1. 版本控制工具

Hub

1. 开发工具

Ide

1. 数据库

Mysql

1. 框架

Spring+springmvc+mybatis

1. 工具

Maven等… jdk1.8

表分析

商品表

用户表

…..

在需要用的文件夹下右键 git bash here

GitHub使用命令记录

Git init 初始化git

Git status 查看状态（可以查看出当前文件夹下的文件）

解决乱码名称：git config --global core.quotepath false

上传文件 git add “文件名称”或者git add –A上传当前目录下所有文件

可以 git commit -m "更新的信息"  来提交

Git log查看日志

Git reset –hard 版本号前7位 版本回退

git clean -xf  删除没有上传的文件

本地关联GitHub 运行 git remote add origin （github中的地址）

git push -u origin master 查看是否关联成功

以后如果想更新文件到github 提交（ git commit -m "更新的信息"  来提交）完后输入 git push就可以同步了

Git

1. 初始化
2. 关联
3. 添加到暂存区
4. 提交到仓库
5. 同步

Idea

关闭一个项目 file-----colse project

快捷键：ctrl+d 复制当前行

Ctrl+x 剪贴选中文本

Ctrl+y删除选中的行

Ctrl+Alt+L，格式化代码

Maven

创建一个maven项目 导入需要的jar包 在pom.xml中

1. spring jar包
2. springMVC jar包
3. mybatis jar包

……..

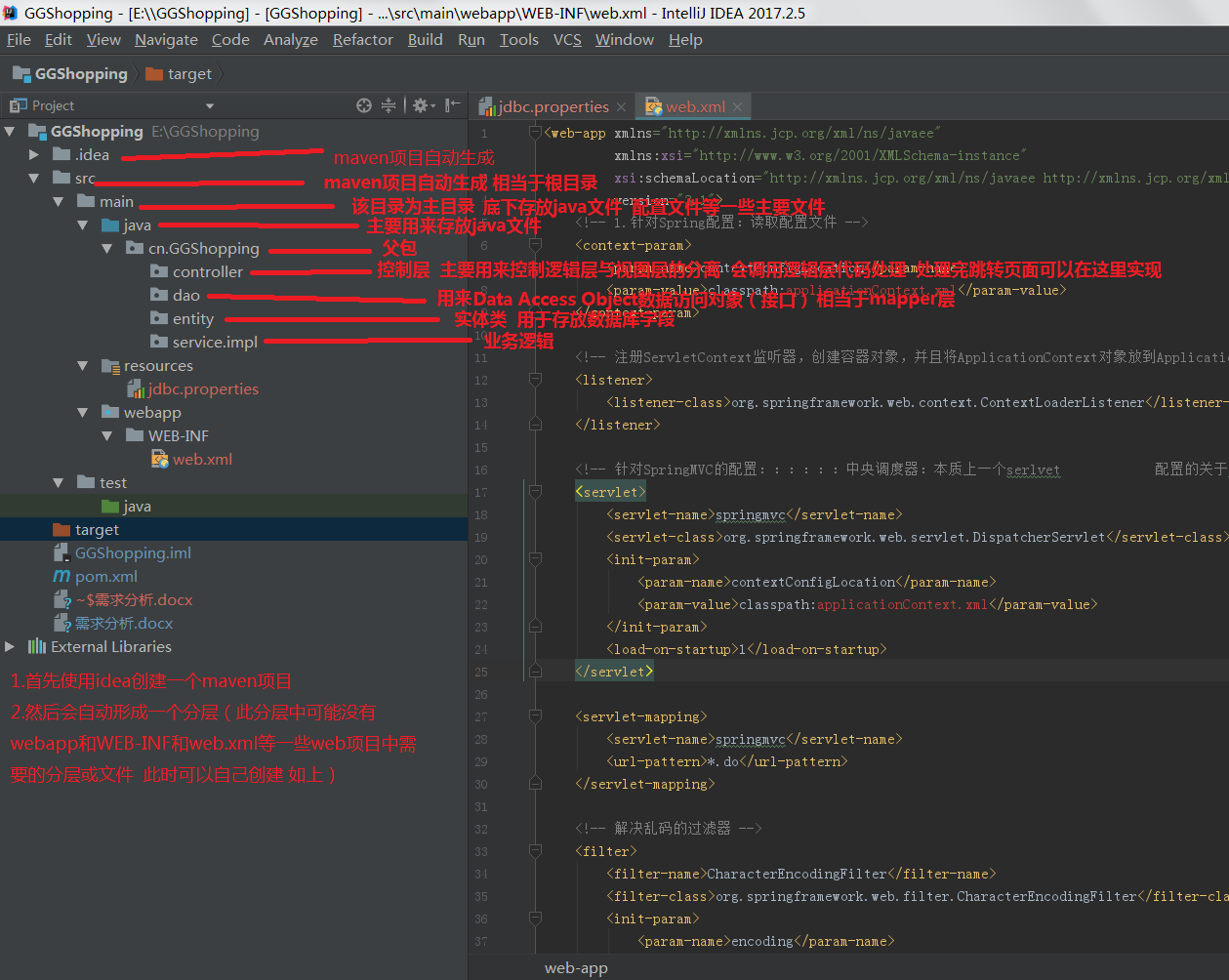
Spring4.2.5

Junit4.1.2

Ssm增删改查学习方式：

1. 首先列出123等几条 写出增删改查应该写的xml文件，java文件等，分出先后
2. 在写每个xml时弄清楚xml中每个标签 每个属性的用处并留笔记（可暂时借助网上参考）
3. 之后尽量不借助网上资料自己写出一个完整的增删改查
4. 每写一遍增删改查就要在这个增删改产中添加新的功能并记录如何实现的 慢慢形成自己的GGShopping

分层展示：



## 持久层：

### DAO层（mapper层）：

主要做数据持久的工作，与数据库进行联络 只是一个接口 可以在spring配置文件中定义此接口的实现类 可以在mybatis中使用此接口来进行增删改查的sql‘语句操作 而service层只需要调用dao层的接口就可以实现业务功能

## 业务层：

### Service层：

先写出接口然后写实现类impl 接着再在Spring的配置文件中配置其实现的关联。这样我们就可以在应用中调用Service接口来进行业务处理。 在service中需要调用dao层来实现具体的功能

## 表现层：

### Controller层（Handler层）

主要负责主要的业务模块流程的控制，需要调用service层 跳转view jsp页面

在resource包下新建Directory：“mapper”（用于存放xxxMapper.xml文件）和“spring”（用于存放spring-xxx.xml配置文件），新建文件：“jdbc.properties”（mysql数据库配置文件）,”log4j.properties”（日志输出配置文件）,”mybatis-config.xml”（mybatis框架配置文件）。

在web-inf目录下新建“jsp”包，存放xxx.jsp显示界面。

第一步：创建maven项目

第二步：分层搭骨架

第三步：pom.xml中导入jar包

Pom.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <project xmlns="http://maven.apache.org/POM/4.0.0"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">   <modelVersion>4.0.0</modelVersion>  <groupId>cn.fyg.GGShopping</groupId>  <artifactId>GGShopping</artifactId>  <packaging>war</packaging>  <version>1.0-SNAPSHOT</version>  <name>GGShopping Maven Webapp</name>  <url>http://maven.apache.org</url>  <dependencies>  <!-- https://mvnrepository.com/artifact/org.springframework/spring-beans -->  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>4.2.5.RELEASE</version>  </dependency>  <!-- https://mvnrepository.com/artifact/junit/junit -->  <dependency>  <groupId>junit</groupId>  <artifactId>junit</artifactId>  <version>4.12</version>  <scope>test</scope>  </dependency>  <dependency>  <groupId>javaee</groupId>  <artifactId>javaee-api</artifactId>  <version>5</version>  </dependency>  <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-core</artifactId>  <version>4.2.2.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-beans</artifactId>  <version>4.2.2.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context</artifactId>  <version>4.2.2.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-tx</artifactId>  <version>4.1.9.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-aspects</artifactId>  <version>4.2.1.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-context-support</artifactId>  <version>4.1.9.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-jdbc</artifactId>  <version>4.2.3.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-test</artifactId>  <version>4.1.9.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-web</artifactId>  <version>4.2.3.RELEASE</version>  </dependency>   <dependency>  <groupId>org.springframework</groupId>  <artifactId>spring-webmvc</artifactId>  <version>4.1.9.RELEASE</version>  </dependency>   <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis</artifactId>  <version>3.2.2</version>  </dependency>   <dependency>  <groupId>org.mybatis</groupId>  <artifactId>mybatis-spring</artifactId>  <version>1.3.0</version>  </dependency>   <dependency>  <groupId>org.javassist</groupId>  <artifactId>javassist</artifactId>  <version>3.17.1-GA</version>  </dependency>   <dependency>  <groupId>cglib</groupId>  <artifactId>cglib</artifactId>  <version>2.2.2</version>  </dependency>   <dependency>  <groupId>c3p0</groupId>  <artifactId>c3p0</artifactId>  <version>0.9.1.2</version>  </dependency>   <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->  <dependency>  <groupId>mysql</groupId>  <artifactId>mysql-connector-java</artifactId>  <version>5.1.38</version>  </dependency>   </dependencies>  <build>  <finalName>GGShopping</finalName>  </build>   </project> |

第四步：建表

商品表：

|  |
| --- |
| CREATE TABLE `goods` (  `goods\_id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '商品id',  `good\_name` varchar(200) NOT NULL COMMENT '商品名称',  `goods\_detail` varchar(200) NOT NULL COMMENT '商品描述',  PRIMARY KEY (`goods\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

用户表：

|  |
| --- |
| CREATE TABLE `shop\_user` (  `user\_id` bigint(20) NOT NULL AUTO\_INCREMENT COMMENT '用户id',  `user\_name` varchar(100) NOT NULL COMMENT '用户名称',  `user\_sex` varchar(5) NOT NULL COMMENT '用户性别',  `user\_age` varchar(20) NOT NULL COMMENT '用户年龄',  `password` varchar(20) NOT NULL COMMENT '密码',  PRIMARY KEY (`user\_id`)  ) ENGINE=InnoDB DEFAULT CHARSET=utf8 |

第五步：创建get、set方法

User.java

|  |
| --- |
| package cn.GGShopping.entity;  public class User {  private long userId;  private String userName;  private String userSex;  private String userAge;  private String password;   public long getUserId() {  return userId;  }   public void setUserId(long userId) {  this.userId = userId;  }   public String getUserName() {  return userName;  }   public void setUserName(String userName) {  this.userName = userName;  }   public String getUserSex() {  return userSex;  }   public void setUserSex(String userSex) {  this.userSex = userSex;  }   public String getUserAge() {  return userAge;  }   public void setUserAge(String userAge) {  this.userAge = userAge;  }   public String getPassword() {  return password;  }   public void setPassword(String password) {  this.password = password;  } } |

Goods.java

|  |
| --- |
| package cn.GGShopping.entity;  public class Goods {  private long goodsId;  private String goodsName;  private String goodsDetail;   public long getGoodsId() {  return goodsId;  }   public void setGoodsId(long goodsId) {  this.goodsId = goodsId;  }   public String getGoodsName() {  return goodsName;  }   public void setGoodsName(String goodsName) {  this.goodsName = goodsName;  }   public String getGoodsDetail() {  return goodsDetail;  }   public void setGoodsDetail(String goodsDetail) {  this.goodsDetail = goodsDetail;  } } |

此处要写service的接口和实现类 此处需要了解增删改查中需要用到的值

增：往数据库插入一条数据 获取前台输入的数据然后插入到数据库 此时需要获取到页面中输入框的数据 所以参数要设置为需要获取的数据 参数即为前台输入的数据 而不需要再往前台传输数据 所以没有返回值 往后台传数据用方法的参数 往前台返回数据用方法的返回值

|  |
| --- |
| public void addUser(User user) {} |

删：删除往往就是删除这个用户或者商品 所以此时需要获取该数据前台选中的id与后台的id是否相同或者是否存在一致的数据 如果存在返回true 则把处理if语句中的代码（把id传到dao中执行sql） 删除该数据 如果不存在返回false 不执行if语句中的sql 不删除该数据（不传id 不执行sql）

|  |
| --- |
| public boolean delUser(int id) {  return false; } |

改：修改就相当于把查询到的数据修改为其他数据 需要把前台修改的数据用方法中的参数传给sql语句 修改成功在返回true 跳转到一览页面 修改失败跳转错误页面

|  |
| --- |
| public User selUser(int id) {  return null; } |

查：查询 就是把数据库的数据一览到页面 需要传过去id 然后返回数据就可以

|  |
| --- |
| public User selUser(int id) {  return null; } |

创建UserService.java(接口)

|  |
| --- |
| package cn.GGShopping.service;  import cn.GGShopping.entity.User;  public interface UserService {  // 增（注册用户）  void addUser(User user);  // 删（注销用户）  boolean delUser(int id);  // 改（修改用户信息）  boolean updUser(User user);  // 查（查询出所有的用户 此功能可不实现）  User selUser (int id); } |

创建GoodsService.java(接口)

|  |
| --- |
| package cn.GGShopping.service;  import cn.GGShopping.entity.Goods;  public interface GoodsService {  //增（添加商品）  void addGoods(Goods goods);  //删（删除商品）  boolean delGoods(int id);  //改（修改商品信息）  boolean updGoods(Goods goods);  //查（一览商品）  Goods seleGoods(int id); } |

创建UserServiceImpl.java（实现类）

|  |
| --- |
| package cn.GGShopping.service.impl;  import cn.GGShopping.entity.User; import cn.GGShopping.service.UserService;  public class UserServiceImpl implements UserService{  @Override  public void addUser(User user) {   }   @Override  public boolean delUser(int id) {  return false;  }   @Override  public boolean updUser(User user) {  return false;  }   @Override  public User selUser(int id) {  return null;  } } |

创建GoodsServiceImpl.java（实现类）

|  |
| --- |
| package cn.GGShopping.service.impl;  import cn.GGShopping.entity.Goods; import cn.GGShopping.service.GoodsService;  public class GoodsServiceImpl implements GoodsService {   @Override  public void addGoods(Goods goods) {   }   @Override  public boolean delGoods(int id) {  return false;  }   @Override  public boolean updGoods(Goods goods) {  return false;  }   @Override  public Goods seleGoods(int id) {  return null;  } } |

创建UserDao.java(接口)

|  |
| --- |
| package cn.GGShopping.dao;  import cn.GGShopping.entity.User;  public interface UserDao {  // 增（注册用户）  void addUser(User user);  // 删（注销用户）  boolean delUser(int id);  // 改（修改用户信息）  boolean updUser(User user);  // 查（查询出所有的用户 此功能可不实现）  User selUser (int id); } |

创建GoodsDao.java（接口）

|  |
| --- |
| package cn.GGShopping.dao;  import cn.GGShopping.entity.Goods;  public interface GoodsDao {  //增（添加商品）  void addGoods(Goods goods);  //删（删除商品）  boolean delGoods(int id);  //改（修改商品信息）  boolean updGoods(Goods goods);  //查（一览商品）  Goods seleGoods(int id); } |

在创建Controller 之前先把之前没有创建的xml配置文件创建一下

首先：

* 在resource/mapper包下新建 UserMapper.xml/GoodsMapper.xml（mybatis框架mapper代理开发配置文件）

**ResultMap标签基本作用：建立SQL查询结果字段与实体属性的映射关系信息**

标签说明：

主标签  
 id:该resultMap的标志  
 type：返回值的类名，此例中返回Goods类  
 子标签：  
 id:用于设置主键字段与领域模型属性的映射关系，此处主键为ID，对应id。

result：用于设置普通字段与领域模型属性的映射关系

GoodsMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="cn.GGShopping.dao.GoodsDao">  <!--ResultMap标签基本作用：建立SQL查询结果字段与实体属性的映射关系信息-->  <resultMap id="GoodsResultMap" type="Goods">  <id property="goodsId" column="goods\_id"/>  <result property="goodsName" column="goods\_name"/>  <result property="goodsDetail" column="goods\_detail"/>  </resultMap>  ……. </mapper> |

UserMapper.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8" ?> <!DOCTYPE mapper  PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"  "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  <mapper namespace="cn.GGShopping.dao.UserDao">  <!--ResultMap标签基本作用：建立SQL查询结果字段与实体属性的映射关系信息-->  <resultMap id="UserResultMap" type="User">  <id property="userId" column="user\_id"/>  <result property="userName" column="user\_name"/>  <result property="userSex" column="user\_sex"/>  <result property="userAge" column="user\_Age"/>  <result property="password" column="password"/>  </resultMap>  …… </mapper> |

* 在resource/spring包下分别新建配置文件：spring-dao.xml , spring-mvc.xml , spring-service.xml

spring-dao.xml：（spring-mybatis整合配置文件）

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd">   <!--配置整合spring-mybatis过程-->  <!--1.配置数据库相关参数properties的属性-->  <context:property-placeholder location="classpath:jdbc.properties"/>  <!--2.数据库连接池-->  <bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  <!-- 配置连接池属性 -->  <property name="driverClass" value="${jdbc.driver}"/>  <property name="jdbcUrl" value="${jdbc.jdbcUrl}"/>  <property name="user" value="${jdbc.user}"/>  <property name="password" value="${jdbc.password}"/>   <!--c3p0的私有连接池-->  <property name="maxPoolSize" value="30"/>  <property name="minPoolSize" value="10"/>  <!--关闭连接后不自动commit-->  <property name="autoCommitOnClose" value="false"/>  <!--获取连接超时时间-->  <property name="checkoutTimeout" value="10000"/>  <!--当获取连接失败重试次数-->  <property name="acquireRetryAttempts" value="2"/>  </bean>  <!--3.配置SqlSessionFactory对象-->  <bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  <!--注入数据库连接池-->  <property name="dataSource" ref="dataSource"/>  <!--配置mybatis全局配置文件：mybatis-config.xml-->  <property name="configLocation" value="classpath:mybatis-config.xml"/>  <!--扫描pojo包 使用别名-->  <property name="typeAliasesPackage" value="cn.GGShopping.entity"/>  <!--扫描sql配置文件：mapper需要的xml文件-->  <property name="mapperLocations" value="classpath:mapper/\*.xml"/>  </bean>  <!--4.配置扫描dao接口包 动态实现Dao接口 注入到spring容器中-->  <bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">  <!--注入SqlSessionFactory-->  <property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"/>  <!--给出需要扫描的dao接口包-->  <property name="basePackage" value="cn.GGShopping.dao"/>  </bean> </beans> |

spring-mvc.xml：（spring mvc配置文件）

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:mvc="http://www.springframework.org/schema/mvc"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd">  <!--配置SpringMVC-->  <!--1.开启SpringMVC注解模式-->  <!--简化配置  (1)自动注册处理器映射器（DefaultAnootationHandlerMapping）、处理器适配器（AnotationMethodHandlerAdapter）  (2)提供一些列:数据绑定、数字和日期的格式 @NumberFormat, @DateTimeFormat, xml,json默认读写支持  -->  <mvc:annotation-driven/>   <!--2.静态资源默认servlet配置  (1)加入对静态资源的处理：js，gif，png  (2)允许使用"/"做整体映射  -->  <mvc:default-servlet-handler/>  <!--3.配置jsp 显示viewResolver-->  <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">  <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>  <property name="prefix" value="/WEB-INF/jsp/"/>  <property name="suffix" value=".jsp"/>  </bean>   <!--4.扫描web相关的bean-->  <bean>  <context:component-scan base-package="cn.GGShopping.controller"/>  </bean> </beans> |

spring-service.xml

|  |
| --- |
| <?xml version="1.0" encoding="UTF-8"?> <beans xmlns="http://www.springframework.org/schema/beans"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xmlns:context="http://www.springframework.org/schema/context"  xmlns:tx="http://www.springframework.org/schema/tx"  xsi:schemaLocation="http://www.springframework.org/schema/beans  http://www.springframework.org/schema/beans/spring-beans.xsd  http://www.springframework.org/schema/context  http://www.springframework.org/schema/context/spring-context.xsd  http://www.springframework.org/schema/tx  http://www.springframework.org/schema/tx/spring-tx.xsd">   <!--1.扫描service包下所有使用注解的类型-->  <context:component-scan base-package="cn.GGShopping.service"/>  <!--配置事务管理器-->  <bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  <!--注入数据库连接池-->  <property name="dataSource" ref="dataSource"/>  </bean>   <!--配置基于注解的声明式事务-->  <tx:annotation-driven transaction-manager="transactionManager"/> </beans> |

Web.xml

|  |
| --- |
| <web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"  version="3.1">  <!-- 1.针对Spring配置：读取配置文件 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:applicationContext.xml</param-value>  </context-param>   <!-- 注册ServletContext监听器，创建容器对象，并且将ApplicationContext对象放到Application域中 -->  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>   <!-- 针对SpringMVC的配置：：：：：：中央调度器：本质上一个serlvet 配置的关于SpringmVC组件 -->  <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:applicationContext.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>   <servlet-mapping>  <servlet-name>springmvc</servlet-name>  <url-pattern>\*.do</url-pattern>  </servlet-mapping>   <!-- 解决乱码的过滤器 -->  <filter>  <filter-name>CharacterEncodingFilter</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>   <init-param>  <param-name>forceEncoding</param-name>  <param-value>true</param-value>  </init-param>  </filter>   <filter-mapping>  <filter-name>CharacterEncodingFilter</filter-name>  <url-pattern>/\*</url-pattern>  </filter-mapping> </web-app> |

ApplicationContext.xml

|  |
| --- |
|  |