- Achitecture Spring: https://viblo.asia/p/spring-architecture-DzVkpBKgGnW

- Spring makes it easy to create Java enterprise applications. It provides everything you need to embrace the Java language in an enterprise environment

- At the heart are the modules of the core container, including a configuration model and a dependency injection mechanism

- The Spring Framework is modular and publishes 20+ different jars:

spring-aop spring-context-indexer spring-instrument spring-orm spring-webflux

spring-aspects spring-context-support spring-jcl spring-oxm spring-webmvc

spring-beans spring-core spring-jdbc spring-test spring-websocket

spring-beans-groovy spring-expression spring-jms spring-tx

spring-context spring-framework-bom spring-messaging spring-web

Một vài module phụ thuộc lẫn nhau như: spring-context phụ thuộc spring-beans, spring-beans lại phụ thuộc vào spring core

- The Spring Artifactory is a repository manager

Core: IoC container, Events, Resources, i18n, Validation, Data Binding, Type Conversion, SpEL, AOP.

- Quan trọng nhất trong số này là Spring Framework’s Inversion of Control (IoC) container

[1.1. Introduction to the Spring IoC container and beans](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-introduction)

- **IoC is also known as dependency injection (DI): là xử lý mà nhờ đó các đối tượng xác định được các dependencies của chúng hay những object chúng làm việc với** :

- only through constructor arguments, arguments to a factory method, or 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 (bean được khởi tạo trước sau đó khi nơi nào cần sẽ được container injects)

Bean điều khiển việc khởi tạo và vị trí các dependencies của nó bằng việc using direct construction of classes, or a mechanism such as the Service Locator pattern

- The org.springframework.beans and org.springframework.context packages are the basis for Spring Framework’s IoC container

The [BeanFactory](https://docs.spring.io/spring-framework/docs/5.0.6.BUILD-SNAPSHOT/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/5.0.6.BUILD-SNAPSHOT/javadoc-api/org/springframework/context/ApplicationContext.html) is a sub-interface of BeanFactory.

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

- **In Spring, the objects that form the backbone of your application and that are managed by the Spring IoC container are called beans**

A bean là 1 object mà được khởi tạo, nhúng hay do IoC quản lý.

[**1.2. Container overview**](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-basics)

The interface org.springframework.context.ApplicationContext represents the Spring IoC container

Container lấy các chỉ thị hướng dẫn trên các objects để khởi tạo, cấu hình và nhúng bằng việc đọc configuration metadata.

**The configuration metadata is represented in XML, Java annotations, or Java code.**

Several implementations of the ApplicationContext interface are [ClassPathXmlApplicationContext](https://docs.spring.io/spring-framework/docs/5.0.6.BUILD-SNAPSHOT/javadoc-api/org/springframework/context/support/ClassPathXmlApplicationContext.html) or [FileSystemXmlApplicationContext](https://docs.spring.io/spring-framework/docs/5.0.6.BUILD-SNAPSHOT/javadoc-api/org/springframework/context/support/FileSystemXmlApplicationContext.html)

While XML has been the traditional format for defining configuration metadata you can **instruct the container to use Java annotations** or code as the metadata format **by providing a small amount of XML configuration** to declaratively enable support.

*The Spring IoC container*



this configuration metadata represents how you as an application developer tell the Spring container to instantiate, configure, and assemble the objects in your application.

XML-based configuration metadata:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="..." class="...">

*<!-- collaborators and configuration for this bean go here -->*

</bean>

<bean id="..." class="...">

*<!-- collaborators and configuration for this bean go here -->*

</bean>

*<!-- more bean definitions go here -->*

</beans>

Khởi tạo 1 container:

ApplicationContext context = **new** ClassPathXmlApplicationContext("services.xml", "daos.xml");

| *Table 1. The bean definition* | |
| --- | --- |
| **Property** | **Explained in…​** |
| class | [Instantiating beans](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-class) |
| name | [Naming beans](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-beanname) |
| scope | [Bean scopes](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-scopes) |
| constructor arguments | [Dependency Injection](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-collaborators) |
| properties | [Dependency Injection](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-collaborators) |
| autowiring mode | [Autowiring collaborators](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-autowire) |
| lazy-initialization mode | [Lazy-initialized beans](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-lazy-init) |
| initialization method | [Initialization callbacks](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-lifecycle-initializingbean) |
| destruction method | [Destruction callbacks](https://docs.spring.io/spring/docs/5.0.6.BUILD-SNAPSHOT/spring-framework-reference/core.html#beans-factory-lifecycle-disposablebean) |

**Dependency injection (DI)** is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or 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.

Constructor-based dependency injection

**public** **class** **SimpleMovieLister** {

*// the SimpleMovieLister has a dependency on a MovieFinder*

**private** MovieFinder movieFinder;

*// a constructor so that the Spring container can inject a MovieFinder*

**public** SimpleMovieLister(MovieFinder movieFinder) {

this.movieFinder = movieFinder;

}

*// business logic that actually uses the injected MovieFinder is omitted...*

}

When a simple type is used, such as <value>true</value>, Spring cannot determine the type of the value, and so cannot match by type without help. Consider the following class:

the container can use type matching with simple types if you explicitly specify the type of the constructor argument using the type attribute. For example:

*- Constructor argument type matching*

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg type="int" value="7500000"/>

<constructor-arg type="java.lang.String" value="42"/>

</bean>

*- Constructor argument index*

Use the index attribute to specify explicitly the index of constructor arguments. For example:

<bean id="exampleBean" class="examples.ExampleBean">

<constructor-arg index="0" value="7500000"/>

<constructor-arg index="1" value="42"/>

</bean>

##### Setter-based dependency injection

Setter-based DI is accomplished by the container calling setter methods on your beans after invoking a no-argument constructor or no-argument static factory method to instantiate your bean.

**public** **class** **SimpleMovieLister** {

*// the SimpleMovieLister has a dependency on the MovieFinder*

**private** MovieFinder movieFinder;

*// a setter method so that the Spring container can inject a MovieFinder*

**public** **void** setMovieFinder(MovieFinder movieFinder) {

this.movieFinder = movieFinder;

}

*// business logic that actually uses the injected MovieFinder is omitted...*

}

The ApplicationContext supports constructor-based and setter-based DI for the beans it manages. It also supports setter-based DI after some dependencies have already been injected through the constructor approach.

Since you can mix constructor-based and setter-based DI, it is a good rule of thumb to use constructors for mandatory dependencies and setter methods or configuration methods for optional dependencies.

The container performs bean dependency resolution as follows:

* The ApplicationContext is created and initialized with configuration metadata that describes all the beans. Configuration metadata can be specified via XML, Java code, or annotations.
* For each bean, its dependencies are expressed in the form of properties, constructor arguments, or arguments to the static-factory method if you are using that instead of a normal constructor. These dependencies are provided to the bean, when the bean is actually created.
* Each property or constructor argument is an actual definition of the value to set, or a reference to another bean in the container.
* Each property or constructor argument which is a value is converted from its specified format to the actual type of that property or constructor argument. By default Spring can convert a value supplied in string format to all built-in types, such as int, long, String, boolean, etc.

<bean … **autowire=”byName”**></bean>: Khi chạy thấy bean nào có cái này, spring container sẽ xem các thuộc tính của lớp đó và tìm các bean khác có cùng tên <bean name=”example” …></bean>, nếu có sẽ wired

<bean … **autowire=”byType”**></bean>: Khi chạy thấy bean nào có cái này, spring container sẽ xem các thuộc tính của lớp đó và tìm các bean nào mà có tên cùng với kiểu của thuộc tính <bean name=”SpellCheck” …></bean>, nếu có sẽ wired:

Class **SpellCheck** {

}

Class Editor { private **SpellCheck** spellcheck; …}