Spring基础

Spring3.1.0版本示例：

Spring核心技术：

**两大核心技术：ioC（控制反转） DI（依赖注入）**

**控制反转：本质是控制权本由应用程序转移到外部容器（spring）中**，控制权的转移就是控制反转（在spring中，由配置可知，当一个bean类中，用<property name=*"userDao"* ref=*"oracle"/*>运行程序就可以知道，将id=oracle中类）

DI依赖注入：

第一个Spring 示例：

1. 导包： 1.spring/dist/\*

2.依赖包: reqlib/Comnces-login.jar

2. 配置：spring-framework-3.1.0.RC1\docs\spring-framework-reference\pdf\打开文档

找到pdf中配置文档复制，再src中建一个applicationContext.xml 是spring默认的配置名称

配置代码：

<?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:tx="http://www.springframework.org/schema/tx"

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

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/tx

http://www.springframework.org/schema/tx/spring-tx.xsd

http://www.springframework.org/schema/aop

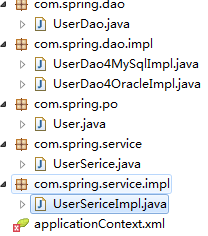
http://www.springframework.org/schema/aop/spring-aop.xsd

">

**<bean类 id=”该类名称” class=”全名” >**

</beans>

3. 再建如下：



4．在Service实现类中写上：

**public** **class** UserSericeImpl **implements** UserSerice {

**private** UserDao userDao;

**public** **void** setUserDao(UserDao userDao) {

**this**.userDao = userDao;

}

**public** **void** saveUser(User user) {

userDao.addUser(user);

}

}

接口之间要继承，并且接口可以继承多个类或接口

一个类是实现一个接口

5.测试类：

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

UserSerice userSerice = **ac.getBean("userService", UserSerice.class);**

**//** UserSerice userSerice = (UserSerice) ac.getBean("userService");

userSerice.saveUser(**null**);

其中ApplicationContext继承与BeanFactory，创建bean类出来实现方法

由测试类可知，其中需要spring控制Bean，来产生类，而不需要用自行创建类，这也就是控制反转，原来创建对象是我来创建，用new出来，现在交个spring控制，来创建对象

<bean id=*"mysql"* class=*"com.spring.dao.impl.UserDao4MySqlImpl"*/>

<bean id=*"oracle"* class=*"com.spring.dao.impl.UserDao4OracleImpl"*/>

<bean id=*"userService"* class=*"com.spring.service.impl.UserSericeImpl"*>

Id 与 name 不同之处

Id 唯一，不能用特殊字符

name可以相同且可多个名称，一般用id

<property name=*"userDao"* ref=*"oracle"*></property>

<property

</bean>

**项目一启动，spring会自动创建beans中所有的类，所以需要一个set方法来调用dao中的方法，所以就知道为什么IOC控制反转和为什么service中为啥有dao中set而不是get**

**Set和构造方法：尽量用set来注入userDao**

**public** UserSericeImpl(UserDao userDao){

**this**.userDao = userDao;

}

配置文件：

<bean id=*"mysql"* class=*"com.spring.dao.impl.UserDao4MySqlImpl"*/>

<bean id=*"oracle"* class=*"com.spring.dao.impl.UserDao4OracleImpl"*/>

<bean id=*"userService"* class=*"com.spring.service.impl.UserSericeImpl"* >

<constructor-arg ref=*"mysql"*>

</constructor-arg> </bean>

**Di依赖注入**：将依赖关系的自动注入，示例就可以知道ref中依赖注入进去

**容器：管理某种对象的生命周期，就称为容器**

**Tomcat就是管理servlet的生命周期，所以也叫容器**

可以看来spring也是一个容器，因为她具有ioc控制bean标签中类的生命周期

Byname

根据名称来区分

byType

就是根据类型注入，所以由userDao转入包，中找到依赖关系

**给实体类成员变量赋值：**

**示例user类：**

<bean id=*"user"* class=*"com.spring.po.User"*>

<property name=*"id"* value=*"1"*></property>

<property name=*"username"* value=*"tom"*></property>

<property name=*"age"*>

<array>

<value>99</value>

<value>100</value>

</array>

</property>

<property name=*"tid"*>

<list>

<value>1012</value>

<value>1023</value>

</list>

</property>

<property name=*"map"*>

<map>

<entry key=*"u1"* value=*"1"*></entry>

<entry key=*"u2"* value=*"2"*></entry>

<entry key=*"u3"* value=*"3"*></entry>

</map>

</property>

</bean>

演示：

User user = ac.getBean("user", User.**class**); System.*out*.println(user.getMap().get("u1"));

问题一：

不知道怎么用Date数据类型的赋值

**判断spring创建的对象是单实例还是多实例的：**

UserSerice userSerice1 = (UserSerice) ac.getBean("userService");

UserSerice userSerice2 = ac.getBean("userService", UserSerice.**class**);

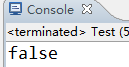
System.*out*.println(userSerice1 == userSerice2);

对于对象比较相等，==号是比较的是地址，equals是值

**说明用spring默认创建对象是单实例的**

**如何创建多实例：**

<bean id=*"userService"* class=*"相应比较类中加上scope=“prototype” "* scope=*"prototy"*>



注解版：

注解版配置：

<?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-3.0.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

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

**<context:annotation-config/>**

</beans>

实现service类

@Service("userService")

**public** **class** UserSericeImpl **implements** UserSerice {

**private** UserDao userDao;

@Resource(name="userDao")

**public** **void** setUserDao(UserDao userDao) {

**this**.userDao = userDao;

}

实现userDao类：

@Component("userDao")

**public** **class** UserDao4MySqlImpl **implements** UserDao

Spring中的错误：

The processing instruction target matching "[xX][mM][lL]" is not allowed.

原因：很简单xml的的第一行在第二行去了。空行了。所以出现了匹配错误。这么简单的错误，只是不知情啊。哈哈。

AOP面向切面编程

1. 横切性关注点：在横切面的角落中

需要加什么（例如： 日志输出 和 权限检查）

1. 切面：对横切面关注点模块的类（动态代理类）
2. Advice：对横切性关注点的实现（方法）分为Before Advice Afert

对于方法是在调用前就实现了，还是调用后实现，例如：

日志是调用后 权限是调用前检查

1. Pointcut 应用范围的指定
2. 织入：把Advice应用到目标对象的过程
3. JoinPoint：切面应用到哪个方法上，这个是JoinPoint
4. 导包：

\spring\aop相关jar\\*

1. 配置：

<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"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.0.xsd*

*http://www.springframework.org/schema/aop*

*http://www.springframework.org/schema/aop/spring-aop-3.0.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.0.xsd"*>

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

<context:annotation-config/>

<aop:aspectj-autoproxy></aop:aspectj-autoproxy>

</beans>

1. 实现方法：

@Aspect //声明此类为一个切面

@Component//必须存在，spring才能创建对象

**public** **class** MyAopUtil{

@Before("execution(\* com.spring.service.impl.\*.save\*(..))")//声明前置通知

**public** **void** doBefore(){

System.*out*.println("前置通知");//可以得到参数值，此时args(name)限制条件限定了"execution(\* com.aoptest.service..\*.\*(..))"中只拦截参数个数只有1个且类型为String的方法，而不是所有方法（该属性在当前拦截中适用，即前置通知适用）

}

}

1. 测试：

**因为是横切面性，当需要时候调用，不需要的不要，所以前提是不需要更改的他的源代码，只需要实现方法（实现于动态代理）**

**由于动态代理目标类必须实现接口，当我们不需要实现接口的时候，就不能用动态代理模式：**

1. **代理模式的优点：**
2. **不直接访问目标类**
3. **扩展性**

**从上看来，我们不使用接口模式就前提是不能直接访问对象，所以只通过扩展性来看，就可以联想到继承**

1. **继承模式（AOP） 不想用接口就用这个**
2. **导包:依赖包 cglib-nodep-2.1\_3.jar**
3. **其余就需要改下配置**

proxy-target-class=*"true"其中没有接口*

