Maven说明与项目总结思考

黄复贵

2016/11/5

**【综述】**在此之前对网站开发比较感兴趣，接触过javaweb jsp，javabean等，使用过SSM框架写小demo，但最重要的SSM框架配置文件，Spring框架的思想却没有好好掌握领悟。因此想通过这次小作业，好好总结Spring的思想，SSM框架的配置文件如何准确书写。

说明：本项目在IntelliJ IDEA 2016.2.2下开发，项目架构为Maven+ SpringMVC+Spring+MyBatis.

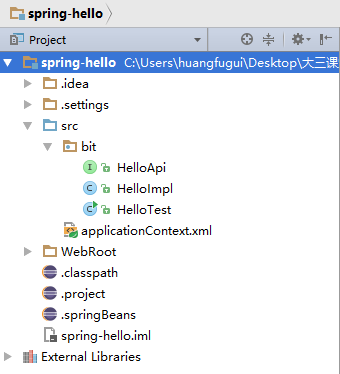
**Maven：**

首先项目是基于project object model思想的maven进行管理的，一直很喜欢用maven进行管理，个人觉得maven有以下好处：

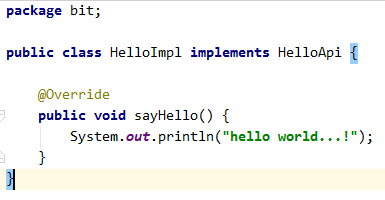
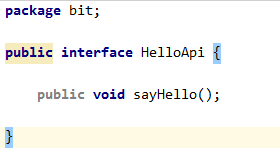
1. 通过阅读pom.xml就可以大概知道整个项目的结构，依赖关系。
2. Maven提供一种创建项目的统一标准格式，src下有main与test文件夹，main下有java（源），resources，webapp文件夹等。统一的标准格式方便程序员之间心领神会，提高团队开发的效率。更重要的是，maven创建项目只需要书写简单的命令行，快捷便利。
3. Maven对jar依赖的管理采用仓库的形式，用户在本地创建一个maven的仓库（同时修改配置文件），这样在pom.xml中只要写入jar依赖的坐标即可从远程仓库中自动下载jar包到本地仓库，项目发布时会将jar包一同发布到target目录下。不需要到处下载jar包并做重复又繁杂的项目jar依赖导入操作。而且，可以将自己写的经常需要用到的jar包发布到自己的本地仓库里，以后要用的时候直接在pom.xml中写相应的坐标就可以导入。

**Spring：**

1. 首先，IOC与AOP独立于Spring存在，只是Spring给出了IOC与AOP（AOP面向切面通过业务逻辑的分离，如：事务管理，权限控制，日志等，实现内聚开发）的具体实现，这里重点总结IOC。
2. 在总结IOC前，有必要说说第一个IOC的例子spring-hello：



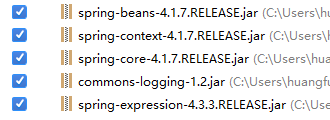
接口HelloApi中定义了一个方法叫做sayHello(); HelloImpl中给出了这个接口的实现：



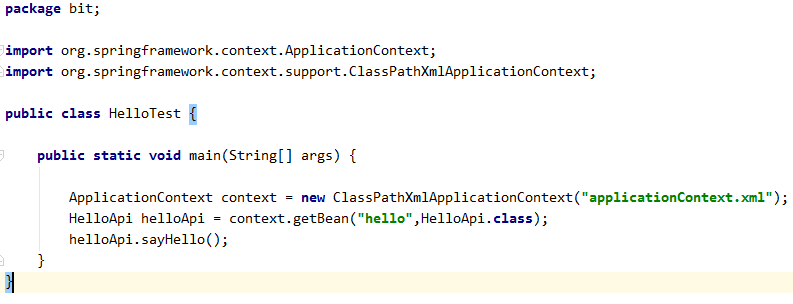
ApplicationContext.xml配置文件：



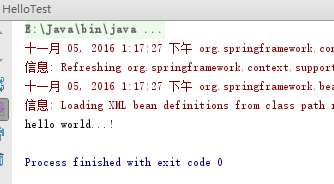
需要导入的jar包：



HelloTest测试：



运行结果：



1. 在测试方法中，并没有像HelloApi helloApi = new HelloImpl()一样new一个对象，而是从容器context中取出一个组件bean直接使用，这个组件的名字叫hello。这样就好像是context这个bean容器帮我们实现了新建对象的过程，新建对象这个控制权不在程序员手上，所以这个过程叫做控制反转，这个context容器叫做SpringIOC容器。
2. 这个简单例子的操作或许看上去有点多余了，首先要通过加载类路径下的配置文件applicationContext.xml初始化得到一个SpringIOC容器context（在配置文件中给出了id为hello的组件bean的类为bit.HelloImpl的配置，容器context将这个实现好的类加入到容器中以待使用，这个过程叫依赖注入，是控制反转的另一种理解方式），然后指定接口名与bean的id得到可以直接使用的相应对象。其实这个操作比直接new一个对象要复杂，但是为什么还这么做呢？
3. 在回答上面这个问题时，要说说面向接口编程。为什么要有面向接口编程？接口其实是一个规范，它是用来统一行为的标准。面向对象的一大特性为多态，通过接口的不同实现（implements）去实现多态。举个例子：一个iAnimals的bark()接口，可以是cat叫，可以是dog叫还可以是bull叫，不同的动物叫声肯定不一样，这就对应了bark()接口应该有不同的实现（即多态）。上面提到了从context容器获取组件hello时只需要指定它的接口，因为它的对应实现已经在配置文件中给出配置，在实际编程逻辑中就不需要new impl()，比普通的接口编程少了new的操作。
4. 少了new impl()有什么好处呢？试想一种情况，在一个类中有3个方法都分别用到了HelloApi，普通的接口编程就是在3个方法中分别HelloApi helloApi = new HelloImpl();若有一天业务逻辑变更要将HelloApi的实现类换成HelloImpl2，这样就要改3个地方，如果这个3是10000呢？就要改10000次。但通过SpringIOC容器，无论是3还是10000，只需要修改配置文件的一处地方，将<bean id=”hello” class=”bit.HelloImpl”>改成<bean id=”hello” class=”bit.HelloImpl2”>，实际编程逻辑中不需要做任何改动。
5. 因此，对上面提到的做一个简单总结就是：SpringIOC将配置信息与实际编程逻辑很大程度上做了解耦，提高了开发与维护的效率。

补充说明：

1. Bean就是由Spring容器初始化、装配和管理的对象。
2. SpringIOC容器的代表：BeanFactory（beans-config.xml），ApplicationContext(applicationContext.xml)。
3. Spring采用依赖注入DI（配置文件的书写，bean关系的维护：设值等）来实现IOC。
4. DI：如果class A需要访问class B的信息，则说类A依赖于类B。Spring的IOC容器以依赖注入的方式实现了Bean对象之间关系的维护，由Bean容器负责组装Bean之间的依赖关系（例如dataSource的rel）。
5. （基于注解模式对Spring的理解）可以把在SpringIOC container自己注入的各个bean看作是平行等级分散的，需要A的时候从容器中装载@Autowired进来不需要new操作直接使用，也不需要关心A它的依赖具体是什么，Spring会基于注解自动扫描自行管理帮助实例化。

Dao层MyBatis与Spring整合的配置文件spring-dao.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"  
 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"**>  
  
 *<!--资源加载 -->* <**context:property-placeholder location="classpath:jdbc.properties"**></**context:property-placeholder**>  
  
 *<!--配置c3p0连接池 -->* <**bean id="dataSource" class="com.mchange.v2.c3p0.ComboPooledDataSource"**>  
  
 *<!--数据源connection-->* <**property name="driverClass" value="${jdbc.driverClass}"**></**property**>  
 <**property name="jdbcUrl" value="${jdbc.url}"**></**property**>  
 <**property name="user" value="${jdbc.username}"**></**property**>  
 <**property name="password" value="${jdbc.password}"**></**property**>  
  
 *<!--连接池属性-->* <**property name="minPoolSize" value="100"**></**property**>  
 <**property name="maxPoolSize" value="200"**></**property**>  
 </**bean**>  
  
 *<!-- 配置sqlSessionFactory-->* <**bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean"**>  
  
 *<!--数据源-->* <**property name="dataSource" ref="dataSource"**></**property**>  
  
 *<!--  
 mybatis配置文件的路径  
 获取自增主键、驼峰命名转换，使用列标签  
 -->* <**property name="configLocation" value="classpath:mybatis-config.xml"**></**property**>  
  
 *<!--  
 配置数据库表名映射entity包下的类时使用别名，否则会报如下错误信息：  
 Error creating bean with name 'sqlSessionFactory' defined in class path resource [spring/spring-dao.xml]  
 Could not resolve type alias 'Contacts'.-->* <**property name="typeAliasesPackage" value="entity"**></**property**>  
  
 *<!--mybatis映射文件路径 -->* <**property name="mapperLocations" value="classpath:mapper/\*.xml"**></**property**>  
 </**bean**>  
  
 *<!--配置需要扫描的dao接口包，动态实现dao接口，注入（DI）到SpringIOC容器中 -->* <**bean class="org.mybatis.spring.mapper.MapperScannerConfigurer"**>  
  
 <**property name="sqlSessionFactoryBeanName" value="sqlSessionFactory"**></**property**>  
 <**property name="basePackage" value="dao"**></**property**>  
 </**bean**>  
  
</**beans**>

Service层spring-service.xml配置文件：

@Component注解提示IOC容器将相应类注入，组织成可以直接使用的bean组件。

*<?***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  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd"**>  
  
 *<!--扫描service包下所有的注解 -->* <**context:component-scan base-package="service"**></**context:component-scan**>  
</**beans**>

Web层（Controller映射请求并响应）spring-web.xml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  
 http://www.springframework.org/schema/context  
 http://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/mvc  
 http://www.springframework.org/schema/mvc/spring-mvc.xsd"**>  
  
 *<!--基于注解模式的Spring MVC-->* <**mvc:annotation-driven**></**mvc:annotation-driven**>  
  
 *<!--webapp下除WEB-INF的静态资源mvc处理程序配置 -->* <**mvc:default-servlet-handler**></**mvc:default-servlet-handler**>  
  
 *<!--配置ViewResolver，用于将controller返回的model与view name的渲染 -->* <**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 <**property name="viewClass" value="org.springframework.web.servlet.view.JstlView"**></**property**>  
 <**property name="prefix" value="/WEB-INF/jsp/"**></**property**>  
 <**property name="suffix" value=".jsp"**></**property**>  
 </**bean**>  
  
 *<!--扫描web包下的注解-->* <**context:component-scan base-package="web"**></**context:component-scan**>  
</**beans**>

WEB-INF下web.xml配置SpringMVC DispatcherServlet实现dao----> service---->controller：

<**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"  
 metadata-complete="true"**>  
  
 <**servlet**>  
   
 <**servlet-name**>telebook-servlet</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
  
 <**init-param**>  
 <**param-name**>contextConfigLocation</**param-name**>  
 <**param-value**>classpath\*:spring/spring-dao.xml,classpath\*:spring/spring-service.xml,classpath\*:spring/spring-web.xml</**param-value**>  
 *<!--spring-web.xml为DispatcherServlet提供controller这个RequestMappingHandlerMapping的bean-->* </**init-param**>  
  
 </**servlet**>  
  
 <**servlet-mapping**>  
 <**servlet-name**>telebook-servlet</**servlet-name**>  
 *<!--dispatcher拦截所有request并做delegate-->* <**url-pattern**>/</**url-pattern**>  
 </**servlet-mapping**>  
  
</**web-app**>