Spring Framework

An Introduction to Spring Framework

Inversion of Control (IoC) and Dependency Injection (DI)

Inversion of Control

IoC is a programming principle/software architecture in which the control flow of a program is inverted. In traditional programming, the main program calls the reusable code available as part of libraries to perform generic tasks, whereas in IoC an framework will take the task of calling the custom code.
Such architectures make your software modular and extensible. Implementations include:

- Service locator pattern
- Dependency injection
- Context Lookup
- Template methods pattern
- Strategy design pattern

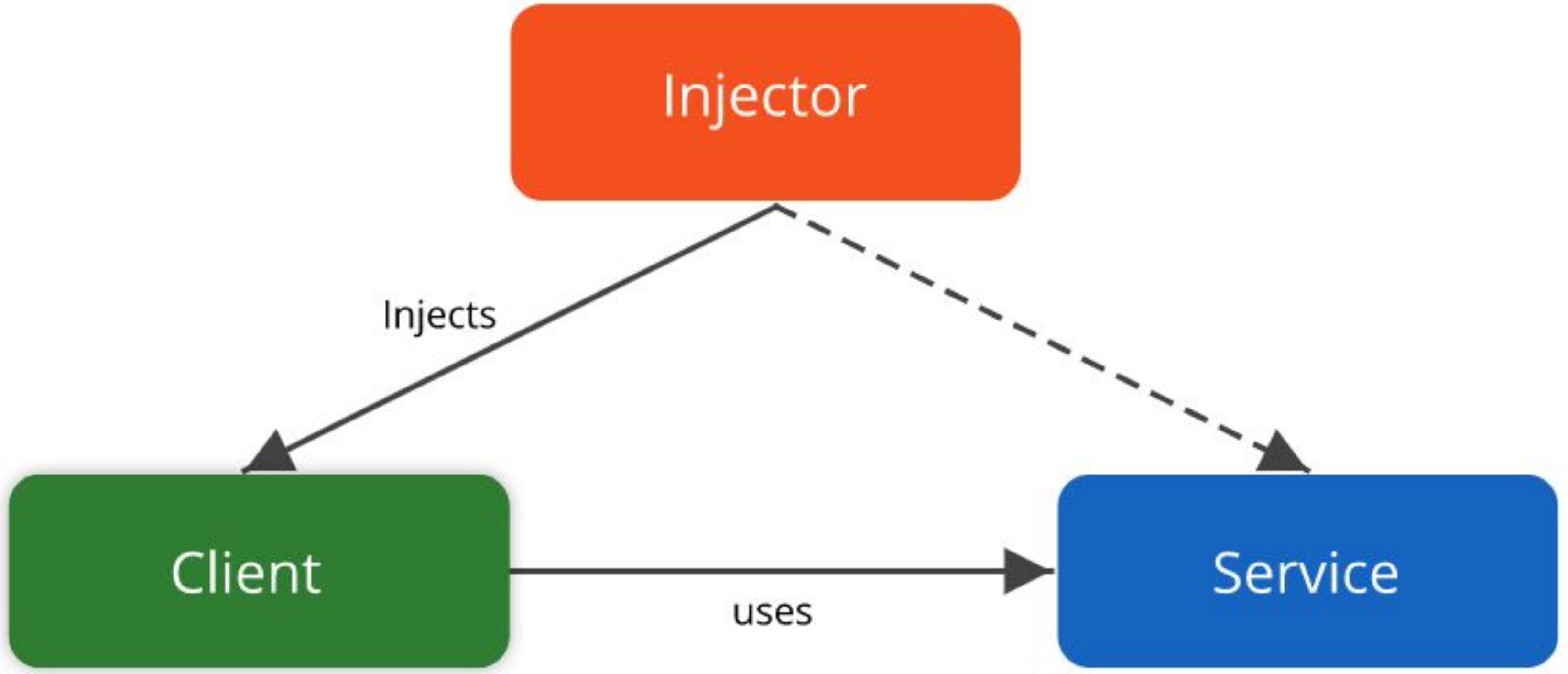
Dependency Injection

DI is a design pattern in which a object receives its dependencies (other objects) as input during its runtime construction. There are 3 main components as part of this architecture/pattern

- Client - Object under construction
- Service - The Dependency
- Injector - Dependency provider

More reading: https://www.martinfowler.com/bliki/InversionOfControl.html
https://www.martinfowler.com/articles/injection.html

Dependency Injection (DI)



Spring Framework

- Open source Application framework and IoC container for Java application
- Contains core modules along with extensions for building JavaEE applications
- Developed and released by Rod Johnson in 2002

Over the course of future releases, spring framework was extended to support multiple functions for Java/JavaEE development.
Available modules include:

- Spring Core - Core container and lifecycle manager
- Aspect oriented Programming
- Authentication and authorization
- Data Access - Repositories and Abstractions for wide range of databases (SQL and NoSQL)
- Transaction management
- Messaging - Abstract templates for asynchronous messaging functionality
- MVC Support - For building web applications
- JMX - Remote Configuration and management

Spring Framework - Key Terms

Spring Context
IoC container responsible for instantiating, configuring and assembling beans based on configuration metadata.

Spring Bean
An Object that is assembled, instantiated and managed by the container. Every bean has a scope defined that is managed by the container.

Spring Framework - Containers

BeanFactory
Interface that provides a simple yet flexible means to configure and manage beans via the Spring IoC container.

- ***BeanFactory*** holds bean definitions and instantiates them whenever asked for by the client application.
- It takes care of the lifecycle of a bean by instantiating it and calling appropriate destruction methods
- It is capable of creating associations between dependent object while instantiating them
- It is important to point that BeanFactory does not support the Annotation-based dependency Injection whereas ***ApplicationContext***, a superset of ***BeanFactory*** does

Spring Framework - Containers

ApplicationContext
Sub-interface of ***BeanFactory*** which supports additional enterprise specific functionality along with the functionality supported by ***BeanFactory***.

Features include:

- Publishing events
- Provide access to resources such as URLs and files
- Resolving messages and support internationalization
- Application-layer specific contexts - hierarchical contexts focussed on specific application layer

Spring Framework - DI Types

Constructor - Dependency is injected via a constructor argument. This method enforces the immutability principle

Setter method - Dependency is injected via setXYZ method, where the argument type is equal to the reference property being injected. This approach makes the bean mutable to further changes

Property/Field Based - Dependency is directly injected(using @Autowired) into the object properties. This is only possible with annotations.

Spring Framework - Demo

BeanFactory

➔ Creating and initializing a container using XML
➔ Initializing beans using the <bean> tag and constructor and setter method injections
➔ Injecting another bean while creating a new bean

ApplicationContext

➔ Initializing the ApplicationContext
➔ Enabling annotation scanning
➔ Using the **@Component** annotation to create a bean
➔ Injecting bean using @Autowired
➔ Resolving ambiguous bean types using @Qualifier and @Primary annotations
➔ Using @Configuration annotation
➔ Listening to ApplicationContext events

Profiles

➔ Using @Profile annotation

Thank You