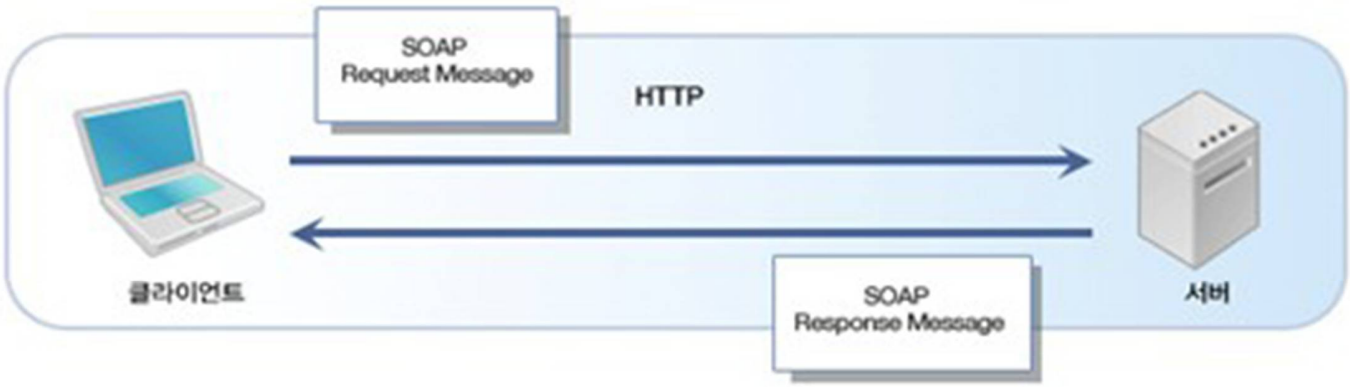
목차

	COM/COM+ → DCOM → XML Web Service → WCF CORBA → JAX-RS, CXF JAX-WS
SOAP	
REST	REST Service



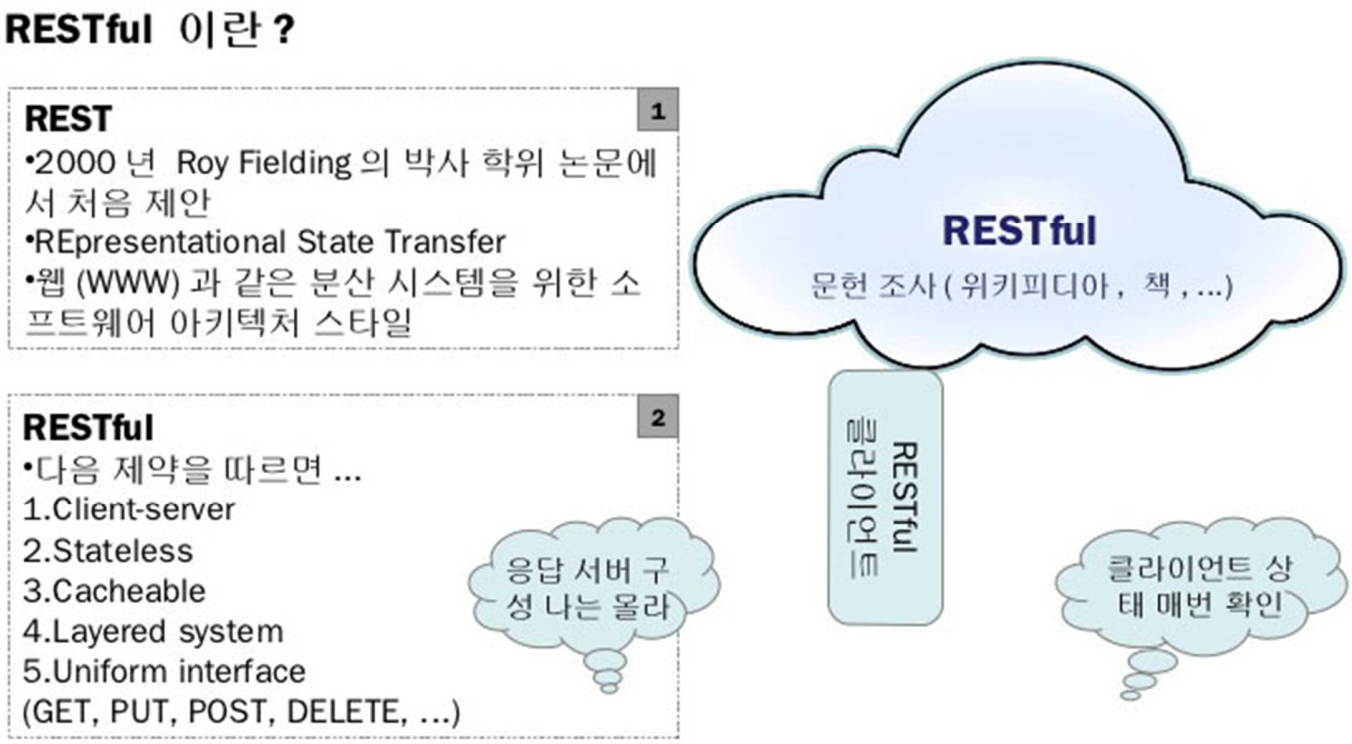
[SOAP Web Services]

[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



REST 란 위에 정의된 것 처럼 HTTP 를 통해 세션 트랙킹 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

REST 의 장단점은 아래와 같습니다.

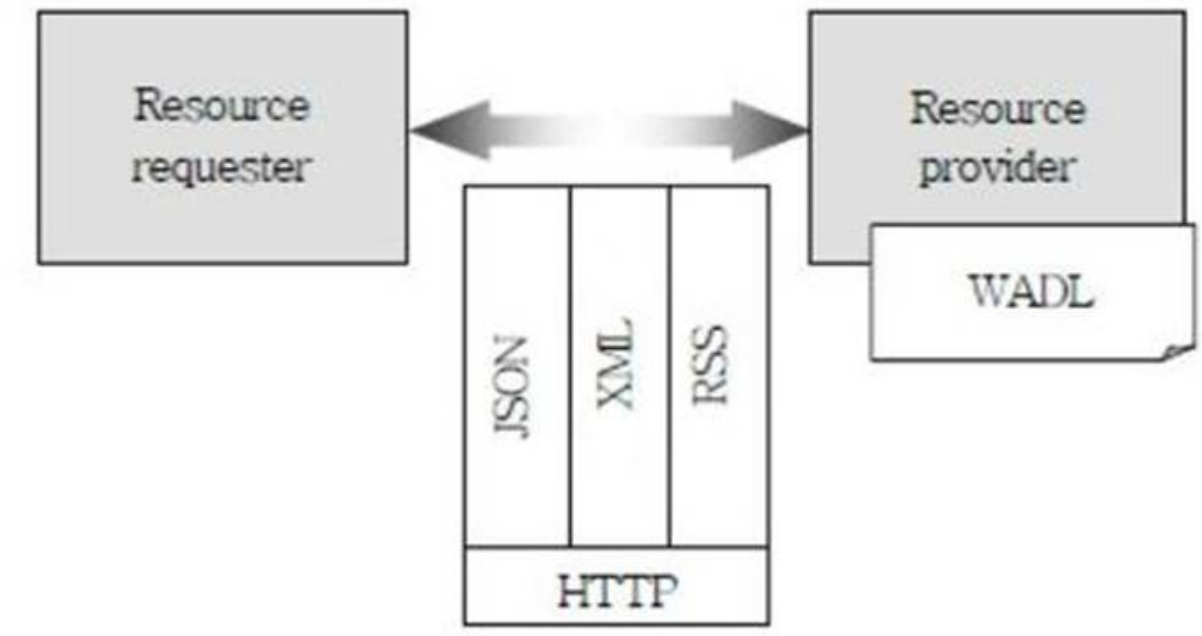
장점

* 플랫폼과 프로그래밍 언어에 독립적이다. (= SOAP)
* SOAP 보다 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



REST 의 아키텍처는 자원 요청자(Resource requester), 자원 제공자(Resource provider) 로 구성됩니다. REST 는 자원을 등록하고 저장해주는 중간 매체 없이 자원 제공자가 직접 자원 요청자에게 제공합니다. REST 는 기본 HTTP 프로토콜의 메소드인 GET/PUT/POST/DELETE 를 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 보내는 2 가지 방법에 대해 알아보겠습니다.

- Jackson 을 사용하는 방법
- JsonView 을 사용하는 방법

(ex) Jackson 사용시 Ajax 결과 값 예시

```
var data = [{ "key1":"value1", "key2":"value2" }]
alert( data[0].key1 )
```

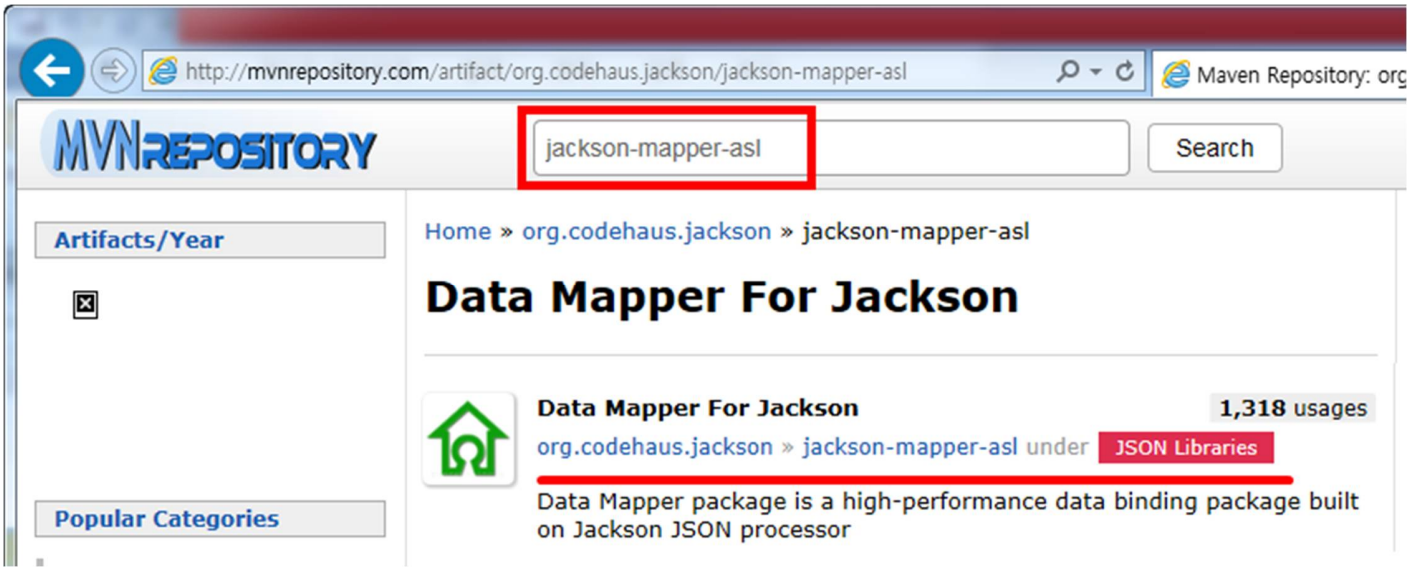
(ex) JsonView 사용시 Ajax 결과 값 예시

```
var data = { result : [{ "key1":"value1", "key2":"value2" }]  }
alert( data.result[0].key1 );
```

2.1 Jackson 을 이용하는 방법(@ResponseBody)

두번째로 살펴볼 방법은 ResponseBody 어노테이션을 사용하는 방법입니다. 메소드의 return 형 앞에 @ResponseBody 를 붙여서 사용하게되면 해당객체가 자동으로 Json 객체로 변환되어 반환됩니다. 구현에 앞서 @ResponseBody 환경을 설정하는 방법을 알아봅니다.

첫째, jackson-mapper-asl 라이브러리를 프로젝트에 추가시켜 주어야 됩니다.



build.gradle

```
243
244     // json library :: @ResponseBody를 이용해 json 데이터를 반환하기 위한 라이브러리
245     compile "org.codehaus.jackson:jackson-mapper-asl:1.9.13"
```

둘째, servlet xml 에 아래와 같이 어노테이션을 설정해줍니다.

<annotation-driven />

@ResponseBody 방식의 Java Controller 구현 코드입니다.

```
//Controller
@RequestMapping(value= "/jackson", method=RequestMethod.GET)
public @ResponseBody ModelBoard  jackson( @RequestParam("id") String id)  {

    ModelBoard board = boardsrv.getBoardOne(boardcd);

    return board;
}
```

@ResponseBody 방식의 Script 테스트 코드

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

    $.ajax({
        url : 'http://localhost/rest/jsonview',
        data: { id:'free' }, // 사용하는 경우에는 { 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('<img src="./loading.gif">');
        }
    }).done( function(data, textStatus, xhr ){
        // 통신이 성공적으로 이루어졌을 때 이 함수를 타게 된다.
        $("#ajax").remove();
        var data = JSON.parse( data );
        if(!data){
            alert("존재하지 않는 ID 입니다");
```

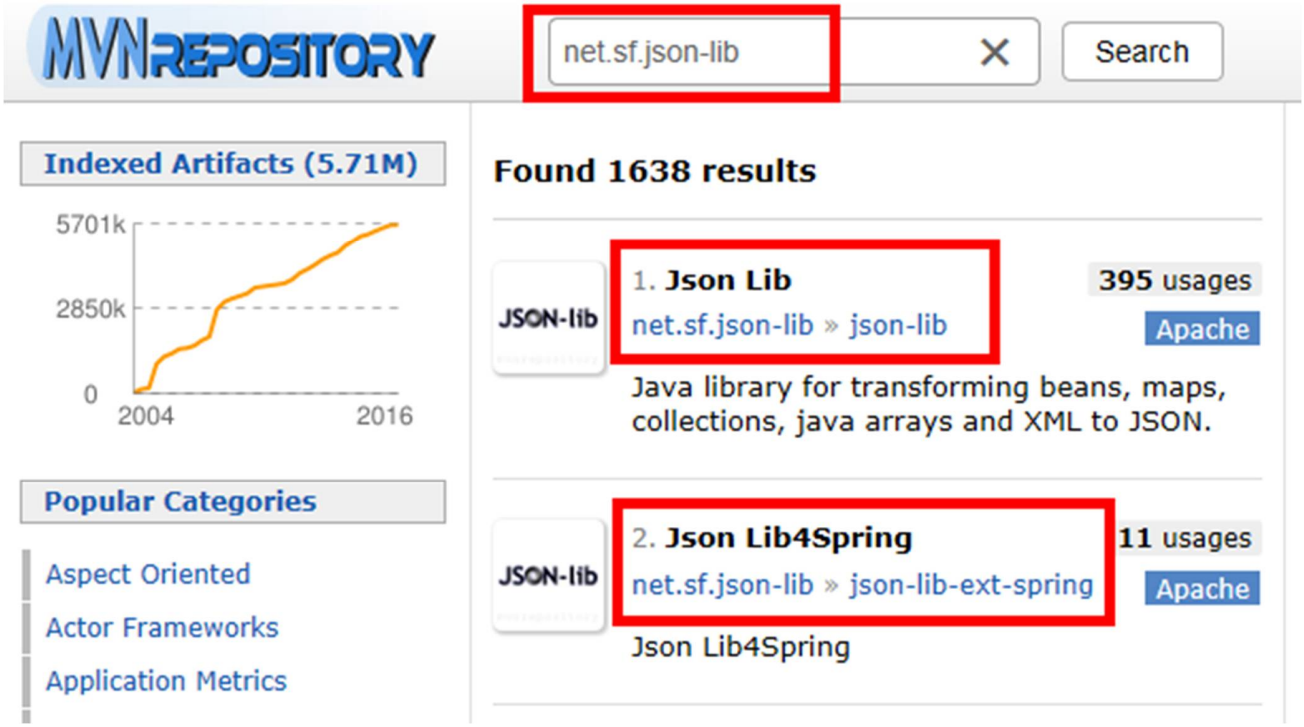
```
            return false;
        }

        var html = '';
        html += '<form class="form-signin" action="" id="ajax">';
        html += '이름<input type="text" name="name" value="'+data.name+'">';
        html += '아이디<input type="text" name=id" value="'+data.id+'">';
        html += '이메일<input type="text" name="email" value="'+data.email+'">';
        html += '</form>';

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

});
```

위 코드는 @ResponseBody 방식에 대한 간단한 예제 코드입니다. Script 코드에서 서버(Controller)에 input 태그에 입력된 id 값을 전송하면 Controller 에서는 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 필요없습니다..

2.2 JsonView 을 이용하는 방법

jsonView 방식을 사용하기 위한 환경 설정.

첫째, json-lib-ext-spring 라이브러리를 프로젝트에 추가시켜 주어야 됩니다.



build.gradle

```
248
249     // https://mvnrepository.com/artifact/net.sf.json-lib/json-lib-ext-spring
250     compile group: 'net.sf.json-lib', name: 'json-lib-ext-spring', version: '1.0.2'
251
```

둘째, servlet-context.xml 에 아래와 같이 bean 객체를 설정 후 viewResolver 를 설정 해줍니다.

```
<!-- jsonView 설정 -->
<beans:bean id="jsonView" class="net.sf.json.spring.web.servlet.view.JsonView"/>
<beans:bean id="viewResolver"
class="org.springframework.web.servlet.view.BeanNameViewResolver">
    <beans:property name="order" value="1" />
</beans:bean>
```

jsonView 방식의 Java Controller 구현 코드입니다

```
// http://localhost/restservice/jsonview
@RequestMapping(value = "/jsonview", method = RequestMethod.GET)
public ModelAndView AjaxView(@RequestParam("boardcd") String boardcd) {
    ModelAndView mav = new ModelAndView();

    ModelBoard board = boardsrv.getBoardOne(boardcd);
    mav.addObject("results", board);
```

```
    mav.setViewName("jsonView");

    return mav;
}
```

// 테스트 Script 코드

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

    $.ajax({
        url : 'http://localhost/rest/jsonview',
        data: { id:'free' }, // 사용하는 경우에는 { 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('<img src="./loading.gif">');
        }
    }).done( function(data, textStatus, xhr ){
        // 통신이 성공적으로 이루어졌을 때 이 함수를 타게 된다.
        $("#ajax").remove();
        var data = JSON.parse( data );
        if(!data){
            alert("존재하지 않는 ID 입니다");
            return false;
        }

        var html = '';
        html += '<form class="form-signin" action="" id="ajax">';
        html += '이름<input type="text" name="name" value="'+data.name+'">';
        html += '아이디<input type="text" name=id" value="'+data.id+'">';
        html += '이메일<input type="text" name="email" value="'+data.email+'">';
        html += '비밀번호<input type="text" name="password"
value="'+data.password+'">';
        html += '</form>';

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

});
```

위 코드는 jsonView 방식에 대한 간단한 예제 코드입니다. Script 코드에서 서버(Controller)에 input 태그에 입력된 id 값을 전송하면 Controller 에서는 해당 데이터를 parameter 로 받고 그 id 값으로 DB 를 조회합니다. 조회결과가 있으면 해당 회원(Person) 객체를 ModelandView 객체에 addObject 메소드를 사용하여 넣어주고, 조회결과가 없으면 null 이 됩니다. 그리고 마지막으로 view 명을 "jsonView"로 하면 ajax success 함수 responseData 에 ModelandView 객체가 Json 객체로 파싱되어서 넘어옵니다. 즉, ModelandView 객체에 맵 형식(키,밸류)으로 Person 객체를 입력하였으므로 responseData 가 아닌 responseData.person 이 Person 객체가 됩니다. 넘어온 Person 객체는 json 형식이기 때문에 속성 값은 data.id, data.name 등으로 구현이 됩니다.

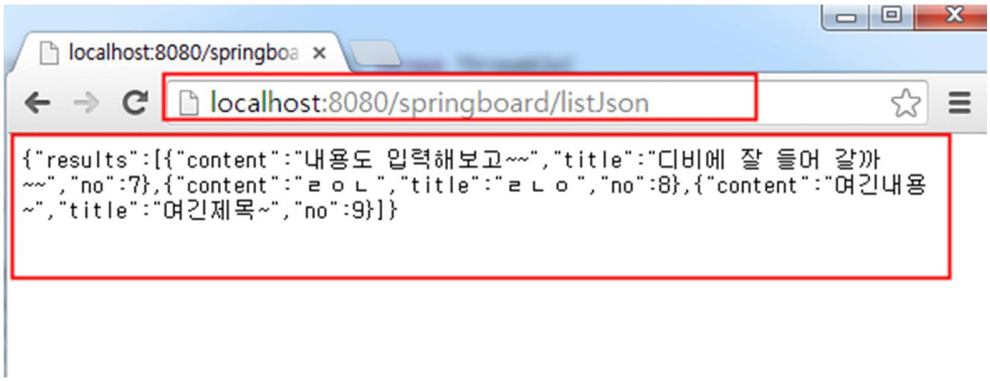
먼저 servlet_context.xml 에 다음과 같이 설정 되있는지 확인한다.

servlet_context.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:beans="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-bean
        http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring
        http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.

    <context:component-scan base-package="spring.board" use-default-filters="false">
        <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>
    </context:component-scan>

    <!-- Enables the Spring MVC @Controller programming model -->
    <mvc:annotation-driven>
        <mvc:message-converters register-defaults="true">
            <bean class = "org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">
                <property name="supportedMediaTypes" value = "text/plain;charset=UTF-8" />
            </bean>
        </mvc:message-converters>
    </mvc:annotation-driven>
```

```
@Controller
public class BoardController {

    @Autowired
    private MainService mainService;

    @RequestMapping("/listJson")
    public  @ResponseBody Map<?,?> listJson(@RequestParam Map<String, Object> paramMap, ModelMap model) throws Throw
        model.put("results", mainService.getList(paramMap));
        return model;
    }
```

자 그럼 json 으로 나오는지 출력해보자



3. SOAP(Simple Object Access Protocol)

SOAP 이란 위에 정의된 것 처럼 일반적으로 널리 알려진 HTTP, HTTPS, SMTP 등을 통해 XML 기반의 메시지를 컴퓨터 네트워크 상에서 교환하는 프로토콜입니다.

3.1 SOAP 장단점

SOAP 의 장단점은 아래와 같습니다.

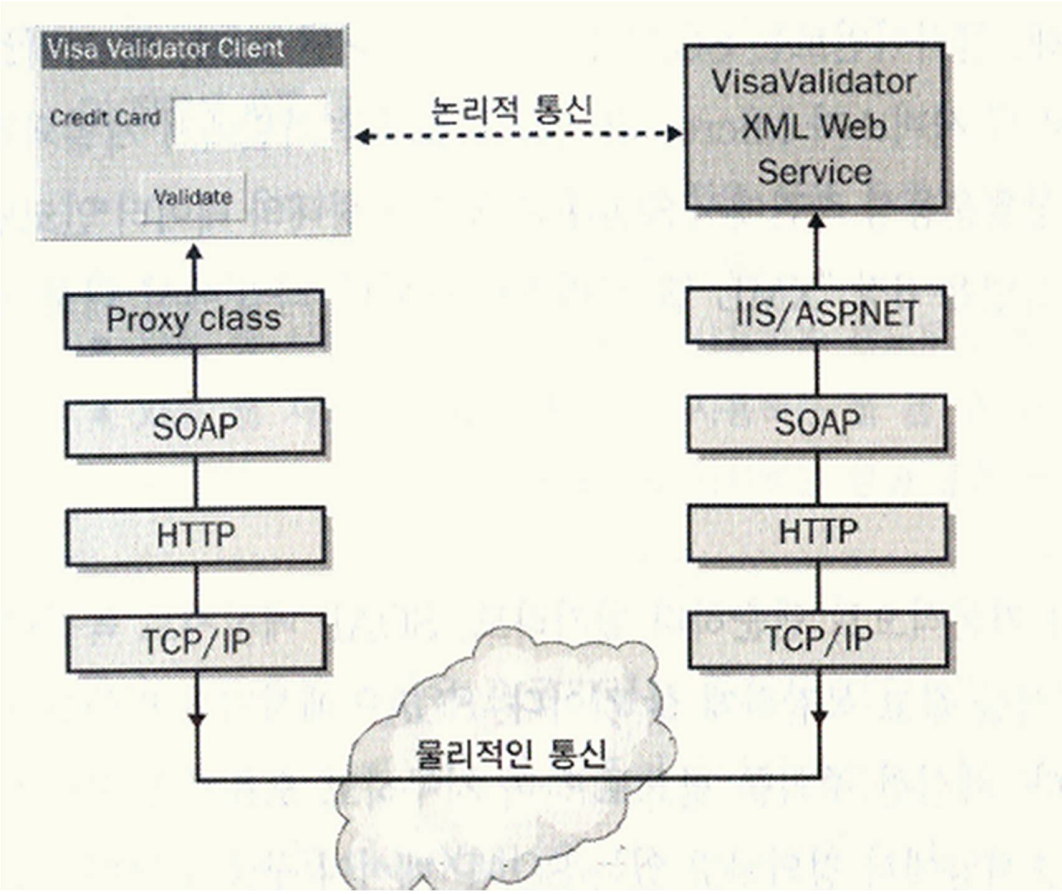
장점

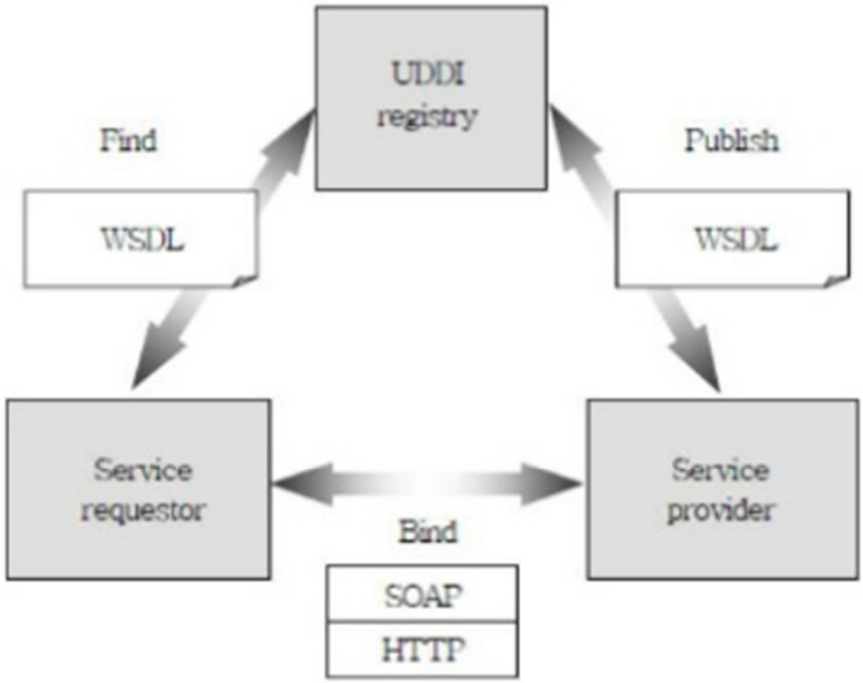
* 기존 원격 기술들에 비해서 프록시와 방화벽에 구애 받지 않고 쉽게 통신이 가능하다.
* 플랫폼과 프로그래밍 언어에 독립적이다.
* 웹 서비스를 제공하기 위한 표준 (WSDL, UDDI, WS-*) 이 잘 정립되어 있다.
* 에러 처리에 대한 내용이 기본으로 내장되어 있다.
* 분산 환경에 적합하다.

단점

* 복잡한 구조로 인한 오버헤드가 있으며, 이는 SOAP 의 확장을 저해하고 있다.
* REST 에 비해 상대적으로 무겁고 속도도 느리다.
* 개발 난이도가 높아 개발 환경의 지원이 필요하다.

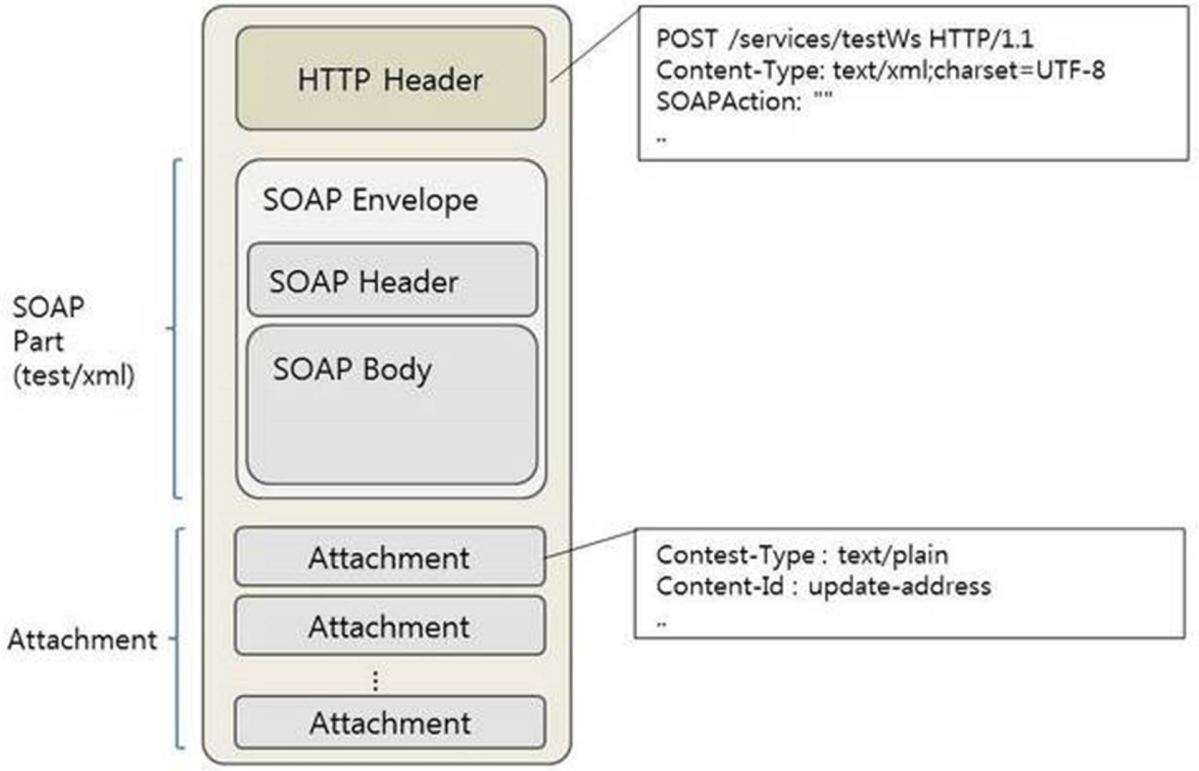
3.2 SOAP 아키텍처


Visa Validator Client
Credit Card
Validate
논리적 통신
VisaValidator XML Web Service
Proxy class
SOAP
HTTP
TCP/IP
IIS/ASP.NET
SOAP
HTTP
TCP/IP
물리적인 통신


UDDI registry
Find
Publish
WSDL
WSDL
Service requestor
Service provider
Bind
SOAP
HTTP

SOAP 의 아키텍처는 크게 UDDI 레지스트리(UDDI registry), 서비스 요청자(Service requestor), 서비스 제공자(Service provider)로 구성됩니다. 서비스 공급자는 웹 서비스를 UDDI 레지스트리에 등록 (Publish)하며, 서비스 요청자는 웹 서비스를 UDDI 레지스트리에서 탐색 (Find) 합니다. 서비스 공급자와 서비스 제공자는 SOAP 형태로 인코딩된 메시지로 바인딩 (Bind) 합니다. 다시말하면 서비스 요청자는 SOAP 으로 인코딩하여 웹 서비스를 요청합니다. 서비스 제공자는 이를 디코딩하여 적절한 서비스 로직을 수행시켜서 결과를 얻고 그 결과를 다시 SOAP 으로 인코딩하여 반환합니다.

3.3 SOAP 메시지의 구조



SOAP 메시지는 HTTP 헤더를 포함하며, SOAP Envelope 에 SOAP Header 와 SOAP Body 를 포함하도록 구성되어 있습니다.

4. Reference

http://www.nextree.co.kr/p11205/

http://wonzopein.com/50

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

http://a07274.tistory.com/m/209
http://a07274.tistory.com/209 - jsonView

https://daehwann.wordpress.com/2014/07/14/building-a-restful-web-service/
http://blog.secmem.org/525