# SPRING BEAN SCOPES

Design by: DieuNT1

## Agenda

## Lesson Objectives

1. • Understand the Spring Bean Scopes
2. • Understand the different types of bean scopes in the Spring framework.
3. • Apply the bean scopes in the Java Spring Projects

Section 1

## Overview of Spring Bean Scopes

# Overview of Spring Bean Scopes

- Spring Bean Scopes allows us to have more granular control of the bean instances creation.
- In Spring, the scope can be defined using spring bean ***@Scope*** annotation.

| cope | Description |
|---|---|
| singleton (*default*) | Single bean instance per Spring IoC container. |
| prototype | A new bean instance is created every time a bean is requested. |
| request | Only a single instance will be created and available during the complete lifecycle of an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext. |
| session | Only a single instance will be created and available during the complete lifecycle of an HTTP Session. Only valid in the context of a web-aware Spring ApplicationContext. |
| application | Only a single instance will be created and available during the complete lifecycle of ServletContext. Only valid in the context of a web-aware Spring ApplicationContext. |
| websocket | Only a single instance will be created and available during the complete lifecycle of WebSocket. Only valid in the context of a web-aware Spring ApplicationContext. |

## Singleton Scope

The *singleton* is default bean scope in the spring container. It tells the container to create and manage only one bean class instance per container.

- This single instance is stored in a cache of such singleton beans, and all subsequent requests and references for that named bean return the cached instance.
- **XML Configuration:**

```
<!-- To specify singleton scope is redundant -->
<bean id="beanId" class="fa.training.entities.BeanClass" scope="singleton" />

//or

<bean id="beanId" class="fa.training.entities.BeanClass" />
```

## Singleton Scope

- Java Configuration:

```
@Configuration
public class AppConfiguration {
        @Bean
        @Scope("singleton")  //This statement is redundant - singleton is default scope
        public BeanClass beanClass() {
                System.out.println("A new BeanClass instance created");
                return new BeanClass();
        }
}
```

```
@Entity
@Scope("singleton")
public class BeanClass {
        // …

}
```

We can also use a constant instead of the *String* value in the following manner:

```
@Scope(value =
        ConfigurableBeanFactory.SCOPE_SINGLETON)
```

## Singleton Scope

- **XML Configuration:**

```
AnnotationConfigApplicationContext ctx = new AnnotationConfigApplicationContext();
ctx.register(AppConfiguration.class);
ctx.refresh();

Person instance1 = ctx.getBean(BeanClass.class);
System.out.println(instance1.hashCode());

BeanClass instance2 = ctx.getBean(BeanClass.class);
System.out.println(instance2.hashCode());

ctx.close();
```

- **Result:**

```
1932274274
1932274274
```

# Prototype Scope

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

- In contrast to the other scopes, Spring does **not manage the complete lifecycle of a prototype bean**.
  - ✓ We should know that **destruction bean lifecycle methods** are **not called prototype scoped beans**; only initialization callback methods are called.
  - ✓ So as a developer, we are responsible for cleaning up prototype-scoped bean instances and any resources they hold using the bean post-processor methods.
- XML Configuration:

```
<bean id="beanId" class="fa.training.entities.BeanClass" scope="prototype" />
```

## Prototype Scope

- **Java Configuration:**

```
@Bean
@Scope("prototype")
public BeanClass beanClass() {
        System.out.println("A new BeanClass instance created");
        return new BeanClass();
}
```

- **Result:**

```
A new BeanClass instance created
A new BeanClass instance created
324234234
756757767
```

## Singleton vs Prototype Scope

*A common question is that when to use singleton and prototype scope in Spring?*

*As a rule, we should use the **prototype scope for all stateful beans** and the **singleton scope for stateless beans**.*

## Web Aware Scopes

- The remaining four scopes i.e. *request*, *session*, *application* and *websocket* are only available in web applications.
- In the case of **non-web applications**, an *IllegalStateException* is thrown with a message for an unknown bean scope.
- **Request Scope**

The Web Container creates a new instance for each and every HTTP request

✓ If the server is currently handling ***50 requests***, then the ***container can have at most 50 individual instances of the bean class***. Any state change to one instance, will not be visible to other instances

✓ **A bean instance is destructed as soon as the request is completed**.

## Web Aware Scopes

- **Request Scope**
  - ✓ Java Configuration

```
@Bean
@Scope("request")
public BeanClass beanClass() {
        System.out.println("A new BeanClass instance created for current request");
        return new BeanClass();
}
//or

@Component
@RequestScope
public class BeanClass {

}
```

  - ✓ XML Configuration

```
<bean id="beanId" class="fa.training.BeanClass" scope="request" />
```

# Web Aware Scopes

- **Request Scope**

The application context creates a new instance for each and every HTTP session.

- ✓ If the server has ***20 active sessions***, then the container can have at most 20 individual instances of the bean class.
- ✓ All HTTP requests within a single session lifetime will have access to the same single bean instance in that session scope.
- ✓ Any state change to one instance will not be visible to other instances
- ✓ **A bean instance is destructed as soon as the session is completed**.

## Web Aware Scopes

- **Session Scope**
  - ✓Java Configuration

```
@Bean
@Scope("session")
public BeanClass beanClass() {
         System.out.println("A new BeanClass instance created for current request");
         return new BeanClass();
}
//or

@Component
@SessionScope
public class BeanClass {

}
```

  - ✓XML Configuration

```
<bean id="beanId" class="fa.training.BeanClass" scope="session" />
```

## Web Aware Scopes

- **Application Scope**

In application scope, the container creates one instance per web application runtime.

- It is almost similar to singleton scope with only two differences i.e:
  - ✓ The ***application*** scoped bean is singleton per ***ServletContext***, whereas ***singleton*** scoped bean is singleton per ***ApplicationContext***. Please note that there can be multiple application contexts within a single application.
  - ✓ The ***application*** scoped bean is visible as a ***ServletContext*** attribute.

# Web Aware Scopes

- **Session Scope**
  - ✓ Java Configuration

```
@Bean
@Scope("application")
public BeanClass beanClass() {
         System.out.println("A new BeanClass instance created for current request");
         return new BeanClass();
}
//or

@Component
@ApplicationScope
public class BeanClass {

}
```

  - ✓ XML Configuration

```
<bean id="beanId" class="fa.training.BeanClass" scope="application" />
```

## Web Aware Scopes

- **WebSocket Scope**

The WebSocket Protocol enables two-way communication between a client and a remote host that has opted-in to communicate with the client.

✓ WebSocket Protocol provides a single TCP connection for traffic in both directions

✓ This is especially useful for multi-user applications with simultaneous editing and multi-user games.

- When first accessed, *WebSocket* scoped beans are stored in the *WebSocket* session attributes. The same bean instance is then returned during the entire *WebSocket* session.

## Summary

- Overview of Spring Bean Scopes
- Singleton Scope
- Prototype Scope
- Web Aware Scopes: request, session, application
- Questions and Answers

# THANK YOU!