Bean scopes

Summary

Description

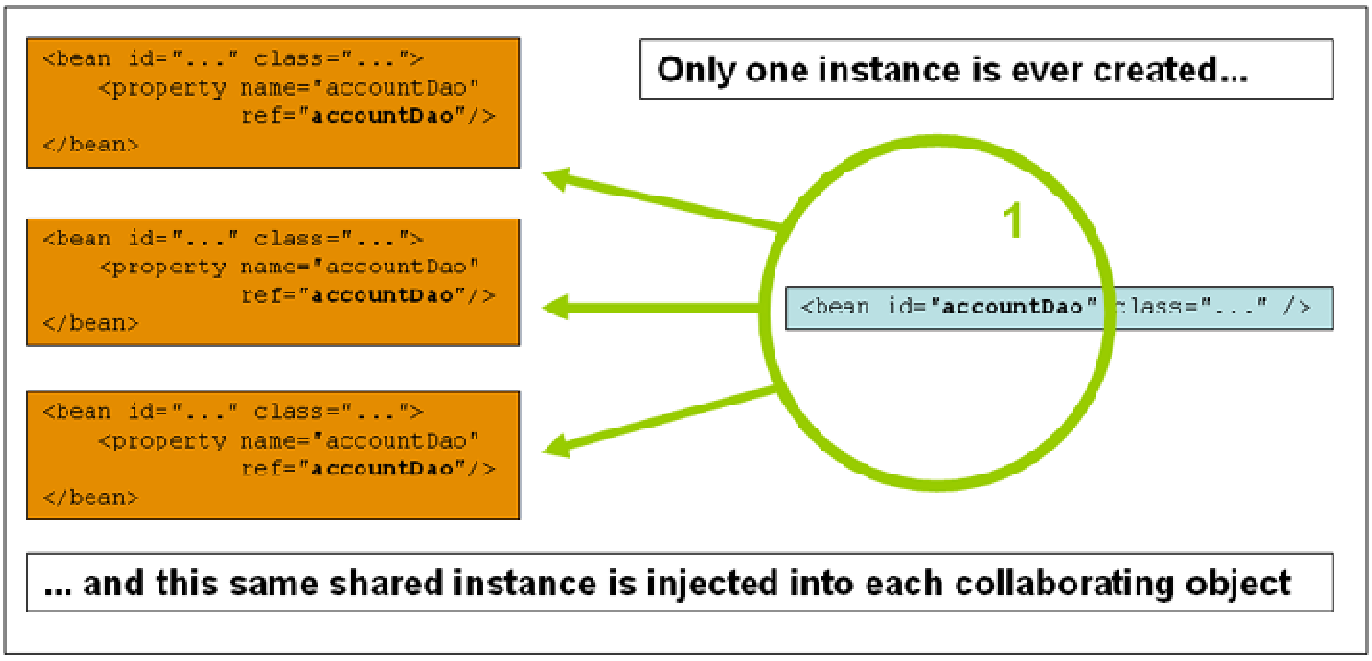
Definition of bean is to define the method of creating the actual bean instance. As the class, there can be many instances under the bean definition.

You can control not only the various dependencies and configuration values that are to be plugged into an object that is created from a particular bean definition, but also the *scope* of the objects created from a particular bean definition. The Spring Framework supports exactly five scopes (of which three are available only if you are using a web-aware ApplicationContext).

Scope	Description
singleton	Scopes a single bean definition to a single object instance per Spring IoC container.
prototype	Scopes a single bean definition to any number of object instances.
request	Scopes a single bean definition to the lifecycle of a single HTTP request; that is each and every HTTP request will have its own instance of a bean created off the back of a single bean definition. Only valid in the context of a web-aware Spring ApplicationContext.
session	Scopes a single bean definition to the lifecycle of a HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext.
global session	Scopes a single bean definition to the lifecycle of a global HTTP Session. Typically only valid when used in a portlet context. Only valid in the context of a web-aware Spring ApplicationContext.

The Singleton Scope

When the bean is singletone, only one shared instance is managed.



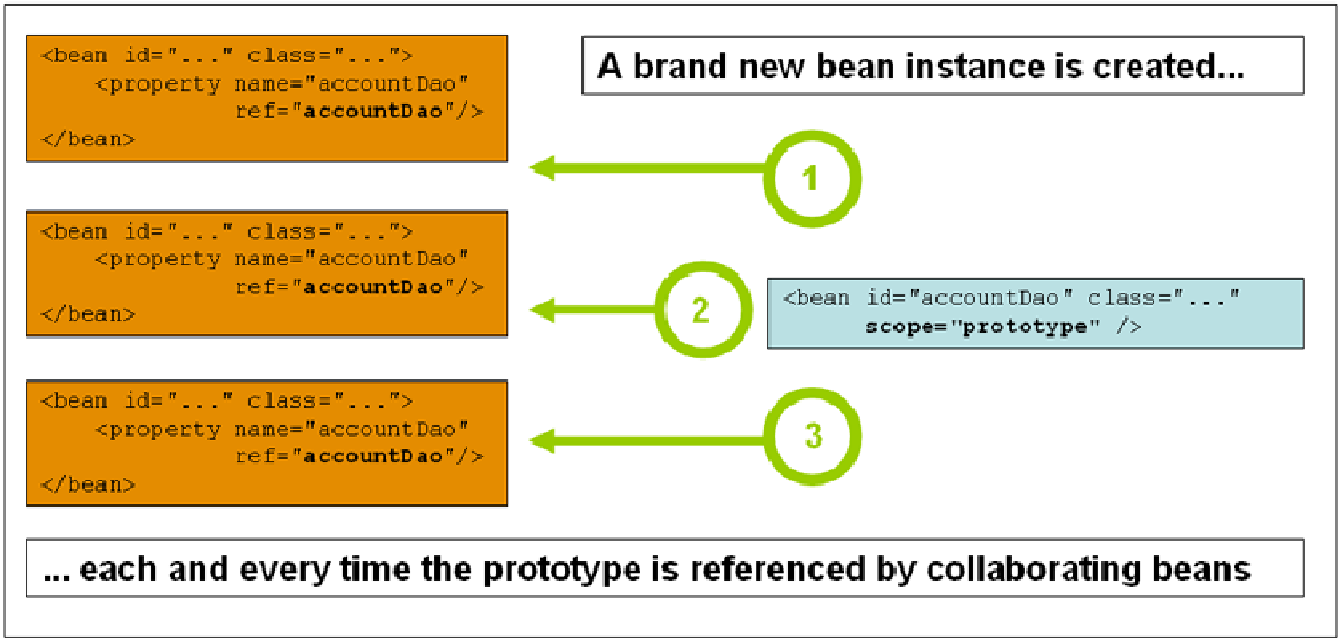
The singleton scope is the default scope in Spring.
<bean id="accountService" class="com.foo.DefaultAccountService"/>

<!-- the following is equivalent, though redundant (singleton scope is the default); using spring-beans-2.0.dtd -->
<bean id="accountService" class="com.foo.DefaultAccountService" scope="singleton"/>

<!-- the following is equivalent and preserved for backward compatibility in spring-beans.dtd -->
<bean id="accountService" class="com.foo.DefaultAccountService" singleton="true"/>

The prototype scope

The bean defined as a form of prototype scope creates new beans every instant.



<!-- using spring-beans-2.0.dtd -->
<bean id="accountService" class="com.foo.DefaultAccountService" scope="prototype"/>

<!-- the following is equivalent and preserved for backward compatibility in spring-beans.dtd -->
<bean id="accountService" class="com.foo.DefaultAccountService" singleton="false"/>

There is one quite important thing to be aware of when deploying a bean in the prototype scope, in that the lifecycle of the bean changes slightly. Spring does not manage the complete lifecycle of a prototype bean: the container instantiates, configures, decorates and otherwise assembles a prototype object, hands it to the client and then has no further knowledge of that prototype instance. This means that while *initialization* lifecycle callback methods will be called on all objects regardless of scope, in the case of prototypes, any configured *destruction* lifecycle callbacks will *not* be called. It is the responsibility of the client code to clean up prototype scoped objects and release any expensive resources that the prototype bean(s) are holding onto.

Singleton beans with prototype-bean dependency

This subject is dealt with in Method Injection.

The other scopes

The request, session and global session scope is only available in web-based applications.

Initial web configuration

To use the request, session and global session scope, it requires additional configurations. The additional configurations vary according to the environment.

If you are accessing scoped beans within Spring Web MVC, i.e. within a request that is processed by the Spring DispatcherServlet, or DispatcherPortlet, then no special setup is necessary: DispatcherServlet and DispatcherPortlet already expose all relevant state.

When using a Servlet 2.4+ web container, with requests processed outside of Spring's DispatcherServlet (e.g. when using JSF or Struts), you need to add the following javax.servlet.ServletRequestListener to the declarations in your web application's 'web.xml' file.

```
<web-app>
  ...
  <listener>
    <listener-class>org.springframework.web.context.request.RequestContextListener</listener-
class>
  </listener>
  ...
</web-app>
```

If you are using an older web container (Servlet 2.3), you will need to use the provided javax.servlet.Filter implementation. The filter mapping depends on the surrounding web application configuration and so you will have to change it as appropriate

```
<web-app>
  ..
  <filter>
    <filter-name>requestContextFilter</filter-name>
    <filter-class>org.springframework.web.filter.RequestContextFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>requestContextFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  ...
</web-app>
```

The request scope

```
<bean id="loginAction" class="com.foo.LoginAction" scope="request"/>
```

With the above bean definition in place, the Spring container will create a brand new instance of the LoginAction bean using the 'loginAction' bean definition for each and every HTTP request. That is, the 'loginAction' bean will be effectively scoped at the HTTP request level. When the request is finished processing, the bean that is scoped to the request will be discarded.

The session scope

```
<bean id="userPreferences" class="com.foo.UserPreferences" scope="session"/>
```

With the above bean definition in place, the Spring container will create a brand new instance of the UserPreferences bean using the 'userPreferences' bean definition for the lifetime of a single HTTP Session. In other words, the 'userPreferences' bean will be effectively scoped at the HTTP Session level. When the HTTP Session is eventually discarded, the bean that is scoped to that particular HTTP Session will also be discarded.

The global session scope

```
<bean id="userPreferences" class="com.foo.UserPreferences" scope="globalSession"/>
```

The global session scope is similar to the standard HTTP Session scope, and really only makes sense in the context of portlet-based web applications. The portlet specification defines the notion of a global Session that is shared amongst all of the various portlets that make up a single portlet web application.

Beans defined at the global session scope are scoped (or bound) to the lifetime of the global portlet Session.

Scoped beans as dependencies

Being able to define a bean scoped to an HTTP request or Session is all very well, but one of the main value-adds of the Spring IoC container is that it manages not only the instantiation of your objects (beans), but also the wiring up of collaborators (or dependencies). If you want to inject a (for example) HTTP request scoped bean into another bean, you will need to inject an AOP proxy in place of the scoped bean.

```
<?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:aop="http://www.springframework.org/schema/aop"
        xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd
http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd">

    <!-- a HTTP Session-scoped bean exposed as a proxy -->
    <bean id="userPreferences" class="com.foo.UserPreferences" scope="session">

        <!-- this next element effects the proxying of the surrounding bean -->
        <aop:scoped-proxy/>
    </bean>

    <!-- a singleton-scoped bean injected with a proxy to the above bean -->
    <bean id="userService" class="com.foo.SimpleUserService">

        <!-- a reference to the proxied 'userPreferences' bean -->
        <property name="userPreferences" ref="userPreferences"/>

    </bean>
</beans>
```

To create such a proxy, you need only to insert a child <aop:scoped-proxy/> element into a scoped bean definition (you may also need the CGLIB library on your classpath so that the container can effect class-based proxying).

Reference

- Spring Framework - Reference Document / 3.4. Bean scopes