Bean Scopes

Scope of a bean determines the life span of a bean instance per container. There are two core scopes

singleton : There's only one instance of bean per Spring container (here container mean per org.springframework.context.ApplicationContext). That means regardless of how many times, we access/inject the bean there will be only one instance provided by the container. This is the default one. This is different than Java famous singleton design pattern in that there's one Spring singleton per spring container, whereas, there's one Java singleton per JVM level.

prototype : A new instance of a bean is created, each time it's injected to some other bean or accessed via the container (springContext.getBean(...)).

The singleton beans can be considered stateless (suitable for a service, DAO or controller etc) , whereas prototypes are stateful with respect to a particular calling session (for example shopping cart, wizard like steps etc).

Why don't we create prototype beans ourselves using new operator rather than registering it to the Spring container? Yes we should if we can, but prototype can be used to have Spring to do some DI in it.

The @Scope annotation

There are two ways to use @Scope annotation to assign the scopes to beans.

Using on bean factory methods of @Configuration class :

@Scope is used in @Configuration annotated class's method. These methods should primarily be annotated with @Bean.

Using on classes annotated with @Component:

@Scope is used on component classes. This classes should be scanned by Spring at startup if @ComponentScan along with packages to scan is defined on @Configuration class.

No Pre Destroy callback for Prototype

Spring does not manage the complete lifecycle of a prototype bean. The container instantiates, configures, a prototype bean instance, and hands it to the client, with no further record of the instance. That's the reason, the prototype bean's method annotated with PreDestroy will never be called. The initialization lifecycle callback methods (@PostConstruct) are always called on all objects regardless of scope.

1-Singleton Bean Example

Following example shows that singleton bean is created only once per Spring container.

Example

A singleton bean

```
public class ServiceBean {


}
```

Other beans injecting singleton bean

```
public class ClientBean1 {
  @Autowired
  private ServiceBean serviceBean;

  public void doSomething(){
      System.out.println("from ClientBean1: serviceBean:
"+System.identityHashCode(serviceBean));
  }
}

public class ClientBean2 {
  @Autowired
  private ServiceBean serviceBean;

  public void doSomething(){
      System.out.println("from ClientBean2: serviceBean:
"+System.identityHashCode(serviceBean));
  }
}
```

Defining beans and running example app

```
public class AppMain {
```

```
@Scope(ConfigurableBeanFactory.SCOPE_SINGLETON)
@Bean
public ServiceBean serviceBean(){
    return new ServiceBean();
}

@Bean
public ClientBean1 clientBean1(){
    return new ClientBean1();
}

@Bean
public ClientBean2 clientBean2(){
    return new ClientBean2();
}

public static void main(String[] args) {
    runApp();
    runApp();
}

private static void runApp() {
    System.out.println("--- running app ---");
```

```
        AnnotationConfigApplicationContext context =
                new AnnotationConfigApplicationContext(AppMain.class);
        context.getBean(ClientBean1.class).doSomething();
        context.getBean(ClientBean2.class).doSomething();
    }
}
```

Output

```
--- running app ---
from ClientBean1: serviceBean: 747161976
from ClientBean2: serviceBean: 747161976
--- running app ---
from ClientBean1: serviceBean: 1696188347
from ClientBean2: serviceBean: 1696188347
```

We created the spring context twice to show that Spring singleton beans have different instance in different spring context (container), so they are not JVM level singletons.

2-Prototype Bean Example

Following example shows that a new instance is used whenever a prototype bean is injected.

Example

A prototype bean

```
public class ServiceBean {
```

```
}
```

Other beans injecting prototype bean

```
public class ClientBean1 {
  @Autowired
  private ServiceBean serviceBean;

  public void doSomething(){
      System.out.println("from ClientBean1: serviceBean:
"+System.identityHashCode(serviceBean));
  }
}

public class ClientBean2 {
  @Autowired
  private ServiceBean serviceBean;

  public void doSomething(){
      System.out.println("from ClientBean2: serviceBean:
"+System.identityHashCode(serviceBean));
  }
}
```

Defining beans and running example app

```
public class AppMain {
  @Scope(ConfigurableBeanFactory.SCOPE_PROTOTYPE)
  @Bean
  public ServiceBean serviceBean(){
      return new ServiceBean();
  }

  @Bean
  public ClientBean1 clientBean1(){
      return new ClientBean1();
  }

  @Bean
  public ClientBean2 clientBean2(){
      return new ClientBean2();
  }

  public static void main(String[] args) {
      AnnotationConfigApplicationContext context =
              new AnnotationConfigApplicationContext(AppMain.class);
```

```
        context.getBean(ClientBean1.class).doSomething();
        context.getBean(ClientBean2.class).doSomething();
    }
}
```

Output

```
from ClientBean1: serviceBean: 661422630
from ClientBean2: serviceBean: 1976448008
```