학생 한 명 조회하기

2021-11-07

이승진

목차

[1. SQL 2](#_Toc87251173)

[2. mapper 구현 3](#_Toc87251174)

[1) StudentMapper.java 수정 3](#_Toc87251175)

[2) 조회 결과 레코드의 수 3](#_Toc87251176)

[3. controller 구현 4](#_Toc87251177)

[1) query string과 request parameter 4](#_Toc87251178)

[2) StudentController.java 수정 5](#_Toc87251179)

[4. 뷰 구현 6](#_Toc87251180)

[1) student/detail.jsp 6](#_Toc87251181)

[5. 실행 7](#_Toc87251182)

[6. 구현 과제 8](#_Toc87251183)

# SQL

Graphical user interface, text, application, email

Description automatically generated

|  |
| --- |
| use student2;  SELECT s.\*, d.name departmentName  FROM student s LEFT JOIN department d ON s.departmentId = d.id  WHERE s.id = 5 |

student 테이블의 primery key인 id로 학생을 조회한다.

학생 한 명이 조회된다.

# mapper 구현

## StudentMapper.java 수정

src/main/java/net/skhu/mapper/StudentMapper.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 | package net.skhu.mapper;  import java.util.List;  import org.apache.ibatis.annotations.Mapper;  import org.apache.ibatis.annotations.Select;  import net.skhu.dto.Student;  @Mapper  public interface StudentMapper {  @Select("SELECT s.\*, d.name departmentName " +  "FROM student s JOIN department d ON s.departmentId = d.id")  List<Student> findAll();  @Select("SELECT s.\*, d.name departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id " +  "WHERE s.name LIKE #{name}")  List<Student> findByName(String name);  @Select("SELECT s.\*, d.name departmentName " +  "FROM student s LEFT JOIN department d ON s.departmentId = d.id " +  "WHERE s.id = #{id}")  Student findById(int id);  } |

StudentMapper에 findById 메소드를 구현하였다.

이 메소드의 SQL 명령은, student 테이블에서 id 필드 값으로 학생 한 명을 조회한다.

이 메소드의 파라미터는 조회할 학생의 id 필드 값이다.

이 메소드 파라미터 값이 SQL 명령의 #{id} 부분에 채워져서, SQL 명령이 실행된다.

id 필드가 기본 키(primary key) 이기 때문에, 이 메소드의 조회 결과는 레코드 한 개이다.

그래서 이 메소드가 리턴하는 것도 Student 객체 한 개이다.

StudentByName 메소드와 StudentById 메소드의 구조를 비교해 보자.

- 조회 조건이 다르고

- 리턴 타입이 다르다.

## 조회 결과 레코드의 수

findByName 메소드의 리턴 타입은 List<Student> 이다.

SQL 명령의 조회 결과 레코드 수가 0 이면, 이 메소드는 크기가 0 인 List<Student> 객체를 리턴한다.

SQL 명령의 조회 결과 레코드 수가 1 이면, 이 메소드는 크기가 1 인 List<Student> 객체를 리턴한다.

SQL 명령의 조회 결과 레코드 수가 2 이면, 이 메소드는 크기가 2 인 List<Student> 객체를 리턴한다.

findById 메소드의 리턴 타입은 Student 이다.

SQL 명령의 조회 결과 레코드 수가 0 이면, 이 메소드는 null 값을 리턴한다.

SQL 명령의 조회 결과 레코드 수가 1 이면, 이 메소드는 Student 객체를 리턴한다.

SQL 명령의 조회 결과 레코드 수가 2 이면, 이 메소드에서 에러(exception)가 발생한다.

# controller 구현

## query string과 request parameter

### query string

서버에 요청되는 URL이 아래와 같을 경우에

|  |
| --- |
| http://localhost:8088/student/detail |

위 URL에서 query string 부분은 없다.

서버에 요청되는 URL이 아래와 같을 경우에

|  |
| --- |
| http://localhost:8088/student/detail?id=5 |

위 URL에서 query string 부분은 ?id=5 이다.

여기서 id 부분이 이름이고, 5 부분이 값이다.

### request parameter

query string으로 전달된 데이터를, 서버에서 꺼낼 때는, request parameter라고 부른다.

즉 URL에서 query string 부분은, 서버에 request parameter 형태로 전달된다.

request parameter 데이터는 이름이 같은 액션 메소드의 파라미터 변수에 자동으로 채워져 전달된다.

## StudentController.java 수정

src/main/java/net/skhu/controller/StudentController.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 | package net.skhu.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 net.skhu.mapper.StudentMapper;  @Controller  public class StudentController {  @Autowired StudentMapper studentMapper;  @RequestMapping("student/list")  public String list(Model model) {  model.addAttribute("students", studentMapper.findAll());  return "student/list";  }  @RequestMapping("student/list1")  public String list1(Model model, String srchText) {  if (srchText == null) srchText = "";  model.addAttribute("students", studentMapper.findByName(srchText + "%"));  model.addAttribute("srchText", srchText);  return "student/list1";  }  @RequestMapping("student/detail")  public String detail(Model model, Integer **id**) {  if (**id** == null) **id** = 5;  model.addAttribute("student", studentMapper.findById(**id**));  return "student/detail";  }  } |

StudentController에 detail 액션 메소드를 구현하였다.

이 메소드의 파라미터는 Integer id 이다.

"id" 이름의 request parameter 값이 이 파라미터 변수에 채워져 전달된다.

이 이름의 request parameter가 없다면, 이 파라미터 변수에 null 값이 채워져 전달된다.

request parameter에 "id" 이름의 데이터가 없다면, id 필드값이 5 인 학생이 조회되고,

있다면 그 값과 id 필드값이 일치하는 학생이 조회된다.

조회된 Student 객체는 "student" 이름의 model attribute 가 되어,

student/detail.jsp 뷰에 전달된다.

# 뷰 구현

## student/detail.jsp

src/main/webapp/WEB-INF/views/student/detail.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  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57 | <%@ 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">  <style>  div.container { width: 600px; margin: 50px auto; }  body { font-family: 굴림체; }  table { width: 500px; border-collapse: collapse; }  table td:nth-child(1) { background-color: #eee; }  td { padding: 5px; border: 1px solid lightgray; }  </style>  </head>  <body>  <div class="container">  <h1>학생</h1>  <table>  <tr>  <td>ID</td>  <td>${ student.id }</td>  </tr>  <tr>  <td>학번</td>  <td>${ student.studentNo }</td>  </tr> <tr>  <td>이름</td>  <td>${ student.name }</td>  </tr>  <tr>  <td>학과ID</td>  <td>${ student.departmentId }</td>  </tr>  <tr>  <td>학과명</td>  <td>${ student.departmentName }</td>  </tr>  <tr>  <td>전화</td>  <td>${ student.phone }</td>  </tr>  <tr>  <td>성별</td>  <td>${ student.sex }</td>  </tr>  <tr>  <td>이메일</td>  <td>${ student.email }</td>  </tr>  </table>  </div>  </body>  </html> |

"student" 이름의 model attribute 값이 출력된다.

# 실행

### http://localhost:8088/student/detail

Table

Description automatically generated

### http://localhost:8088/student/detail?id=3

Table

Description automatically generated

# 구현 과제

### book3 데이터베이스

book3.sql 파일을 import 하자.

이전 강의노트 참고 (01 MySQL 설치)

Diagram

Description automatically generated

book 테이블의 categoryId 필드가 외래키이다.

### book3 프로젝트를 만들자.

Graphical user interface, application, email

Description automatically generated

메뉴: File - New - Spring Starter Project

mybatis1 프로젝트 생성 강의노트 참고 (02 데이터목록조회)

### application.properties

|  |
| --- |
| 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/book3?useUnicode=yes&characterEncoding=UTF-8&allowMultiQueries=true&serverTimezone=Asia/Seoul  spring.datasource.username=user1  spring.datasource.password=skhuA+4.5  server.port=8088 |

데이터베이스 이름: book3

### book 테이블 조회

(1) 전체 목록 조회를 구현하자.

http://localhost:8088/book/list

Table

Description automatically generated

(2) 책 한 개 조회를 구현하자.

http://localhost:8088/book/detail?id=2

Table

Description automatically generated