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 웹 서비스와 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 -->
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-annotations</artifactId>
      <version>2.9.9</version>
    </dependency>

7)web.xml의 DispatcherServlet url-pattern 변경
  --기존--
  <servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>*.do</url-pattern>
  </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 받
          을 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/fhbjgbiflinjbdggehcddcbncdddomop130
153     2)[앱실행]버튼 클릭
154     3)Log in
155
156
157 5. Data 변환 - JSON으로 변환
158     1)System이 복잡해지면서 다른 system과 정보를 주고받을 일이 발생하는데, 이 때 data 교환 format으로
        JSON을 사용할 수 있다.
159     2)검색결과를 JSON data로 변환하려면 가장 먼저 jackson2 library를 download 받아야 한다.
160     3)Jackson2는 Java 객체를 JSON으로 변환하거나 JSON을 Java 객체로 변환해주는 library다.
161     4)https://www.concretepage.com/spring-4/spring-4-rest-xml-response-example-with-jackson-2
        참조
162     5)https://www.mkyong.com/java/jackson-2-convert-java-object-to-from-json/ 참조
163     6)pom.xml에 다음과 같이 dependency를 추가한다.
164       <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
165       <dependency>
166         <groupId>com.fasterxml.jackson.core</groupId>
167         <artifactId>jackson-databind</artifactId>
168         <version>2.9.9</version>
169       </dependency>
170       <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
171       <dependency>
172         <groupId>com.fasterxml.jackson.core</groupId>
173         <artifactId>jackson-core</artifactId>
174         <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 Tools > 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>
```

```
289           <artifactId>spring-jdbc</artifactId>
290           <version>4.3.24.RELEASE</version>
291         </dependency>
292         <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
293         <dependency>
294           <groupId>org.springframework</groupId>
295           <artifactId>spring-webmvc</artifactId>
296           <version>4.3.24.RELEASE</version>
297         </dependency>
298         <dependency>
299           <groupId>javax.servlet</groupId>
300           <artifactId>jstl</artifactId>
301           <version>1.2</version>
302         </dependency>
303         <dependency>
304           <groupId>com.oracle</groupId>
305           <artifactId>ojdbc8</artifactId>
306           <version>12.2</version>
307         </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     7)Build path에 config foler 추가
341       -project right-click > Build Path > Configure Build Path > Select [Source] tab
342       -Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
343       -Folder name : config > Finish > OK > Apply and Close
344       -Java Resources > config folder 확인
345
346     8)config folder에 applicationContext.xml file 생성
347       -Spring Perspective로 전환
348       -config right-click > New > Spring Bean Configuration File
349       -File name :  applicationContext.xm
```

```
-생성시 beans,context, mvc check
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:context="http://www.springframework.org/schema/context"
  xmlns:mvc="http://www.springframework.org/schema/mvc"
  xsi:schemaLocation="http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc-4.3.xsd
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.3.xsd">

</beans>

9)ContextLoaderListener class 설정
  -Business logic의 Spring 설정 file (ex:applicationContext.xml)을 작성했기 때문에 listener로
  ContextLoaderListener class를 정의해야 한다.
  -ContextLoaderListener class는 Spring 설정 file(default에서 file명 applicationContext.xml)을 load
  하면 ServletContextListener interface를 구현하고 있기 때문에 ServletContext instance 생성시
  (Tomcat으로 application이 load된 때)에 호출된다.
  -즉, ContextLoaderListener class는 DispatcherServlet class의 load보다 먼저 동작하여 business
  logic층을 정의한 Spring 설정 file을 load한다.
  -web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener
  -ContextLoaderListener] 를 선택하면 아래의 code가 자동 삽입
    <!-- needed for ContextLoaderListener -->
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>location</param-value>
    </context-param>

    <!-- Bootstraps the root web application context before servlet initialization -->
    <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

  -아래 code로 변환
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:applicationContext.xml</param-value>
    </context-param>

10)DispatcherServlet Class 추가
  -web.xml에서 Ctrl + Spacebar 하면 나타나는 Context Menu에서 [#dispatcherservlet
  -DispatcherServlet declaration] 선택하면 아래의 code가 자동 추가된다.

  <!-- The front controller of this Spring Web application, responsible for handling all application
  requests -->
  <servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>location</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <!-- Map all requests to the DispatcherServlet for handling -->
  <servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
```

```
401         <url-pattern>url</url-pattern>
402       </servlet-mapping>
403
404       -아래의 code로 변환
405       <servlet>
406         <servlet-name>springDispatcherServlet</servlet-name>
407         <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
408         <init-param>
409           <param-name>contextConfigLocation</param-name>
410           <param-value>classpath:beans.xml</param-value>
411         </init-param>
412         <load-on-startup>1</load-on-startup>
413       </servlet>
414
415       <servlet-mapping>
416         <servlet-name>springDispatcherServlet</servlet-name>
417         <url-pattern>/</url-pattern>
418       </servlet-mapping>
419
420     11)MemberVO class 생성
421       -src/com.example.vo package 생성
422       -src/com.example.vo.MemberVO class 생성
423
424         package com.example.vo;
425
426         public class MemberVO {
427           private String name, userid, gender, city;
428           public MemberVO() {}
429           public MemberVO(String name, String userid, String gender, String city) {
430             this.name = name;
431             this.userid = userid;
432             this.gender = gender;
433             this.city = city;
434           }
435           public String getName() {
436             return name;
437           }
438           public void setName(String name) {
439             this.name = name;
440           }
441           public String getUserid() {
442             return userid;
443           }
444           public void setUserid(String userid) {
445             this.userid = userid;
446           }
447           public String getGender() {
448             return gender;
449           }
450           public void setGender(String gender) {
451             this.gender = gender;
452           }
453           public String getCity() {
454             return city;
455           }
456           public void setCity(String city) {
457             this.city = city;
458           }
459           @Override
460           public String toString() {
461             return "MemberVO [name=" + name + ", userid=" + userid + ", gender=" + gender + ",
```

```
            city=" + city + "]";
462       }
463     }
464
465  12)MemberDao 객체 생성
466    -src/com.example.dao package 생성
467    -src/com.example.dao.MemberDao interface
468
469      package com.example.dao;
470
471      import java.util.List;
472
473      import com.example.vo.MemberVO;
474
475      public interface MemberDao {
476        void create(MemberVO member);
477        List<MemberVO> readAll();
478        MemberVO read(String userid);
479        void update(MemberVO member);
480        void delete(String userid);
481      }
482
483    -src/com.example.dao.MemberDaoImpl.java 생성
484
485      package com.example.dao;
486
487      import java.util.List;
488
489      import org.springframework.beans.factory.annotation.Autowired;
490      import org.springframework.stereotype.Repository;
491      import org.apache.ibatis.session.SqlSession;
492
493      import com.example.vo.MemberVO;
494
495      @Repository("memberDao")
496      public class MemberDaoImpl implements MemberDao {
497        @Autowired
498        private SqlSession sqlSession;
499
500        @Override
501        public void create(MemberVO member) {
502          this.sqlSession.insert("Member.insert", member);
503        }
504
505        @Override
506        public List<MemberVO> readAll() {
507          return this.sqlSession.selectList("Member.select");
508        }
509
510        @Override
511        public MemberVO read(String userid) {
512          return this.sqlSession.selectOne("Member.selectMember", userid);
513        }
514
515        @Override
516        public void update(MemberVO member) {
517          this.sqlSession.update("Member.update", member);
518        }
519
520        @Override
521        public void delete(String userid) {
```

```
522             this.sqlSession.delete("Member.delete", userid);
523           }
524         }
525
526
527     13)MemberService 객체 생성
528       -src/com.example.service package 생성
529       -src/com.example.service.MemberService interface
530
531         package com.example.service;
532
533         import java.util.List;
534
535         import com.example.vo.MemberVO;
536
537         public interface MemberService {
538           void insertMember(MemberVO member);
539           List<MemberVO> select();
540           MemberVO selectMember(String userid);
541           void updateMember(MemberVO member);
542           void deleteMember(String userid);
543         }
544
545       -src/com.example.service.MemberServiceImpl.java
546
547         package com.example.service;
548
549         import java.util.List;
550
551         import org.springframework.beans.factory.annotation.Autowired;
552         import org.springframework.stereotype.Service;
553
554         import com.example.dao.MemberDao;
555         import com.example.vo.MemberVO;
556
557         @Service("memberService")
558         public class MemberServiceImpl implements MemberService {
559           @Autowired
560           private MemberDao memberDao;
561
562           @Override
563           public void insertMember(MemberVO member) {
564             this.memberDao.create(member);
565           }
566
567           @Override
568           public List<MemberVO> select() {
569             return this.memberDao.readAll();
570           }
571
572           @Override
573           public MemberVO selectMember(String userid) {
574             return this.memberDao.read(userid);
575           }
576
577           @Override
578           public void updateMember(MemberVO member) {
579             this.memberDao.update(member);
580           }
581
582           @Override
```

```
583         public void deleteMember(String userid) {
584           this.memberDao.delete(userid);
585         }
586       }
587
588   14)HomeController 객체 생성
589     -src/com.example.controller package 생성
590     -com.example.controller.HomeController class 생성
591
592       package com.example.controller;
593
594       import org.springframework.beans.factory.annotation.Autowired;
595       import org.springframework.stereotype.Controller;
596       import org.springframework.ui.Model;
597       import org.springframework.web.bind.annotation.RequestMapping;
598       import org.springframework.web.bind.annotation.RequestParam;
599
600       import com.example.service.MemberService;
601       import com.example.vo.MemberVO;
602
603       @Controller
604       public class HomeController {
605         @Autowired
606         private MemberService service;
607
608       }
609
610   15)config/dbinfo.properties file 생성
611
612     db.driver=oracle.jdbc.driver.OracleDriver
613     db.url=jdbc:oracle:thin:@192.168.56.3:1521:ORCL
614     db.username=scott
615     db.password=tiger
616
617   16)applicationContext.xml
618
619     <context:component-scan base-package="com.example" />
620       <context:property-placeholder location="classpath:dbinfo.properties"/>
621       <bean id="dataSource"
       class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
622         <property name="driverClass" value="${db.driver}"/>
623         <property name="url" value="${db.url}"/>
624         <property name="username" value="${db.username}"/>
625         <property name="password" value="${db.password}"/>
626       </bean>
627
628   17)beans.xml
629       <mvc:annotation-driven />
630       <mvc:default-servlet-handler/>
631
632   ※<mvc:default-servlet-handler/>
633   1)Tomcat은 client의 요청 URL을 보고 Servlet Mapping에 따라 URL에 mapping된 Servlet이 처리를 하는
       구조이다.
634   2)그리고 URL에 mapping되는 Servlet이 없다면, 예를들어 CSS, image file 같은 정적자원들은
       defaultSevlet이 처리하도록 되어 있다.
635   3)즉 CSS , image file들은 server 외부에서 직접 접근 할 수 없는 /WEB-INF/assets folder 아래에 위치하
       는 것이 일반적인데, CSS , image file에 접근하기 위한 Servlet Mapping을 하지 않았으면 Tomcat이
       defaultServlet으로 처리하여 정적 file에 접근한다.
636   4)일반적으로 정적 file에 대해 Servlet Mapping을 하지 않는다는 것이다.
637   5)Spring에서는 DispatcherServlet이 모든 요청을 받아 들인 후 Handler mapping table에 따라 controller
       로 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.memberService.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

```
742         @Override
743         public List<MemberVO> select() {
744           return this.memberDao.readAll();
745         }
746
747       -Postman
748         GET http://localhost:8080/RestfulDemo/members Send
749         Body
750
751           {
752             "code": "success",
753             "data": [
754               {
755                 "userId": "jimin",\
756                 "name": "한지민",
757                 "gender": "여",
758                 "city": "서울"
759               },
760               {
761                 "userId": "example",
762                 "name": "조용필",
763                 "gender": "남성",
764                 "city": "부산"
765               },
766               {
767                 "userId": "javaexpert",
768                 "name": "이미자",
769                 "gender": "여성",
770                 "city": "광주"
771               }
772             ]
773           }
774
775       -WebContent/index.html
776
777         <!DOCTYPE html>
778         <html>
779           <head>
780             <meta charset="UTF-8">
781             <title>Welcome</title>
782             <script src="js/jquery-1.12.4.js"></script>
783             <script>
784               $(document).ready(function(){
785                 $.ajax({
786                   url:"/RestfulDemo/members",
787                   type : "GET",
788                   dataType : "json",
789                   success : function(data){
790                     var str = "";
791                     var members = data.data;
792                     for(var i = 0 ; i < members.length ; i++){
793                       str += "<tr>";
794                       var userid = members[i].userid;
795                       str += "<td><a href='view.html?userid=" + userid + "'>" + userid +
                          "</a></td>" +
796                           "<td>" + members[i].name + "</td>" +
797                           "<td>" + members[i].gender + "</td>" +
798                           "<td>" + members[i].city + "</td>";
799                       str += "</tr>";
800                     }
801                     $("#result").html(str);
```

```
                }
              });
            });
          </script>
        </head>
        <body>
          <h1>Member List</h1>
          <div style="text-align:center">
            <a href="register.html">Member Add</a>
          </div>
          <table border="1">
            <thead>
              <tr>
                <th>아이디</th><th>이름</th>
                <th>성별</th><th>거주지</th>
              </tr>
            </thead>
            <tbody id="result">
            </tbody>
          </table>
        </body>
      </html>

22)특정 사용자 조회하기
  -HomeController.java

    @RequestMapping(value = "/members/{userid}", method = RequestMethod.GET)
    @ResponseBody
    public Map memberInfo(@PathVariable String userid) {
      //System.out.println("userid = " + userid);
      MemberVO member = this.memberService.selectMember(userid);
      Map<String, Object> map = new HashMap<String, Object>();
      map.put("code", "success");
      map.put("data", member);
      return map;
    }

  -mybatis-mapper.xml

    <select id="selectMember" parameterType="String" resultType="memberVO">
      SELECT * FROM Member WHERE userid = #{userid}
    </select>

  -MemberDaoImpl.java

    @Override
    public MemberVO read(String userid) {
      return this.sqlSession.selectOne("Member.selectMember", userid);
    }

  -MemberServiceImpl.java

    @Override
    public MemberVO selectMember(String userid) {
      return this.memberDao.read(userid);
    }

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

```
        {
          "code": "success",
          "data": {
            "userId": "example",
            "name": "조용필",
            "gender": "남성",
            "city": "부산"
          }
        }

    -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>


  23)사용자 등록 구현하기
    -HomeController.java

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

```
924         }
925
926       -mabatis-mapper.xml
927
928         <insert id="insert" parameterType="memberVO">
929           INSERT INTO Member(name,userid,gender,city)
930           VALUES(#{name}, #{userid}, #{gender}, #{city})
931         </insert>
932
933       -MemberDaoImpl.java
934
935         @Override
936         public void create(MemberVO member) {
937           this.sqlSession.insert("Member.insert", member);
938         }
939
940       -MemberServiceImpl.java
941
942         @Override
943         public void insertMember(MemberVO member) {
944           this.memberDao.create(member);
945         }
946
947       -Postman
948         POST http://localhost:8080/RestfulDemo/members
949         Body
950           raw
951
952           {
953             "userId" : "girlsage",
954             "name" : "소녀시대",
955             "gender" : "여성",
956             "city" : "수원"
957           }
958
959         Send 버튼 클릭하면
960
961           Body
962             {"code": "success"}
963
964       -WebContent/register.html
965
966         <!DOCTYPE html>
967         <html>
968           <head>
969             <meta charset="UTF-8">
970             <title>Member Add</title>
971             <script src="js/jquery-1.12.4.js"></script>
972             <script>
973               $(function(){
974                 $("input[type='button']").bind("click", function(){
975                   $.ajax({
976                     url : "/RestfulDemo/members",
977                     contentType : "application/json;charset=utf-8",
978                     type : "POST",
979                     data : JSON.stringify({
980                       "userid" : $("#userid").val(),
981                       "name" : $("#name").val(),
982                       "gender" : $(".gender:checked").val(),
983                       "city" : $("#city").val()
984                     }),
```

```
                    dataType : "json",
                    success : function(data){
                      alert(data.code);
                      location.href = "/0605/";
                    }
                  });
                });
              });
            </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>

24)사용자 정보 수정 구현하기
  -HomeController.java

    @RequestMapping(value = "/members", method = RequestMethod.PUT)
    @ResponseBody
    public Map update(@RequestBody MemberVO memberVO) {
      this.memberService.updateMember(memberVO);
      Map<String, Object> map = new HashMap<String, Object>();
      map.put("code", "success");
      return map;
    }

  -mabatis-mapper.xml

    <update id="update" parameterType="memberVO">
      UPDATE Member SET name = #{name}, gender = #{gender}, city = #{city}
      WHERE userid = #{userid}
    </update>

  -MemberDaoImpl.java

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

  -MemberServiceImpl.java

    @Override
    public void updateMember(MemberVO member) {
      this.memberDao.update(member);
    }

  -Postman
    PUT http://localhost:8080/RestfulDemo/members
    Body
```

```
1046          raw
1047
1048          {
1049            "userId" : "girlsage",
1050            "name" : "소년시대",
1051            "gender" : "남성",
1052            "city" : "부산"
1053          }
1054
1055        Send 버튼 클릭하면
1056
1057          Body
1058            {"code": "success"}
1059
1060      -WebContent/view.html 수정
1061        --아래 code를 추가한다.
1062          <a href = "javascript:void(0)" onclick="javascript:member_update()">수정하기</a>
1063
1064          var flag = false;
1065          function member_update(){
1066            if(!flag){   //수정하기를 click하면
1067              var name = $("#name").text();
1068              $("span#name")
1069              .replaceWith("<input type='text' id='name' value='" + name + "'/>");
1070              var gender = $("#gender").text();
1071              var str = null;
1072              if(gender == "남성"){
1073                str = "<input type='radio' class='gender' value='남성' checked/>남성  "
                   +
1074                 "<input type='radio' class='gender' value='여성' />여성";
1075              }else{
1076                str = "<input type='radio' class='gender' name='gender' value='남성' />남성
                     " +
1077                 "<input type='radio' class='gender' name='gender' value='여성' checked />여성";
1078              }
1079              $("span#gender").replaceWith(str);
1080              var city = $("#city").text();
1081              $("span#city")
1082              .replaceWith("<input type='text' id='city' value='" + city + "'/>");
1083              flag = true;
1084            }else{
1085              $.ajax({
1086                url : "/RestfulDemo/members",
1087                type : "PUT",
1088                data : JSON.stringify({
1089                  "userid" : userid,
1090                  "name" : $("#name").val(),
1091                  "gender" : $(".gender:checked").val(),
1092                  "city" : $("#city").val()
1093                }),
1094                contentType : "application/json;charset=utf-8",
1095                success : function(data){
1096                  alert(data.code);
1097                  location.reload();
1098                }
1099              });
1100              flag = false;
1101            }
1102          }
1103
1104
```

25)사용자 정보 삭제 구현하기
-HomeController.java

```
@RequestMapping(value = "/members/{userid}", method = RequestMethod.DELETE)
@ResponseBody
public Map delete(@PathVariable String userid) {
  this.memberService.deleteMember(userid);
  Map<String, Object> map = new HashMap<String, Object>();
  map.put("code", "success");
  return map;
}
```

-mabatis-mapper.xml

```
<delete id="delete" parameterType="String">
  DELETE FROM Member WHERE userid = #{userid}
</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 : "/0605/members/" + userid,
    type : "DELETE",
    success : function(data){
      alert(data.code);
      location.href = "/0605/";
    }
  });
}
```

20. 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

```
1166
1167      <dependency>
1168        <groupId>org.springframework</groupId>
1169        <artifactId>spring-oxm</artifactId>
1170        <version>4.3.24.RELEASE</version>
1171      </dependency>
1172
1173    4)Maven Repository에서 'jaxb'로 검색, Jaxb Api에서 2.3.0
1174    5)아래의 dependency를 pom.xml에 추가 > Maven Clean > Mavan Install
1175
1176      <dependency>
1177        <groupId>javax.xml.bind</groupId>
1178        <artifactId>jaxb-api</artifactId>
1179        <version>2.3.1</version>
1180      </dependency>
1181
1182    6)src/com.example.vo/UserListVO.java 생성
1183
1184      package com.example.vo;
1185
1186      import java.util.List;
1187
1188      import javax.xml.bind.annotation.XmlAccessType;
1189      import javax.xml.bind.annotation.XmlAccessorType;
1190      import javax.xml.bind.annotation.XmlElement;
1191      import javax.xml.bind.annotation.XmlRootElement;
1192
1193      import org.springframework.stereotype.Component;
1194
1195      @XmlRootElement(name="userList")
1196      @XmlAccessorType(XmlAccessType.FIELD)
1197      @Component
1198      public class UserListVO {
1199        @XmlElement(name="user")
1200        private List<UserVO> userList;
1201
1202        public List<UserVO> getUserList() {
1203          return userList;
1204        }
1205
1206        public void setUserList(List<UserVO> userList) {
1207          this.userList = userList;
1208        }
1209      }
1210
1211      -XML 문서는 반드시 단 하나의 root element를 가져야 한다.
1212      -여러 UserVO를 표현하려면 root element로 사용할 또 다른 Java class가 필요하다.
1213      -새로 생성한 UserListVO객체는 이 객체가 root element에 해당하는 객체이며, root element 이름을
          userList로 설정하겠다는 의미로 @XmlRootElement(name="userList") 설정을 추가했다.
1214      -그리고 userList 변수 위에도 @XmlElement(name="user")를 추가했는데, UserVO 마다 element 이름
          을 user로 변경할 것이다.
1215
1216    7)src/com.example.vo.MemberVO.java 수정
1217
1218      package com.example.vo;
1219
1220      import javax.xml.bind.annotation.XmlAccessType;
1221      mport javax.xml.bind.annotation.XmlAccessorType;
1222      import javax.xml.bind.annotation.XmlAttribute;
1223      import javax.xml.bind.annotation.XmlRootElement;
1224
```

```
1225    import org.springframework.stereotype.Component;
1226
1227    @XmlRootElement(name="user")
1228    @XmlAccessorType(XmlAccessType.FIELD)
1229    @Component
1230    public class UserVO {
1231      @XmlAttribute
1232      private String userId;
1233
1234    -VO class에 선언된 @XmlAccessorType은 UserVO 객체를 XML로 변환할 수 있다는 의미이다.
1235    -그리고 XmlAccessType.FIELD 때문에 이 객체가 가진 field, 즉 변수들은 자동으로 자식 element로 표현
        된다.
1236    -하지만, 이 중에서 userId에만 @XmlAttribute가 붙었는데, 이는 userId를 속성으로 표현하라는 의미이
        다.
1237    -만일 JSON 변환시 @JsonIgnore가 변환시 제외하는 것처럼, XML변환시에도 제외할 변수는
        @XmlTransient를 붙이면 된다.
1238    -마지막으로 변환시 변수가 참조형이면 반드시 기본 생성자가 있어야만 한다.
1239
1240  8)Spring 설정 file에서 p와 oxm check후, 아래 code 추가
1241    -JSON 변환시 Java 객체를 JSON response body로 변환해주는
        MappingJackson2HttpMessageConverter를 Spring 설정 file에 추가해야 하는데, 설정 file에
        <mvc:annotation-driven />으로 대체했었다.
1242    -마찬가지로 Java 객체를 XML response body로 변환할 때는 아래의 code를 추가한다.
1243
1244    <bean id="xmlViewer" class="org.springframework.web.servlet.view.xml.MarshallingView">
1245      <constructor-arg>
1246        <bean class="org.springframework.oxm.jaxb.Jaxb2Marshaller"
          p:classesToBeBound="com.example.vo.UserListVO"/>
1247      </constructor-arg>
1248    </bean>
1249
1250  9)UserController.java code 추가
1251
1252    @RequestMapping(value="/userlist.do", produces="application/xml")
1253    @ResponseBody
1254    public UserListVO userlist(){
1255      UserListVO listVO = new UserListVO();
1256      listVO.setUserList(this.userService.getUserList());
1257      return listVO;
1258    }
1259
1260  10)실행결과
1261    <userList>
1262      <user userId="jimin">
1263        <name>한지민</name>
1264        <gender>여</gender>
1265        <city>서울</city>
1266      </user>
1267    </userList>
```