课程目标:

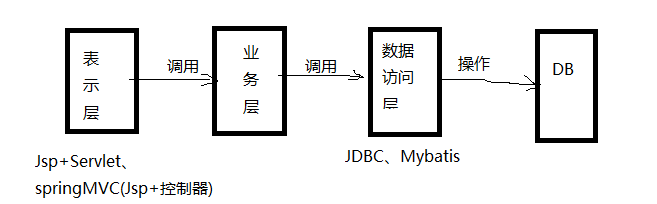
1. Spring+Springmvc+Mybatis三大框架整合(简称为:SSM)

>>理解ssm整合的思路

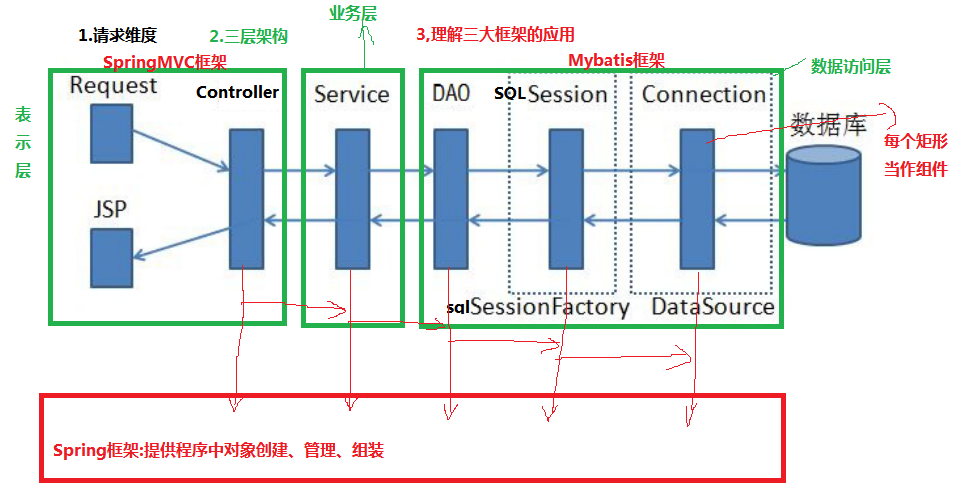
>>spring整合Mybatis框架

>>spring整合SpringMVC

1. 结合SSM实现学生查询
2. 理解SSM三大框架
3. 回顾三层架构

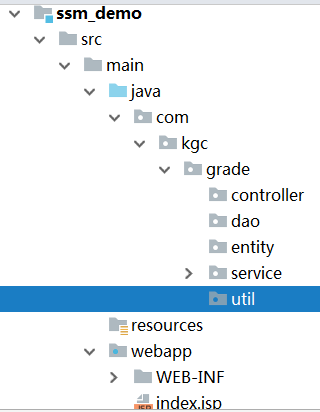


1. SSM整合思路



* **SSM架构**
  1. **Spring + SpringMVC + Mybatis**
  2. **以Spring作为核心框架，数据持久化使用Mybatis完成，表现层使用springmvc**
  3. **Spring提供对象管理、面向切面编程等实用功能**
  4. **通过Spring提供的服务简化编码、降低开发难度、提高开发效率**

1. 使用ssm实现学生查询
2. 创建工程



1. 在项目中添加三大框架的支持(重点)
   1. 添加三大框架的依赖：mybatis、springmvc、spring

参考文件

* 1. 添加三大框架的配置文件

>>添加mybatis配置文件并修改

定义实体别名、配置数据库(不要啦)、加载sql映射

>>添加sprngmvc的配置文件并修改

修改web.xml文件(配置核心控制器、乱码过滤器)

>>添加spring的配置文件

1. 反向工程

生成实体，接口，sql映射文件

1. 修改pom文件

添加resources标签用于解决java目录中的xml文件打包问题

添加tomcat插件

|  |
| --- |
| *<!--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tomcat插件-->* <**plugins**>  <**plugin**>  <**groupId**>org.apache.tomcat.maven</**groupId**>  <**artifactId**>tomcat7-maven-plugin</**artifactId**>  <**version**>2.2</**version**>  *<!--如果configuration不加，默认商品为8080  path 项目名称  port 表示端口  servlet 表示服务器名称tomcat7  -->* <**configuration**>  <**path**>/</**path**>  <**port**>8080</**port**>  <**server**>tomcat7</**server**>  </**configuration**>  </**plugin**> </**plugins**> |

1. Spring整合Mybatis的步骤(重点):在applicationContext.xml配置文件中添加以下配置

必需导入: mybatis-spring依赖包

|  |
| --- |
| *<!--spring整合mybatis-->  <!--1.定义数据源组件c3p0 作用：用于连接数据库-->* <**bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"**>  <**property name="driverClassName" value="com.mysql.jdbc.Driver"**></**property**>  <**property name="url" value="jdbc:mysql://localhost:3306/k9051db?useUnicode=true&amp;characterEncoding=utf-8"**></**property**>  <**property name="username" value="root"**></**property**>  <**property name="password" value="123"**></**property**> </**bean**> *<!--2.定义sqlSessionFactory工厂组件-->* <**bean id="sqlSessionFatory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  *<!--指定数据源-->* <**property name="dataSource" ref="dataSource"**></**property**>  *<!--指定mybatis的配置文件-->* <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**> </**bean**> *<!--3.spring接管dao层组件 (理解为创建dao层实现类的对象)  注意:1. sql映射的命名空间必需是接口的限定名，持久化操作的id值必需和接口方法中相同  2. 动态生成实现类的bean对象id值，就是接口名称首字母小写 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  *<!--指定sqlSessionFacory对象-->* <**property name="sqlSessionFactoryBeanName" value="sqlSessionFatory"**></**property**>  *<!--扫描dao层接口，动态生成实现类的对象-->* <**property name="basePackage" value="com.kgc.grade.dao"**></**property**> </**bean**> |

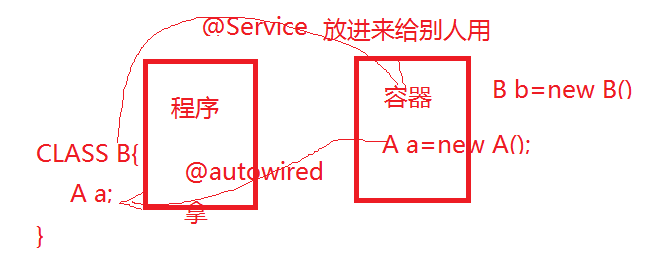
1. 开发业务组件，并交给spring容器接管
2. 在spring配置文件中添加扫描业务组件对象

|  |
| --- |
| *<!--扫描业务组件配置 注意:扫描业务实现类的对象-->* <**context:component-scan base-package="com.kgc.grade.service.impl"**></**context:component-scan**> |

1. 开发业务组件

|  |
| --- |
| 接口:  **public interface** StudentService {  **public** List<Students> getAllStudent(); }  实现:  //@Service(value = "studentSerivce") //表示业务组件 组件id为"studentSerivce" @Service //表示业务组件 组件id为实现类的首字小写 **public class** StudentServiceImpl **implements** StudentService {  //声明dao层接口属性  //使用@Autowired进行自动注入 默认按类型  //从容器中找相应接口的对象进行自动赋值  @Autowired  **private** StudentsMapper **studentsMapper**;  @Override  **public** List<Students> getAllStudent() {  **return this**.**studentsMapper**.selectByExample(**new** StudentsExample());  } } |

理解@Service注解和@Autowire注解



1. 编写控制器控制器.(spring整合springmvc无缝集成)

注意:在springmvc配置文件中添加控制器扫描的配置

|  |
| --- |
| @Controller **public class** StudentController {  *//定义业务接口的属性* @Autowired(required = **false**) *//自动注入(省略setter和getter方法) 按类型区配* **private** StudentService **studentService**;   @RequestMapping(**"showStudent"**)  **public** String showStudent(Model model){  *//调用业务* List<Students> list=**studentService**.getAllStudent();  *//填充model* model.addAttribute(**"list"**,list);  **return "index"**;  } } |

1. 编写视图层代码

|  |
| --- |
| 学生信息如下:<**br**/> <**table**>  <**tr**>  <**td**>学号</**td**>  <**td**>姓名</**td**>  <**td**>年龄</**td**>  <**td**>性别</**td**>  <**td**>地址</**td**>  <**td**>生日</**td**>  <**td**>状态</**td**>  </**tr**>  <**c:forEach items="${**list**}" var="l"**>  <**tr**>  <**td**>**${**l.xh**}**</**td**>  <**td**>**${**l.name**}**</**td**>  <**td**>**${**l.age**}**</**td**>  <**td**>**${**l.sex**}**</**td**>  <**td**>**${**l.address**}**</**td**>  <**td**>**${**l.birthday**}**</**td**>  <**td**>**${**l.state**}**</**td**>  </**tr**>  </**c:forEach**> </**table**> |

1. 在web.xml文件中使用监听器启动spring容器

|  |
| --- |
| *<!--启动spring容器--> <!--指定spring配置文件的位置-->* <**context-param**>  <**param-name**>contextConfigLocation</**param-name**>  <**param-value**>classpath:applicationContext.xml</**param-value**> </**context-param**> <**listener**>  <**listener-class**>org.springframework.web.context.ContextLoaderListener</**listener-class**> </**listener**> |