1. Open API?
   1)개방형 API
   2)프로그래밍에서 사용할 수 있는 개방되어 있는 상태의 interface를 말한다.
   3)Portal이나 통계청, 기상청 등과 같은 관공서에서 가지고 있는 데이터를 외부 응용 프로그램에서 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 웹 서비스 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 아키텍처
      -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 웹 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 메소드를 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 포맷
      -JavaScript에서 객체를 만들 때 사용하는 표현식을 의미
      -JSON 표현식은 사람과 기계 모두 이해하기 쉬우며 용량이 작아서, 최근에는 XML을 대체해서 데이터 전송등에 많이 사용된다.
      -특정 언어에 종속되지 않으며, 대부분의 프로그래밍 언어에서 JSON 포맷의 데이터를 핸들링할 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
-데이터의 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' 2.9.5 버전을 pom.xml에 추가
-'Jackson Core' 2.9.5 버전을 pom.xml에 추가
-'Jackson Annotations' 2.9.5 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>2.9.5</version>
   </dependency>
   <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core
   -->
   <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-core</artifactId>
      <version>2.9.5</version>
   </dependency>
   <!--
   https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
    -->
   <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-annotations</artifactId>
```

```
            <version>2.9.5</version>
        </dependency>

    7)web.xml의 DispatcherServlet url-pattern 변경
      --기존--
      <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>*.do</url-pattern>
      </servlet-mapping>

      --변경--
      <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
      </servlet-mapping>

    8)Spring Bean Configuration File(beans.xml) 설정
      -Spring MVC에 필요한 Bean들을 자동으로 등록해주는 Tag
        <mvc:annotation-driven />

    9)Spring MVC기반 RESTful 웹 서비스 구현 절차
      -RESTful 웹서비스를 처리할 RestfulController 클래스 작성 및 Spring Bean으로 등록
      -요청을 처리할 메소드에 @RequestMapping @RequestBody와 @ResponseBody annotation 선언
      -REST Client Tool(Postman)을 사용하여 각각의 메소드 테스트
      -Ajax 통신을 하여 RESTful 웹서비스를 호출하는 HTML 페이지 작성

    10)사용자 관리 RESTful 웹서비스 URI와 Method
      Action Resource URI HTTP Method
      -사용자 목록 /users GET
      -사용자 보기 /users/{id} GET
      -사용자 등록 /users POST
      -사용자 수정 /users PUT
      -사용자 삭제 /user/{id} DELETE

    11)RESTful Controller를 위한 핵심 Annotation
      -Spring MVC에서는 Client에서 전송한 XML이나 JSON 데이터를 Controller에서 Java 객체로 변환해서 받을 수 있는 기능(수신)을 제공하고 있다.
      -Java객체를 XML이나 JSON으로 변환해서 전송할 수 있는 기능(송신)을 제공하고 있다.

      -Annotation 설명
        --@RequestBody : HTTP Request Body(요청 몸체)를 Java객체로 전달받을 수 있다.
        --@ResponseBody : Java객체를 HTTP Response Body(응답 몸체)로 전송할 수 있다.


4. Google Postman 설치
    1) https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcddcbncdddomop130
    2)[앱실행]버튼 클릭
    3)Log in


5. 데이터 변환 - JSON으로 변환
    1)시스템이 복잡해지면서 다른 시스템과 정보를 주고받을 일이 발생하는데, 이 때 데이터 교환 포맷으로 JSON을 사용할 수 있다.
    2)검색결과를 JSON 데이터로 변환하려면 가장 먼저 jackson2 라이브러리를 다운받아야 한다.
    3)Jackson2는 자바 객체를 JSON으로 변환하거나 JSON을 자바 객체로 변환해주는 라이브러리다.
    4) https://www.concretepage.com/spring-4/spring-4-rest-xml-response-example-with-jackso
```

n-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>2.9.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core -->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-core</artifactId>
    <version>2.9.5</version>
</dependency>
<!--
https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
-->
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-annotations</artifactId>
    <version>2.9.5</version>
</dependency>
```

7)Maven clean > Maven Install하면 Maven Dependencies에 아래와 같은 jar파일이 추가된다.
-jackson-databind-2.9.5.jar
-jackson-annotations-2.9.5.jar
-jackson-core-2.9.5.jar

8)보통 User가 Servlet이나 JSP를 요청하면 서버는 요청한 파일을 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 클래스는 여러가지가 있으며, 이 중에서 Java 객체를 JSON responsebody로 변환할 때는 MappingJackson2HttpMessageConverter를 사용한다.
14)따라서 MappingJackson2HttpMessageConverter를 Spring 설정 파일에 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.javasoft.controller.UserController.java에 다음과 같이 수정한다.

```
package com.javasoft.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.javasoft.service.UserService;
import com.javasoft.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 메소드와 달리 @ResponseBody라는 annotation을 추가했는데, Java 객체를 Http Response 프로토콜의 body로 변환하기 위해 사용된다.

18)이미 Spring Configuration File에 <mvc:annotation-driven>을 추가했기 때문에 @ResponseBody가 적용된 메소드의 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.javasoft.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.xml파일에 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>4.3.17.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>4.3.17.RELEASE</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc -->
    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-webmvc</artifactId>
        <version>4.3.17.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>
    <!--
    https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-databind -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.9.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-core
    -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.9.0</version>
    </dependency>
    <!--
    https://mvnrepository.com/artifact/com.fasterxml.jackson.core/jackson-annotations
     -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.9.0</version>
```

```
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis-spring</artifactId>
            <version>1.3.2</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
        <dependency>
            <groupId>org.mybatis</groupId>
            <artifactId>mybatis</artifactId>
            <version>3.4.6</version>
        </dependency>
    </dependencies>

7)Build path에 config foler 추가
    -project right-click > Build Path > Configure Build Path > Select [Source] tab
    -Click [Add Folder] > Select 현재 project > Click [Create New Folder...]
    -Folder name : config > Finish > OK > Apply and Close
    -Java Resources > config 폴더 확인

8)config folder에 applicationContext.xml 파일 생성
    -Spring Perspective로 전환
    -config right-click > New > Spring Bean Configuration File
    -File name :  applicationContext.xm
    -생성시 beans,context, mvc 체크
    <?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
    -비즈니스 로직용의 스프링 설정 파일 (ex:applicationContext.xml)을 작성했기 때문에 listener로
    ContextLoaderListener 클래스를 정의해야 한다.
    -ContextLoaderListener 클래스는 스프링 설정 파일(디폴트에서 파일명
    applicationContext.xml)을 로드하면 ServletContextListener 인터페이스를 구현하고 있기
    때문에 ServletContext 인스턴스 생성시(톰켓으로 어플리케이션이 로드된 때)에 호출된다.
    -즉, ContextLoaderListener 클래스는 DispatcherServlet 클래스의 로드보다 먼저 동작하여
    비즈니스 로직층을 정의한 스프링 설정 파일을 로드한다.
    -web.xml에서 Ctrl + Spacebar를 하면 나타나는 Context Menu에서 [#contextloaderlistener
    -ContextLoaderListener] 를 선택하면 아래의 코드가 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</listene
```

```
            r-class>
        </listener>

    -아래 코드로 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 코드가 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>
        <url-pattern>url</url-pattern>
    </servlet-mapping>

    -아래의 코드로 변환
    <servlet>
        <servlet-name>springDispatcherServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <init-param>
            <param-name>contextConfigLocation</param-name>
            <param-value>classpath:applicationContext*.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

11)MemberVO class 생성
    -src/com.javasoft.vo package 생성
    -src/com.javasoft.vo.MemberVO class 생성

        package com.javasoft.vo;

        public class MemberVO {
            private String name, userid, gender, city;
            public MemberVO() {}
            public MemberVO(String name, String userid, String gender, String city) {
                this.name = name;
                this.userid = userid;
                this.gender = gender;
```

```
435             this.city = city;
436          }
437          public String getName() {
438             return name;
439          }
440          public void setName(String name) {
441             this.name = name;
442          }
443          public String getUserid() {
444             return userid;
445          }
446          public void setUserid(String userid) {
447             this.userid = userid;
448          }
449          public String getGender() {
450             return gender;
451          }
452          public void setGender(String gender) {
453             this.gender = gender;
454          }
455          public String getCity() {
456             return city;
457          }
458          public void setCity(String city) {
459             this.city = city;
460          }
461          @Override
462          public String toString() {
463             return "MemberVO [name=" + name + ", userid=" + userid + ", gender=" +
                gender + ", city=" + city + "]";
464          }
465       }
466
467    12)MemberDao 객체 생성
468       -src/com.javasoft.dao package 생성
469       -src/com.javasoft.dao.MemberDao interface
470
471          package com.javasoft.dao;
472
473          import java.util.List;
474
475          import com.javasoft.vo.MemberVO;
476
477          public interface MemberDao {
478             void create(MemberVO member);
479             List<MemberVO> readAll();
480             MemberVO read(String userid);
481             void update(MemberVO member);
482             void delete(String userid);
483          }
484
485       -src/com.javasoft.dao.MemberDaoImpl.java 생성
486
487          package com.javasoft.dao;
488
489          import java.util.List;
490
491          import org.springframework.beans.factory.annotation.Autowired;
492          import org.springframework.stereotype.Repository;
493          import org.apache.ibatis.session.SqlSession;
```

```
494
495         import com.javasoft.vo.MemberVO;
496
497         @Repository("memberDao")
498         public class MemberDaoImpl implements MemberDao {
499            @Autowired
500            private SqlSession sqlSession;
501
502            @Override
503            public void create(MemberVO member) {
504               this.sqlSession.insert("Member.insert", member);
505            }
506
507            @Override
508            public List<MemberVO> readAll() {
509               return this.sqlSession.selectList("Member.select");
510            }
511
512            @Override
513            public MemberVO read(String userid) {
514               return this.sqlSession.selectOne("Member.selectMember", userid);
515            }
516
517            @Override
518            public void update(MemberVO member) {
519               this.sqlSession.update("Member.update", member);
520            }
521
522            @Override
523            public void delete(String userid) {
524               this.sqlSession.delete("Member.delete", userid);
525            }
526         }
527
528
529      13)MemberService 객체 생성
530         -src/com.javasoft.service package 생성
531         -src/com.javasoft.service.MemberService interface
532
533            package com.javasoft.service;
534
535            import java.util.List;
536
537            import com.javasoft.vo.MemberVO;
538
539            public interface MemberService {
540               void insertMember(MemberVO member);
541               List<MemberVO> select();
542               MemberVO selectMember(String userid);
543               void updateMember(MemberVO member);
544               void deleteMember(String userid);
545            }
546
547         -src/com.javasoft.service.MemberServiceImpl.java
548
549            package com.javasoft.service;
550
551            import java.util.List;
552
553            import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;

import com.javasoft.dao.MemberDao;
import com.javasoft.vo.MemberVO;

@Service("memberService")
public class MemberServiceImpl implements MemberService {
   @Autowired
   private MemberDao memberDao;

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

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

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

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

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

14)HomeController 객체 생성
-src/com.javasoft.controller package 생성
-com.javasoft.controller.HomeController class 생성

```
package com.javasoft.controller;

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

import com.javasoft.service.MemberService;
import com.javasoft.vo.MemberVO;

@Controller
public class HomeController {
   @Autowired
   private MemberService service;

}
```

15)config/dbinfo.properties 파일 생성

```
db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@192.168.56.3:1521:ORCL
db.username=scott
db.password=tiger
```

16)applicationContext.xml 수정

```
<context:component-scan base-package="com.javasoft" />
    <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>

    <!-- <bean id="viewResolver"
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean> -->

    <mvc:annotation-driven />
    <mvc:default-servlet-handler/>
```

17)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.javasoft.vo.MemberVO" alias="memberVO"/>
    </typeAliases>
</configuration>
```

18)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>
```

19)applicationContext.xml 아래 코드 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>
```

20)전체 사용자 조회하기

-HomeController 객체 코드 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

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

-Postman

GET http://localhost:8080/RestfulDemo/members Send

Body

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

```
                "gender": "남성",
                "city": "부산"
            },
            {
                "userId": "javaexpert",
                "name": "이미자",
                "gender": "여성",
                "city": "광주"
            }
        ]
    }

-WebContent/index.html

    <!DOCTYPE html>
    <html>
        <head>
            <meta charset="UTF-8">
            <title>Welcome</title>
            <script src="js/jquery-1.12.4.js"></script>
            <script>
                $(document).ready(function(){
                    $.ajax({
                        url:"/RestfulDemo/members",
                        type : "GET",
                        dataType : "json",
                        success : function(data){
                            var str = "";
                            var members = data.data;
                            for(var i = 0 ; i < members.length ; i++){
                                str += "<tr>";
                                var userid = members[i].userid;
                                str += "<td><a href='view.html?userid=" + userid + "'>" +
                                userid + "</a></td>" +
                                    "<td>" + members[i].name + "</td>" +
                                    "<td>" + members[i].gender + "</td>" +
                                    "<td>" + members[i].city + "</td>";
                                str += "</tr>";
                            }
                            $("#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>
```

21)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/javasoft Send

Body

```
{
    "code": "success",
    "data": {
        "userId": "javasoft",
        "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>


22)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;
      }

   -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
```

```
911
912         @Override
913         public void insertMember(MemberVO member) {
914            this.memberDao.create(member);
915         }
916
917      -Postman
918         POST http://localhost:8080/RestfulDemo/members
919         Body
920            raw
921
922            {
923               "userId" : "girlsage",
924               "name" : "소녀시대",
925               "gender" : "여성",
926               "city" : "수원"
927            }
928
929         Send 버튼 클릭하면
930
931            Body
932               {"code": "success"}
933
934      -WebContent/register.html
935
936         <!DOCTYPE html>
937         <html>
938            <head>
939               <meta charset="UTF-8">
940               <title>Member Add</title>
941               <script src="js/jquery-1.12.4.js"></script>
942               <script>
943                  $(function(){
944                     $("input[type='button']").bind("click", function(){
945                        $.ajax({
946                           url : "/RestfulDemo/members",
947                           contentType : "application/json;charset=utf-8",
948                           type : "POST",
949                           data : JSON.stringify({
950                              "userid" : $("#userid").val(),
951                              "name" : $("#name").val(),
952                              "gender" : $(".gender:checked").val(),
953                              "city" : $("#city").val()
954                           }),
955                           dataType : "json",
956                           success : function(data){
957                              alert(data.code);
958                              location.href = "/0605/";
959                           }
960                        });
961                     });
962                  });
963               </script>
964            </head>
965            <body>
966               <h1>Member Add</h1>
967               <ul>
968                  <li>Name : <input type="text" id="name" /></li>
969                  <li>ID : <input type="text" id="userid" /></li>
970                  <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>

23)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
            raw

            {
                "userId" : "girlsage",
                "name" : "소년시대",
                "gender" : "남성",
                "city" : "부산"
            }

        Send 버튼 클릭하면

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

-WebContent/view.html 수정
--아래 코드를 추가한다.

```
<a href = "javascript:void(0)"
onclick="javascript:member_update()">수정하기</a>

var flag = false;
function member_update(){
   if(!flag){   //수정하기를 클릭하면
      var name = $("#name").text();
      $("span#name")
      .replaceWith("<input type='text' id='name' value='" + name + "'/>");
      var gender = $("#gender").text();
      var str = null;
      if(gender == "남성"){
         str = "<input type='radio' class='gender' value='남성'
         checked/>남성  " +
           "<input type='radio' class='gender' value='여성' />여성";
      }else{
         str = "<input type='radio' class='gender' name='gender' value='남성'
         />남성  " +
           "<input type='radio' class='gender' name='gender' value='여성'
           checked />여성";
      }
      $("span#gender").replaceWith(str);
      var city = $("#city").text();
      $("span#city")
      .replaceWith("<input type='text' id='city' value='" + city + "'/>");
      flag = true;
   }else{
      $.ajax({
         url : "/RestfulDemo/members",
         type : "PUT",
         data : JSON.stringify({
            "userid" : userid,
            "name" : $("#name").val(),
            "gender" : $(".gender:checked").val(),
            "city" : $("#city").val()
         }),
         contentType : "application/json;charset=utf-8",
         success : function(data){
            alert(data.code);
            location.reload();
         }
      });
      flag = false;
   }
}
```

24)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 버튼 클릭하면

        Body
            {"code": "success"}

-WebContent/view.html 수정
    --아래의 코드를 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. 데이터 변환 - XML로 변환
  1)Maven Repository에서 'spring xml'로 검색
  2)Spring Object/XML Marshalling에서 4.3.13.RELEASE 선택
  3)pom.xml에 아래 dependency 추가 > Maven Clean > Mavan Install

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-oxm</artifactId>842 <version>4.3.13.RELEASE</version>
    </dependency>

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

```
<dependency>
   <groupId>javax.xml.bind</groupId>
   <artifactId>jaxb-api</artifactId>
   <version>2.3.0</version>
</dependency>
```

6)src/com.javasoft.vo/UserListVO.java 생성

```
package com.javasoft.vo;

import java.util.List;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

import org.springframework.stereotype.Component;

@XmlRootElement(name="userList")
@XmlAccessorType(XmlAccessType.FIELD)
@Component
public class UserListVO {
   @XmlElement(name="user")
   private List<UserVO> userList;

   public List<UserVO> getUserList() {
      return userList;
   }

   public void setUserList(List<UserVO> userList) {
      this.userList = userList;
   }
}
```

-XML 문서는 반드시 단 하나의 root element를 가져야 한다.
-여러 UserVO를 표현하려면 root element로 사용할 또 다른 Java class가 필요하다.
-새로 생성한 UserListVO객체는 이 객체가 root element에 해당하는 객체이며, root element 이름을 userList로 설정하겠다는 의미로 @XmlRootElement(name="userList") 설정을 추가했다.
-그리고 userList 변수 위에도 @XmlElement(name="user")를 추가했는데, UserVO 마다 element 이름을 user로 변경할 것이다.

7)src/com.javasoft.vo.MemberVO.java 수정

```
package com.javasoft.vo;

import javax.xml.bind.annotation.XmlAccessType;
mport javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlAttribute;
import javax.xml.bind.annotation.XmlRootElement;

import org.springframework.stereotype.Component;

@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.907
-그리고 XmlAccessType.FIELD 때문에 이 객체가 가진 필드, 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 파일에서 p와 oxm 체크후, 아래 코드 추가
-JSON 변환시 Java 객체를 JSON response body로 변환해주는 MappingJackson2HttpMessageConverter를 Spring 설정파일에 추가해야 하는데, 설정 파일에 <mvc:annotation-driven />으로 대체했었다.
-마찬가지로 Java 객체를 XML response body로 변환할 때는 아래의 코드를 추가한다.

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

9)UserController.java 코드 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>
```