**mybatis 시작**

2021-09-15

이승진

**학습목표**

Spring boot + mybatis 기술을 사용하여 DB 조회, 수정, 삽입, 삭제 기능을 구현한다.

redirect 매커니즘

**목차**

[1. mybatis1 프로젝트 2](#_Toc52719023)

[1) 프로젝트 생성 2](#_Toc52719024)

[2) pom.xml 4](#_Toc52719025)

[3) application.properties 5](#_Toc52719026)

[2. Java 클래스 구현 6](#_Toc52719027)

[1) Department DTO 구현 6](#_Toc52719028)

[2) Department mapper 구현 7](#_Toc52719029)

[3) Department Controller 10](#_Toc52719030)

[4) request parameter 전달 12](#_Toc52719031)

[3. 정적 컨텐츠 13](#_Toc52719032)

[1) common.css 13](#_Toc52719033)

[4. 뷰 구현 14](#_Toc52719034)

[1) department/list.jsp 14](#_Toc52719035)

[2) department/edit.jsp 15](#_Toc52719036)

[5. 실행 16](#_Toc52719037)

[1) 실행 16](#_Toc52719038)

[2) request parameter 이해 17](#_Toc52719039)

[3) 수정 기능 실행 과정 설명 18](#_Toc52719040)

[4) redirect 복습 21](#_Toc52719041)

[5) 생성 기능 실행 과정 설명 22](#_Toc52719042)

[6. 삭제 기능 구현 25](#_Toc52719043)

[1) department/list.jsp 수정 25](#_Toc52719044)

[2) department/edit.jsp 27](#_Toc52719045)

[3) 참조 무결성 위반 에러 28](#_Toc52719046)

# mybatis1 프로젝트

## 프로젝트 생성

메뉴: File - New - Spring Starter Project

Graphical user interface, text, application, email

Description automatically generated

|  |  |
| --- | --- |
| Name | mybatis1 |
| Package | net.skhu |

Graphical user interface, text, application

Description automatically generated

Spring Booot Version: 2.5.# 버전을 선택함.

### 체크할 항목들

Developer Tools 항목 아래

Spring Boot DevTools

SQL 항목 아래

MySQL Driver

MyBatis Framework

Web 항목 아래

Spring Web

## pom.xml

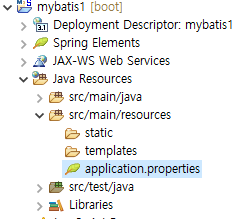
|  |
| --- |
| <dependency>  <groupId>org.apache.tomcat.embed</groupId>  <artifactId>tomcat-embed-jasper</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>jstl</artifactId>  </dependency> |

위 항목들을 maven dependency 라고 부른다.

pom.xml 파일에서 <dependency> 태그들이 있는 곳에 추가하자.

pom.xml은 maven 설정 파일이다.

## application.properties



application.properties는 spring Boot 설정 파일이다.

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/  spring.mvc.view.suffix=.jsp  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.datasource.url=jdbc:mysql://localhost:3306/student1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

|  |
| --- |
| spring.mvc.view.prefix=/WEB-INF/views/ |

뷰 파일(\*.jsp)이 위치할 폴더를 지정한다.

|  |
| --- |
| spring.mvc.view.suffix=.jsp |

뷰 파일의 확장자를 지정한다.

|  |
| --- |
| spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver |

JDBC 드라이버 클래스의 이름를 지정한다.

위 이름은 MySQL JDBC 드라이버 클래스이다.

|  |
| --- |
| spring.datasource.url=jdbc:mysql://localhost:3306/student1?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul |

데이터베이스 서버 IP 와 데이터베이스 이름을 지정한다.

MySQL 설정: utf-8 인코딩 설정, 서버 타임존 설정

|  |
| --- |
| spring.datasource.username=user1  spring.datasource.password=test123 |

데이터베이스에 연결할 계정을 지정한다.

|  |
| --- |
| server.port=8088 |

spring boot app에 내장된 톰캣 서버가 사용할 포트 번호

# Java 클래스 구현

## Department DTO 구현

### src/main/java/net/skhu/dto/Department.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | package net.skhu.dto;  public class Department {  int id;  String departmentName;  public int getId() {  return id;  }  public void setId(int id) {  this.id = id;  }  public String getDepartmentName() {  return departmentName;  }  public void setDepartmentName(String departmentName) {  this.departmentName = departmentName;  }  } |

Department 테이블에서 조회한 데이터를 채울 DTO (Data Transfer Object) 이다.

## Department mapper 구현

### src/main/java/net/skhu/mapper/DepartmentMapper.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Delete;  import org.apache.ibatis.annotations.Insert;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Options;  import org.apache.ibatis.annotations.Select;  import org.apache.ibatis.annotations.Update;  import net.skhu.dto.Department;  @Mapper  public interface DepartmentMapper {  @Select("SELECT \* FROM department")  List<Department> findAll();  @Select("SELECT \* FROM department WHERE id = #{id}")  Department findOne(int id);  @Insert("INSERT department (departmentName) VALUES (#{departmentName})")  @Options(useGeneratedKeys=true, keyProperty="id")  void insert(Department department);    @Update("UPDATE department SET departmentName = #{departmentName} WHERE id = #{id}")  void update(Department department);  @Delete("DELETE FROM department WHERE id = #{id}")  void delete(int id);  } |

### @Mapper

mybatis mapper에는 @Mapper 애너테이션이 붙어있어야 한다.

### findAll

|  |
| --- |
| @Select("SELECT \* FROM Department")  List<Department> findAll(); |

findAll 메소드는, department 테이블의 모든 레코드를 조회해서 리턴한다.

findAll 메소드를 호출하면, "SELECT \* FROM department" SQL 명령이 mySQL 엔진에서 실행된다.

이 메소드의 조회 결과 레코드 한 개당 Department 객체 한 개가 생성되어,

조회 결과 레코드의 컬럼들이 Department 객체의 속성에 채워진다.

이때 컬럼명과 객체의 속성명이 일치하는 것만 채워지고, 일치하지 않는 것은 무시된다.

이렇게 생성된 Department 객체들이 List 타입의 객체에 채워져 리턴된다.

조회 결과 레코드가 없을 경우에, 이 메소드는 size가 0 인 List<Department> 객체를 리턴한다.

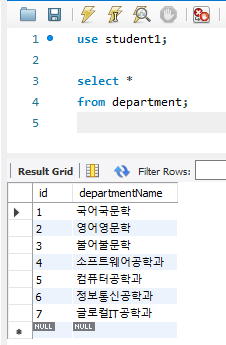
(참고: Java에서 List는 클래스가 아니고 interface 이다.

List interface를 구현한 대표적인 클래스가 ArrayList 이다.)

### 조회 결과 컬럼명 일치

SELECT SQL 명령의 조회 결과가 Java 객체에 자동으로 채워질 때,

조회 결과 컬럼명과 Java 객체의 setter 이름이 일치해야 한다.



|  |  |
| --- | --- |
| 조회 결과 컬럼명 | Java 객체의 setter |
| id | setId |
| departmentName | setDepartmentName |

노란색으로 칠한 문자만 대문자이고, 나머지 문자는, 조회결과 컬럼명과 대소문자까지 일치해야 한다.

일치하지 않을 경우, 에러가 발생하지는 않고, 값이 채워지지 않는다.

### findOne

|  |
| --- |
| @Select("SELECT \* FROM department WHERE id = #{id}")  Department findOne(int id); |

findOne 메소드는, department 테이블의 레코드 한 개를 조회해서 리턴한다.

findOne 메소드를 호출하면, "SELECT \* FROM department WHERE id = #{id}" 명령이 mySQL 엔진에서 실행된다.

이때, 이 메소드의 파라미터 id 변수의 값이, #{id} 부분에 채워진 후 실행된다.

이 메소드는, Department 객체를 한 개 생성해서, 조회 결과 레코드의 각 컬럼의 값을,

이 Department 객체에 채워서 리턴한다.

이때 컬럼명과 객체의 속성명이 일치하는 것만 채워지고, 일치하지 않는 것은 무시된다.

조회 결과가 없을 경우에, 이 메소드는 null을 리턴한다.

### insert

|  |
| --- |
| @Insert("INSERT department (departmentName) VALUES (#{departmentName})")  @Options(useGeneratedKeys=true, keyProperty="id")  void insert(Department department); |

insert 메소드의 파라미터 변수가 Department 객체이다.

이 Java 객체의 속성값이, INSERT SQL 명령의 #{...} 부분에 채워져서 실행된다.

Java 객체의 속성명과, #{...} 부분의 이름이 일치해야 속성값이 채워질 수 있다.

#{...} 부분의 이름과 일치하는 Java 객체의 속성이 없을 경우에, 에러가 발생한다.

@Options(useGeneratedKeys=true, keyProperty="id")

INSERT할 테이블의 기본키(primary key) 필드 이름이 "id" 이고,

이 필드의 값은 자동으로 생성된다는 선언이다. (Auto Increment 필드)

### update

|  |
| --- |
| @Update("UPDATE department SET departmentName = #{departmentName} WHERE id = #{id}")  void update(Department department); |

update 메소드의 실행 과정은, 위 insert 메소드와 동일하다.

### 파라미터 이름 일치

mapper 메소드의 파라미터 Java 객체의 속성값이, SQL 명령의 #{...} 부분에 채워질 때

객체의 getter 이름과 #{...} 부분의 이름이 일치해야 한다.

|  |  |
| --- | --- |
| Java 객체의 getter | mybatis 파라미터 |
| getId | #{id} |
| getDepartmentName | #{departmentName} |

### delete

|  |
| --- |
| @Delete("DELETE FROM department WHERE id = #{id}")  void delete(int id); |

delete 메소드의 파라미터 변수 id 값이, SQL 명령의 #{id} 부분에 채워져서,

SQL 명령이 mySQL 엔진에서 실행된다.

## Department Controller

### src/main/java/net/skhu/controller/DepartmentController.java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59 | package net.skhu.controller;  import java.util.List;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Controller;  import org.springframework.ui.Model;  import org.springframework.web.bind.annotation.GetMapping;  import org.springframework.web.bind.annotation.PostMapping;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestParam;  import net.skhu.dto.Department;  import net.skhu.mapper.DepartmentMapper;  @Controller  @RequestMapping("department")  public class DepartmentController {  @Autowired DepartmentMapper departmentMapper;  @RequestMapping("list")  public String list(Model model) {  List<Department> departments = departmentMapper.findAll();  model.addAttribute("departments", departments);  return "department/list";  }  @GetMapping("create")  public String create(Model model) {  model.addAttribute("department", new Department());  return "department/edit";  }  @PostMapping("create")  public String create(Model model, Department department) {  departmentMapper.insert(department);  return "redirect:list";  }  @GetMapping("edit")  public String edit(Model model, @RequestParam("id") int id) {  Department department = departmentMapper.findOne(id);  model.addAttribute("department", department);  return "department/edit";  }  @PostMapping("edit")  public String edit(Model model, Department department) {  departmentMapper.update(department);  return "redirect:list";  }  @RequestMapping("delete")  public String delete(Model model, @RequestParam("id") int id) {  departmentMapper.delete(id);  return "redirect:list";  }  } |

### @Autowired

@Autowired DepartmentMapper departmentMapper;

DepartmentMapper 인터페이스를 구현한 Java 클래스를, spring mybatis가 자동으로 구현해 주고,

그 클래스의 객체를 한 개 생성하여, departmentMapper 멤버 변수에 자동으로 대입(autowired)해 준다.

그래서 departmentMapper 멤버 변수를 선언했을 뿐이고, 어떤 객체도 대입해 주지 않았지만,

mybatis spring이 자동으로 생성해준 객체가 이 멤버 변수에 자동으로 대입되어 있기 때문에,

액션 메소드에서 이 객체를 사용할 수 있다.

### list 액션 메소드

@RequestMapping("department") 컨트롤러 클래스 어노테이션의 URL과

@RequestMapping("list") 액션 메소드 어노테이션의 URL을 결합한

"department/list" URL이 톰캣 서버에 요청되면,

list 액션 메소드가 자동으로 호출된다.

### create 액션 메소드

create 액션 메소드가 두 개이다.

@GetMapping("create")

"department/create" URL이 GET 방식으로 요청되면, 이 액션 메소드가 자동으로 호출된다.

@PostMapping("create")

"department/create" URL이 POST 방식으로 요청되면, 이 액션 메소드가 자동으로 호출된다.

컨트롤러 액션 메소드 이름은 URL과 무관하고, 아무 이름이나 부여해도 상관없다.

### list 액션 메소드 본문

List<Department> departments = departmentMapper.findAll();

model.addAttribute("departments", departments);

DB의 department 테이블의 모든 레코드를 조회한 결과 List<Department> 타입의 객체를

"departments" 이름으로 model 객체에 넣는다.

이런 데이터를 model attribute 라고 부른다.

model attribute 데이터는 뷰(view)에 전달된다.

### list 액션 메소드의 리턴값

return "department/list";

액션 메소드가 리턴하는 문자열은, 뷰(view)의 이름이다.

application.properties 파일에 등록된, view prefix와 view suffix를 뷰 이름 앞 뒤에 붙이면,

뷰 파일의 경로명이 된다. 이 뷰 파일이, 액션 메소드가 리턴된 후, 실행된다.

### create 액션 메소드의 리턴값

return "redirect:list";

액션 메소드를 실행한 후, "list" URL로 redirect 하라는 명령을 웹브라우저에 출력한다.

이 명령을 받은 웹브라우저는, 그 URL을 서버에 즉시 다시 요청하게 된다.

"list" URL은 상대 URL 이다. 상대 URL에서 절대 URL을 만드는 계산 방법을 이해해야 한다.

"redirect:list" 이 문자열에 혹시 공백이 포함되어 있으면 에러가 발생한다.

## request parameter 전달

액션 메소드의 파라미터에 request parameter 값이 채워져서 액션 메소드에 전달된다.

### 값을 하나씩 전달 받을 때 - @RequestParam

@GetMapping("edit")

public String edit(Model model, @RequestParam("id") int id) {

위 소스 코드에서 @RequestParam("id") int id 부분에서 일어나는 일은 아래와 같다.

String s = request.getParameter("id");

int id = Integer.parseInt(s);

### 값을 여러 개 전달 받을 때 - 파라미터 객체

@PostMapping("edit")

public String edit(Model model, Department department) {

request parameter 데이터가 Department 객체에 자동으로 채워진다.

request parameter 데이터의 이름과,

액션 메소드의 파라미터 객체의 속성 이름이 일치할 경우에 (set 메소드 이름이 일치할 경우에),

request parameter 데이터가 그 속성에 자동으로 채워진다.

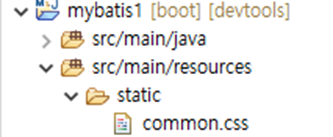
|  |  |
| --- | --- |
| request parameter 이름, | Department 클래스 set 메소드 |
| id | setId |
| departmentName | setDepartmentName |

# 정적 컨텐츠

웹서버에서 실행되지 않고, 파일 내용 그대로 웹브라우저로 전달되는 파일들을 정적 컨텐츠라고 한다.

(예: \*.jpg, \*.png, \*.css, \*.js)

spring boot app에서 정적 컨텐츠는 src/main/resources/static 폴더에 저장하는 것이 좋다.



## common.css

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | div.container { width: 800px; margin: 10px auto; font-size: 10pt; }  .btn { padding: 0.4em 1em; border: 1px solid gray;  border-radius: 0.5em; background: linear-gradient(#fff, #ddd);  text-decoration: none; color: black;  display: inline-block; }  .btn:active {  -ms-transform: translateY(2px);  -webkit-transform: translateY(2px);  transform: translateY(2px);  background-color: #ccc; }  table.list { border-collapse: collapse; width: 100%; }  table.list td { padding: 4px; border: 1px solid gray; }  table.list th { padding: 4px; border: 1px solid gray; background-color: #eee; }  input { padding: 4px; }  select { padding: 4px; } |

여러 JSP 파일에 공통으로 적용될 CSS 서식을 구현함

HTML 태그와 CSS 서식은 이 수업의 학습목표가 아니기 때문에, 설명하지 않는다.

그렇지만 HTML 태그와 CSS 서식에 대해 질문은 해도 됨.

(내 웹프로그래밍 강의에서 CSS 서식에 대해 설명함.)

목표가 백엔드 개발자라면, 촌스럽지 않을 정도의 화면을 만들 수 있을 정도로 CSS를 공부해야 한다.

이 강의 예제에서 구현된 CSS가 딱 이정도.

목표가 프론트엔드 개발자라면, 멋진 화면을 만들 수 있어야 하니, CSS 공부를 열심히 해야 한다.

# 뷰 구현

## department/list.jsp

### src/main/webapp/WEB-INF/views/department/list.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <%@ 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 http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  a.btn { float: right; margin: -20px 0 5px 0; }  td:nth-child(1) { text-align: center; }  </style>  </head>  <body>  <div class="container">  <h1>학과목록</h1>  <a href="create" class="btn">학과등록</a>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>학과명</th>  </tr>  </thead>  <tbody>  <c:forEach var="department" items="${ departments }">  <tr>  <td>${ department.id }</td>  <td><a href="edit?id=${ department.id }">${ department.departmentName }</a></td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

c: 으로 시작하는 JSTL 확장 태그를 사용하기 위해 필요한 선언.

<link rel="stylesheet" type="text/css" href="/common.css" />

common.css 파일에 적용된 서식이, 이 페이지에 적용된다.

<c:forEach var="department" items="${ departments }">

JSTL 확장 태그

모델에서 "departments" 이름으로 등록된 학과 목록의 학과 객체 각각에 대해서,

그 객체를 department 변수에 대입하고,

<c:forEach>...</c:forEach> 태그 사이의 내용을 반복 출력한다.

## department/edit.jsp

### src/main/webapp/WEB-INF/views/department/edit.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | <%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  <!DOCTYPE html>  <html>  <head>  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  form { width: 600px; margin: auto;  box-shadow: 0 0 4px lightgray, 2px 2px 4px gray; overflow: auto; }  div.title { font-size: 20pt; padding: 10px; background-color: #eee; }  table { margin: 20px; }  td { min-width: 100; padding: 5px; }  td:nth-child(1) { text-align: right; }  button { margin: 5px 0 20px 20px; }  </style>  </head>  <body>  <div class="container">  <form method="post">  <div class="title">학과 정보</div>  <table>  <tr>  <td>id</td>  <td><input type="text" value="${ department.id }" readonly disabled /></td>  </tr>  <tr>  <td>학과명</td>  <td><input type="text" name="departmentName" value="${ department.departmentName }" /></td>  </tr>  </table>  <button type="submit" class="btn">저장</button>  </form>  </div>  </body>  </html> |

# 실행

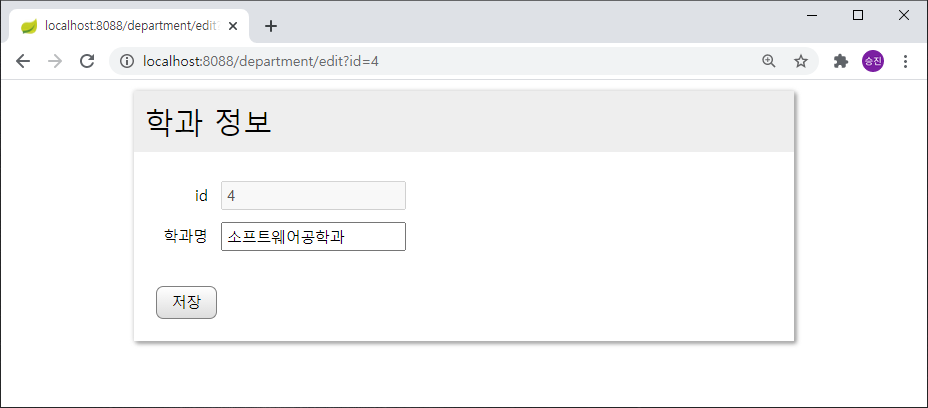
## 실행

메뉴: Run - Run As - Spring Boot App

### http://localhost:8088/department/list



소프트웨어공학과를 클릭하자.



## request parameter 이해

### request parameter란?

웹브라우저에서 웹서버에 URL의 실행을 요청할 때 (http request)

그 요청에 같이 담아 전송하는 데이터를 request parameter라고 부른다.

즉 request parameter는 웹브라우저에서 웹서버로 전송되는 데이터이다.

* 웹브라우저 -> 서버 (전송되는 데이터)
* 언제 전송? 웹브라우저가 서버에 URL을 요청할 때 (http request),  
  그 요청과 같이 전송됨.
* http request는 언제 발생
  + 웹브라우저 주소칸에 URL 입력하고 엔터
  + a 태그 클릭 (요청되는 URL: <a href="URL주소">)
  + submit 버튼 클릭 (요청되는 URL: submit 버튼을 포함하는 <form action="URL주소"> 이거나,  
     action 애트리뷰트가 없으면 현재URL)

### 현재URL은?

* 웹브라우저에서 현재URL은? 웹브라우저창의 주소칸
* 서버에서 현재URL은? 방금 웹브라우저가 요청한 URL (http request)  
  (상대URL 절대URL 계산 방법을 파악해야함)

### 무엇이 request parameter가 되는가?

자동으로 request parameter로 전송되는 것은 다음과 같다.

- URL의 query string

- 입력폼에 입력된 데이타

### request parameter name과 value

request parameter는 name과 value로 구성된다.

- query string의 경우 ?name1=value1&name2=value2 형태이다.

query string name이 request parameter name이 되고

query string value가 request parameter value가 되어 서버에 전송된다.

- 입력폼의 경우에 request paramter name은 태그의 name 값이다.

예: <input name="..." />

<select name="...">

- 입력폼의 경우에 request parameter value는 다음과 같다.

<input type="text" name="...">

태그의 name 값이 request parameter name이 되고

태그에 입력된 값이 request parameter value가 되어 서버에 전송된다.

<select name="...">

<option value="2">소프트웨어공확과</option>

태그의 name 값이 request parameter name이 되고

선택된 option 태그의 value 값이 request parameter value가 되어 서버에 전송된다.

<input type="checkbox" name="..." value="..." >

체크되지 않은 경우에

이 태그는 request parameter 데이터를 서버에 전송하지 않는다.

체크된 경우에

태그의 name 값이 request parameter name이 되고

태그의 value 값이 request parameter value가 되어 서버에 전송된다.

태그의 value 값을 지정하지 않은 경우에는 디폴트 value 값은 "on" 문자열이다.

<input type="radio" name="..." value="..." > 태그:

체크된 태그의 name 값이 request parameter name이 되고

그 태그의 value 값이 request parameter value가 되어 서버에 전송된다.

## 수정 기능 실행 과정 설명

Run - Run on Server 메뉴로 실행했을 때, context path는 **/mybatis1** 이다. (프로젝트명)

Run - Run as Spring Boot App 메뉴로 실행했을 때, context path는 **/** 이다.

(1) 웹브라우저에서, http://localhost:8088/department/list URL을

서버에 요청한다. GET 방식 요청.

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다.

없으면 404 에러.

@RequestMapping("department")

@Controller

class DepartmentController {

@RequestMapping("list")

public String list(...) {

"department" + "list" = "department/list"

(주의할 점: @RequestMapping에 URL을 적을 때, context path를 생략하거나, / 으로 입력해야 한다.)

(3) 요청된 URL과 일치하는 DepartmentController의 list 메소드 실행됨.

departmentMapper.findAll() 메소드를 호출하여

리턴된 학과 목록을 model 객체에 추가함.

List<Department> departments = departmentMapper.findAll();

model.addAttribute("departments", departments);

(4) 액션 메소드 뒤를 이어 실행될 뷰 이름을 리턴함.

return "department/list";

(5) "department/list" 뷰 이름 앞 뒤에 view prefix, view suffix가 붙는다.

(application.properties 참고)

/WEB-INF/views/department/list.jsp 파일이 실행된다.

(6) 뷰는 model에 들어있는 학과 목록 데이터를 출력한다. (이 model attribute 데이터의 이름은 departments)

뷰의 실행 결과 출력은 html 태그이다.

뷰가 출력한 html 태그들이 웹브라우저로 전송되어 웹브라우저창에 그려진다.

- - -

(1) 웹브라우저창의 학과 목록에서 소프트공학과 부분

소스코드

<td><a href="edit?id=${ department.id }">${ department.departmentName }</a></td>

출력

<td><a href="edit?id=4">소프트웨어공학과</a></td>

이 a 태그를 클릭하자.

클릭된 a 태그의 href="edit?id=4" URL이 서버에 요청된다

이 요청은 GET 방식이다.

edit?id=4 이것은 상대 URL 이다.

현재 문맥은 웹브라우저에서 a 태그가 클릭된 것이므로,

웹브라우저의 주소칸의 현재 URL http://localhost:8088/department/list 에서

이름 부분만 상대 URL로 바뀐다.

http://localhost:8088/department/edit?id=4

이 절대 URL이 서버에 요청된다.

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다. 없으면 404 에러.

@RequestMapping("department")

@Controller

class DepartmentController {

@GetMapping("edit")

public String edit(Model model, @RequestParam("id") int id) {

Department department = departmentMapper.findOne(id);

model.addAttribute("department", department);

return "department/edit";

}

(3) request parameter로 전달된 id 값에 해당하는 학과 레코드를 조회해서 model에 추가.

Department department = departmentMapper.findOne(id);

model.addAttribute("department", department);

(4) 액션 메소드 뒤를 이어 실행될 뷰 이름을 리턴함.

return "department/edit";

(5) "department/edit" 뷰 이름 앞 뒤에 view prefix, view suffix가 붙는다.

(application.properties 참고)

/WEB-INF/views/department/edit.jsp 파일이 실행된다.

(6) department/edit.jsp 뷰는 model에 들어있는 학과 데이터를 input 태그에 출력한다.

소스코드:

<input type="text" value="${ department.id }" readonly disabled />

<input type="text" name="departmentName" value="${ department.departmentName }" />

출력:

<input type="text" value="4" readonly disabled />

<input type="text" name="departmentName" value="소프트웨어공학과" />

- - - -

(1) 웹브라우저 창에서 학과명을 '소프트웨어공학전공' 이라고 수정하고, 저장버튼을 클릭.

저장 버튼은 submit 버튼이다.

웹브라우저창의 현재 URL이 다시 서버에 요청된다.

이때 입력폼에 입력된 데이터가 request parameter로 전송된다.

서버에 요청되는 URL: /department/edit?id=4

POST 방식 요청

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다. 없으면 404 에러.

@RequestMapping("department")

@Controller

class DepartmentController {

@PostMapping("edit")

public String edit(Model model, Department department) {

departmentMapper.update(department);

return "redirect:list";

}

request parameter 데이터가 액션 메소드 파라미터 객체에 채워져 전달된다. (Department 객체)

(3) Department 객체에 채워져 전달된 request parameter 데이터가 DB에 저장된다.

departmentMapper.update(department);

(4) "list" URL로 리다이렉트 하라는 명령이 웹브라우저에 전송된다.

return "redirect:list";

a) 현재 서버에 요청된 URL: http://localhost:8088/department/edit?id=3

b) "list" 상대 URL

a)와 b)를 결합하면,

절대 URL: http://localhost:8088/department/list

(5) 리다이렉트 명령을 받은 웹브라우저는 요청된 URL을 서버에 즉시 요청한다.

http://localhost:8088/department/list

(6) 서버에서 DepartmentController의 list 액션 메소드가 실행된다.

DB에서 학과 목록을 조회한다.

이 학과 목록에는, 방금 저장된 학과도 포함되어 있다.

(7) "department/list" 뷰가 실행된다.

(8) 실행결과 출력이 웹브라우저로 전송된다.

## redirect 복습

지낙학기 웹프로그래밍 수업에서 배웠던 redirect 개념을 복습하자.

예를 들어 게시글 수정 화면에서 저장 버튼을 눌럿을 때,

서버에서 데이터가 잘 저장되었으면, 게시글 목록 화면으로 넘어간다.

이렇게 다른 화면으로 넘어가는 과정이 리다이렉트이다.

대부분의 경우 서버의 저장 작업이 성공한 경우에만 목록 화면으로 리다이렉트를 해야하고,

저장 작업에 문제가 발생했다면, 편집 화면이 다시 출력되어야 한다.

즉 저장 작업이 성공하면, 서버는 리다이렉트 명령을 출력해야 하고,

저장 작업에 문제가 발생했으면, 서버는 편집 화면 html 태그들을 출력해야 한다.

|  |  |  |
| --- | --- | --- |
| 웹브라우저 | 통신 | 서버 |
| (1) 학과목록에서  소프트웨어공학과 클릭 | => (2) edit?id=4 GET 방식 요청 | (3) GET 방식 edit 메소드 실행 |
| (6) 학과 수정 폼 출력됨 | <= (5) edit.jsp 실행 결과 출력 | (4) edit.jsp 뷰 실행 |
| (7) 학과 정보를 수정하고,  저장 버튼 클릭 | => (8) edit?id=4 POST 방식 요청 | (9) POST 방식 edit 메소드 실행 |
|  | <= (11) list URL 리다이렉트 명령 | (10) "redirect:list" 리턴 |
| (12) list URL 즉시 요청 | => (12) list URL GET 방식 요청 | (13) list 액션 메소드 실행 |
| (16) 학과 목록 화면 | <= (15) list.jsp 실행 결과 출력 | (14) list.jsp 뷰 실행 |

## 생성 기능 실행 과정 설명

(1) 웹브라우저에서, http://localhost:8088/department/list URL을

서버에 요청한다. GET 방식 요청.

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다.

@RequestMapping("department")

@Controller

class DepartmentController {

@RequestMapping("list")

public String list(...) {

(3) 요청된 URL과 일치하는 DepartmentController의 list 메소드 실행됨.

departmentMapper.findAll() 메소드를 호출하여

리턴된 학과 목록을 model 객체에 추가함.

List<Department> departments = departmentMapper.findAll();

model.addAttribute("departments", departments);

(4) 액션 메소드 뒤를 이어 실행될 뷰 이름을 리턴함.

return "department/list";

(5) "department/list" 뷰 이름 앞 뒤에 view prefix, view suffix가 붙는다.

(application.properties 참고)

/WEB-INF/views/department/list.jsp 파일이 실행된다.

(6) 뷰는 model에 들어있는 학과 목록 데이터를 출력한다. (이 model attribute 데이터의 이름은 departments)

뷰의 실행 결과 출력은 html 태그이다.

뷰가 출력한 html 태그들이 웹브라우저로 전송되어 웹브라우저창에 그려진다.

- - -

(1) 웹브라우저창에서 학과등록 버튼을 클릭한다.

<a href="create" class="btn">학과등록</a>

클릭된 a 태그의 href="create" URL이 서버에 요청된다

이 요청은 GET 방식 요청이다.

create 이것은 상대 URL 이다.

현재 문맥은 웹브라우저에서 a 태그가 클릭된 것이므로,

웹브라우저의 주소칸의 현재 URL http://localhost:8088/department/list 에서

이름 부분만 상대 URL로 바뀐다.

http://localhost:8088/department/create

이 절대 URL이 서버에 요청된다.

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다. 없으면 404 에러.

@RequestMapping("department")

@Controller

class DepartmentController {

@GetMapping("create")

public String create(Model model) {

model.addAttribute("department", new Department());

return "department/edit";

}

(3) Department 객체를 한 개 생성해서 model에 추가.

model.addAttribute("department", new Department());

(4) 액션 메소드 뒤를 이어 실행될 뷰 이름을 리턴함.

return "department/edit";

/WEB-INF/views/department/edit.jsp 파일이 실행된다.

edit 액션 메소드와 create 액션 메소드가 department/edit 뷰 이름을 리턴함에 주목하자.

액션 메소드 마다 뷰를 따로 구현해야 하는 것은 아니다.

액션 메소드들이 출력해야 하는 화면이 동일하다면, 뷰 파일을 공유할 수 있다.

수정화면과 등록 화면에서 edit.jsp 뷰 파일을 공유한다.

(5) department/edit.jsp 뷰는 model에 들어있는 학과 데이터를 input 태그에 출력한다.

소스코드:

<input type="text" value="${ department.id }" readonly disabled />

<input type="text" name="departmentName" value="${ department.departmentName }" />

출력:

<input type="text" value="0" readonly disabled />

<input type="text" name="departmentName" value="" />

모델에 들어있는 department 이름의 객체는 new Department()로 생성한 객체이다.

이 객체의 멤버 변수에 값이 대입되지 않았다.

변수에 아직 값이 대입되지 않은 경우, 참조형 변수의 값은 null 이고, 숫자형 변수의 값은 0 이다.

${ ... } JSP EL 태그로 출력할 때, null 값을 출력하면, 아무것도 출력되지 않는다.

- - - -

(1) 웹브라우저 창에서 학과명을 'IT전자공학부' 라고 입력하고, 저장버튼을 클릭.

저장 버튼은 submit 버튼이다.

웹브라우저창의 현재 URL이 다시 서버에 요청된다.

이때 입력폼에 입력된 데이터가 request parameter로 전송된다.

서버에 요청되는 URL: /department/create

POST 방식 요청

(2) 요청된 URL에 연결된(request mapping) 액션 메소드가 실행된다. 없으면 404 에러.

@RequestMapping("department")

@Controller

class DepartmentController {

@PostMapping("create")

public String create(Model model, Department department) {

departmentMapper.insert(department);

return "redirect:list";

}

request parameter 데이터가 액션 메소드 파라미터 객체에 채워져 전달된다. (Department 객체)

(3) Department 객체에 채워져 전달된 request parameter 데이터가 DB에 저장된다.

departmentMapper.insert(department);

(4) "list" URL로 리다이렉트 하라는 명령이 웹브라우저에 전송된다.

return "redirect:list";

a) 현재 서버에 요청된 URL: http://localhost:8088/department/create

b) "list" 상대 URL

a)와 b)를 결합하면,

절대 URL: http://localhost:8088/department/list

(5) 리다이렉트 명령을 받은 웹브라우저는 요청된 URL을 서버에 즉시 요청한다.

http://localhost:8088/department/list

(6) 서버에서 DepartmentController의 list 액션 메소드가 실행된다.

DB에서 학과 목록을 조회한다.

이 학과 목록에는, 방금 저장된 새 학과도 포함되어 있다.

(7) "department/list" 뷰가 실행된다.

(8) 실행결과 출력이 웹브라우저로 전송된다.

# 삭제 기능 구현

삭제 기능 액션 메소드는 이미 구현되어 있다.

목록 화면에 링크를 만들거나, 수정 화면에 삭제 버튼을 만들 수 있다.

둘 다 만들자.

## department/list.jsp 수정

### src/main/webapp/WEB-INF/views/department/list.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | <%@ 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 http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  a.btn { float: right; margin: -20px 0 5px 0; }  td:nth-child(1) { text-align: center; }  </style>  </head>  <body>  <div class="container">  <h1>학과목록</h1>  <a href="create" class="btn">학과등록</a>  <table class="list">  <thead>  <tr>  <th>id</th>  <th>학과명</th>  <th></th>  </tr>  </thead>  <tbody>  <c:forEach var="department" items="${ departments }">  <tr>  <td>${ department.id }</td>  <td><a href="edit?id=${ department.id }">${ department.departmentName }</a></td>  <td><a href="delete?id=${ department.id }">삭제</a></td>  </tr>  </c:forEach>  </tbody>  </table>  </div>  </body>  </html> |

### 실행

http://localhost:8088/department/list



예를 들어 소프트웨어공학전공의 삭제 링크는 다음과 같은 형태로 출력된다.

<td><a href="delete?id=4">삭제</a></td>

IT융합자율학부의 삭제 링크는 다음과 같은 형태로 출력된다.

<td><a href="delete?id=11">삭제</a></td>

(1) IT융합자율학부의 삭제 링크를 클릭하면,

서버에 "delete?id=11" 상대 URL이 요청된다.

현재 웹브라우저 창의 URL이 http://localhost:8088/department/list 이므로,

여기에 "delete?id=11" 상대 URL을 결합하면

http://localhost:8088/department/delete?id=11 절대 URL이 서버에 요청된다.

(2)

@Controller

@RequestMapping("department")

public class DepartmentController {

@RequestMapping("delete")

public String delete(Model model, @RequestParam("id") int id) {

departmentMapper.delete(id);

return "redirect:list";

}

그 URL 요청에 의해서, 위 액션 메소드가 서버에서 자동으로 호출된다.

이름이 "id" 인 request parameter 값 11이 id 변수에 채워져서 전달된다.

(3)

departmentMapper.delete(id);

return "redirect:list";

DB에서 id 필드값이 11인 레코드가 삭제된다.

http://loccalhost:8088/department/list URL로 리다이렉트 하라는 명령이 출력된다.

## department/edit.jsp

### src/main/webapp/WEB-INF/views/department/edit.jsp

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | <%@ 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 http-equiv="Content-Type" content="text/html; charset=UTF-8">  <link rel="stylesheet" type="text/css" href="/common.css" />  <style>  form { width: 600px; margin: auto;  box-shadow: 0 0 4px lightgray, 2px 2px 4px gray; overflow: auto; }  div.title { font-size: 20pt; padding: 10px; background-color: #eee; }  table { margin: 20px; }  td { min-width: 100; padding: 5px; }  td:nth-child(1) { text-align: right; }  button { margin: 5px 0 20px 20px; }  </style>  </head>  <body>  <div class="container">  <form method="post">  <div class="title">학과 정보</div>  <table>  <tr>  <td>id</td>  <td><input type="text" value="${ department.id }" readonly disabled /></td>  </tr>  <tr>  <td>학과명</td>  <td><input type="text" name="departmentName" value="${ department.departmentName }" /></td>  </tr>  </table>  <button type="submit" class="btn">저장</button>  <c:if test="${ department.id > 0 }">  <a href="delete?id=${ department.id }" class="btn">삭제</a>  </c:if>  </form>  </div>  </body>  </html> |

edit.jsp 뷰는 등록 화면과 수정 화면에서 사용된다.

수정 화면에서는 삭제 버튼이 있어도 되지만, 등록 화면에는 삭제 버튼이 없어야 한다.

수정 화면에서만 삭제 버튼 태그가 출력되도록 <c:if> 확장 태그를 사용하였다.

수정할 때, 모델의 department 객체의 id 속성 값은 1 이상이지만,

등록할 때, 모델의 dpeartment 객체의 id 속성 값은 0 이다.

따라서 등록 화면에서 <c:if test="${ department.id > 0 }"> test 조건식이 false 이므로,

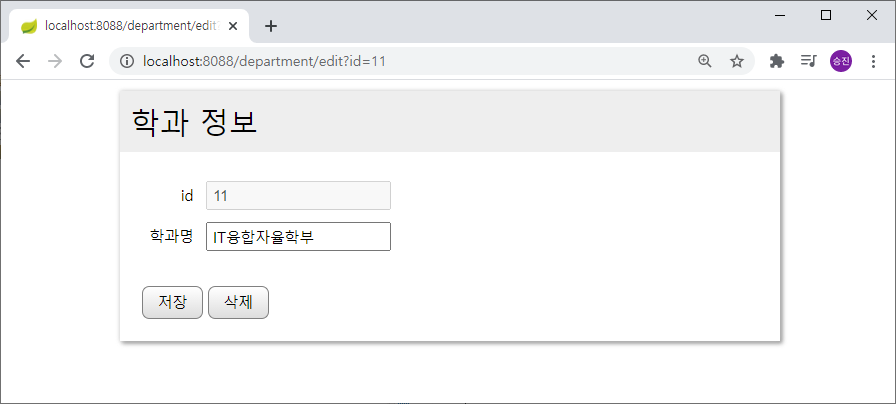
삭제 링크 태그가 웹브라우저로 출력되지 않는다.

class="btn" 서식 클래스 부분에 주목하자.

이 서식은 common.js 파일에 정의되어 있다.

이 서식이 적용된 태그는 모서리가 둥근 버튼 형태로 웹브라우저창에 그려진다.

button 태그 뿐만 아니라 a 태그에도 이 서식을 적용할 수 있다.

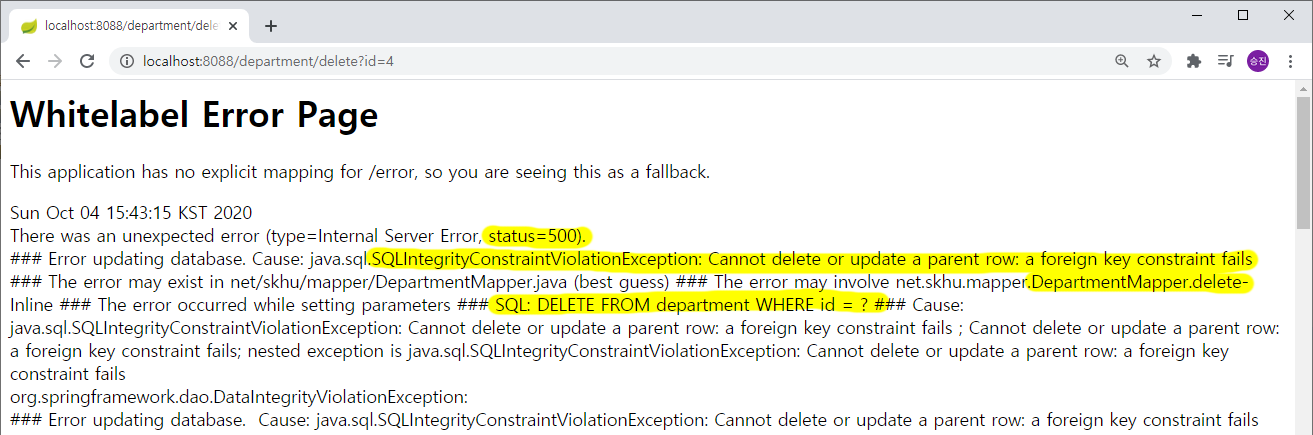


## 참조 무결성 제약 위반 에러

referential integrity constraint violation error

새로 등록한 학과는 삭제 잘 되지만, DB에 이미 있던 학과를 삭제하면

참조 무결성 위반 에러가 발생한다.



Java에서 실행도중 exception이 발생하면, HTTP 상태 코드 500 에러가 출력된다.

SQLIntegrityConstraintViolationException: Cannot delete or update a parent row: a foreign key constraint fails

foreign key로 참조되고 있는 레코드를 삭제할 수 없다는 에러이다.

쉽게 말해서, 소프트웨어공학과 소속 학생 레코드가 있으니, 소프트웨어공학과 레코드를 삭제할 수 없다는 에러이다. 이 소속 관계를 정의하는 것이 student 태이블의 departmentId 외래키(foreign key)이다.

새로 등록한 학과는 소속 학생이 없으니 에러 없이 삭제할 수 있다.