1. Spring
2. 概述

Spring是分层的Java SE/EE应用轻量级开源框架，以IOC（Inverse Of Control）翻转控制和AOP（Aspect Oriented Programming）面向切面编程为内核。

Spring提供了展现层SpringMVC，持久层Spring JDBCTemplate以及业务层事务管理等众多的企业级应用技术，还能整合其他开源的第三方框架和类库。

1. Spring的优势
2. 方便解耦，简化开发

通过Spring提供的IOC容器，可以将对象间的依赖关系交由Spring控制，避免硬编码造成的过度耦合。

1. 声明式事务的支持

通过声明式方式灵活地进行事务管理，提高开发效率和质量。

1. 方便程序的调试

可以用非容器依赖的编程方式进行几乎所有的测试工作。

1. Spring程序原理

假设我们创建了一个名为UserDao的接口，并且通过UserDaoImpl类实现了这个接口，之前要创建一个UserDao的对象，是直接new UserDaoImpl实现类即可，现在使用Spring框架就要分层进行。

Spring框架设置了xml配置文件，通过id标识配置了UserDaoImpl实现类的全类名，那么在Spring框架内要创建一个UserDao的对象，就可以通过读取xml配置文件，根据id标识来获取该实现类的全类名，通过反射创建Bean对象。如：

UserDao userDao = Spring客户端.getBean(id标识)

1. Spring程序开发步骤
2. 通过Maven创建项目，设置项目的project Structure，设置使用的SDK，JDK，输出路径。再为项目添加Web包，记住要修改Web包的路径。
3. 在项目的po.xml配置文件中导入所需要的jar包的坐标。所谓坐标是指Maven项目中jar包仓库的坐标，如下所示：

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.2.1.RELEASE</version>  
</dependency>

如果这里的导入jar包不成功，说明本地仓库中可能没有，需要从中央仓库中下载，我们需要设置Maven的Settings.xml配置文件的镜像源为国内镜像，防止下载失败。

1. 编写接口和实现类

这里我们简单地编写了一个UserDao类和UserDaoImpl实现类。

1. 之前的做法是通过UserDaoImpl实现类创建UserDao类，Spring中是使用配置文件来创建，在src/main/resources目录下创建一个配置文件applicationContext.xml，将UserDaoImpl实现类的全包名设置为键值对的格式，存储在bean标签中，如下所示：

<bean id="userDao" class="com.spring.dao.impl.UserDaoImpl"></bean>

1. 使用Spring的API获取Bean实例

在src/main/java目录中创建一个测试类UserDaoDemo，创建一个Spring客户端app，通过该app获取配置文件中的Bean对象userDao，取出该全类名所对应的UserDaoImpl实现类对象，并调用该对象的方法。如下所示：

public class UserDaoDemo {  
 public static void main(String[] args) {  
 ApplicationContext app = new ClassPathXmlApplicationContext("applicationContext.xml");  
 UserDao userDao = (UserDao)app.getBean("userDao");  
 userDao.save();  
 }  
}

1. 至此，一个完整的Spring框架的Maven Demo搭建完成。
2. Spring配置文件
3. Bean标签
4. Bean标签的基本配置

Bean标签的作用是将配置对象交由Spring来创建，它代表着创建一个类对象，默认调用的是类中的无参构造函数，如果类中没有无参构造函数则不能创建成功。

Bean标签的基本属性是：

id: Bean实例在Spring容器中的唯一标识

class: Bean的全限定名称

1. Bean标签的范围配置

Bean标签的范围配置的作用是限定Bean实例的作用范围，有如下几种范围：

singleton : 单例的，即Spring容器中仅有一个Bean实例，在加载配置文件创建Spring客户端时就已经创建成功了。只要容器存在，那么这个Bean实例就会一直存在，除非容器被销毁。

prototype: 多例的，即Spring容器中可以有多个Bean实例，每个Bean实例在调用方法时才会创建。只要该Bean实例一直在使用中，那么Bean实例就不会消失，当该Bean实例长时间不用时，Java的垃圾回收器就将它回收了。

request : Web项目中，Spring创建一个Bean的对象，将对象存入request域中

session : Web项目中，Spring创建一个Bean的对象，将对象存入到session域中

global session : Web项目中，应用在Portlet环境中，如果没有Portlet环境则globalSession相当于session

范围的设置是通过Bean标签的属性值来设置的，如下所示：

<bean id="userDao" class="com.spring.dao.impl.UserDaoImpl" scope=”singleton”></bean>

1. Bean标签的生命周期配置
2. init-method

init-method属性指定了类中的初始化方法的名称，即在创建这个Bean实例时会自动进行初始化的方法，该初始化方法定义在类中。

1. destroy-method

destroy-method属性指定了类中销毁方法的名称，即在销毁该Bean实例时自动调用的方法，同样，该方法定义在类中。

1. Bean实例化的三种方式
2. 无参构造方法实例化

我们之前使用的Bean标签中定义id和全类名就是使用的无参构造方法进行实例化的。

<bean id="userDao" class="com.spring.dao.impl.UserDaoImpl"></bean>

1. 工厂静态方法实例化

工厂的静态方法其实就是一个类中的静态方法，Bean标签中设置它的id和工厂类的全类名，并且添加属性 factory-method 为该工厂类的静态方法的名称即可。

那么，Bean实例在实例化时使用的就是该静态方法了。

<bean id="userDao" class="com.spring.factory.StaticFactory" factory-method=”getUserDao”></bean>

1. 工厂实例方法实例化

上面的静态方法不需要工厂实例即可直接调用，而这个工厂实例方法需要一个工厂类的实例才能调用，因此首先我们需要创建一个Bean实例为工厂实例，然后再创建一个bean实例为方法实例。如下所示：

<bean id=”factory” class=”com.spring.factory.DynamicFactory”></bean>

<bean id=”userDao” factory-bean=”factory” factory-method=”getUserDao”/>

注意，下面的bean实例的factory-bean属性意思是工厂的id名，factory-method属性意思是工厂方法的名称。

1. Bean的依赖注入分析

假设我们要生成两个Bean实例，一个是service层的UserService实例，一个是dao层的UserDao实例，UserService实例需要调用UserDao实例，因此我们可以直接将UserDao实例设置到UseService内部。

要将UserDao对象设置到UserService内部，可以使用两种方法，一种是set方法，设置一个setUserDao方法；另一种方法是通过UserService的有参构造函数将UserDao实例传进去。

1. Bean的依赖注入概念

依赖注入（Dependency Injection）是Spring框架核心IOC的具体实现。通过IOC，我们将对象的创建交给了Spring，IOC的解耦降低了各个层之间的耦合关系，但是肯定还会有依赖关系存在，如service层调用dao层的方法。这种依赖关系依然要靠Spring来处理。

1. 使用set方法依赖注入

UserServiceImpl类需要调用UserDao类，因此可以通过set方法在UserServiceImpl类中导入一个Userdao类对象。

private UserDao userDao;

public void setUserDao(UserDao userDao){

this.userDao = userDao;

}

并且将applicationContext.xml的配置文件的UserService的bean标签中添加property子标签，name属性为set方法后面的字符串首字母小写，如setUserDao即为userDao，ref属性为注入的Bean对象的id名称。

<bean id=”userDao” class=’com.spring.dao,imp.UserDaoImpl”></bean>

<bean id=”userService” class=”com.spring.service.impl.UserServiceImpl”>

<property name=”userDao” ref=”userDao”></property>

</bean>

整体表示的是UserServiceImpl类中有一个setUserDao方法，要将一个UserDaoImpl类对象注入到UserServiceImpl类中。

要是嫌使用子标签property比较麻烦，可以使用命名空间来注入，在applicationContext.xml最上面的命名空间处添加一个p命名空间：

xmls:p=<http://www.springframework.org/scheme/p>

接着就可以修改bean标签了，去掉property子标签，用p:userDao-ref=”userDao”代替：

<bean id=”userService” class=”com.spring.service.impl.UserServiceImpl” p:userDao-ref=”userDao”>

</bean>

1. 使用有参构造函数进行依赖注入

同样的场景，UserServiceImpl需要调用UserDao的方法，因此构造UserServiceImpl的有参构造函数，传入UserDao的对象参数，如下所示：

private UserDao userDao;

public UserServiceImpl(UserDao userDao){

this,userDao = userDao;

}

同样的，需要在applicationContext.xml中配置bean标签，添加一个constructor-arg子标签，name属性为有参构造中传的对象的名称userDao，ref属性为注入的Bean对象UserDaoImpl的id名userDao。虽说两个都是userDao，可是含义可是完全不一样的。

<bean id=”userDao” class=”com.spring.dao,impl.UserDaoImpl”></bean>

<bean id=”userService” class=”com.spring.service.impl.UserServiceImpl”>

<constructor-arg name=”userDao” ref=”userDao”></constructor-arg>

</bean>

整体表示的是UserServiceImpl类有一个有参构造函数传入的是名称为userDao的bean对象，id名称为userDao，对应的类对象为UserDaoImpl对象，这样UserDaoImpl对象就注入到了UserServiceImpl类中。

1. Bean的依赖注入的数据类型
2. 注入数据的三种数据类型：

普通数据类型

引用Bean对象

集合数据类型

之前讲的都是引用Bean对象，下面讲讲注入普通数据类型和集合数据类型。

（2）注入普通数据类型

要将String类型的username注入到UserDaoImpl类中，首先要在UserDaoImpl类中定义一个username，然后利用set方法或者有参构造方法将username传入。之后在配置文件中添加子标签property，name属性为该数据的名称，value属性为该数据的值。如下所示：

<bean id=”userDao” class=”com.spring.dao.impl.UserDaoImpl”>

<property name=”username” value=”wxx” />

</bean>

1. 注入集合类型

集合有List集合，Map集合，Properties集合等等。

这些集合在类中的定义和注入都差不多，使用set方法或者有参构造传入即可，配置文件中的配置稍有差别。

List集合：在Bean标签内添加子标签property，name属性为List集合的名称，在其中再添加一个子标签list，list标签内再添加一个子标签value，存放list集合中的值。如下所示：

<bean id=”userDao” class=”com.spring.dao.impl.UserDaoImpl”>

<property name=”strList”>

<list>

<value>aa</value>

<value>bb</value>

</list>

</property>

</bean>

Map集合：

Bean标签内添加子标签property，name属性为map集合的名称，在其中添加子标签map，map子标签内再添加子标签entry, map集合中的键和值可以使用字符串或者对象，因此key属性，value属性，key-ref属性，value-ref属性。key属性为map的键值对的键名，key-ref为键引用的Bean对象的id，value为键名，value-ref为值引用的Bean对象的id。如下所示：

<bean id=”userDao” class=”com.spring.dao.impl.UserDaoImpl”>

<property name=”userMap”>

<map>

<entry key=”user1” value-ref=”user2”></entry>

</map>

</property>

</bean>

Property集合：

Bean标签内添加子标签property，name属性为property集合的名称，在其中添加子标签props，props标签内再添加子标签prop，每个prop就代表着property文件中的每个键值对。如下所示：

<bean id=”userDao” class=”com.spring.dao.impl.UserDaoImpl”>

<property name=”properties”>

<props>

<prop key=”p1”>aaa</prop>

<prop key=”p2”>bbb</prop>

</props>

</property>

</bean>

1. 分模块分解配置文件

可以看到，我们需要在配置文件中配置很多信息，这样单个配置文件的体积就非常大，也不利于查询，现在可以将单个配置文件按照项目的需求分为多个模块，如control层的，service层的，dao层的。

那么这么多的xml配置文件，如何将他们关联起来呢？通过import标签进行导入配置文件，如在Spring的主配置文件中通过import标签导入其他的配置文件：

<import resource=”applicationContext-xxx.xml” />

1. Spring相关API
2. ApplicationContext
3. 概述

ApplicationContext是一个接口类型，代表应用上下文，可以通过该接口的实例加载配置文件，获取Spring容器中的Bean对象。

1. ApplicationContext的实现类

我们通过调用AppliactionContext接口的实现类来获取Spring容器中的Bean对象。有三种实现类，如下所示：

1. ClassPathXmlApplicationContext

这个实现类是从类的根路径下加载配置文件，直接写配置文件的名称即可，因为该实现类是从该类的加载路径下加载配置文件的，配置文件一般都在类的路径下。我们一般使用的都是这种，因为这种不是硬编码，即使配置文件路径改变也不会出错。

ApplicationContext app = new ClassPathXmlAppliactionContext(“applicationContext.xml”);

1. FileSystemXmlApplicationContext

该实现类是从磁盘路径上加载配置文件，配置文件可以放在磁盘的任意位置，即利用绝对路径来加载配置文件。这种方式比较死板，采用硬编码的方式加载，灵活性不好。

ApplicationContext app = new FileSystemXmlApplicationContext(“D:\\Java\\Spring\\test\\src\\main\\resources\\applicationContext.xml”);

1. AnnotationConfigApplicationContext

在使用注解配置容器对象时，需要使用此类来创建Spring容器，他用来读取注解。

1. getBean()方法的使用

getBean()方法有两种签名，一个传入Bean对象的id，一个传入Class字节码。

1. 传入Bean对象的id

public Object getBean(String name) throws BeansException{

assertBeanFactoryActive();

return getBeanFactory().getBean(name);

}

这种方式获取Bean对象可以通过不同的Bean对象id获取多个同类对象，因为每个Bean对象的id 是唯一的，Sping容器中可以有一个类的多个对象。因此，通常用的都是这种方式。

1. 传入类的Class字节码

public<T> T getBean(Class<T> requiredType) throws BeansException{

assertBeanFactoryActive();

return getBeanFactory().getBean(requiredType);

}

通过传入类的字节码就限制了获取多个Bean对象，只能获取类的一个Bean对象。因为即使Spring容器中有一个类的多个对象，通过类的字节码是无法区分这些对象的。