Enterprise Programming using JAVA

Chapter-5: Spring

Prof. ARNIKA PATEL
Assistant Professor
Department of CSE

Parul® University

NAAC A++

Content

INDEX

Autowiring

- **Autowiring** in the Spring framework can inject dependencies automatically. The Spring container detects those dependencies specified in the configuration file and the relationship between the beans. This is referred to as **Autowiring in Spring**.
- To enable Autowiring in the Spring application we should use @**Autowired** annotation. Autowiring in Spring internally uses constructor injection. An autowired application requires fewer lines of code comparatively but at the same time, it provides very little flexibility to the programmer.

Modes of Autowiring

Modes	Description
No	This mode tells the framework that autowiring is not supposed to be done. It is the default mode used by Spring.
byName	It uses the name of the bean for injecting dependencies.
byType	It injects the dependency according to the type of bean.
Constructor	It injects the required dependencies by invoking the constructor.
Autodetect	The autodetect mode uses two other modes for autowiring - constructor and byType.

Parul® University NAAC A++

Modes of Autowiring

1. No

This mode tells the framework that autowiring is not supposed to be done. It is the default mode used by Spring.

```
<bean id="state" class="sample.State">
        <property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"></bean>
```

Modes of Autowiring

2. byName

It uses the name of the bean for injecting dependencies. However, it requires that the name of the property and bean must be the same. It invokes the setter method internally for autowiring.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"
autowire="byName"></bean>
```

Modes of Autowiring

3. byType

It injects the dependency according to the type of the bean. It looks up in the configuration file for the class type of the property. If it finds a bean that matches, it injects the property. If not, the program throws an error. The names of the property and bean can be different in this case. It invokes the setter method internally for autowiring.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"
autowire="byType"></bean>
```

Modes of Autowiring

4. constructor

It injects the required dependencies by invoking the constructor. It works similar to the “byType” mode but it looks for the class type of the constructor arguments. If none or more than one bean are detected, then it throws an error, otherwise, it autowires the “byType” on all constructor arguments.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"
autowire="constructor"></bean>
```

Modes of Autowiring

5. autodetect

The autodetect mode uses two other modes for autowiring - constructor and byType. It first tries to autowire via the constructor mode and if it fails, it uses the byType mode for autowiring. It works in Spring 2.0 and 2.5 but is deprecated from Spring 3.0 onwards.

```
<bean id="state" class="sample.State">
<property name="name" value="UP" />
</bean>
<bean id="city" class="sample.City"
autowire="autodetect"></bean>
```

Autowiring

Advantage of Autowiring: It requires less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring: No control of the programmer and It can't be used for primitive and string values.

Spring- Application Context

- **ApplicationContext** belongs to the **Spring** framework.
- **Spring IoC contain**er is responsible for instantiating, wiring, configuring, and managing the entire life cycle of beans or objects.
- **BeanFactory** and **ApplicationContext** represent the Spring IoC Containers.
- **ApplicationContext** is the sub-interface of BeanFactory. It is used when we are creating an enterprise-level application or web application. ApplicationContext is the superset of BeanFactory, whatever features provided by BeanFactory are also provided by ApplicationContext.

Spring- Application Context

ApplicationContext Features

ApplicationContext provides basic features in addition to enterprise-specific functionalities which are as follows:

- Publishing events to registered listeners by resolving property files.
- Methods for accessing application components.
- Supports Internationalization.
- Loading File resources in a generic fashion.

Spring- Application Context

ApplicationContext Implementation Classes

1. AnnotationConfigApplicationContext container
2. AnnotationConfigWebApplicationContext
3. XmlWebApplicationContext
4. FileSystemXmlApplicationContext
5. ClassPathXmlApplicationContext

Spring- Application Context

Container 1: AnnotationConfigApplicationContext

AnnotationConfigApplicationContext class was introduced in Spring 3.0. It accepts classes annotated with @Configuration, @Component, and JSR-330 compliant classes. The constructor of AnnotationConfigApplicationContext accepts one or more classes.

```
ApplicationContext context = new
AnnotationConfigApplicationContext(AppConfig.class,
AppConfig1.class);
```

Spring- Application Context

Container 2: AnnotationConfigWebApplicationContext

AnnotationConfigWebApplicationContext class was introduced in Spring 3.0. It is similar to AnnotationConfigApplicationContext for a web environment. It accepts classes annotated with @Configuration, @Component, and JSR-330 compliant classes. These classes can be registered via ***register() method*** or passing base packages to ***scan() method***.

Spring- Application Context

Container 2: AnnotationConfigWebApplicationContext

```
// Implementing WebApplicationInitializer
public class MyWebApplicationInitializer implements
WebApplicationInitializer {
// Servlet container
 public void onStartup(ServletContext container) throws
ServletException {
AnnotationConfigWebApplicationContext context = new
AnnotationConfigWebApplicationContext();
context.register(AppConfig.class);
context.setServletContext(container);
// Servlet configuration } }
```

Spring- Application Context

Container 3: XmlWebApplicationContext

Spring MVC Web-based application can be configured completely using XML or Java code. Configuring this container is similar to the AnnotationConfigWebApplicationContext container, which implies we can configure it in web.xml or using java code.

```
XmlWebApplicationContext context = new
XmlWebApplicationContext();
context.setConfigLocation("/WEB-INF/spring/applicationCont
ext.xml"); context.setServletContext(container);
```

Spring- Application Context

Container 4: FileSystemXmlApplicationContext

FileSystemXmlApplicationContext is used to load XML-based Spring Configuration files from the file system or from URL. We can get the application context using Java code. It is useful for standalone environments and test harnesses. The following code shows how to create a container and use the XML as metadata information to load the beans.

```
ApplicationContext context = new
FileSystemXmlApplicationContext(path);
```

Parul® University NAAC A++

Spring- Application Context

Container 5: ClassPathXmlApplicationContext

FileSystemXmlApplicationContext is used to load XML-based Spring Configuration files from the classpath. We can get the application context using Java code. It is useful for standalone environments and test harnesses. The following code shows how to create a container and use the XML as metadata information to load the beans.

```
ApplicationContext context = new
ClassPathXmlApplicationContext("applicationcontext/student-
bean-config.xml");
```

Spring - Annotation Based Configuration

Starting from Spring 2.5 it became possible to configure the dependency injection using **annotations**. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.

Annotation injection is performed before XML injection. Thus, the latter configuration will override the former for properties wired through both approaches.

Spring - Annotation Based Configuration

Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file. So consider the following configuration file in case you want to use any annotation in your Spring application.

```
<beans…>
        <context:annotation-config/>
        <!-- bean definitions go here -->
</beans>
```

Spring - Annotation Based Configuration

Once <context:annotation-config/> is configured, you can start annotating your code to indicate that Spring should automatically wire values into properties, methods, and constructors.

Spring - Annotation Based Configuration

Sr.No	Annotation & Description
1	**@Required** The @Required annotation applies to bean property setter methods.
2	**@Autowired** The @Autowired annotation can apply to bean property setter methods, non-setter methods, constructor and properties.
3	**@Qualifier** The @Qualifier annotation along with @Autowired can be used to remove the confusion by specifiying which exact bean will be wired.
4	**JSR-250 Annotations** Spring supports JSR-250 based annotations which include @Resource, @PostConstruct and @PreDestroy annotations.

Parul® University NAAC A++

PPT Content Resources Reference Sample:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly

2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..

3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .

4. **Sources**

   https://developers.redhat.com/topics/enterprise-java

5. **Article**

   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/