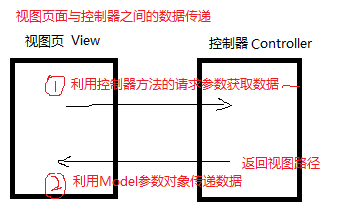
课程目标:

1. 总结控制器与视图之间的数据传递
2. (会)视图解析器
3. 综合练习:springmvc+mybatis采用三层架构实现功能

>>实现查询所有学生业务

>>实现学生的添加业务

1. (重点理解)总结控制器与视图页面之间的数据传递



*//1.在控制器方法参数中添加Model对象进行数据传递  
 //model.addAttribute("键",值);* model.addAttribute(**"userinfo"**,userInfo);

1. 配置springmvc的视图解析器
2. 修改springmvc配置文件定义视图解析器

|  |
| --- |
| *<!--配置视图解析器-->* <**bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  *<!--前缀-->* <**property name="prefix" value="/"**></**property**>  *<!--后缀-->*  <**property name="suffix" value=".jsp"**></**property**> </**bean**> |

1. 在控制器方法中返回视图名称即可(针对请求转发跳转)

**return "success"**; *//返回视图名称(文件名)*

*结果：*

*return “success.jsp”*

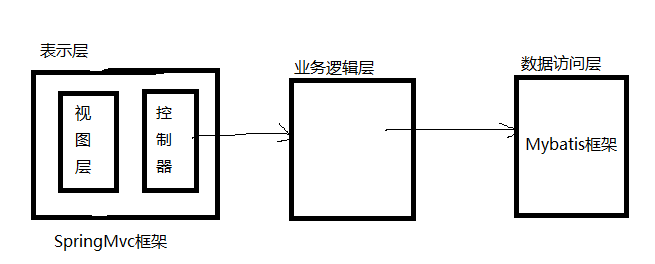
扩展:以重定向的方式返回页面

（控制器就不能使用model给页面返回数据、视图解析也不起作用啦）

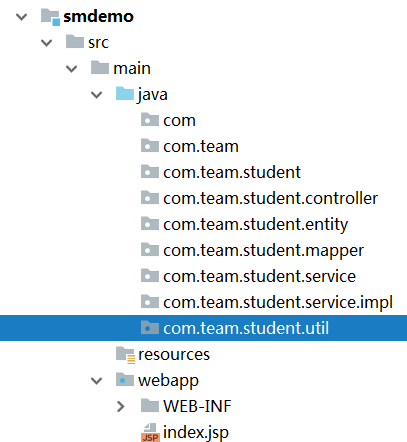
return “redirect:返回视图页面的路径”;

1. 采用三层架构实现学生查询

技术:springmvc+mybatis



1. 创建maven工程

.

1. (关键)搭建springmvc和mybatis的开发环境
   1. 搭建mybatis的开发环境

导入依赖

|  |
| --- |
| *<!-- MySQL依赖 start -->* <**dependency**>  <**groupId**>mysql</**groupId**>  <**artifactId**>mysql-connector-java</**artifactId**>  <**version**>5.1.25</**version**> </**dependency**> *<!-- MySQL依赖 end -->*  *!-- 加入MyBatis 依赖 start -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis</**artifactId**>  <**version**>3.2.5</**version**> </**dependency**> *<!-- 加入MyBatis 依赖 end --> <!-- mybatis自动生成所需的依赖start -->* <**dependency**>  <**groupId**>org.mybatis.generator</**groupId**>  <**artifactId**>mybatis-generator-core</**artifactId**>  <**version**>1.3.3</**version**> </**dependency**> *<!-- mybatis自动生成所需的依赖end -->* |

注意:解决maven打包问题

|  |
| --- |
| *<!--配置相关的资源进行打包-->* <**resources**>  <**resource**>  <**directory**>src/main/java</**directory**>  <**includes**>  <**include**>\*\*/\*.xml</**include**>  </**includes**>  </**resource**>  <**resource**>  <**directory**>src/main/resources</**directory**>  <**includes**>  <**include**>\*\*/\*.\*</**include**>  </**includes**>  </**resource**> </**resources**> |

添加mybatis相关的配置文件并做出相应修改

|  |
| --- |
| <**configuration**>  *<!--配置实体类 类型的别名-->* <**typeAliases**>  <**package name="com.kgc.project.entity"**></**package**>  </**typeAliases**>   *<!-- 指定义连接数据的配置 -->* <**environments default="development"**>  <**environment id="development"**>  <**transactionManager type="JDBC"**/>  <**dataSource type="POOLED"**>  <**property name="driver" value="com.mysql.jdbc.Driver"**/>  <**property name="url" value="jdbc:mysql://localhost:3306/k9051db?useUnicode=true&amp;characterEncoding=utf-8"**/>  <**property name="username" value="root"**/>  <**property name="password" value="123"**/>  </**dataSource**>  </**environment**>  </**environments**>   *<!-- 加载sql映射文件 -->* <**mappers**>  *<!--一个一个加载-->  <!--<mapper resource="com/team/dao/GradeMapper.xml"/>  <mapper resource="com/team/dao/StudentsMapper.xml"/>-->  <!--加载指定包下的所有sql映射文件-->* <**package name="com.kgc.project.mapper"**></**package**>  </**mappers**> </**configuration**> |

* 1. 搭建springmvc的开发环境

导入依赖

|  |
| --- |
| *<!-- 初始化框架的版本号 -->* <**properties**>  <**spring.version**>4.3.3.RELEASE</**spring.version**> </**properties**>  *<!--spring支持的相关和springmvc依赖(spring-web、spring-webmvc)-->* <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-web</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-webmvc</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-core</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-oxm</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-tx</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-jdbc</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-aop</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-context-support</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**>  <**dependency**>  <**groupId**>org.springframework</**groupId**>  <**artifactId**>spring-test</**artifactId**>  <**version**>${spring.version}</**version**> </**dependency**> *<!-- 引入Spring 依赖 end -->  <!--添加jstl的依赖--> <!-- JSTL -->* <**dependency**>  <**groupId**>jstl</**groupId**>  <**artifactId**>jstl</**artifactId**>  <**version**>1.2</**version**> </**dependency**> *<!-- 加入ServletAPI -->* <**dependency**>  <**groupId**>javax.servlet</**groupId**>  <**artifactId**>servlet-api</**artifactId**>  <**version**>2.3</**version**>  <**scope**>provided</**scope**> </**dependency**> |

添加springmvc配置文件,并做出相应修改

|  |
| --- |
| *<!--使用component-scans标签扫描springmvc注解配置-->* <**context:component-scan base-package="com.kgc.project.controller"**></**context:component-scan**> ***<****!--配置视图解析器-->* <**bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  *<!--前缀-->* <**property name="prefix" value="/"**></**property**>  *<!--后缀-->* <**property name="suffix" value=".jsp"**></**property**> </**bean**> |

修改web.xml文件(配置核心控制器，解决乱码问题)

|  |
| --- |
| *<!--核心控制器  作用1:拦截用户发请求，根据请求的名称，去springmvc配置文件中找到对应的控制器进行分发  作用2:读取配置文件，加载所有请求 -->* <**servlet**>  <**servlet-name**>springmvc</**servlet-name**>  <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  <**init-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:springmvc.xml</**param-value**>  </**init-param**>  <**load-on-startup**>1</**load-on-startup**> </**servlet**> <**servlet-mapping**>  <**servlet-name**>springmvc</**servlet-name**>  *<!--拦截的请求-->* <**url-pattern**>/</**url-pattern**> </**servlet-mapping**>   *<!--配置解决乱的过滤器-->* <**filter**>  <**filter-name**>encodingFilter</**filter-name**>  <**filter-class**>org.springframework.web.filter.CharacterEncodingFilter</**filter-class**>  <**init-param**>  <**param-name**>encoding</**param-name**>  <**param-value**>utf-8</**param-value**>  </**init-param**>  </**filter**>  <**filter-mapping**>  <**filter-name**>encodingFilter</**filter-name**>  <**url-pattern**>/\*</**url-pattern**>  </**filter-mapping**> |

1. 实现mybatis的反向工程:生成实体，接口，sql映射文件

略

注意:开发项目的功能时，基础

1. 实现查询所有学生信息
2. 编写dao层

采用自动生成，已经存在持久化操作(略)

注意:如果没有持久化操作需要自己编写持久化

1. 编写业务层

一业务一功能一方法

接口方法：

|  |
| --- |
| **public interface** StudentService {  *//一业务一功能一方法* **public** List<Students> getAllStudent(); } |

实现:

|  |
| --- |
| @Override **public** List<Students> getAllStudent() {  *//调用dao层，省了dao实现，需要用getMapper生成dao实现* SqlSession sqlSession=MyBatisUtil.*getSession*();  StudentsMapper studentsMapper=sqlSession.getMapper(StudentsMapper.**class**);  *//调用dao层方法* List<Students> list= studentsMapper.selectByExample(**new** StudentsExample());  **return** list; } |

1. 编写表示层

控制器:

|  |
| --- |
| @Controller **public class** StudentController {  *//创建业务对象* StudentService **studentService**=**new** StudentServiceImpl();  @RequestMapping(**"/showStudent"**)  **public** String showStudent(Model model){  *//调用业务* List<Students> list=**studentService**.getAllStudent();  *//利用model对象将数据传递到页面* model.addAttribute(**"list"**,list);  **return "index"**; *//视图解析支持* } } |

Jsp视图展示：

|  |
| --- |
| <%@**page language**="**java**" **contentType**="**text/html;utf-8**" **pageEncoding**="**utf-8**" %> <%@ **taglib prefix**="**c**" **uri**="**http://java.sun.com/jsp/jstl/core**" %> <**html**> <**body**> <**table**>  <**tr**>  <**td**>学号</**td**>  <**td**>姓名</**td**>  <**td**>年龄</**td**>  <**td**>性别</**td**>  <**td**>地址</**td**>  <**td**>状态</**td**>  <**td**>生日</**td**>  </**tr**>  <**c:forEach items="${**list**}" var="s"**>  <**tr**>  <**td**>**${**s.xh**}**</**td**>  <**td**>**${**s.name**}**</**td**>  <**td**>**${**s.age**}**</**td**>  <**td**>**${**s.sex**}**</**td**>  <**td**>**${**s.address**}**</**td**>  <**td**>**${**s.state**}**</**td**>  <**td**>**${**s.birthday**}**</**td**>  </**tr**>  </**c:forEach**> </**table**> </**body**> </**html**> |

1. 添加学生业务
2. 编写dao层

存在添加的持久化方法。略

1. 编写业务层

接口:

|  |
| --- |
| *//添加学生* **int** addStudent(Student student); |

实现:

|  |
| --- |
| @Override **public int** addStudent(Student student) {  *//调用dao层的持久化操作  //获取dao层对象* **int** temp=-1;  **try** {  SqlSession sqlSession=MyBatisUtil.*getSession*();  StudentMapper studentMapper=sqlSession.getMapper(StudentMapper.**class**);  *//调用方法* temp=studentMapper.insertSelective(student);  }**catch** (Exception e) {  e.printStackTrace();  } **finally** {  MyBatisUtil.*closeSession*();  }  **return** temp; } |

1. 编写控制层

|  |
| --- |
| *//处理添加学生的请求* @RequestMapping(**"/addStudent"**) **public** String addStudent(Student stu){  *//定义学生实体参数，接收所有的学生信息 保证:属性的名称与表单对象的名称一样  //调用业务* **int** temp=**this**.studentService.addStudent(stu);  **if**(temp>0)  **return "redirect:showStudent"**; *//跳转控制器* **else  return "addStudent"**; } |

1. 页面

|  |
| --- |
| *<%--  Created by IntelliJ IDEA.  User: Administrator  Date: 2020/9/6  Time: 11:18  To change this template use File | Settings | File Templates. --%>* <%@ **page contentType**="**text/html;charset=UTF-8**" **language**="**java**" %> <**html**> <**head**>  <**title**>Title</**title**> </**head**> <**body**> 添加学生 <**form id="form1" name="form1" method="post" action="addStudent"**>  <**p**>学号:  <**input type="text" name="xh" id="ss"** />  </**p**>  <**p**>姓名:  <**input type="text" name="name" id="textfield"** />  </**p**>  <**p**>年龄:  <**input type="text" name="age" id="textfield2"** />  </**p**>  <**p**>性别:  <**input name="sex" type="radio" id="radio" value="男" checked="checked"** />  男  <**input type="radio" name="sex" id="radio2" value="女"** />  女  </**p**>  <**p**>生日:  <**input type="date" name="birthday"**>  </**p**>  <**p**>地址:  <**input type="text" name="address" id="textfield4"** />  </**p**>  <**p**>状态:  <**input type="text" name="state" id="textfield5"** />  </**p**>  <**p**>年级:  <**input type="text" name="gid" id="textfield6"** />  下拉列表，从数据库中查询年级  </**p**>  <**p**>  <**input type="submit" name="button" id="button" value="添加"** />  </**p**> </**form**> </**body**> </**html**> |

问题:springmvc进行参数绑定时，不支持日期(java.util.Date)转换.

解决:

1. 在实体属性上使用DateTimeFormat解决日期转换的支持

@DateTimeFormat(pattern = **"yyyy-MM-dd"**)  
 **private** Date **birthday**;

1. 在spingmvc配置文件中添加注解驱动配置

*<!--注解驱动-->*<**mvc:annotation-driven**></**mvc:annotation-driven**>