Dependency Injection

Inversion of Control (IOC)

Definition/What

It is a design guideline.

In IOC, the spring container takes care of object creations and injecting the dependencies to the class.

At runtime, IOC builds the class's object and makes it available wherever it is required.

Need/Why

Need 1 : The mechanism to achieve loose-coupling between Objects dependencies.

Need 2 : Decoupling the execution of a task from its implementation

Need 3 : Greater modularity of a program

Implementation/How it internally works

We can achieve Inversion of Control through dependency injection

Dependency Injection (DI)

Definition/What

A technique, which is used to implement inversion of control.

Need/Why

Need 1: loose coupling of components.

Need 2: Moves the responsibility of managing components onto the container

Implementation/How it internally works

Spring container injects the objects/manages dependency injection.

Configuration of dependencies are read from XML, annotations or JavaConfig. Then Spring DI engine wires the dependencies based on the metadata from the configuration using the Java reflection API.

Real Time Example

```
class Employee
{
Address address;
}
```

In above class address bean is injected to Employee Class using dependency injection

Can be achieved through either setter based /constructor based injection

Alternatives/Differences

Difference between IOC and DI

IOC: It is a design pattern which says underlying environment is responsible to inject dependencies

DI: Implementation of IOC design pattern in spring which says spring container is responsible to inject dependencies

Setter Injection

Definition/What

Injecting the bean dependencies using the Setter methods on an Object.

Need/Why

Need 1: While constructor injection may suffer from cyclic dependencies, setter injection can easily avoid such situation without modifying the classes.

Need 2 : Some classes from third party libraries may not have proper constructors, here setter based injection can be used

Implementation/How it internally works

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.

I.e. the object is created first and then the dependency is injected.

Real Time Example

Objects can be configured through XML configuration or annotation

XML based configuration:

<property name="__">

<value> ____ </value>

</property>

Annotation based configuration:

Use @Autowired on setter method.

Constructor Injection

Definition/What

Object's constructor is used to inject dependencies

Need/Why

Need 1: Helps in creating immutable objects because a constructor's signature is the only possible way to create objects.

Need 2 : All required dependencies are available at initialization time

Implementation/How it internally works

The IoC container makes sure that all the arguments provided in the constructor are available before passing them into the constructor.

This helps in preventing the NullPointerException.

Real Time Example

Objects can be configured through XML configuration or annotation

XML based configuration:

<constructor-arg> <value> ____ </value> </constructor-arg>

Annotation based configuration:

Use @Autowired on bean constructor method.

ADDITIONAL INFORMATION

This type of injection is safer as the objects won't get created if the dependencies aren't available or dependencies cannot be resolved

Constructor injection is extremely useful since we do not have to write separate business logic everywhere to check if all the required dependencies are loaded, thus simplifying code complexity.

Alternatives/Differences

Difference between setter based and constructor based injection:

Setter based dependency injection should be used for optional dependencies.

Constructor based dependency injection should be used for mandatory dependencies.

Autowiring

DEFINITION/WHAT

Enables you to inject the object dependency implicitly

Eliminate the need for <property> or <constructor-arg> elements by letting spring automatically figure out how to wire dependency

NEED/WHY

Need 1: Autowiring makes the container to search the bean configurations and do the collaboration among beans, without the developer specifically mentioning these.

Need 2: Spring configuration can be minimized

IMPLEMENTATION/HOW IT INTERNALLY WORKS

Autowiring happens by placing an instance of one bean into the desired field in an instance of another bean.

Both classes should be beans, i.e. they should be defined to live in the application context, "living" in the application context means that the context instantiates the objects, not you(developer)

ADDITIONAL INFORMATION

Autowiring can't be used to inject primitive and string values. It works with reference only

The explicit wiring using <property> and <construcor-arg> elements always have more priority than autowiring

ALTERNATIVES/DIFFERENCES

There are 4 modes of autowiring -"no", "byName", "byType", "constructor"

Difference between modes of autowiring:

No - Explicit wiring is required using either setter /constructor based annotation

byName - The container automatically wired if the property name in bean class matches with 'id' attribute of <bean> element in configuration file. Unmatched property names will remain unwired

byType -Automatically wired if the bean types are assignable to the property data type. Unmatched property names will remain unwired

constructor - Automatically wired if the bean types are assignable to the constructor parameter types

Injecting Collections

Definition/What

Injecting Bean references as collection

Need/Why

Need 1: Enables creating multiple beans of same class and inject all of them to a collection

Implementation/How it internally works

Spring framework supports the injection of the java collection types List, Set, Map and properties.

ADDITIONAL INFORMATION

Spring framework provides the facility to inject collection values via constructor or setter method.

Alternatives/Differences

Java Collection types such as List, Set, Map, and Properties can be injected as

<list> : use this for wiring /injecting list of values , allowing duplicates
<set> : use this for wiring a set of values but without any duplicates
<map> : use this to inject a collection of name-value pairs where name and value can be of any type
<props> : This can be used to inject a collection of name-value pairs where the name and value are both Strings.

Inner Beans

Definition/What

Beans that are defined within the scope of another bean.

Need/Why

Need 1: whenever a bean is used for only one particular property, it's advise to declare it as an inner bean.

Implementation/How it internally works

A <bean/> element inside the <property/> or <constructor-arg/> elements is called inner bean and it is shown below.

```
<bean id="outerBean" class="___">
    <property name="target">
        <bean id="innerBean" class="___"/>
    </property>
</bean>
```

ADDITIONAL INFORMATION

The id or name value in bean class is not necessary in an inner bean, it will simply ignored by the Spring container