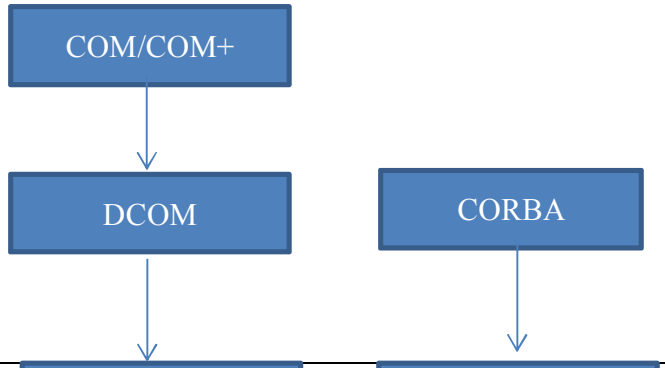
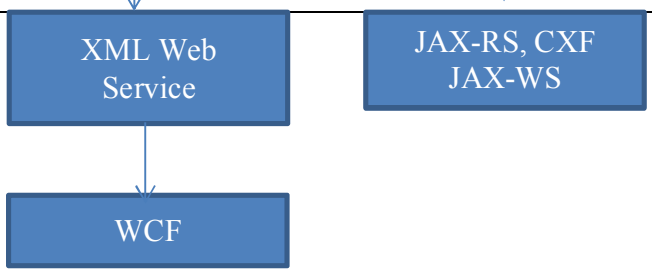
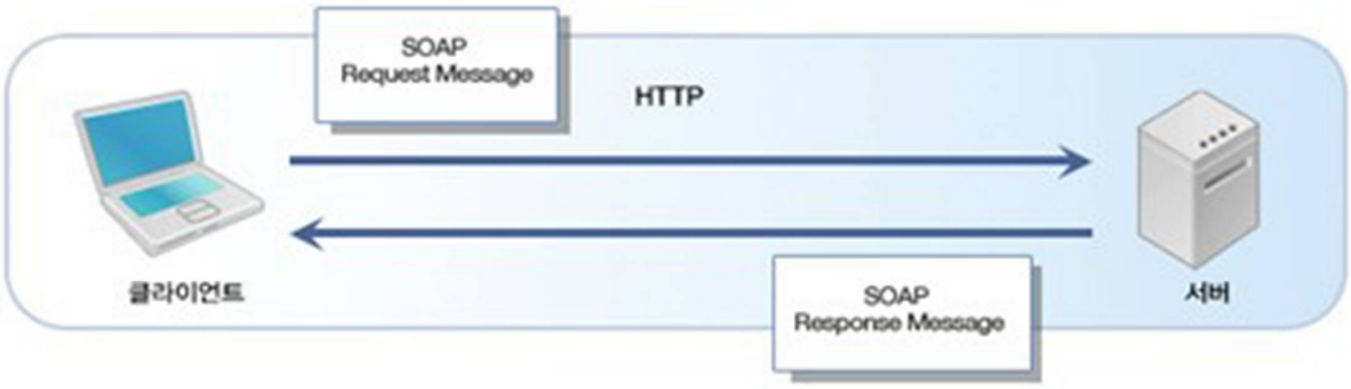
st09.REST 서비스만들기

목차

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>SOAP</td><td> 
</td></tr>
<tr><td>REST</td><td> 
</td></tr>
</table>



[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?

REST 1

•2000 년 Roy Fielding 의 박사 학위 논문에서 처음 제안
•REpresentational State Transfer
•웹 (WWW) 과 같은 분산 시스템을 위한 소프트웨어 아키텍처 스타일

RESTful 2

•다음 제약을 따르면 ...
1.Client-server
2.Stateless
3.Cacheable
4.Layered system
5.Uniform interface
(GET, PUT, POST, DELETE, ...)

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.

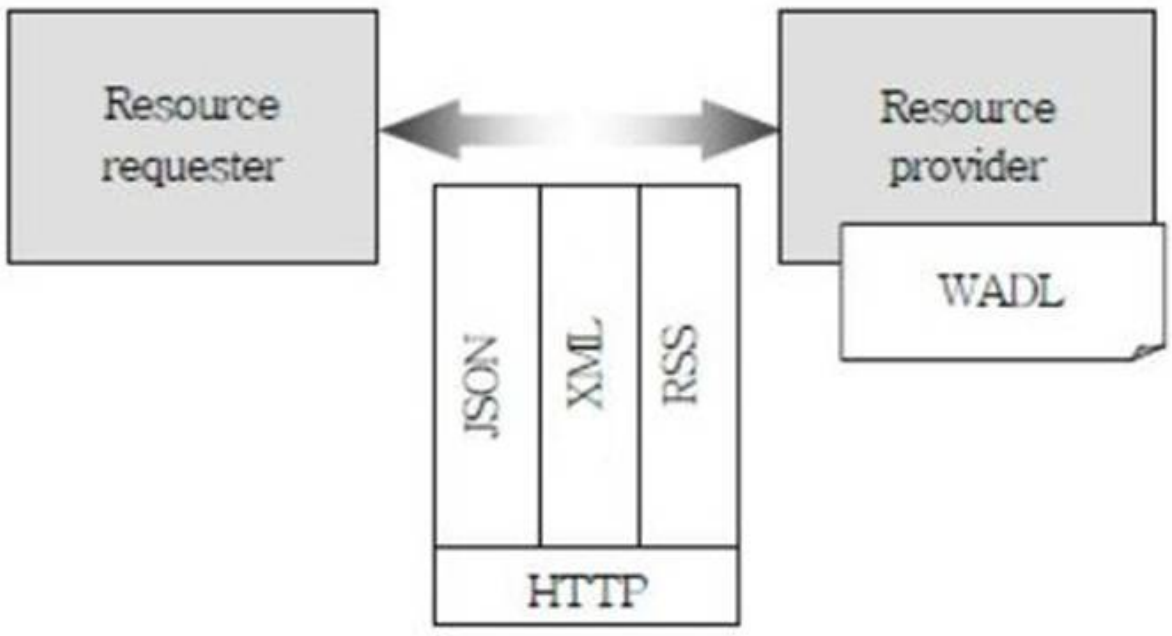
장점

* 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

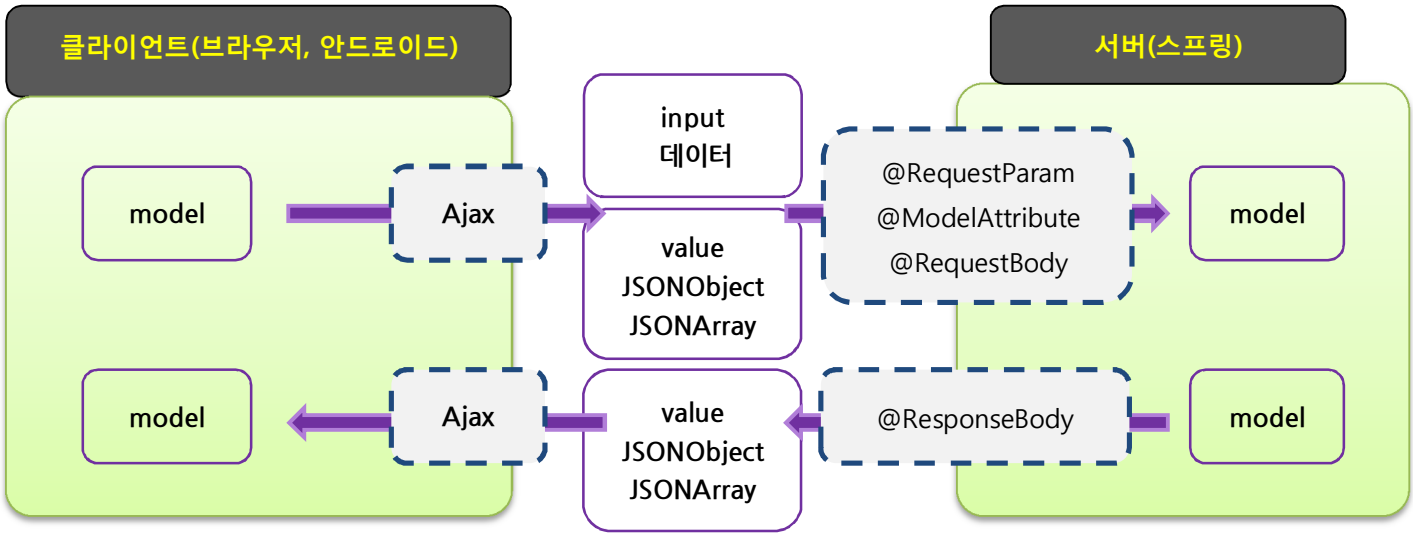


REST 의 아키텍처는 자원 요청자(Resource requester), 자원 제공자(Resource provider) 로 구성됩니다. REST 는 자원을 등록하고 저장해주는 중간 매체 없이 자원 제공자가 직접 자원 요청자에게 제공합니다. REST 는 기본 HTTP 프로토콜의 메소드인 GET/PUT/POST/DELETE 를 이용하여 자원 요청자는 자원을 요청합니다. 자원 제공자는 다양한 형태로 표현된 (JSON, XML, RSS 등)의 리소스를 반환합니다.

1.3 REST 메시지의 구조

2. Json

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



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

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

3. REST 서비스를 위한 스프링 설정

- 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'

// gson : json 객체를 자바 인스턴스로 바꾸는 구글 라이브러리
compile group: "com.google.code.gson", name: "gson", version: "2.8.0"
compile group: "org.json"            , name: "json", version: "20140107"
```

- servlet-context.xml 에 어노테이션 설정해줍니다.

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

- servlet-context.xml 에 아래 코드 추가

```
<!-- step7. @ReponseBody 를 사용 할 때 한글 깨짐 해결 설정 -->
<annotation-driven>
    <message-converters>
        <!-- @ResponseBody 로 String 처리할때 한글처리 -->
        <beans:bean
            class="org.springframework.http.converter.StringHttpMessageConverter">
            <beans:property name="supportedMediaTypes">
                <beans:list>
                    <beans:value>text/html;        charset=utf8</beans:value>
                    <beans:value>application/json; charset=utf8</beans:value>
                    <beans:value>application/xml;  charset=utf8</beans:value>
                </beans:list>
            </beans:property>
        </beans:bean>
    </message-converters>
</annotation-driven>
```

4. DataStorePerson.java 만들기

DataStorePerson 클래스는 ModelPerson 을 List 로 처리하기 위한 데이터 저장소 역활을 하는 클래스다.

- DataStorePerson.java 전체 소스

```
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Logger;
import java.util.stream.Collectors;

import org.springframework.stereotype.Service;

import com.spring61.rest.model.ModelPerson;

@Service
public final class DataStorePerson {

    private static final Logger logger =
Logger.getLogger(DataStorePerson.class.getName());

    private static List<ModelPerson> lists = null;

    public DataStorePerson() {
        lists = DataStorePerson.makePersonData(1, 100);
    }

    public static int login(String id, String pw) {
        logger.info("login " );
        synchronized (lists) {

            List<ModelPerson> result = lists.stream()
                              .filter( e-> e.getId().equals(id) && e.getPw().equals(pw) )
                              .collect( Collectors.toList() );

            return result != null ? result.size() : 0;
        }
    }

    public static List<ModelPerson> select(ModelPerson person) {
        logger.info("select " + person);
        synchronized (lists) {
            return new ArrayList<ModelPerson>(lists);
        }
```

```
    }

    public static boolean insert(ModelPerson person) {
        logger.info("insert " + person);
        synchronized (lists) {
            return lists.add(person);
        }
    }

    public static boolean insert( List<ModelPerson> persons) {
        logger.info("insert " + persons);
        synchronized (lists) {
            return lists.addAll(lists.size(), persons);
        }
    }

    public static boolean update(ModelPerson oldp, ModelPerson newp) {
        logger.info("update " + oldp + " to " + newp);

        synchronized (lists) {
            lists.remove(oldp);
            lists.add(newp);
            return true;
        }
    }

    public static boolean delete(ModelPerson person) {
        logger.info("delete " + person);
        synchronized (lists) {
            return lists.remove(person);
        }
    }

    public static String getRandString() {
        String str = "";

        for (int i = 1; i <= (int) (Math.random() * 10000); i++) {
            str += String.valueOf((char) ((Math.random() * 26) + 97));
        }

        return str;
    }

    public static List<ModelPerson> makePersonData(Integer start, Integer end ) {
        List<ModelPerson> result = new ArrayList<>();
```

```
        for(int i = start ; i<= end; i++) {
            ModelPerson person = new ModelPerson();

            person.setId   ( "id"    + String.valueOf(i) );
            person.setPw   ( "pw"    + String.valueOf(i) );
            person.setName ( "name"  + String.valueOf(i) );
            person.setEmail( "email" + String.valueOf(i) );

            result.add(person);
        }

        return result;
    }
}
```

5. 클라언트에서 서버로 데이터를 보내는 방법

메소드 앞에 @ResponseBody 를 붙여서 사용하게되면 해당 객체가 자동으로 Json 객체로 변환되어 반환됩니다.

1. 매개변수 앞에 @RequestParam, @ModelAttribute 를 붙이는 방법
2. 매개변수 앞에 @RequestBody 을 붙이는 방법
   A. 클라이언트로 부터 넘겨 받은 json 데이터가 자바의 모델 인스턴스로 자동 변환된다.

5.1 RestPersonController

```
@Controller
@RequestMapping(value = "/rest" )
public class RestPersonController {

     @Autowired
     DataStorePerson service;

}
```

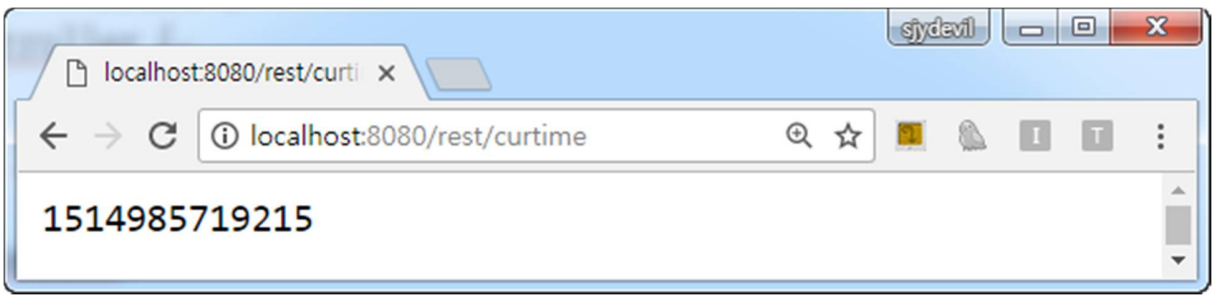
5.2 curtime 구현하기

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

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

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

- 실행 화면



5.3 login 구현하기

Query String 으로 클라이언트의 데이터를 서버로 전송

- http://localhost:8080/rest/loginperson

```
@RequestMapping(value = "/loginperson", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public int login( RequestParam(value="id", defaultValue="") String id
                , @RequestParam(value="pw", defaultValue="") String pw) {
    logger.info("/rest/loginperson");

    return DataStorePerson.loginperson( id, pw );
}
```

5.4 selectparam 구현하기

Query String 으로 클라이언트의 데이터를 서버로 전송

- http://localhost:8080/rest/selectparam

```
// /rest/selectparam?name=id1 :: Query String 으로 클라이언트의 데이터를 서버로 전송
@RequestMapping(value = "/selectparam", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public List<ModelPerson> selectparam(Model model
        , @RequestParam(value="name", defaultValue="") String name ) {
    logger.info("/rest/selectparam");

    ModelPerson person = new ModelPerson();
    person.setName(name);

    return DataStorePerson.select(person);
}
```

5.5 selectmodel 구현하기

form 태그를 이용하여 post 방식으로 클라이언트의 데이터를 서버로 전송하는 REST 메서드를 만들어 본다.

- http://localhost:8080/rest/selectmodel

```
// /rest/selectmodel :: form 태그를 이용하여 클라이언트의 데이터를 서버로 전송
// <form action="/rest/selectmodel" method="post">
//     <input type="text"     name="id"    />
//     <input type="password" name="pw"    />
//     <input type="text"     name="name"  />
//     <input type="text"     name="email" />
```

```
//     <input type="button"   value="send" />
// </form>
@RequestMapping(value = "/selectmodel", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public List<ModelPerson> selectmodel(@ModelAttribute ModelPerson person ) {
    logger.info("/rest/selectmodel");

    return DataStorePerson.select(person);
}
```

5.6 selectjson

ajax 를 이용하여 json 데이터를 클라이언트에서 서버로 전송할 때 사용하는 방법이다.

- http://localhost:8080/rest/selectjson

```
// /rest/selectjson :: ajax 를 이용하여 json 데이터를 클라이언트에서 서버로 전송.
// 클라이언트 ---전송(JSONObject)---> 서버 --- 반환(JSONArray)--->클라이언트
// var model = { 'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
@RequestMapping(value = "/selectjson", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public List<ModelPerson> selectjson(@RequestBody ModelPerson person ) {
    logger.info("/rest/selectjson");

    return DataStorePerson.select(person);
}
```

5.7 selectmap

gson 은 json 객체를 자바 인스턴스로 바꾸는 구글 라이브러리다.
클라이언트에서 json 안에 여러 개의 자바스크립트 객체를 담아서 서버로 보내는 경우에는 스프링의 메서드에서는 매개변수를 Map 으로 처리하여야 한다.

- http://localhost:8080/rest/selectmap

```
/**
 * 클라이언트 ---전송(JSONObject)---> 서버 --- 반환(JSONArray)--->클라이언트
 *
 * 1. 안드로이드 ---> 스프링     : 안드로이드는 json 을 스프링으로 전송한다.
 * 2. 스프링     ---> 안드로이드 : 스프링에서 안드로이드로 값을 1 개 반환한다.
 *
 * var map = {
```

```
 *            'searchvalue' : { 'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
 *          , 'orderby'     : 'desc'
 *          };
 */
@RequestMapping(value = "/selectmap", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public List<ModelPerson> selectmap( RequestBody Map<String, Object> map) {
    logger.info("/rest/selectmap");

    // gson 을 이용하여 json 을 model 로 변환.
    ModelPerson person = new Gson().fromJson(map.get("searchvalue").toString(),
ModelPerson.class);
    String      name   = (String) map.get("orderby");

    return  DataStorePerson.makePersonData(100, 200);
}
```

5.8 selectpaging

- http://localhost:8080/rest/selectpaging?start=1&end=200

```
// /rest/selectpaging?start=100&end=200
@RequestMapping(value = "/selectpaging", method = RequestMethod.GET)
@ResponseBody
public List<ModelPerson> selectpaging(
         @RequestParam(value="start", defaultValue = "0" ) Integer start
       , @RequestParam(value="end"  , defaultValue = "10") Integer end   ) {
    logger.info("/rest/selectpaging");

    return DataStorePerson.makePersonData(start, end) ;
}
```

6. 서버에서 클라언트로 데이터를 보내는 방법 : @ResponseBody

메소드 앞에 @ResponseBody 를 붙여서 사용하게되면 해당 객체가 자동으로 Json 객체로 변환되어 반환됩니다.

6.1 @ResponseBody 설정 방법

- 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 />
```

6.2 insertparam

- http://localhost:8080/rest/insertparam?name=inserttest

```
// /rest/insertparam?name=inserttest
@RequestMapping(value = "/insertparam", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public boolean insertparam(@RequestParam(value="name", defaultValue="") String name) {
    logger.info("/rest/insertparam");

    ModelPerson person = new ModelPerson();
    person.setName(name);

    return DataStorePerson.insert(person);
}
```

6.3 insertmodel

- http://locahost:8080/rest/insertmodel

```
/**
 * /rest/insertmodel :: form 태그를 이용하여 클라이언트의 데이터를 서버로 전송
 *
 * <form action="/rest/insertmodel" method="post">
```

```
 *     <input type="text"     name="id"    />
 *     <input type="password" name="pw"    />
 *     <input type="text"     name="name"  />
 *     <input type="text"     name="email" />
 *     <input type="button"   value="send" />
 * </form>
 */
@RequestMapping(value = "/insertmodel", method = {RequestMethod.GET, RequestMethod.POST} )
@ResponseBody
public boolean insertmodel(@ModelAttribute ModelPerson person) {
    logger.info("/rest/insertmodel");

    return DataStorePerson.insert(person);
}
```

6.4 insertjsonobject

- http://localhost:8080/rest/insertjsonobject

```
/**
 *  ajax를 이용하여 json 데이터를 클라이언트에서 서버로 전송.
 *
 * 클라이언트 ---전송(json:JSONObject)---> 서버 --- 반환(json:boolean)--->클라이언트
 *
 *  var model = {'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
 *
 * 1. 안드로이드 ---> 스프링     : 안드로이드는 json을 스프링으로 전송한다.
 * 2. 스프링     ---> 안드로이드 : 스프링에서 안드로이드로 값을 1개 반환한다.
 */
@RequestMapping(value = "/insertjsonobject", method = {RequestMethod.GET,
RequestMethod.POST} )
@ResponseBody
public boolean insertjsonobject(Model model, @RequestBody ModelPerson person) {
    logger.info("/rest/insertjsonobject");

    return DataStorePerson.insert(person);
}
```

6.5 insertjsonarray

- http://localhost:8080/rest/insertjsonarray

```
/**
```

```
 * ajax를 이용하여 json 데이터를 클라이언트의 데이터를 서버로 전송.
 *
 * 클라이언트---전송(json:JSONArray)---> 서버 --- 반환(json:boolean)--->클라이언트
 *
 * var persons = [ {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
 *               , {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
 *               , {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
 *               ];
 */
@RequestMapping(value = "/insertjsonarray", method = {RequestMethod.GET,
RequestMethod.POST} )
@ResponseBody
public boolean insertjsonarray(@RequestBody List<ModelPerson> persons) {
    logger.info("/rest/insertjsonarray");

    return DataStorePerson.insert(persons);
}
```

7. 전체 소스

7.1 RestPersonControllerl.java 전체 소스

```
import java.util.*;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import com.google.gson.Gson;
import com.spring61.rest.model.*;

@Controller
@RequestMapping(value = "/rest")
public class RestPersonController {

      private static final Logger logger = LoggerFactory.getLogger(RestPersonController.class);

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

        return new Date().getTime();
    }

    @RequestMapping(value = "/login", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public int login(Model model
            , @RequestParam(value="id", defaultValue="") String id
            , @RequestParam(value="pw", defaultValue="") String pw) {
        logger.info("/rest/login");

        return DataStorePerson.login( id, pw );
    }

    // /rest/selectparam?name=id1 :: Query String 에 클라이언트의 데이터를 서버로 전송
    @RequestMapping(value = "/selectparam", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public List<ModelPerson> selectparam(Model model
            , @RequestParam(value="name", defaultValue="") String name ) {
        logger.info("/rest/selectparam");

        ModelPerson person = new ModelPerson();
        person.setName(name);
```

```
        return DataStorePerson.select(person);
    }

    // /rest/selectmodel :: form 태그를 이용하여 클라이언트의 데이터를 서버로 전송
    // <form action="/rest/selectmodel" method="post">
    //     <input type="text"     name="id"    />
    //     <input type="password" name="pw"    />
    //     <input type="text"     name="name"  />
    //     <input type="text"     name="email" />
    //     <input type="button"   value="send" />
    // </form>
    @RequestMapping(value = "/selectmodel", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public List<ModelPerson> selectmodel(@ModelAttribute ModelPerson person ) {
        logger.info("/rest/selectmodel");

        return DataStorePerson.select(person);
    }

    // /rest/selectjson :: ajax 를 이용하여 json 데이터를 클라이언트의 데이터를 서버로 전송.
    // var model = { 'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
    @RequestMapping(value = "/selectjson", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public List<ModelPerson> selectjson(@RequestBody ModelPerson person ) {
        logger.info("/rest/selectjson");

        return DataStorePerson.select(person);
    }

    /**
     * 클라이언트 ---전송(json:JSONObject)---> 서버(spring) --- 반환(json:JSONArray)--->클라이언트
     *
     * 1. 안드로이드 ---> 스프링     : 안드로이드는 json 을 스프링으로 전송한다.
     * 2. 스프링     ---> 안드로이드 : 스프링에서 안드로이드로 값을 1 개 반환한다.
     *
     * var map = {
     *                'searchvalue' : { 'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
     *              , 'orderby'     : 'desc'
     *           };
     */
    @RequestMapping(value = "/selectmap", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public List<ModelPerson> selectmap(Model model
            , @RequestBody Map<String, Object> map) {
        logger.info("/rest/selectmap");

        // gson 을 이용하여 json 을 model 로 변환.
        ModelPerson person = new Gson().fromJson(map.get("searchvalue").toString(),
ModelPerson.class);
```

```
        String      name   = (String) map.get("orderby");

        return DataStorePerson.makePersonData(100, 200);
    }

    //rest/selectpaging?start=100&end=200
    @RequestMapping(value = "/selectpaging", method = RequestMethod.GET)
    @ResponseBody
    public List<ModelPerson> selectpaging( Model model
          , @RequestParam(value="start", required = false, defaultValue = "0" ) Integer start
          , @RequestParam(value="end"  , required = false, defaultValue = "10") Integer end) {
        logger.info("/rest/selectpaging");


        List<ModelPerson> result = DataStorePerson.makePersonData(start, end);

        return result ;
    }




    @RequestMapping(value = "/insertparam", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public boolean insertparam(@RequestParam(value="name", defaultValue="") String name) {
        logger.info("/rest/insertparam");

        ModelPerson person = new ModelPerson();
        person.setName(name);

        return DataStorePerson.insert(person);
    }

    /**
     * /rest/insertmodel :: form 태그를 이용하여 클라이언트의 데이터를 서버로 전송
     *
     * <form action="/rest/insertmodel" method="post">
     *     <input type="text"     name="id"    />
     *     <input type="password" name="pw"    />
     *     <input type="text"     name="name"  />
     *     <input type="text"     name="email" />
     *     <input type="button"   value="send" />
     * </form>
     */
    @RequestMapping(value = "/insertmodel", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public boolean insertmodel(@ModelAttribute ModelPerson person) {
        logger.info("/rest/insertmodel");

        return DataStorePerson.insert(person);
```

```
    }

    /**
     *  ajax를 이용하여 json 데이터를 클라이언트에서 서버로 전송.
     *
     * 클라이언트 ---전송(json:JSONObject)---> 서버(spring) --- 반환(json:boolean)--->클라이언트
     *
     *  var model = {'id': '???', 'pw':'???', 'name':'???', 'email':'???' }
     *
     * 1. 안드로이드 ---> 스프링     : 안드로이드는 json을 스프링으로 전송한다.
     * 2. 스프링     ---> 안드로이드 : 스프링에서 안드로이드로 값을 1개 반환한다.
     */
    @RequestMapping(value = "/insertjsonobject", method = {RequestMethod.GET,
RequestMethod.POST} )
    @ResponseBody
    public boolean insertjsonobject(Model model, @RequestBody ModelPerson person) {
        logger.info("/rest/insertjsonobject");

        return DataStorePerson.insert(person);
    }

    /**
     * ajax를 이용하여 json 데이터를 클라이언트의 데이터를 서버로 전송.
     *
     * 클라이언트 ---전송(json:JSONArray)---> 서버(spring) --- 반환(json:boolean)--->클라이언트
     *
     * var persons = [ {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
     *               , {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
     *               , {'id': '???', 'pw':'???', 'name':'???', 'email':'???'}
     *               ];
     */
    @RequestMapping(value = "/insertjsonarray", method = {RequestMethod.GET, RequestMethod.POST} )
    @ResponseBody
    public boolean insertjsonarray(Model model, @RequestBody List<ModelPerson> persons) {
        logger.info("/rest/insertjsonarray");

        return DataStorePerson.insert(persons);
    }
}
```