IoC Container

Theory | Practice | 100% completed, 1 problem solved

Theory

14 minutes reading

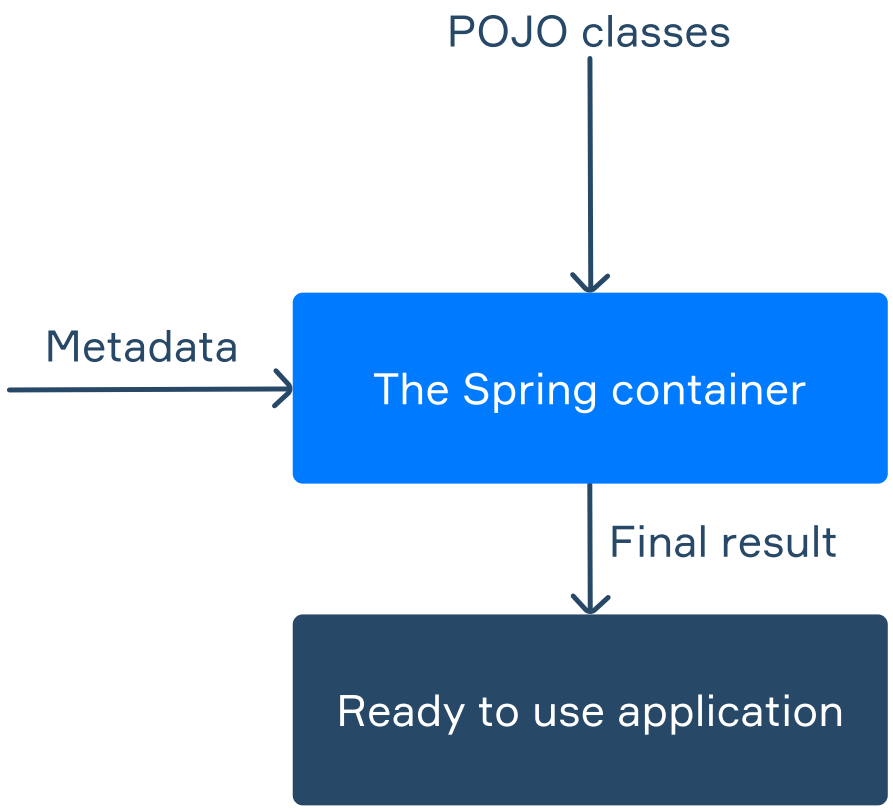
Unskip this topic | Start practicing

IoC, which stands for **Inversion of Control**, is the mechanism used by Spring to implement dependency injection. When we create applications, we often need different objects to implement various functionalities. Some objects will need to use other objects as their dependencies, which in turn may require other objects, and so on. In order to simplify this long and complex process, Spring uses dependency injection.

Using the IoC process, objects can define the dependencies they need to run successfully. These dependencies are defined through constructor arguments, factory method arguments, or properties set on the object instance. In this topic, we will overview the components of IoC, and see how they work together to create the dependency injection functionality.

§1. Spring container

When we need to have a ready-to-use spring application, we need a few components to help implement the required functionality. The image below shows a typical structure of a Spring application.



Let's start by looking at the Spring container since it is the core of the Spring framework. The Spring container manages the lifecycle of the application from start to finish. It manages various components created for the application and handles any required dependency injections. The Spring container can be configured through metadata in a number of ways. There are two types of metadata that are used in Spring: **XML** and **annotations**. The XML approach involves defining class-related data in an external XML file, which can then be loaded and used in the Spring application.

In our topics, we use the annotation-based approach, and it is recommended for all new Spring-based applications.

The annotation-based approach involves adding annotations to simple classes in order to provide context and functionality for Spring. These annotations will start

2 required topics

- Dependency injection
- Basic project structure

2 dependent topics

- Spring beans
- Introduction to Spring Testing

with the `@` character, and provide a specific value that we wish to add to our class. These values will allow us to build objects with the required features and configurations. These objects are known as **POJO classes**, and in the next section, we will see how they fit into the Spring framework.

§2. POJO

In the previous diagram, we saw that the Spring container takes in a POJO class. The term **POJO** stands for **Plain Old Java Object**. A POJO is the most basic type of an object and it contains no ties to frameworks. This means that POJOs are valid objects for any application. The idea of a POJO is that it is the simplest possible unit of code available for an application. They can implement properties, as well as getters and setters for these properties, but cannot extend or implement framework-specific classes and interfaces classes or contain annotations.

The simplicity of POJO makes them ideal building blocks for any application component we need to implement. Beside POJOs, Spring can use a special type of POJO called **JavaBean**. With **JavaBeans**, we add a few more requirements: for example, classes are required to be serializable. In addition, they require private fields and a no-argument constructor to be available. These classes can also be customized and configured using Spring metadata. To do this, we can add various annotations to the classes we create in Spring. For example, the `@Bean` annotation can be added to a factory method to define that the class it produces is a **Spring Bean** which means an object managed by the IoC container. With these annotations, it is possible to add any configurations to preexisting classes without the need for creating additional files. This will allow you to take full advantage of the features provided by Spring.

To sum it all up: POJO is a simple object which doesn't depend on the framework; Java Bean is a POJO with some additional requirements and restrictions; and Spring Bean is POJO or JavaBean created and managed by an instance of the Spring IoC container.

§3. Contexts and bean factory

When we work with the Spring IoC, there are two components we should be aware of. The first one is the `BeanFactory`, an interface that allows for configuration and management of objects. The `BeanFactory` can be used to produce container-managed objects known as beans, which can organize the backbone of your application. These beans look like regular Java objects, but they can be created during application startup, registered, and injected into different parts of the application by the container.

The second component is the `ApplicationContext`, which is a sub-interface of the `BeanFactory`. The goal of the `ApplicationContext` is to facilitate integration with Spring's Aspect Oriented Programming (AOP) functionality. This functionality includes a variety of components, ranging from message resource handling to application layer-specific contexts. There are three main `ApplicationContext` implementations that we typically see in applications:

- `FileSystemXmlApplicationContext`
- `ClassPathXmlApplicationContext`
- `WebApplicationContext`

The `FileSystemXmlApplicationContext` will load bean definitions from an XML file that is provided as a full file system path to the constructor. This means that beans are initialized based on the contents of a file from the file system of the application that is being run. For `ClassPathXmlApplicationContext`, beans are still loaded from an XML file, however, the file is provided as the CLASSPATH property rather than a full system path in the constructor. Finally, `WebApplicationContext` is generally used to set the configuration of a web application in Spring. When using `WebApplicationContext`, you will often set the servlet configuration within a

`web.xml` file. Inside this file, you can specify configurations for each servlet that the application uses.

With our `ApplicationContext`, we can configure the Spring IoC container, allowing us to create an application that is ready for use.

§4. Conclusion

The IoC container in Spring enables many important mechanisms required to create applications. With the IoC, we can create `ApplicationContext` objects, which can provide important configurations for the application setup. In addition, it allows us to create POJOs such as Java beans, which can be used to create our own entities in the application. These entities can help us bring life to our app, and achieve the required functionality for it to do its job the best way possible. You will learn how to use IoC to create Spring Beans in practice in subsequent topics.

Report a typo

127 users liked this piece of theory. 64 didn't like it. **What about you?**

Start practicing

Unskip this topic

Table of contents:

Comments (8) Useful links (8) Show discussion