课程目标:

1. (掌握)优化数据源的配置
2. (掌握)配置ssm整合的事务支持（spring事务配置）
3. 查询年级
4. 添加学生信息
5. 优化数据源的配置:jdbc连接的参数定义在属性文件中
6. 定义jdbc.properties属性文件

|  |
| --- |
| **jdbc.driverClass**=**com.mysql.jdbc.Driver jdbc.url**=**jdbc:mysql://localhost:3306/k0507db?useUnicode=true&characterEncoding=utf-8 jdbc.username**=**root jdbc.password**=**123** |

1. 在spring配置文件中使用**PropertyPlaceholderConfigurer类加载属性文件**

|  |
| --- |
| <**bean id="configurer" class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer"**>  <**property name="locations"**>  <**list**>  <**value**>classpath:jdbc.properties</**value**>  </**list**>  </**property**> </**bean**> |

1. 在spring配置使用${属性文件中的键}引用属性文件中的值

|  |
| --- |
| <**bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"**>  <**property name="driverClassName" value="${jdbc.driverClass}"**></**property**>  <**property name="url" value="${jdbc.url}"**></**property**>  <**property name="username" value="${jdbc.username}"**></**property**>  <**property name="password" value="${jdbc.password}"**></**property**> </**bean**> |

1. 查询所有年级（强化SSM开发流程）
2. 编写数据访问层

略

1. 编写业务层

接口:

|  |
| --- |
| **public interface** GradeService {  List<Grade> getAllGrade(); } |

实现:

|  |
| --- |
| @Service **public class** GradeServiceImpl **implements** GradeService {  *//定义dao层接口,引用入dao实现对象* @Autowired  **private** GradeMapper **gradeMapper**;    @Override  **public** List<Grade> getAllGrade() {  **return this**.**gradeMapper**.selectByExample(**new** GradeExample());  } } |

1. 编写控制器

|  |
| --- |
| @Controller **public class** GradeController {  @Autowired  **private** GradeService **gradeService**;  @RequestMapping(**"/showGrade"**)  **public** String showGrade(Model model){  *//调用业务* List<Grade> list=**gradeService**.getAllGrade();  model.addAttribute(**"list"**,list);  **return "grade"**;  } } |

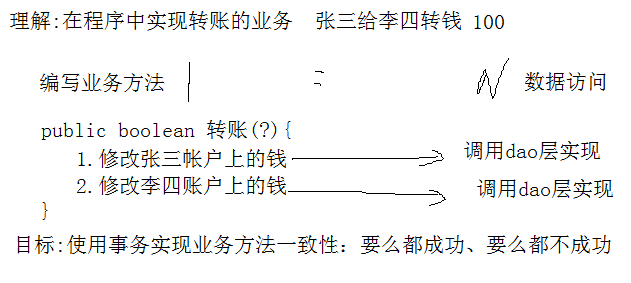
1. 编写视图

|  |
| --- |
| <**table border="2"**>  <**tr**>  <**td**>编号</**td**>  <**td**>年级名称</**td**>  <**td**>操作</**td**>  </**tr**>  <**c:forEach items="${**list**}" var="g"**>  <**tr**>  <**td**>**${**g.gid**}**</**td**>  <**td**>**${**g.gname**}**</**td**>  <**td**>删除</**td**>  </**tr**>  </**c:forEach**> </**table**> |

1. 配置ssm框架整合的事务支持
2. 事务分类:

数据库事务、程序事务

1. 理解程序中事务

注意:ssm事务是添加在业务实现上的

1. 实现**事务支持的注解配置**
   1. 修改spring配置文件添加事务支持的配置

|  |
| --- |
| *<!--配置Spring的事务管理器 -->* <**bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  <**property name="dataSource" ref="dataSource"**/> </**bean**> *<!-- 注解方式配置事物 -->* <**tx:annotation-driven transaction-manager="transactionManager"**></**tx:annotation-driven**> |

注意:在spring配置文件中导入tx标签

**xmlns:tx="http://www.springframework.org/schema/tx"  
xsi:schemaLocation="   
http://www.springframework.org/schema/tx  
http://www.springframework.org/schema/tx/spring-tx-4.3.xsd"**

* 1. 在业务类中使用@transactional注解支持事务

@Transactional 注意:放在类上表示类中所有方法都基于事务执行

public class TestServiceBean implements TestService {….}

当类中某些方法不需要事物时:

@Transactional

public class TestServiceBean implements TestService {

//挂起事务，不基地事务执行

@Transactional(propagation = Propagation.NOT\_SUPPORTED)

public List<Object> getAll() {

return null;

}

}

注意:

1. 添加事务支持的业务方法中不能使用try.catch语句,因为事务实现是业务方法报错 时进切面的事务增强处理.
2. @Transactional的propagation属性，表示事务的传播机制

了解:Spring中七种Propagation类的事务属性详解：

 REQUIRED：支持当前事务，如果当前没有事务，就新建一个事务。这是最常见的选择。

 SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行。

 MANDATORY：支持当前事务，如果当前没有事务，就抛出异常。

 REQUIRES\_NEW：新建事务，如果当前存在事务，把当前事务挂起。

 NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

 NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

 NESTED：支持当前事务，如果当前事务存在，则执行一个嵌套事务，如果当前没有事务，就新建一个事务。

1. (掌握)在maven工程中编写主函数测试业务组件

|  |
| --- |
| **public class** StudentServiceImpl  {  **public static void** main(String[] args) {  *//1.从spring容器中获取业务组件对象* ApplicationContext ctx=**new** ClassPathXmlApplicationContext(**"applicationContext.xml"**);  StudentService studentService=(StudentService)ctx.getBean(**"studentServiceImpl"**);  *//2.调用业务方法* studentService.delMoreStudent();  System.***out***.println(**"成功执行"**);  } } |

1. 删除年级和同时删除学生
2. 删除年级和同时删除学生
   1. 编写dao层

删除年级的方法:略

删除学生的方法:

接口:

**int** deleteStudentByGrade(Integer gid);

持义化操作:

<**delete id="deleteStudentByGrade" parameterType="int"**>  
 DELETE FROM student WHERE gid=#{gid}  
 </**delete**>

* 1. 编写业务层

接口方法:

|  |
| --- |
| *//删除年级的同时删除学生* **public void** delGradeStudent(Integer gid); |

实现:

|  |
| --- |
| @Service @Transactional *//支持事务，要么都成功，要么都不成功* **public class** GradeServiceImpl **implements** GradeService {  *//定义dao层接口,引用入dao实现对象* @Autowired  **private** GradeMapper **gradeMapper**;  @Autowired  **private** StudentMapper **studentMapper**;  @Transactional(propagation =Propagation.***NOT\_SUPPORTED*** ) *//不基于事务执行* **public** List<Grade> getAllGrade()  {  **return  this**.**gradeMapper**.selectByExample(**new** GradeExample());  }    *//删除年级和对应的学生* @Override  @Transactional   **public void** delGradeStudent(Integer gid) {  *//1.删除年级* **gradeMapper**.deleteByPrimaryKey(gid);  //**int** i=0,j=9;  // j=j/i  *//2.删除学生* **studentMapper**.deleteStudentByGrade(gid);  } } |

* 1. 编写控制层

|  |
| --- |
| @Controller **public class** GradeController {  @Autowired  **private** GradeService **gradeService**;  @RequestMapping(**"/delGrade"**)  **public** String delGrade(Integer gid){  *//调用业务* **try** {  **gradeService**.delGradeStudent(gid);  }**catch** (Exception e){  e.printStackTrace();  }  **return "redirect:showGrade"**;  } } |

1.4 在显示年级的页面添加删除的导航跳转到控制器

1. 添加学生

1. 编写视图: 添加学生表单 **addStudent.jsp**

|  |
| --- |
| <**form id="form1" name="form1" method="post" action="addStu"**>  <**p**>学号:  <**input type="text" name="xh" id="textfield"** />  </**p**>  <**p**>姓名:  <**input type="text" name="name" id="textfield2"** />  </**p**>  <**p**>性别:  <**input type="radio" name="sex" value="男"**/>男  <**input type="radio" name="sex" value="女"** />女  </**p**>  <**p**>年龄:  <**input type="text" name="age" id="textfield4"** />  </**p**>  <**p**>生日:  <**input type="date" name="birthday" id="textfield5"** />  </**p**>  <**p**>地址:  <**input type="text" name="address" id="textfield6"** />  </**p**>   <**p**>年级:  <**select name="gid"**>  <**c:forEach items="${**list**}" var="g"**>  <**option value="${**g.gid**}"**>**${**g.gname**}**</**option**>  </**c:forEach**>  </**select**>  <**input type="submit" name="button" id="button" value="提交"** />  </**p**> </**form**> |

1. 添加学生
   1. 编写dao层

略

* 1. 编写业务层

接口方法:

|  |
| --- |
| **public int** addStudent(Students students); |

实现 ：

|  |
| --- |
| @Override **public int** addStudent(Students students) {  **return studentsMapper**.insertSelective(students); } |

* 1. 编写控制器

|  |
| --- |
| @RequestMapping(**"/addStudent"**) *//属性名与表音对象名称对应* **public** String addStudent(Students stu) {  *//设置在读状态为:0* stu.setState(**new** Byte(**"0"**));  *//调用业务* **int** temp = **studentService**.addStudent(stu);  **if** (temp > 0) *//去显示* **return "redirect:showStudents"**;  **else  return "redirect:showGrade"**; } |