

Node.js 개요 및 실습

송실대학교 AI융합학부
School of AI Convergence

1. node.js 개요와 설치

❖ node.js 개요와 설치

- 2008년에 크롬 웹 브라우저가 출시되면서 자바스크립트의 속도가 점점 빨라지자 자바스크립트를 웹 브라우저가 아닌 곳에서 쓸 수 있게 만들자는 의견이 많아지면서 CommonJS 표준이 작성
- node.js는 CommonJS 표준에 따라 라이언 달(Ryan Dahl)이 크롬 V8 엔진을 기반으로 개발한 플랫폼
- node.js를 사용하면 웹 브라우저가 아닌 곳에서 자바스크립트로 프로그램을 개발할 수 있음

그림 19-6 node.js



1. node.js 개요와 설치

❖ node.js 개요와 설치

- node.js를 설치
- node.js의 공식 홈페이지는 <http://nodejs.org/>
- 공식 홈페이지에서 Install 버튼을 눌러 설치 파일을 내려받기

그림 19-7 <http://nodejs.org/>



그림 19-8 node.js 설치



1. node.js 개요와 설치

❖ node.js 개요와 설치

- 설치가 완료되었으면 명령 프롬프트를 실행
- 적당한 폴더를 생성하고 폴더에서 shift를 누르고 마우스 오른쪽 버튼을 누른 후 그림 19-9처럼 [여기서 명령 창 열기]를 누르면 명령 프롬프트가 실행

그림 19-9 여기서 명령 창 열기

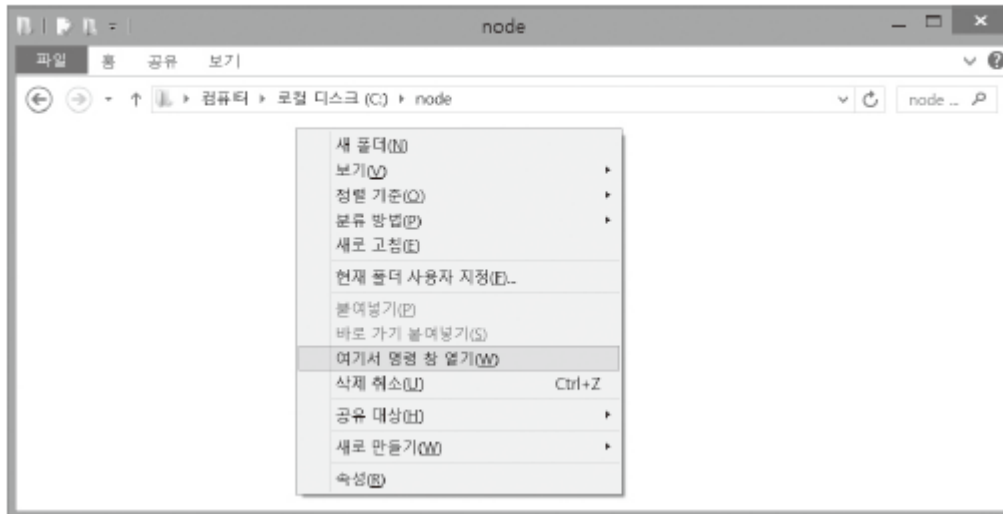


그림 19-10 명령 프롬프트



1. node.js 개요와 설치

❖ node.js 개요와 설치

- 명령 프롬프트에서 다음 node 명령어를 입력
- 명령어를 입력했을 때 그림 19-11처럼 출력하면 node.js가 정상적으로 설치

```
> node
```

그림 19-11 node 명령어를 사용한 REPL 실행



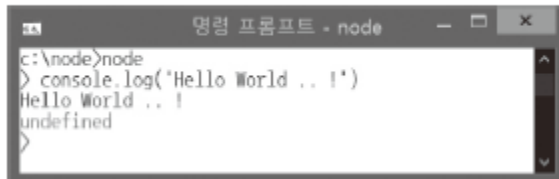
1. node.js 개요와 설치

❖ node.js 개요와 설치

- 명령 프롬프트에 node 명령어를 입력하면 REPL가 실행
- 이곳에는 자바스크립트 코드를 한 줄씩 입력해서 실행할 수 있음

```
> console.log('Hello World .. !')
```

그림 19-12 REPL을 사용한 Hello World 예제



```
명령 프롬프트 - node
C:\node>node
> console.log('Hello World .. !')
Hello World .. !
undefined
>
```

2. 기본 파일 실행

❖ 기본 파일 실행

- 파일을 만들어 실행하는 방법

그림 19-15 폴더 구성



- JavaScript.js 파일 구성

코드 19-2 JavaScript.js 파일 구성

```
var output = '';  
for (var i = 0; i < 10; i++) {  
    console.log(output += '*');  
}
```

2. 기본 파일 실행

❖ 기본 파일 실행

- 코드를 모두 입력했으면 파일을 저장
- 폴더에서 를 누르고 마우스 오른쪽 버튼을 눌러 [여기서 명령 창 열기]를 선택

그림 19-16 여기서 명령 창 열기



2. 기본 파일 실행

❖ 기본 파일 실행

- 명령 프롬프트가 실행되면 다음 명령어를 입력해 실행

```
> node JavaScript.js
```

그림 19-17 실행 결과



3. 내부 모듈

❖ 내부 모듈 (Built-in modules)

- Node.js는 기능을 확장하고자 모듈이라는 개념을 사용
 - 애플리케이션을 이루는 기본 조각, 부품, library를 의미함
- 모듈 중에서 Node.js에 기본적으로 있는 모듈을 내부 모듈이라 함
- 내부 모듈은 <http://nodejs.org/api/>의 문서에서 확인 가능
- 예: os, path, url, querystring, crypto, util
 - https://www.w3schools.com/nodejs/ref_modules.asp

그림 19-18 <http://nodejs.org/api/>

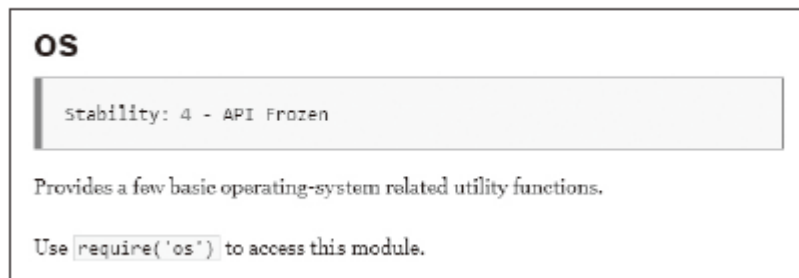


3. 내부 모듈

❖ 내부 모듈

- os 모듈을 사용하며 내부 모듈을 사용하는 방법

그림 19-19 os 모듈 페이지



코드 19-3 os 모듈 추출

```
// 모듈을 추출합니다.  
var os = require('os');  
  
// 속성을 출력합니다.  
console.log(os);
```

3. 내부 모듈

❖ 내부 모듈

- os 모듈 내부에 있는 속성과 메서드

표 19-4 os 모듈의 메서드

메서드 이름	설명
hostname()	운영체제의 호스트 이름을 리턴합니다.
type()	운영체제의 이름을 리턴합니다.
platform()	운영체제의 플랫폼을 리턴합니다.
arch()	운영체제의 아키텍처를 리턴합니다.
release()	운영체제의 버전을 리턴합니다.
uptime()	운영체제가 실행된 시간을 리턴합니다.
loadavg()	로드 에버리지 정보를 담은 배열을 리턴합니다.
totalmem()	시스템의 총 메모리를 리턴합니다.
freemem()	시스템의 사용 가능한 메모리를 리턴합니다.
cpus()	CPU의 정보를 담은 객체를 리턴합니다.
getNetworkInterfaces()	네트워크 인터페이스의 정보를 담은 배열을 리턴합니다.

3. 내부 모듈

❖ 내부 모듈

- os 모듈의 메서드

코드 19-4 os 모듈의 메서드

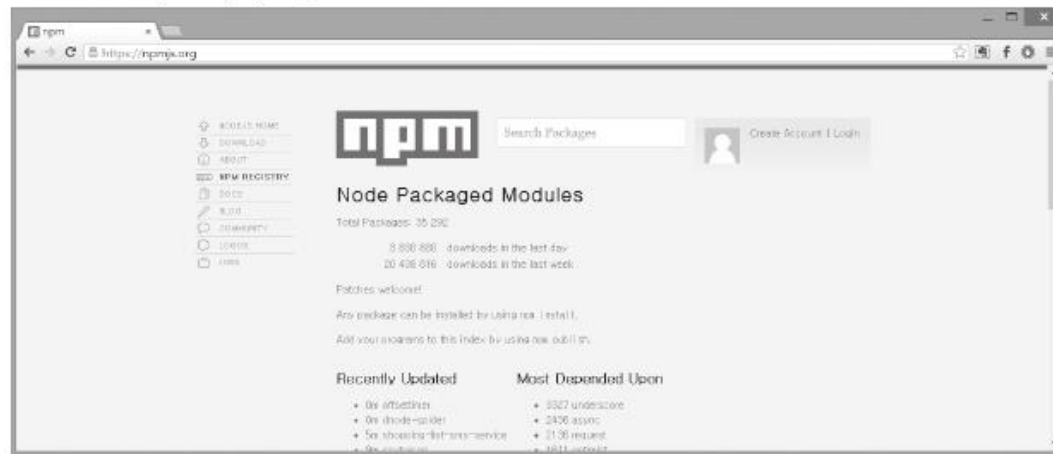
```
// 모듈을 추출합니다.  
var os = require('os');  
  
// 모듈을 사용합니다.  
console.log(os.hostname());  
console.log(os.type());  
console.log(os.platform());  
console.log(os.arch());  
console.log(os.release());  
console.log(os.uptime());  
console.log(os.loadavg());  
console.log(os.totalmem());  
console.log(os.freemem());  
console.log(os.cpus());  
console.log(os.getNetworkInterfaces());
```

4. 외부 모듈

❖ 외부 모듈(third-party modules, external modules)

- node.js가 기본적으로 갖지 않고 node 개발자인 개인 또는 단체가 만들어 배포하는 모듈을 외부 모듈이라 함
 - js file 형태로 구성됨
- NPM (Node Package Manager) : 외부 모듈 install, 유지관리 위해 사용
- 외부 모듈은 <https://npmjs.org/>에서 확인 가능
- 가장 많이 사용하는 외부 모듈(or packages)
 - Express, Socket.IO, MySQL, React, WebPack, Angular
 - <https://colorlib.com/wp/npm-packages-node-js/>

그림 19-22 <https://npmjs.org/>



4. 외부 모듈

❖ 외부 모듈

- 다음 명령어를 입력해 외부 모듈을 설치

```
> npm install 모듈명
```

- request 모듈 설치 후 다음과 같이 입력

```
> npm install request
```

- request module은 널리 사용된 모듈이나 현재 단계에서 다른 모듈을 추천
 - <https://github.com/request/request/issues/3143>

4. 외부 모듈

❖ 외부 모듈

- request 모듈 설치

그림 19-23 request 모듈 설치



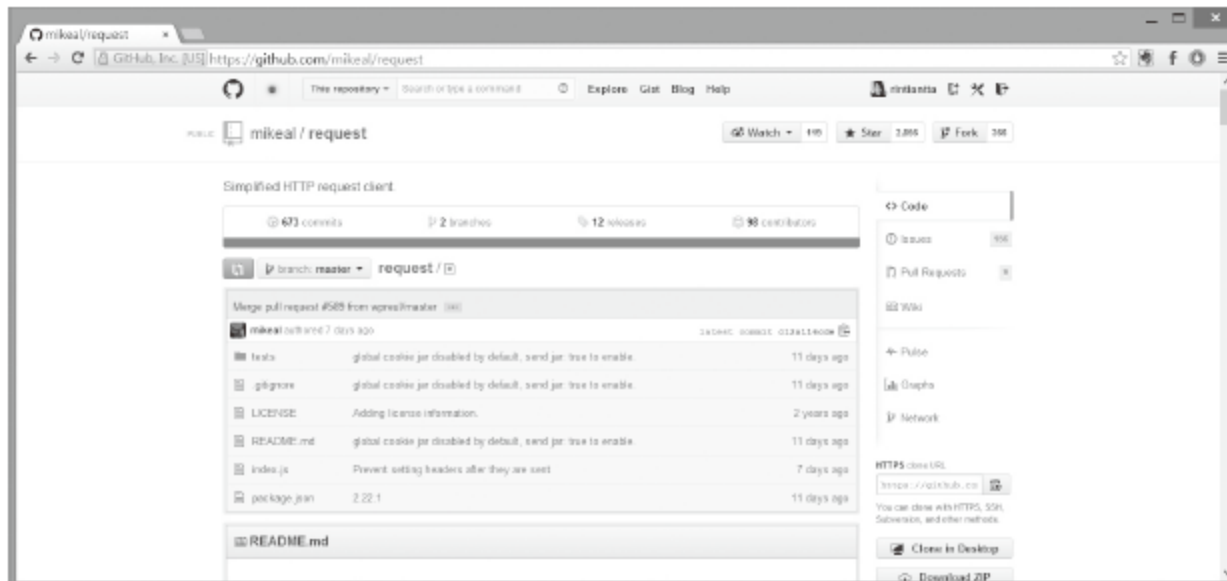
```
c:\node>npm install request
npm http GET https://registry.npmjs.org/request
npm http 304 https://registry.npmjs.org/request
npm http GET https://registry.npmjs.org/request/-/request-2.22.0.tgz
npm http 200 https://registry.npmjs.org/request/-/request-2.22.0.tgz
npm http GET https://registry.npmjs.org/node-uuid
npm http GET https://registry.npmjs.org/aws-sign
npm http GET https://registry.npmjs.org/tunnel-agent
npm http GET https://registry.npmjs.org/oauth-sign
npm http GET https://registry.npmjs.org/http-signature
npm http GET https://registry.npmjs.org/forever-agent
npm http GET https://registry.npmjs.org/qs
npm http GET https://registry.npmjs.org/json-stringify-safe
npm http GET https://registry.npmjs.org/cookie-jar
npm http GET https://registry.npmjs.org/mime
npm http GET https://registry.npmjs.org/form-data/0.0.8
npm http GET https://registry.npmjs.org/hawk
npm http 200 https://registry.npmjs.org/tunnel-agent
npm http 200 https://registry.npmjs.org/oauth-sign
npm http 200 https://registry.npmjs.org/http-signature
npm http GET https://registry.npmjs.org/http-signature/-/http-signature-0.1
npm http 200 https://registry.npmjs.org/node-uuid
```


4. 외부 모듈

❖ 외부 모듈

- 모듈과 관련된 설명은 모듈의 공식 홈페이지에서 볼 수 있음
- request 모듈의 공식 홈페이지는 <https://github.com/mikeal/request>
- <https://npmjs.org/>에서 찾을 수 있음

그림 19-24 <https://github.com/mikeal/request>



4. 외부 모듈

❖ 외부 모듈

- request 모듈의 기본 사용 방법

그림 19-25 request 모듈의 기본 사용 방법

Super simple to use

Request is designed to be the simplest way possible to make http calls. It supports HTTPS and follows redirects by default.

```
var request = require('request');
request('http://www.google.com', function (error, response, body) {
  if (!error && response.statusCode == 200) {
    console.log(body) // Print the google web page.
  }
})
```

- 공식 홈페이지에 있는 설명대로 코드 입력

코드 19-5 request 모듈

```
// 모듈을 추출합니다.
var request = require('request');

// 웹 페이지를 긁습니다.
request('http://www.google.com', function (error, response, body) {
  console.log(body);
});
```

4. 외부 모듈

❖ 외부 모듈

- 코드를 실행하면 그림 19-26처럼 구글 메인 페이지의 소스 코드를 출력

그림 19-26 request 모듈



```
c:\node>node JavaScript.js
<!doctype html><html itemscope="itemscope" itemtype="http://schema.org/WebPage"><head><meta itemprop="image" content="/images/google_favicon_128.png">
<title>Google</title><script>(function(){
window.google={kEI:"tYPmUfDSNoakkGwMyYBY",getEI:function(a){for(var b;a&&(!
a.getAttribute||(b=a.getAttribute("eid")));)a=a.parentNode;return b||google
e.kEI},https:function(){return"https:"+window.location.protocol},kEXPI:"17
259,4000116,4002855,4003242,4004205,4004334,4004844,4004948,4005865,4005875
,4006038,4006229,4006291,4006442,4006449,4006466,4006602,4006727,4007055,40
07080,4007117,4007173,4007232,4007244,4007566,4007638,4007661,4007668,40076
88,4007762,4007779,4007798,4007804,4007874,4007893,4007917,4008028,4008041,
4008059,4008067,4008079,4008115,4008133,4008142,4008170,4008184,4008191,400
8268,4008290,4008396,4008424,4008495,4008526,4008569,4008639,4008698".kcsi:
[e:"17259,4000116,4002855,4003242,4004205,4004334,4004844,4004948,4005865,4
005875,4006038,4006229,4006291,4006442,4006449,4006466,4006602,4006727,4007
055,4007080,4007117,4007173,4007232,4007244,4007566,4007638,4007661,4007668
,4007688,4007762,4007779,4007798,4007804,4007874,4007893,4007917,4008028,40
08041,4008059,4008067,4008079,4008115,4008133,4008142,4008170,4008184,40081
91,4008268,4008290,4008396,4008424,4008495,4008526,4008569,4008639,4008698"
,ei:"tYPmUfDSNoakkGwMyYBY"},authuser:0,ml:function(){},kHL:"ko",time:functi
on(){return(new Date).getTime()},log:function(a,b,c,l,k){var d=new Image,f=
google.lc,e=google.li,g="",h="gen_204";k&&(h=
```

5. 서버 생성 및 실행

❖ 서버 생성 및 실행

- server.js 파일을 생성

그림 19-27 폴더 구성



- 웹 서버를 만들 때는 express라는 외부 모듈 사용

```
> npm install express
```

- 명령어를 입력하면 그림 19-28처럼 모듈을 내려받고 설치

그림 19-28 express 모듈 설치

```
C:\windows\system32\cmd.exe
C:\node>npm install express
npm http GET https://registry.npmjs.org/express
npm http 304 https://registry.npmjs.org/express
npm http GET https://registry.npmjs.org/fresh/0.1.0
npm http GET https://registry.npmjs.org/debug
npm http 304 https://registry.npmjs.org/fresh/0.1.0
npm http 304 https://registry.npmjs.org/debug
```

5. 서버 생성 및 실행

❖ 서버 생성 및 실행

- server.js 파일에 코드 19-6처럼 입력해서 http 모듈과 express 모듈을 추출
 - http module : 내장 모듈, server 인스턴스 생성하는 createServer() 함수 제공
- 코드 19-7처럼 입력하면 웹 서버가 생성 및 실행

코드 19-6 모듈 추출

```
// 모듈을 추출합니다.  
var http = require('http');  
var express = require('express');
```

코드 19-7 기본 서버

```
// 모듈을 추출합니다.  
var http = require('http');  
var express = require('express');  
  
// 웹 서버를 생성합니다.  
var app = express();  
  
// 웹 서버를 실행합니다.  
http.createServer(app).listen(52273, function () {  
    console.log('Server Running at http://127.0.0.1:52273');  
});
```

포트를 지정합니다

5. 서버 생성 및 실행

❖ 서버 생성 및 실행

- 기본 응답
- 코드를 저장하고 명령 프롬프트를 실행
- 이어서 파일이 저장되어 있는 폴더에서 다음 명령어를 입력

코드 19-8 기본 응답

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 웹 서버를 생성합니다.
var app = express();
app.use(function (request, response) {
    response.send('<h1>안녕하세요!</h1>');
});

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

```
> node server.js
```

5. 서버 생성 및 실행

❖ 서버 생성 및 실행

- 서버 실행
- 웹 브라우저를 실행하고 `http://127.0.0.1:52273`에 접속
- 그림 19-30처럼 출력한다면 서버가 정상적으로 실행된 것임

그림 19-29 서버 실행



그림 19-30 `http://127.0.0.1:52273`



6. 미들웨어

❖ 미들웨어

- app.use () 메서드에 입력하는 콜백 함수는 request 이벤트 리스너
- 사용자가 서버에 접속하면 자동으로 실행
- request 이벤트 리스너는 다음 형태

```
app.use(function (request, response, next) {  
  
  });
```


6. 미들웨어

❖ 미들웨어

- app.use () 메서드를 세 번 사용

코드 19-9 미들웨어

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 웹 서버를 생성합니다.
var app = express();
app.use(function (request, response, next) { });
app.use(function (request, response, next) { });
app.use(function (request, response, next) { });

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

6. 미들웨어

❖ 미들웨어

- 각각의 콜백 함수는 코드 19-10처럼 입력

코드 19-10 미들웨어 구성

```
// 웹 서버를 생성합니다.  
var app = express();  
app.use(function (request, response, next) {  
  console.log('first');  
  next();  
});  
app.use(function (request, response, next) {  
  console.log('second');  
  next();  
});  
app.use(function (request, response, next) {  
  response.send('<h1>Hello Middleware .. !</h1>');  
});
```

6. 미들웨어

❖ 미들웨어

- 코드를 실행하고 웹 브라우저로 `http://127.0.0.1:52273`에 접속하면 웹 브라우저에 문자열 'Hello Middleware .. !'를 출력
- 중요한 것은 웹 브라우저가 아니라 명령 프롬프트 화면
- 각각의 콜백 함수를 통과하며 `console.log ()` 메서드를 실행한 것
- 사용자의 요청을 처리하면서 지나가는 `app.use ()` 메서드의 콜백 함수를 미들웨어라고 함

그림 19-34 `http://127.0.0.1:52273`



그림 19-35 미들웨어 확인



6. 미들웨어

❖ 미들웨어

- 이렇게 미들웨어를 사용하면 request 객체와 response 객체에 기능을 추가할 수 있음
- 코드 19-11은 첫 번째 미들웨어에서 request 객체와 response 객체에 test 속성을 추가하고 두 번째 미들웨어에서 이를 출력

코드 19-11 미들웨어 데이터 전달

```
// 웹 서버를 생성합니다.
var app = express();
app.use(function (request, response, next) {
  request.test = 'request.test';
  response.test = 'response.test';
  next();
});
app.use(function (request, response, next) {
  response.send('<h1>' + request.test + '::' + response.test + '</h1>');
});
```

6. 미들웨어

❖ 미들웨어

■ 미들웨어 활용

코드 19-12 미들웨어 활용

```
// 웹 서버를 생성합니다.  
var app = express();  
app.use(express.logger());  
app.use(express.bodyParser());  
app.use(express.cookieParser());  
app.use(express.session());  
app.use(express.static('public'));  
app.use(app.router);
```

7. 정적 파일 제공

❖ 정적 파일 제공

- static 미들웨어는 정적 파일을 제공할 때 사용하는 미들웨어
- 정적 파일이란 변화되지 않는 일반 파일을 의미
- 사용 방법 :
 - `app.use(express.static(path.join(__dirname, 'public')));`
 - 의미 : 정적파일이 있는 folder를 'public' 디렉토리로 지정
 - "`app.use(express.static('public'));`" 으로도 사용 가능 (권장 않함)
- 경로명 지정 가능
 - `app.use('/img', express.static(path.join(__dirname, 'public')));`
 - 의미 : public폴더 안에 abc.png가 있다고 가정하면 앞에 /img 경로를 붙인 `http://localhost:3000/img/abc.png` 주소로 접근 가능
 - 이 때, static 미들웨어는 요청에 부합하는 정적 파일을 발견한 경우 response로 해당 파일을 client에 전송
 - 이 경우 response를 client에 보내서 다음에 나오는 라우터가 실행되지 않음

7. 정적 파일 제공

❖ 정적 파일 제공

- static 미들웨어의 사용법

그림 19-38 프로젝트 폴더 구성



그림 19-39 public 폴더 구성



코드 19-13 index.html 파일 구성

```
<!DOCTYPE html>
<html>
<head>
  <title>node.js express</title>
</head>
<body>
  <h1>index.html</h1>
  <p>Lorem ipsum dolor sit amet.</p>
  <p>Cras dapibus velit nec est tempor.</p>
</body>
</html>
```

7. 정적 파일 제공

❖ 정적 파일 제공

- static 미들웨어의 사용법

코드 19-14 static 미들웨어 사용

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 웹 서버를 생성합니다.
var app = express();
app.use(express.static('public'));
app.use(function (request, response) {
    response.send('<h1>Hello Middleware .. !</h1>');
});

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

- 다음 명령어로 코드 실행

```
> node server.js
```


8. 라우터

❖ 라우터

- 라우트 한다 : 사용자의 요청에 따라 사용자가 필요한 정보를 제공하는 것
- 이러한 기능을 수행하는 미들웨어를 "라우터router"라고 부름
- router 미들웨어는 다음과 같이 사용함
- static 미들웨어와 다르게 app 객체의 속성이라는 것에 주의

코드 19-16 router 미들웨어 사용

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 웹 서버를 생성합니다.
var app = express();
app.use(express.static('public'));
app.use(app.router);

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

8. 라우터

❖ 라우터

■ app 객체의 메서드

표 19-5 app 객체의 메서드

메서드 이름	설명
app.get()	클라이언트의 GET 요청을 처리합니다.
app.post()	클라이언트의 POST 요청을 처리합니다.
app.put()	클라이언트의 PUT 요청을 처리합니다.
app.del()	클라이언트의 DELETE 요청을 처리합니다.
app.all()	클라이언트의 모든 요청을 처리합니다.

8. 라우터

❖ 라우터

■ all() 메서드

코드 19-17 all() 메서드

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 웹 서버를 생성합니다.
var app = express();
app.use(express.static('public'));
app.use(app.router);

// 라우트합니다.
app.all('/a', function (request, response) { });
app.all('/b', function (request, response) { });
app.all('/c', function (request, response) { });

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

8. 라우터

❖ 라우터

■ 라우트와 응답

코드 19-18 라우트와 응답

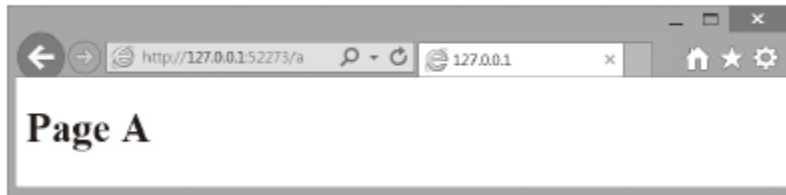
```
// 라우트합니다.  
app.all('/a', function (request, response) {  
    response.send('<h1>Page A</h1>')  
});  
app.all('/b', function (request, response) {  
    response.send('<h1>Page B</h1>')  
});  
app.all('/c', function (request, response) {  
    response.send('<h1>Page C</h1>')  
});
```

8. 라우터

❖ 라우터

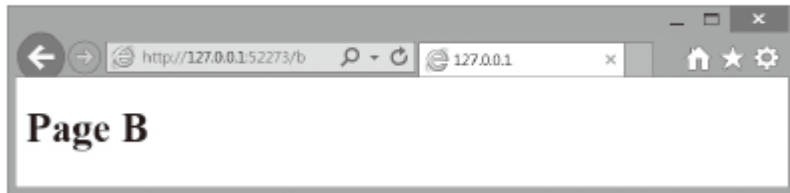
- 코드를 실행하고 `http://127.0.0.1:52273/a`에 접속

그림 19-43 `http://127.0.0.1:52273/a`



- 코드를 실행하고 `http://127.0.0.1:52273/b`에 접속

그림 19-44 `http://127.0.0.1:52273/b`



9. 응답과 응답 형식

❖ 응답과 응답 형식

- 정적 파일 제공과 라우터를 활용하는 방법
- 라우트를 하는 예제
 - /data.html 데이터를 HTML 형식으로 제공합니다.
 - /data.json 데이터를 JSON 형식으로 제공합니다.
 - /data.xml 데이터를 XML 형식으로 제공합니다.

9. 응답과 응답 형식

❖ 응답과 응답 형식

■ 기본 서버 구성

코드 19-19 기본 서버 구성

```
// 모듈을 추출합니다.
var http = require('http');
var express = require('express');

// 변수를 선언합니다.
var items = [];

// 웹 서버를 생성합니다.
var app = express();
app.use(express.static('public'));
app.use(app.router);

// 라우트합니다.
app.all('/data.html', function (request, response) { });
app.all('/data.json', function (request, response) { });
app.all('/data.xml', function (request, response) { });

// 웹 서버를 실행합니다.
http.createServer(app).listen(52273, function () {
    console.log('Server Running at http://127.0.0.1:52273');
});
```

9. 응답과 응답 형식

❖ 응답과 응답 형식

■ 변수 items 구성

코드 19-20 변수 items 구성

```
// 변수를 선언합니다.  
var items = [{  
    name: '우유',  
    price: '2000'  
}, {  
    name: '홍차',  
    price: '5000'  
}, {  
    name: '커피',  
    price: '5000'  
}];
```


9. 응답과 응답 형식

❖ HTML 응답

- 코드 19-21처럼 문자열 output을 생성하고 send () 메서드의 매개변수에 넣어줌

코드 19-21 HTML 응답

```
app.all('/data.html', function (request, response) {  
    var output = '';  
    response.send(output);  
});
```

9. 응답과 응답 형식

❖ HTML 응답

- 문자열 output은 코드 19-22처럼 조합

코드 19-22 HTML 문자열 생성

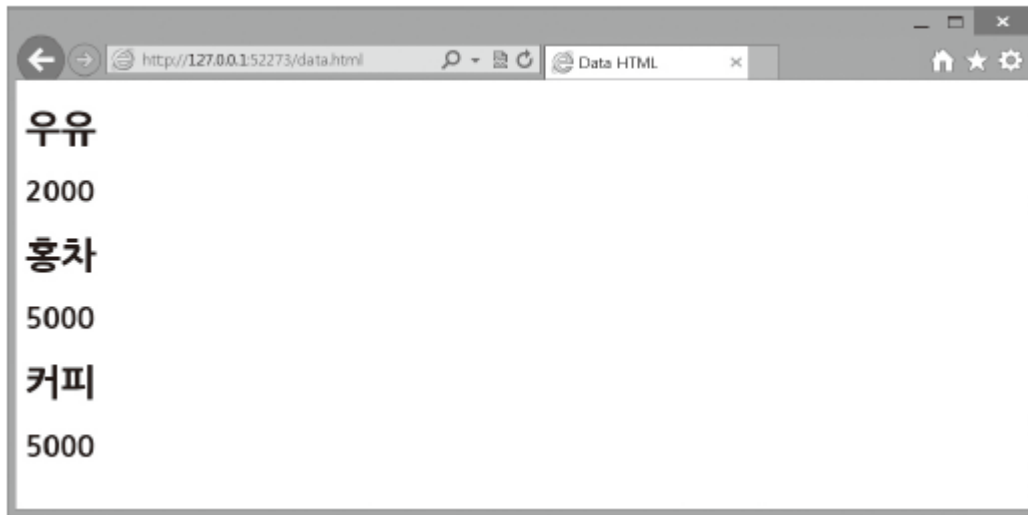
```
app.all('/data.html', function (request, response) {  
  var output = '';  
  output += '<!DOCTYPE html>';  
  output += '<html>';  
  output += '<head>';  
  output += '  <title>Data HTML</title>';  
  output += '</head>';  
  output += '<body>';  
  items.forEach(function (item) {  
    output += '<div>';  
    output += '  <h1>' + item.name + '</h1>';  
    output += '  <h2>' + item.price + '</h2>';  
    output += '</div>';  
  });  
  output += '</body>';  
  output += '</html>';  
  response.send(output);  
});
```

9. 응답과 응답 형식

❖ HTML 응답

- 서버를 실행하고 <http://127.0.0.1:52273/data.html>에 들어가면 그림 19-45처럼 출력

그림 19-45 <http://127.0.0.1:52273/data.html>



9. 응답과 응답 형식

❖ JSON 응답

- send() 메서드의 매개변수에 따른 응답 형식

표 19-6 send() 메서드의 매개변수에 따른 응답 형식

자료형	응답 형식
문자열	HTML
배열	JSON
객체	JSON

- JSON 응답

코드 19-23 JSON 응답

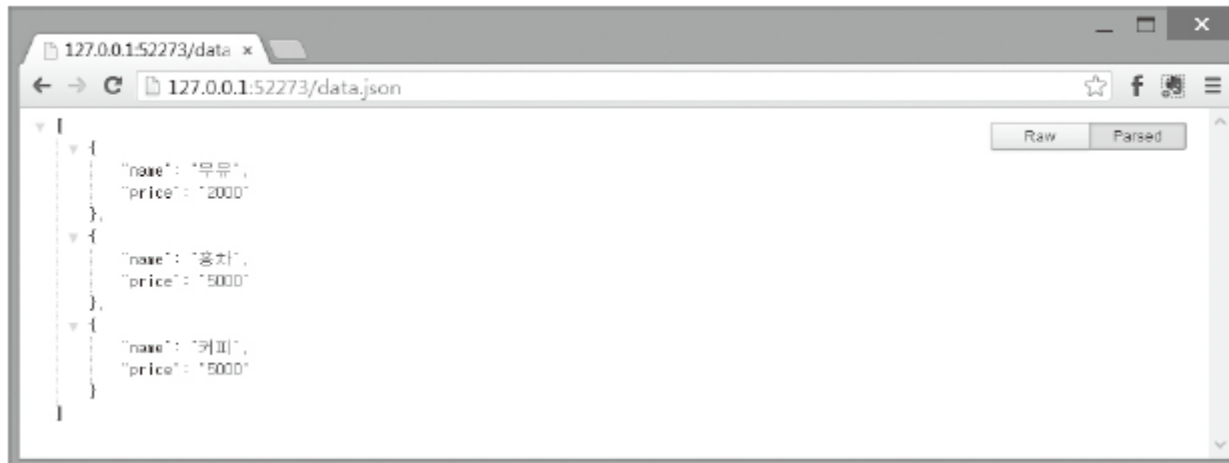
```
app.all('/data.json', function (request, response) {  
    response.send(items);  
});
```

9. 응답과 응답 형식

❖ JSON 응답

- 서버를 실행하고 `http://127.0.0.1:52273/data.json`에 접속하면 그림 19-46처럼 JSON 형식으로 데이터를 제공

그림 19-46 `http://127.0.0.1:52273/data.json`



9. 응답과 응답 형식

❖ XML 응답

■ XML 문자열 생성과 전달

코드 19-24 XML 문자열 생성과 전달

```
app.all('/data.xml', function (request, response) {
  var output = '';
  output += '<?xml version="1.0" encoding="UTF-8" ?>';
  output += '<products>';
  items.forEach(function (item) {
    output += '<product>';
    output += '  <name>' + item.name + '</name>';
    output += '  <price>' + item.price + '</price>';
    output += '</product>';
  });
  output += '</products>';
  response.send(output);
});
```

그림 19-47 http://127.0.0.1:52273/data.xml

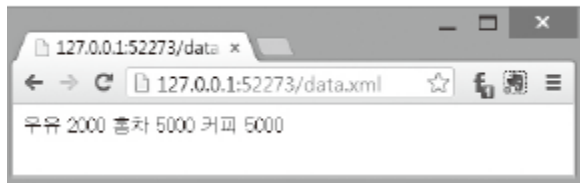


9. 응답과 응답 형식

❖ XML 응답

- XML 문자열을 인식하지 못하는 웹 브라우저

그림 19-48 XML 문자열을 인식하지 못하는 웹 브라우저



```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html; charset=utf-8
Content-Length: 255
Date: Thu, 11 Jul 2013 02:36:00 GMT
Connection: keep-alive
```

9. 응답과 응답 형식

❖ XML 응답

- type() 메서드를 사용한 응답 형식 지정

코드 19-25 type() 메서드를 사용한 응답 형식 지정

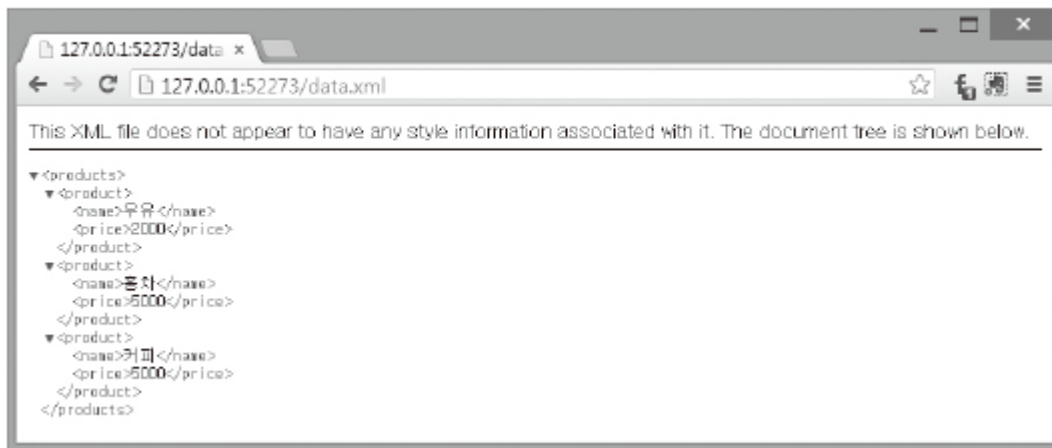
```
app.get('/data.xml', function (request, response) {  
  var output = '';  
  output += '<?xml version="1.0" encoding="UTF-8" ?>';  
  output += '<products>';  
  items.forEach(function (item) {  
    output += '<product>';  
    output += '  <name>' + item.name + '</name>';  
    output += '  <price>' + item.price + '</price>';  
    output += '</product>';  
  });  
  output += '</products>';  
  response.type('text/xml');  
  response.send(output);  
});
```


9. 응답과 응답 형식

❖ XML 응답

- 코드를 실행해 <http://127.0.0.1:52273/data.xml>에 접속

그림 19-49 정상 출력



HTTP/1.1 200 OK

X-Powered-By: Express

Content-Type: text/xml

Content-Length: 255

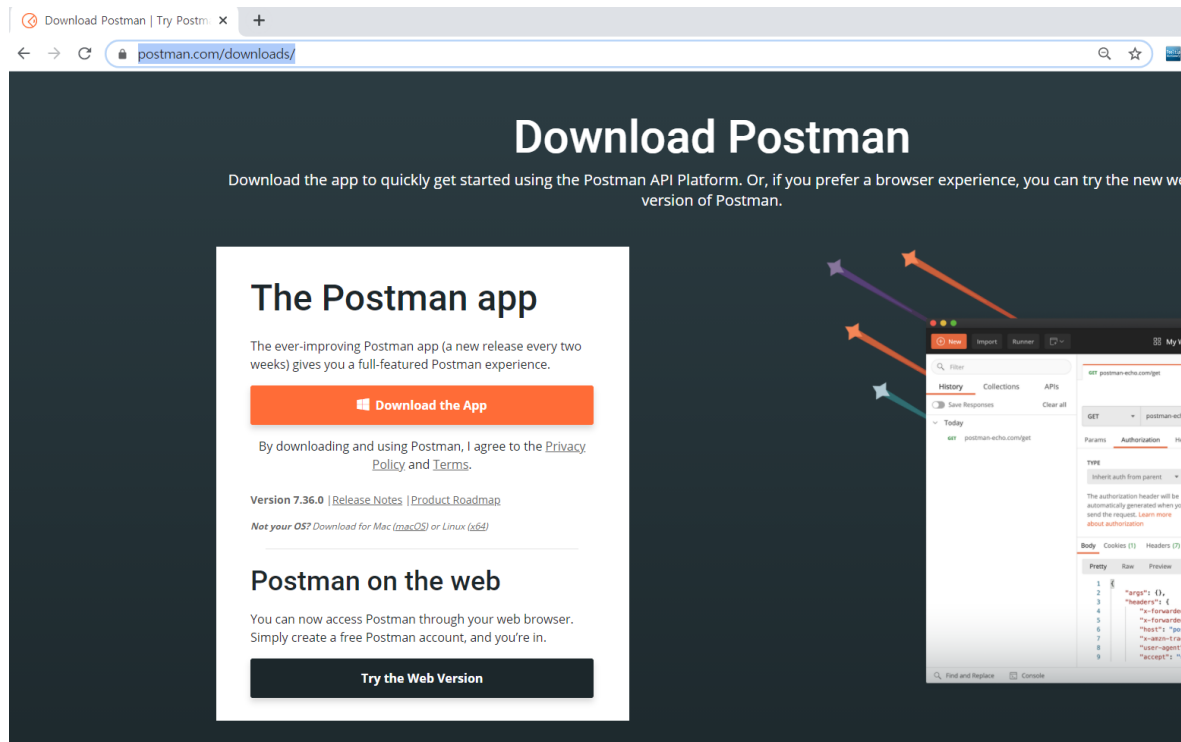
Date: Thu, 11 Jul 2013 02:40:04 GMT

Connection: keep-alive

10. Postman 프로그램

❖ Postman

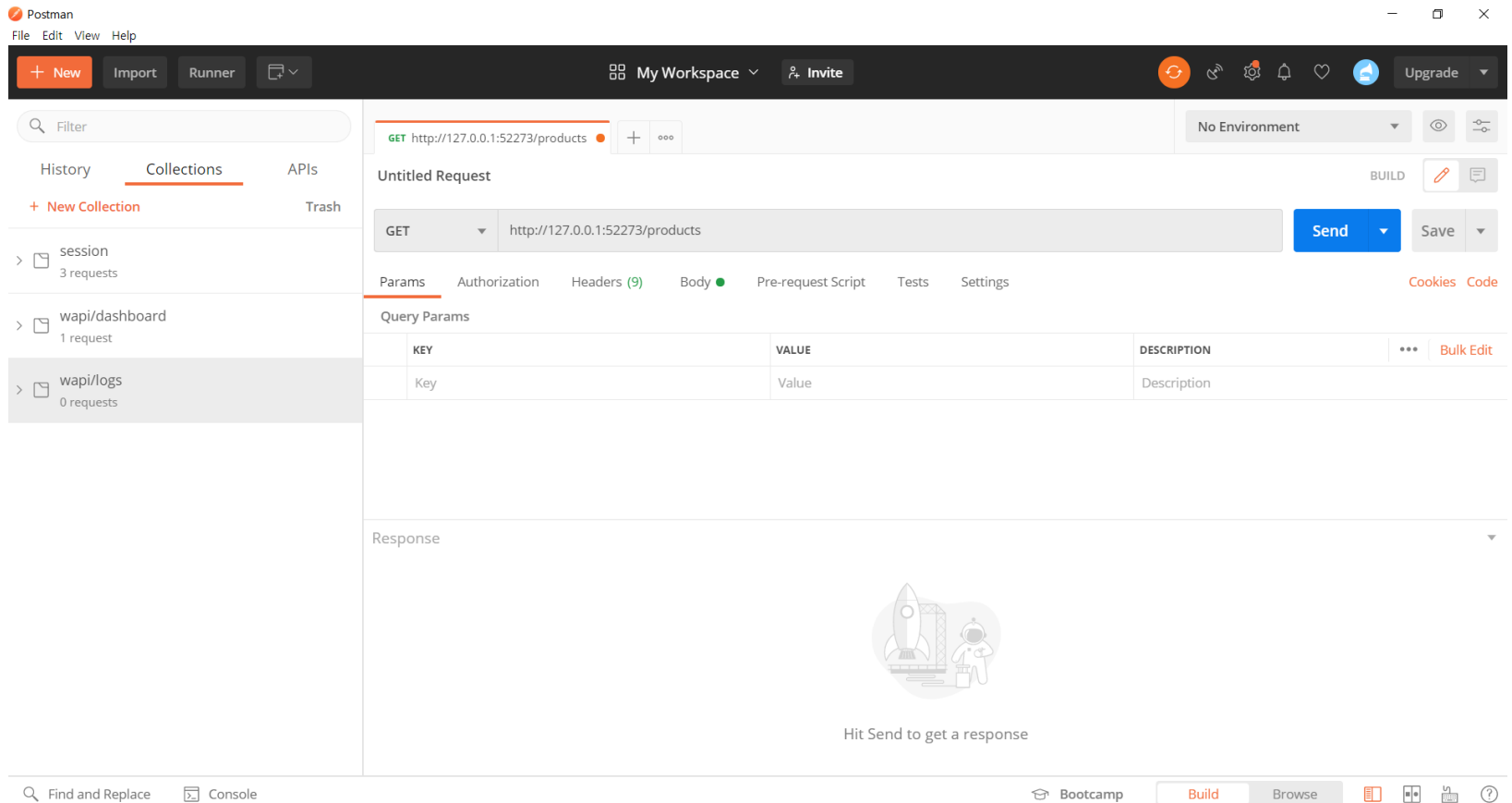
- Postman은 HTTP 요청을 수행하는 프로그램으로 API test tool로 사용
- Postman을 사용하면 손쉽게 HTTP 요청을 수행할 수 있음
- Postman download 하여 설치 (<https://www.postman.com/downloads/>)
- 코드를 실행해 <http://127.0.0.1:52273/data.xml>에 접속



10. Postman 프로그램

❖ Postman 프로그램

- 설치된 애플리케이션을 실행하면 다음과 같음



10. Postman 프로그램

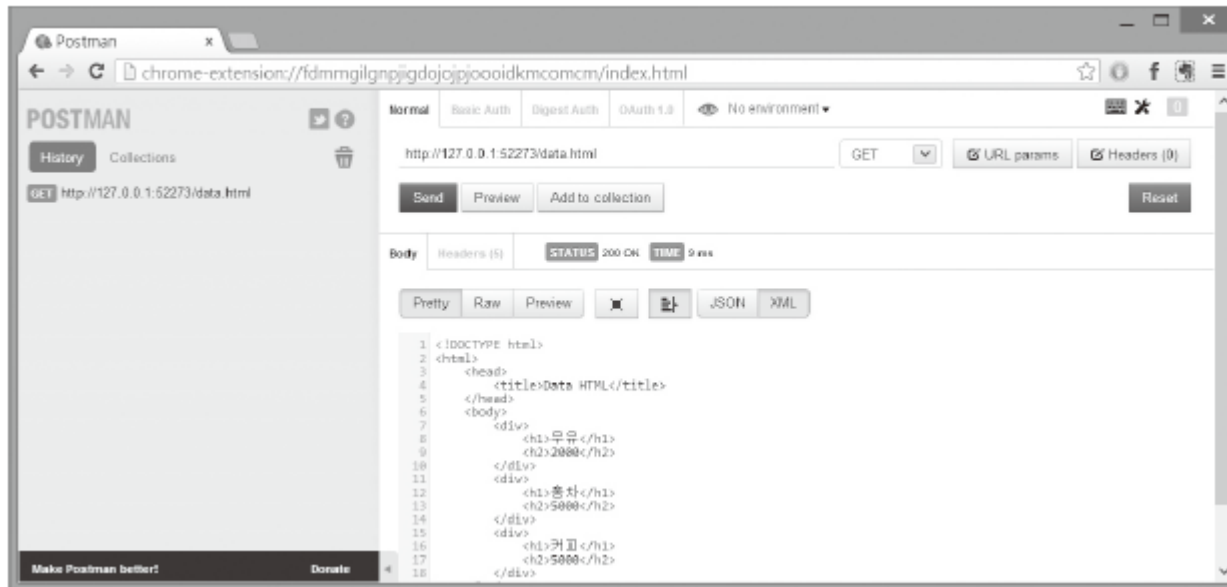
❖ Postman 프로그램

■ 요청 구성

그림 19-52 요청 구성



그림 19-53 요청



11. 요청과 요청 매개변수

❖ 요청과 요청 매개변수

- 클라이언트가 서버로 어떻게 데이터를 전달하는지 직접 확인

그림 19-54 다음 검색



`http://search.daum.net/search?w=tot&DA=YZRR&t__nil_searchbox=btn&sug=&q=html5`

`http://주소/경로?키A=값A&키B=값B`

11. 요청과 요청 매개변수

❖ 요청과 요청 매개변수

- 다음 검색의 요청 매개변수
- 클라이언트가 서버로 전달하는 데이터를 “요청 매개변수”라 함

표 19-8 다음 검색의 요청 매개변수

키	값
w	tot
DA	YZRR
t_nil_searchbox	byn
sug	(빈 문자열)
q	html5

11. 요청과 요청 매개변수

❖ 일반 요청 매개변수

- 일반 요청 매개변수는 이전에 살펴본 다음 검색처럼 키=값 블록으로 데이터를 전달하는 방법

코드 19-28 /parameter 라우트

```
app.all('/parameter', function (request, response) {  
  
  });
```

코드 19-29 param() 메서드

```
app.all('/parameter', function (request, response) {  
  // 변수를 선언합니다.  
  var name = request.param('name');  
  var region = request.param('region');  
  
  // 응답합니다.  
  response.send('<h1>' + name + ':' + region + '</h1>');  
});
```

11. 요청과 요청 매개변수

❖ 일반 요청 매개변수

■ 요청 매개변수 확인

그림 19-55 요청 매개변수 확인

The screenshot shows a web browser's developer tools interface. At the top, the URL bar displays `http://127.0.0.1:52273/parameter?name=InSung®ion=Seoul, Korea` with a dropdown menu set to `GET`. To the right of the URL bar are buttons for `URL params` and `Headers (0)`. Below the URL bar, a table lists the request parameters:

name	InSung	✕
region	Seoul, Korea	✕
URL Parameter Key	Value	

Below the table are buttons for `Send`, `Preview`, `Add to collection`, and `Reset`. The response section shows the `Body` tab selected, with `Headers (5)` visible. The status is `STATUS 200 OK` and the time is `TIME 28 ms`. Below the status bar are buttons for `Pretty`, `Raw`, `Preview`, and a `JSON` button. The response body is displayed as `<h1>InSung:Seoul, Korea</h1>`.

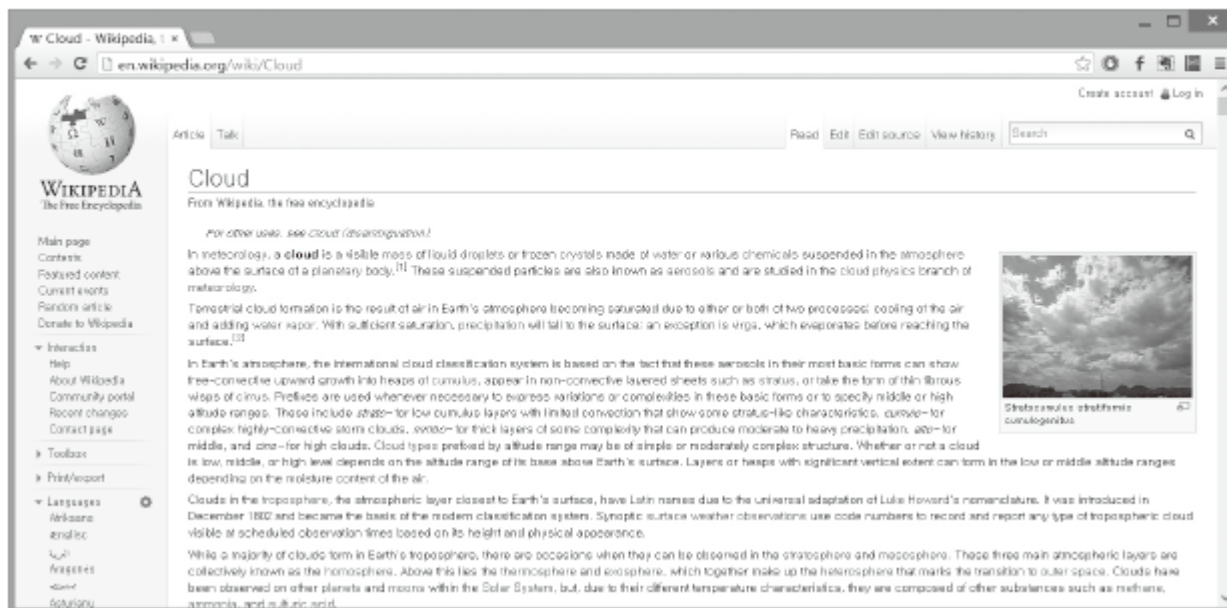
11. 요청과 요청 매개변수

❖ 동적 라우트 요청 매개변수

- 키=값 형태의 요청 매개변수를 사용하는 것이 아닌 경로에 직접 입력

- HTML 검색 <http://en.wikipedia.org/wiki/HTML>
- Cloud 검색 <http://en.wikipedia.org/wiki/Cloud>
- Sky 검색 <http://en.wikipedia.org/wiki/Sky>

그림 19-56 <http://en.wikipedia.org/wiki/Cloud>



11. 요청과 요청 매개변수

❖ 동적 라우트 요청 매개변수

- 동적 라우트 : 동적으로 변할수 있는 부분을 처리하는 라우트

코드 19-30 동적 라우트

```
app.all('/parameter/:id', function (request, response) {  
  // 변수를 선언합니다.  
  var id = request.param('id');  
  
  // 응답합니다.  
  response.send('<h1>' + id + '</h1>');  
});
```

동적 라우트합니다.

12. 요청 방식

❖ 요청 방식

- 동작을 나타내는 것은 "요청 방식method"로 처리

표 19-9 요청 방식

메서드	의미
GET	자원 조회
POST	자원 추가
PUT	자원 수정
DELETE	자원 삭제
HEAD	자원의 메타 데이터 조회
OPTIONS	사용 가능한 요청 방식 조회
TRACE	테스트 목적의 데이터 조회
CONNECT	연결 요청

12. 요청 방식

❖ 요청 방식

■ 요청 방식 처리 메서드

표 19-10 요청 방식 처리 메서드

메서드 이름	설명
app.get()	클라이언트의 GET 요청을 처리합니다.
app.post()	클라이언트의 POST 요청을 처리합니다.
app.put()	클라이언트의 PUT 요청을 처리합니다.
app.del()	클라이언트의 DELETE 요청을 처리합니다.

12. 요청 방식

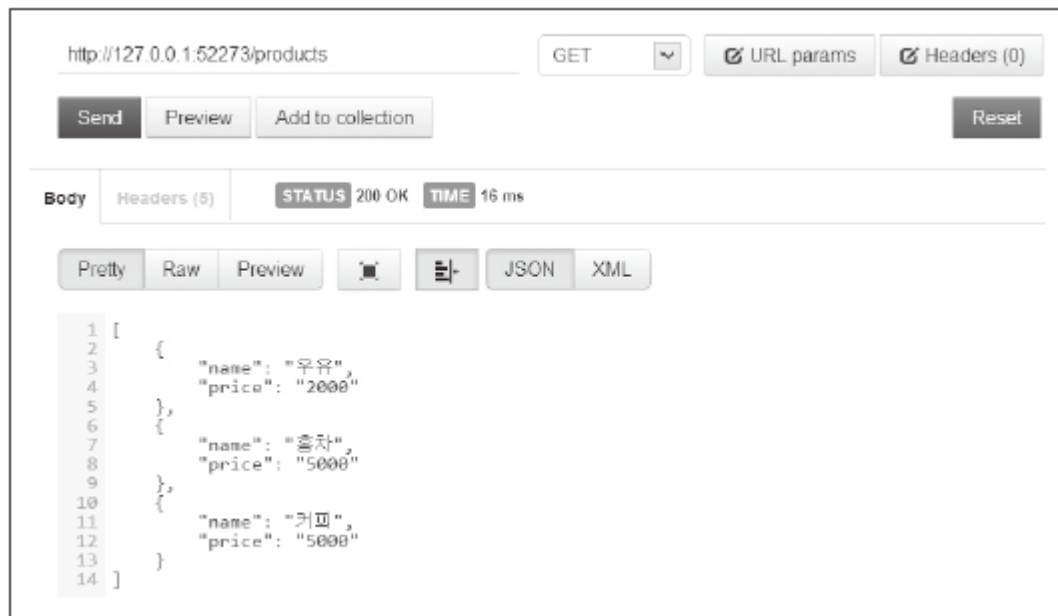
❖ 데이터 조회

■ 데이터 조회

코드 19-33 데이터 조회

```
app.get('/products', function (request, response) {  
    response.send(items);  
});
```

그림 19-58 데이터 조회



12. 요청 방식

❖ 데이터 조회

■ 개별 데이터 조회

코드 19-34 개별 데이터 조회

```
app.get('/products/:id', function (request, response) {  
    // 변수를 선언합니다.  
    var id = Number(request.param('id'));  
  
    // 응답합니다.  
    response.send(items[id]);  
});
```

12. 요청 방식

❖ 데이터 조회

■ 개별 데이터 조회 보완

코드 19-35 개별 데이터 조회 보완

```
app.get('/products/:id', function (request, response) {  
  // 변수를 선언합니다.  
  var id = Number(request.param('id'));  
  
  if (isNaN(id)) {  
    // 오류: 잘못된 경로  
    response.send({  
      error: '숫자를 입력하세요!'  
    });  
  } else if (items[id]) {  
    // 정상  
    response.send(items[id]);  
  } else {  
    // 오류: 요소가 없을 경우  
    response.send({  
      error: '존재하지 않는 데이터입니다!'  
    });  
  }  
});
```

12. 요청 방식

❖ 데이터 추가

■ 데이터 추가

코드 19-36 데이터 추가

```
app.post('/products', function (request, response) {  
    // 변수를 선언합니다.  
    var name = request.param('name');  
    var price = request.param('price');  
    var item = {  
        name: name,  
        price: price  
    };  
  
    // 데이터를 추가합니다.  
    items.push(item);  
  
    // 응답합니다.  
    response.send({  
        message: '데이터를 추가했습니다.',  
        data: item  
    });  
});
```


12. 요청 방식

❖ 데이터 수정

■ 데이터 수정

코드 19-37 데이터 수정

```
app.put('/products/:id', function (request, response) {
  // 변수를 선언합니다.
  var id = Number(request.param('id'));
  var name = request.param('name');
  var price = request.param('price');

  if (items[id]) {
    // 데이터를 수정합니다.
    if (name) { items[id].name = name; }
    if (price) { items[id].price = price; }

    // 응답합니다.
    response.send({
      message: '데이터를 수정했습니다.',
      data: items[id]
    });
  } else {
    // 오류: 요소가 없을 경우
    response.send({
      error: '존재하지 않는 데이터입니다!'
    });
  }
});
```

12. 요청 방식

❖ 데이터 삭제

■ 데이터 삭제

코드 19-38 데이터 삭제

```
app.del('/products/:id', function (request, response) {  
    // 변수를 선언합니다.  
    var id = Number(request.param('id'));  
  
    if (isNaN(id)) {  
        // 오류: 잘못된 경로  
        response.send({  
            error: '숫자를 입력하세요!'  
        });  
    } else if (items[id]) {  
        // 정상: 데이터 삭제  
        items.splice(id, 1);  
        response.send({  
            message: '데이터를 삭제했습니다.'  
        });  
    } else {  
        // 오류: 요소가 없을 경우  
        response.send({  
            error: '존재하지 않는 데이터입니다!'  
        });  
    }  
});
```

12. 요청 방식

❖ 클라이언트 페이지

■ 클라이언트 페이지

코드 19-39 index.html 파일 구성

```
<!DOCTYPE html>
<html>
<head>
  <title>Data Process</title>
</head>
<body>
  <form action="/products" method="get"></form>
  <form action="/products" method="post"></form>
</body>
</html>
```

코드 19-40 GET 요청 입력 양식

```
<form action="/products" method="get">
  <input type="submit" />
</form>
```

12. 요청 방식

❖ 클라이언트 페이지

■ 클라이언트 페이지

코드 19-41 POST 요청 입력 양식

```
<form action="/products" method="post">
  <input name="name" />
  <input name="price" />
  <input type="submit" />
</form>
```

그림 19-65 웹 페이지 기본 구성



The image shows a basic web form layout. At the top left, there is a button labeled '취리 전송' (Submit). Below it are two text input fields. To the right of the second input field, there is another button labeled '취리 전송' (Submit).

12. 요청 방식

❖ 클라이언트 페이지

■ 클라이언트 페이지

코드 19-42 웹 페이지 구성

```
<!DOCTYPE html>
<html>
<head>
  <title>Data Process</title>
</head>
<body>
  <form action="/products" method="get">
    <fieldset>
      <legend>GET</legend>
      <input type="submit" />
    </fieldset>
  </form>
  <form action="/products" method="post">
    <fieldset>
      <legend>POST</legend>
      <input name="name" />
      <input name="price" />
      <input type="submit" />
    </fieldset>
  </form>
</body>
</html>
```

12. 요청 방식

❖ 클라이언트 페이지

- 클라이언트 페이지

그림 19-67 GET 요청

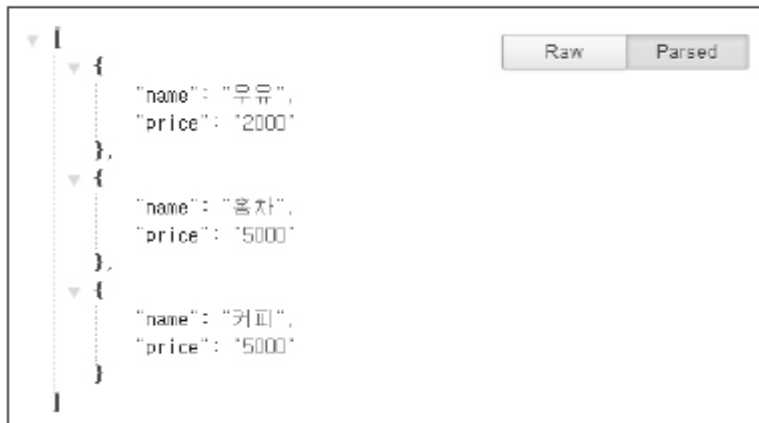
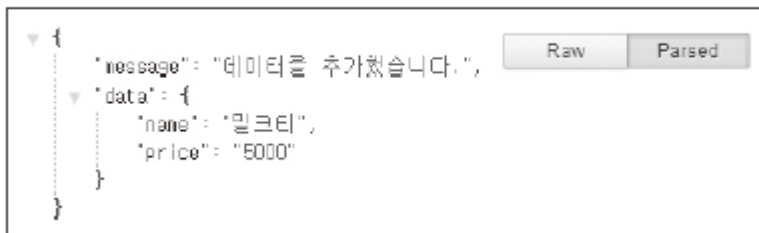


그림 19-68 POST 요청



13. 서버 정리

❖ 서버 정리

■ 서버 정리

그림 19-69 서버 정리

