Spring IoC Container

LAHIRU
LIYANAPATHIRANA

What is Inversion Of Control?

Inversion of Control (IoC) is a software design principle that transfers the control of an application's flow from the application code to an external source, such as a framework.

What is Dependency Injection?

Dependency injection is a design pattern used to implement IoC by passing necessary dependencies to a class as arguments, usually through its constructor.

This allows for flexibility in choosing which implementation to use at runtime.

What is IoC Container?

The Spring IoC (Inversion of Control) Container is the core component of the Spring Framework.

It is responsible for instantiating, configuring, and managing the life cycle of the Objects (Beans) in Spring.

The container uses Dependency Injection to provide necessary dependencies to the beans.

Benefits of IoC Container

- Reduced Coupling: Loosely coupled components make the code more maintainable and testable.
- Simplified Development: The container handles much of the boilerplate code, and allows to focus on business logic.
- Flexibility: The container supports multiple configuration options
- Improved Testability: With dependency injection, mocking dependencies for unit testing is easier.

Types of IoC Containers

Spring framework offers two major implementations for bean management.

They are:

- BeanFactory
- ApplicationContext

Even though they both serve as IoC containers, they significantly differ in functionality and use cases.

BeanFactory

BeanFactory provides the configuration framework for dependecy injection and basic functionality.

ApplicationContext

ApplicationContext extends BeanFactory and adds more enterprise-specific functionalities like:

- Event propagation
- Internationalization (i18n)
- AOP support

ApplicationContext is a complete superset of the BeanFactory.

BeanFactory Vs ApplicationContext (1/3)

Loading Strategy	
BeanFactory	**ApplicationContext**
Loads beans on demand	By default, loads beans on startup

Resource Management	
BeanFactory	**ApplicationContext**
Doesn't provide access to resources and resource management.	Provides access to resources, (URLs, files) through ResourceLoader interface.

BeanFactory Vs ApplicationContext (2/3)

Event Publishing	
BeanFactory	**ApplicationContext**
Does not support event publishing.	Supports event publishing through the ApplicationEvent class.

Auto-Scanning and Annotations	
BeanFactory	**ApplicationContext**
Limited support for annotation-based configuration.	Full support for annotations and component scanning.

BeanFactory Vs ApplicationContext (3/3)

Scope Support	
BeanFactory	**ApplicationContext**
Supports only singleton and prototype scopes	Supports all bean scopes, including request, session, and global session scopes.

BeanFactory Usecases

Basic Functionality

Useful when basic DI features are required without the extra overhead of advanced functionalities provided by the ApplicationContext.

Memory-Constrained Applications

Useful in scenarios where memory usage is critical (only loads beans when needed).

Suitable for lightweight applications with limited resources.

BeanFactory Usecases

Testing Environments

Ideal for unit tests or simple applications where minimal configuration is required.

Allows for quick setup and teardown of test scenarios.

ApplicationContext Usecases

Enterprise Applications

Ideal for most enterprise-level applications due to its advanced feature set.

Provides support for complex configurations and multiple bean definitions.

Web Applications

Applications that work with servlets, the WebApplicationContext is a specialized version of ApplicationContext designed to work with servlets.

ApplicationContext Usecases

Event Handling

Applications that require support for publishing and listening for application events.

AOP Integration

Applications that require integration with Spring AOP features for cross-cutting concerns like logging or security.

Internationalization

Applications that need support for multiple languages and localization.

IoC Container Do's And Don'ts

Configuration

✅ Do

- Simplify configuration
- Use Java-based configuration for larger applications
- Break configurations into modules
- Follow consistent naming patterns

❌ Don't

- Mix different configuration styles (Java or XML)
- Create monolithic config files

IoC Container Do's And Don'ts

Architecture

✅ Do:

- Keep design simple
- Review code regularly
- Document architectural decisions

❌ Don't:

- Over-engineer solutions
- Add unnecessary complexity
- Skip documentation

IoC Container Do's And Don'ts

Dependencies

✅ Do:

- Use constructor injection
- Design with interfaces
- Keep dependencies minimal

❌ Don't:

- Create circular dependencies
- Tightly couple components
- Overuse autowiring

IoC Container Do's And Don'ts

Performance

✅ Do:

- Use lazy initialization when appropriate
- Monitor resource usage
- Clean up unused resources

❌ Don't:

- Ignore memory leaks
- Initialize all beans eagerly which are not used immediately
- Skip performance testing

IoC Container Do's And Don'ts

Testing and Debugging

✅ Do:

- Write unit tests
- Mock dependencies
- Test different configurations
- Add proper exception handling

❌ Don't:

- Skip integration tests
- Ignore edge cases
- Skip error handling
- Ignore warning signs

Did You Find This Post Useful?

Stay Tuned for More Spring Related Posts Like This

LAHIRU LIYANAPATHIRANA