**SSM系统架构设计**

# **SSM框架整体介绍**

## SSM框架简介

Spring + Spring MVC +Mybatis

## Spring框架知识

|  |
| --- |
| @Autowired按byType自动注入  @Resource默认按 byName自动注入 |

## Spring注解知识

|  |
| --- |
| 传统的Spring做法是使用.xml文件来对bean进行注入或者是配置aop、事物，这么做有两个缺点：  1、如果所有的内容都配置在.xml文件中，那么.xml文件将会十分庞大；如果按需求分开.xml文件，那么.xml文件又会非常多。总之这将导致配置文件的可读性与可维护性变得很低  2、在开发中在.java文件和.xml文件之间不断切换，是一件麻烦的事，同时这种思维上的不连贯也会降低开发的效率  为了解决这两个问题，Spring引入了注解，通过"@XXX"的方式，让注解与Java Bean紧密结合，既大大减少了配置文件的体积，又增加了Java Bean的可读性与内聚性。  使用注解：  使用必须告诉spring一下我要使用注解了，告诉的方式有很多，<context:component-scan base-package="xxx" />是一种最简单的，spring会自动扫描xxx路径下的注解。 |
| @Service用于标注业务层组件，表示定义一个bean，自动根据bean的类名实例化一个首写字母为小写的bean  @Controller用于标注控制层组件（如struts中的action）  @Repository用于标注数据访问组件，即DAO组件  @Component泛指组件，当组件不好归类的时候，我们可以使用这个注解进行标注。 |
| @Scope默认是"singleton"即单例  "prototype"表示原型即每次都会new一个新的出来 |
| @Autowired和@Resource两个注解的区别：   * @Autowired默认按照byType方式进行bean匹配，   @Resource默认按照byName方式进行bean匹配   * @Autowired是Spring的注解，   @Resource是J2EE的注解，  这个看一下导入注解的时候这两个注解的包名就一清二楚了  **@Autowired 自动装配**   * 可以省略get和set方法； * 当Spring发现@Autowired注解时，将自动在代码上下文中找到和其匹配（默认是类型匹配）的Bean，并自动注入到相应的地方去   **@Autowired接口注入**   * 如果有一个接口，对应一个实现类，则可以正常装配； * 如果有一个接口，有多个实现: * **@Autowired** * **@Qualifier("xxx")** * 注意@Qualifier注解**括号里面的应当是接口实现类的类名**   **@Resource的装配顺序：**   * @Resource后面没有任何内容，默认通过name属性去匹配bean，找不到再按type去匹配 * 指定了name或者type则根据指定的类型去匹配bean   指定了name和type则根据指定的name和type去匹配bean，任何一个不匹配都将报错 |
| 把DAO实现类注入到service实现类中，  把service的接口(注意不要是service的实现类)注入到action中 |
| 把DAO实现类注入到service实现类中，把service的接口(注意不要是service的实现类)注入到action中，注   入时不要new 这个注入的类，因为spring会自动注入，如果手动再new的话会出现错误，然后属性加上   @Autowired后不需要getter()和setter()方法，Spring也会自动注入。至于更具体的内容，等对注入的方式更   加熟练后会做个完整的例子上来。 |

## SpringMVC框架体系

|  |
| --- |
| Spring ＭＶＣ的控制器是单例模式，线程不安全 |
| 1. 用户发送请求至前端控制器DispatcherServlet。 2. DispatcherServlet收到请求调用HandlerMapping处理器映射器。 3. 处理器映射器根据请求url找到具体的处理器，生成处理器对象及处理器拦截器(如果有则生成)一并返回给DispatcherServlet。 4. DispatcherServlet通过HandlerAdapter处理器适配器调用处理器。 5. 执行处理器(Controller，也叫后端控制器)。 6. Controller执行完成返回ModelAndView。 7. HandlerAdapter将controller执行结果ModelAndView返回给DispatcherServlet。 8. DispatcherServlet将ModelAndView传给ViewReslover视图解析器。 9. ViewReslover解析后返回具体View。 10. DispatcherServlet对View进行渲染视图（即将模型数据填充至视图中）。   DispatcherServlet响应用户 |
|  |

## SpringMVC注解&驱动器

|  |
| --- |
| <context:component-scan/> 扫描指定的包中的类上的注解，常用的注解有：  @Controller 声明Action组件 控制器的注解，  @Service    声明Service组件    @Service("myMovieLister")  @Repository 声明Dao组件  @Component   泛指组件, 当不好归类时.  @RequestMapping("/menu")  请求映射  @Resource  用于注入，( j2ee提供的 ) 默认按名称装配，@Resource(name="beanName")  @Autowired 用于注入，(srping提供的) 默认按类型装配  @Transactional( rollbackFor={Exception.class}) 事务管理  @ResponseBody  @Scope("prototype")   设定bean的作用域、  @RequestMapping  作用类上是父路径  作用于方法上处理器  建议不要舍弃类定义的RequestMapping  支持占位符风格定义  URL中的{xxx}占位符可以通过@PathVariable(“xxx”)绑定到操作的入参上  @RequestMapping有六个参数。分别是：  1.value  用于设置方法或者类的映射路径，可以直接写路径，即@RequestMapping("/Student");==@RequestMapping(value="/Student");  2.method  用于指定请求的方法，可以设置单个或多个，如果请求方法不满足条件则会请求失败。  3.params   这个参数比较特殊，他和后续讲到的@RequestParam的作用有点像，不过params是可以指定多个url携带的参数，而@RequestParam只可以对一个参数进行注解。  4.headers  headers这个参数，让人有种望文生义的感觉。看着这个单词就大概可以猜到，他是和请求头部有关，而事实上也确实是这样。用于指定请求的headers，必须要含有这个headers才可以请求。  前四个感觉还是很重要的，特别是前两个，但是最后两个就不清楚以后是否要用到。  5.consumes  用于指定处理何种请求的提交内容类型context-type，如果不是指定的类型，则不处理。提交的类型会在页面就有提示，比如说，一般的网页是text/html  6. produces  按道理来说这个也应该是要满足什么要求才进行处理，所以说这个应该是请求中accept含有produces中指定的内容类型才可以进行处理。例如  produces="application/json"  意思就是，请求中accept含有application/json的内容类型才可以进行处理。  这个完全不知道什么回事。暂且先这样留着。 |

## SpringMVC和Spring父子容器

|  |
| --- |
| * Spring框架，容器是核心思想，用管理Bean的整个生命周期的； * Spring中可以包括多个容器，容器有上下层关系 * 最常是项目中引入Spring和SpringMVC这两个框架，它其实就是两个容器， * Spring是父容器，SpringMVC是其子容器， * 在Spring父容器中注册的Bean对于SpringMVC容器中是可见的， * 而在SpringMVC容器中注册的Bean对于Spring父容器中是不可见的， * 就是子容器可以看见父容器中的注册的Bean，反之就不行   Spring和Springmvc和作为两个独立的容器，  会把扫描到的注解对象分别放到两个不同的容器中  Springmvc容器是spring容器的一部分，他们访问对象的范围如下所示：  ①：springmvc子容器可以访问spring父容器中的对象  ②：spring父容器不能访问springmvc子容器中的对象 |

# **SSM框架-DAO层**

## 数据库设计规范

|  |
| --- |
| 一:基础规范 **1、使用InnoDB存储引擎**  支持事务、行级锁、并发性能更好、CPU及内存缓存页优化使得资源利用率更高  **2、推荐使用utf8mb4字符集**  无需转码，无乱码风险, 支持emoji表情以及部分不常见汉字  **3、表、字段必须加注释**  方便他人理解字段意思。  **4、不在数据库做计算**  禁止使用存储过程、视图、触发器、Event。  在并发量大的情况下，这些功能很可能将数据库拖跨，业务逻辑放到服务层具备更好的扩展性，能够轻易实现“增机器就加性能”  **5、禁止存储文件**  文件存储在文件系统，数据库里存URI  **6、控制单表数据量**  单表记录控制在千万级 二:命名规范 **1、库名、表名、字段名：小写，下划线风格**  非唯一索引名idxxxx，唯一索引名uniqxxx  **2、表必须有主键，例如自增主键**  a）主键递增，数据行写入可以提高插入性能  b）主键要选择较短的数据类型，Innodb引擎普通索引都会保存主键的值，较短的数据类型可以有效的减少索引的磁盘空间，提高索引的缓存效率  c）保证实体的完整性，唯一性  **3、不要使用外键，如果有外键约束，用应用程序控制**  外键会导致表与表之间耦合，update与delete操作都会涉及相关联的表，十分影响sql 的性能，甚至会造成死锁。高并发情况下容易造成数据库性能下降，大数据高并发业务场景数据库使用以性能优先 三:字段设计规范 **1、把字段定义为NOT NULL并且提供默认值**  a）null的列使索引/索引统计/值比较都更加复杂，对MySQL来说更难优化  b）null 这种类型MySQL内部需要进行特殊处理，增加数据库处理记录的复杂性；同等条件下，表中有较多空字段的时候，数据库的处理性能会降低很多  c）null值需要更多的存储空间，无论是表还是索引中每行中的null的列都需要额外的空间来标识  d）对null 的处理时候，只能采用is null或is not null，而不能采用=、in、<、<>、!=、not in这些操作符号。如：where name!=’zhangsan’，如果存在name为null值的记录，查询结果就不会包含name为null值的记录  **2、不要使用TEXT、BLOB类型**  会浪费更多的磁盘和内存空间，非必要的大量的大字段查询会淘汰掉热数据，导致内存命中率急剧降低，影响数据库性能,如果必须要使用则独立出来一张表，用主键来对应，避免影响其它字段索引效率  **3、不要使用小数存储货币**  建议使用整数，小数容易导致钱对不上  **4、必须使用varchar存储手机号**  手机号会去做数学运算么？  **5、为提高效率可以牺牲范式设计，冗余数据**  a)不是频繁修改的字段  b)不是 varchar 超长字段，更不能是 text 字段 四:索引设计规范 **1、禁止在更新十分频繁、区分度不高的属性上建立索引**  a）更新会变更B+树，更新频繁的字段建立索引会大大降低数据库性能  b）“性别”这种区分度不大的属性，建立索引是没有什么意义的  **2、建立组合索引，必须把区分度高的字段放在最左边**  如果 where a=? and b=? ， a 列的几乎接近于唯一值，那么只需要单建 idx\_a 索引即可  **3、 页面搜索严禁左模糊或者全模糊**  索引文件具有 B-Tree 的最左前缀匹配特性，如果左边的值未确定，那么无法使用此索引, 如果需要请走搜索引擎来解决 五:SQL使用规范 **1、禁止使用SELECT \*，只获取必要的字段，需要显示说明列属性**  a）消耗cpu，io，内存，带宽  b）不能有效的利用覆盖索引  c）使用SELECT \*容易在增加或者删除字段后出现程序BUG, 不具有扩展性  **2、使用INSERT INTO t\_xxx VALUES(xxx)，必须显示指定插入的列属性**  容易在增加或者删除字段后出现程序BUG  **3、务必请使用“同类型”进行比较，否则可能全表扫面**  SELECT name FROM t\_user WHERE phone=1333333333 会导致全表扫描.  **4、禁止在WHERE条件的上使用函数或者计算**  解读：SELECT naem FROM tuser WHERE date(createdatatime)='2017-12-15' 会导致全表扫描  推荐的写法是：SELECT name FROM tuser WHERE createdatatime>= '2017-02-15 ' and create\_datatime < '2017-02-16 '  **5、禁止负向查询，以及%开头的模糊查询**  a）负向查询条件：NOT、!=、<>、!<、!>、NOT IN、NOT LIKE等，会导致全表扫描  b）%开头的模糊查询，会导致全表扫描  **6、不要大表使用JOIN查询，禁止大表使用子查询**  会产生临时表，消耗较多内存与CPU，极大影响数据库性能  **7、OR改写为IN()或者UNION**  原因很简单or不会走索引  **8、简单的事务**  事务就像程序中的锁一样粒度尽可能要小  **9、不要一次更新大量数据**  数据更新会对行或者表加锁，应该分为多次更新 |

## 创建数据库

数据表名称规范

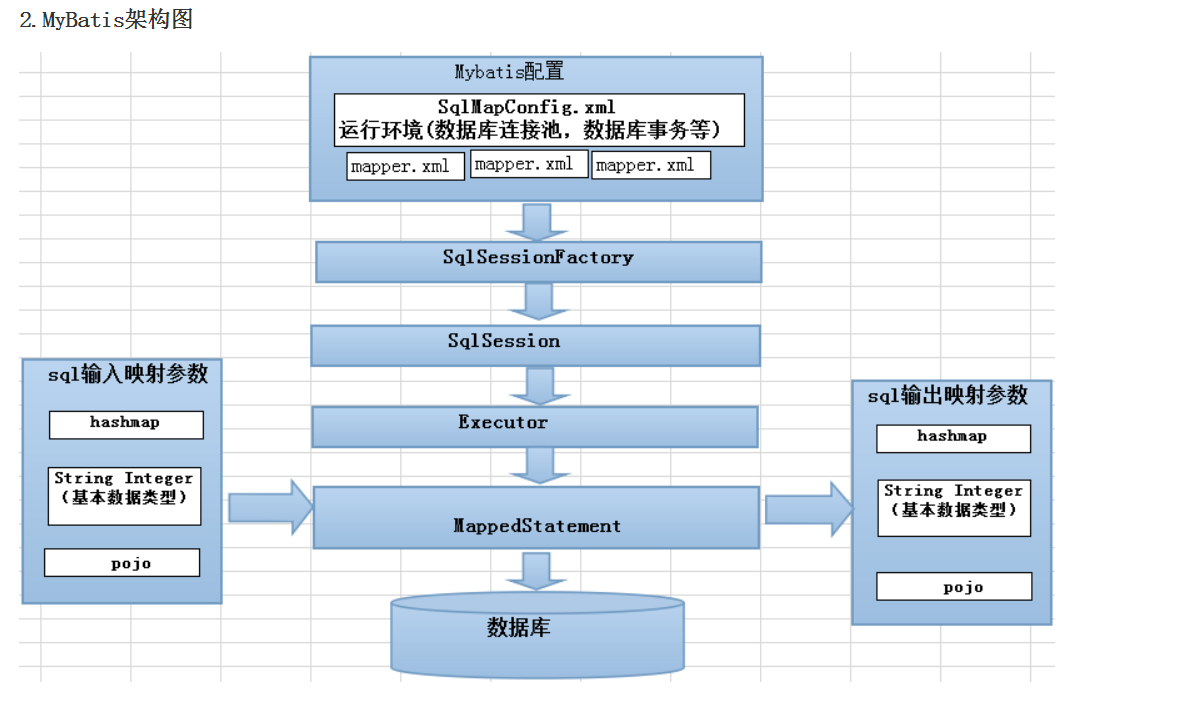
尽可能减少表关联问题

|  |
| --- |
| 优点：   * 有利于分库分表 * 有利于查询速度，性能好   缺点：   * 数据冗余 * 不符合数据库范式 * 硬盘空间增加 |

## 逆向工程生产Mybatis文件

|  |
| --- |
| 逆向工程：配置数据库的连接信息生产Mybatis需要的pojo和mapper包  生产工具：生成工具mybatis-generator-core-1.3.2 执行java文件  需要配下 ：generatorConfig.xml  注意：逆向工程生产是追加；记住需要先删后生； |

## Mybatis框架简介



|  |
| --- |
| **1、mybatis配置**  **SqlMapConfig.xml，此文件作为mybatis的全局配置文件，配置了mybatis的运行环境等信息。**  **mapper.xml文件即sql映射文件，文件中配置了操作数据库的sql语句。此文件需要在SqlMapConfig.xml中加载。**  **2、通过mybatis环境等配置信息构造SqlSessionFactory即会话工厂**  **3、由会话工厂创建sqlSession即会话，操作数据库需要通过sqlSession进行。**  **4、mybatis底层自定义了Executor执行器接口操作数据库，Executor接口有两个实现，一个是基本执行器、一个是缓存执行器。**  **5、Mapped Statement也是mybatis一个底层封装对象，它包装了mybatis配置信息及sql映射信息等。mapper.xml文件中一个sql对应一个Mapped Statement对象，sql的id即是Mapped statement的id。**  **6、Mapped Statement对sql执行输入参数进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql前将输入的java对象映射至sql中，输入参数映射就是jdbc编程中对preparedStatement设置参数。**  **7、Mapped Statement对sql执行输出结果进行定义，包括HashMap、基本类型、pojo，Executor通过Mapped Statement在执行sql后将输出结果映射至java对象中，输出结果映射过程相当于jdbc编程中对结果的解析处理过程。** |

|  |
| --- |
| **1.parameterType和resultType**  parameterType:在映射文件中通过parameterType指定输入 参数的类型。　　resultType:在映射文件中通过resultType指定输出结果的类型  **2.#{}和${}**  #{}表示一个占位符号，#{}接收输入参数，类型可以是简单类型，pojo、hashmap;  如果接收简单类型，#{}中可以写成value或其它名称;  #{}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。  表示一个拼接符号，会引用sql注入，所以不建议使用表示一个拼接符号，会引用sql注入，所以不建议使用{}；  ${}接收输入参数，类型可以是简单类型，pojo、hashmap;  如果接收简单类型，${}中只能写成value;  ${}接收pojo对象值，通过OGNL读取对象中的属性值，通过属性.属性.属性...的方式获取对象属性值。  **3.selectOne()和selectList()**  selectOne表示查询出一条记录进行映射。如果使用selectOne可以实现使用selectList也可以实现（list中只有一个对象）。  selectList表示查询出一个列表（多条记录）进行映射。如果使用selectList查询多条记录，不能使用selectOne。 |

## Spring和Mybatis整合

|  |
| --- |
| 通过spring管理SqlSessionFactory、mapper代理对象。  需要mybatis和spring的整合包：mybatis-spring.jar  *<!-- 添加mybatis与Spring整合的核心包 -->* <**dependency**>  <**groupId**>org.mybatis</**groupId**>  <**artifactId**>mybatis-spring</**artifactId**>  <**version**>1.2.2</**version**> </**dependency**> |

## Spring配置数据库连接池

创建applicationContext-dao.xml配置数据库连接

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"**>   *<!-- 数据库连接池 -->  <!-- 加载配置文件 -->* <**context:property-placeholder location="classpath:resource/\*.properties"** />   *<!-- 数据库连接池 -->* <**bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource" init-method="init" destroy-method="close"**>  <**property name="driverClassName" value="${jdbc.driver}"** />  <**property name="url" value="${jdbc.url}"** />  <**property name="username" value="${jdbc.username}"** />  <**property name="password" value="${jdbc.password}"** />  *<!-- 配置初始化大小、最小、最大 -->* <**property name="initialSize" value="1"** />  <**property name="minIdle" value="1"** />  <**property name="maxActive" value="10"** />  *<!-- 配置获取连接等待超时的时间 -->* <**property name="maxWait" value="10000"** />  *<!-- 配置间隔多久才进行一次检测，检测需要关闭的空闲连接，单位是毫秒 -->* <**property name="timeBetweenEvictionRunsMillis" value="60000"** />  *<!-- 配置一个连接在池中最小生存的时间，单位是毫秒 -->* <**property name="minEvictableIdleTimeMillis" value="300000"** />  <**property name="testWhileIdle" value="true"** />  *<!-- 这里建议配置为TRUE，防止取到的连接不可用 -->* <**property name="testOnBorrow" value="true"** />  <**property name="testOnReturn" value="false"** />  *<!-- 打开PSCache，并且指定每个连接上PSCache的大小 -->* <**property name="poolPreparedStatements" value="true"** />  <**property name="maxPoolPreparedStatementPerConnectionSize" value="20"** />  *<!-- 这里配置提交方式，默认就是TRUE，可以不用配置 -->* <**property name="defaultAutoCommit" value="true"** />  *<!-- 验证连接有效与否的SQL，不同的数据配置不同 -->* <**property name="validationQuery" value="select 1 "** />  </**bean**>   *<!-- 配置sqlsessionFactory -->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  <**property name="configLocation" value="classpath:mybatis/SqlMapConfig.xml"**></**property**>  <**property name="dataSource" ref="dataSource"**></**property**>  </**bean**>  *<!-- 配置扫描包，加载mapper代理对象 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  <**property name="basePackage" value="com.abyiyi.mapper"**></**property**>  </**bean**> </**beans**> |

## Spring配置数据库事务

创建applicationContext-trans.xml 文件，配置spring的事务管理

|  |
| --- |
| *<?***xml version="1.0" encoding="UTF-8"***?>* <**beans xmlns="http://www.springframework.org/schema/beans"  xmlns:context="http://www.springframework.org/schema/context" xmlns:p="http://www.springframework.org/schema/p"  xmlns:aop="http://www.springframework.org/schema/aop" xmlns:tx="http://www.springframework.org/schema/tx"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd  http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd  http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd  http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"**>   *<!-- 事务管理器 -->* <**bean id="transactionManager"  class="org.springframework.jdbc.datasource.DataSourceTransactionManager"**>  *<!-- 数据源 -->* <**property name="dataSource" ref="dataSource"** />  </**bean**>  *<!-- 通知 -->* <**tx:advice id="txAdvice" transaction-manager="transactionManager"**>  <**tx:attributes**>  *<!-- 传播行为 -->* <**tx:method name="save\*" propagation="REQUIRED"** />  <**tx:method name="insert\*" propagation="REQUIRED"** />  <**tx:method name="add\*" propagation="REQUIRED"** />  <**tx:method name="create\*" propagation="REQUIRED"** />  <**tx:method name="delete\*" propagation="REQUIRED"** />  <**tx:method name="update\*" propagation="REQUIRED"** />  <**tx:method name="find\*" propagation="SUPPORTS" read-only="true"** />  <**tx:method name="select\*" propagation="SUPPORTS" read-only="true"** />  <**tx:method name="get\*" propagation="SUPPORTS" read-only="true"** />  </**tx:attributes**>  </**tx:advice**>  *<!-- 切面 -->* <**aop:config**>  <**aop:advisor advice-ref="txAdvice"  pointcut="execution(\* com.abyiyi.service.\*.\*(..))"** />  </**aop:config**> </**beans**> |

## Mybatis常见问题一[Mybatis 加载 Mapper配置的四种方式](http://blog.csdn.net/bestcxx/article/details/72966768)

问题1：

|  |
| --- |
| org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.apache.ibatis.binding.BindingException: Invalid bound statement (not found): |

解决方法:Mapper文件的四种加载方式

|  |
| --- |
| 第一种依据 Mapper 类 具体路径 配置 Mapper 如果是非注解模式的话xml配置文件必须和这个类在同一级目录，且与Mapper类同名    存在 xml 配置文件的情况下，文件结构如下，没有 xml 配置文件的情况下，只有 Mapper 类 http://img.blog.csdn.net/20170609174252874?watermark/2/text/aHR0cDovL2Jsb2cuY3Nkbi5uZXQvYmVzdGN4eA==/font/5a6L5L2T/fontsize/400/fill/I0JBQkFCMA==/dissolve/70/gravity/Center 第二种依据 Mapper 类所在的package包路径 配置 Mapper 这种情况下，如果是非注解模式的话xml配置文件必须也处于同一级 package 下，且与Mapper类同名    文件结构  http://img.blog.csdn.net/20170610170428946 第三种是把 Mapper 的xml配置文件单独放置到 resources 中，和Mapper 类分开了 这种方式的好处是便于统一管理 xml 配置文件，不好的的地方是无法使用注解模式了    然后Mapper类还是呆在该在的位置  http://img.blog.csdn.net/20170610185522158  但是mapper 的 xml文件的位置如下  http://img.blog.csdn.net/20170610185914879 上面的配置在mybitas-config.xml（名字是可以自定义的）中集中管理，Mapper 的 xml 文件需要关联第一，是mybatis-config.xml 文件负责加载 Mapper的xml文件，即上面所说的配置是在这个mybatis-config.xml文件中第二，mapper对应的xml 配置文件内部要指定对应的Mapper类第四种，直接在spring中配置Mapper的路径，但是mybatis.xml还是必不可少的 在sqlSessionFactory中设定 mybitas的xml和aliases |
| 设置Mapper类 |

# **SSM框架-Service层**

## 管理Service配置

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <context:component-scan base-package=*"com.taotao.service"*/>  </beans> |

## 事务配置

创建applicationContext-trans.xml

|  |
| --- |
| <beans xmlns=*"http://www.springframework.org/schema/beans"*  xmlns:context=*"http://www.springframework.org/schema/context"* xmlns:p=*"http://www.springframework.org/schema/p"*  xmlns:aop=*"http://www.springframework.org/schema/aop"* xmlns:tx=*"http://www.springframework.org/schema/tx"*  xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.0.xsd*  *http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-4.0.xsd http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-4.0.xsd*  *http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-4.0.xsd"*>  <!-- 事务管理器 -->  <bean id=*"transactionManager"*  class=*"org.springframework.jdbc.datasource.DataSourceTransactionManager"*>  <!-- 数据源 -->  <property name=*"dataSource"* ref=*"dataSource"* />  </bean>  <!-- 通知 -->  <tx:advice id=*"txAdvice"* transaction-manager=*"transactionManager"*>  <tx:attributes>  <!-- 传播行为 -->  <tx:method name=*"save\*"* propagation=*"REQUIRED"* />  <tx:method name=*"insert\*"* propagation=*"REQUIRED"* />  <tx:method name=*"add\*"* propagation=*"REQUIRED"* />  <tx:method name=*"create\*"* propagation=*"REQUIRED"* />  <tx:method name=*"delete\*"* propagation=*"REQUIRED"* />  <tx:method name=*"update\*"* propagation=*"REQUIRED"* />  <tx:method name=*"find\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"select\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  <tx:method name=*"get\*"* propagation=*"SUPPORTS"* read-only=*"true"* />  </tx:attributes>  </tx:advice>  <!-- 切面 -->  <aop:config>  <aop:advisor advice-ref=*"txAdvice"*  pointcut=*"execution(\* com.taotao.service.\*.\*(..))"* />  </aop:config>  </beans> |

# **SSM框架-表现层**

## web.xml加载spring和springmvc基础配置

|  |
| --- |
| <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://java.sun.com/xml/ns/javaee"* xmlns:web=*"http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  xsi:schemaLocation=*"http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_2\_5.xsd"*  id=*"WebApp\_ID"* version=*"2.5"*>  <display-name>xxx</display-name>  <welcome-file-list>  <welcome-file>login.html</welcome-file>  </welcome-file-list>  <!-- 加载spring容器 -->  <context-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/applicationContext\*.xml</param-value>  </context-param>  <listener>  <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  </listener>  <!-- 解决post乱码 -->  <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>  <!-- springmvc的前端控制器 -->  <servlet>  <servlet-name>taotao-manager</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <!-- contextConfigLocation不是必须的， 如果不配置contextConfigLocation， springmvc的配置文件默认在：WEB-INF/servlet的name+"-servlet.xml" -->  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>classpath:spring/springmvc.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <servlet-mapping>  <servlet-name>taotao-manager</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>  </web-app> |

## springmvc的配置详解

|  |  |
| --- | --- |
| <?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:p=*"http://www.springframework.org/schema/p"*  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*  *http://www.springframework.org/schema/mvc http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd*  *http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd"*>  <context:component-scan base-package=*"com.taotao.controller"* />  <mvc:annotation-driven />  <bean  class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>  <property name=*"prefix"* value=*"/WEB-INF/jsp/"* />  <property name=*"suffix"* value=*".jsp"* />  </bean>  </beans> |  |

# **SSM框架-前端知识**

## Ajax和Jsonp区别

|  |
| --- |
| json是一种轻量级的数据交换格式。  jsonp是一种跨域数据交互协议。  json的优点：  （1）基于纯文本传递极其简单，  （2）轻量级数据格式适合互联网传递，  （3）容易编写和解析。  ajax和jsonp的区别：  相同点：都是请求一个url  不同点：ajax的核心是通过xmlHttpRequest获取内容  　　　　jsonp的核心则是动态添加<script>标签来调用服务器 提供的js脚本。  它不像XMLHttpRequest对象实现的Ajax请求那样受到同源策略的限制；它的兼容性更好，在更加古老的浏览器中都可以运行，不需要XMLHttpRequest或ActiveX的支持；并且在请求完毕后可以通过调用callback的方式回传结果  Jsonp的缺点：  支持GET请求而不支持POST等其它类型的HTTP请求；  而jsonp的核心则是动态添加标签来调用服务器提供的js脚本。  **jsonp原理：**   1.首先在客户端注册一个callback, 然后把callback的名字传给服务器。  2.服务器先生成 json 数据。 然后以 javascript 语法的方式，生成一个function , function 名字就是传递上来的参数 jsonp. 最后将 json 数据直接以入参的方式，放置到 function 中，这样就生成了一段 js 语法的文档，返回给客户端。  3.客户端浏览器，解析script标签，并执行返回的 javascript 文档，此时数据作为参数，传入到了客户端预先定义好的 callback 函数里.（动态执行回调函数） |