1. 建立工程：

file->new->Moudle->maven->maven-archetype-webapp

1. (等项目创建完成后再做这步操作）在pox.xml中增加如下内容（项目依赖）

<packaging>war</packaging>

<name>IA Maven Webapp</name>  
<!-- *FIXME change it to the project's website* -->  
<url>http://www.example.com</url>  
  
<properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <spring.version>5.0.2.RELEASE</spring.version>  
 <slf4j.version>1.6.6</slf4j.version>  
 <log4j.version>1.2.12</log4j.version>  
 <mysql.version>8.0.16</mysql.version>  
 <mybatis.version>3.5.2</mybatis.version>  
</properties>  
  
<dependencies>  
 <!-- spring -->  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.6.8</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-web</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-test</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-tx</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-jdbc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.12</version>  
 <scope>compile</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>${mysql.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>javax.servlet</groupId>  
 <artifactId>servlet-api</artifactId>  
 <version>2.5</version>  
 <scope>provided</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>javax.servlet.jsp</groupId>  
 <artifactId>jsp-api</artifactId>  
 <version>2.0</version>  
 <scope>provided</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>jstl</groupId>  
 <artifactId>jstl</artifactId>  
 <version>1.2</version>  
 </dependency>  
  
 <!-- log start -->  
 <dependency>  
 <groupId>log4j</groupId>  
 <artifactId>log4j</artifactId>  
 <version>${log4j.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-api</artifactId>  
 <version>${slf4j.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.slf4j</groupId>  
 <artifactId>slf4j-log4j12</artifactId>  
 <version>${slf4j.version}</version>  
 </dependency>  
  
 <!-- log end -->  
 <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis</artifactId>  
 <version>${mybatis.version}</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.mybatis</groupId>  
 <artifactId>mybatis-spring</artifactId>  
 <version>1.3.0</version>  
 </dependency>  
  
 <dependency>  
 <groupId>c3p0</groupId>  
 <artifactId>c3p0</artifactId>  
 <version>0.9.1.2</version>  
 <type>jar</type>  
 <scope>compile</scope>  
 </dependency>  
</dependencies>  
  
<build>  
 <finalName>IA</finalName>  
 <pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->  
 <plugins>  
 <plugin>  
 <artifactId>maven-clean-plugin</artifactId>  
 <version>3.0.0</version>  
 </plugin>  
 <!-- see http://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_war\_packaging -->  
 <plugin>  
 <artifactId>maven-resources-plugin</artifactId>  
 <version>3.0.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.7.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-surefire-plugin</artifactId>  
 <version>2.20.1</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-war-plugin</artifactId>  
 <version>3.2.0</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-install-plugin</artifactId>  
 <version>2.5.2</version>  
 </plugin>  
 <plugin>  
 <artifactId>maven-deploy-plugin</artifactId>  
 <version>2.8.2</version>  
 </plugin>  
 </plugins>  
 </pluginManagement>  
</build>

1. 完善项目目录

在src的main目录下创建两个文件夹java（写代码）和resources（放配置文件），在java上右键选择Mark directory as->Sources Root ,在resources上右键选择Mark directory as->Resources Root

1. 创建如下包

在java上右键选择new->package,名字为it.ztzq.domain

在java上右键选择new->package,名字为it.ztzq.dao

在java上右键选择new->package,名字为it.ztzq.service

在java上右键选择new->package,名字为it.ztzq.controller

**----------------------------------------------搭建spring框架----------------------------------------------------**

1. 创建javabean,实现序列化的接口，设置属性，生成get和set方法

在domain上右键选择new->Java Class->Account

1. 创建dao的接口

在dao上右键选择new->Java Class->选择Interface,命名为IAccountDao

定义两个方法

public List<Account> findAll();  
public void saveAccount(Account account);

dao无需写实现类，因为mybatis框架会生成代理对象，从而实现功能。

1. 创建service接口

在service上右键选择new->Java Class->选择Interface,命名为IAccountService

定义两个方法（跟dao接口的中的方法一样）

public List<Account> findAll();  
public void saveAccount(Account account);

service接口需要自己定义实现类

1. 在service上创建包impl，并在这个包上创建类AccountServiceImpl

在service上右键选择new->package,名字为impl

在impl上右键选择new->JavaClass,名字为AccountServiceImpl，并继承IAccountService接口，实现接口中的两个方法

public class AccountServiceImpl implements IAccountService {  
  
 @Override  
 public List<Account> findAll() {  
 System.*out*.println("业务层，查询所有账户。。。。");  
 return null;  
 }  
  
 @Override  
 public void saveAccount(Account account) {  
 System.*out*.println("业务层，保存账户。。。。");  
 }  
}

1. 在Controller上创建类 AccountController

在Controller上右键选择new->Java Class,名字为AccountController

1. 将service类和dao对象交给IOC的容器去做管理，并且使用注解（有配置文件，开启注解扫描，在方法或者类上加注解）的方式，创建spring的配置文件

在resources上右键选择new->XML Configuration File->spring config，名字为applicationContext

首先在xml文件中增加约束

<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:aop="http://www.springframework.org/schema/aop"  
 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/aop  
 http://www.springframework.org/schema/aop/spring-aop.xsd  
 http://www.springframework.org/schema/tx  
 http://www.springframework.org/schema/tx/spring-tx.xsd">

然后增加注解扫描的配置，spring只扫描it.ztzq下面的service和dao，不扫描controller

<!--开启注解扫描,只扫描service和dao，不扫描controller-->  
<context:component-scan base-package="it.ztzq">  
 <!--配置哪些注解不进行扫描-->  
 <context:exclude-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  
</context:component-scan>

在service上增加注解，这样相当于把AccountServiceImpl这个类交给了IOC管理

@Service(“accountservice”)  
public class AccountServiceImpl implements IAccountService {

1. 在it.ztzq下创建一个test包，并在test上增加一个类用来测试spring框架是否可用

在java上右键选择new->package,名字为it.ztzq.test

在test上右键选择new->Java Class,名字为TestSpring

1. 在TestSpring中编写测试方法

public class TestSpring {  
 @Test  
 public void run(){  
 //加载spring的配置文件  
 ApplicationContext ac = new ClassPathXmlApplicationContext("applicationContext.xml");  
 //获取对象  
 IAccountService as = (IAccountService)ac.getBean("accountService");  
 //调用方法  
 as.findAll();  
 }  
}

**---------------------搭建springMVC框架-----------------**

1. 在webapp下的WEB-INF下的web.xml进行配置

<web-app>  
 <display-name>Archetype Created Web Application</display-name>  
 <!--配置前端控制器-->  
 <servlet>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  
 <!--加载Springmvc.xml配置文件-->  
 <init-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:springmvc.xml</param-value>  
 </init-param>  
 <!--启动服务器，创建该servlet-->  
 <load-on-startup>1</load-on-startup>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>dispatcherServlet</servlet-name>  
 <url-pattern>/</url-pattern>  
 </servlet-mapping>  
  
 <!--解决中文乱码的过滤器-->  
 <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>  
 </filter>  
 <filter-mapping>  
 <filter-name>characterEncodingFilter</filter-name>  
 <url-pattern>/\*</url-pattern>  
 </filter-mapping>  
</web-app>

1. 增加web.xml中提到的springmvc.xml

在resources上右键选择new->XML Configuration File->spring config,名字为springmvc

首先在xml中增加约束

<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:context="http://www.springframework.org/schema/context"  
 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.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd">

增加如下配置

<!--开启注解扫描，只扫描controller-->  
<context:component-scan base-package="it.ztzq">  
 <context:include-filter type="annotation" expression="org.springframework.stereotype.Controller"/>  
</context:component-scan>  
  
<!--配置视图解析对象-->  
<bean id="internalResourceViewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">  
 <property name="prefix" value="/WEB-INF/pages/"/>  
 <property name="suffix" value=".jsp"/>  
</bean>  
  
<!--过滤静态资源-->  
<mvc:resources location="/css/" mapping="/css/\*\*" />  
<mvc:resources location="/images/" mapping="/images/\*\*" />  
<mvc:resources location="/js/" mapping="/js/\*\*" />  
  
<!--开启springmvc注解支持-->  
<mvc:annotation-driven/>

1. 在Controller包中的AccountController中实现方法findAll，并且添加注解，使点击链接时能根据链接地址跳转到此方法进行执行，返回为页面跳转的jsp的名字，会通过springmvc.xml中的视图解析器，跳转到WEB-INFO/pages/list.jsp页面如下：

@Controller  
@RequestMapping("/account")  
public class AccountController {  
 @RequestMapping("/findAll")  
 public String findAll(){  
 System.*out*.println("表现层：查询所有账户。。。。");  
 return "list";  
 }  
}

1. 在WEB-INFO下创建文件夹pages，在pages下创建jsp文件list.jsp,并且在页面上显示一句“查询了所有账户信息”，内容如下：

<body>  
 <h3>查询了所有账户信息</h3>  
</body>

1. 进行项目部署

在run->Edit configuration中选择Tomcat Server,在server插页将On “update” action 选择为Redeploy，在Deployment中点击+，选择需要部署的项目IA:war exploded,点击OK，部署完成

1. 点击右上角的，则启动了tomcat服务器,点击链接跳转成功，且表现层方法已经执行。

**-------------------springMVC框架和spring框架整合-----------------**

**整合成功：点击超链接发请求，执行controller的方法，在controller方法中应该调用service，如果能成功调用到service的方法，则表示整合成功了。**

**解决方案：在controller中调用service的方法，需要在controller中有service的对象，所以需要往依赖注入的方向思考，controller已经放到容器中了，如果把service也放入容器中，在controller中定义service的成员属性，就可以注入service对象，然后就可以调用方法，controller的配置文件在启动服务器时加载了（在web.xml中配置了启动服务器时加载springmvc.xml，所以才进行了springmvc.xml中配置的包的扫描）**

1. 在启动服务器时加载spring的配置文件applicationContext.xml(spring提供了一个监听器可以实现这个配置文件的加载），在web.xml中配置监听器

<!--配置监听器,默认只加载WEB-INF目录下的applicationContext.xml.所以需要修改配置文件的路径-->  
<listener>  
 <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
</listener>  
<!--设置配置文件的路径-->  
<context-param>  
 <param-name>contextConfigLocation</param-name>  
 <param-value>classpath:applicationContext.xml</param-value>  
</context-param>

1. 在controller中增加成员属性accountservice，添加依赖注入，并且调用service中的方法

@Controller  
@RequestMapping("/account")  
public class AccountController {  
  
 @Autowired  
 private IAccountService accountService;  
  
 @RequestMapping("/findAll")  
 public String findAll(){  
 System.*out*.println("表现层：查询所有账户。。。。");  
 //调用service方法  
 accountService.findAll();  
 return "list";  
 }  
}

如果注入成功，则运行项目，会打印出 表现层和业务层两条日志。

**-------------------搭建Mybatis框架-----------------**

**采用的是主配置文件SqlMapConfig.xml+注解的方式**

1. 在AccountDao中添加注解（查询sql语句的注解）

public interface IAccountDao {  
  
 @Select("select \* from account")  
 public List<Account> findAll();  
  
 @Insert("insert into account(name,money) values(#{name},#{money})")  
 public void saveAccount(Account account);  
}

1. 编写核心配置文件SqlMapConfig.xml

首先在xml中添加约束

<?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">

增加根标签<configuration>,在<configuration>中进行配置如下：

<configuration>  
 <!--配置环境-->  
 <environments default="mysql">  
 <environment id="mysql">  
 <transactionManager type="JDBC"></transactionManager>  
 <dataSource type="POOLED">  
 <property name="driver" value="com.mysql.cj.jdbc.Driver"/>  
 <property name="url" value="jdbc:mysql:///IA?serverTimezone=UTC"/>  
 <property name="username" value="root"/>  
 <property name="password" value="1234"/>  
 </dataSource>  
 </environment>  
 </environments>  
  
 <!--引入映射配置文件-->  
 <mappers>  
 <!--<mapper class="it.ztzq.dao.IAccountDao"/>-->  
 <package name="it.ztzq.dao"/>  
 </mappers>  
</configuration>

1. 测试mybatis框架

在test中增加TestMyBatis类，内容如下：

public class TestMyBatis {  
  
 @Test  
 public void run() throws Exception{  
 //加载配置文件  
 InputStream in = Resources.getResourceAsStream("SqlMapConfig.xml");  
 //创建SqlSessionFactory对象  
 SqlSessionFactory factory = new SqlSessionFactoryBuilder().build(in);  
 //创建SqlSession对象  
 SqlSession session = factory.openSession();  
 //获取代理对象  
 IAccountDao dao = session.getMapper(IAccountDao.class);  
 //调用dao方法  
 List<Account> list = dao.findAll();  
 for(Account account : list){  
 System.out.println(account);  
 }  
 //关闭资源  
 session.close();  
 in.close();  
 }  
}

运行输出了account表中的数据，则MyBatis框架搭建成功。

**-----------------Spring框架和Mybatis框架整合-----------------**

**整合成功：service能够成功调用到dao对象，可以做数据查询或者把数据存入到数据库中。service已经放在IOC容器里面了，dao是一个接口，可以生成代理对象，如果能把代理对象放到IOC容器内，在service中声明一个dao类型的属性，然后注入依赖。**

1. 在applicationConfig.xml中增加如下配置

<!--spring整合MyBatis框架-->  
  
<!--配置连接池-->  
<bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource">  
 <property name="driverClass" value="com.mysql.jdbc.Driver"/>  
 <property name="jdbcUrl" value="jdbc:mysql:///IA"/>  
 <property name="user" value="root"/>  
 <property name="password" value="1234"/>  
  
</bean>  
  
<!--配置SQLSessionFactory工厂-->  
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">  
 <property name="dataSource" ref = "dataSource"/>  
</bean>  
  
<!--配置AccountDao接口所在包-->  
<bean id="mapperScanner" class="org.mybatis.spring.mapper.MapperScannerConfigurer">  
 <property name="basePackage" value="it.ztzq.dao"/>  
</bean>

有了这个配置，就可以把SqlMapConfig.xml删掉了。

1. 在IAccountDao上增加注解，使其交给IOC容器进行管理

@Repository  
public interface IAccountDao {  
  
 @Select("select \* from account")  
 public List<Account> findAll();  
  
 @Insert("insert into account(name,money) values(#{name},#{money})")  
 public void saveAccount(Account account);  
}

1. 在AccountServiceImpl中注入IAccountDao这个接口，并进行调用

@Service("accountService")  
public class AccountServiceImpl implements IAccountService {  
  
 @Autowired  
 private IAccountDao accountDao;  
  
 @Override  
 public List<Account> findAll() {  
 System.out.println("业务层，查询所有账户。。。。");  
 return accountDao.findAll();  
 }  
  
 @Override  
 public void saveAccount(Account account) {  
 System.out.println("业务层，保存账户。。。。");  
 }  
}

对findAll()进行测试

关于保存，需要手动进行事务的管理，增加声明式的事务管理

<!--配置spring框架声明式事务管理-->  
  
<!--配置事务管理器-->  
<bean id="transactionManager" class="org.springframework.jdbc.datasource.DataSourceTransactionManager">  
 <property name="dataSource" ref="dataSource"/>  
</bean>  
<!--配置事务通知-->  
<tx:advice id="txAdvice" transaction-manager="transactionManager">  
 <tx:attributes>  
 <tx:method name="find\*" read-only="true"/>  
 <tx:method name="\*" isolation="DEFAULT"/>  
 </tx:attributes>  
</tx:advice>  
  
<!--配置AOP增强-->  
<aop:config>  
 <aop:advisor advice-ref="txAdvice" pointcut="execution(\* it.ztzq.service.impl.\*ServiceImpl.\*(..))"/>  
</aop:config>