

Node.js

Rest API - 김근형 강사

Rest API

► Rest API

{ REST API }

Representational State Transfer API

Rest API

▶ Rest API

- ▶ REST(Representational State Transfer)는 월드 와이드 웹과 같은 분산 하이퍼미디어 시스템을 위한 소프트웨어 아키텍처의 한 형식이다.
- ▶ 이 용어는 로이 필딩(Roy Fielding)의 2000년 박사학위 논문에서 그 당시 웹(HTTP) 설계의 우수성에 비해 제대로 사용되어지지 못하는 모습에 안타까워하며 웹의 장점을 최대한 활용할 수 있는 아키텍처로써 REST를 발표
- ▶ REST 서버는 클라이언트로 하여금 HTTP 프로토콜을 사용해 서버의 정보에 접근 및 변경을 가능케 합니다. 여기서 정보는 text, xml, json 등 형식으로 제공되는데 주로 json 형태의 데이터를 제공하여 비동기 통신을 하는데 사용이 된다.

Rest API

▶ Rest API 특징

- ▶ 서버에 있는 모든 **resource**는 각 **resource** 당 클라이언트가 바로 접근 할 수 있는 고유 **URI**가 존재한다.
- ▶ 모든 요청은 클라이언트가 요청할 때마다 필요한 정보를 주기 때문에 서버에서는 세션 정보를 보관할 필요가 없다. 그렇기 때문에 서비스에 자유도가 높아지고 유연한 아키텍처 적용이 가능하다.
- ▶ **HTTP** 메소드를 사용한다. 모든 **resource**는 일반적으로 **http** 인터페이스인 **GET, POST, PUT, DELETE** 4개의 메소드로 접근 되어야한다.
- ▶ 서비스 내에 하나의 **resource**가 주변에 연관 된 리소스들과 연결되어 표현이 되어야 한다.

Rest API

▶ Rest API 구성 요소

▶ 자원(Resource): URI

- ▶ 모든 자원에 고유한 ID가 존재하고, 이 자원은 Server에 존재한다.
- ▶ 자원을 구별하는 ID는 '/groups/:group_id'와 같은 HTTP URI 다.
- ▶ 여기서 슬래시 구분자(/)는 계층 관계를 나타내는 데 사용한다.
- ▶ 해당 계층 관계를 통해 그 사람이 어떤 것을 요청하는지 명시적으로 명확하게 알아야 한다.(중요)
- ▶ Client는 URI를 이용해서 자원을 지정하고 해당 자원의 상태(정보)에 대한 조작을 Server에 요청한다.

<http://www.happy-zoo/animals/dogs/john>

해석 : happy-zoo의 animal중 dog계열의 이름이 john

Rest API

▶ Rest API 구성 요소

▶ 행위(Verb): HTTP Method

- ▶ HTTP 프로토콜의 Method를 사용한다.
- ▶ HTTP 프로토콜은 GET, POST, PUT, DELETE 와 같은 메서드를 제공한다.
- ▶ 각각의 메서드가 하는 역할은 다음과 같다.
 - ▶ POST(생성) : POST를 통해 해당 URI를 요청하면 리소스를 생성한다.
 - ▶ GET(검색) : GET를 통해 해당 리소스를 조회한다. 리소스를 조회하고 해당 문서에 대한 자세한 정보를 가져온다.
 - ▶ PUT(수정) : PUT를 통해 해당 리소스를 수정한다.
 - ▶ DELETE(삭제) : DELETE를 통해 리소스를 삭제한다.

Rest API

▶ Rest API 구성 요소

▶ 행위(Verb): HTTP Method

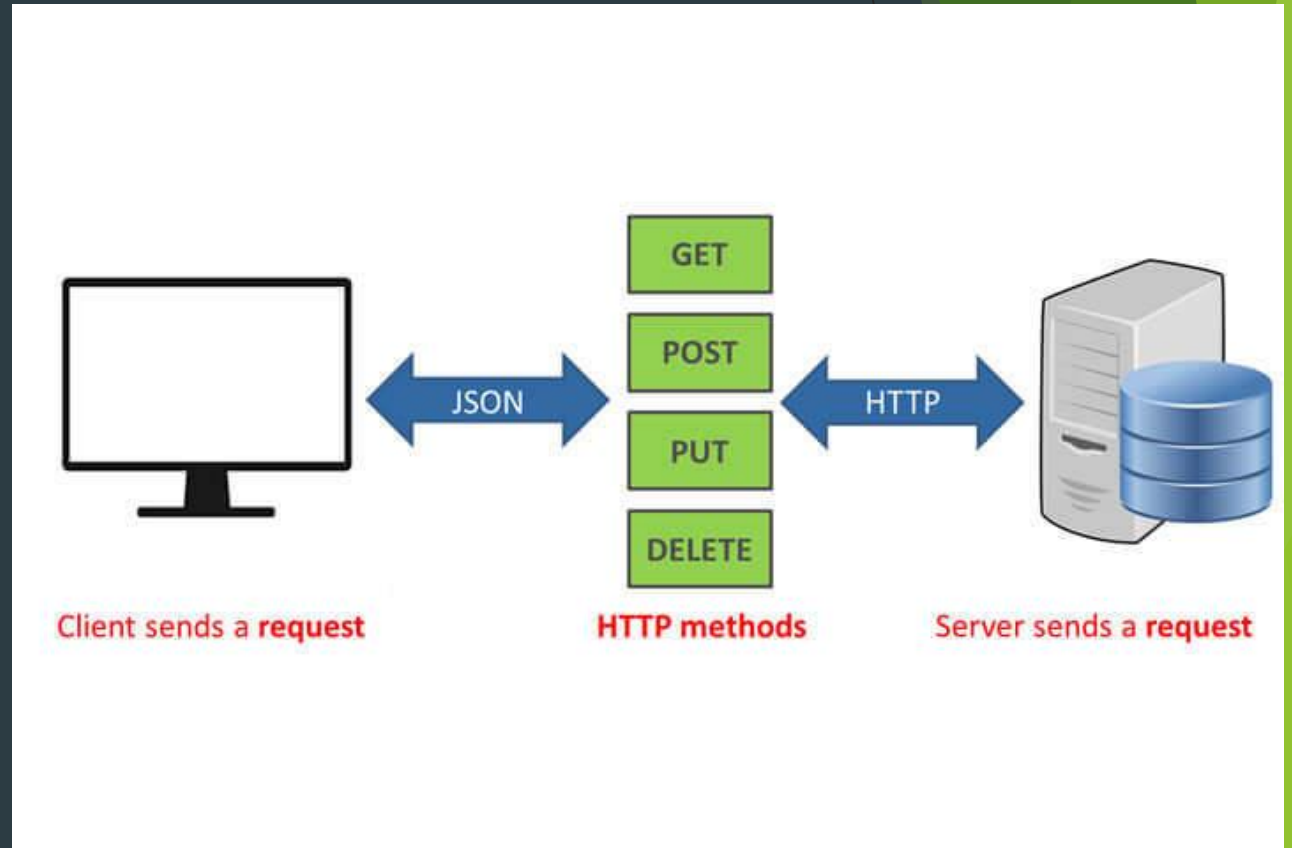
HTTP메소드	URI(자원)	Endpoint의 행위
POST	http://api.domain.com/books	새로운 도서정보 생성
GET	http://api.domain.com/books	도서정보 목록 조회
GET	http://api.domain.com/books/1	1번 도서정보 조회
PUT	http://api.domain.com/books/1	1번 도서정보 수정
DELETE	http://api.domain.com/books/1	1번 도서정보 삭제

Rest API

▶ Rest API 구성 요소

▶ 표현(Representation of Resource)

- ▶ Client가 자원의 상태(정보)에 대한 조작을 요청하면 Server는 이에 적절한 응답(Representation)을 보낸다.
- ▶ REST에서 하나의 자원은 JSON, XML, TEXT, RSS 등 여러 형태의 Representation으로 나타내어 질 수 있다.
- ▶ JSON 혹은 XML를 통해 데이터를 주고 받는 것이 일반적이다.



Metadata

▶ Metadata

- ▶ 데이터에 관한 구조화된 데이터로, 다른 데이터를 설명해 주는 데이터
- ▶ 대량의 정보 가운데에서 찾고 있는 정보를 효율적으로 찾아내서 이용하기 위해 일정한 규칙에 따라 콘텐츠에 대하여 부여되는 데이터
- ▶ 어떤 데이터 즉 구조화된 정보를 분석, 분류하고 부가적 정보를 추가하기 위해 그 데이터 뒤에 함께 따라가는 정보

Metadata

▶ Metadata 언어

- ▶ 다른 데이터를 기술하기 위해 사용되는 언어
- ▶ 타 시스템간의 연동, DB 동기화, 웹 호환성 향상에 도움을 주며 그 외에 많은 분야에서 쓰이고 있다
- ▶ 대표적 메타언어 : XML, JSON

XML(Extensible Markup Language)

▶ XML(Extensible Markup Language)

- ▶ W3C에서 여러 특수 목적의 마크업 언어를 만드는 용도에서 권장되는 다목적 마크업 언어
- ▶ 1996년 제안된 언어로, 기존의 **HTML**과 달리 웹상에서 구조화된 문서를 전송가능하도록 설계됨
- ▶ 데이터에 의미를 부여하는 메타데이터를 기술할 수 있다.
- ▶ 수많은 종류의 데이터를 유연하고 자유롭게 기술하는 데 적용할 수 있어서 다양한 용도로 응용할 수 있으며, 인터넷으로 연결된 시스템끼리 쉽게 식별 가능한 데이터를 주고받을 수 있게 됨

XML(Extensible Markup Language)

▶ XML 구조

```
<?xml version="1.0" encoding="euc-kr"?>  
<?xml:stylesheet type="text/xsl" href="booklist.xsl"?>  
<!DOCTYPE booklist SYSTEM "booklist.dtd">
```

서두
부분

```
<!-- 엘리먼트 부분입니다 -->  
<booklist>  
  <book kind="소설">  
    <title>시인과도둑</title>  
    <author>이문열</author>  
  </book>  
</booklist>
```

엘리먼트
부분

JSON(JavaScript Object Notation)

▶ JSON(JavaScript Object Notation)

- ▶ 속성-값 쌍으로 이루어진 데이터 오브젝트를 전달하기 위해 인간이 읽을 수 있는 텍스트를 사용하는 개방형 표준 포맷
- ▶ 비동기 브라우저/서버 통신 (AJAX)을 위해, 넓게는 XML을 대체하는 주요 데이터 포맷
- ▶ 인터넷에서 자료를 주고 받을 때 그 자료를 표현하는 방법으로 자주 쓰임.
- ▶ 자료의 종류에 큰 제한은 없으며, 특히 컴퓨터 프로그램의 변수 값을 표현하는 데 적합
- ▶ 자바스크립트 언어로부터 파생되어 자바스크립트의 구문 형식을 따르지만 언어 독립형 데이터 포맷

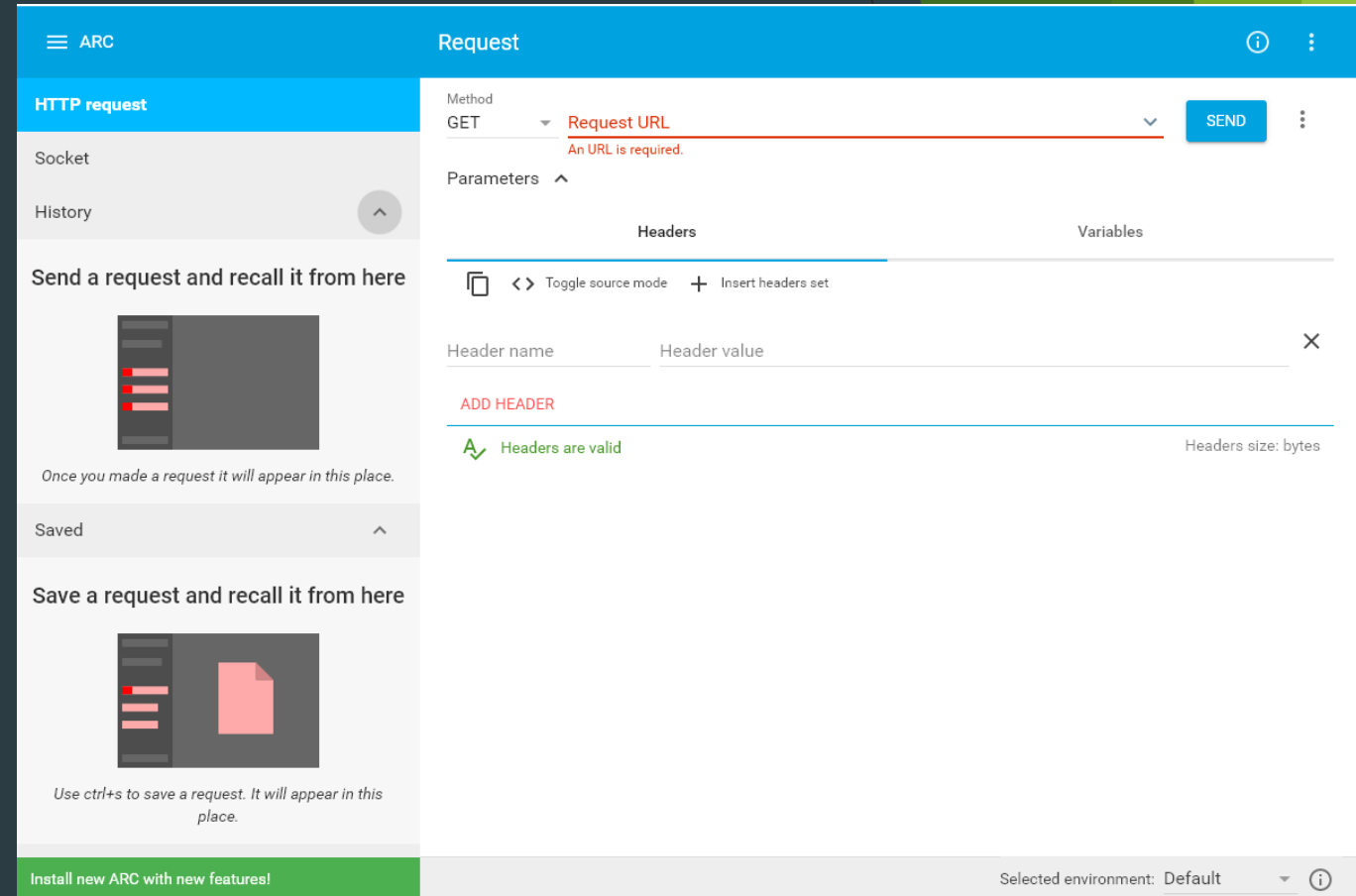
JSON(JavaScript Object Notation)

▶ JSON(JavaScript Object Notation) 구조

```
{
  "family": "hong",
  "data": [
    {
      "name": "홍길동",
      "tel": "010-1111-2222"
    },
    {
      "name": "홍길순",
      "tel": "011-2222-3333"
    }
  ]
}
```

Advanced REST client

- ▶ Advanced REST client
 - ▶ 구글 Chrome에서 제공하는 웹 플러그인
 - ▶ get, post, set, delete의 restful 방식의 요청을 client에서 가볍게 보내기 위한 어플리케이션
 - ▶ <https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddfdnphfgcellkdfbfbjeloo/related?hl=ko>



Express Rest Api 받기





▶ get 형식 전송 받기





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

app.get("/", function(req, res){
  res.send("ROOT");
});

app.get("/test1", function(req, res){
  res.send("TEST");
});

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

Method	Request URL
GET	http://localhost:2000
200 OK 25.78 ms	
   	
ROOT	

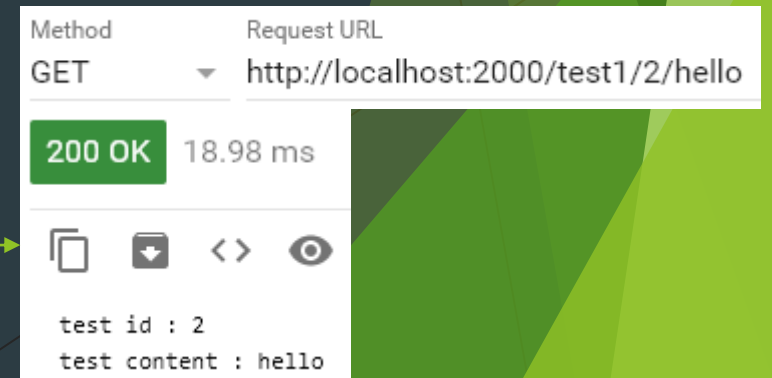
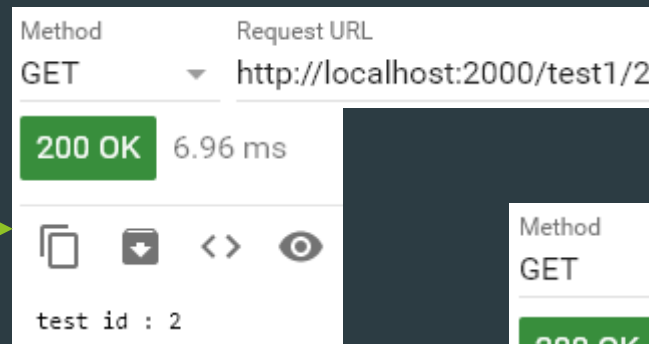
Method	Request URL
GET	http://localhost:2000/test1
200 OK 5.98 ms	
   	
TEST	

Express Rest Api 받기

▶ get 형식 전송 받기

- ▶ Rest 방식에서 url의 요소 중 어느 한 부분을 인자로 받고 싶을 경우 url의 특정 경로에 `[:paramname]`을 붙여 데이터 형태로 받아올 수 있다.
- ▶ 이 데이터를 받아 사용할 경우 `request` 객체에서 `req.params[:paramname]` 형태로 받아올 수 있다.

```
app.get("/test1/:id", function(req, res){  
  res.send("test id : "+req.params.id);  
});  
  
app.get("/test1/:id/:content", function(req, res){  
  res.send(" test id : "+req.params.id+  
    "\n test content : "+req.params.content);  
});
```







Express Rest Api 받기

- ▶ get 형식 전송 받기 - 파라미터와 같이 받아오기

```
app.get("/test1/:id/:content", function(req, res){  
  var color1 = (req.query.color1 == null)? "none" : req.query.color1;  
  var color2 = (req.query.color2 == null)? "none" : req.query.color2;  
  res.send(" test id : "+req.params.id+  
    "\n test content : "+req.params.content+  
    "\n test color1 : "+color1+  
    "\n test color2 : "+color2  
  );  
});
```

Method	Request URL
GET	http://localhost:2000/test1/2/hello?color1=red&color2=blue

200 OK	12.99 ms
--------	----------

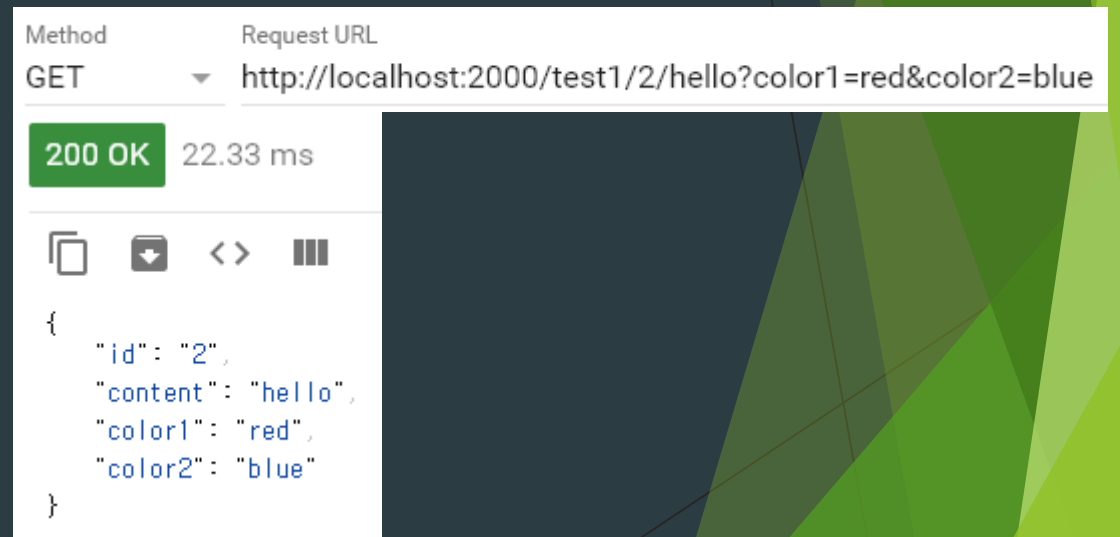


```
test id : 2  
test content : hello  
test color1 : red  
test color2 : blue
```

Express Rest Api 받기

- ▶ get 형식 전송 받기 - json 형태로 내보내기
 - ▶ json 형태로 내보내기 위해 response 객체에서는 json이라는 메서드를 제공한다.

```
app.get("/test1/:id/:content", function(req, res){
  var color1 = (req.query.color1 == null)? "none" : req.query.color1;
  var color2 = (req.query.color2 == null)? "none" : req.query.color2;
  // res.send(" test id : "+req.params.id+
  // "\n test content : "+req.params.content+
  // "\n test color1 : "+color1+
  // "\n test color2 : "+color2
  // );
  var data = {
    id : req.params.id,
    content : req.params.content,
    color1 : color1,
    color2 : color2
  }
  res.json(data);
});
```



Express Rest Api 받기





▶ post 형식 전송 받기





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

app.post("/", function(req, res){
  res.send("ROOT");
});

app.post("/test1", function(req, res){
  res.send("TEST");
});

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

Method	Request URL
POST	http://localhost:2000
200 OK	21.08 ms
   	
ROOT	

Method	Request URL
POST	http://localhost:2000/test1
200 OK	5.21 ms
   	
TEST	

Express Rest Api 받기

▶ post 형식 전송 받기

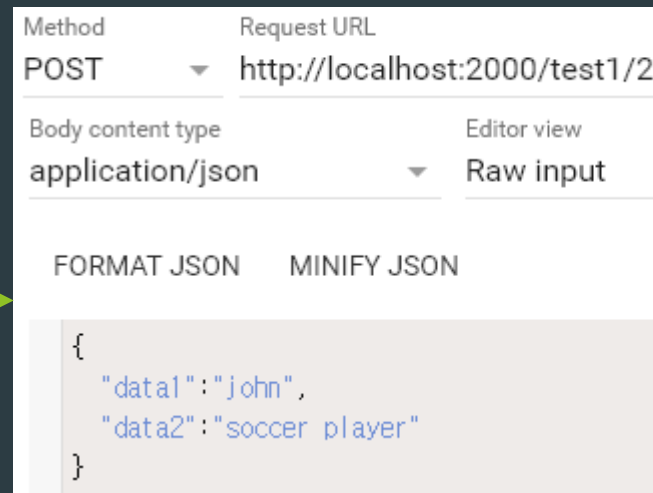
- ▶ post에서 json 타입의 데이터를 받기 위해서는 반드시 json 데이터의 처리를 express에서 해주어야 한다.

```
var express = require('express');
var app = express();
app.use(express.json())

app.post("/", function(req, res){ ...
});

app.post("/test1", function(req, res){ ...
});

app.post("/test1/:id", function(req, res){
  var data = {
    id : req.params.id,
    data1 : req.body.data1,
    data2 : req.body.data2
  };
  res.json(data);
});
```



Method POST Request URL http://localhost:2000/test1/2

Body content type application/json Editor view Raw input

FORMAT JSON MINIFY JSON

```
{
  "data1": "john",
  "data2": "soccer player"
}
```



200 OK 53.76 ms

```
{
  "id": "2",
  "data1": "john",
  "data2": "soccer player"
}
```

Express Rest Api 받기





▶ put 형식 전송 받기





```
var express = require('express');
var app = express();
app.use(express.json())

app.put("/", function(req, res){
  res.send("ROOT");
});

app.put("/test1", function(req, res){
  res.send("TEST");
});

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

Method	Request URL
PUT	http://localhost:2000
200 OK	56.31 ms
   	
ROOT	

Method	Request URL
PUT	http://localhost:2000/test1
200 OK	6.53 ms
   	
TEST	

Express Rest Api 받기

▶ put 형식 전송 받기 - json 타입 데이터 전송 받기

```
app.put("/test1/:id", function(req, res){  
  var data = {  
    id : req.params.id,  
    data1 : req.body.data1,  
    data2 : req.body.data2  
  };  
  res.json(data);  
});
```

Method	Request URL
PUT	▼ http://localhost:2000/test1/2

```
{  
  "data1": "john",  
  "data2": "soccer player"  
}
```

```
{  
  "id": "2",  
  "data1": "john",  
  "data2": "soccer player"  
}
```

Express Rest Api 받기

▶ delete 형식 전송 받기





```
var express = require('express');
var app = express();
app.use(express.json())





app.delete("/", function(req, res){
  res.send("ROOT");
});








app.delete("/test1", function(req, res){
  res.send("TEST");
});

app.delete("/test1/:id", function(req, res){
  var data = {
    id : req.params.id,
    data1 : req.body.data1,
    data2 : req.body.data2
  };
  res.json(data);
});

var server = app.listen(2000, function(){
  console.log('포트 2000번으로 서버 실행');
});
```

Method	Request URL
DELETE	http://localhost:2000
200 OK	72.59 ms
   	
ROOT	

Method	Request URL
DELETE	http://localhost:2000/test1
200 OK	7.21 ms
   	
TEST	

Method	Request URL
DELETE	http://localhost:2000/test1/2
  	{ "data1": "john", "data2": "soccer player" }
200 OK	9.76 ms
   	{ "id": "2", "data1": "john", "data2": "soccer player" }