

진짜 기초



NHN NEXT 김우진

# node.js?

“서버사이드 JavaScript”

# 목표

“서버가 하는 일을 익히는 것.”

“node.js의 기본적인 동작을 익히는 것.”

# Hello World!

```
// helloworld.js  
console.log("Hello World!");
```

```
// terminal  
$ node helloworld.js
```

# 우리가 만들 거.

이름을 입력하세요.

입력



안녕하세요. 김우진 님

<http://localhost:8888/start>

<http://localhost:8888/hello>

쉽죠?

# 우리가 필요한 거.

요청



HTTP서버 : 웹페이지 제공

라우터 : 요청과 요청을 처리할 핸들러들을 연결

요청 핸들러 : 요청을 만족시키기위한 핸들러

뷰 로직 : 브라우저로 콘텐츠를 만들어 보내기 위한 로직



응답

# 기본서버 만들기.

```
// 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(8888);
```

```
// terminal
```

```
$ node server.js
```

# 기본서버 만들기. - 함수전달

```
// server.js
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(8888);
console.log("Server has started.");
```



이 함수는 클라이언트의 요청을 받으면  
거꾸로 호출(callback)된다.



# 기본서버 만들기. - 모듈화

```
// 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();
  }

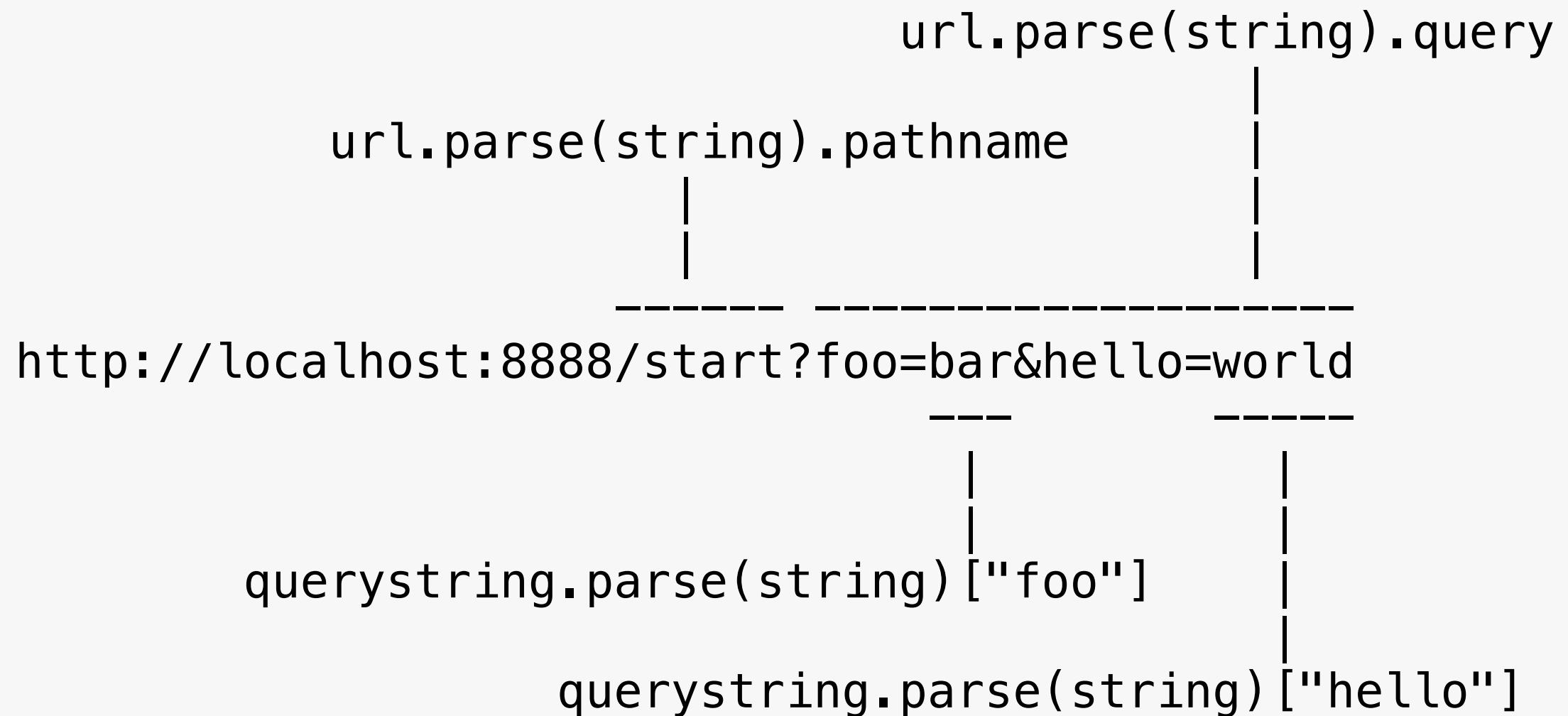
  http.createServer(onRequest).listen(8888);
  console.log("Server has started.");
}

exports.start = start;
```

# 기본서버 만들기. - 모듈화

```
// index.js  
var server = require("./server");  
  
server.start();
```

# 요청을 route



# 요청을 route

```
// server.js
var http = require("http");
var url = require("url");

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

# 요청을 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;
```

# 요청을 route

```
// router.js
function route(pathname) {
  console.log("About to route a request for " + pathname);
}

exports.route = route;
```

# 요청을 route

```
// index.js  
var server = require("./server");  
var router = require("./router");  
  
server.start(router.route);
```

# request handler

요청별로 다른 함수에서 처리하고 싶다!

```
// requestHandlers.js
function start() {
  console.log("Request handler 'start' was called.");
}

function hello() {
  console.log("Request handler 'hello' was called.");
}

exports.start = start;
exports.hello = hello;
```



# request handler

요청별로 다른 함수에서 처리하고 싶다!

```
// index.js
var server = require("./server");
var router = require("./router");
var requestHandlers = require("./requestHandlers");

var handle = {}
handle["/"] = requestHandlers.start;
handle["/start"] = requestHandlers.start;
handle["/hello"] = requestHandlers.hello;

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

# request handler

요청별로 다른 함수에서 처리하고 싶다!

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

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

# request handler

요청별로 다른 함수에서 처리하고 싶다!

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

exports.route = route;
```

# request handler가 응답하게

```
// requestHandlers.js
function start() {
  console.log("Request handler 'start' was called.");
  return "Hello Start";
}

function hello() {
  console.log("Request handler 'hello' was called.");
  return "Hello Hello";
}

exports.start = start;
exports.hello = hello;
```

# request handler가 응답하게

```
// route.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 request handler found for " + pathname);
    return "404 Not found";
  }
}

exports.route = route;
```

# request handler가 응답하게

```
// 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.");
    response.writeHead(200, {"Content-Type": "text/plain"});
    var content = route(handle, pathname);
    response.write(content);
    response.end();
  }

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

exports.start = start;
```

# Blocking 과 Non-Blocking

```
// requestHandlers.js
function start() {
  console.log("Request handler 'start' was called.");

  function sleep(milliseconds) {
    var startTime = new Date().getTime();
    while (new Date().getTime() < startTime + milliseconds);
  }

  sleep(10000);

  // 이 함수가 리턴 할 때까지 서버가 기다린다.
  return "Hello Start";
}
```

동시에 여러명이 접속할 때 문제가 발생!

# Blocking 과 Non-Blocking

“node.js는 **Single Thread**,  
동시작업을 **Event Loop**를 실행하여 처리”

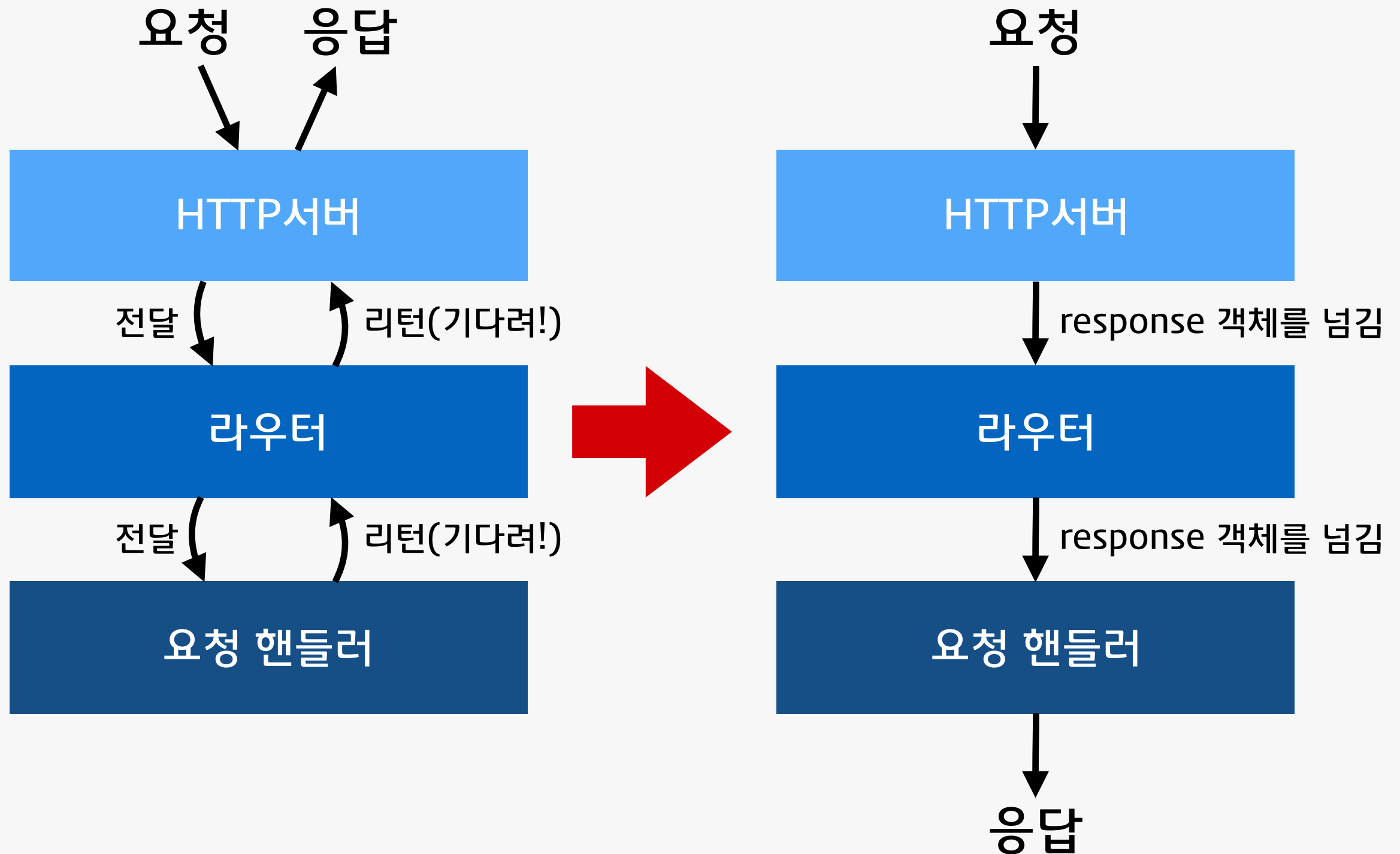


# Blocking 과 Non-Blocking

“헤이, `probablyExpensiveFunction()`, 니 일을 해줘. 하지만 나 Single Node.js 스레드는 **네가 끝낼 때까지 여기서 기다리지 않을거야.** 네 아래에 있는 코드 라인을 계속 실행할거야. 그러니 **여기 이 `callbackFunction()`을 가져가서 네가 너의 비싼 일을 모두 끝냈을 때 호출(callback)해 주겠니? 고마워!**”

# Blocking 과 Non-Blocking

요청 핸들러에서 응답을 처리하게 -> Non-Blocking으로 통제가능



# Blocking 과 Non-Blocking

```
// 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);
  }

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

exports.start = start;
```

# Blocking 과 Non-Blocking

```
// router.js
function route(handle, pathname, response) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}

exports.route = route;
```

# Blocking 과 Non-Blocking

```
// requestHandlers.js
function start(response) {
  console.log("Request handler 'start' was called.");
  // Non-Blocking 함수를 사용.
  setTimeout(function(){
    response.writeHead(200, {"Content-Type": "text/plain"});
    response.write("Hello Start");
    response.end();
  }, 10000);
}
```

```
function hello(response) {
  console.log("Request handler 'hello' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Hello");
  response.end();
}
```

```
exports.start = start;
exports.hello = hello;
```

해결! 하지만 큰 의미는 없으니 지웁시다.

# Blocking 과 Non-Blocking

```
// requestHandlers.js
function start(response) {
  console.log("Request handler 'start' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Start");
  response.end();
}

function hello(response) {
  console.log("Request handler 'hello' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello Hello");
  response.end();
}

exports.start = start;
exports.hello = hello;
```

# 뷰 로직을 만들고 연동하기.

```
// requestHandlers.js
function start(response) {
  console.log("Request handler 'start' was called.");

  var body = '<html>' + '<head>' +
    '<meta http-equiv="Content-Type" content="text/html; ' +
    'charset=UTF-8" />' +
    '</head>' + '<body>' +
    '이름을 입력하세요.' + '<br>' +
    '<form action="/hello" method="post">' +
    '<input type="text" name="text"></input>' +
    '<input type="submit" value="입력" />' +
    '</form>' + '</body>' + '</html>';

  response.writeHead(200, {"Content-Type": "text/html"});
  response.write(body);
  response.end();
}
```

# 뷰 로직을 만들고 연동하기.

POST Data를 Non-Blocking으로 받기.

```
request.addListener("data", function(chunk) {  
    // called when a new chunk of data was received  
});
```

```
request.addListener("end", function() {  
    // called when all chunks of data have been received  
});
```



# 뷰 로직을 만들고 연동하기.

POST Data를 Non-Blocking으로 받기.

```
// server.js
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);
  });
}
```

# 뷰 로직을 만들고 연동하기.

POST Data를 Non-Blocking으로 받기.

```
// router.js
function route(handle, pathname, response, postData) {
  console.log("About to route a request for " + pathname);
  if (typeof handle[pathname] === 'function') {
    handle[pathname](response, postData);
  } else {
    console.log("No request handler found for " + pathname);
    response.writeHead(404, {"Content-Type": "text/plain"});
    response.write("404 Not found");
    response.end();
  }
}
```

# 뷰 로직을 만들고 연동하기.

POST Data를 받아서 사용하기.

```
// requestHandlers.js
function hello(response, postData) {
  console.log("Request handler 'hello' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("안녕하세요. " + postData + "님");

  response.end();
}
```

# 뷰 로직을 만들고 연동하기.

POST Data를 받아서 사용하기. -> text만 가져오기.

```
// requestHandlers.js
var querystring = require("querystring");

// ...

function hello(response, postData) {
  console.log("Request handler 'upload' was called.");
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("안녕하세요. " +
    querystring.parse(postData).text + "님");

  response.end();
}
```

# 공부하기

GET Method와 POST Method의 차이점

# 과제

같은 로직을 GET Method를 사용하여 만들어 보기.

# reference

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



**KEEP  
CALM  
AND  
YNCA!**

**끝**