Spring Framework Overview

Version 5.2.8.BUILD-SNAPSHOT

Table of Contents

1. What We Mean by "Spring" 2
2. History of Spring and the Spring Framework 3
3. Design Philosophy 4
4. Feedback and Contributions 5
5. Getting Started 6

Spring makes it easy to create Java enterprise applications. It provides everything you need to embrace the Java language in an enterprise environment, with support for Groovy and Kotlin as alternative languages on the JVM, and with the flexibility to create many kinds of architectures depending on an application's needs. As of Spring Framework 5.1, Spring requires JDK 8+ (Java SE 8+) and provides out-of-the-box support for JDK 11 LTS. Java SE 8 update 60 is suggested as the minimum patch release for Java 8, but it is generally recommended to use a recent patch release.

Spring supports a wide range of application scenarios. In a large enterprise, applications often exist for a long time and have to run on a JDK and application server whose upgrade cycle is beyond developer control. Others may run as a single jar with the server embedded, possibly in a cloud environment. Yet others may be standalone applications (such as batch or integration workloads) that do not need a server.

Spring is open source. It has a large and active community that provides continuous feedback based on a diverse range of real-world use cases. This has helped Spring to successfully evolve over a very long time.

Chapter 1. What We Mean by "Spring"

The term "Spring" means different things in different contexts. It can be used to refer to the Spring Framework project itself, which is where it all started. Over time, other Spring projects have been built on top of the Spring Framework. Most often, when people say "Spring", they mean the entire family of projects. This reference documentation focuses on the foundation: the Spring Framework itself.

The Spring Framework is divided into modules. Applications can choose which modules they need. At the heart are the modules of the core container, including a configuration model and a dependency injection mechanism. Beyond that, the Spring Framework provides foundational support for different application architectures, including messaging, transactional data and persistence, and web. It also includes the Servlet-based Spring MVC web framework and, in parallel, the Spring WebFlux reactive web framework.

A note about modules: Spring's framework jars allow for deployment to JDK 9's module path ("Jigsaw"). For use in Jigsaw-enabled applications, the Spring Framework 5 jars come with "Automatic-Module-Name" manifest entries which define stable language-level module names ("spring.core", "spring.context" etc) independent from jar artifact names (the jars follow the same naming pattern with "-" instead of ".", e.g. "spring-core" and "spring-context"). Of course, Spring's framework jars keep working fine on the classpath on both JDK 8 and 9+.

Chapter 2. History of Spring and the Spring Framework

Spring came into being in 2003 as a response to the complexity of the early J2EE specifications. While some consider Java EE and Spring to be in competition, Spring is, in fact, complementary to Java EE. The Spring programming model does not embrace the Java EE platform specification; rather, it integrates with carefully selected individual specifications from the EE umbrella:

- Servlet API (JSR 340)
- WebSocket API (JSR 356)
- Concurrency Utilities (JSR 236)
- JSON Binding API (JSR 367)
- Bean Validation (JSR 303)
- JPA (JSR 338)
- JMS (JSR 914)
- as well as JTA/JCA setups for transaction coordination, if necessary.

The Spring Framework also supports the Dependency Injection (JSR 330) and Common Annotations (JSR 250) specifications, which application developers may choose to use instead of the Spring-specific mechanisms provided by the Spring Framework.

As of Spring Framework 5.0, Spring requires the Java EE 7 level (e.g. Servlet 3.1+, JPA 2.1+) as a minimum - while at the same time providing out-of-the-box integration with newer APIs at the Java EE 8 level (e.g. Servlet 4.0, JSON Binding API) when encountered at runtime. This keeps Spring fully compatible with e.g. Tomcat 8 and 9, WebSphere 9, and JBoss EAP 7.

Over time, the role of Java EE in application development has evolved. In the early days of Java EE and Spring, applications were created to be deployed to an application server. Today, with the help of Spring Boot, applications are created in a devops- and cloud-friendly way, with the Servlet container embedded and trivial to change. As of Spring Framework 5, a WebFlux application does not even use the Servlet API directly and can run on servers (such as Netty) that are not Servlet containers.

Spring continues to innovate and to evolve. Beyond the Spring Framework, there are other projects, such as Spring Boot, Spring Security, Spring Data, Spring Cloud, Spring Batch, among others. It's important to remember that each project has its own source code repository, issue tracker, and release cadence. See spring.io/projects for the complete list of Spring projects.

Chapter 3. Design Philosophy

When you learn about a framework, it's important to know not only what it does but what principles it follows. Here are the guiding principles of the Spring Framework:

- Provide choice at every level. Spring lets you defer design decisions as late as possible. For example, you can switch persistence providers through configuration without changing your code. The same is true for many other infrastructure concerns and integration with third-party APIs.
- Accommodate diverse perspectives. Spring embraces flexibility and is not opinionated about how things should be done. It supports a wide range of application needs with different perspectives.
- Maintain strong backward compatibility. Spring's evolution has been carefully managed to force few breaking changes between versions. Spring supports a carefully chosen range of JDK versions and third-party libraries to facilitate maintenance of applications and libraries that depend on Spring.
- Care about API design. The Spring team puts a lot of thought and time into making APIs that are intuitive and that hold up across many versions and many years.
- Set high standards for code quality. The Spring Framework puts a strong emphasis on meaningful, current, and accurate javadoc. It is one of very few projects that can claim clean code structure with no circular dependencies between packages.

Chapter 4. Feedback and Contributions

For how-to questions or diagnosing or debugging issues, we suggest using Stack Overflow. Click here for a list of the suggested tags to use on Stack Overflow. If you're fairly certain that there is a problem in the Spring Framework or would like to suggest a feature, please use the GitHub Issues.

If you have a solution in mind or a suggested fix, you can submit a pull request on Github. However, please keep in mind that, for all but the most trivial issues, we expect a ticket to be filed in the issue tracker, where discussions take place and leave a record for future reference.

For more details see the guidelines at the CONTRIBUTING, top-level project page.

Chapter 5. Getting Started

If you are just getting started with Spring, you may want to begin using the Spring Framework by creating a Spring Boot-based application. Spring Boot provides a quick (and opinionated) way to create a production-ready Spring-based application. It is based on the Spring Framework, favors convention over configuration, and is designed to get you up and running as quickly as possible.

You can use start.spring.io to generate a basic project or follow one of the "Getting Started" guides, such as Getting Started Building a RESTful Web Service. As well as being easier to digest, these guides are very task focused, and most of them are based on Spring Boot. They also cover other projects from the Spring portfolio that you might want to consider when solving a particular problem.