Spring Framework 6

Baraneetharan Ramasamy

Base JSR used in Spring Framework:

In the Spring Framework, the **Base JSRs** refer to the Java EE specifications that Spring integrates with. Here are some of the key ones:

1. **Servlet API (JSR 340):** Provides the foundation for web applications by defining the servlet container and request/response handling.
2. **WebSocket API (JSR 356):** Enables real-time communication between clients and servers using WebSockets.
3. **Concurrency Utilities (JSR 236):** Offers abstractions for managing concurrent tasks and scheduling.
4. **JSON Binding API (JSR 367):** Defines standard annotations for mapping Java objects to JSON and vice versa.
5. **Bean Validation (JSR 303):** Provides validation rules for Java beans using annotations.
6. **JPA (JSR 338):** Java Persistence API for working with databases and ORM (Object-Relational Mapping).
7. **JMS (JSR 914):** Java Messaging Service for asynchronous communication between distributed components.

These specifications allow Spring applications to leverage standardized features while maintaining flexibility and ease of use

Spring support for Web, Reactive Stacks:

1. **Spring Web:**
    - **Spring MVC (Model-View-Controller):** A traditional web framework that follows the MVC pattern. It's suitable for building web applications with synchronous request-response cycles.
    - **Annotations:** Spring MVC uses annotations like `@Controller`, `@RequestMapping`, and `@ResponseBody` to define controllers, request mappings, and response handling.
    - **View Technologies:** Supports various view technologies such as JSP, Thymeleaf, and FreeMarker.
    - **Filters and Interceptors:** Provides filters and interceptors for pre- and post-processing of requests and responses.
    - **RESTful APIs:** Spring MVC is commonly used for building RESTful APIs.
2. **Spring WebFlux (Reactive Stack):**
    - **Reactive Programming:** WebFlux embraces reactive programming principles, allowing applications to handle a large number of concurrent requests efficiently.
    - **Non-Blocking I/O:** WebFlux uses Netty or Servlet 3.1+ containers to achieve non-blocking I/O.
    - **Functional Endpoints:** You can define routes and handlers using a functional programming style.
    - **Flux and Mono:** WebFlux uses `Flux` (for multiple items) and `Mono` (for single items) to represent reactive streams.
    - **Annotations:** Annotations like `@RestController` and `@GetMapping` are used to create reactive endpoints.
    - **WebClient:** Provides a reactive client for making HTTP requests to other services.
3. **Choosing Between Spring Web and WebFlux:**
    - **Use Spring Web (MVC) When:**

    - You have a traditional web application with synchronous processing.
    - You're comfortable with the familiar MVC pattern.
    - Your application doesn't require high concurrency.
- **Use Spring WebFlux When:**
    - You need to handle a large number of concurrent connections.
    - You want to embrace reactive programming.
    - Your use case involves streaming data or real-time communication.

> Remember that both Spring Web and WebFlux have their strengths, and the choice depends on your specific requirements and familiarity with reactive programming.

Jakarta EE versions along with their corresponding base Java SE versions:

Jakarta EE Version	Base Java SE Version
Jakarta EE 10	Java SE 11
Jakarta EE 9.1	Java SE 11
Jakarta EE 9	Java SE 8
Jakarta EE 8	Java SE 8

Spring Framework versions:

Spring Framework Version	Jakarta EE Version	Web Server Version
6.1.x	9-11 (jakarta)	Tomcat 10, Jetty 11, WildFly 24, Payara 6
6.0.x	9-10 (jakarta)	Tomcat 10, Jetty 11, WildFly 23, Payara 5
5.3.x	7-8 (javax)	Tomcat 9, Jetty 9, WildFly 22, Payara 4
4.x	7 (javax)	Tomcat 8, Jetty 9, WildFly 10, Payara 4
3.x	6 (javax)	Tomcat 7, Jetty 8, WildFly 8, Payara 3

Spring Framework 3 and 4 are older versions, but they are still used in some legacy projects. Keep in mind that Spring Boot 3.0, which is based on Spring Framework 6.0, is planned for release in November 2022 and will require Java 17 or above, using Jakarta EE 9 APIs. Let me know if you need further assistance! 😊

Spring Framework 6 and its alignment with Java 21:

Spring Framework 6.1, released on November 16, 2023, aligns with **Java 21**. Here are the key points:

1. **Java 21 Features:**
    - **Pattern Matching:** Java 21 introduces pattern matching, making code more concise and expressive.
    - **Virtual Threads (Project Loom):** Virtual threads enable lightweight concurrency, improving scalability.
    - **Multiline Strings:** Java 21 supports multiline strings, enhancing readability for JSON, SQL, and more.
2. **Spring Framework 6.1:**
    - **Java 21 Support:** Spring Framework 6.1 runs on Java 21, leveraging its features.
    - **Virtual Threads:** Spring 6.1 simplifies concurrent programming using virtual threads.
    - **Reactive Programming:** Enhancements for reactive programming and Kotlin coroutines.
    - **GraalVM Compatibility:** Compatible with GraalVM for JDK 21 while tracking GraalVM 22.3.

In summary, Spring Framework 6.1 embraces Java 21's capabilities, making it an exciting release for JVM developers.

What's New in Spring Framework 6.x:

> https://github.com/spring-projects/spring-framework/wiki/What's-New-in-Spring-Framework-6.x
>
> Introduction to OpenRewrite | OpenRewrite by Moderne

Spring Boot 2.x to Spring Boot 3.x

Migrating from **Spring Boot 2.x** to **Spring Boot 3.x** can indeed be challenging due to several factors:

1. **Core Changes**:
   - **Spring Boot 3.0** introduces significant modifications to its core components. Some of these changes include:
     - Property key modifications (e.g., `spring.redis` to `spring.data.redis`).
     - Jakarta EE 10 updates (e.g., Servlet specification version 6.0, JPA specification version 3.1).
     - Hibernate version updates (e.g., Hibernate Core 6.1.4.Final).
   - These changes require careful adjustments in your application configuration and code1.
2. **Compatibility Challenges**:
   - Spring Boot 3.0 mandates **Java 17** as the minimum version. If your existing codebase relies on older Java versions, you'll need to update it.
   - Some third-party dependencies may not be compatible with Spring Boot 3.x, leading to additional migration efforts.
   - Mixing Spring Boot 2.x and 3.x applications in the same pipeline can cause issues due to different Spring Cloud Stream binder versions2.
3. **Documentation and Migration Guides**:
   - The Spring Boot team provides migration guides, but understanding and applying the changes can be time-consuming.
   - Upgrading Spring Security from 5.8 to 6.0 is a prerequisite for Spring Boot 3.0 migration3.
   - Deprecated APIs and package changes (from `javax` to `jakarta`) require code adjustments.

Despite the challenges, proper planning, testing, and gradual migration can help mitigate the pain.