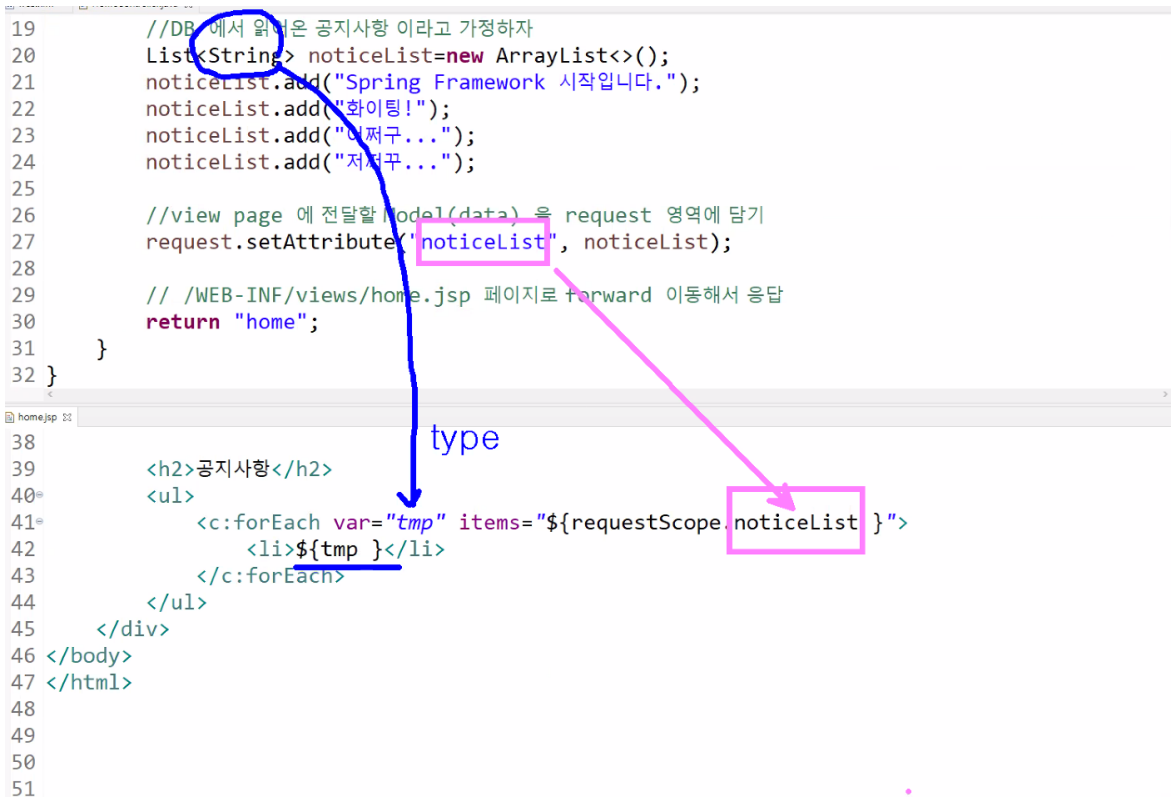
8_31_2022_SPRING

분석

▼ 공지사항

Controller에서 jsp로 forward 이동



▼ 파라미터 추출방법

1. HttpServletRequest 객체의 request메소드 이용

```
<form action="${pageContext.request.contextPath }/send.do" method="post">
    <input type="text" name="msg" placeholder="할말 입력..."/>
    <button type="submit">전송</button>
</form>

<form action="${pageContext.request.contextPath }/send2.do" method="post">
    <input type="text" name="msg" placeholder="할말 입력..."/>
    <button type="submit">전송</button>
</form>
```

```
SendController.java
 *
 * 메소드의 인자로 HttpServletRequest 객체를 전달 받은다음
 * 해당객체의 getParameter() 메소드를 이용해서 추출한다( 서블릿 or jsp 하던 방식 그대로)
 */
@RequestMapping("/send")
public String send(HttpServletRequest request) {
    // msg 라는 파라미터 명으로 전송되는 문자열 추출하기
    String msg=request.getParameter("msg");
    // 콘솔창에 출력하기
    System.out.println("msg:"+msg);
```

2. 파라미터명과 메소드명 일치시키

```
<form action="${pageContext.request.contextPath }/send2.do" method="post">
    <input type="text" name="msg" placeholder="할말 입력..."/>
    <button type="submit">전송</button>
</form>

<h2>post 방식 요청</h2>
<form action="${pageContext.request.contextPath }/add.do" method="post">
```

```
web.xml  SendController.java
 *  xxx.do?num=xxx  -->  int num  or  String num
 */
@RequestMapping("/send2")
public String send2(String msg) {
    // 콘솔창에 출력하기
    System.out.println("msg:"+msg);
    //응답하기
```

▼ 파라미터 추출방법2

1. HttpServletRequest 객체의 request메소드 이용

```
<body>
    <h1>회원 추가 폼 입니다.</h1>
    <form action="${pageContext.request.contextPath }/member/insert.do" method="post">
        번호 <input type="text" name="num"/> <br />
        이름 <input type="text" name="name"/> <br />
        주소 <input type="text" name="addr"/> <br />
        <button type="submit">추가</button>
    </form>
```

```
MemberController.java
    }

    @RequestMapping("/member/insert")
    public String insert(HttpServletRequest request) {
        //HttpServletRequest 객체의 메소드를 이용해서 파라미터 추출하기
        int num=Integer.parseInt(request.getParameter("num"));
        String name=request.getParameter("name");
        String addr=request.getParameter("addr");

        System.out.println(num+"|"+name+"|"+addr);
```

2. 파라미터명과 메소드명 일치시키기

```
    <button type="submit">추가</button>
</form>
<form action="${pageContext.request.contextPath }/member/insert2.do" method="post">
    번호 <input type="text" name="num"/> <br />
    이름 <input type="text" name="name"/> <br />
    주소 <input type="text" name="addr"/> <br />
    <button type="submit">추가</button>
</form>
```

```
@RequestMapping("/member/insert2")
public String insert2(int num, String name, String addr) {

    System.out.println(num+"|"+name+"|"+addr);

    return "member/insert";
```

3. 컨트롤러 메소드의 매개변수에 dto 선언

```
    </form>
    <form action="${pageContext.request.contextPath }/member/insert3.do" method="post">
        번호 <input type="text" name="num"/> <br />
        이름 <input type="text" name="name"/> <br />
        주소 <input type="text" name="addr"/> <br />
        <button type="submit">추가</button>
    </form>
</body>
</html>
```

```
class MemberDto{
  private int num
  private String name
  private String addr

}
```

```
}
//컨트롤러 메소드의 매개변수에 dto 를 선언해도 된다.
@RequestMapping("/member/insert3")
public String insert3(MemberDto dto) {

    System.out.println(dto.getNum()+"|"+dto.getName()+"|"+dto.getAddr());

    return "member/insert";
```

어노테이션

spring 은 @들로 엄청난 일을 하고 있다.

- 어노테이션이 붙은 클래스는 그 모양 그대로 사용되지 않고 런타임시에 변형되어서 사용된다.

어떤 변형이 될까?

- 특정 클래스를 상속받기도 하고, 특정 인퍼테이스를 구현하기도 하고
  없던 메소드가 만들어지기도 하고, 없던 필드가 추가되기도 하고 등등의 작업이 자동화되어 있다.

요청 방식을 구분해서 사용

주소창에 요청 경로를 입력하거나 링크를 클릭하면 무조건 get 방식 요청

```
<h2>post 방식 요청</h2>
<form action="${pageContext.request.contextPath }/add.do" method="post">
    <input type="text" name="content" placeholder="내용입력..."/>
    <button type="submit">추가</button>
</form>
<h2>get 방식 요청</h2>
<form action="${pageContext.request.contextPath }/add.do" method="get">
    <input type="text" name="content" placeholder="내용입력..."/>
    <button type="submit">추가</button>
</form>
```

Step03_MyBatis

DB 연동하는 작업을 배울 것이다.

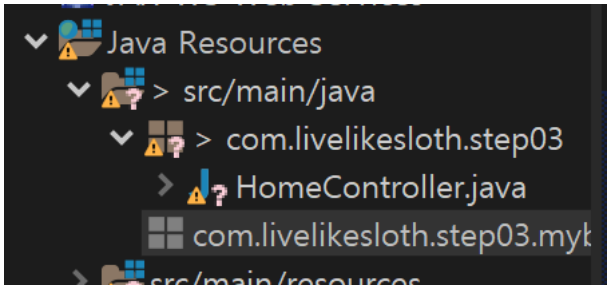
▼ /WEB-INF/spring/appServlet/servlet-context.xml

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

     <!--
      JNDI 데이터 소스 객체 얻어오는 설정
      Servers/context.xml 에 설정된 oracle 접속정보 가 있어야 된다.  복붙

      <Resource name="jdbc/myoracle" auth="Container"
              type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
              url="jdbc:oracle:thin:@127.0.0.1:1521:xe"
              username="scott" password="tiger" maxTotal="20" maxIdle="10"
              maxWaitMillis="-1"/>
   -->
   <beans:bean id="dataSource"
      class="org.springframework.jndi.JndiObjectFactoryBean">
      <beans:property name="jndiName" value="java:comp/env/jdbc/myoracle"/>
   </beans:bean>

   <!--
      위는 아래의 코드와 같다
      dataSource = new JndiObjectFactoryBean();
      dataSource.setJndiName("java:comp/env/jdbc/myoracle");
    -->


   <!--
      SqlSessionFactory 객체
      Configuration.xml 문서가 어디에 있는지 알려야 한다.
   -->
   <beans:bean id="sessionFactory"
      class="org.mybatis.spring.SqlSessionFactoryBean">
      <beans:property name="dataSource" ref="dataSource"/>
      <beans:property name="configLocation"
   value="classpath:com/gura/spring02/mybatis/Configuration.xml"/>
   </beans:bean>
   <!--
      sessionFactory=new SqlSessionFactoryBean();
      sessionFactory.setDataSource(dataSource);
      sessionFactory.setConfigLocation("classpath:com/gura/xxx");
    -->


   <!--
      SqlSession 인터페이스를 구현한
      SqlSessionTemplate(SqlSession) 객체
      Dao 가 의존하는 객체
   -->
   <beans:bean id="sqlSessionTemplate" class="org.mybatis.spring.SqlSessionTemplate">
      <beans:constructor-arg name="sqlSessionFactory"
         ref="sessionFactory"/>
   </beans:bean>



   <!--
      위의 bean 설정을 java code 로 환산 하면 아래와 같다

      dataSource = new JndiObjectFactoryBean();
      dataSource.setJndiName("java:comp/env/jdbc/myoracle");

      sessionFactory=new SqlSessionFactoryBean();
      sessionFactory.setDataSource(dataSource);
      sessionFactory.setConfigLocation("classpath:com/gura/xxx");

      new SqlSessionTemplate(sessionFactory);
    -->


  <!-- DispatcherServlet Context: defines this servlet's request-processing infrastructure -->
```

```
  <!-- Enables the Spring MVC @Controller programming model -->
  <annotation-driven />

  <!-- Handles HTTP GET requests for /resources/** by efficiently serving up static resources in the ${webappRoot}/resources direc
  <resources mapping="/resources/**" location="/resources/" />

  <!-- Resolves views selected for rendering by @Controllers to .jsp resources in the /WEB-INF/views directory -->
  <beans:bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <beans:property name="prefix" value="/WEB-INF/views/" />
    <beans:property name="suffix" value=".jsp" />
  </beans:bean>

  <context:component-scan base-package="com.livelikesloth.step03" />



</beans:beans>
```

▼ src/main/java

새 프로젝트 만든다.



▼ MyBatis

sql문을 자바 문자열로 작성하기 불편했는데 그 문장을 xml문서로 따로 편하게 작성하게 만들어 준다. 뻔한 준비과정을 mybatis가 대신 해준다. db 연동할 때 코딩량이 반 이하로 줄어든다

https://mybatis.org/mybatis-3/ko/getting-started.html

시작하기

```
<!DOCTYPE configuration
PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-config.dtd">
```

좀 전에 만든 패키지에 Configuration.xml 문서를 만들어 붙여넣기

ctrl space 해서 configuration만든다

```
<?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>

  </configuration>
```

▼ Configuration.xml 문서가 어디에 있는지 알리기

servlet-context.xml

```
<!--
      SqlSessionFactory 객체
      Configuration.xml 문서가 어디에 있는지 알려야 한다.
   -->
   <beans:bean id="sessionFactory"
      class="org.mybatis.spring.SqlSessionFactoryBean">
```

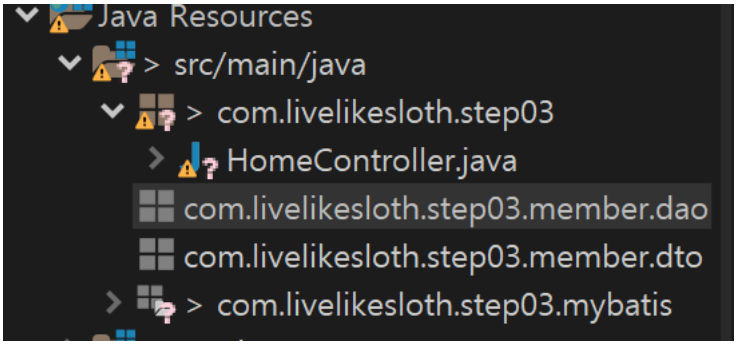
```
    <beans:property name="dataSource" ref="dataSource"/>
    <beans:property name="configLocation"
value="classpath:com/livelikesloth/step03/mybatis/Configuration.xml"/>
</beans:bean>
<!--
```

Spring Bean Container

- spring framework 가 직접 관리하는 container
- servlet-xontext.xml의 설정대로 객체가 관리된다.
- Controller, Service, Dao, SqlSessionTemplate 등의 객체들이 들어있다.

Dao

▼ dao, dto package 추가



▼ 인터페이스 구현

- dto package에 class 만들기

▼ dao package에 interface 만들기

```
package com.livelikesloth.step03.member.dao;

import java.util.List;

import com.livelikesloth.step03.member.dto.MemberDto;

public interface MemberDao {
  public void insert(MemberDto dto);
  public void update(MemberDto dto);
  public void delete(int num);
  public MemberDto getData(int num);
  public List<MemberDto> getList();

}
```

▼ dao package에 daoimpl 클래스 만들기

오버라이드

```
package com.livelikesloth.step03.member.dao;

import java.util.List;

import org.apache.ibatis.session.SqlSession;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

import com.livelikesloth.step03.member.dto.MemberDto;

//MemberDaoImpl 객체도 spring bean container 에서 관리가 되도록 한다.
@Repository
public class MemberDaoImpl implements MemberDao {

  // "spring bean container 에서 sqlSession type 객체를 찾아서 주입해라(DI)" 라는 의미
```

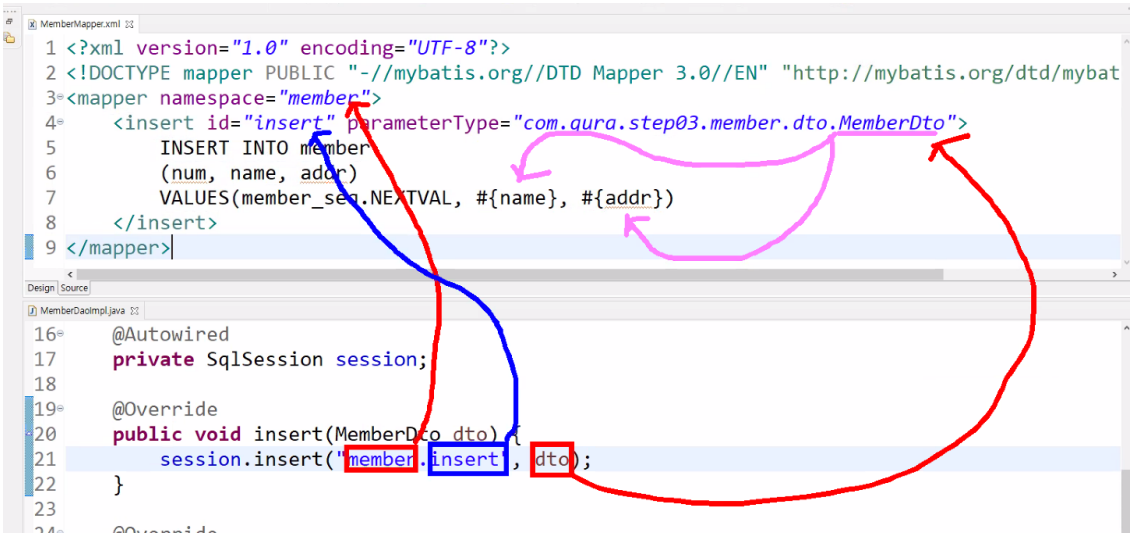
```
  @Autowired
  private SqlSession session;

  @Override
  public void insert(MemberDto dto) {
    session.insert("member.insert",dto);

  }

  @Override
  public void update(MemberDto dto) {
    // TODO Auto-generated method stub

  }

  @Override
  public void delete(int num) {
    // TODO Auto-generated method stub

  }

  @Override
  public MemberDto getData(int num) {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public List<MemberDto> getList() {
    // TODO Auto-generated method stub
    return null;
  }

}
```

▼ MyBatis package에 MyBatis XML Mapper파일 만들기

MemberMapper.xml

ctrl space로 불러와 코딩하면 된다.

```
<?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">
  <insert id="insert" parameterType="com.livelikesloth.step03.member.dto.MemberDto">
    insert into member
    (num, name, addr)
    values(member_seq.nextval, #{name},#{addr})
  </insert>
</mapper>
```



연결되는 모양

▼ Configuration에 연결하기

ctrl space로 코드

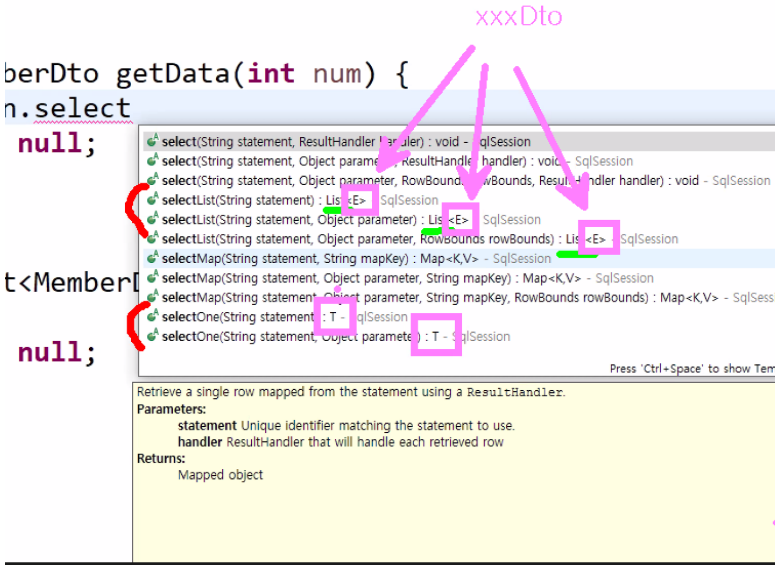
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
  <configuration>
  <!--  사용할 Mapper xml 문서 등록하기 -->
    <mappers>
      <mapper resource="com/livelikesloth/step03/mybatis/MemberMapper.xml"/>
    </mappers>
  </configuration>
```

▼ MemberDaoImpl

insert, update, delete 등은 간단하다.

```
@Autowired
  private SqlSession session;

  @Override
  public void insert(MemberDto dto) {
    session.insert("member.insert",dto);

  }

  @Override
  public void update(MemberDto dto) {
    session.update("member.update", dto);
        /*
         *  Mapper's namespace : member
         *  sql's id : insert
         *  parameterType : MemberDto
         */
  }
@Override
  public void delete(int num) {
    session.delete("member.delete", num);
    /*
     * Mapper's namespace : member
     * sql's id : delete
     * parameterType : int
     */
  }
```

select 하는 방법

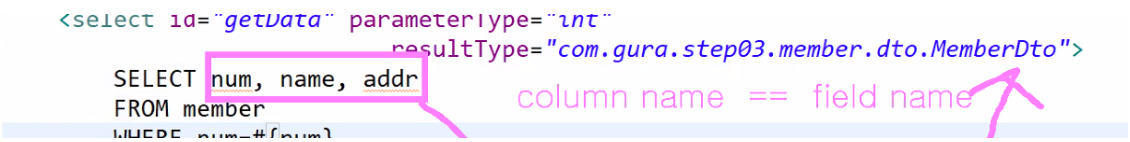


selectOne - row가 하나인경우 사용 - 리턴타입은 고정이 아님
selectList - row가 여러개인 경우 사용-리턴타입은 List 제너릭 타입은 고정이 아님

- getData - row 하나

```
@Override
  public MemberDto getData(int num) {
    /*
     * Mapper's namespace :
     * member sql's id : getData
     * parameterType : int
     * resultType : MemberDto
     */
    MemberDto a=session.selectOne("member.getData", num);
    return a;
  }
```

```
<!--  resultType="select된 row 하나를 담을 데이터 type -->
  <select id="getData" parameterType="int" resultType="com.livelikesloth.step03.member.dto.MemberDto">
    select num, name, addr
    from member
    where num=#{num}
  </select>
```



- getList - row 여러개

```
@Override
  public List<MemberDto> getList() {
    /*
     * Mapper's namespace :
     * member sql's id : getList
     * parameterType : none
     * resultType : MemberDto
     */
    List<MemberDto> list = session.selectList("member.getList");
    return list;
  }
```

```
<select id="getList" resultType="MemberDto">
    select num, name, addr
    from member
    order by num asc
  </select>
```

회원 목록을 불러오는 방법

- Spring에서는 new 해서 불러오지 않고 프레임 워크에 맡긴다.

▼ home에 링크를 달고 그 링크의 package와 Controller 를 만들어준다.

```
package com.livelikesloth.step03.member.controller;

import java.util.List;

import javax.servlet.http.HttpServletRequest;

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

import com.livelikesloth.step03.member.dao.MemberDao;
import com.livelikesloth.step03.member.dto.MemberDto;

@Controller
```

```
public class MemberController {

   // spring bean container 로 부터 MemberDao 인터페이스 type 의 참조값을 DI 받는다.
   @Autowired
   private MemberDao dao;


   @RequestMapping("/member/list")
   public String list(HttpServletRequest request) {


      List<MemberDto> list=dao.getList();

      request.setAttribute("list", list);
      /*
       *  /WEB-INF/views/member/list.jsp 페이지로 forward 이동해서
       *  회원 목록을 출력하면된다.
       *
       *  그런데... 회원목록은 어떻게 얻어오지?
       */
      return "member/list";
   }
}
```

수정 삭제 기능 추가하기

/member/updateform.do

/member/update.do

/member/delete.do

회원 목록에 대해서 수정 삭제 기능을 완성시켜 보기

```
<td><a href="${pageContext.request.contextPath}/member/delete.do?num=${tmp.num}">삭제</a></td>
```

?num=${tmp.num} 를 사용해 지정한 요소 선택하기

update

membercontroller

```
@RequestMapping("/member/updateform")
   public String updatrform(int num, HttpServletRequest request) {
     //MemberDao 객체를 이용해서 수정할 회원의 정보를 DB에서 불러온다.
    MemberDto dto = dao.getData(num);

    //view page에 전달할 모델(data)를 request 영역에 담는다.
    request.setAttribute("dto", dto);
    //view page로 forward 이동해서 수정할 회원폼을 출력해 준다.
     return "member/updateform";
   }
```

updateform

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>/view/member/updateform.jsp</title>
</head>
<body>
  <div class="container">
    <h1>회원정보 수정 폼</h1>
    <form action="${pageContext.request.contextPath}/member/update.do" method="post">
      <input type="hidden" name="num" value="${dto.num }" />
      <div>
```

```
        <label for="num">번호</label>
        <input type="text" name="num" id="num" value="${requestScope.dto.num }" disabled />
      </div>
      <div>
        <label for="name">이름</label>
        <input type="text" name="name" id="name" value"${dto.name } />
      </div>
      <div>
        <label for="addr">주소</label>
        <input type="text" name="addr" id="addr" value="${dto.addr }" />
      </div>
      <button type="submit">수정</button>
      <button typr="reset">취소</button>

    </form>
  </div>

</body>
</html>
```

💡 ${requestScope.dto.XXX}으로 선택한 정보를 불러올 수 있다.
줄여서 ${dto.XXX}

할일 목록 만들기

1. 패키지 만들기
    a. .todo.dto
    b. .todo.dao
    c. .todo.controller
2. 클래스, 인터페이스 만들기
    a. TodoDto
    b. TodoDao, TodoDaoImpl
    c. TodoController
3. TodoMapper.xml
    a. Configuration.xml 문서에 TodoMapper.xml 사용할 준비하기
4. /views/todo/ 폴더를 만들고 폴더안 채우기
    a. list.jsp
    b. insertform.jsp
    c. insert.jsp
    d. delete.jsp
    e. updateform.jsp
    f. update.jsp
5. 처리할 요청
    a. /todo/list.do , /todo/insertform.do , /todo.insert.do , /todo/updateform.do , /todo/update.do , /todo.delete.do