**Spring Bean Scope**

Before we start with Spring Bean Scope lets discuss how spring creates object.

As stated earlier in my previous post Spring Bean Factory We have Spring Container responsible to create objects.

Object

Object

Object

Object

Object

Object

Spring Container

Object

![ spring\_beanscopes\_im1\_springcontainer]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im1\_springcontainer.png)

Spring Container takes care of instantiation of objects,destruction of objects etc.

We can also have some objects outside spring container.

We have a Factory class which used to produce the objects and send it across to us.

But how it’s possible and does factory comes to know that what properties values should it have to instantiate the object.

This all is done by XML configuration file or Annotation configuration.

Factory Class

XML Configuration file

Our Class

New Object

![ spring\_beanscopes\_im2\_xmlconfiguration]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im2\_xmlconfiguration.png)

Here our class need to have one object, the factory class read the xml file and instantiate the object for us and transfer the same to us.

Now the same happened in Spring using BeanFactory.

Our Class

Spring Bean Factory

Spring Configuration file

Spring Bean

![ spring\_beanscopes\_im2\_xmlconfiguration]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im2\_xmlconfiguration.png)

Point here to be notice is that when Spring Bean Factory creates objects ??

Spring Beans Factory create objects at the time of we start our application or we define Application Context ie it reads spring configuration file and create objects.

Application Context

Our Class

Our Class

Our Class

Our Class

Spring XML

![ spring\_beanscopes\_im3\_applicationcontext]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im3\_applicationcontext.png)

We can also control the event of bean creation like create bean when we asks Aplication Context using getBean or at time we initialize Application Context.

This concenpt is known as “Eager Loading” and “Lazy Loading”.

**Eager Loading**: beans are created at the time of inilization of Application Context.

**Lazy Loading**: sbeans are created at the time of getBean is called.

Creating bean leads us to the concept of Bean Scopes. We have two main bean scopes:

**Singleton:** it is very much similar to Singleton design pattern and creates only single bean and It tells the container to create and manage only one instance of bean class, per container. This single instance is stored in a cache of such [singleton](https://howtodoinjava.com/design-patterns/creational/singleton-design-pattern-in-java/) beans, and all subsequent requests and references for that named bean return the cached instance.

xample of singleton scope bean using Java config –

|  |
| --- |
| @Component  //This statement is redundant - singleton is default scope  @Scope("singleton") //This statement is redundant  public class BeanClass {    } |

Example of singleton scope bean using XML config –

|  |
| --- |
| <!-- To specify singleton scope is redundant -->  <bean id="beanId" class="com.piyush.BeanClass" scope="singleton" />  //or  <bean id="beanId" class="com.piyush.BeanClass" /> |

**Prototype:** prototype scope results in the creation of a new bean instance every time a request for the bean is made by application code.

You should know that destruction [bean lifecycle methods](https://howtodoinjava.com/spring/spring-core/spring-bean-life-cycle/) are not called prototype scoped beans, only initialization callback methods are called. So as developer, you are responsible for clean up prototype-scoped bean instances and any resource there hold.

Java config example of prototype bean scope –

|  |
| --- |
| @Component  @Scope("prototype")  public class BeanClass {  } |

XML config example of prototype bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="singleton" /> |

**Singleton**

**Prototype**

Logger Bean

Logger Bean

Logger Bean

Logger Bean

Logger Bean

Logger Bean

Logger Bean

Logger Bean

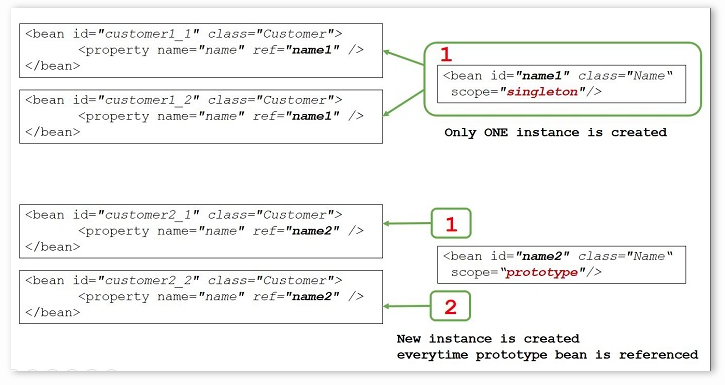
Logger Bean

Logger Bean

Logger Bean

Logger Bean

![ spring\_beanscopes\_im4\_singletonVSproto]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im4\_singletonVSproto.png)



![ spring\_beanscopes\_im5\_singletonVSproto]( https://github.com/PiyushMittl/spring/blob/master/spring\_beanscopes\_im5\_singletonVSproto.png)

**Request:**In request scope, container creates a new instance for each and every HTTP request. So, if server is currently handling 50 requests, then container can have at most 50 individual instances of bean class. Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the request is completed.

Java config example of request bean scope –

|  |
| --- |
| @Component  @Scope("request")  public class BeanClass {  }    //or    @Component  @RequestScope  public class BeanClass {  } |

XML config example of request bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="request" /> |

**Session:** In session scope, container creates a new instance for each and every HTTP session. So, if server has 20 active sessions, then container can have at most 20 individual instances of bean class. All HTTP requests within single session lifetime will have access to same single bean instance in that session scope.

Any state change to one instance, will not be visible to other instances. These instances are destructed as soon as the session is destroyed/end on server.

Java config example of session bean scope –

|  |
| --- |
| @Component  @Scope("session")  public class BeanClass {  }    //or    @Component  @SessionScope  public class BeanClass {  } |

XML config example of session bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="session" /> |

Actually Spring help you create Session scope bean instead traditional way

httpSession.setAttribute("Object",new Object());

&&

httpSession.getAttribute("Object");

and Spring provide this efficient way

@Component

@Scope("session")

public class Foo{

}

now it's headache of spring to create and destroy this associated session object using Factory Pattern

**Application:** In application scope, container creates one instance per web application runtime. It is almost similar to singleton scope, with only two differences i.e.

1. application scoped bean is singleton per ServletContext, whereas singleton scoped bean is singleton per ApplicationContext. Please note that there can be multiple application contexts for single application.
2. application scoped bean is visible as a ServletContext attribute.

Java config example of application bean scope –

|  |
| --- |
| @Component  @Scope("application")  public class BeanClass {  }    //or    @Component  @ApplicationScope  public class BeanClass {  } |

XML config example of application bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="application" /> |

**Websocket:** The [WebSocket Protocol](https://tools.ietf.org/html/rfc6455) enables two-way communication between a client and a remote host that has opted-in to communication with client. WebSocket Protocol provides a single TCP connection for traffic in both directions. This is specially useful for multi-user applications with simultaneous editing and multi-user games.

In this type of web applications, HTTP is used only for the initial handshake. Server can respond with [HTTP status](https://restfulapi.net/http-status-codes/) 101 (switching protocols) if it agrees – to handshake request. If the handshake succeeds, the TCP socket remains open and both client and server can use it to send messages to each other.

Java config example of websocket bean scope –

|  |
| --- |
| @Component  @Scope("websocket")  public class BeanClass {  } |

XML config example of websocket bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="websocket" /> |

Please note that websocket scoped beans are typically singletons and live longer than any individual WebSocket session.

**Custom Thread:** Spring also provide a non-default thread scope using class SimpleThreadScope. To use this scope, you must use register it to container using CustomScopeConfigurer class.

|  |
| --- |
| <bean class="org.springframework.beans.factory.config.CustomScopeConfigurer">      <property name="scopes">          <map>              <entry key="thread">                  <bean class="org.springframework.context.support.SimpleThreadScope"/>              </entry>          </map>      </property>  </bean> |

Every request for a bean will return the same instance within the same thread.

Java config example of thread bean scope –

|  |
| --- |
| @Component  @Scope("thread")  public class BeanClass {  } |

XML config example of thread bean scope –

|  |
| --- |
| <bean id="beanId" class="com.howtodoinjava.BeanClass" scope="thread" /> |