

# **说一说Spring的单独使用**

到了SSM整合这一步，那你肯定已经学过了这三个具体的框架了，让你说一说Spring的话，大概最容易想到的就是IOC和AOP，对了，使用Spring能干啥，能够IOC和AOP，那么该如何使用？

## **关于导包**

既然要用到Spring,那肯定要导入Spring的依赖开发包，那么，是不是什么aop，什么bean，什么context的那几个包，而我一般只会导入这一个包

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

为什么？因为我们大多数都是使用maven开发，而maven的一个好处，就是把相关依赖包会一并加载，至于其他的一些什么包，用到的时候再加呗！

## **离不开的AplicationContext.xml**

在Spring开发中，这个配置文件可以说是必须的了，那么这个配置文件是用来干嘛呢？配置一些东西，比如配置包扫描，说白了，只有配置了一些包扫描，你在某个包下才能使用注解开发，还会配置切面什么的，就是IOC和AOP的一些相关配置。

工欲善其事必先利其器，也就是说你如果想进行Spring开发，那你需要准备好你的武器，那就是加载必须的依赖包，配置好需要的配置文件即可！

## **bean的装配**

我们在学习的时候能直观感受到的就是bean的装配，就是IOC，你不需要自己使用new来创建对象，把类交给Spring来管理，然后从容器中获取对象。

那么如何把类交给Spring来管理，也就是如何进行bean的装配，你可能见到的首先是这样的形式

*<!--装配User类-->*

<bean id="user" class="com.ithuangqing.spring.pojo.User"/>

然后如何获取这个对象呢？

ApplicationContext applicationContext = new

ClassPathXmlApplicationContext("aplicationContext.xml");

User user = (User) applicationContext.getBean("user");

也就是先要获取容器对象，然后再获取这个User类。

以上bean的装配属于采用xml配置的方式，在实际情况中，这样的装配方式或许是比较多的，首先，配置包扫描

*<!--扫描配置包-->*

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

然后在具体的类上使用注解

**@Repository(value = "user")public** **class** **User**

然后同样是需要获取容器对象再获取这个类。

以上简单了说了bean的xml配置装配和使用注解装配的两种方式，那么这个AplicationContext.xml干啥的

**<?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 http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd">

*<!--装配User类-->*

<bean id="user" class="com.ithuangqing.spring.pojo.User"/>

*<!--扫描配置包-->*

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

</beans>

# **先说mybatis**

SSM按理说，应该是SpringMVC了吧，你应该也知道SpringMVC和Spring是可以无缝整合的，毕竟是一家子吗，反倒是这个mybatis，需要嫁入Spring这个豪门，怎么办，mybatis就整了一个和Spring整合的开发包

<dependency>

*<!--mybatis与spring整合包，正是这个我们不需要再手动获取SqlSession-->*

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.3</version>

</dependency>

注意了是mybatis-spring而不是spring-mybatis。这是归属权的问题，是mybatis带过去的嫁妆！

所以我觉得ssm整合的难点就在于如何将mybatis融入进去，下面说说mybatis

## **mapper接口动态代理开发**

首先看一下传统的不使用mapper动态代理开发我们使用的执行sql语句的方式

Student student = sqlSession.selectOne("com.ithuangqing.mybatis.dao.StudentDao.queryById", 1);

### ****SqlSession发送SQL****

就比如这个查找一条数据，如果不使用mapper动态代理开发的话，我们主要是使用sqlsession发送sql语句，那么如何确定这个执行语句呢？就需要mapper文件中的命名空间和sql语句的id来唯一确定这一个sql语句，然后执行的时候再跟上相应的参数。

但是实际上这样是比较麻烦的，在实际开发中都是使用mapper接口动态代理开发，像这样

StudentDao mapper = sqlSession.getMapper(StudentDao.class); Student student = mapper.queryById(3);

### ****使用Mapper接口发送SQL****

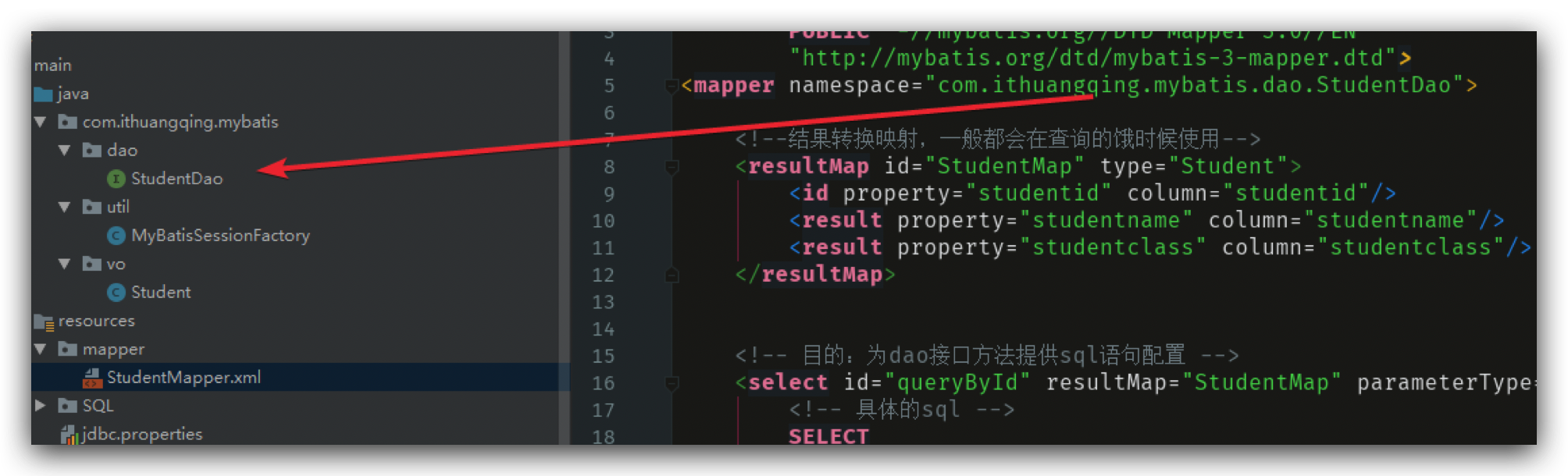
而且在实际开发中一般都是会有一个dao层来与数据库打交道，所以一般在实际开发中都是使用这种mapper接口发送sql语句的方式。

****使用mapper接口开发满足的点****

但是在使用mapper接口开发的时候也需要满足一些前提：

1. 接口中传入的参数和返回值类型必须和mapper文件中sql语句中的传参类型和返回值类型保持一致。
2. 接口中的方法名称要和mapper文件中的sql语句的ID一致。
3. 就是要把接口和mapper文件绑定在一起，这个主要通过将mapper文件中的命名空间写成接口的全限定名称。

## **再说mapper动态代理开发**



这里的这个命名空间是可以随便写的，在原始dao开发中是为了和sql语句的id结合来确定要执行的sql语句，比如下面这个

*/\*\**

*\* 根据id查找*

*\*/*

Student student = sqlSession.selectOne("com.ithuangqing.mybatis.dao.StudentDao.queryById", 1);

比如这个就是用命名空间加上sql语句的id来唯一确定要执行的这个sql语句。

但是在实际开发中都是使用Mapper的动态代理开发，所以这里的命名空间会写成接口类的全限定名称，这样的话就不需要实现类，可以直接使用mapper的动态代理开发。

那么动态代理开发是怎样的呢？经过上面的命名空间的改写就和实体类的接口对应上了，就可以这样实现查询

*/\*\**

*\* 动态代理开发*

*\*/*

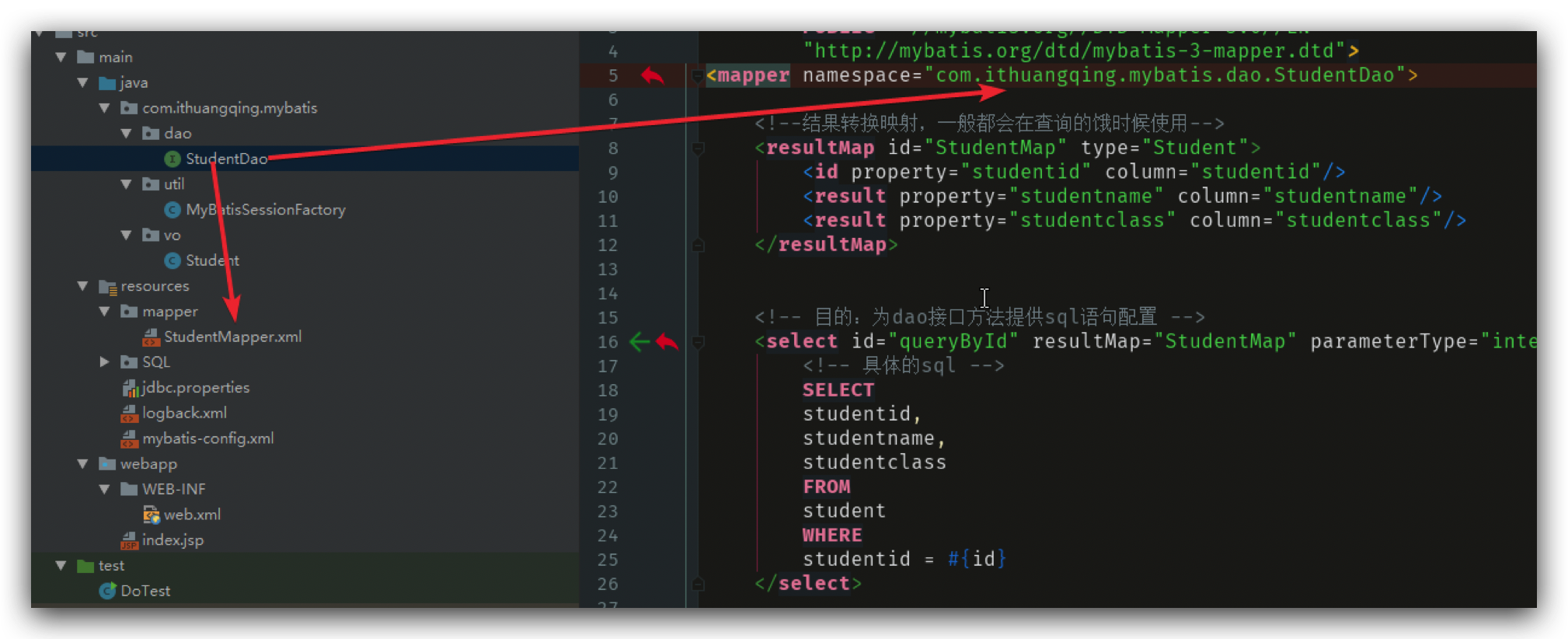
SqlSession sqlSession = MyBatisSessionFactory.getSqlSession();

StudentDao mapper = sqlSession.getMapper(StudentDao.class);

Student student = mapper.queryById(3);

System.out.println(student);

这是实际开发中用到的方式，很简单。



通过这样的方式就让接口和mapper文件关联了起来，然后就可以实现动态代理开发了，不过这里要有几个前提

1. mapper文件中的sql语句的id要和接口中的方法名称一致
2. mapper文件中的sql语句的接收参数类型要和接口中方法接收的参数类型一致
3. mapper文件中的sql语句的返回值类型要和接口中方法返回值类型一致
4. 命名空间来绑定接口

## **重点**

到这里你会发现，无论是mapper接口开发还是传统dao层开发的方式，我们都需要先获取sqlsession，这个使我们在执行sql语句前必须做的，因为关于sql的一些操作指令都在这个SqlSession当中，但是与Spring整合之后我们无需再手动的去获取这个SqlSession了，Spring已经帮我们去做这件事情了！

# **Spring和Mybatis的整合**

关于ssm整合的一个难点理解在于mybatis的处理，在单独使用mybatis的时候，如果我们使用动态代理开发，也就是mapper接口开发，我们需要这样来发送sql

SqlSession sqlSession = MyBatisSessionFactory.getSqlSession();

StudentDao mapper = sqlSession.getMapper(StudentDao.class);

Student student = mapper.queryById(3);

System.out.println(student);

也就是说我们需要先生成一个sqlsession，然后才能使用mapper接口，得到一个dao层接口，然后利用这个mapper接口发送sql。

所以重点是，单独使用mybatis我们需要自己生成这个sqlsession，但是与spring整合之后，我们就不需要手动的去生成这个sqlsession了，这个可以由spring来帮我们完成，主要靠如下的配置

*<!-- 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MyBaties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="com.ithuangqing.ssm.pojo" />

*<!-- 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Dao接口包 -->*

<property name="basePackage" value="com.ithuangqing.ssm.dao" />

</bean>

经过这个配置，我们就不需要再手动的去生成sqlssesion对象了，可以直接这么使用

**@Autowired**

**private** StudentDao studentDao;

**@Test**

**public** **void** **testQueryById**(){

Integer id = 201515113;

Student student = studentDao.queryById(id);

System.out.println(student);

}

## **Spring如何加载自己的配置文件**

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

通过这样的方式加载配置文件，就可以通过下面的方式拿到对象了

UserDao userdaoimpl = (UserDao) applicationContext.getBean("userdaoimpl")*;*

userdaoimpl.show()*;*

这个括号的参数是bean的id。

## **mybatis如何加载配置文件**

InputStream input = Resources.getResourceAsStream("mybatis-config.xml");

factory = **new** SqlSessionFactoryBuilder().build(input);

可以得出，mybatis配置文件的加载大体上就是为了得到sqlsession来执行sql语句，而spring配置文件的加载可以方便我们使用ioc和aop，那么两者整合之后

1. 将mybatis配置文件交给spring的配置文件，只需要加载spring的配置文件即可
2. 不用手动生成sqsession了

主要就是通过mybatis-spring这个整合包来实现，主要是以下的配置（spring配置）

*<!-- 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MyBaties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="com.ithuangqing.ssm.pojo" />

*<!-- 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Dao接口包 -->*

<property name="basePackage" value="com.ithuangqing.ssm.dao" />

</bean>

这个是整合后spring配置文件中的一部分整合mybatis的，可以看到mybatis的配置文件被放进这个配置里面了，然后加载的话只需要加载spring的这个配置文件即可，而不再需要加载mybaits的配置。

与SpringMVC的整合无非就是不需要这样加载spring配置文件了，

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

直接将配置文件放在SpringMVC的前端控制器中进行加载就实现了三者的整合

<servlet>

<servlet-name>seckill-dispatcher</servlet-name>

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

*<!-- 配置springMVC需要加载的配置文件*

*spring-dao.xml,spring-service.xml,spring-web.xml*

*Mybatis - > spring -> springmvc*

*-->*

<init-param>

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

<param-value>classpath:spring/spring-\*.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>seckill-dispatcher</servlet-name>

*<!-- 默认匹配所有的请求 不拦截jsp-->*

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

</servlet-mapping>

主要就是这个init-param。

# **SpringMVC的加入**

我们知道Spring的一个特性就是我们只要把类交给Spring处理，它就会帮我们创建对象，实际上是我们把我们的对象通过注解或者xml配置的方式装配成Spring容器中的一个bean，需要这个对象的时候我们需要从Spring容器中去获取，在Spring单独使用的时候，我们装配完成之后，要想获得bean，我们必须启动一个Spring容器，获得这个容器对象之后然后从这个容器中拿到我们需要的bean，一般是这样做

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

但是与SpringMVC整合后，我们需要在服务器中运行，这个时候就不需要再向上面那样加载配置文件获取容器对象了，我们可以在服务器启动的时候就加载配置文件从而获得容器对象，这个一般就在SpringMVC的前端控制器中完成加载。

也就是说

与SpringMVC的整合无非就是不需要这样加载spring配置文件了，

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

直接将配置文件放在SpringMVC的前端控制器中进行加载就实现了三者的整合

<servlet>

<servlet-name>seckill-dispatcher</servlet-name>

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

*<!-- 配置springMVC需要加载的配置文件*

*spring-dao.xml,spring-service.xml,spring-web.xml*

*Mybatis - > spring -> springmvc*

*-->*

<init-param>

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

<param-value>classpath:spring/spring-\*.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>seckill-dispatcher</servlet-name>

*<!-- 默认匹配所有的请求 不拦截jsp-->*

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

</servlet-mapping>

主要就是这个init-param。服务器在启动的时候就帮我们启动了Spring容器，我们的bean也都已经装配进去了，使用的时候直接使用注解的方式获取就行了！

# **总结！**

到这里，你可以发现，对于小白来说，直观的感受着三个框架，Spring就是IOC和AOP了，Mybatis就是获取SqlSession执行sql语句，SpringMVC就是一个前端控制器了。

而这些都跟配置文件有关，我们利用Mybatis和Spring的整合包可以将myabtis的配置写在Spring配置文件中，而SpringMVC与Spring又是一家的，配置文件依然是在一起的，因此在搭建SSM整合的时候，就需要一个配置文件，我叫它

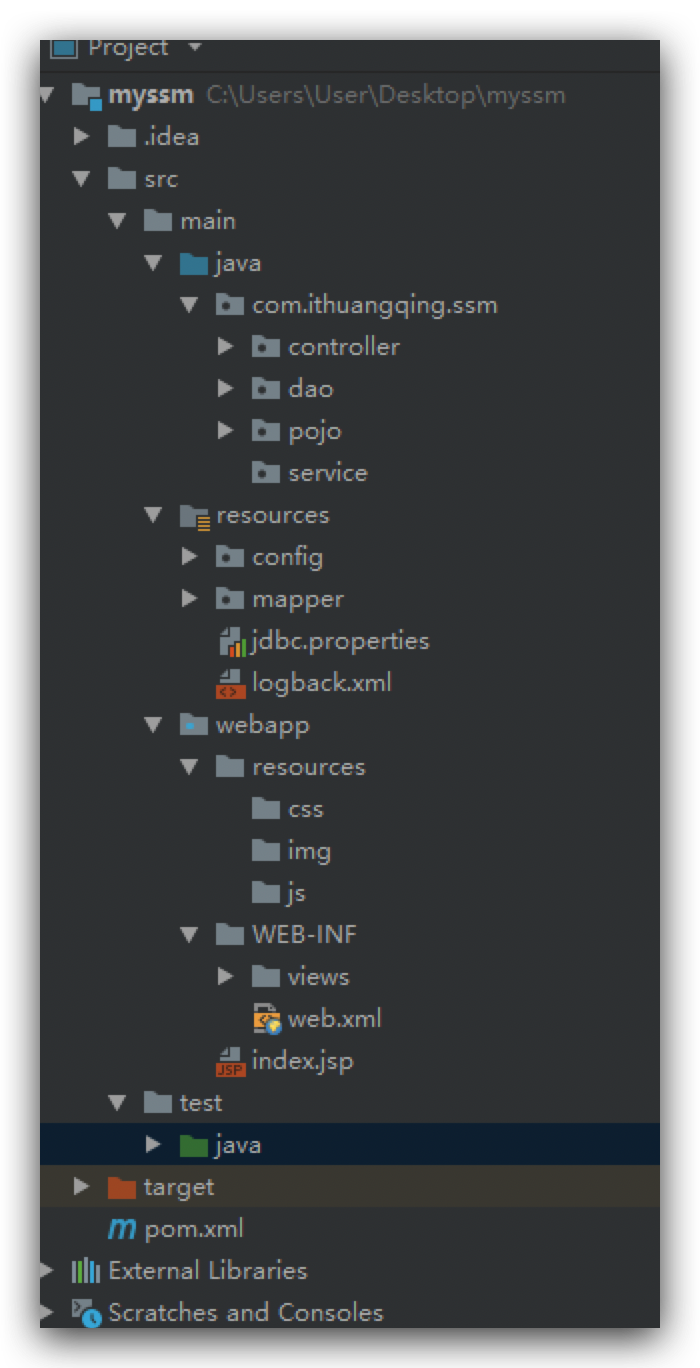
****ssm-config.xml****

# **开始搭建**

下面就正是开始搭建！

## **创建maven项目**

首先，我们创建一个maven项目，目录结构是这样的



这里只是创建了基础的四个包，分别是controller，dao，pojo和service，其实还可以有util包，需要了就再添加呗。

然后在资源文件下有两个文件夹，分别是config和mapper，这个config文件夹主要就是存放我们的配置文件，而mapper文件夹就是存放mapper映射文件的，然后还有一个jdbc.properties用来存放数据库的一些信息，而logback则是关于日志记录的，这些都是必备的。

然后在webapp的目录下的WEB-INF下有一个views文件夹主要就是存放jsp页面文件的，其他的，想必你一看就懂。

这个目录结构很简单！

## **加载相关依赖包**

接下来我们看看需要加入哪些依赖开发包。

### **1、测试包**

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.11</version>

<scope>test</scope>

</dependency>

### **2、日志记录**

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.1.1</version>

</dependency>

### **3、MySql**

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>5.1.37</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>c3p0</groupId>

<artifactId>c3p0</artifactId>

<version>0.9.1.2</version>

</dependency>

### **4、Servlet**

<dependency>

<groupId>taglibs</groupId>

<artifactId>standard</artifactId>

<version>1.1.2</version>

</dependency>

<dependency>

<groupId>jstl</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.5.4</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

### **5、Spring**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

### **6、SpringMVC**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.1.7.RELEASE</version>

</dependency>

### **7、mybatis**

<dependency>

<groupId>org.mybatis</groupId>

<artifactId>mybatis</artifactId>

<version>3.3.0</version>

</dependency>

<dependency>

*<!--mybatis与spring整合包，正是这个我们不需要再手动获取SqlSession-->*

<groupId>org.mybatis</groupId>

<artifactId>mybatis-spring</artifactId>

<version>1.2.3</version>

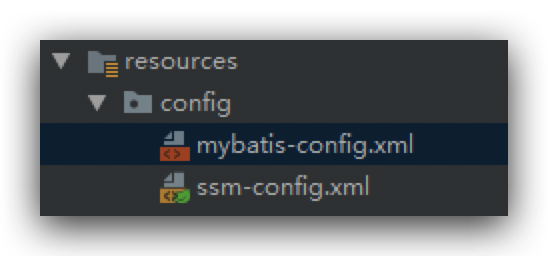
</dependency>

加载上面的这些依赖包，基本上都是必须的，你可以不知道其中有些包是干嘛的，但是你就这样做就对了。

## **重要的配置文件ssm-config.xml**

接下来就是非常重要的配置文件整合，整个整合的精髓就在这些配置文件上。

首先在config文件夹下创建这两个文件夹



所有的配置都是集中在这个ssm-config中的，至于这里还有一个mybatis-config文件我是参考网上的一些说法，因为有些配置可以单独分出来，比如这个全局配置文件

## **mybatis-config.xml**

**<?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>

*<!-- 配置全局属性 -->*

<settings>

*<!-- 使用jdbc的getGeneratedKeys获取数据库自增主键值 -->*

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

*<!-- 使用列别名替换列名 -->*

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

*<!-- 开启驼峰命名转换 -->*

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

</settings>

</configuration>

## **开始配置ssm-config.xml**

### **配置包大扫描**

<**context**:**component**-scan base-**package**="com.ithuangqing.ssm" />

### **1、配置数据库相关属性**

<**context**:**property**-placeholder location="classpath:jdbc.properties" />

### **2、配置数据库连接池**

<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.url}" />

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

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

*<!-- c3p0连接池的私有属性 -->*

<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MyBaties全局配置文件:mybatis-config.xml -->*

<property name="configLocation" value="classpath:config/mybatis-config.xml" />

*<!-- 扫描pojo包 并且使用别名 -->*

<property name="typeAliasesPackage" value="com.ithuangqing.ssm" />

*<!-- 加载mapper映射文件 -->*

<property name="mapperLocations" value="classpath:mapper/\*.xml" />

</bean>

### **4、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Dao接口包 -->*

<property name="basePackage" value="com.ithuangqing.ssm.dao" />

</bean>

### **5、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" />

### **6、开启SpringMVC注解模式**

**<mvc:annotation-driven/>**

### **7、支持静态资源**

<mvc:**default**-servlet-**handler**/>

### **8、配置视图解析器**

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

<**property** name="viewClass" value="org.springframework.web.servlet.view.JstlView" />

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

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

</bean>

好了，就是这些配置！

### **简单说下**

以上的这些配置，基本上遵循最少原则，够用即可！

# **在前端控制器中加载**

**<?xml version="1.0" encoding="UTF-8"?>**<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\_4\_0.xsd"

version="4.0">

*<!-- 配置DispatcherServlet -->*

<servlet>

<servlet-name>dispatcher</servlet-name>

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

*<!-- 配置springMVC需要加载的配置文件-->*

<init-param>

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

<param-value>classpath:config/ssm-config.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

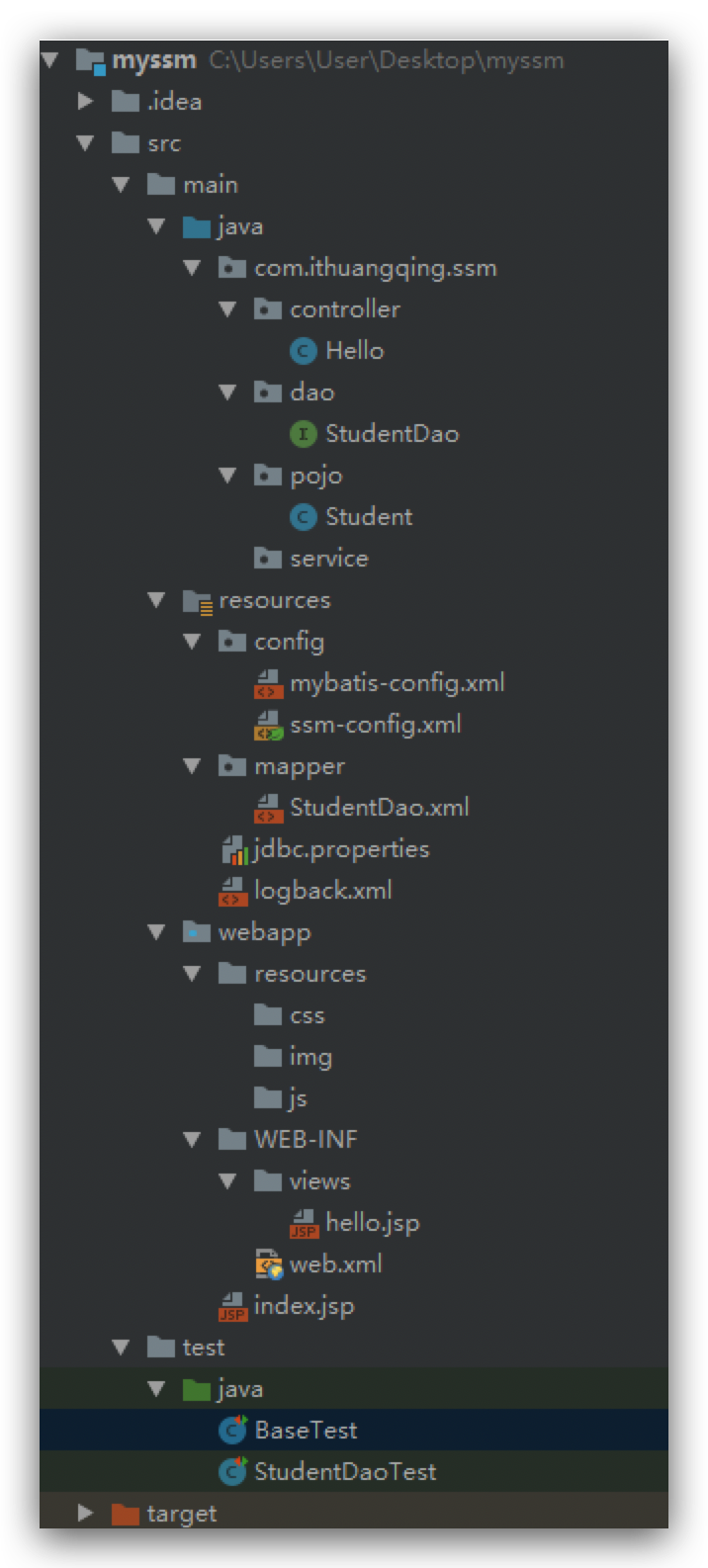
*<!-- 默认匹配所有的请求 不拦截jsp-->*

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

</servlet-mapping></web-app>

到这里，全部搞定。

# **最终的目录结构**



# **补充：建表语句**

**SET** FOREIGN\_KEY\_CHECKS=0;

*-- ------------------------------ Table structure for `student`-- ----------------------------***DROP** **TABLE** **IF** **EXISTS** `student`;**CREATE** **TABLE** `student` (

`studentname` varchar(20) **NOT** NULL,

`studentid` int(10) **NOT** NULL,

`studentclass` varchar(20) **NOT** NULL,

PRIMARY **KEY** (`studentid`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8;

# **使用方法**

你可以将项目克隆或者下载下来，然后修改包名。

如果你想测试一下，首先创建一个student数据库，然后创建表student，以下是建表语句

**SET** FOREIGN\_KEY\_CHECKS=0;

*-- ------------------------------ Table structure for `student`-- ----------------------------***DROP** **TABLE** **IF** **EXISTS** `student`;**CREATE** **TABLE** `student` (

`studentname` varchar(20) **NOT** NULL,

`studentid` int(10) **NOT** NULL,

`studentclass` varchar(20) **NOT** NULL,

PRIMARY **KEY** (`studentid`)

) **ENGINE**=**InnoDB** **DEFAULT** **CHARSET**=utf8;

**INSERT** **INTO** `student` **VALUES** ('庆哥小白', '101', '公众号');

然后修改jdbc.properties文件

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql:*//localhost:3306/student?useUnicode=true&characterEncoding=utf8*

jdbc.username=

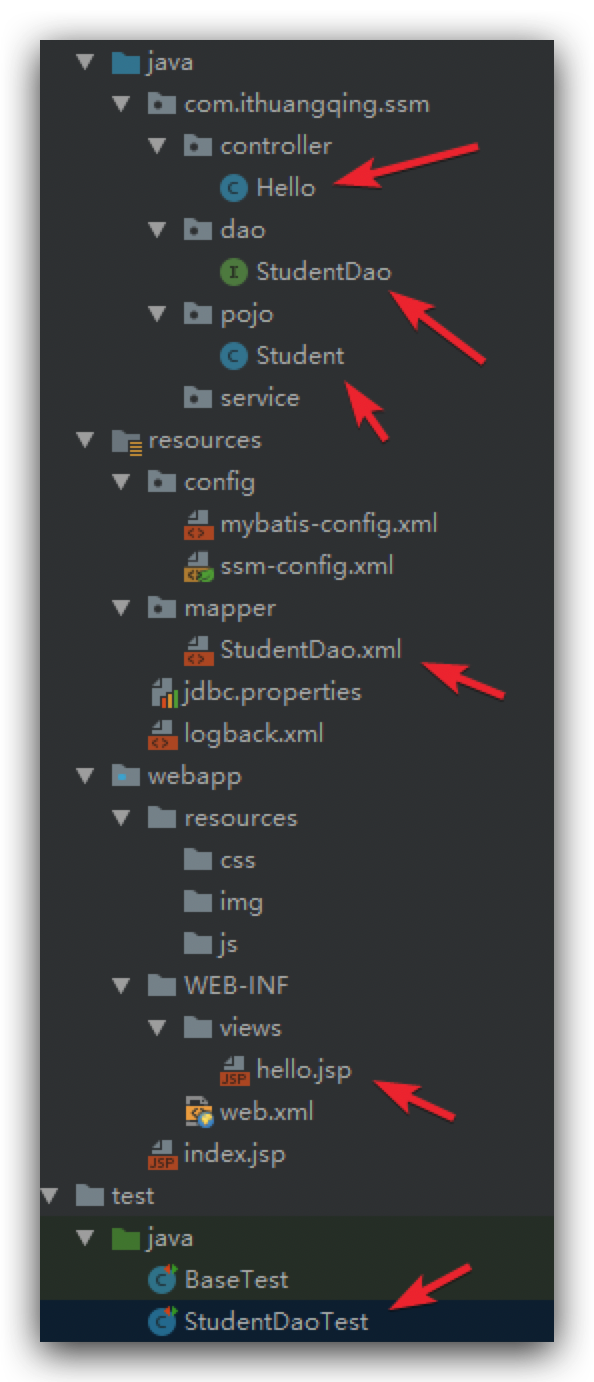
jdbc.password=

写上你的数据库账户和密码，然后执行测试代码StudentDaoTest即可得到



如果你想要一个全新的SSM框架，你可以这样做

1. 克隆或者下载本项目
2. 修改成自己的包名
3. 看图



删除这些文件，就是一个清爽原始的ssm整合后的项目了，你就可以进行自己的项目开发了！

# 如何获取被Spring装配的bean

我们知道Spring的一个特性就是我们只要把类交给Spring处理，它就会帮我们创建对象，实际上是我们把我们的对象通过注解或者xml配置的方式装配成Spring容器中的一个bean，需要这个对象的时候我们需要从Spring容器中去获取，在Spring单独使用的时候，我们装配完成之后，要想获得bean，我们必须启动一个Spring容器，获得这个容器对象之后然后从这个容器中拿到我们需要的bean，一般是这样做

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

但是与SpringMVC整合后，我们需要在服务器中运行，这个时候就不需要再向上面那样加载配置文件获取容器对象了，我们可以在服务器启动的时候就加载配置文件从而获得容器对象，这个一般就在SpringMVC的前端控制器中完成加载。

也就是说

与SpringMVC的整合无非就是不需要这样加载spring配置文件了，

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

直接将配置文件放在SpringMVC的前端控制器中进行加载就实现了三者的整合

<servlet>

<servlet-name>dispatcher</servlet-name>

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

*<!-- 配置springMVC需要加载的配置文件-->*

<init-param>

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

<param-value>classpath:spring/spring-\*.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

*<!-- 默认匹配所有的请求 不拦截jsp-->*

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

</servlet-mapping>

主要就是这个init-param。服务器在启动的时候就帮我们启动了Spring容器，我们的bean也都已经装配进去了，使用的时候直接使用注解的方式获取就行了！

# Spring和Mybatis的整合

关于ssm整合的一个难点理解在于mybatis的处理，在单独使用mybatis的时候，如果我们使用动态代理开发，也就是mapper接口开发，我们需要这样来发送sql

SqlSession sqlSession = MyBatisSessionFactory.getSqlSession();

StudentDao mapper = sqlSession.getMapper(StudentDao.class);

Student student = mapper.queryById(3);

System.out.println(student);

也就是说我们需要先生成一个sqlsession，然后才能使用mapper接口，得到一个dao层接口，然后利用这个mapper接口发送sql。

所以重点是，单独使用mybatis我们需要自己生成这个sqlsession，但是与spring整合之后，我们就不需要手动的去生成这个sqlsession了，这个可以由spring来帮我们完成，主要靠如下的配置

*<!-- 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MyBaties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="com.ithuangqing.ssm.pojo" />

*<!-- 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Dao接口包 -->*

<property name="basePackage" value="com.ithuangqing.ssm.dao" />

</bean>

经过这个配置，我们就不需要再手动的去生成sqlssesion对象了，可以直接这么使用

**@Autowired**

**private** StudentDao studentDao;

**@Test**

**public** **void** **testQueryById**(){

Integer id = 201515113;

Student student = studentDao.queryById(id);

System.out.println(student);

}

## **Spring如何加载自己的配置文件**

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

通过这样的方式加载配置文件，就可以通过下面的方式拿到对象了

UserDao userdaoimpl = (UserDao) applicationContext.getBean("userdaoimpl")*;*

userdaoimpl.show()*;*

这个括号的参数是bean的id。

## **mybatis如何加载配置文件**

InputStream input = Resources.getResourceAsStream("mybatis-config.xml");

factory = **new** SqlSessionFactoryBuilder().build(input);

可以得出，mybatis配置文件的加载大体上就是为了得到sqlsession来执行sql语句，而spring配置文件的加载可以方便我们使用ioc和aop，那么两者整合之后

1. 将mybatis配置文件交给spring的配置文件，只需要加载spring的配置文件即可
2. 不用手动生成sqsession了

主要就是通过mybatis-spring这个整合包来实现，主要是以下的配置（spring配置）

*<!-- 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MyBaties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="com.ithuangqing.ssm.pojo" />

*<!-- 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Dao接口包 -->*

<property name="basePackage" value="com.ithuangqing.ssm.dao" />

</bean>

这个是整合后spring配置文件中的一部分整合mybatis的，可以看到mybatis的配置文件被放进这个配置里面了，然后加载的话只需要加载spring的这个配置文件即可，而不再需要加载mybaits的配置。

与SpringMVC的整合无非就是不需要这样加载spring配置文件了，

ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");

直接将配置文件放在SpringMVC的前端控制器中进行加载就实现了三者的整合

<servlet>

<servlet-name>dispatcher</servlet-name>

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

*<!-- 配置springMVC需要加载的配置文件*

*spring-dao.xml,spring-service.xml,spring-web.xml*

*Mybatis - > spring -> springmvc*

*-->*

<init-param>

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

<param-value>classpath:spring/spring-\*.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

*<!-- 默认匹配所有的请求 不拦截jsp-->*

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

</servlet-mapping>

主要就是这个init-param。

# 关于ssm整合的一些零碎知识点

## 保证使用mapper动态代理开发

首先看一下传统的不使用mapper动态代理开发我们使用的执行sql语句的方式

Student student = sqlSession.selectOne("com.ithuangqing.mybatis.dao.StudentDao.queryById", 1);

### ****SqlSession发送SQL****

就比如这个查找一条数据，如果不使用mapper动态代理开发的话，我们主要是使用sqlsession发送sql语句，那么如何确定这个说起来语句呢？就需要mapper文件中的命名空间和sql语句的id来唯一确定这一个sql语句，然后执行的时候再跟上响应的参数。

但是实际上这样是比较麻烦的，在实际开发中都是使用mapper接口动态代理开发，像这样

StudentDao mapper = sqlSession.getMapper(StudentDao.class); Student student = mapper.queryById(3);

## ****使用Mapper接口发送SQL****

而且在实际开发中一般都是会有一个dao层来与数据库打交道，所以一般在实际开发中都是使用这种mapper接口发送sql语句的方式。

### **使用mapper接口开发满足的点**

但是在使用mapper接口开发的时候也需要满足一些前提：

1. 接口中传入的参数和返回值类型必须和mapper文件中sql语句中的传参类型和返回值类型保持一致。
2. 接口中的方法名称要和mapper文件中的sql语句的ID一致。
3. 就是要把接口和mapper文件绑定在一起，这个主要通过将mapper文件中的命名空间写成接口的全限定名称。