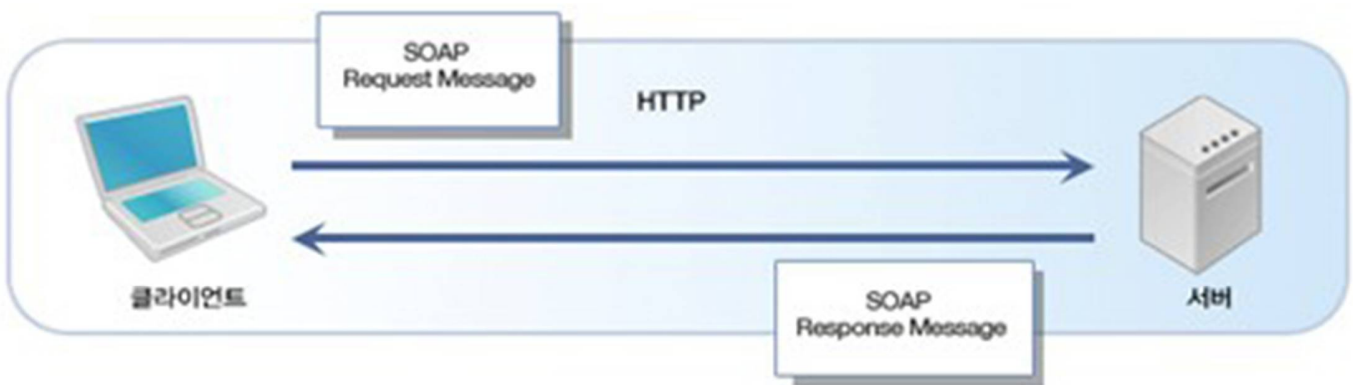
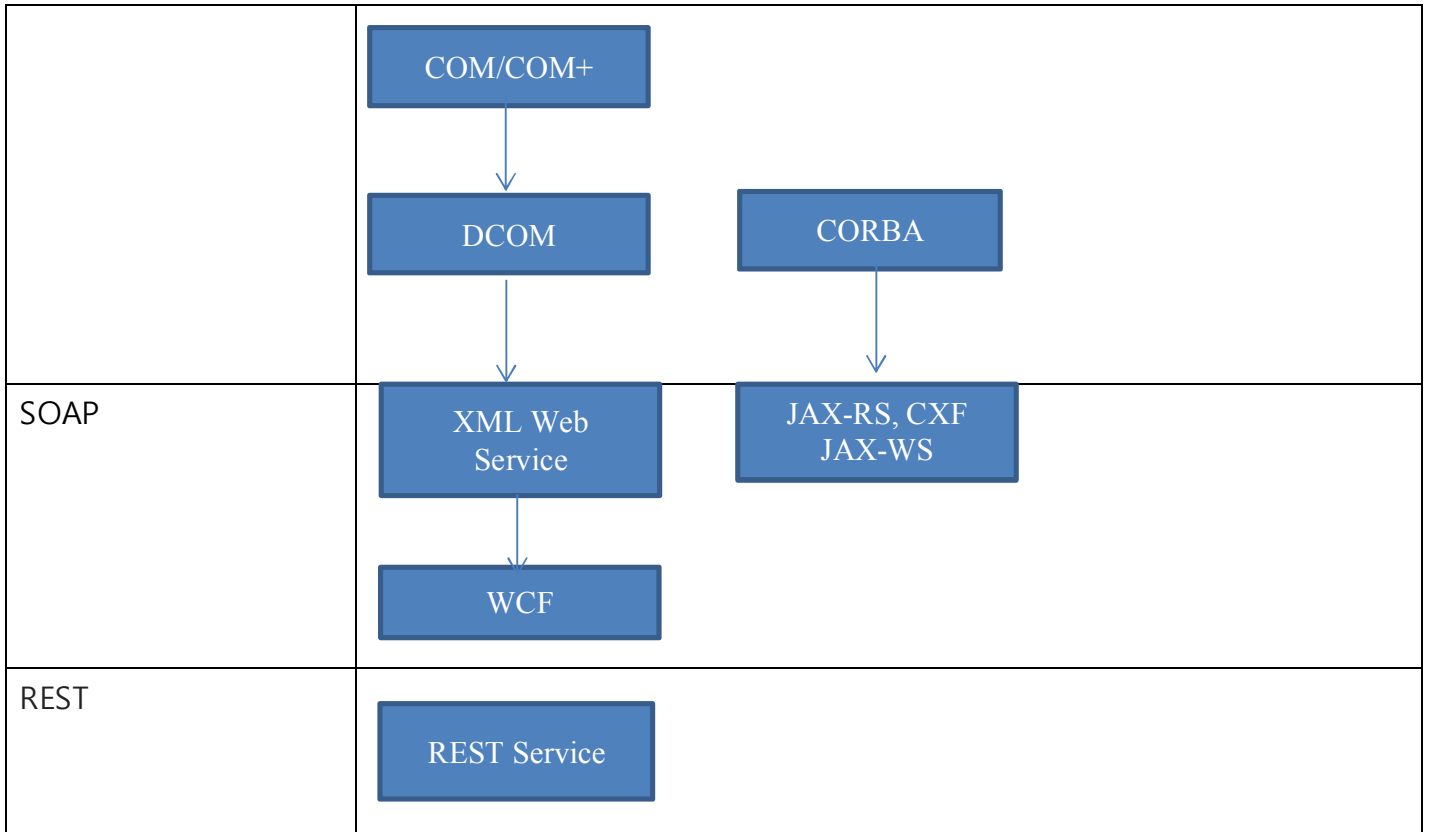


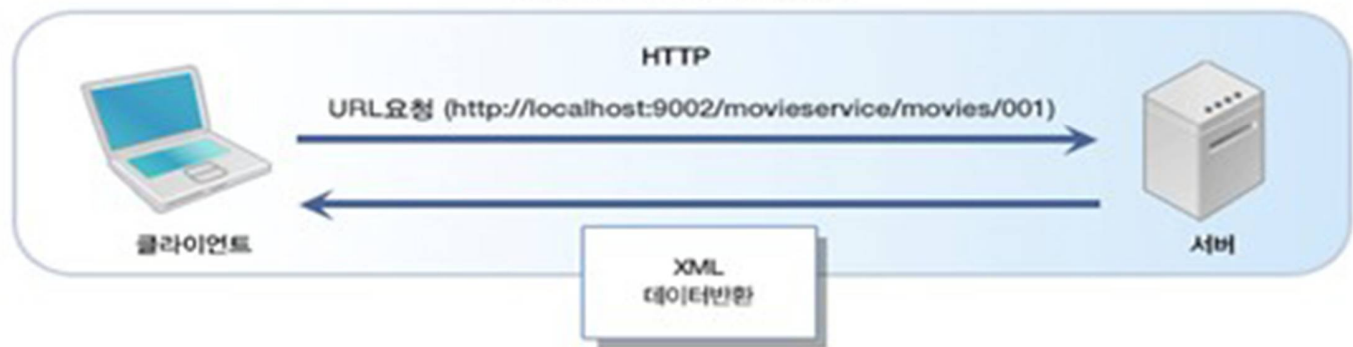
목차

1. REST(REpresentational State Transfer)	3
1.1 REST 장단점.....	4
1.2 REST 아키텍처	4
1.3 REST 메시지의 구조	5
2. Json	6
3. 서버에서 클라이언트로 json 데이터를 보내는 방법 : @ResponseBody	7
3.1 curtime 구현하기	8
3.2 selectuserlist 구현하기	9
3.3 getBoardOne 구현하기	10
4. @RequestBoyd 실습	12
5. Reference.....	13

st09.REST 서비스만들기



[SOAP Web Services]



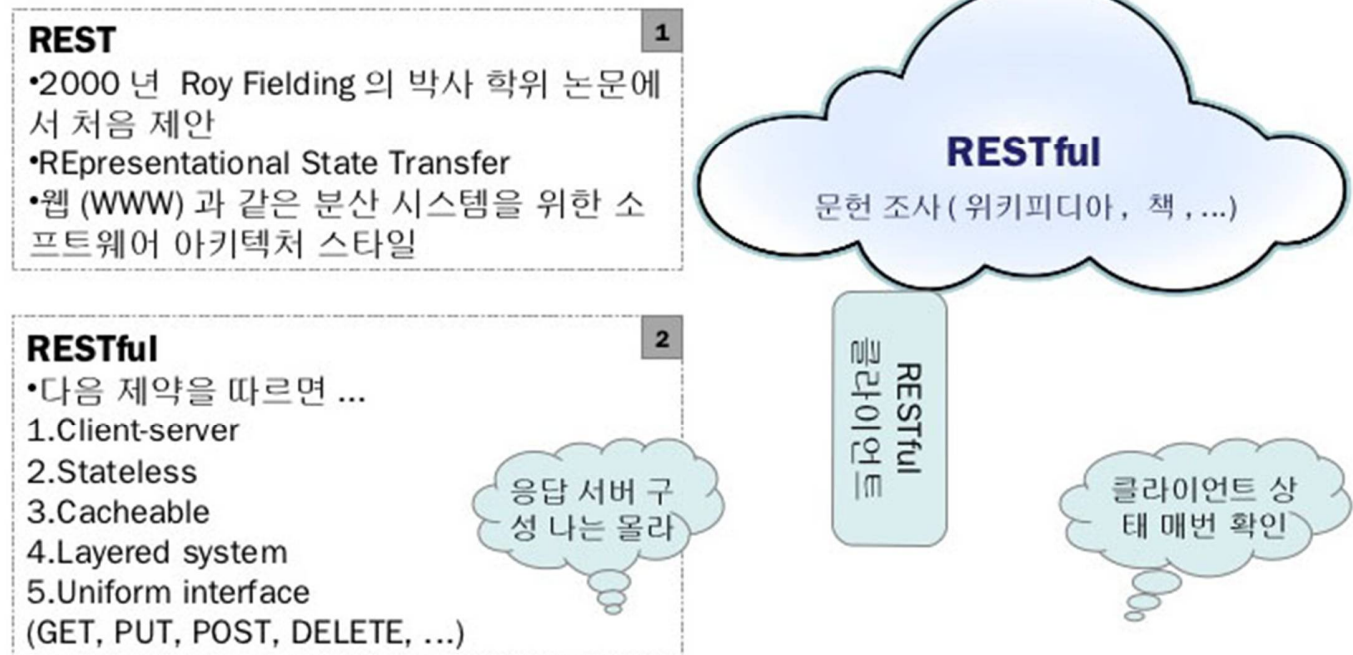
[RESTful Web Services]

1. REST(REpresentational State Transfer)

REST 서비스는 HTTP 를 통해 데이터를 전송하기 위한 웹 메서드다.

- URI 기반으로 리소스에 접근하는 기술
 - 예) Username이 1인 사용자의 정보를 보내줘
 - Request : http://www.mydomain.com/user/1
 - Response : XML or JSON or String or ...
- 프로토콜은 어느 장비에서나 지원하는 HTTP를 사용
- HTTP 프로토콜의 간단함을 그대로 시스템간 통신시 사용
- HTTP 프로토콜 그 자체에 집중

RESTful 이란 ?



REST란 위에 정의된 것 처럼 HTTP를 통해 세션 트래킹 같은 부가적인 전송 레이어 없이, 전송하기 위한 아주 간단한 인터페이스 입니다. 또한 HTTP 등의 기본 개념에 충실히 따르는 웹 서비스 입니다.

1.1 REST 장단점

REST의 장단점은 아래와 같습니다.

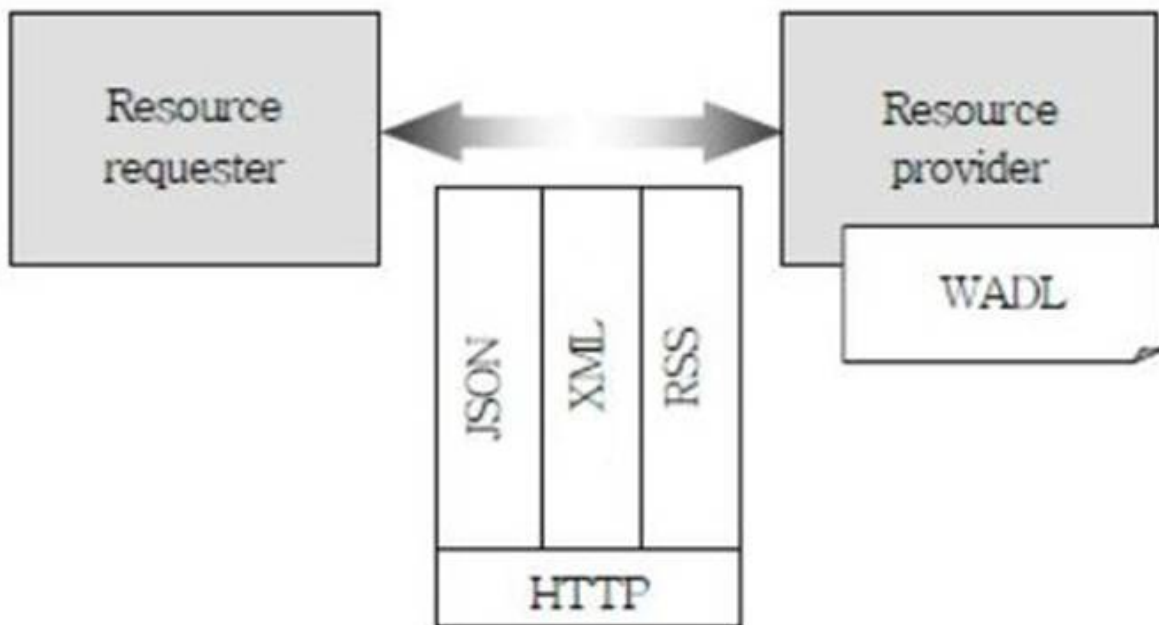
장점

- * 플랫폼과 프로그래밍 언어에 독립적이다. (= SOAP)
- * SOAP보다 개발하기 단순하고 도구가 거의 필요없다.
- * 간결하므로 추가적인 메시지 계층이 없다.

단점

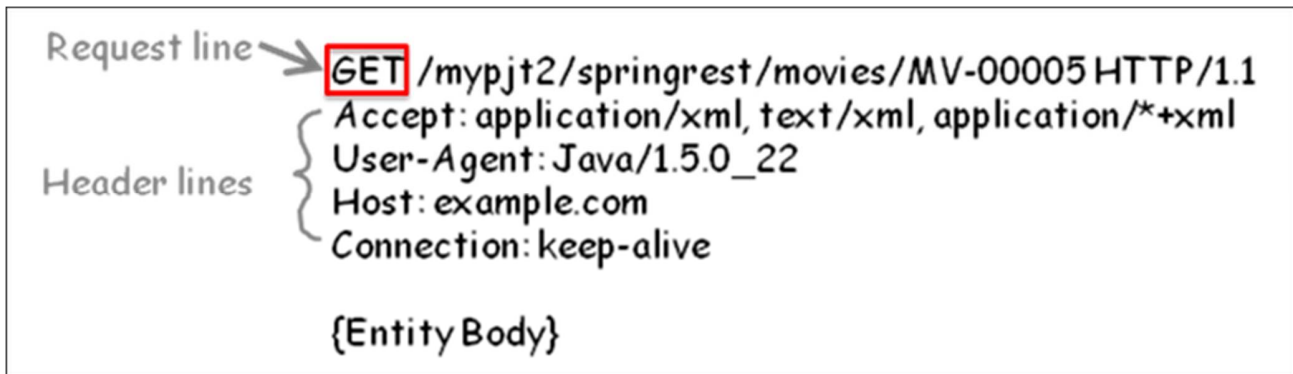
- * Point-to-point 통신 모델을 가정하므로 둘 이상으로 상호작용하는 분산환경에는 유용하지 않다.
- * 보안, 정책 등에 대한 표준이 없다.
- * HTTP 통신 모델만 지원한다.

1.2 REST 아키텍처



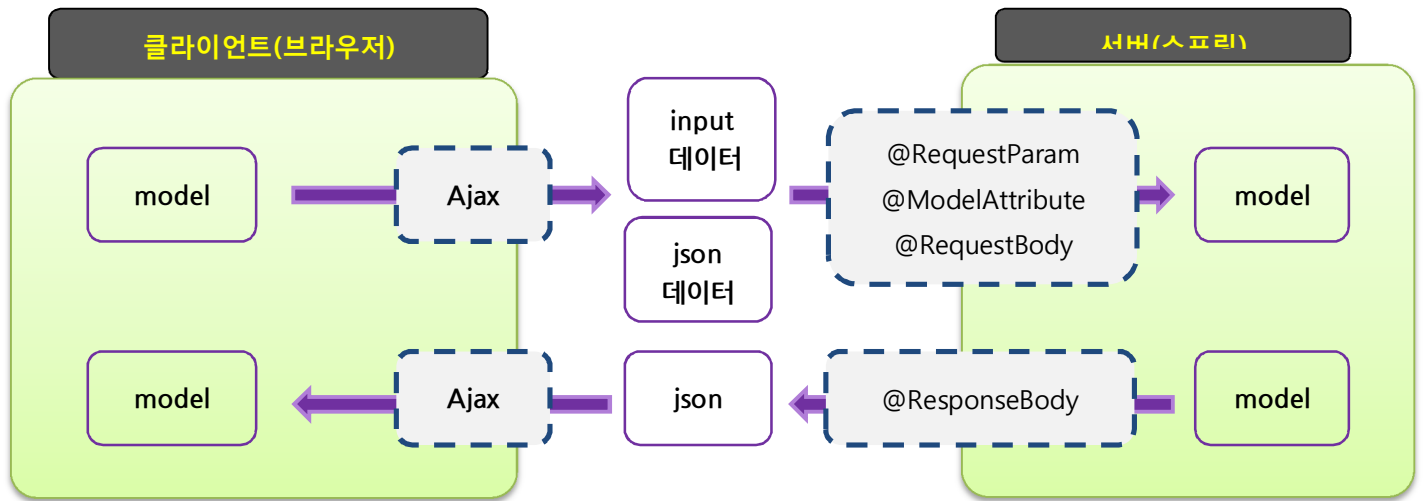
REST의 아키텍처는 자원 요청자(Resource requester), 자원 제공자(Resource provider)로 구성됩니다. REST는 자원을 등록하고 저장해주는 중간 매체 없이 자원 제공자가 직접 자원 요청자에게 제공합니다. REST는 기본 HTTP 프로토콜의 메소드인 GET/PUT/POST/DELETE를 이용하여 자원 요청자는 자원을 요청합니다. 자원 제공자는 다양한 형태로 표현된 (JSON, XML, RSS 등)의 리소스를 반환합니다.

1.3 REST 메시지의 구조



2. Json

Spring MVC 에서 data 를 Json 형식으로 보내고 받는 방법에 대해 알아보겠습니다.



어노테이션	설명
@RequestParam	클라이언트의 input 태그에서 데이터를 받아 하나씩 처리할 때
@ModelAttribute	클라이언트의 input 태그에서 데이터를 받아 모델 바꾸어 처리 할 때
@RequestBody	클라이언트로부터 json 데이터를 받아 모델로 처리할 때 POST 방식에서만 사용 가능
@ResponseBody	클라이언트로 json 데이터를 보낼 때

3. 서버에서 클라이언트로 json 데이터를 보내는 방법 : @ResponseBody

메소드 앞에 @ResponseBody 를 붙여서 사용하게되면 해당 객체가 자동으로 Json 객체로 변환되어 반환됩니다. @ResponseBody 환경을 설정하는 방법을 알아봅니다.

- build.gradle 에 추가

```
// json library :: @ResponseBody 를 이용해 json 데이터를 반환하기 위한 라이브러리
compile 'com.fasterxml.jackson.core:jackson-core:2.4.1'
compile 'com.fasterxml.jackson.core:jackson-databind:2.4.1.1'
```

- servlet-context.xml 에 어노테이션 설정해줍니다.

```
<!-- annotation 설정 -->
<!-- @Controller, @RequestMapping 등과 같은 어노테이션을 사용하는 경우 설정 -->
<annotation-driven />
```

3.1 curtime 구현하기

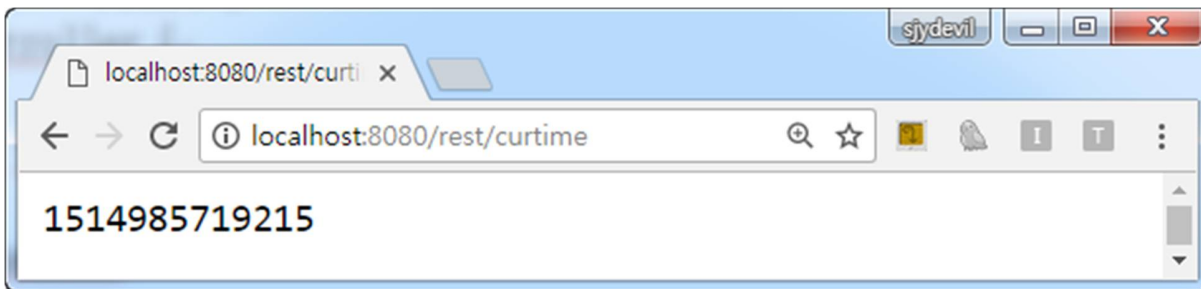
- <http://localhost:8080/rest/curtime>

```
@Controller
public class RestController {

    @RequestMapping(value = "/rest/curtime", method = {RequestMethod.GET} )
    @ResponseBody
    public long curtime(Model model) {
        logger.info("/rest/curtime");

        return new Date().getTime();
    }
}
```

- 실행 화면



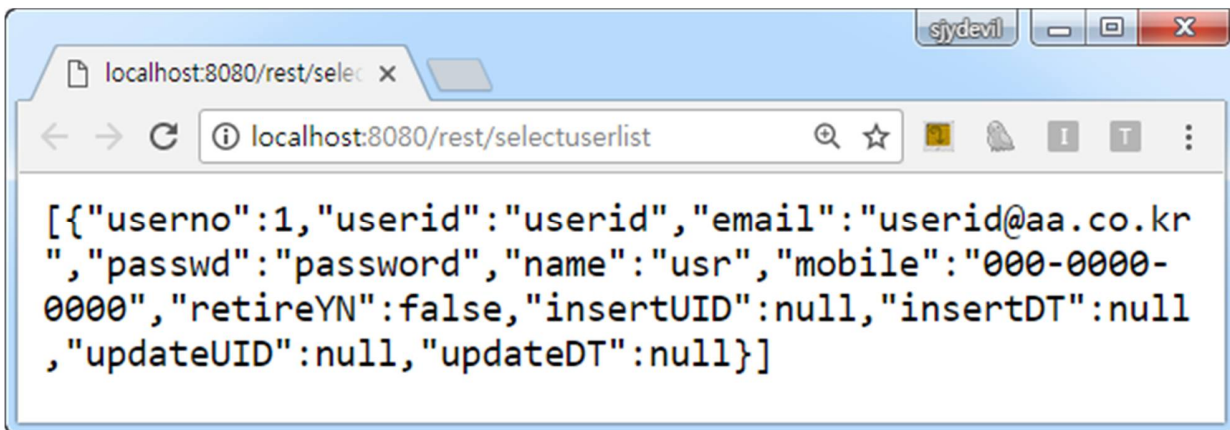
3.2 selectuserlist 구현하기

- <http://localhost:8080/rest/selectuserlist>

```
@Controller
public class RestController {

    // ServiceUser 인스턴스 만들기.
    @Autowired
    IServiceUser usersvr;

    // http://localhost:8080/rest/selectuserlist
    @RequestMapping(value = "/rest/selectuserlist", method = {RequestMethod.GET} )
    @ResponseBody
    public List<ModelUser> home( Model model) {
        return usersvr.selectUserList(null);
    }
}
```



3.3 getBoardOne 구현하기

- <http://localhost:8080/rest/getboardone>

```

@RequestMapping(value= "/rest/getboardone", method={RequestMethod.GET})
@ResponseBody
public ModelBoard getBoardOne ( @RequestParam("cd") String boardcd) {
    ModelBoard board = boardsvr.getBoardOne(boardcd);
    return board;
}

```

↑ 같은 타입



- Script 테스트 코드

```

$("#joinOk").click( function(e){

$.ajax({
    url : '/rest/getboardone',
    data: { 'cd' : 'free' }, // { 'data1':'test1', 'data2':'test2' }
    type: 'get',           // get, post
    timeout: 30000,        // 30 초
    dataType: 'json',      // text, html, xml, json, jsonp, script
    beforeSend : function() {
        // 통신이 시작되기 전에 이 함수를 타게 된다.
        $('#message1').html('');
    }
}).done( function(data, textStatus, xhr ){
    // 통신이 성공적으로 이루어졌을 때 이 함수를 타게 된다.
    if(!data){
        alert("존재하지 않는 boardcd 입니다");
    }
    else {
        alert("존재하는 boardcd 입니다");
    }
}
}

```

```

    }
  }).fail( function(xhr, textStatus, error ) {
    // 통신이 실패했을 때 이 함수를 타게 된다.
    var msg = '';
    msg += "code:"      + xhr.status          + "\n";
    msg += "message:"   + xhr.responseText    + "\n";
    msg += "status:"    + textStatus          + "\n";
    msg += "error  : " + error                  + "\n";

    console.log(msg);
  }).always( function(data, textStatus, xhr ) {
    // 통신이 실패했어도 성공했어도 이 함수를 타게 된다.
    $('#message1').html('');
  });
});

```

위 코드는 @ResponseBody 방식에 대한 간단한 예제 코드입니다. Script 코드에서 서버(Controller)에 input 태그에 입력된 id 값을 전송하면 Controller에서는 해당 데이터를 parameter 로 받고 그 id 값으로 DB 를 조회합니다. 앞서 말하였듯 return 형 앞에 @ResponseBody 를 사용하고 해당 객체를 return 해주기만 하면 ajax success 함수의 data 에 person 객체가 Json 객체로 변환 후 전송되어 파싱이 필요 없습니다.

4. @RequestBody 실습

POST 방식만 가능

```
{"id":"test3id","pw":"test3pw","name":"test3name","email":"test3email"}
```

5. Reference

<http://www.nexttree.co.kr/p11205/>

<http://wonzopein.com/50>

<http://hmkcode.com/spring-mvc-json-json-to-java/> - @ResponseBody