[Bean的作用域及生命周期](http://blog.csdn.net/feihong247/article/details/7798474)

标签： [bean](http://www.csdn.net/tag/bean)[prototype](http://www.csdn.net/tag/prototype)[spring](http://www.csdn.net/tag/spring)[database](http://www.csdn.net/tag/database)[xml](http://www.csdn.net/tag/xml)[测试](http://www.csdn.net/tag/%e6%b5%8b%e8%af%95)

2012-07-29 10:04 6243人阅读 [评论](http://blog.csdn.net/feihong247/article/details/7798474#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/feihong247/article/details/7798474#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

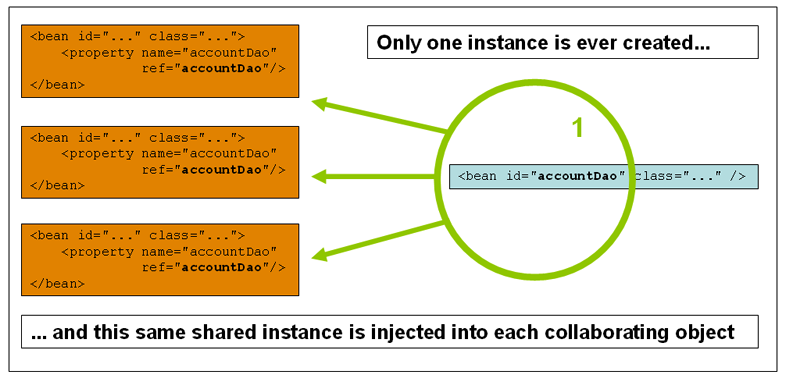
spring（17） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

一、Spring Framework支持五种作用域（其中有三种只能用在基于web的Spring ApplicationContext）。   
内置支持的作用域分列如下：

作用域 描述

|  |
| --- |
|  |
| singleton | 在每个Spring IoC容器中一个bean定义对应一个对象实例。 |
| prototype | 一个bean定义对应多个对象实例。 |
| request | 在一次HTTP请求中，一个bean定义对应一个实例；即每次HTTP请求将会有各自的bean实例， 它们依据某个bean定义创建而成。该作用域仅在基于web的Spring ApplicationContext 情形下有效。 |
| session | 在一个HTTP Session 中，一个bean定义对应一个实例。该作用域仅在基于web的SpringApplicationContext 情形下有效。 |
| global session | 在一个全局的HTTP Session 中，一个bean定义对应一个实例。典型情况下，仅在使用portlet context的时候有效。该作用域仅在基于web的Spring ApplicationContext 情形下有效。 |

**1.Singleton作用域**

当一个bean的作用域为singleton, 那么Spring IoC容器中只会存在一个共享的bean实例，并且所有对bean的请求，只要id与该bean定义相匹配，则只会返回bean的同一实例。换言之，当把一个bean定义设置为singlton作用域时，Spring IoC容器只会创建该bean定义的唯一实例。这个单一实例会被存储到单例缓存（singleton cache）中，并且所有针对该bean的后续请求和引用都将返回被缓存的对象实例。  
  
 请注意Spring的singleton bean概念与GoF模式一书中定义的Singleton模式是完全不同的。经典的GoF Singleton模式中所谓的对象范围是指在每一个ClassLoader 中 指定class创建的实例有且仅有一个 。把Spring的singleton作用域描述成一个container (容器)对应一个bean 实例最为贴切。也就是说假如在单个Spring容器内定义了某个指定class的bean，那么Spring容器将会创建一个并且仅有一个 由该bean定义指定的类实例。Singleton作用域是Spring中的缺省作用域 。要在XML中将bean定义成singleton，可以这样配置：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="accountService" class="com.foo.DefaultAccountService"**/>**
3. <!-- the following is equivalent, though redundant (singleton scope is the default); using spring-beans-2.0.dtd  or upper-->
4. **<bean** id="accountService" class="com.foo.DefaultAccountService" scope="singleton"**/>**
6. <!-- the following is equivalent and preserved for backward compatibility in spring-beans.dtd -->
7. **<bean** id="accountService" class="com.foo.DefaultAccountService" singleton="true"**/>**

 测试：

beans.xml配置文件中配置如下所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="examples.test.PersonServiceBean" **></bean>**

 测试代码如下所示：

**Java代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

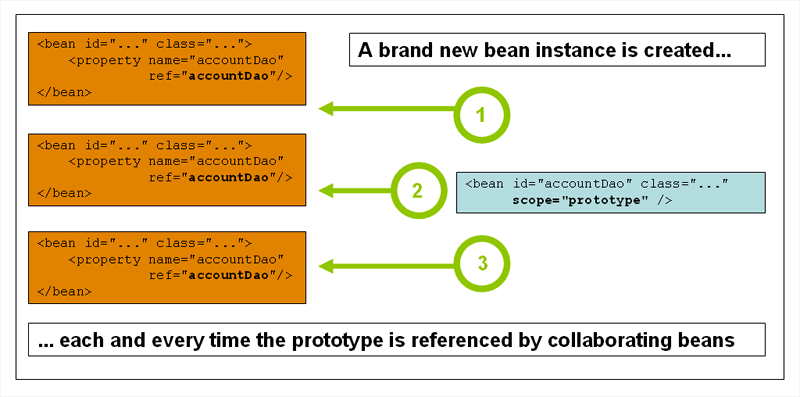
1. **public** **class** SpringTest {
3. @Test
4. **public** **void** instanceSpring(){
5. ApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");
6. PersonService personService1 = (PersonService)ctx.getBean("personService");
7. PersonService personService2 = (PersonService)ctx.getBean("personService");
8. System.out.println(personService1==personService2);
9. }
10. }

 运行测试程序，结果输出为: **true** .说明是只创建了一个PersonServiceBean.

**2. Prototype作用域**

Prototype作用域的bean会导致在每次对该bean请求（将其注入到另一个bean中，或者以程序的方式调用容器的getBean() 方法）时都会*创建一个新的bean实例* 。根据经验，对有状态的bean应该使用prototype作用域，而对无状态的bean则应该使用singleton作用域。

下图演示了Spring的prototype作用域。*请注意，通常情况下，DAO不会被配置成prototype，因为DAO通常不会持有任何会话状态，因此应该使用singleton作用域。*

  
 在XML中将bean定义成prototype，可以这样配置：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. <!-- using spring-beans-2.0.dtd or upper -->
2. **<bean** id="accountService" class="com.foo.DefaultAccountService" scope="prototype"**/>**
3. <!-- the following is equivalent and preserved for backward compatibility in spring-beans.dtd -->
4. **<bean** id="accountService" class="com.foo.DefaultAccountService" singleton="false"**/>**

 将beans.xml配置文件中的配置信息改为如下所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="cn.itcast.service.impl.PersonServiceBean" scope="prototype"**></bean>**

 再运行上面的测试程序，输出结果为: **false** ，说明创建了两个PersonServiceBean.

其他作用域，即request、session以及global session 仅在基于web的应用中使用（不必关心你所采用的是什么web应用框架）。这些作用域仅仅在使用基于web的Spring ApplicationContext实现（如XmlWebApplicationContext）时有用。 如果在普通的Spring IoC容器中，比如像XmlBeanFactory或ClassPathXmlApplicationContext， 尝试使用这些作用域，你将会得到一个IllegalStateException异常（未知的bean作用域）。

二、Bean的生命周期

(1)什么时候初始化bean实例

当scope＝singleton，即默认情况，会在容器初始化时实例化。但我们可以指定Bean节点的lazy-init=”true”来延迟初始化bean,这时候，只有第一次获取bean才会初始化bean，即第一次请求该bean时才初始化。如下配置所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id=”xxx” class=”examples.test.OrderServiceBean” lazy-init=”true” **/>**

 如果想对所有bean都应用延迟初始化，可以在根节点beans设置default-lazy-init=”true”,如下所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<beans** default-lazy-init=”true” …**>**

当scope=prototype时，也会延迟初始化bean,即第一次请求该bean时才初始化(如调用getBean()方法时).

测试如下：

PersonServiceBean代码如下所示：

**Java代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **public** **class** PersonServiceBean **implements** PersonService {
2. **public** **void** init(){
3. System.out.println("Get the database connection and initialize other beans!");
4. }
5. **public** PersonServiceBean(){
6. System.out.println("PersonServiceBean is initialized!");
7. }
8. **public** **void** destroy(){
9. System.out.println("close the database connection!");
10. }
11. }

 beans.xml配置文件如下所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="examples.test.PersonServiceBean" **/>** <!-- 1 -->
2. **<bean** id="personService" class="examples.test.PersonServiceBean" scope="singleton" **/>** <!-- 2 -->

测试程序如下所示：

**Java代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **public** **class** SpringTest {
2. @Test
3. **public** **void** instanceSpring(){
4. ApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");
5. System.out.println("------------------");
6. PersonService personService = (PersonServiceBean)ctx.getBean("personService");
7. }
8. }

 分1，2两种情况运行程序，控制台打印信息如下所示：

PersonServiceBean is initialized!  
------------------

将配置文件修改成如下所示：

**Java代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. <bean id="personService" **class**="examples.test.PersonServiceBean"  lazy-init="true" /> <!-- 1 -->
2. <bean id="personService" **class**="examples.test.PersonServiceBean" scope="prototype" /> <!-- 2 -->

  分1，2两种情况运行程序，控制台打印信息如下所示：

------------------  
PersonServiceBean is initialized!

  以上测试说明：

Bean默认是在容器初始化时初始化的，即

ApplicationContext ctx = new ClassPathXmlApplicationContext("beans.xml");语句执行时就实例化bean了。如果把scope设成scope=”prototype” 或设置lazy-init=”true”，则会延迟bean的实例化，bean会在PersonService personService = (PersonServiceBean)ctx.getBean("personService");语句执行时才实例化。

当配置文件为如下所示时，测试结果同上。

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<beans** xmlns="http://www.springframework.org/schema/beans"
2. xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3. xmlns:context= "http://www.springframework.org/schema/context"
4. xsi:schemaLocation="http://www.springframework.org/schema/beans
5. http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
6. http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd" default-lazy-init="true"
7. **>**
8. **<bean** id="personService" class="cn.itcast.service.impl.PersonServiceBean" **>**
9. **</beans>**

  （2）生命周期：

构造器、init方法、获取bean后的操作、destroy方法（ctx.close时执行）.

注意:如果bean的scope设为prototype时，当ctx.close时，destroy方法不会被调用.

原因：对于prototype作用域的bean，有一点非常重要，那就是Spring不能对一个prototype bean的整个生命周期负责：容器在初始化、配置、装饰或者是装配完一个prototype实例后，将它交给客户端，随后就对该prototype实例不闻不问了。不管何种作用域，容器都会调用所有对象的初始化生命周期回调方法。但对prototype而言，任何配置好的析构生命周期回调方法都将*不会* 被调用。清除prototype作用域的对象并释放任何prototype bean所持有的昂贵资源，都是客户端代码的职责。（让Spring容器释放被prototype作用域bean占用资源的一种可行方式是，通过使用bean的后置处理器，该处理器持有要被清除的bean的引用。）谈及prototype作用域的bean时，在某些方面你可以将Spring容器的角色看作是Java new 操作的替代者。任何迟于该时间点的生命周期事宜都得交由客户端来处理。

配置文件信息如下：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="cn.itcast.service.impl.PersonServiceBean" init-method="init" destroy-method="destroy"**/>**

  测试程序如下：

**Java代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **public** **class** SpringTest {
2. @Test
3. **public** **void** instanceSpring(){
4. AbstractApplicationContext ctx = **new** ClassPathXmlApplicationContext("beans.xml");
5. System.out.println("------------------");
6. PersonService personService = (PersonServiceBean)ctx.getBean("personService");
7. ctx.close();
8. }

运行测试程序，控制台打印信息如下所示：

PersonServiceBean is initialized!  
Get the database connection and initialize other beans!  
------------------  
close the database connection!

修改配置文件信息如下：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="cn.itcast.service.impl.PersonServiceBean" scope="prototype" init-method="init" destroy-method="destroy"**/>**

再次运行测试程序，控制台打印信息如下所示：

------------------  
PersonServiceBean is initialized!  
Get the database connection and initialize other beans!

修改配置文件信息如下所示：

**Xml代码  [收藏代码](http://ryxxlong.iteye.com/blog/657348)**

1. **<bean** id="personService" class="cn.itcast.service.impl.PersonServiceBean" lazy-init="true" init-method="init" destroy-method="destroy"**/>**

此时再次运行测试程序，控制台打印信息如下所示：

------------------  
PersonServiceBean is initialized!  
Get the database connection and initialize other beans!  
close the database connection!