BEAN SCOPES

SpringTutors January 10, 2016 Category 100 Views

Bean Scopes

Bean scope is a concept which is provided by spring to help the programmer in deciding the number of objects they want to create for that bean. To specify the scope of a bean we have to use scope attribute at bean tag level with their modes.

In spring we can declare a bean with five different scopes as follows-

1. singleton– By default every bean declared in the configuration file is singleton. This indicates when you try to refer the bean through injection or factory.getBean() the same bean instance will be returned from the core container.
2. prototype- When we declare a bean scope as prototype this indicates every reference to the bean will return a unique instance of it.
3. request- When we declare a bean scope as request, for every Http Request new bean instance will be injected.
4. session- For every new Http Session, new bean instance will be injected.
5. Global Session- This globalsession csope has been removed from spring3.0. This is used in spring MVC portlet framework where if we want to inject bean for a portal session we need to use this scope.

By the above it is clear that we can use request and session in case of web application. So we will postpone the discussion on these till Spring MVC. Let's understand how to use singleton and prototype.

Example

```
<bean id="a" class="A"/>
```

As we have not specified the scope of the bean then by default its scope is singleton. Now when we request the IOC container to create the object of bean first it will check it scopes as it is singleton so, it checks weather the object for that bean is already created or not in IOC contaainer, as we have requested first time for that bean's object so it will not be there, then IOC container will create the object and store it and return to us.

```
BeanFactory factory=new XmlBeanFactory(new ClassPathResource("application-context.xml"));
A a=factory.getBean("a",A.class);
A a1=factory.getBean("a",A.class);
```

Here we are trying to create two objects of bean but actually a and a1 both are same. when we request second time for creating the object of bean “a” IOC container will check the scope of bean definition as it is singleton first it checks weather the object for that bean is available or not in IOC container, as it is available then IOC container will return the same object to us.

a=a1 (true)

We can specified the singleton scope as below also-

```
<bean id="a" class="A" scope="singleton"/>
```

Now when we configure the scope of bean as prototype as follows-

```
<bean id="a" class="A" scope="prototype"/>
```

In this case when we request for the bean object the IOC container will check the scope of the bean as it is prototype it will create the object of that bean and return to us. In this case IOC container will not look for the availability of the object. It means whenever we are requesting for the bean object it will check the scope, if scope is prototype then it will create a new object and return to us.

```
BeanFactory factory=new XmlBeanFactory(new ClassPathResource("application-context.xml"));
A a=factory.getBean("a",A.class);
A a1=factory.getBean("a",A.class);
```

In this case a!=a1

That is a and a1 are two separate object created by IOC container.

a=a1(false)

Now we can say that in case of singleton an IOC container will create only one object per bean definition and in case of prototype IOC container will create as many objects as it is requested to create.

Let’s see an application on this-

```
package com.bs.bean;
public class Toy {
private int id;
private String manufacturer;
public void setId(int id) {
 this.id = id;
}
public void setManufacturer(String manufacturer) {
 this.manufacturer = manufacturer;
}
public String toString() {
 return "Toy [id=" + id + ", manufacturer=" + manufacturer + "]";
}
}
```

Spring Bean Configuration File-

application-context.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"
 xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
<bean id="toy" class="com.bs.bean.Toy" scope="singleton">
<property name="id" value="101"/>
<property name="manufacturer" value="Jhon"/>
</bean>
<bean id="toy1" class="com.bs.bean.Toy" scope="prototype">
<property name="id" value="1012"/>
<property name="manufacturer" value="Jhon"/>
</bean>
</beans>
```

Test the above application

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.core.io.ClassPathResource;
package com.bs.bean;
public class BSTest {
 public static void main(String[] args) {
 BeanFactory factory=new XmlBeanFactory(new ClassPathResource("com/bs/common/application-context.xml"));
 BeanFactory factory1=new XmlBeanFactory(new ClassPathResource("com/bs/common/application-context.xml"));
 Toy t1=(Toy)factory.getBean("toy");
 Toy t2=(Toy)factory.getBean("toy");
 Toy t3=(Toy)factory.getBean("toy1");
 Toy t4=(Toy)factory1.getBean("toy1");
 System.out.println(t1);
 System.out.println(t2);
 System.out.println(t3);
 if(t3==t4)
 {
 System.out.println("True");
 }
 else
 {
 System.out.println("false");
 }
 }
}
```

O/P

```
Toy [id=101, manufacturer=Jhon]
Toy [id=101, manufacturer=Jhon]
Toy [id=1012, manufacturer=Jhon]
false
```

In the test class t1=t2(singleton) but t3!=t4(prototype).