**Bean Scopes in Spring**  
The [spring framework](https://www.geeksforgeeks.org/introduction-to-spring-framework/) provides five scopes for a bean. We can use three of them only in the context of web-aware **Spring ApplicationContext** and the rest of the two is available for both **IoC container and Spring-MVC container**. The following are the different scopes provided for a bean:

1. **Singleton:** Only one instance will be created for a single bean definition per Spring IoC container and the same object will be shared for each request made for that bean.
2. **Prototype:**A new instance will be created for a single bean definition every time a request is made for that bean.
3. **Request:**A new instance will be created for a single bean definition every time an HTTP request is made for that bean. But Only valid in the context of a web-aware Spring ApplicationContext.
4. **Session:**Scopes a single bean definition to the lifecycle of an HTTP Session. But Only valid in the context of a web-aware Spring ApplicationContext.

i) A new bean will be created for each HTTP session by the container.

ii)This is also a scope that's used in the web-aware Application Context. It's created per every HTTP session and is present during the whole HTTP session lifecycle.

1. **Global-Session:**Scopes a single bean definition to the lifecycle of a global HTTP Session.

This scope is valid only in a web-aware Application Context. It is defined as one per global HTTP session lifecycle and is used in portlet applications. Since portlet applications have a number of portlets, and those portlets have their own sessions, **sometimes is necessary to have global variables available to all portlets**. This is where the global session scope is used.

Below two scopes are in spring new version

**6)Application Scope**

The application scope is one of two scopes introduced in the newest version of Spring. It creates one bean instance for the lifecycle of the Servlet Context.

@Scope(

value = WebApplicationContext.SCOPE\_APPLICATION, proxyMode = ScopedProxyMode.TARGET\_CLASS)

@Bean

@ApplicationScope

public MyBean myBeanApplication() {

return new MyBean();

}

or

## 7)Websocket Scope

This scope means that one bean will be created per the Websocket lifecycle. It behaves like a singleton but is bound strictly to the WebSocket session. We can define it this way:

## Difference Between Request Scope and Session Scope

First, let's mention that the HTTP protocol is stateless. If we set the bean scope to request and a user makes more than one request for a web page in his/her user session, then a new bean would be created on every request.

In case that bean scope is defined as session, if a user makes a request for a web page more than once, then the same bean is used on every request **as long as the requests are within the same user session and made from a client which is capable of maintaining the session**. For example, if we're using curl, we can't expect it to maintain the user session unless we pass the cookie/session identifier header.

BeanFactory vs ApplicationContext in Spring

**BeanFactory**

* Beanfactory present in org.springframework.beans.factory.BeanFactory package.
* If you want to use basic IOC and DI features then you should go for Beanfactory Container.
* It is a lazy initializer. It means it can’t create the bean objects at the time of creating IOC container. It creates the bean objects on our demand when we will call the method of getBean() on BeanFactory object.
* Annotation based Dependency Injection is not supported by BeanFactory.
* The XmlBeanFactory is the implementation class of the BeanFactory interface. So, to use the BeanFactory, we need to create the instance of XmlBeanFactory class.

**ApplicationContext**

ApplicationContext present in org.springframework.context.ApplicationContext package.

If you want to use advanced IOC and DI features then you should go for ApplicationContext container.

It is eager initializer. It means, at the time of creating the IOC container itself it instantiates all the beans which scope is only singleton.

Annotation based Dependency Injection is supported by ApplicationContext supports using annotation such as @PreDestroy, @Autowired

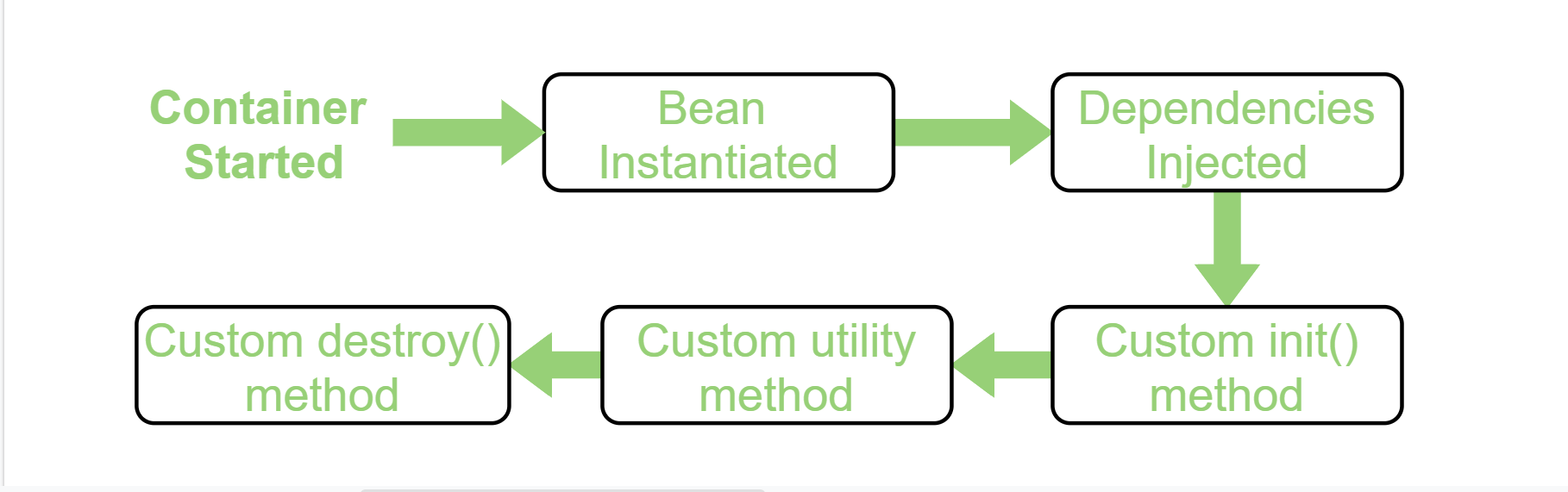
It implements the BeanFactory interface so, We can get all the functionality of the BeanFactory using it.

It supports internationalization. Hence Convenient MessageSource access (for i18n) is available in it.

The ClassPathXmlApplicationContext class is the implementation class of the ApplicationContext interface. We need to instantiate the ClassPathXmlApplicationContext class to use the ApplicationContext.

Spring :

**Bean life cycle in Java Spring:**



**Using Annotation:** To provide the facility to the created bean to invoke custom **init()** method on the startup of a spring container and to invoke the custom **destroy()** method on closing the container, we need annotate **init()** method by **@PostConstruct** annotation and **destroy()** method by **@PreDestroy** annotation.

**Note:** To invoke the **destroy()** method we have to call the **close()** method of ConfigurableApplicationContext.

Therefore, the following steps are followed:

* Firstly, we need to create a bean HelloWorld.java in this case and annotate the custom init() method with @PostConstruct and destroy() method with @PreDestroy.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| // Java program to create a bean  // in the spring framework  package beans;    import javax.annotation.PostConstruct;  import javax.annotation.PreDestroy;    // HelloWorld class  public class HelloWorld {        // Annotate this method to execute it      // automatically as the bean is      // instantiated      @PostConstruct      public void init() throws Exception      {          System.out.println(              "Bean HelloWorld has been "              + "instantiated and I'm the "              + "init() method");      }        // Annotate this method to execute it      // when Spring container is closed      @PreDestroy      public void destroy() throws Exception      {          System.out.println(              "Conatiner has been closed "              + "and I'm the destroy() method");      }  }  **What is Dependency Injection:** Dependency Injection is the main functionality provided by [Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) IOC(Inversion of Control). The Spring-Core module is responsible for injecting dependencies through either Constructor or Setter methods. The design principle of Inversion of Control emphasizes keeping the Java classes independent of each other and the container frees them from object creation and maintenance. These classes, managed by [Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/), must adhere to the standard definition of Java-Bean. Dependency Injection in [Spring](https://www.geeksforgeeks.org/introduction-to-spring-framework/) also ensures loose-coupling between the classes.  **\*\* ServletConfig** and **ServletContext**, both are objects created at the time of [servlet](https://www.geeksforgeeks.org/introduction-java-servlets/) initialization and used to provide some initial parameters or configuration information to the servlet. But, the difference lies in the fact that information shared by ServletConfig is for a specific servlet, while information shared by ServletContext is available for all servlets in the web application.  String email              = getServletConfig()                    .getInitParameter("Email");          String website              = getServletContext()                    .getInitParameter("Website-name");   | ServletConfig | ServletContext | | --- | --- | | ServletConfig is servlet specific | ServletContext is for whole application | | Parameters of servletConfig are present as name-value pair in <init-param> inside <servlet>. | Parameters of servletContext are present as name-value pair in <context-param> which is outside of <servlet> and inside <web-app> | | ServletConfig object is obtained by getServletConfig() method. | ServletContext object is obtained by getServletContext() method. | | Each servlet has got its own ServletConfig object. | ServletContext object is only one and used by different servlets of the application. | | Use ServletConfig when only one servlet needs information shared by it. | Use ServletContext when whole application needs information shared by it | |

\*How to get the servletContext and servletConfig object in spring bean?

here are two ways to get Container specific objects in the spring bean:

* Implementing Spring \*Aware interfaces, for these ServletContextAware and ServletConfigAware interfaces. So your bean needs to implement ServletContextAware and ServletConfigAware interfaces and override the setServletContext() and setServletConfig() methods.
* Using @Autowired annotation with bean variable of type ServletContext and ServletConfig.
* **HTTP Idempotent:**

when making multiple identical requests has the same effect as making a single request – then that REST API is called **idempotent**.

## Idempotency with HTTP Methods

If you follow REST principles in designing API, you will have automatically **idempotent REST APIs** for GET, PUT, DELETE, HEAD, OPTIONS and TRACE HTTP methods. Only POST APIs will not be idempotent.

1. POST is NOT idempotent.
2. GET, PUT, DELETE, HEAD, OPTIONS and TRACE are idempotent.

Let’s analyze how the above HTTP methods end up being idempotent – and why POST is not.

#### HTTP POST

Generally – not necessarily – POST APIs are used to create a new resource on server. So when you invoke the same POST request N times, you will have N new resources on the server. So, **POST is not idempotent**.

#### HTTP GET, HEAD, OPTIONS and TRACE

GET, HEAD, OPTIONS and TRACE methods NEVER change the resource state on server. They are purely for retrieving the resource representation or meta data at that point of time. So invoking multiple requests will not have any write operation on server, so **GET, HEAD, OPTIONS and TRACE are idempotent**.

#### HTTP PUT

Generally – not necessarily – PUT APIs are used to update the resource state. If you invoke a PUT API N times, the very first request will update the resource; then rest N-1 requests will just overwrite the same resource state again and again – effectively not changing anything. Hence, **PUT is idempotent**.

**What is filter in web.xml ?**

In CM Role web.xml, we have used form based authentication mechanism to authorize the resources

Also we use Role based security

**What is security constraints in web.xml?**

**Web services:**

Added security-constraints into web.xml where we defined access privileges for different role with resource URL

**Securing RESTful Web Services:**

**e.g**

<security-constraint>

<web-resource-collection>

<web-resource-name>Orders</web-resource-name>

<url-pattern>/orders</url-pattern>

<http-method>GET</http-method>

<http-method>POST</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

<realm-name>default</realm-name>

</login-config>

<security-role>

<role-name>admin</role-name>

</security-role>

### **Diff between PathParam and QueryParam**

Path params are part of the url where as query parameters are added after the ? mark symbol and separated from other query parameters by & symbol.

#### **PathParam example// is equivalent to @pathVariable**

GET http://base-url/students/{roll-number}

#### **QueryParam example//is equivalent to reuestParam**

GET http://base-url/students?grade=10

**\*Difference between @RestController and @Controller**

1. The @Controller is a common annotation that is used to mark a class as Spring MVC Controller while @RestController is a special controller used in [RESTFul web services](http://javarevisited.blogspot.sg/2015/08/difference-between-soap-and-restfull-webservice-java.html) and the equivalent of @Controller + @ResponseBody.
2. @RestController= interface defines @ResponseBody +@Controller
3. @Controller=@Component

**1)What is the difference between a REST and SOAP web service?**

REST supports different formats like text, JSON and XML whereas SOAP supports only XML.

REST works only over HTTP(S) on a transport layer while SOAP can be used with different protocols on a transport layer.

REST works with resources, each unique URL is some representation of a resource while SOAP works with operations, which implements some business logic through different interfaces.

SOAP based reads cannot be cached, need to provide caching for SOAP while REST based reads can be cached.

SOAP supports SSL security and WS-security(Web Service-security) while REST only supports SSL security.

SOAP supports ACID (Atomicity, Consistency, Isolation, Durability) while REST supports transactions, but it is neither ACID compliant nor can provide two phase commit. .

**Explain @Path annotation.**

@Path annotation binds URI pattern to a Java method.

**Explain @PathParam annotation in spring REST.**

@PathParam annotation injects the value of URI parameter that defined in @Path expression.

**What @QueryParam annotation does?**

@QueryParam annotation injects URI query parameter into Java method.

What is the difference between web api and rest api

|  |  |
| --- | --- |
| **Web Service** | **API** |
| All web services are APIs. | All APIs are not web services. |
| It supports XML. | Responses are formatted using Web API's MediaTypeFormatter into XML, JSON, or any other given format. |
| You need a SOAP protocol to send or receive and data over the network. Therefore it does not have light-weight architecture. | API has a light-weight architecture. |
| It can be used by any client who understands XML. | It can be used by a client who understands JSON or XML. |
| Web service uses three styles: REST, SOAP, and XML-RPC for communication. | API can be used for any style of communication. |
| It provides supports only for the HTTP protocol. | It provides support for the HTTP/s protocol: URL Request/Response Headers, etc. |

How ibatis is configured in LUMIS:

1)Sql-map-config-dao.xml: Added ibatis query xml

<sqlMapConfig>

<sqlMap resource=”cm/java/ibatis/ResinstMiscComDetails.xml”/>

</sqlMapConfig>

2)spring-ibatis-dao.xml

It has mentioned datasource i.e jdbc.Lumis4 is used and initializing all the the above sql-map-config-dao.xml

e.g

<bean id=”dataSource” class=”org.framwork.jndi.jndiObjectFcatoryBean”

P:jndiName=”jdbc/lumis4” p:proxyInterface=”javax.sql.DataSource”/>

<bean id=”anbDataAccessSqlMapClient” class=”org.framwork.orm.ibatis.sqlMapClientFcatoryBean”

P:configLocation=”/spring/ sql-map-config-dao.xml”

P:dataSource-ref=”dataSource”/>

<bean id=’reinstMiscImpl’ class=”com.genworth.net.ibatis.db.dao.ReinstMiscImpl”/>

Then all the daoimpl need to config into this file