Bean and Bean Lifecycle

What are Beans?

Spring Boot beans are objects managed by the Spring framework's Inversion of Control (IoC) container. These beans can be used to define the various components of an application, such as services, repositories, controllers, etc.

Here are some critical points about beans in Spring Boot:

- Beans are created by the Spring IoC container and managed throughout the application's lifecycle.
- Beans can be defined in several ways, including through Java configuration, XML configuration, and annotations.
- Beans can be wired together through dependency injection, allowing them to collaborate and work together within the application.
- Spring Boot provides several built-in beans, including the application context, the core container for managing beans, and the environment, which allows access to application properties and configuration.
- Beans can also be scoped, meaning specific context or scope determines their lifecycle. Spring Boot supports several scopes, including singleton, prototype, request, session, and Websocket.
- Overall, beans are a fundamental concept in Spring Boot and are essential for building and managing complex applications in a scalable and maintainable way.

Bean Scope

Bean scope refers to the lifecycle and visibility of a bean within the Spring container. Spring Boot supports several bean scopes, including Singleton, Prototype, Request, Session, and WebSocket.

A. Singleton

Singleton is the default bean scope in Spring Boot. A singleton bean is created only once by the Spring IoC container, and a single instance of the bean is shared throughout the application. This means the container returns the same instance whenever the application requests the same bean.

The singleton bean scope is suitable for beans that are stateless and do not have any mutable state. Examples of such beans are utility or service classes that don't maintain any state. Singleton beans are naturally thread-safe, as they are only created once and shared among multiple threads. This can improve performance and reduce memory usage, as only one instance of the bean is created and maintained throughout the application's lifecycle.

For Example, **Social Media Application** that we saw in the lecture

```
<?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="simplePostList" class="SimplePostList" scope=”singleton”>
    </bean>

    <bean id=”SimpleUser” class=”simpleUser” scope=”singleton”>
    <property name=”postList” ref=”simplePostList” />
    </bean>

</beans>
```

B. Prototype

The prototype bean scope in Spring Boot creates a new instance of a bean every time the bean is requested from the container. This means that every time the application requests a prototype bean, the container creates a new instance of the bean instead of returning an existing instance.

The prototype bean scope is suitable for beans that maintain a state or have mutable properties and where multiple instances of the bean are required. Examples of such beans are command objects or form objects that must maintain state across multiple requests.

However, prototype beans can be memory-intensive and lead to performance issues if not appropriately managed, as a new instance of the bean is created every time it is requested. Also, prototype beans are not naturally thread-safe, so the application developer is responsible for ensuring thread safety if multiple threads access the same bean instance.

For Example:

```
<?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="simplePost" class="SimplePost" scope=”prototype”>
    </bean>

    <bean id="simplePostList" class="SimplePostList" scope=”singleton”>
    </bean>

    <bean id=”SimpleUser” class=”simpleUser” scope=”singleton”>
    <property name=”postList” ref=”simplePostList” />
    </bean>

</beans>
```

C. Web Aware Scopes

C. 1 Request

The request bean scope in Spring Boot creates a new instance of a bean for every HTTP request made to the application. This means that whenever a request is made to the application, the container creates a new instance of the request-scoped bean, and the bean instance is available only within that specific request.

The request bean scope is suitable for beans that maintain a state-specific HTTP request, such as form data, user session information, or request-specific data that needs to be processed by multiple beans. This helps keep the application's state separate across numerous requests, improving the application's stability and reliability.

C. 2 Session

Session scope in Spring Boot is used to create a single instance of a bean for each user session. This means that each user accessing the application will have their unique instance of the bean, which will be created and managed by the Spring container.

When a bean is defined with session scope, a new instance is created for each HTTP session. This is useful when you want to maintain state information across multiple requests from the same user. For example, if your application has a shopping cart feature, you would like to retain the cart's contents for each user session. Defining the shopping cart bean with session scope would ensure each user has a unique cart instance.

D.Other Scopes

Some custom scopes are available in the Spring framework but not built-in in Spring Boot. The "Global Session" and "Application" are lesser-used custom scopes.

The **Java Servlet** specification (*which will study in brief in the upcoming lectures or refer to links in the later section*) provides the "**Global Session**" scope, and it defines a single bean definition of the lifecycle of a global HTTP session. This scope is valid only when used in a Servlet-based web application, such as a portlet context. When an application is built of portlets, each portlet has its session. Still, if you want to store variables globally for all portlets in your application, you can use the "Global Session" scope to create a single bean definition shared across all portlets.

The "**Application**" scope is not standard in Spring Boot or the Java Servlet specification. However, some third-party libraries or frameworks may define their custom scopes with different lifecycles and semantics, and one of them may be called the "Application" scope. When Spring creates a bean instance per web application runtime, it is similar to the Singleton scope, with one significant difference.

A Singleton scoped bean is a singleton per ApplicationContext, whereas an Application scoped bean is a singleton per ServletContext. It's important to note that multiple contexts exist for a single application.

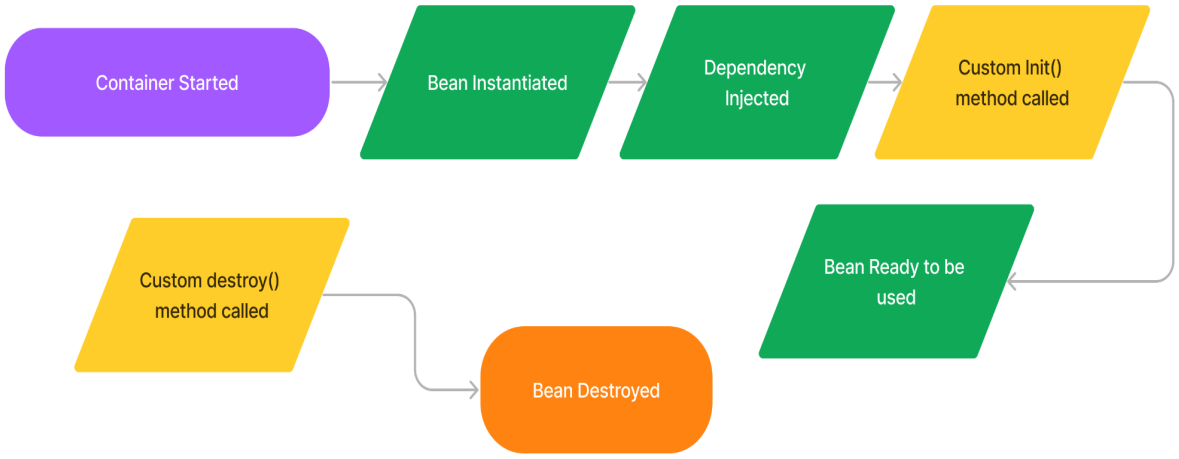
Bean Lifecycle

A bean's lifecycle in Spring Boot refers to the various phases, from creation to destruction. The bean lifecycle can be divided into the following phases:

- **Instantiation**: During this phase, a bean is created by invoking its constructor or a factory method.
- **Dependency Injection**: After instantiating a bean, any dependencies are injected using a constructor or setter injection.
- **Initialization**: Once all the dependencies have been injected, any initialisation logic that needs to be performed on the bean can be executed. This may include setting default values or configuring the bean. Developers can specify initialisation methods for a bean by using the *@PostConstruct* annotation on a method in the bean class method or by specifying an *init-method* attribute in the XML configuration file.
- **Usage**: During this phase, the bean performs its intended functionality.
- **Destruction**: When the bean is no longer needed, it is destroyed. This may involve releasing the bean's resources, such as file handles or network connections. Developers can specify destruction methods for a bean by using the *@PreDestroy* annotation on a method in the bean class method or by specifying a *destroy-method* attribute in the XML configuration file.

In Spring, the container responsible for managing the bean lifecycle is ApplicationContext. When the ApplicationContext is started, it creates and initializes all the beans defined in the application context, and when the application is shut down, it destroys all the beans.

Developers can customize the bean lifecycle by providing callback methods that Spring will invoke at specific phases in the bean lifecycle. These methods can be defined using annotations or in the XML configuration file.



For Example:

```
<?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="simplePost" class="SimplePost" scope=”prototype”
    destroy-method=”destroy”>
    </bean>

    <bean id="simplePostList" class="SimplePostList" scope=”singleton”
    init-method=”init” destroy-method=”destroy”>
    </bean>

    <bean id=”SimpleUser” class=”simpleUser” scope=”singleton”>
    <property name=”postList” ref=”simplePostList” />
    </bean>

</beans>
```

Instructor Codes

- Social Media Application

References:

1. Official spring documentation.
2. Bean Scopes
3. Bean Lifecycle
4. Medium Article on bean lifecycle
5. Java Servlets
6. Session Scope
7. HTTP Request
8. Web Socket
9. Thread safety