SOPT 16th - Develop

06. Integrated Seminar - Guide [Server]

1. 개요

이번 6차 세미나의 목표는 'REST - Representational State Transfer'의 정의, 특징 및 가이드라인에 대해 알아본 후, 이에 관련된 간단한 조별 실습을 진행하면서 a) 어떻게 서버와 클라이언트가 소통하는지 경험하고 b) 주어진 상황에서 어떻게 API를 설계하고 코드로 적용할 수 있는지에 대해 숙지하는 것입니다.

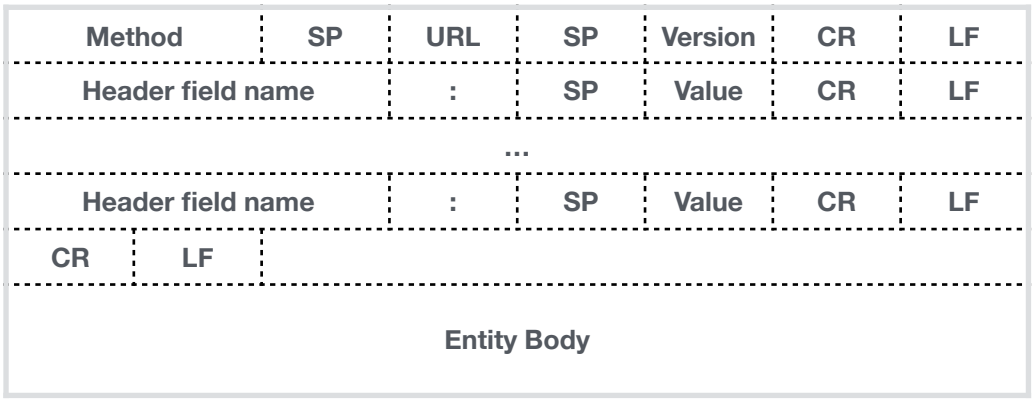
2. 기본 이론

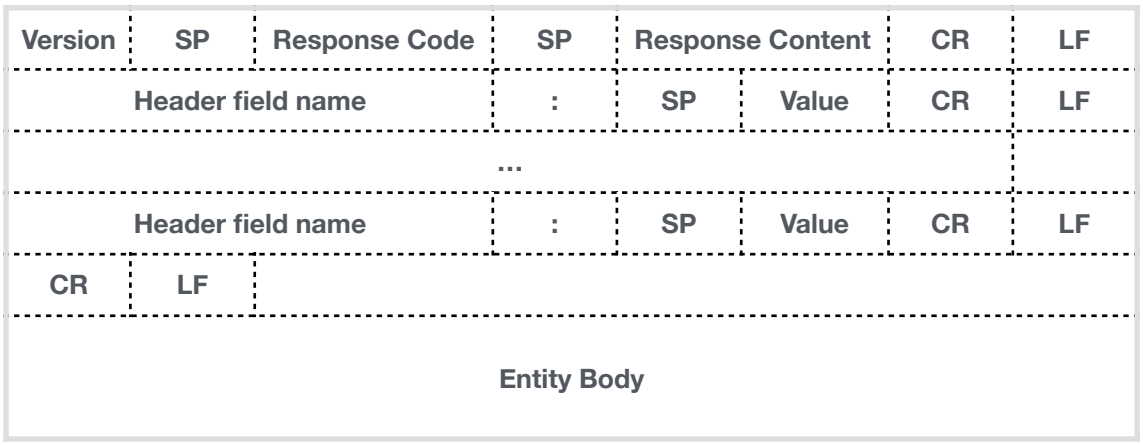
a) 서버 / 클라이언트

- 서버 : 특정 사용자로부터 받은 요청을 수행하여 처리한 후, 다시 응답할 수 있는 프로그램을 구동할 수 있는 컴퓨팅 시스템을 지칭합니다.

- 클라이언트 : 특정 기능을 수행할 목적으로 서버에 접속하여 필요한 요청을 발생시키는 프로그램이나 컴퓨팅 시스템을 지칭합니다.

b) HTTP : 일반적으로 TCP/IP 같이 신뢰성이 있는 프로토콜을 기반으로 동작하는 비연결성 프로토콜이며 서버와 클라이언트 사이에서 발생하는 요청과 응답으로 구성됩니다. 프로토콜은 일종의 규약이기 때문에 HTTP에서의 요청과 응답에 대한 규격이 정해져 있으며, HTTP는 약속들을 기반으로 이미지나 영상, 소리 등 다양한 형태의 자료를 주고 받습니다.





3. REST

a) 개요

Roy Fielding이 2000년대 초반에 발표한 소프트웨어 아키텍쳐의 형식 중 하나로, 웹의 장점을 최대한 활용할 수 있도록 설계되었습니다. REST는 Representational State Transfer의 약자로, 직역하면 '표현적인 상태를 전송한다'는 의미를 갖습니다.

최근에는 클라이언트가 컨텍스트 정보 (로그인 정보, 세션 등)를 직접 책임지고 관리하며, 서버는 비즈니스 로직 처리, 저장을 담당하는 방식으로 업무가 분리되고 있습니다. REST 구조에서 클라이언트는 자신이 관리하는 정보를 기반으로 요청 메시지를 생성하여 전송하는데, 이 때 메시지가 클라이언트가 어떤 정보를 요청하는지, 어떤 상태인지에 대해 충분히 기술되어 있으므로 이런 이름이 붙은 것 같습니다.

b) 제약 : 다음 제약 조건을 준수하는 한, 각 컴포넌트는 자유롭게 구성할 수 있습니다.

- **서버 / 클라이언트 구조 : 두 계층은 일관된 인터페이스로 분리되어야 합니다.**
- **계층화 : 클라이언트는 대상 서버, 중간 서버에 연결되는지 알 필요가 없습니다.**
- **무상태 : 요청은 클라이언트의 상태를 반영하며, 서버에 저장되지 않아야 합니다.**
- **캐시 가능 : HTTP 기반으로 작성된 REST 아키텍쳐 웹 캐시 등의 기능을 사용할 수 있습니다.**
- **인터페이스 일관성 : 아키텍쳐를 단순하게 만들고 구성 요소간 결합력이 약해야 합니다.**

c) REST 아키텍쳐 설계 원칙 가이드

- **자원 식별 : 웹 시스템에서의 URI로 특정 자원을 식별할 수 있어야 합니다.**
- **메세지로 자원 조작 : 자원 식별자와 메타 데이터만으로도 서버의 데이터를 제어할 수 있어야 합니다.**
- **자기 서술적 메시지 : 메세지 자체만으로도 어떤 내용을 요청하는 것인지 이해할 수 있어야 합니다.**
- **HATEOAS : 응답으로 오는 메시지에 특정 자원이 포함되면, 별도 처리없이 쉽게 접근 가능해야합니다.**

d) URI 네이밍 규칙

- **패스의 마지막에는 '/' 를 생략합니다.**
- **대상이 특정 자원일 경우에는 명사를, 동작을 의미하면 동사를 사용합니다.**
- **패스에는 가독성을 위해 underscore - '_' 대신 hyphen - '-' 을 사용합니다.**
- **단일 자원은 단수, 자원을 포함하는 상위 개념은 복수를 사용해야 합니다.**
- **패스의 '/' 로 계층 관계를 표현합니다.**
- **모든 URI에서 파일 확장자는 생략합니다. 이는 Accept 헤더로 대체할 수 있습니다.**
- **요청에 대한 액션은 패스에 포함하는 대신 HTTP Method로 표현합니다.**

4. 실습 - 서버

a) 개요

6차 세미나에서 서버 파트분들이 하게 될 일은 특정 상황을 가정해서 REST API 규약을 작성하고 이를 바탕으로 클라이언트와 소통하는 백엔드 서버를 개발해 아마존 웹 서비스에 호스팅하는 것입니다.

b) API 스펙

GET /thumbnails

> 게시글의 제목 리스트를 JSON Array 형태로 반환합니다
> 출력 결과

title : 게시글의 제목
timestamp : 작성 일시 (UTC 기준)

GET /contents/{content-id}

> 특정 게시물에 대한 정보를 JSON 형태로 반환합니다
> 출력 결과

id : 게시글의 고유 식별자
title : 게시글의 제목
content : 게시물의 본문
timestamp : 작성 일시 (UTC 기준)

POST /contents

> 신규 게시물을 서버에 전달합니다
> 입력해야 할 내용

title : 신규 게시물의 제목
content : 신규 게시물의 본문

> 출력 결과

id : 게시글의 고유 식별자
title : 게시글의 제목
timestamp : 작성 일시 (UTC 기준)
result : 처리 결과
reason : (처리가 성공적으로 이뤄지지 않았을 경우) 사유

c) 코드 작성

1) express-generator를 생성합니다

> npm install -g express-generator
(OS X는 상황에 따라 명령어 앞에 sudo를 붙여야 할 수도 있습니다)

2) express 프로젝트를 생성합니다

> express Server —ejs
> cd Server
> npm install
> npm install mysql

3) app.js를 다음과 같이 수정합니다

```
var express = require('express');
var path = require('path');
var logger = require('morgan');
var bodyParser = require('body-parser');

var contents = require('./routes/contents');
var thumbnails = require('./routes/thumbnails');

var app = express();

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));
app.use(express.static(path.join(__dirname, 'public')));

app.use('/contents', contents);
app.use('/thumbnails', thumbnails);

// catch 404 and forward to error handler
app.use(function(req, res, next) {

    var err = new Error('Not Found');
    err.status = 404;
    next(err);
});
```

```
// error handlers
if (app.get('env') === 'development') {

    app.use(function(err, req, res, next) {

      res.status(err.status || 500);
      res.render('error', { message: err.message, error: err });
  });
}

app.use(function(err, req, res, next) {

    res.status(err.status || 500);
    res.render('error', { message: err.message, error: {} });
});

module.exports = app;
```

4) routes 폴더에 thumbnails.js를 만들고 다음과 같이 수정합니다

```
var express = require('express');
var mysql = require('mysql');
var router = express.Router();
var connection = mysql.createConnection({

          'host' : '', 'user' : '', 'password' : '', 'database' : '',
});

router.get('/', function(req, res, next) {

    connection.query('select id, title, timestamp from board ' +
                          'order by timestamp desc;', function (error, cursor) {

           res.json(cursor);
    });
});

module.exports = router;
```

5) routes 폴더에 contents.js를 만들고 다음과 같이 수정합니다

```
var express = require('express');
var mysql = require('mysql');
var router = express.Router();
var connection = mysql.createConnection({

   'host' : '', 'user' : '', 'password' : '', 'database' : '',
});

router.get('/:content_id', function(req, res, next) {

   connection.query('select * from board where id=?;',
                           [req.params.content_id], function (error, cursor) {

      if (cursor.length > 0)
         res.json(cursor[0]);
      else
         res.status(503).json({  result : false, reason : "Cannot find selected article" });
   });
});
router.post('/', function(req, res, next) {

   connection.query('insert into board(title, content) values (?, ?);',
                           [req.body.title, req.body.content], function (error, info) {

      if (error == null) {

         connection.query('select * from board where id=?;',
                              [info.insertId], function (error, cursor) {

            if (cursor.length > 0) {

               res.json({

                  result : true, id : cursor[0].id, title : cursor[0].title,
                  timestamp : cursor[0].timestamp,
               });
            }
            else
               res.status(503).json({ result : false, reason : "Cannot post article" });
         });
      }
      else
         res.status(503).json(error);
   });
});

module.exports = router;
```

7) AWS EC2, RDS 인스턴스를 생성함과 동시에 Github 저장소에 해당 프로젝트를 퍼블리싱합니다

8) XShell이나 터미널을 사용하여 이전 단계에 생성한 EC2 인스턴스로 접속하고 git 명령어로
프로젝트를 해당 저장소로부터 복사합니다

9) MySQL Workbench나 터미널을 사용하여 RDS 인스턴스로 접속하고 테이블을 하나 생성해줍니다

```
> CREATE TABLE board (

      id int(11) NOT NULL AUTO_INCREMENT,
      title varchar(200) COLLATE utf8_unicode_ci DEFAULT NULL,
      content longtext COLLATE utf8_unicode_ci,
      timestamp timestamp NULL DEFAULT CURRENT_TIMESTAMP,
      PRIMARY KEY (id)
)

ENGINE=InnoDB AUTO_INCREMENT=54 DEFAULT CHARSET=utf8
COLLATE=utf8_unicode_ci;
```

10) 프로젝트를 실행하기에 앞서 thumbnails.js, contents.js를 수정해 RDS 인스턴스와 연결합니다

```
var connection = mysql.createConnection({

        'host' : 'rds_name.xxxxxxxx.location.rds.amazonaws.com',
        'user' : 'user_id',
        'password' : 'user_password',
        'database' : 'default_schema',
});
```

10) 프로젝트를 실행합니다

```
> DEBUG=Server ./bin/www
```

d) 코드에 대한 설명

1) 라우팅 처리

```
var contents = require('./routes/contents');
var thumbnails = require('./routes/thumbnails');

app.use('/contents', contents);
app.use('/thumbnails', thumbnails);
```

위 코드는 각 패스로 시작하는 요청을 지정한 모듈이 처리하게끔 선언하는 것입니다.
각 요청을 의미 단위로 묶어 파일 단위로 처리할 수 있어 분업이 쉽게 가능해집니다.