# 노드찡과 함께하는 뻔다멘탈 웹-어프리케이숑

썩zoo

# Javascript?

객체기반

Script language

Front-end에 주로 써왔던 바로 그것.

주로 웹브라우저에서 실행된다.

~~류석영-교수님이-쓰레기라고-말하는-언어~~

# Node.js

Event-driven I/O framework

자바스크립트를 서버에서 실행한다.

# Node.js 사용법

vi hello.js

[hello.js]

```
console.log(“hello node!”);
```

node hello.js

```
C:₩>node
> console.log("hello node!");
hello node!
undefined
```

# 끝.

node [source.js] 이렇게 쓰면 됨다 ^^

....ㅈㅅㅈㅅ

# Web application?

HTTP Server

Router

Request handler

Request data handling

View logic

Upload handling

# HTTP Server

웹 페이지를 제공해주는 역할!

Hello world를 띄워보자!

# server.js

```
var http = require(“http”);


http.createServer(function(request, response) {
    response.writeHead(200, {“Content-Type” : “text/plain”});
    response.write(“Hello, world!”);
    response.end();
}).listen([port]);
```

언제나 그렇듯 포트 넘버가 같으면……(부끄)

# 실행 결과

node server.js

bit.sparcs.org:[port]


Hello world

# 우리가 한 일?

var http = require(“http”);

node.js에 있는 http 모듈을 읽어들인다.

var server = http.createServer();

http 서버를 하나 만든다.

server.listen([port]);

[port] 포트를 listen 시킨다.

# Function, as a parameter

```
var say = function(something){
    console.log(“Hello, “ + something);
}
var Do = function(action, value) {
    action(value);
}
Do(say);
```

# createServer(function)

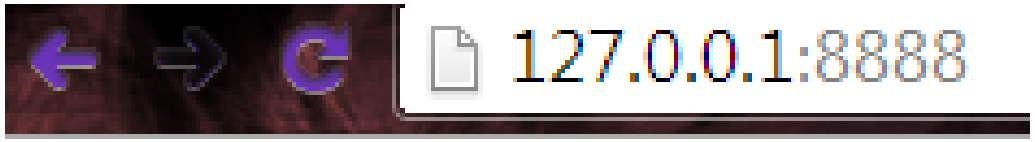
```
http.createServer(function(request, response) {
    response.writeHead(200, {“Content-Type” : “text/plain”});
    response.write(“Hello, world!”);
    console.log(“connected”);
    response.end();
}).listen(8888);
```

# Event-driven callbacks

```
var http = require(“http”);
function onRequest(request, response) {
    console.log(“request received”);
    response.writeHead(200, {“Content-Type” : “text/plain”});
    response.write(“Hello, world!”);
    response.end();
}
http.createServer(onRequest).listen([port]);
console.log(“Server started”);
```

# Event-driven callbacks

```
C:₩>node node-practice.js
Server started
```




Hello, world!

# Event-driven callbacks

분명 Server started가 출력 되었음에도 불구하고

bit.sparcs.org:[port]는 반응한다?

onRequest를 listen에 넘겨주었다.

bit.sparcs.org:[port]에 연결 된 순간, listen이 onRequest를 호출.

Event가 발생하면 역으로 함수를 호출 : event-driven callbacks

# Server.js 이용하기

우린 HTTP 서버를 짰다.

서버가 들은 코드에 모든 function을 짜야되나?
코드가 더러워진다.

# Require

var http = require(“http”);

Node.js의 내부에 http라는 이름을 가진 모듈이 있다.

이 모듈의 기능을 모두 이용하겠다는 의미.

var server = require(“./server”);

server 모듈을 이용하겠다는 의미가 되겠져

server의 어떤 기능을 이용할지 export를 해줘야 한다!

# Server.js

```
var http = require(“http”);
function start() {
    function onRequest(request, response) {
        console.log(“Request received”);
        response.writeHead(200, {“Content-Type” : “text/plain”});
        response.write(“Hello, world!”);
        response.end();
    }
    http.createServer(onRequest).listen([port]);
}
exports.start = start;
```

# Index.js

```
var server = require(“./server.js”);
server.start();
```

# Web application?

~~HTTP Server~~

Router

Request handler

Request data handling

View logic

Upload handling

# Routing

Requesting URL + GET/POST -> router
  라우터는 어떤 코드를 실행할지 결정.

우리가 라우터를 통해서 구현할 것?
  Parse URL
  Parse Method(GET/POST)
  -> Request에 전부 들어있는 정보!

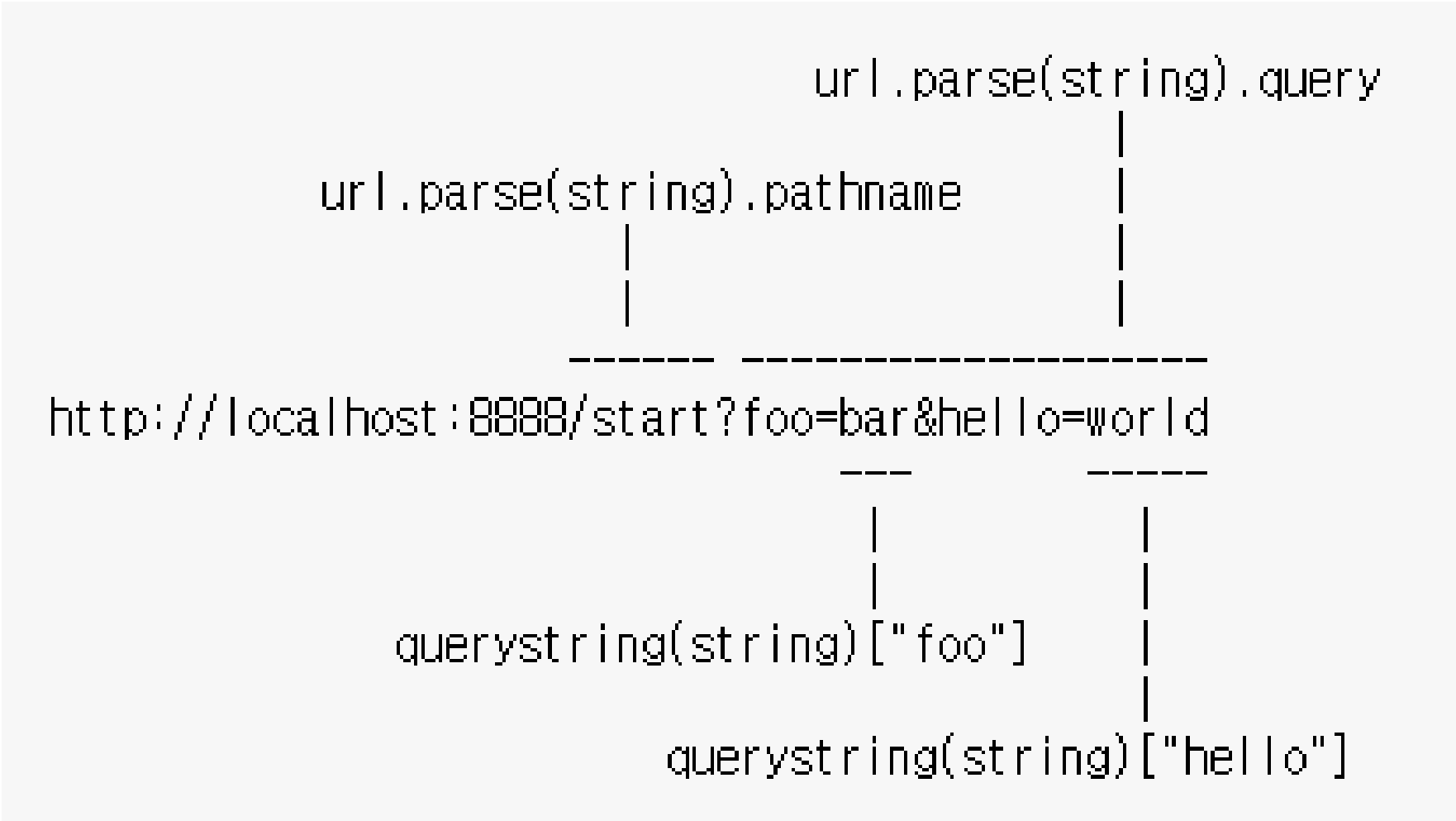
Modules : url, querystring

# You are at : [location]

일단은 server.js에서 수정!

```
function onRequest(request, response) {
        var pathname = url.parse(request.url).pathname;
        console.log( " request received " );
        response.writeHead(200, {  " Content-Type " :  " text/plain "  });
        response.write("Now you are at : "+pathname);
        response.end();
    }
```

# URL, Querystring modules



# 진짜 router를 짜보자!

router.js 파일을 새로 만든다.

# router.js

```
function route(pathname) {
    console.log(“About to route a request for “ + pathname);
}

exports.route=route
```

# server.js

```
var http = require("http");
var url = require("url");

function start(route) {
  function onRequest(request, response) {
    var pathname = url.parse(request.url).pathname;
    console.log("Request for " + pathname + " received.");

    route(pathname);

    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello World");
    response.end();
  }

  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

# index.js

```
var server = require(“./server”);
var router = require(“./router”);
server.start(router.route);
```

다 했으면 node index.js 하고 아무 루트나 입력해보기!

# Web application?

~~HTTP Server~~

~~Router~~

Request handler

Request data handling

View logic

Upload handling

# Request handling

Router는 요청에 대해 실제로 action을 하는 부분이 아님!
실질적인 부분은 Request handler에 있다.

Request handler : 요청이 route되는 함수.

# requestHandler.js

```
function start() {
    console.log("Request handler 'start' has called");
}
function upload() {
    console.log("Request handler 'upload' has called");
}
exports.start = start;
exports.upload = upload;
```

# requestHandler.js

실제 실행될 함수들을 모아놓은 모듈이라고 볼 수 있다.

# index.js with request handler

```
var server = require("./server");
var router = require("./router");
var requestHandler = require("./requestHandler");

var handle = {}
handle["/"] = requestHandler.start;
handle["/start"] = requestHandler.start;
handle["/upload"] = requestHandler.upload;

server.start(router.route, handle);
```

# index.js

requestHandler 모듈로부터 request handler들을 받아온다.

javascript의 object는 일종의 dictionary이다.

이를 이용하여 handle[pathname]을 통해 바로 request handler를 불러오려 한다.

# server.js with request handler

```
var http = require("http");
var url = require("url");
function start(route, handle) {
    function onRequest(request, response) {
        var pathname = url.parse(request.url).pathname;
        console.log("request for " + pathname + " received");
        route(handle, pathname);
        response.writeHead(200, { "Content-Type": "text/plain" });
        response.write("Now you are at : "+pathname);
        response.end();
    }
    http.createServer(onRequest).listen(8888);
    console.log("Server started");
}
exports.start = start;
```

# server.js

router에 핸들을 넘겨준다.

router는 핸들을 참조하여 어떤 코드가 실행되어야 할지 결정하게 될 것이다.

# router.js with request handler

```
function route(handle, pathname) {
    console.log("About to route a request for " + pathname);
    if (typeof handle[pathname] == 'function') {
        handle[pathname]();
    }
    else {
        console.log("No such handle for request " + pathname);
    }
}
exports.route = route;
```

# router.js

handle[pathname]으로 request handler에 direct하게 접근할 수 있었다.

handler가 function
우리가 이전에 mapping을 해 놓은 바로 그 것이다.

handler가 function이 아님
mapping 되지 않았다. 따라서 type이 function이 아니다.

# How to write request handler?

직관적인 방식

request handler에서 return받고 받은 데이터를 서버에서 쏴주기

이렇게만 하면 되나?

# 그렇게 해보자 – requestHandler.js

```
function start() {
    console.log("Request handler 'start' has called");
    function Sleep(mseconds) {
        var startTime = new Date().getTime();
        while (new Date().getTime() < startTime + mseconds);
    }
    Sleep(10000);
    return "Hello start";
}
function upload() {
    console.log("Request handler 'upload' has called");
    return "Hello upload";
}
exports.start = start;
exports.upload = upload;
```

# router.js

```
function route(handle, pathname) {
    console.log("About to route a request for " + pathname);
    if (typeof handle[pathname] == 'function') {
        return handle[pathname]();
    }
    else {
        console.log("No such handle for request " + pathname);
        return "404 not found";
    }
}
exports.route = route;
```

# server.js

```
var http = require("http");
var url = require("url");
function start(route, handle) {
    function onRequest(request, response) {
        var pathname = url.parse(request.url).pathname;
        console.log("request for " + pathname + " received");
        var content = route(handle, pathname);
        response.writeHead(200, { "Content-Type": "text/plain" });
        response.write(content);
        response.end();
    }
    http.createServer(onRequest).listen(8888);
    console.log("Server started");
}
exports.start = start;
```

# Blocking / Non-blocking

다른 모든 짓을 수행하지 못 하게 막는 짓 : blocking

우리는 non-blocking하게 request handler를 짜야한다

# 이 방법이 먹히나

bit.sparcs.org:[port]/start를 한 탭에서 ㄱㄱ

탭하나 더 열어서 bit.sparcs.org:[port]/upload ㄱㄱ

# Node의 괴랄한 특성

Node.js는 모든 것을 병렬로 처리한다.

니 코드만 빼고.

# Non-blocking style – requesthandler.js

```
var exec = require("child_process").exec;
function start() {
    console.log("Request handler 'start' has called");
    var content='empty';
    exec("tree ..", function (error, stdout, stderr) {
        content = stdout;
    });
    return content;
}
function upload() {
    console.log("Request handler 'upload' has called");
    return "Hello upload";
}

exports.start = start;
exports.upload = upload;
```

shell에서 가능한 가장 무거운 command!
linux는 find /

# exec

대표적인 non-blocking function

인자로 두개를 받는다.

앞의 인자는 shell operation,

뒤의 인자는 callback function, 즉 shell operation이 끝나고 실행 될 함수

이렇게 callback을 받는 몇몇 함수들이 있다.

db.query, setTimeout, setInterval, ..... → heavy function

# 결과?

아까같이 start와 upload 동시에 연결?
  둘다 바로 response가 온다!

그런데 start의 내용은?
  callback이 미처 실행되기 전에 return 된 “empty”

# Nonblocking+Direct Response

사실 아까 그 콜백이 언젠가 실행되긴 실행된다.
그거보다 return “empty”가 더 빨리 튀어나가는게 문제였다.

request handler에서 content를 리턴하는  대신
response에 직접 때려박으면 해결!

# response 옮기기 – server.js

```
var http = require("http");
var url = require("url");
function start(route, handle) {
    function onRequest(request, response) {
        var pathname = url.parse(request.url).pathname;
        console.log("request for " + pathname + " received");
        var content = route(handle, pathname, response);
    }
    http.createServer(onRequest).listen(8888);
    console.log("Server started");
}
exports.start = start;
```

# response 옮기기 – router.js

```
function route(handle, pathname, response) {
    console.log("About to route a request for " + pathname);
    if (typeof handle[pathname] == 'function') {
        return handle[pathname](response);
    }
    else {
        console.log("No such handle for request " + pathname);
        response.writeHead(404, { "Content-Type": "text/html" });
        response.write("<html><h1>Not found<br/></h1>you got wrong address baby</html>");
        response.end();
    }
}
exports.route = route;
```

# response 옮기기 – requestHandler.js

```
var exec = require("child_process").exec;
function start(response) {
    console.log("Request handler 'start' has called");
    exec("tree ..", function (error, stdout, stderr) {
        response.writeHead(200, { "Content-Type": "text/plain" });
        response.write(stdout);
        response.end();
    });
}
function upload(response) {
    console.log("Request handler 'upload' has called");
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write("<h1>Hello upload!</h1>");
    response.end();
}

exports.start = start;
exports.upload = upload;
```

# 결과

start는 결과를 받아오기 위해 최선을 다 하고 있고
→ 나중에 보면 결과 받아와서 페이지에 띄워져있고

그 와중에 upload에 들어가면 돌아간다.

(깨알 바뀐점 : writeHead 부분 text/plain -> text/html) 깨알이니깐 깨알만큼 작게써야지

# Web application?

~~HTTP Server~~

~~Router~~

~~Request handler~~

Request data handling

View logic

Upload handling

# Request data handling

User input을 받고

그에 대해 어떻게 처리할지 결정해주는 단계

네글자로 요약하면 : POST

# requestHandler.js → start(response)

```
function start(response) {
    console.log("Request handler 'start' has called");
    var body = '<html>' +
    '<head>' +
    '<meta http-equiv="Content-Type" content="text/html; ' +
    'charset=UTF-8" />' +
    '</head>' +
    '<body>' +
    '<form action="/upload" method="post">' +
    '<textarea name="text" rows="20" cols="60"></textarea>' +
    '<input type="submit" value="Submit text" />' +
    '</form>' +
    '</body>' +
    '</html>';
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write(body);
    response.end();
}
```

Submit text

# Get POST data from request

**Form Data** view source view URL encoded

**text:** my message!

# Get POST data from request

```
function onRequest(request, response) {
    var postData = "";
    var pathname = url.parse(request.url).pathname;
    console.log("request for " + pathname + " received");
    request.setEncoding('utf8');
    request.addListener("data", function (postDataChunk) {
        postData += postDataChunk;
        console.log("Received post data chunk '" + postDataChunk + "'.");
    });
    request.addListener("end", function () {
        route(handle, pathname, response, postData);
    });
}
```

# What we've done

1. Encode : UTF-8
2. add listener : post data start → "data"
3. add listener : post data end → "end"

# Get POST data from request

```
function route(handle, pathname, response, postData) {
    console.log("About to route a request for " + pathname);
    if (typeof handle[pathname] == 'function') {
        return handle[pathname](response, postData);
    }
    else {
        console.log("No such handle for request " + pathname);
        response.writeHead(404, { "Content-Type": "text/html" });
        response.write("<html><h1>Not found<br/></h1>you got wrong address baby</html>");
        response.end();
    }
}
exports.route = route;
```

# Get POST data from request

```
function upload(response, postData) {
    console.log("Request handler 'upload' has called");
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write("<h1>Your message : <br/></h1>" + postData);
    response.end();
}
```

# Not parsed :(

Your message :

text=my+message%21

# Clean data - querystring

```
function upload(response, postData) {
    console.log("Request handler 'upload' has called");
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write("<h1>Your message : <br/></h1>" + querystring.parse(postData).text);
    response.end();
}
```

# Web application?

~~HTTP Server~~

~~Router~~

~~Request handler~~

~~Request data handling~~

View logic

Upload handling

# Web application?

~~- HTTP Server~~

~~- Router~~

~~- Request handler~~

~~- Request data handling~~

- View logic : URL request → output
- Upload handling

# Web application?

~~- HTTP Server~~

~~- Router~~

~~- Request handler~~

~~- Request data handling~~

~~- View logic~~

- Upload handling

# Upload handling

사용자가 사진을 올리거나
귀중한 자료를 달린다거나
이런 post data를 처리할 수 있을까?

달린다

1-5

6-10

# Upload handling

음....

어.....

음.....

이게 어떻게 하는거지.....

# ~~Upload Handling~~ → Use node packages

사실 이건 나도 어찌 하는지는 잘 모름!

~~Socket Gay는 파싱 잘 해서 할 수도 있고 ㅌㅌㅌㅌ~~

이런 upload를 잘 처리할 수 있는 패키지가 있다고 함

펠릭스 가이젠되르뷔씨가 만든 node-formidable이란 패키지

이거를 써보자.

실습 전 : cp /home/suckzoo/test.png ./test.png (if you‘re working on bit)

# NPM (Node Package Module)

apt-get처럼 모듈을 다운로드!

우리가 사용할 모듈 : node-formidable

npm install formidable

! : 이거 루트가 하는거 아닙니다 로컬패키지임.

! : 글로벌 패키지 설치 일부러 안함. (global : npm install –g formidable)

# 이제 우리가 할 일

formidable module은 request를 잘 파싱해준다.

request 자체를 requestHandler로 넘겨주자.

크게 다음으로 볼 수 있다.

1. router, server에서 request를 requestHandler로 넘겨주기
2. upload form 만들기
3. /show 만들기
4. formidable을 이용하여 /upload 만들기

# 1단계 : server, router → request 전달

server에서 request를 받고

받은 request를 router로 넘겨주고

router에서는 requestHandler로 넘겨주고

# Server.js

```
var http = require("http");
var url = require("url");
function start(route, handle) {
    function onRequest(request, response) {
        var pathname = url.parse(request.url).pathname;
        console.log("request for " + pathname + " received");
        route(handle, pathname, response, request);
    }
    http.createServer(onRequest).listen(8888);
    console.log("Server started");
}
exports.start = start;
```

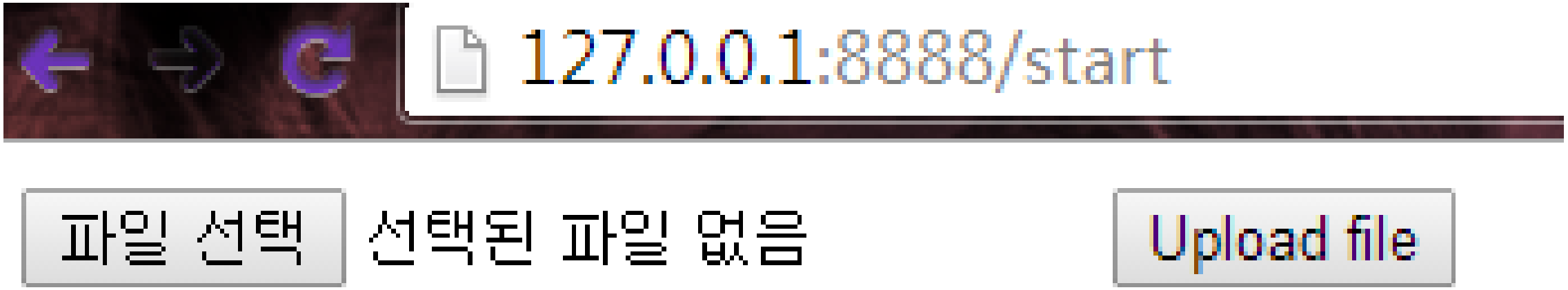
# router.js

는 직접 해보시죠

# router.js

```
function route(handle, pathname, response, request) {
    console.log("About to route a request for " + pathname);
    if (typeof handle[pathname] == 'function') {
        return handle[pathname](response, request);
    }
    else {
        console.log("No such handle for request " + pathname);
        response.writeHead(404, { "Content-Type": "text/html" });
        response.write("<html><h1>Not found<br/></h1>you got wrong address baby</html>");
        response.end();
    }
}
exports.route = route;
```

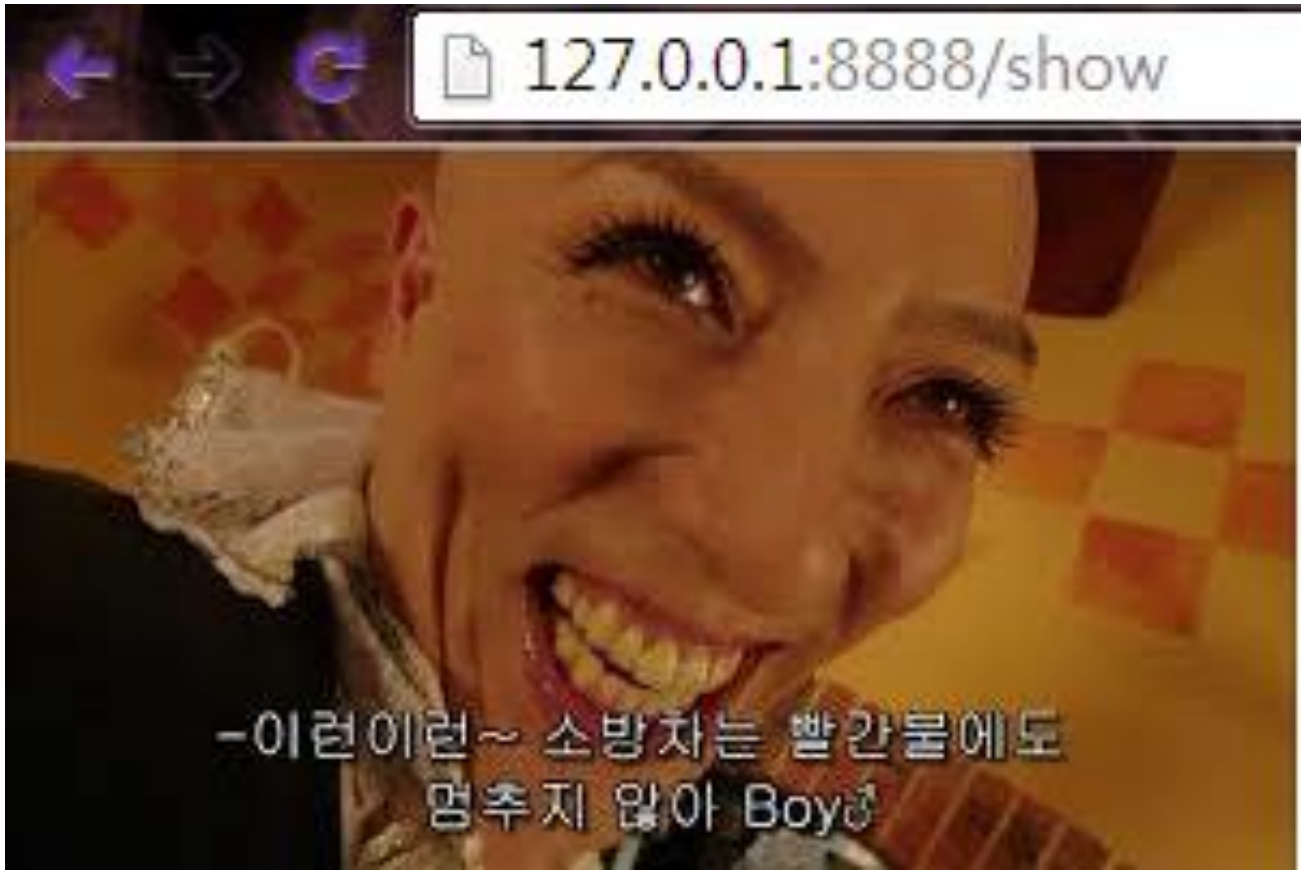
# 2단계 : 제출 form을 만든다



```
function start(response, request) {
    console.log("Request handler 'start' has called");
    var body = '<html>' +
    '<head>' +
    '<meta http-equiv="Content-Type" ' +
    'content="text/html; charset=UTF-8" />' +
    '</head>' +
    '<body>' +
    '<form action="/upload" enctype="multipart/form-data" ' +
    'method="post">' +
    '<input type="file" name="upload">' +
    '<input type="submit" value="Upload file" />' +
    '</form>' +
    '</body>' +
    '</html>';
    response.writeHead(200, { "Content-Type": "text/html" });
    response.write(body);
    response.end();
}
```

# 3단계 : 결과창 만들기

이미지를 binary로 열어서 /show에 띄우자



```
function show(response, request) {
    console.log("Request handler 'show' has called");
    fs.readFile("./test.png", "binary", function (error, file) {
        if (error) {
            response.writeHead(500, { "Content-Type": "text/plain" });
            response.write(error + "\n");
            response.end();
        } else {
            response.writeHead(200, { "Content-Type": "image/png" });
            response.write(file, "binary");
            response.end();
        }
    });
}
```

# 4단계 : formidable 이용하기.

다들 이제 require 쓰는 거 정도는 알잖아요?

```
function upload(response, request) {
    console.log("Request handler 'upload' has called");
    var form = new formidable.IncomingForm();
    console.log("about to parse...");
    form.parse(request, function (error, fields, files) {
        console.log("parsing done");
        fs.renameSync(files.upload.path, "../test.png");
        response.writeHead(200, { "Content-Type": "text/html" });
        response.write("received image :<br/>");
        response.write("<img src='/show' />");
        response.end();
    });
}
```

# 실행 ㄱㄱ

직접 다 해보시죠

# Web application?

~~HTTP Server~~

~~Router~~

~~Request handler~~

~~Request data handling~~

~~View logic~~

~~Upload handling~~ → ~~Using NPM~~

# 오늘 세미나의 목적

Understanding web application with node.js!

Mat-bo-gi of Node.js!

# References

http://www.nodebeginner.org/index-kr.html

Jimin Park

# 다음 세미나

사실 노드도 후레임 와아-크가 있다.

익스-프뤠스.줴이에-스를 해보자.