
          response.end();
        }
        response.writeHead(200, {'Content-Type' : 'text/html'});
        response.write(data);
        response.end();
      })
    }else{
      console.log("Not Found");
      response.writeHead(404);
      response.end();
    }
  });

  server.listen(80, function(){
    console.log("I'm ready...");
  });

  -index.html
  <!DOCTYPE html>
  <html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Welcome My Home...</title>
    <link rel="stylesheet" href="css/style.css" type='text/css'>
  </head>
```

```
742     <body>
743       <div>Welcome...</div>
744       <img src='images/jimin.jpg' alt='한지민 이미지'>
745     </body>
746     </html>
747
748   20)Lab : httpdemo1.js
749     var http = require('http');
750     var fs = require('fs');
751     var url = require('url');
752
753     var server = http.createServer(function(request, response){
754       //console.log(url.parse(request.url));
755       var pathname = url.parse(request.url).pathname;
756       if(pathname == '/') pathname = '/index.html';
757       if(pathname.match(/^\//g)){
758         fs.readFile(pathname.substring(1), function(err, data){
759           if(err){
760             console.log("Server Error");
761             response.writeHead(200);
762             response.write("Sorry^^.");
763             response.end();
764           }
765           response.writeHead(200, {'Content-Type' : 'text/html'});
766           response.write(data);
767           response.end();
768         })
769       }else{
770         console.log("Not Found");
771         response.writeHead(404);
772         response.end();
773       }
774     });
775
776     server.listen(8888, function(){
777       console.log("I'm ready...");
778     });
779
780   21)Lab : httpdemo2.js
781     var http = require('http');
782     var fs = require('fs');
783     var url = require('url');
784
785     var server = http.createServer(function(request, response){
786       if(request.url != '/favicon.ico'){
787         var pathname = url.parse(request.url, true).pathname;
788         if(pathname == '/') pathname = '/index.html';
789         if(pathname.match(/\//g)){
790           if(pathname == '/register'){
791             response.writeHead(200);
792             response.write(JSON.stringify(url.parse(request.url, true).query));
```

```
            response.end();
          }
          fs.readFile(pathname.substring(1), 'utf8', function(err, data){
            if(err) {
              console.log("Error");
              response.writeHead(200);
              response.write("Sorry");
              response.end();
            }
            response.writeHead(200, {'Content-Type' : 'text/html'});
            response.write(data);
            response.end();
          });
        }
      }
    });

    server.listen(80);

  22)Lab : httpdemo3.js
    var http = require('http');
    var fs = require('fs');

    var server = http.createServer(function(request, response){
      if(request.url == '/favicon.ico'){
      }else if(request.url == "/"){
        fs.readFile('./index.html', 'utf8', function(err, data){
          response.writeHead(200, {'Content-Type' : 'text/html'});
          response.write(data);
          response.end();
        });
      }else if(request.url == '/images/jimin.jpg'){
        fs.access('./images/jimin.jpg', function(err){
          if(err){
            //response.writeHead(404);
            response.statusCode = 404;
            response.end();
          }
          fs.readFile('./images/jimin.jpg', function(err, data){
            response.writeHead(200, {'Content-Type' : 'image/jpeg'});
            response.write(data);
            response.end();
          });
        });
      }
    });

    server.listen(80);

  23)Lab : URL Query string 이용하기
    http://192.168.56.5/count?start=1&end=100
```

```
844
845     <query.js>
846     var http = require('http');
847     var url = require('url');
848     var server = http.createServer(function(request, response) {
849         // URL 분석 : 쿼리 문자열
850         var parsed = url.parse(request.url, true);
851         var query = parsed.query;
852
853         // start와 end
854         var start = parseInt(query.start);
855         var end = parseInt(query.end);
856
857         if ( !start || !end ) {
858             response.statusCode = 404;
859             response.end('Wrong Parameter');
860         }else {
861             // 합계 구하기
862             var result = 0;
863             for(var i = start ; i <= end ; i++) {
864                 result += i;
865             }
866             response.statusCode = 200;
867             response.end('Result : ' + result);
868         }
869
870     }).listen(80);
871
872   24)Lab : querystringdemo.js
873     var http = require('http');
874     var url = require('url');
875     var fs = require('fs');
876
877     var server = http.createServer(function(request, response){
878       if(request.url == '/input1.html'){
879         response.writeHead(200, {'Content-Type':'text/html'});
880         fs.createReadStream('./input1.html').pipe(response);
881         return;
882       }else if(request.url.match(/^\/demo/g)){
883         var obj = url.parse(request.url, true).query;
884         var first = parseInt(obj['first']);
885         var second = parseInt(obj.second);
886         if(!first || !second){
887           response.statusCode = 404;
888           response.end("Invalid Parameter");
889         }else{
890           response.statusCode = 200;
891           response.end(first + ' + ' + second + ' = ' + (first + second));
892         }
893       }
894     }).listen(80);
```

```
    //localhost/demo?first=5&second=9

    [input1.html]
      <!DOCTYPE html>
      <html lang="en">
      <head>
        <meta charset="UTF-8">
        <title>Query String Demo</title>
      </head>
      <body>
        <form action='/demo' method='get'>
          First : <input type='number' name='first' /><br />
          Second : <input type='number' name='second' /><br />
          <input type='submit' value='합계구하기' />
        </form>
      </body>
      </html>

  25)Lab : server.js
    var http = require('http');
    var fs = require('fs');

    var server = http.createServer(function(request, response){
      if(request.url == '/'){
        response.writeHead(200, {'Content-Type' : 'text/html'});
        fs.createReadStream('./index.html').pipe(response);
        return;
      }else if(request.url == '/favicon.ico'){
        response.writeHead(200, {'Content-Type' : 'image/x-icon'});
        fs.createReadStream('./favicon.ico').pipe(response);
        return;
      }else if(request.url == '/images/jimin.jpg'){
        response.writeHead(200, {'Content-Type' : 'image/jpeg'});
        fs.createReadStream('./images/jimin.jpg').pipe(response);
        return;
      }else if(request.url == '/css/style.css'){
        response.writeHead(200, {'Content-Type' : 'text/css'});
        fs.createReadStream('./css/style.css').pipe(response);
        return;
      }
    });

    server.listen(80, function(){
      console.log("Starting...");
    });


7. POST Request
  1)개요
    -글을 작성하거나 사진을 업로드 하는 요청은 Post 메소드를 사용한다.
    -Post 메소드는 요청 정보를 body(entity)에 작성한다.
```

946
947 2)GET과 POST request 비교
948 -GET request
949 --URL로 request 정보 전달
950 --URL만 분석해도 되므로 서버에서는 빠르게 반응
951 --길이 제한, 암호화 불리, 데이타 노출
952 --Link를 통해서도 request 전달 가능
953 --웹브라우저에서 주소창을 이용해서 입력가능
954 -POST request
955 --메세지 바디(entity)로 request 정보 전달
956 --request 바디를 분석해야 하기 때문에 GET 요청에 비해서 느리다는 단점
957 --길이제한 없음, 암호화 유리, 데이타 노출 안함
958 --반드시 form의 submit을 해야 함.
959 --웹 브라우저의 폼 입력(GET/POST 모두 가능)
960 <form method='post | get' action='/upload'>
961 </form>
962
963
964 8. HTML Form
965 1)HTML에서 Post request은 form을 이용한다.
966 2)폼의 내부에 정보 입력은 input 태그 같은 태그로 작성한다.
967 <input type='' name='' />
968
969 3)form type
970 -file
971 -date
972 -text
973 -radio, checkbox
974 -reset
975 -submit
976 ...
977
978 4)예
979 <form method='post' action='/upload'>
980
981 date : <input type='date' name='date'>
982 email : <input type='email' name='email'>
983 file : <input type='file' name='file'>
984 radio : <input type='radio' name='radio' value='YES'>YES
985 <input type='radio' name='radio' value='NO'>NO
986 checkbox : <input type='checkbox' name='checkbox'>
987 text : <input type='text' name='text'>
988 password : <input type='password' name='password'>
989 number : <input type='number' name='number'>
990 textarea : <textarea name='textarea'>Textarea</textarea>
991 <input type='submit' value='Upload'>
992
993 </form>
994
995 5)form encoding
996 -폼에 작성한 데이터는 request 바디에 인코딩돼서 서버로 전달된다.

```
997       -폼 태그의 enctype에 인코딩 방식을 작성
998         --application/x-www-form-urlencoded
999           ---공백은 +로 인코딩한다.
1000          ---특수 기호도 코드로 인코딩한다.
1001        --multipart/form-data
1002          ---인코딩하지 않고 전송한다.
1003          ---파일 전송시 사용
1004        --text/plain
1005          ---공백문자만 +로 인식하고 다른 문자는 변경하지 않는다.
1006      -기본값은 form-urlencoded 이다.
1007        --이름=값 형태로 &를 구분자로 메세지 바디에 기록한다.
1008        --헤더의 Content-Type에는 application/x-www-form-urlencoded로 기록된다.
1009
1010    6)Lab : postdemo.js
1011      //POST 방식
1012      var http = require('http');
1013      var fs = require('fs');
1014
1015      var server = http.createServer(function(request, response){
1016        if(request.url == '/input2.html'){
1017          response.writeHead(200, {'Content-Type':'text/html'});
1018          fs.createReadStream('./input2.html').pipe(response);
1019          return;
1020        }
1021        var body = '';
1022        request.on('data', function(chunk){
1023          body += chunk;
1024        });
1025        request.on('end', function(){
1026          console.log("Data End...");
1027          console.log(body);
1028        });
1029      }).listen(80);
1030
1031      [input2.html]
1032        <!DOCTYPE html>
1033        <html lang="en">
1034        <head>
1035          <meta charset="UTF-8">
1036          <title>Query String Demo</title>
1037        </head>
1038        <body>
1039          <form action='/demo1' method='post'
          enctype="application/x-www-form-urlencoded">
1040            First : <input type='number' name='first' /><br />
1041            Second : <input type='number' name='second' /><br />
1042            <input type='submit' value='합계구하기' />
1043          </form>
1044        </body>
1045        </html>
1046
```

7)Lab : postdemo1.js

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

var server = http.createServer(function(request, response){
  if(request.method.toLocaleLowerCase() == 'post'){
    inputPage(request, response);
  }else if(request.method.toLocaleLowerCase() == 'get'){
    readPage(request, response);
  }
}).listen(80);

function inputPage(req, res){
  var body = '';
  req.on('data', function(chunk){
    console.log("%d bytes received", chunk.length);
    body += chunk;
  });
  req.on('end', function(){
    console.log("Data End...");
    console.log("Data : ", body);
  })
}
function readPage(req, res){
  res.writeHead(200, {'Content-Type':'text/html'});
  fs.createReadStream('./input2.html').pipe(res);
  return;
}
```

9. POST Request 처리하기

1)Request 메시지 : request.IncomingMessage

2)Readable Stream

-이벤트 : data, end

-data : 스트림으로 도착한 일정 크기의 데이터(chunk)를 읽을 수 있을 때 발생

-end : 스트림에 읽을 수 있는 데이터가 없을 때 발생

3)예

```
var body = '';
request.on('data', function(chunk){
  console.log('received %d bytes of data', chunk.length);
  body += chunk;
}
request.on('end', function(){
  console.log('There will be no more data.');
  console.log('end : ' + body);
});
```

4)end 이벤트

-전송이 끝나면 발생

-end 이벤트 핸들러 함수 : querystring 모듈로 분석

```
        request.on('end', function(){
          var parsed = querystring.parse(body);
          console.log('name1 : ' + parsed.name1);
          console.log('name2 : ' + parsed.name2);
        });

    5)Post request 후 Refresh(F5) --> 중복 Post request
      "양식 다시 제출 확인" 창

    6)PRG 패턴
      -PRG(Post-Redirect-Get) 패턴
      -Post request 후 사용자가 웹 브라우저에서 리프레쉬를 누르면 Post 요청이 중복해서 서버로 전송되는 현상
      발생
      -중복 POST request 방지 패턴
      -Post request 이후에 redirection하는 패턴
      -응답 메세지 작성 코드
        --Redirection : 클라이언트 주소 옮기기
        --상태코드 : 302
        --헤더필드 : Location
          response.writeHead(302, {'Location' : 'http://google.com'});

    7)PRG 패턴 적용 코드
      request.on('end', function(){
        //POST request 메시지 바디 분석 및 처리
        response.statusCode = 302;
        response.setHeader('Location', URL);
        response.end();
      });

    8)Lab : post.js
        var http = require('http');
        var querystring = require('querystring');

        var movieList = [{ title: '스타워즈4', director: '조지루카스' }];

        var server = http.createServer(function (req, res) {
            if (req.method.toLowerCase() == 'post') {
                addNewMovie(req, res);
            }else if(req.method.toLowerCase() == 'get'){
                showList(req, res);
            }
        });
        server.listen(80);

        function showList(req, res) {
          res.writeHead(200, {'Content-Type' : 'text/html'});
          res.write('<!DOCTYPE html>');
          res.write('<html lang="en">');
          res.write('<head>');
          res.write('   <meta charset="UTF-8">');
          res.write('   <meta name="viewport" content="width=device-width,
```

```
            initial-scale=1.0">');
1148        res.write('    <title>Favorite Movie Lists</title>');
1149        res.write('</head>');
1150        res.write('<body>');
1151        res.write('<h2>좋아하는 영화 목록</h2>');
1152         res.write('<ul>');
1153         for(var i = 0 ; i < movieList.length ; i++){
1154            res.write('<li>' + movieList[i].title + '[' + movieList[i].director + ']</li>');
1155         }
1156        res.write('</ul>');
1157        res.write("<form action='' method='post'>");
1158        res.write("<div>영화 제목 : <input type='text' name='title' placeholder='영화제목을 입
            력해 주세요'></div>");
1159        res.write("<div>영화 감독 : <input type='text' name='director' placeholder='감독이름을
            넣어주세요'></div>");
1160        res.write("<input type='submit' value='전송'/>");
1161        res.write('</form>');
1162        res.write('</body>');
1163        res.write('</html>');
1164        res.end();
1165      }
1166
1167      function addNewMovie(req, res) {
1168        var body = '';
1169        req.on('data', function(chunk) {
1170             body += chunk;
1171        });
1172        req.on('end', function() {
1173             var data = querystring.parse(body);
1174             var title = data.title;
1175             var director = data.director;
1176
1177             movieList.push({title:title, director:director});
1178
1179             //res.end('Success');
1180             res.statusCode = 302;
1181             res.setHeader('Location', '.');
1182             res.end();
1183        });
1184      }
1185
1186   9)Chrome Extension : Postman 사용하기
1187     -메소드 :POST
1188     -주소 : 192.168.56.5
1189     -Body
1190     -x-www-form-urlencoded
1191     -Key                    Value
1192       title                  아바타
1193       director                제임스 카메런
1194     -Click [Send] button
1195
```

10. Multipart를 사용하는 Post request
  1)Post 요청 메세지
    -사진 올리기
    -글과 사진 올리기
    -SNS 서비스에서

  2)메세지 바디 기록 방식
    multipart/form-data

  3)파트 구분 정보
    -content-type 헤더 항목의 boundary로 정의
    Content-Type : multipart/form-data;boundary=XXXYYYZZZ

  4)메세지 바디 내 파트 구성
    -파트 구분자(--XXXYYYZZZ);
    -파트 인코딩
    -파트 내 정보

  5)멀티파트 메세지
    Content-Type : multipart/form-data;boundary=XXXYYYZZZ

    --XXXYYYZZZ
    Content-Disposition:form-data;name='key1'

    value1
    --XXXYYYZZZ
    Content-Type:application/octet-stream
    Content-Transfer-Encoding:base64

    PGhdkedkgdk+ejtke9rdgjdkvzgjekgatektrter35dfjdfe8r3kghbnske
    teitkgubndlt4e8t857dgbk9sr92ty0bvdetrtygd
    --XXXYYYZZZ

  6)HTML Form
    -form enctype : multipart/form-data
    -Input type : file
      <form method='post' action='upload' enctype='multipart/form-data'>
        <input type='text' name='name'.
        <input type='file' name='poster'>
      </form>

  7)http-client(Postman)사용
    -method는 Post로
    -body type은 form-data
    -입력 항목 타입은 file로

  8)멀티 파트 request 분석
    -메세지 request 헤더
      Content-Type : multipart/form-data;boundary=XXXYYYZZZ
    -메세지 바디 분석

```
1247        --각 파트 구분
1248        --파트별 구분
1249
1250    9)컨텐츠 타입 분석 코드
1251      var contentType = request.headers['content-type'];
1252      var elements = contentType.split(';');
1253      var firstElem = elements[0];  //multipart-form-data
1254      var mainContentType = firstElem.split('/')[0];  //multipart
1255
1256      var secondElem = elements[1].trim();  //boundary=XXXYYYZZZ
1257      var boundary = secondElem.split('=')[1]; //XXXYYYZZZ
1258
1259    10)body 분석
1260      var buffer = '';
1261      request.on('data', function(chunk){
1262        buffer += chunk.toString();
1263      });
1264      request.on('end', function(){
1265        //boundary로 각 파트 구분
1266        var parts = buffer.split('--' + boundary);
1267
1268        for(var i = 0 ; i < parts.length ; i++){
1269          //각 파트별 분석
1270        }
1271        response.end('Multipart EncType message example');
1272      });
1273
1274    11)멀티파트 분석 모듈
1275      -formidable
1276      -multer
1277
1278    12)Lab : multipartdemo.js
1279      //multipart post
1280      var http = require('http');
1281      var fs = require('fs');
1282
1283      var server = http.createServer(function(request, response){
1284        if(request.method.toLocaleLowerCase() =='get'){
1285          response.writeHead(200, {'Content-Type':'text/html'});
1286          fs.createReadStream('./demo.html').pipe(response);
1287          return;
1288        }else if(request.method.toLocaleLowerCase() == 'post'){
1289          var body = '';
1290          request.on('data', function(chunk){
1291            body += chunk.toString();
1292          });
1293          request.on('end', function(){
1294            var conType = request.headers['content-type'];
1295            var boundary = conType.split(';')[1].split('=')[1];
1296            var array = body.split(boundary);
1297            console.log(array[1]);
```

```
            console.log(array[2]);
            //for(var i = 0 ; i < array.length ; i++){
            //    console.log(array[i]);
            // }
            response.end();
          });
        }
      });

      server.listen(80, function(){
        console.log("Starting...");
      })
```

11. formidable module
  1)파일 업로드는 멀티 파트를 이용해서 업로드한다.
  2)이 모듈은 멀티파트로 전송되는 post 요청을 처리하는 모듈
  3)https://github.com/felixge/node-formidable
  4)Installation
    $ npm install formidable
    C:\NodeHome
        |
        |-------formidable@1.1.1
  5)클래스
    -Formidable.IncomingForm : 요청 분석 클래스
    -Formidable.File : 업로드된 파일 정보
  6)Formidable.IncomingForm
    -클라이언트의 요청은 이 객체로 파싱한다.
    var form = new formidable.IncomingForm();
    form.parse(request, function(err, fields, files){});
    -fields : 요청 메세지 바디로 전달되는 값. 키-밸류 방식
    -files : 요청 메세지 바디로 전달되는 파일. 키-밸류 방식
    -Event
      --field : 이름/값 도착 이벤트
      --file : 파일 도착 이벤트
      --aborted : 요청 중지(클라이언트)
      --end : 종료
    -Property
      --form.uploadDir : 파일 업로드 폴더
      --form.keepExtensions : 확장자 보존
      --form.multiples : 다중 파일 업로드

  7)요청 메세지를 formidable의 폼 객체로 파싱하는 코드

```
    function handlePostRequest(req, res){
      var form = new formidable.IncomingForm();
      form.encoding = 'utf-8';
      form.keepExtension = true;
      form.uploadDir = uploadDir;

      form.parse(req, function(err, fields, files){
      });
    });
```

8)Formidable.File
-클라이언트가 전송한 파일로서 다음의 속성을 가진다.
--file.size : 업로드된 파일의 크기(바이트)
--file.path : 파일 경로
--file.name : 파일 이름
--file.type : 파일 타입
--file.lastModifiedDate : 최종 변경일
--file.hash : 해쉬값
-업로드된 파일의 이름은 서로 겹치지 않도록 임의의 이름으로 변경돼서 저장됨.
-파일의 원래의 이름은 file.name 에서 얻을 수 있다.
-업로드 설정 폴더로 저장(설정 안하면 OS의 임시 폴더)

9)파일 업로드 서비스
-파일 업로드 - 임시 폴더
-파일 업로드 후
--파일을 임시 폴더 --> 리소스 저장소로 이동
--리소스 저장소에서 이름이 충돌되지 않도록 이름 변경
--날짜, 일련번호, 사용자 계정...

10)Lab : fileupload.js

```
var fs = require('fs'),
    path = require('path'),
    http = require('http'),
    formidable = require('formidable');

// 업로드 된 데이터 목록
var movieList = [];

var server = http.createServer(function (request, response) {
      // 루트 경로로 요청
      if (request.url == '/' && request.method.toLowerCase() == 'get') {
         showList(response);
      }
      // <img> 태그로 인한 이미지 요청
      else if (request.method.toLowerCase() == 'get' &&
      request.url.indexOf('/image') == 0) {
         var imgpath = __dirname + request.url;
         response.writeHead(200, { 'Content-Type': 'image/jpeg' })
         fs.createReadStream(imgpath).pipe(response);
      }
      // 업로드 요청
      else if (request.method.toLowerCase() == 'post') {
         addList(request, response);
      }

});

function showList(response) {
    response.writeHeader(200, { 'Content-Type': 'text/html' });
```

```
1399            var body = '<!DOCTYPE html>';
1400            body += '<html>';
1401            body += '<head>';
1402            body += '<title>Favorite Movie Poster Collection</title>';
1403            body += '<meta charset="UTF-8">';
1404            body += '</head>';
1405            body += '<body>';
1406            body += '<h3>Favorite Movie Poster Collection</h3>';
1407
1408            body += '<form action="." enctype="multipart/form-data" method="post">';
1409            body += '<div><label>영화 제목 : </label><input type="text"
                name="title"></div>';
1410            body += '<div><label>영화 포스트 : </label><input type="file" name="image"
                value="영화 포스터선택"></div>';
1411            body += '<input type="submit" value="upload">';
1412            body += '</form>';
1413            body += '<hr />';
1414
1415            movieList.forEach(function (item, index) {
1416                body += '<figure>';
1417                if (item.image) {
1418                    body += "<img src='" + item.image + "' style='height:100pt;width:80pt'
                    />";
1419                    body += '<figcaption>' + item.title + '</figcaption>';
1420                }
1421                body += '</figure>';
1422            });
1423            body += '</body>';
1424            body += '</html>';
1425            response.end(body);
1426        }
1427
1428        server.listen(80, function () {
1429              console.log('Server is running on 80');
1430        });
1431
1432        function addList(request, response) {
1433            var form = formidable.IncomingForm();
1434
1435            form.parse(request, function(err, fields, files) {
1436                var title = fields.title;
1437                var image = files.image;
1438                //console.log(image);
1439
1440                var date = new Date();
1441                var imgname = 'image_' + date.getHours() + date.getMinutes() +
                    date.getSeconds();
1442                var ext = path.parse(image.name).ext;
1443                var newName = './images/' + imgname + ext;
1444                fs.renameSync(image.path, newName);
1445
```

```
1446            var obj = { title : title, image:newName };
1447            movieList.push(obj);
1448
1449            response.statusCode = 302;
1450            response.setHeader('Location', '.');
1451            response.end();
1452          });
1453        }
```