1. Open API?
  1)개방형 API
  2)Programming에서 사용할 수 있는 개방되어 있는 상태의 interface를 말한다.
  3)Portal이나 통계청, 기상청 등과 같은 관공서에서 가지고 있는 data를 외부 응용 program에서 사용할 수 있도록 Open API를 제공하고 있다.
  4)Open API와 함께 사용하는 기술 중 REST가 있고, 대부분의 Open API는 REST 방식으로 지원되고 있다.

2. RESTful Web Service 개요
  1)REST(REpresentational Safe Trasfer)
    -HTTP URI + HTTP Method
    -HTTP URI를 통해 제어할 자원(Resource)을 명시하고, HTTP Method(GET, POST, PUT, DELETE)를 통해 해당 Resource를 제어하는 명령을 내리는 방식의 architecture.
    -HTTP protocol에 정의된 4개의 method들이 Resource에 대한 CRUD Operation을 정의
      --POST : Create(Insert)
      --GET : Read(Select)
      --PUT : Update or Create
      --DELETE : Delete

  2)RESTful API?
    -HTTP와 URI 기반으로 자원에 접근할 수 있도록 제공하는 Application 개발 interface.
    -즉, REST의 원리를 따르는 System을 가리키는 용어로 사용
    -기존의 Web 접근 방식과 RESTful API 방식과의 차이점(예:게시판)

기존 게시판	RESTful API를 지원하는 게시판
--글읽기 : GET /list.do?no=4&name=Spring	GET /board/Spring/4
--글등록 : POST /insert.do	POST /board/Spring/4
--글삭제 : GET /delete.do?no=4&name=Spring	DELETE /board/Spring/4
--글수정 : POST /update.do	PUT /board/Spring/4

    -기존의 게시판은 GET과 POST만으로 자원에 대한 CRUD를 처리하며, URI는 Action을 나타낸다.
    -RESTful 게시판은 4가지 method를 모두 사용하여 CRUD를 처리하며, URI는 제어하려는 자원을 나타낸다.

3. JSON과 XML
  1)RESTful Web Service와 JSON XML.png 그림 참조

  2)JSON(JavaScript Object Notation)?
    -http://www.json.org
    -경량의 Data 교환 format
    -JavaScript에서 객체를 만들 때 사용하는 표현식을 의미
    -JSON 표현식은 사람과 기계 모두 이해하기 쉬우며 용량이 작아서, 최근에는 XML을 대체해서 data 전송등에 많이 사용된다.
    -특정 언어에 종속되지 않으며, 대부분의 programming 언어에서 JSON format의 data를 handling할 수 있는 library를 제공하고 있다.
    -name : value 형식의 pair

```
{
  "name" : "조용필",
  "gender" : "남성",
  "age" : 50,
  "city" : "Seoul",
  "hobby" : ["등산", "낚시", "게임"]
}
```

  3)JSON library - Jackson

```
   -http://jackson.codehous.org
   -High-Performance JSON Processor!
   -Jackson은 JSON 형태를 Java 객체로, Java 객체를 JSON 형태로 변환해 주는 Java용 JSON library이다.
   -가장 많이 사용하는 JSON library이다.

   JSON(Browser) <---> Java Object(Back-end) <---> RDBMS(Storage)
                       Jackson                     Mybatis

4)XML?
   -eXtensible Markup Language
   -Data를 저장하고 전달/교환하기 위한 언어
   -인간/기계 모두에게 읽기 편한 언어
   -data의 구조와 의미를 설명
   -HTML이 Data의 표현에 중점을 두었다면 XML은 Data를 전달하는 것에 중점을 맞춘 언어
   -HTML은 미리 정의된 Tag만 사용 가능하지만, XML은 사용자가 Tag를 정의할 수 있다.
   <?xml version="1.0" encoding="UTF-8"?>
   <products>
     <product>
       <name>Ballpen</name>
       <price 단위="원">150</price>
       <maker>모나미</maker>
       <color>black</color>
     </product>
   </products>

5)Jackson version 1 library 설치
   -http://mvnrepository.com에서 'jackson mapper'로 검색
   -'Data Mapper For Jackson' 1.9.13 버전을 pom.xml에 추가
   <!-- https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-mapper-lgpl -->
     <dependency>
     <groupId>org.codehaus.jackson</groupId>
     <artifactId>jackson-mapper-lgpl</artifactId>
     <version>1.9.13</version>
   </dependency>

6)Jackson2 API 설치
   -http://mvnrepository.com에서 'jackson databind'로 검색
   -'Jackson Databind' 2.9.9 버전을 pom.xml에 추가
   -'Jackson Core' 2.9.9 버전을 pom.xml에 추가
   -'Jackson Annotations' 2.9.9 버전을 pom.xml에 추가

     <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
     <dependency>
       <groupId>com.fasterxml.jackson.core</groupId>
       <artifactId>jackson-databind</artifactId>
       <version>2.9.9</version>
     </dependency>
     <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
     <dependency>
       <groupId>com.fasterxml.jackson.core</groupId>
       <artifactId>jackson-core</artifactId>
       <version>2.9.9</version>
     </dependency>
     <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
```

```
        -->
105     <dependency>
106       <groupId>com.fasterxml.jackson.core</groupId>
107       <artifactId>jackson-annotations</artifactId>
108       <version>2.9.9</version>
109     </dependency>
110
111   7)web.xml의 DispatcherServlet url-pattern 변경
112     --기존--
113     <servlet-mapping>
114       <servlet-name>springDispatcherServlet</servlet-name>
115       <url-pattern>*.do</url-pattern>
116     </servlet-mapping>
117
118     --변경--
119     <servlet-mapping>
120       <servlet-name>springDispatcherServlet</servlet-name>
121       <url-pattern>/</url-pattern>
122     </servlet-mapping>
123
124   8)Spring Bean Configuration File(beans.xml) 설정
125     -Spring MVC에 필요한 Bean들을 자동으로 등록해주는 Tag
126       <mvc:annotation-driven />
127
128   9)Spring MVC기반 RESTful Web Service 구현 절차
129     -RESTful Web Service를 처리할 RestfulController class 작성 및 Spring Bean으로 등록
130     -요청을 처리할 method에 @RequestMapping @RequestBody와 @ResponseBody annotation 선언
131     -REST Client Tool(Postman)을 사용하여 각각의 method test
132     -Ajax 통신을 하여 RESTful web service를 호출하는 HTML page 작성
133
134   10)사용자 관리 RESTful Web Service URI와 Method
135     Action Resource URI HTTP Method
136     -사용자 목록 /users GET
137     -사용자 보기 /users/{id} GET
138     -사용자 등록 /users POST
139     -사용자 수정 /users PUT
140     -사용자 삭제 /users/{id} DELETE
141
142   11)RESTful Controller를 위한 핵심 Annotation
143     -Spring MVC에서는 Client에서 전송한 XML이나 JSON data를 Controller에서 Java 객체로 변환해서 받을 수
        있는 기능(수신)을 제공하고 있다.
144     -Java객체를 XML이나 JSON으로 변환해서 전송할 수 있는 기능(송신)을 제공하고 있다.
145
146     -Annotation 설명
147       --@RequestBody : HTTP Request Body(요청 몸체)를 Java객체로 전달받을 수 있다.
148       --@ResponseBody : Java객체를 HTTP Response Body(응답 몸체)로 전송할 수 있다.
149
150
151 4. Google Postman 설치
152   1)https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop?hl
      =en
153   2)[Launch app] button click
154   3)Log in - Sign with Google
155
```

5. Data 변환 - JSON으로 변환
1)System이 복잡해지면서 다른 system과 정보를 주고받을 일이 발생하는데, 이 때 data 교환 format으로 JSON을 사용할 수 있다.
2)검색결과를 JSON data로 변환하려면 가장 먼저 jackson2 library를 download 받아야 한다.
3)Jackson2는 Java 객체를 JSON으로 변환하거나 JSON을 Java 객체로 변환해주는 library다.
4)https://www.concretepage.com/spring-4/spring-4-rest-xml-response-example-with-jackson-2 참조
5)https://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/ 참조
6)pom.xml에 다음과 같이 dependency를 추가한다.

```
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.9</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.9</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations -->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-annotations</artifactId>
  <version>2.9.9</version>
</dependency>
```

7)Maven clean > Maven Install하면 Maven Dependencies에 아래와 같은 jar file이 추가된다.
-jackson-databind-2.9.9.jar
-jackson-annotations-2.9.9.jar
-jackson-core-2.9.9.jar

8)보통 User가 Servlet이나 JSP를 요청하면 server는 요청한 file을 찾아서 실행한다.
9)그 실행결과는 HTTP Response package의 body에 저장하여 Browser에 전송한다.
10)그런데, 이 응답결과를 HTML이 아니라 JSON이나 XML로 변환하여 body에 저장하려면 Spring에서 제공하는 변환기(Converter)를 사용해야 한다.
11)Spring은 HttpMessageConverter를 구현한 다양한 변환기를 제공한다.
12)이 변환기를 이용하면 Java 객체를 다양한 타입으로 변환하여 HTTP Response body에 설정할 수 있다.
13)HttpMessageConverter를 구현한 class는 여러가지가 있으며, 이 중에서 Java 객체를 JSON responsebody로 변환할 때는 MappingJackson2HttpMessageConverter를 사용한다.
14)따라서 MappingJackson2HttpMessageConverter를 Spring 설정 file에 등록하면 되는데, 혹시 이후에 XML 변환도 처리할 예정이라면 다음처럼 설정한다.

```
<mvc:annotation-driven />
```

15)Spring Bean Configuration File에 위와 같이 설정하면 HttpMessageConverter를 구현한 모든 변환기가 생성된다.
16)src/com.example.controller.UserController.java에 다음과 같이 수정한다.

```
package com.example.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
```

```
import org.springframework.web.bind.annotation.ResponseBody;

import com.example.service.UserService;
import com.example.vo.UserVO;

@Controller
public class UserController {
  @Autowired
  private UserService userService;

  /*@RequestMapping("/userinfo.do")
  public String getUserList(@RequestParam("userId") String userId, Model model) {
    UserVO user = userService.getUser(userId);
    model.addAttribute("user", user);
    return "userinfo.jsp";
  }*/

  @RequestMapping("/userinfo.do")
  @ResponseBody
  public UserVO userinfo(@RequestParam("userId") String userId) {
    return userService.getUser(userId);
  }
}
```

17)이전 method와 달리 @ResponseBody라는 annotation을 추가했는데, Java 객체를 Http Response protocol의 body로 변환하기 위해 사용된다.
18)이미 Spring Configuration File에 <mvc:annotation-driven>을 추가했기 때문에 @ResponseBody가 적용된 method의 실행 결과는 JSON으로 변환되어 HTTP Response Body에 다음과 같이 설정된다.

```
{"userId":"jimin","name":"한지민","gender":"여","city":"서울"}
```

19)만일 이때, Java 객체를 JSON으로 변환할 때, 특정 변수를 제외시키려면 @JsonIgnore annotation을 해당 변수의 getter에 설정하면 된다.

```
package com.example.vo;
import com.fasterxml.jackson.annotation.JsonIgnore;
public class UserVO {
  ...
  @JsonIgnore
  public String getGender() {
    return gender;
  }
```

20)이렇게 하면 아래와 같이 성별이 포함되지 않는다는 것을 알 수 있다.

```
{"userId":"jimin","name":"한지민","city":"서울"}
```

21)Postman test
GET http://localhost:8080/SpringWebDemo/userinfo.do/jimin Send

Body JSON

```
{
  "userId": "jimin",
```

```
      "name": "한지민",
      "gender": "여",
      "city": "서울"
    }


6. Lab
  1)In J2EE Perspective
  2)Project Explorer > right-click > New > Dynamic Web Project
  3)Project name : RestfulDemo > Next > Check [Generate web.xml deployment descriptor] > Finish
  4)Convert to Maven Project
    -project right-click > Configure > Convert to Maven Project > Finish

  5)Add Spring Project Nature
    -project right-click > Spring > Add Spring Project Nature

  6)새로 생성된 pom.xml file에 필요한 library 추가 > Maven Clean > Maven Install
    <dependencies>
      <!-- https://mvnrepository.com/artifact/org.springframework/spring-context -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>4.3.24.RELEASE</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/junit/junit -->
      <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.12</version>
        <scope>test</scope>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-jdbc</artifactId>
        <version>4.3.24.RELEASE</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
      <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.3.24.RELEASE</version>
      </dependency>
      <dependency>
        <groupId>javax.servlet</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
      </dependency>
      <dependency>
        <groupId>com.oracle</groupId>
        <artifactId>ojdbc8</artifactId>
        <version>12.2</version>
      </dependency>
```

```
308         <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
309         <dependency>
310           <groupId>com.fasterxml.jackson.core</groupId>
311           <artifactId>jackson-databind</artifactId>
312           <version>2.9.9</version>
313         </dependency>
314         <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
315         <dependency>
316           <groupId>com.fasterxml.jackson.core</groupId>
317           <artifactId>jackson-core</artifactId>
318           <version>2.9.9</version>
319         </dependency>
320         <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
            -->
321         <dependency>
322           <groupId>com.fasterxml.jackson.core</groupId>
323           <artifactId>jackson-annotations</artifactId>
324           <version>2.9.9</version>
325         </dependency>
326         <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
327         <dependency>
328           <groupId>org.mybatis</groupId>
329           <artifactId>mybatis-spring</artifactId>
330           <version>2.0.1</version>
331         </dependency>
332         <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
333         <dependency>
334           <groupId>org.mybatis</groupId>
335           <artifactId>mybatis</artifactId>
336           <version>3.5.1</version>
337         </dependency>
338       </dependencies>
339
340       -pom.xml > right-click > Run As > Maven install
341         [INFO] BUILD SUCCESS
342
343     7)Build path에 config foler 추가
344       -project right-click > Build Path > Configure Build Path > Select [Source] tab
345       -Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
346       -Folder name : config > Finish > OK > Apply and Close
347       -Java Resources > config folder 확인
348
349     8)config folder에 applicationContext.xml file 생성
350       -Spring Perspective로 전환
351       -config right-click > New > Spring Bean Configuration File
352       -File name :  applicationContext.xm
353       -생성시 beans,context, mvc check
354       <?xml version="1.0" encoding="UTF-8"?>
355       <beans xmlns="http://www.springframework.org/schema/beans"
356         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
357         xmlns:context="http://www.springframework.org/schema/context"
358         xmlns:mvc="http://www.springframework.org/schema/mvc"
359         xsi:schemaLocation="http://www.springframework.org/schema/mvc
            http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
```

```
360         http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring-beans.xsd
361         http://www.springframework.org/schema/context
            http://www.springframework.org/schema/context/spring-context-4.3.xsd">
362
363     </beans>
364
365     9)ContextLoaderListener class 설정
366       -Business logic의 Spring 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로
          ContextLoaderListener class를 정의해야 한다.
367       -ContextLoaderListener class는 Spring 설정 file(default에서 file명 applicationContext.xml)을 load
          하면 ServletContextListener interface를 구현하고 있기 때문에 ServletContext instance 생성시(Tomcat
          으로 application이 load된 때)에 호출된다.
368       -즉, ContextLoaderListener class는 DispatcherServlet class의 load보다 먼저 동작하여 business logic
          층을 정의한 Spring 설정 file을 load한다.
369       -web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener
          -ContextLoaderListener] 를 선택하면 아래의 code가 자동 삽입
370         <!-- needed for ContextLoaderListener -->
371         <context-param>
372           <param-name>contextConfigLocation</param-name>
373           <param-value>location</param-value>
374         </context-param>
375
376         <!-- Bootstraps the root web application context before servlet initialization -->
377         <listener>
378           <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class
              >
379         </listener>
380
381       -아래 code로 변환
382         <context-param>
383           <param-name>contextConfigLocation</param-name>
384           <param-value>classpath:applicationContext.xml</param-value>
385         </context-param>
386
387     10)DispatcherServlet Class 추가
388       -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet
          -DispatcherServlet declaration] 선택하면 아래의 code가 자동 추가된다.
389
390       <!-- The front controller of this Spring Web application, responsible for handling all
          application requests -->
391       <servlet>
392         <servlet-name>springDispatcherServlet</servlet-name>
393         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
394         <init-param>
395           <param-name>contextConfigLocation</param-name>
396           <param-value>location</param-value>
397         </init-param>
398         <load-on-startup>1</load-on-startup>
399       </servlet>
400
401       <!-- Map all requests to the DispatcherServlet for handling -->
402       <servlet-mapping>
403         <servlet-name>springDispatcherServlet</servlet-name>
```

```
404         <url-pattern>url</url-pattern>
405       </servlet-mapping>
406
407       -아래의 code로 변환
408       <servlet>
409         <servlet-name>springDispatcherServlet</servlet-name>
410         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
411         <init-param>
412           <param-name>contextConfigLocation</param-name>
413           <param-value>classpath:beans.xml</param-value>
414         </init-param>
415         <load-on-startup>1</load-on-startup>
416       </servlet>
417
418       <servlet-mapping>
419         <servlet-name>springDispatcherServlet</servlet-name>
420         <url-pattern>/</url-pattern>
421       </servlet-mapping>
422
423     11)MemberVO class 생성
424       -src/com.example.vo package 생성
425       -src/com.example.vo.MemberVO class 생성
426
427         package com.example.vo;
428
429         public class MemberVO {
430           private String name, userid, gender, city;
431           public MemberVO() {}
432           public MemberVO(String name, String userid, String gender, String city) {
433             this.name = name;
434             this.userid = userid;
435             this.gender = gender;
436             this.city = city;
437           }
438           public String getName() {
439             return name;
440           }
441           public void setName(String name) {
442             this.name = name;
443           }
444           public String getUserid() {
445             return userid;
446           }
447           public void setUserid(String userid) {
448             this.userid = userid;
449           }
450           public String getGender() {
451             return gender;
452           }
453           public void setGender(String gender) {
454             this.gender = gender;
455           }
456           public String getCity() {
457             return city;
```

```
458         }
459         public void setCity(String city) {
460           this.city = city;
461         }
462         @Override
463         public String toString() {
464           return "MemberVO [name=" + name + ", userid=" + userid + ", gender=" + gender +
              ", city=" + city + "]";
465         }
466       }
467
468   12)MemberDao 객체 생성
469     -src/com.example.dao package 생성
470     -src/com.example.dao.MemberDao interface
471
472       package com.example.dao;
473
474       import java.util.List;
475
476       import com.example.vo.MemberVO;
477
478       public interface MemberDao {
479         void create(MemberVO member);
480         List<MemberVO> readAll();
481         MemberVO read(String userid);
482         void update(MemberVO member);
483         void delete(String userid);
484       }
485
486     -src/com.example.dao.MemberDaoImpl.java 생성
487
488       package com.example.dao;
489
490       import java.util.List;
491
492       import org.springframework.beans.factory.annotation.Autowired;
493       import org.springframework.stereotype.Repository;
494       import org.apache.ibatis.session.SqlSession;
495
496       import com.example.vo.MemberVO;
497
498       @Repository("memberDao")
499       public class MemberDaoImpl implements MemberDao {
500         @Autowired
501         private SqlSession sqlSession;
502
503         @Override
504         public void create(MemberVO member) {
505
506         }
507
508         @Override
509         public List<MemberVO> readAll() {
510           return null;
```

```
511         }
512
513         @Override
514         public MemberVO read(String userid) {
515           return null;
516         }
517
518         @Override
519         public void update(MemberVO member) {
520
521         }
522
523         @Override
524         public void delete(String userid) {
525
526         }
527       }
528
529
530   13)MemberService 객체 생성
531     -src/com.example.service package 생성
532     -src/com.example.service.MemberService interface
533
534       package com.example.service;
535
536       import java.util.List;
537
538       import com.example.vo.MemberVO;
539
540       public interface MemberService {
541         void insertMember(MemberVO member);
542         List<MemberVO> select();
543         MemberVO selectMember(String userid);
544         void updateMember(MemberVO member);
545         void deleteMember(String userid);
546       }
547
548     -src/com.example.service.MemberServiceImpl.java
549
550       package com.example.service;
551
552       import java.util.List;
553
554       import org.springframework.beans.factory.annotation.Autowired;
555       import org.springframework.stereotype.Service;
556
557       import com.example.dao.MemberDao;
558       import com.example.vo.MemberVO;
559
560       @Service("memberService")
561       public class MemberServiceImpl implements MemberService {
562         @Autowired
563         private MemberDao memberDao;
564
```

```
565         @Override
566         public void insertMember(MemberVO member) {
567
568         }
569
570         @Override
571         public List<MemberVO> select() {
572           return null;
573         }
574
575         @Override
576         public MemberVO selectMember(String userid) {
577           return null;
578         }
579
580         @Override
581         public void updateMember(MemberVO member) {
582
583         }
584
585         @Override
586         public void deleteMember(String userid) {
587
588         }
589       }
590
591   14)HomeController 객체 생성
592     -src/com.example.controller package 생성
593     -com.example.controller.HomeController class 생성
594
595       package com.example.controller;
596
597       import org.springframework.beans.factory.annotation.Autowired;
598       import org.springframework.stereotype.Controller;
599       import org.springframework.ui.Model;
600       import org.springframework.web.bind.annotation.RequestMapping;
601       import org.springframework.web.bind.annotation.RequestParam;
602
603       import com.example.service.MemberService;
604       import com.example.vo.MemberVO;
605
606       @Controller
607       public class HomeController {
608         @Autowired
609         private MemberService service;
610
611       }
612
613   15)config/dbinfo.properties file 생성
614
615     db.driver=oracle.jdbc.driver.OracleDriver
616     db.url=jdbc:oracle:thin:@localhost:1521:XE
617     db.username=scott
618     db.password=tiger
```

16)applicationContext.xml

```
<context:property-placeholder location="classpath:dbinfo.properties"/>
<bean id="dataSource"
class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
  <property name="driverClass" value="${db.driver}"/>
  <property name="url" value="${db.url}"/>
  <property name="username" value="${db.username}"/>
  <property name="password" value="${db.password}"/>
</bean>
```

17)config > right-click > New > Spring Bean configuration File >
-File name : beans.xml
-mvc, context check

```
<context:component-scan base-package="com.example" />
<mvc:annotation-driven />
<mvc:default-servlet-handler/>
```

※<mvc:default-servlet-handler/>
1)Tomcat은 client의 요청 URL을 보고 Servlet Mapping에 따라 URL에 mapping된 Servlet이 처리를 하는 구조이다.
2)그리고 URL에 mapping되는 Servlet이 없다면, 예를들어 CSS, image file 같은 정적자원들은 defaultSevlet이 처리하도록 되어 있다.
3)즉 CSS , image file들은 server 외부에서 직접 접근 할 수 없는 /WEB-INF/assets folder 아래에 위치하는 것이 일반적인데, CSS , image file에 접근하기 위한 Servlet Mapping을 하지 않았으면 Tomcat이 defaultServlet으로 처리하여 정적 file에 접근한다.
4)일반적으로 정적 file에 대해 Servlet Mapping을 하지 않는다는 것이다.
5)Spring에서는 DispatcherServlet이 모든 요청을 받아 들인 후 Handler mapping table에 따라 controller로 분기 한다.
6)그렇기 때문에 DispatcherServlet은 정적 file에 대해 Tomcat이 defaultServlet으로 실행할 수 있는 기회를 뺏어간다.
7)모든 요청은 일단 DispatcherServlet에서 처리해버리기 때문이다.
8)정적 자원에 접근하기 위한 경로 설정을 JSP/Servlet과 똑같이 해도 Spring에서는 경로를 읽지 못한다.
9)그런데 Spring project가 아닌 JSP/Servlet project를 만들어서 똑같은 directory 구조로 같은 code로 <img> tag를 추가하면 정상적으로 응답되는 것을 알 수 있다.
10)JSP/Servlet에서는 Tomcat이 defaultServlet이 있기 때문에 처리가 가능하지만, Spring에서는 DispatcherServlet이 모든 요청을 받아들이기 때문에 Tomcat의 defaultServlet이 정적 file을 처리할 수 있는 기회를 잃게 되는 것이다.
11)이제 Spring에서도 CSS, image file 등이 정상적으로 응답 될 수 있도록 환경 설정을 하도록 하는 방법을 소개한다.
12)spring-servlet.xml 에 아래의 code를 추가하면 된다.

```
<!-- Servlet Container의 default servlet 위임 handler -->
<mvc:default-servlet-handler />
```

13)이 code는 만약 Handler Mapping에 해당하는 URL이 없으면 default-servlet으로 처리하겠다는 의미이다.
14)즉 Mapping이 되지 않은 URL은 webapp folder를 시작으로 경로를 찾아가게 되고, 여기에서도 해당 경로의 자원이 존재하지 않으면 404 Not found가 발생한다.
15)정적 file의 경로를 작성할 때 자신의 application 경로를 project folder 이름으로 작성하는 것 말고, JSTL 표기법으로 작성할 수도 있다.
16)두 번째와 같이 JSTL로 context 경로를 설정하는 방법의 이점은 project folder의 이름을 URL 주소로 사용하고 싶지 않을 때이다.

```
<img src="/guestbook/assets/images/1.jpg">
<img src="${pageContext.servletContext.contextPath }/assets/images/1.jpg">
```

17)JSTL 표기법으로 context 경로를 설정하면, context 경로를 변경했을 때 일일이 /guestbook 을 새로운 경로로 바꿔주는 수고를 덜 수 있다.
18)정리하면,

-<mvc:default-servlet-handler />(context 설정에 따라 <default-servlet-handler />) 설정을 추가하면, default servlet handler가 bean으로 등록되며, Spring MVC는 다음과 같이 동작한다.
i. 요청 URL에 mapping되는 controller를 검색한다.
-존재할 경우, controller를 이용해서 client 요청을 처리한다.
ii. Default servlet handler가 등록되어 있지 않다면, <-- "<mvc:default-servlet-handler />"를 써주지 않았다면
-404응답 error를 전송한다.
iii. Default servlet handler가 등록되어 있으면, default servlet handler에 요청을 전달한다.
-Default servlet handler는 WAS의 Default servlet 에 요청을 전달한다.
19)첨부.
-각 WAS는 servlet mapping에 존재하지 않는 요청을 처리하기 위한 default servlet을 제공하고 있다.
-예를 들어, controller @RequestMapping에 등록되지 않는 요청 또는 JSP에 대한 요청을 처리하는 것이 바로 default servlet이다.
-DispatcherServlet의 mapping URL pattern(web.xml에서 설정)을 "/"로 지정하면 JSP를 제외한 모든 요청이 DispathcerServlet으로 가기 때문에, WAS가 제공하는 default servlet이 *.html이나 *.css와 같은 요청을 처리할 수 없게 된다.
-Default servlet handler는 바로 이 default servlet에 요청을 전달해주는 handler로서, 요청 URL에 mapping되는 controller가 존재하지 않을 때 404 응답대신, default servlet이 해당 요청 URL을 처리하도록 한다.
-따라서, *.css와 같은 controller에 mapping되어 있지 않은 URL 요청은 최종적으로 3.A 과정을 통해 default servlet에 전달되어 처리된다.

18)config/mybatis-config.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>
  <typeAliases>
    <typeAlias type="com.example.vo.MemberVO" alias="memberVO"/>
  </typeAliases>
</configuration>
```

19)config/member-mapper.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="Member">

</mapper>
```

20)applicationContext.xml 아래 code 추가

```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
  <property name="dataSource" ref="dataSource" />
  <property name="configLocation" value="classpath:mybatis-config.xml" />
  <property name="mapperLocations">
    <list>
      <value>classpath:member-mapper.xml</value>
    </list>
```

```
    </property>
  </bean>
  <bean id="sqlSession" class="org.mybatis.spring.SqlSessionTemplate">
    <constructor-arg ref="sqlSessionFactory" />
  </bean>

21)전체 사용자 조회하기
  -HomeController 객체 code 추가

    @RequestMapping(value = "/members", method = RequestMethod.GET)
    @ResponseBody
    public Map members() {
      List<MemberVO> list = this.service.select();
      Map<String, Object> map = new HashMap<String, Object>();
      map.put("code", "success");
      map.put("data", list);
      return map;
    }

  -mybatis-mapper.xml

    <resultMap type="memberVO" id="selectMap">
      <result property="name" column="name"/>
      <result property="userid" column="userid"/>
      <result property="gender" column="gender" />
      <result property="city" column="city"/>
    </resultMap>

    <select id="select" resultMap="selectMap">
      SELECT * FROM Member
    </select>

  -MemberDaoImpl.java

    @Override
    public List<MemberVO> readAll() {
      return this.sqlSession.selectList("Member.select");
    }

  -MemberServiceImpl.java

    @Override
    public List<MemberVO> select() {
      return this.memberDao.readAll();
    }

  -Postman
    GET http://localhost:8080/RestfulDemo/members Send
    Body

      {
        "code": "success",
        "data": [
          {
```

```
759               "userId": "jimin",
760               "name": "한지민",
761               "gender": "여",
762               "city": "서울"
763             },
764             {
765               "userId": "example",
766               "name": "조용필",
767               "gender": "남성",
768               "city": "부산"
769             },
770             {
771               "userId": "javaexpert",
772               "name": "이미자",
773               "gender": "여성",
774               "city": "광주"
775             }
776           ]
777         }
778
779     -WebContent > right-click > New > Folder > js
780       -js/jquery-1.12.4.js
781     -WebContent/index.html
782
783       <!DOCTYPE html>
784       <html>
785         <head>
786           <meta charset="UTF-8">
787           <title>Welcome</title>
788           <script src="js/jquery-1.12.4.js"></script>
789           <script>
790             $(document).ready(function(){
791               $.ajax({
792                 url:"/RestfulDemo/members",
793                 type : "GET",
794                 dataType : "json",
795                 success : function(data){
796                   var str = "";
797                   var members = data.data;
798                   for(var i = 0 ; i < members.length ; i++){
799                     str += "<tr>";
800                     var userid = members[i].userid;
801                     str += "<td><a href='view.html?userid=" + userid + "'>" + userid +
                       "</a></td>" +
802                         "<td>" + members[i].name + "</td>" +
803                         "<td>" + members[i].gender + "</td>" +
804                         "<td>" + members[i].city + "</td>";
805                     str += "</tr>";
806                   }
807                   $("#result").html(str);
808                 }
809               });
810             });
811           </script>
```

```
812         </head>
813         <body>
814           <h1>Member List</h1>
815           <div style="text-align:center">
816             <a href="register.html">Member Add</a>
817           </div>
818           <table border="1">
819             <thead>
820               <tr>
821                 <th>아이디</th><th>이름</th>
822                 <th>성별</th><th>거주지</th>
823               </tr>
824             </thead>
825             <tbody id="result">
826             </tbody>
827           </table>
828         </body>
829       </html>
830
831     -index.html > right-click > Run As > Run on Server
832
833   22)특정 사용자 조회하기
834     -HomeController.java
835
836       @RequestMapping(value = "/members/{userid}", method = RequestMethod.GET)
837       @ResponseBody
838       public Map memberInfo(@PathVariable String userid) {
839         //System.out.println("userid = " + userid);
840         MemberVO member = this.service.selectMember(userid);
841         Map<String, Object> map = new HashMap<String, Object>();
842         map.put("code", "success");
843         map.put("data", member);
844         return map;
845       }
846
847     -mybatis-mapper.xml
848
849       <select id="selectMember" parameterType="String" resultType="memberVO">
850         SELECT * FROM Member WHERE userid = #{userid}
851       </select>
852
853     -MemberDaoImpl.java
854
855       @Override
856       public MemberVO read(String userid) {
857         return this.sqlSession.selectOne("Member.selectMember", userid);
858       }
859
860     -MemberServiceImpl.java
861
862       @Override
863       public MemberVO selectMember(String userid) {
864         return this.memberDao.read(userid);
865       }
```

-Postman
GET http://localhost:8080/RestfulDemo/members/jimin Send
Body

```
{
  "code": "success",
  "data": {
    "userId": "jimin",
    "name": "한지민",
    "gender": "여",
    "city": "서울"
  }
}
```

-WebContent/view.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>회원 정보 페이지</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script>
      var userid = null;

      $(function(){
        userid = location.search.substring(1).split("=")[1];
        $.ajax({
          url : "/RestfulDemo/members/" + userid,
          type : "GET",
          success : function(data){
            var member = data.data;
            $("#userid").text(member.userid);
            $("#name").text(member.name);
            $("#gender").text(member.gender);
            $("#city").text(member.city);
          }
        });
      });
    </script>
  </head>
  <body>
    <h1>Member Information</h1>
    <ul>
      <li>아이디 : <span id="userid"></span></li>
      <li>이름 : <span id="name"></span></li>
      <li>성별 : <span id="gender"></span></li>
      <li>거주지 : <span id="city"></span></li>
    </ul>
    <a href = "javascript:void(0)" onclick="javascript:history.back();">목록으로</a>
  </body>
</html>
```

```
  -view.html > right-click > Run As > Run on Server
  -view.html?userid=jimin or index.html에서 jimin link click

23)사용자 등록 구현하기
  -HomeController.java

    @RequestMapping(value = "/members", method = RequestMethod.POST)
    @ResponseBody
    public Map insert(@RequestBody MemberVO memberVO) {
      System.out.println(memberVO);
      this.service.insertMember(memberVO);
      Map<String, Object> map = new HashMap<String, Object>();
      map.put("code", "success");
      return map;
    }

  -mabatis-mapper.xml

    <insert id="insert" parameterType="memberVO">
      INSERT INTO Member(name,userid,gender,city)
      VALUES(#{name}, #{userid}, #{gender}, #{city})
    </insert>

  -MemberDaoImpl.java

    @Override
    public void create(MemberVO member) {
      this.sqlSession.insert("Member.insert", member);
    }

  -MemberServiceImpl.java

    @Override
    public void insertMember(MemberVO member) {
      this.memberDao.create(member);
    }

  -web.xml에 한글이 post방식으로 처리할 때 깨짐 방지 하기 위한 fileter 복사해서 넣기
  -Postman
    POST http://localhost:8080/RestfulDemo/members
    Body > raw > JSON(application/json)

      {
        "userid" : "girlsage",
        "name" : "소녀시대",
        "gender" : "여성",
        "city" : "수원"
      }

    Send 버튼 클릭하면

      Body
        {"code": "success"}
```

-WebContent/register.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Member Add</title>
    <script src="js/jquery-1.12.4.js"></script>
    <script>
      $(function(){
        $("input[type='button']").bind("click", function(){
          $.ajax({
            url : "/RestfulDemo/members",
            contentType : "application/json;charset=utf-8",
            type : "POST",
            data : JSON.stringify({
              "userid" : $("#userid").val(),
              "name" : $("#name").val(),
              "gender" : $(".gender:checked").val(),
              "city" : $("#city").val()
            }),
            dataType : "json",
            success : function(data){
              alert(data.code);
              location.href = "/RestfulDemo/";
            }
          });
        });
      });
    </script>
  </head>
  <body>
    <h1>Member Add</h1>
    <ul>
      <li>Name : <input type="text" id="name" /></li>
      <li>ID : <input type="text" id="userid" /></li>
      <li>Gender :
        <input type="radio" class="gender" name="gender" value="남성">남성
          
        <input type="radio" class="gender" name="gender" value="여성">여성
      </li>
      <li>City : <input type="text" id="city" /></li>
    </ul>
    <input type="button" value="가입하기" />
  </body>
</html>
```

-register.html > right-click > Run As > Run on Server

24)사용자 정보 수정 구현하기

-HomeController.java

```
@RequestMapping(value = "/members", method = RequestMethod.PUT)
@ResponseBody
```

```
1027        public Map update(@RequestBody MemberVO memberVO) {
1028          this.service.updateMember(memberVO);
1029          Map<String, Object> map = new HashMap<String, Object>();
1030          map.put("code", "success");
1031          return map;
1032        }
1033
1034      -mabatis-mapper.xml
1035
1036        <update id="update" parameterType="memberVO">
1037          UPDATE Member SET name = #{name}, gender = #{gender}, city = #{city}
1038          WHERE userid = #{userid}
1039        </update>
1040
1041      -MemberDaoImpl.java
1042
1043        @Override
1044        public void update(MemberVO member) {
1045          this.sqlSession.update("Member.update", member);
1046        }
1047
1048      -MemberServiceImpl.java
1049
1050        @Override
1051        public void updateMember(MemberVO member) {
1052          this.memberDao.update(member);
1053        }
1054
1055      -Postman
1056        PUT http://localhost:8080/RestfulDemo/members
1057        Body > raw > JSON(application/json)
1058
1059          {
1060            "userId" : "girlsage",
1061            "name" : "소년시대",
1062            "gender" : "남성",
1063            "city" : "부산"
1064          }
1065
1066        Send 버튼 클릭하면
1067
1068          Body
1069            {"code": "success"}
1070
1071      -WebContent/view.html 수정
1072        --아래 code를 추가한다.
1073          <a href = "javascript:void(0)" onclick="javascript:member_update()">수정하기</a>
1074
1075          var flag = false;
1076          function member_update(){
1077            if(!flag){   //수정하기를 click하면
1078              var name = $("#name").text();
1079              $("span#name")
1080              .replaceWith("<input type='text' id='name' value='" + name + "'/>");
```

```
1081              var gender = $("#gender").text();
1082              var str = null;
1083              if(gender == "남성"){
1084                str = "<input type='radio' class='gender' value='남성' checked/>남성  "
                    +
1085                  "<input type='radio' class='gender' value='여성' />여성";
1086              }else{
1087                str = "<input type='radio' class='gender' name='gender' value='남성' />남성
                      " +
1088                  "<input type='radio' class='gender' name='gender' value='여성' checked />여성";
1089              }
1090              $("span#gender").replaceWith(str);
1091              var city = $("#city").text();
1092              $("span#city")
1093              .replaceWith("<input type='text' id='city' value='" + city + "'/>");
1094              flag = true;
1095            }else{
1096              $.ajax({
1097                url : "/RestfulDemo/members",
1098                type : "PUT",
1099                data : JSON.stringify({
1100                  "userid" : userid,
1101                  "name" : $("#name").val(),
1102                  "gender" : $(".gender:checked").val(),
1103                  "city" : $("#city").val()
1104                }),
1105                contentType : "application/json;charset=utf-8",
1106                success : function(data){
1107                  alert(data.code);
1108                  location.reload();
1109                }
1110              });
1111              flag = false;
1112            }
1113          }
1114
1115
1116    25)사용자 정보 삭제 구현하기
1117      -HomeController.java
1118
1119        @RequestMapping(value = "/members/{userid}", method = RequestMethod.DELETE)
1120        @ResponseBody
1121        public Map delete(@PathVariable String userid) {
1122          this.service.deleteMember(userid);
1123          Map<String, Object> map = new HashMap<String, Object>();
1124          map.put("code", "success");
1125          return map;
1126        }
1127
1128      -mabatis-mapper.xml
1129
1130        <delete id="delete" parameterType="String">
1131          DELETE FROM Member WHERE userid = #{userid}
1132        </delete>
```

-MemberDaoImpl.java

```
@Override
public void delete(String userid) {
  this.sqlSession.delete("Member.delete", userid);
}
```

-MemberServiceImpl.java

```
@Override
public void deleteMember(String userid) {
  this.memberDao.delete(userid);
}
```

-Postman
DELETE http://localhost:8080/RestfulDemo/members/girlsage

Send button click하면

Body
{"code": "success"}

-WebContent/view.html 수정
--아래의 code를 추가한다.

```
<a href = "javascript:void(0)" onclick="javascript:member_delete()">삭제하기</a>

function member_delete(){
  $.ajax({
    url : "/RestfulDemo/members/" + userid,
    type : "DELETE",
    success : function(data){
      alert(data.code);
      location.href = "/RestfulDemo/";
    }
  });
}
```

7. data 변환 - XML로 변환
1)Maven Repository에서 'spring xml'로 검색
2)Spring Object/XML Marshalling에서 4.3.24.RELEASE 선택
3)pom.xml에 아래 dependency 추가 > Maven Clean > Mavan Install

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-oxm</artifactId>
  <version>4.3.24.RELEASE</version>
</dependency>
```

4)Maven Repository에서 'jaxb'로 검색, Jaxb Api에서 2.3.0
5)아래의 dependency를 pom.xml에 추가 > Maven Clean > Mavan Install

```
1187      <dependency>
1188        <groupId>javax.xml.bind</groupId>
1189        <artifactId>jaxb-api</artifactId>
1190        <version>2.3.1</version>
1191      </dependency>
1192
1193    6)src/com.example.vo/UserListVO.java 생성
1194
1195      package com.example.vo;
1196
1197      import java.util.List;
1198
1199      import javax.xml.bind.annotation.XmlAccessType;
1200      import javax.xml.bind.annotation.XmlAccessorType;
1201      import javax.xml.bind.annotation.XmlElement;
1202      import javax.xml.bind.annotation.XmlRootElement;
1203
1204      import org.springframework.stereotype.Component;
1205
1206      @XmlRootElement(name="userList")
1207      @XmlAccessorType(XmlAccessType.FIELD)
1208      @Component
1209      public class UserListVO {
1210        @XmlElement(name="user")
1211        private List<UserVO> userList;
1212
1213        public List<UserVO> getUserList() {
1214          return userList;
1215        }
1216
1217        public void setUserList(List<UserVO> userList) {
1218          this.userList = userList;
1219        }
1220      }
1221
1222      -XML 문서는 반드시 단 하나의 root element를 가져야 한다.
1223      -여러 UserVO를 표현하려면 root element로 사용할 또 다른 Java class가 필요하다.
1224      -새로 생성한 UserListVO객체는 이 객체가 root element에 해당하는 객체이며, root element 이름을 userList
          로 설정하겠다는 의미로 @XmlRootElement(name="userList") 설정을 추가했다.
1225      -그리고 userList 변수 위에도 @XmlElement(name="user")를 추가했는데, UserVO 마다 element 이름을
          user로 변경할 것이다.
1226
1227    7)src/com.example.vo.MemberVO.java 수정
1228
1229      package com.example.vo;
1230
1231      import javax.xml.bind.annotation.XmlAccessType;
1232      mport javax.xml.bind.annotation.XmlAccessorType;
1233      import javax.xml.bind.annotation.XmlAttribute;
1234      import javax.xml.bind.annotation.XmlRootElement;
1235
1236      import org.springframework.stereotype.Component;
1237
1238      @XmlRootElement(name="user")
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@Component
public class UserVO {
  @XmlAttribute
  private String userId;
```

-VO class에 선언된 @XmlAccessorType은 UserVO 객체를 XML로 변환할 수 있다는 의미이다.
-그리고 XmlAccessType.FIELD 때문에 이 객체가 가진 field, 즉 변수들은 자동으로 자식 element로 표현된다.
-하지만, 이 중에서 userId에만 @XmlAttribute가 붙었는데, 이는 userId를 속성으로 표현하라는 의미이다.
-만일 JSON 변환시 @JsonIgnore가 변환시 제외하는 것처럼, XML변환시에도 제외할 변수는 @XmlTransient를 붙이면 된다.
-마지막으로 변환시 변수가 참조형이면 반드시 기본 생성자가 있어야만 한다.

8)Spring 설정 file에서 p와 oxm check후, 아래 code 추가
-JSON 변환시 Java 객체를 JSON response body로 변환해주는 MappingJackson2HttpMessageConverter를 Spring 설정 file에 추가해야 하는데, 설정 file에 <mvc:annotation-driven />으로 대체했었다.
-마찬가지로 Java 객체를 XML response body로 변환할 때는 아래의 code를 추가한다.

```
<bean id="xmlViewer" class="org.springframework.web.servlet.view.xml.MarshallingView">
  <constructor-arg>
    <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller"
    p:classesToBeBound="com.example.vo.UserListVO"/>
  </constructor-arg>
</bean>
```

9)UserController.java code 추가

```
@RequestMapping(value="/userlist.do", produces="application/xml")
@ResponseBody
public UserListVO userlist(){
  UserListVO listVO = new UserListVO();
  listVO.setUserList(this.userService.getUserList());
  return listVO;
}
```

10)실행결과

```
<userList>
  <user userId="jimin">
    <name>한지민</name>
    <gender>여</gender>
    <city>서울</city>
  </user>
</userList>
```