# 三大框架整合

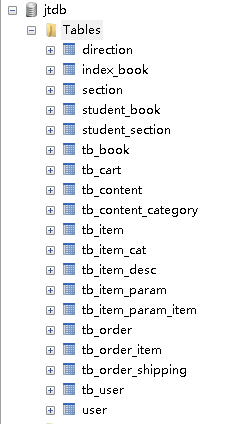
## 任务需求

### 需求说明

说明:通过框架额调用实现用户的列表信息展现. 要求使用.jsp展现并且使用jstl表达式动态获取参数.

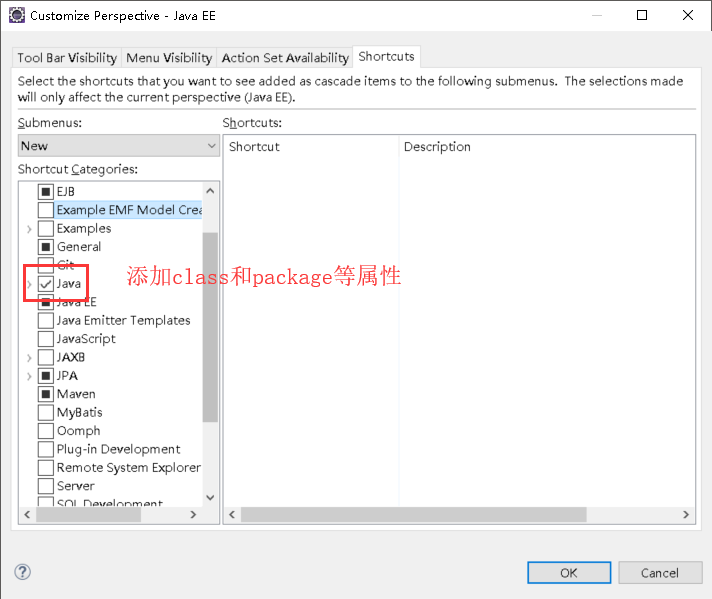
### 导入数据库

说明:将课前资料中的数据库文件导入到本机,通过工具导入.注意数据表的个数,如果个数不一致,使用京淘课前资料中的sqlyog工具导入.安装的过程参数资料,其中包含破解码.

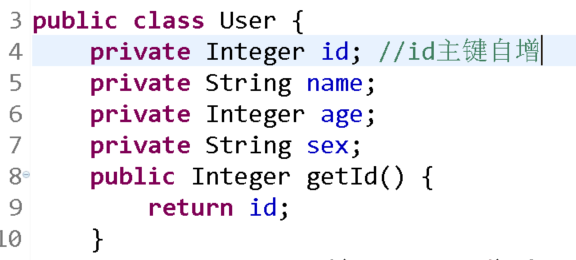


### 添加显示

路径:



### 编辑pojo对象

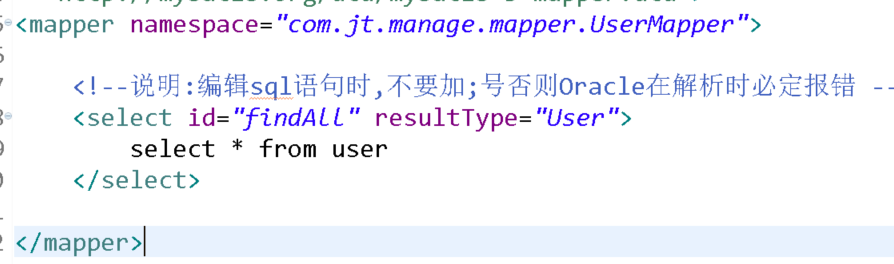


### 编辑Mapper接口和映射文件

1.编辑mapper接口

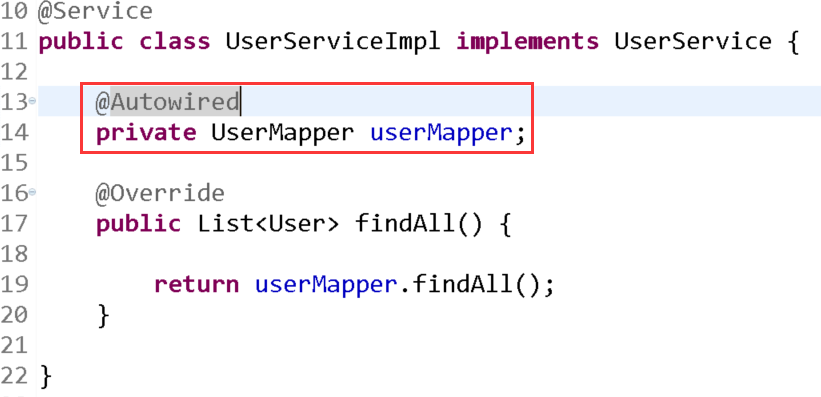


2.编辑Mappe的映射文件



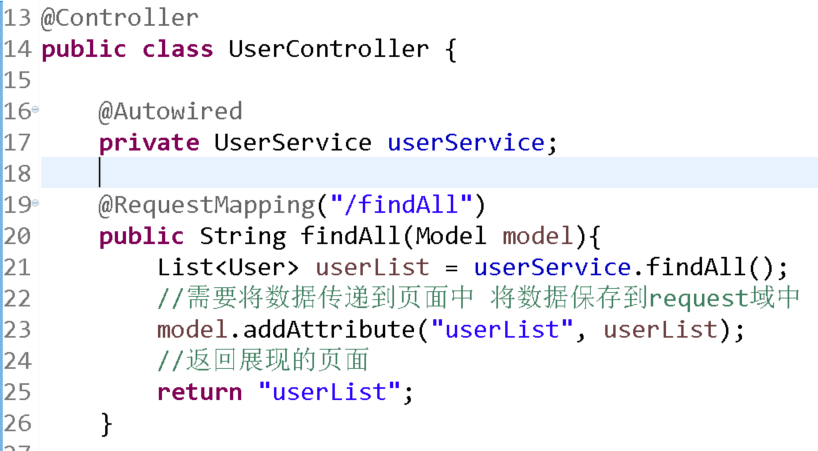
### 编辑Service

说明:编辑service实现接口方法



### 编辑controller

说明:通过controller调用service完成业务层操作



### 页面实现

1. 导入标签



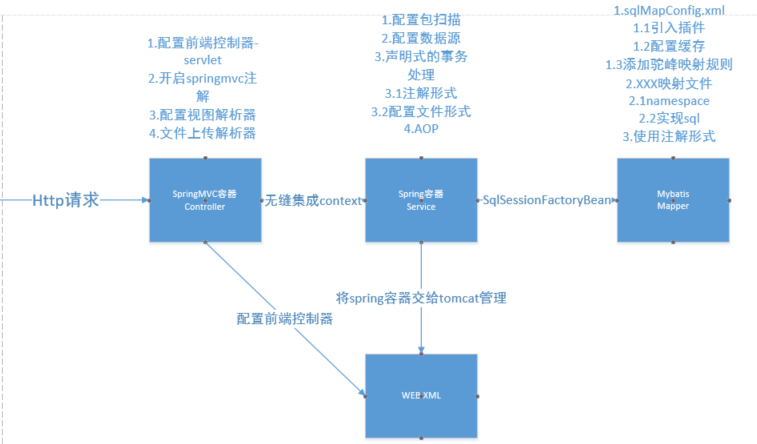
1. 循环遍历数据



# 配置文件编辑

## 架构设计

### 架构设计图



## SpringMVC配置

### 编辑applicationContext-mvc.xml

<!--1.开启mvc注解形式 -->

<mvc:annotation-driven/>

<!--2.配置内部资源视图解析器,配置前缀和后缀

注意:配置前缀是切记添加"/" 否则将来转向存在问题

-->

<bean id=*"viewResolver"* class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<!--set方法注入 -->

<property name=*"prefix"* value=*"/WEB-INF/views/"*/>

<property name=*"suffix"* value=*".jsp"*/>

</bean>

<!--3.配置文件上传视图解析器 -->

### 编辑web.xml配置文件

说明:编辑web.xml 配置前端控制器和过滤器

<!--配置前端控制器

说明:当有请求时,才会调用servlet实现请求的准发.

这时springMVC容器才会启动,用户等待的时间较长,

所以应该当tomcat启动时,就让springMVC和spring容器启动.

-->

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!--指定配置文件的路径 -->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:/spring/applicationContext-mvc.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<!--

拦截路劲的写法 前缀型路径.后缀型路径=全路径

\*.do 匹配以do结尾的请求

\*.action 匹配以action结尾的请求

/ 只匹配前缀型路径 /hello,

目的使用restFul传参方便

规则:/拦截前缀型的请求,同时拦截静态资源,

但是放行.jsp等动态资源

/\* 过滤器中使用/\* 全部的资源请求 禁止使用/\*

-->

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<!--全站乱码的解决 主要解决post乱码问题 -->

<filter>

<filter-name>characterEncoding</filter-name>

<filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>

<!--指定过滤器的字符集编码 -->

<init-param>

<param-name>encoding</param-name>

<param-value>UTF-8</param-value>

</init-param>

</filter>

<filter-mapping>

<filter-name>characterEncoding</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

## Spring配置

### 编辑spring的配置文件

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

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">

<!--1.配置包扫描 -->

<context:component-scan base-package="com.jt"/>

<!--2.引入外部的配置文件 -->

<bean id="placeholderConfigurer"

class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">

<!--设定参数 Resource[] locations;-->

<property name="locations">

<list>

<value>classpath:/properties/jdbc.properties</value>

</list>

</property>

</bean>

<!--说明:

1.改标签可以导入外部的properties文件,

但是只能导入一个或一类文件

jdbc\*.pro... 表示导入以jdbc开头的配置文件

\*.pro 配置所有的配置文件

-->

<!-- <context:property-placeholder

location="classpath:/properties/jdbc.properties"/> -->

<!--2.配置数据源 -->

<bean id="dataSource" class="com.jolbox.bonecp.BoneCPDataSource" destroy-method="close">

<!-- 数据库驱动 -->

<property name="driverClass" value="${jdbc.driver}" />

<!-- 相应驱动的jdbcUrl -->

<property name="jdbcUrl" value="${jdbc.url}" />

<!-- 数据库的用户名 -->

<property name="username" value="${jdbc.username}" />

<!-- 数据库的密码 -->

<property name="password" value="${jdbc.password}" />

<!-- 检查数据库连接池中空闲连接的间隔时间，单位是分，默认值：240，如果要取消则设置为0 -->

<property name="idleConnectionTestPeriod" value="60" />

<!-- 连接池中未使用的链接最大存活时间，单位是分，默认值：60，如果要永远存活设置为0 -->

<property name="idleMaxAge" value="30" />

<!-- 每个分区最大的连接数 -->

<property name="maxConnectionsPerPartition" value="150" />

<!-- 每个分区最小的连接数 -->

<property name="minConnectionsPerPartition" value="5" />

</bean>

<!--3.引入声明式的事务控制

1.注解形式 开启事务的注解

@Transactional(propagation=Propagation.REQUIRED)

注解形式的弊端:每一个方法都需要手动的添加事务控制,比较的繁琐

2.配置文件的方式 ~~~核心思想AOP~~~~~代理

通知+切入点(切入点表达式)

-->

<!-- <tx:annotation-driven/> -->

<!--3.0 配置事务管理器 -->

<bean id="transactionManager"

class="org.springframework.jdbc.datasource.DataSourceTransactionManager">

<property name="dataSource" ref="dataSource"/>

</bean>

<!--3.1配置声明式事务处理的通知

transactionManager:表示事务管理器,

需要控制事务是否提交/回滚,

如果事务管理器的名称是默认的,则可以省略该配置

事务的管理策略:

1.增/删/改 需要事务控制

2.查询 不需要事务,配置只读

3.其他方法 不需要事务,配置只读

事务的传播属性:

propagation="REQUIRED" 必须添加事务

propagation="SUPPORTS" 事务支持的()

propagation="NEVER" 不使用事务

-->

<tx:advice id="txAdvice" transaction-manager="transactionManager">

<tx:attributes>

<tx:method name="save\*" 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="\*" propagation="SUPPORTS" read-only="true"/>

</tx:attributes>

</tx:advice>

<!--3.2配置AOP的切面

回顾:AOP的动态代理方式 1.JDK 2.cglib动态代理

默认的代理方式:

如果实现了接口则采用jdk的动态代理方式

如果没有实现接口,采用cglib动态代理方式

proxy-target-class="true"

表示强制使用cgLib实现动态代理

补充说明:一般采用spring中的默认的策略.

知识回顾

表达式的写法:

1.within(包名.类名)

控制方式是类级别的,全部的方法都需要添加事务控制

2.execution(返回值类型 包名.类名.方法名(参数列表))

-->

<aop:config>

<!--配置切入点 -->

<aop:pointcut expression="execution(\* com.jt.manage.service..\*.\*(..))" id="pc"/>

<!--配置通知 -->

<aop:advisor advice-ref="txAdvice" pointcut-ref="pc"/>

</aop:config>

</beans>

## Mybatis配置

### 编辑核心配置文件

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE configuration

PUBLIC "-//mybatis.org//DTD Config 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-config.dtd">

<configuration>

<!--1.Mybatis的配置文件有顺序要求.

Content Model : (properties?, settings?, typeAliases?, typeHandlers?, objectFactory?,

objectWrapperFactory?, reflectorFactory?, plugins?, environments?, databaseIdProvider?,

mappers?)

2.功能:

1.添加mybatis自己的配置

3. mybatis的优点:

可以实现自动的对象关系映射(orm)

select \* from user 结果集 (反射技术) User对象

3.1 映射问题:

如果字段和对象的属性不一致时,使用resultType不能

实现自动的映射.那么必须按照要求编辑resultMap.

但是编辑resultMap的格式较为繁琐和复杂.所以除非特殊

的环境要求(多表关联查询),否则少使用resultMap

3.2解决方案:使用驼峰规则映射

<setting name="mapUnderscoreToCamelCase" value="true" />

说明:如果mybatis开启了驼峰映射规则,那么可以将

column= user\_name 和 User(username)

实现自动的映射

步骤:

1.根据sql语句获取结果集对象,之后通过反射规则进行对象的

赋值操作.

2.将字段中的"\_"去掉,动态匹配对象中的属性,进行赋值操作.

3.驼峰映射的现象: column= user\_name ~~~~~userName|userNAME|username....

只要字母相同即可.

4.Mybatis中的缓存

4.1一级缓存

级别:sqlSession级别

说明:在同一个sqlSession内实现数据共享

默认是开启的.

类比:相当于获取数据库链接,在一个链接上实现数据共享.

SqlSession sqlSession = connection.getSqlSession();

sqlSession.selectList("映射文件的路径.sqlId")

sqlSession.selectList("映射文件的路径.sqlId")

4.2二级缓存

级别:sqlSessionFactory

说明:由工厂模式所生产的sqlSession都实现数据共享.

补充:由于mybatis提供的二级缓存效率太低,开发中几乎不同.

-->

<settings>

<!-- 开启驼峰自动映射 -->

<setting name=*"mapUnderscoreToCamelCase"* value=*"true"* />

<!-- 二级缓存的总开关，被redis替代 -->

<setting name=*"cacheEnabled"* value=*"false"* />

</settings>

</configuration>

## Spring整合mybatis

### 编辑配置文件

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:util="http://www.springframework.org/schema/util"

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">

<!--Spring整合mybatis

整合步骤:

1.spring管理sqlSessionFactoryBean

2.配置数据源

3.添加核心配置文件

4.添加映射文件

5.配置别名包

6.为mapper接口生成代理对象(JDK代理)

-->

<bean id="sqlSessionFactoryBean" class="org.mybatis.spring.SqlSessionFactoryBean">

<!--配置数据源 -->

<property name="dataSource" ref="dataSource"/>

<!--添加mybatis核心配置文件 -->

<property name="configLocation"

value="classpath:/mybatis/mybatis-config.xml"/>

<!--添加映射文件 -->

<property name="mapperLocations"

value="classpath:/mybatis/mappers/\*.xml"/>

<!--导入别名包 -->

<property name="typeAliasesPackage"

value="com.jt.manage.pojo"/>

</bean>

<!--为mapper生成代理对象 -->

<bean class="org.mybatis.spring.mapper.MapperScannerConfigurer">

<property name="basePackage" value="com.jt.manage.mapper"/>

</bean>

</beans>

### 效果展现



# 作业:

根据配置文件的解析方式,画出程序从tomcat启动一直到数据正确返回的流程图.!!!!

说明:加载步骤

1. web.xml配置文件
2. 解析前端控制器

加载配置文件.顺序按照 spring springMVC mybatis顺序

1. 到此为止整个springMVC +spring+mybatis全部启动成功
2. 当用户发起localhost:8091/findAll时,
   1. 前端控制器将请求转发到Controller中,之后进行业务调用

Controller-service-mapper

1. 最终返回结果,经过视图解析器返回给用户数据和页面

