Spring core

# Spring context

The Spring Framework implements the Inversion of Control (IoC) principle (IoC is also known as dependency injection (DI)). What is inversion of control (IoC) - <https://stackoverflow.com/questions/3058/what-is-inversion-of-control>.

It is a process whereby objects define their dependencies only through:

* constructor arguments;
* arguments to a factory method;
* properties that are set on the object instance after it is constructed or returned from a factory method.

The container then injects those dependencies when it creates the bean.

The basis for Spring Framework’s IoC container are packages:

* org.springframework.beans;
* org.springframework.context.

The [BeanFactory](https://docs.spring.io/spring-framework/docs/5.3.10/javadoc-api/org/springframework/beans/factory/BeanFactory.html) interface provides an advanced configuration mechanism capable of managing any type of object. [ApplicationContext](https://docs.spring.io/spring-framework/docs/5.3.10/javadoc-api/org/springframework/context/ApplicationContext.html) is a sub-interface of BeanFactory. It adds:

* Easier integration with Spring’s AOP features;
* Message resource handling (for use in internationalization);
* Event publication;
* Application-layer specific contexts such as the WebApplicationContext for use in web applications.

In short, the **BeanFactory** provides the configuration framework and basic functionality, and the **ApplicationContext** adds more enterprise-specific functionality.

The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata. The configuration metadata is represented in:

* XML;
* Java annotations;
* Java code.

Often, each individual XML configuration file represents a logical layer or module in your architecture.

# Beans

A Spring IoC container manages one or more beans. These beans are created with the configuration metadata that you supply to the container (for example, in the form of XML <bean/> definitions).

Within the container itself, these bean definitions are represented as **BeanDefinition** objects, which contain (among other information) the following metadata:

| **Property** | **Explained in…​** |
| --- | --- |
| Class | [Instantiating Beans](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-class) |
| Name | [Naming Beans](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-beanname) |
| Scope | [Bean Scopes](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-scopes) |
| Constructor arguments | [Dependency Injection](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-collaborators) |
| Properties | [Dependency Injection](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-collaborators) |
| Autowiring mode | [Autowiring Collaborators](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-autowire) |
| Lazy initialization mode | [Lazy-initialized Beans](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-lazy-init) |
| Initialization method | [Initialization Callbacks](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-lifecycle-initializingbean) |
| Destruction method | [Destruction Callbacks](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-factory-lifecycle-disposablebean) |

Every bean has one or more identifiers. These identifiers must be unique within the container that hosts the bean. A bean usually has only one identifier. However, if it requires more than one, the extra ones can be considered aliases. In XML-based configuration metadata, you use:

* the “id” attribute;
* the “name” attribute;
* both.

to specify the bean identifiers. The id attribute lets you specify exactly one id.

Specifying all aliases where the bean is actually defined is not always adequate, however. It is sometimes desirable to introduce an alias for a bean that is defined elsewhere. This is commonly the case in large systems where configuration is split amongst each subsystem, with each subsystem having its own set of object definitions. In XML-based configuration metadata, you can use the **<alias/>** element to accomplish this. The following example shows how to do so:

<alias name="fromName" alias="toName"/>

# Dependency injection

Dependency injection (DI) is a process whereby objects define their dependencies only through:

* constructor arguments;
* arguments to a factory method;
* properties (that are set on the object instance after it is constructed or returned from a factory method).

DI exists in two major variants: [Constructor-based dependency injection](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-constructor-injection) and [Setter-based dependency injection](https://docs.spring.io/spring-framework/docs/current/reference/html/core.html#beans-setter-injection). Calling a static factory method with specific arguments to construct the bean is nearly equivalent, and this discussion treats arguments to a constructor and to a static factory method similarly.

# References

* IDE for developing enterprise applications using Spring Framework and Spring Boot, the new generation of Spring Tools provides world-class development support for your Spring applications - <https://spring.io/tools>.