Deep Dive into Jakarta EE 10

Baraneetharan Ramasamy

What is JSR, JCP, JEP, TCK:

1. **JSR (Java Specification Request)**:
   - A JSR is a formal proposal submitted to the **Java Community Process (JCP)** for defining new features, APIs, or modifications to existing ones in the Java platform.
   - Each JSR focuses on a specific area (e.g., language enhancements, APIs, or platform improvements).
   - The **expert group** associated with a JSR collaborates to create the specification and reference implementation.
   - Example: JSR 379 was the umbrella JSR for Java SE 9, defining features like modules and enhancements.
2. **JCP (Java Community Process)**:
   - The JCP is the governing body responsible for managing the evolution of Java technologies.
   - It oversees the creation and approval of JSRs.
   - The JCP Executive Committee (EC) approves JSRs and ensures compatibility across implementations.
   - The JCP fosters collaboration among vendors, developers, and the community.
   - Example: The JCP ensures that Java evolves in a standardized and compatible manner.
3. **JEP (Java Enhancement Proposal)**:
   - JEPs propose enhancements to the Java platform, including language features, APIs, and runtime improvements.
   - Unlike JSRs, JEPs are more lightweight and focus on specific technical changes.
   - JEPs are part of the OpenJDK development process.
   - Example: JEP 355 introduced the “text blocks” feature in Java 13.
4. **TCK (Technology Compatibility Kit)**:

- The TCK ensures that Java implementations (e.g., JDKs) conform to the specifications defined by JSRs.
- It includes a set of tests that implementations must pass to be considered compatible.
- TCKs are crucial for maintaining compatibility across different Java platforms.
- Example: The OpenJDK TCK validates that an OpenJDK distribution complies with the Java SE specification.

> JSRs propose changes, the JCP oversees the process, JEPs drive specific enhancements, and TCKs validate compatibility. Together, they form the backbone of Java's evolution and standardization. Feel free to explore further or ask if you have more questions!

Role of Eclipse foundation:

The Eclipse Foundation is home to the Eclipse IDE, Jakarta EE, and hundreds of open source projects, including runtimes, tools, specifications, and frameworks for cloud and edge applications, IoT, AI, automotive, systems engineering, open processor designs, and many others.

https://www.eclipse.org/downloads/

The **Eclipse Foundation** serves as a crucial platform that drives open-source software collaboration and innovation. Its primary purpose is to facilitate the development, promotion, and distribution of a growing collection of open-source software frameworks, tools, and runtimes. Here are some key points about the Eclipse Foundation:

1. **Stewardship and Governance**:
   - The Eclipse Foundation is an independent, Europe-based not-for-profit corporation.
   - It acts as a steward of the **Eclipse open source software development community**.
   - Legal jurisdiction lies within the European Union.
   - The Foundation ensures transparent, vendor-neutral collaboration.
2. **Services Provided**:
   - **Intellectual Property (IP) Management**: Ensures legal compliance and IP cleanliness for Eclipse projects.
   - **Ecosystem Development**: Fosters collaboration among developers, organizations, and communities.
   - **IT Infrastructure**: Provides tools, hosting, and resources for project development.
3. **Project Landscape**:
   - Home to over **425 open-source projects**, spanning various domains (e.g., IoT, cloud, automotive).
   - Notable projects include **Eclipse IDE**, widely used for Java development.

- More than **90% of its codebase** is written in Java.

4. **Membership**:
   - Four types: **Strategic, Contributing, Associate, and Committer**.
   - Strategic Members (e.g., European Space Agency, Microsoft, Oracle) invest in Eclipse technology.
   - Contributing Members participate in ecosystem development.
5. **Events and Collaborations**:
   - Hosts **DemoCamps, Hackathons, and conferences** (e.g., EclipseCon).
   - Engages in **22 Industry Collaborations** (e.g., IoT, scientific research).

Jakarta EE Specifications:

https://jakarta.ee/about/jakarta-ee/

1. **What Are Jakarta EE Specifications?**
   - **Previously Known as Java EE**: Jakarta EE specifications were formerly known as Java EE specifications.
   - **Developed by Industry Leaders**: These specifications are developed by well-known industry leaders following the **Jakarta EE Specification Process (JESP)**.
   - **Components of Each Specification**:
     - **Specification Document**: Describes how the technology is implemented.
     - **Javadoc (APIs)**: Provides detailed information about the APIs associated with the specification.
2. **Key Points**:
   - **Jakarta EE Platform and Profile Specifications**: These umbrella specifications cover individual specifications within Jakarta EE.
   - **Technology Compatibility Kit (TCK)**: Ensures compatibility across implementations.
   - **Compatible Implementations**: Certified products that adhere to Jakarta EE specifications.

> Jakarta EE specifications drive the evolution of enterprise Java, ensuring standardized APIs and compatibility across implementations.

Adoptium JDK & Temurin:

Java™ is the world's leading programming language and platform. The Adoptium Working Group promotes and supports high-quality, TCK certified runtimes and associated technology for use across the Java ecosystem. Eclipse Temurin is the name of the OpenJDK distribution from Adoptium.

https://adoptium.net/en-GB/

Adoptium JDK and **Eclipse Temurin**:

1. **Adoptium JDK**:
   - **Goal**: The **Adoptium Working Group** aims to promote and support high-quality, **TCK-certified runtimes** (Java implementations) for use across the Java ecosystem.
   - **Binaries**: Adoptium provides **OpenJDK-based binaries** under the **Eclipse Temurin project**.
   - **Certification**: These runtimes undergo **Java SE TCK (Technology Compatibility Kit)** testing to ensure compatibility.
   - **Name**: Eclipse Temurin is the name of the **OpenJDK distribution** from Adoptium.
2. **Eclipse Temurin**:
   - **Purpose**: The Eclipse Temurin project builds **cross-platform**, **high-performance**, and **enterprise-caliber** runtime binaries.
   - **Open Source**: It's open-source licensed and based upon **OpenJDK**.
   - **General Use**: These binaries are suitable for general use across the Java ecosystem.
   - **Regular Updates**: Eclipse Temurin releases are **regularly updated** and supported by the Adoptium community.

TCK (Technology Compatibility Kit) testing plays a crucial role in ensuring that Java runtimes conform to the **Java SE specification**. Here are the key points:

1. **What Is TCK Testing?**
   - The TCK verifies that a Java distribution adheres to the Java SE specification.
   - It contains **over 100,000 individual tests**, covering APIs, libraries, language features, and other components.
   - TCK compliance establishes technology credibility, interoperability, and standardization.
   - Passing TCK tests allows a provider to call their implementation "Java" or "Java-SE compliant."
2. **What TCK Testing Does NOT Cover**:
   - TCK does **not** test performance (e.g., throughput, startup time, latency).
   - It ensures functionality but doesn't assess how efficiently it's done.
   - TCK results won't reveal differences in performance between distributions.
3. **Threshold for Passing TCK Tests**:
   - A distribution must pass **ALL** TCK tests to prove Java SE specification conformance.
   - For example, Azul Platform Prime passes all TCK tests, ensuring compatibility.

> TCK compliance is essential for standardization and legal/business requirements.

Web, Reactive Stacks in Jakarta EE:

HTML 5 - The first working draft of HTML 5 was published in **January 2008**. Adoption of HTML 5 was gradual between 2008 and 2012, and then steady. HTML 5 was finally issued as a W3C Recommendation in 2014.

A **Jakarta Servlet** is a Java class in Jakarta EE that conforms to the Jakarta Servlet API, a standard for implementing Java classes that respond to requests. Servlets could in principle communicate over any client–server protocol, but they are most often used with HTTP. In principle, any servlets can extend the `GenericServlet` class; however, realistically speaking, all servlets extend the `HttpServlet` class. Thus "servlet" is often used as shorthand for "HTTP servlet". Thus, a servlet can be used to add dynamic content to a web server using the Java platform. The generated content is commonly HTML, but may be other data such as XML and more commonly, JSON.

Servlet API History

Servlet API version	Released	Specification	Platform	Important Changes
Jakarta Servlet 6.0	May 31, 2022	6.0	Jakarta EE 10	remove deprecated features and implement requested enhancements
Jakarta Servlet 5.0	Oct 9, 2020	5.0	Jakarta EE 9	API moved from package `javax.servlet` to `jakarta.servlet`
Jakarta Servlet 4.0.3	Sep 10, 2019	4.0	Jakarta EE 8	Renamed from "Java" trademark
Java Servlet 4.0	Sep 2017	JSR 369	Java EE 8	HTTP/2
Java Servlet 3.1	May 2013	JSR 340	Java EE 7	Non-blocking I/O, HTTP protocol upgrade mechanism (WebSocket)[19]
Java Servlet 3.0	December 2009	JSR 315	Java EE 6, Java SE 6	Pluggability, Ease of development, Async Servlet, Security, File Uploading
Java Servlet 2.5	September 2005	JSR 154	Java EE 5, Java SE 5	Requires Java SE 5, supports annotation
Java Servlet 2.4	November 2003	JSR 154	J2EE 1.4, J2SE 1.3	web.xml uses XML Schema
Java Servlet 2.3	August 2001	JSR 53	J2EE 1.3, J2SE 1.2	Addition of `Filter`
Java Servlet 2.2	August 1999	JSR 902, JSR 903	J2EE 1.2, J2SE 1.2	Becomes part of J2EE, introduced independent web applications in .war files
Java Servlet 2.1	November 1998	2.1a	Unspecified	First official specification, added `RequestDispatcher`, `ServletContext`
Java Servlet 2.0	December 1997	—	JDK 1.1	Part of April 1998 Java Servlet Development Kit 2.0[20]
Java Servlet 1.0	December 1996	—		Part of June 1997 Java Servlet Development Kit (JSDK) 1.0[14]

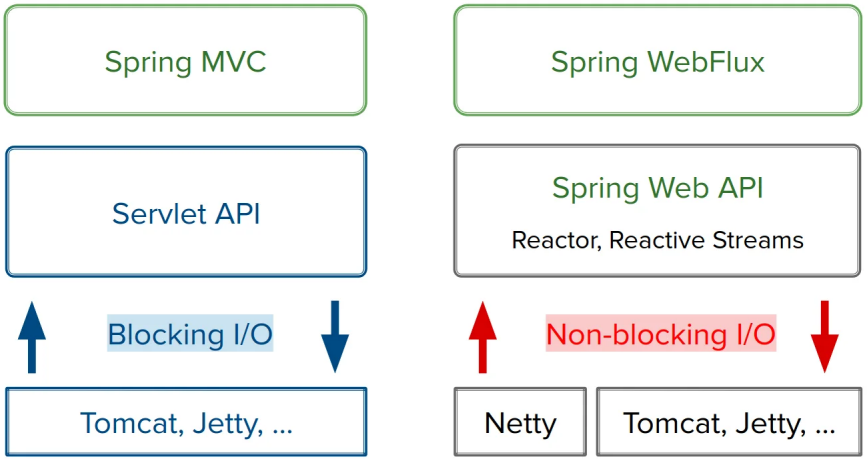
The servlet stack - blocking I/O

The servlet stack is a classic servlet container and Servlet API as the web framework. In the beginning the Servlet API was built on a "thread-per-request" model, i.e. making a full pass through the filter-servlet chain on a single thread, blocking along the way as needed.

The Reactive stack - non-blocking I/O

Servlet 3.1, released as part of Java EE 7, introduced a key feature: **non-blocking I/O**. Prior to this release, **Async Servlets** allowed asynchronous request processing, but only with traditional blocking I/O. This limitation could impact the scalability of applications.

Below is a diagram of the two stacks side by side:



Reactive Streams is an integration API that facilitates asynchronous streaming between different libraries, connectors, and data sources. It ensures predictable behavior for backpressure, completion, cancellation, and error handling. Here are some key points:

1. **Java Flow API**: Reactive Streams was adopted by the JDK in the form of the `java.util.concurrent.Flow` API. This API allows asynchronous streaming libraries to connect with each other seamlessly.
2. **Rich Ecosystem**: There's a rich ecosystem of open-source libraries supporting Reactive Streams. Since its inclusion in JDK 9, several implementations have targeted the JDK, including the incubating JDK 9 HTTP Client and the Asynchronous Database Adapter (ADBA).
3. **Integration Benefits**: The primary reason for supporting Reactive Streams in Jakarta EE is twofold:
    - **Connecting APIs**: It simplifies connecting different streaming APIs within Jakarta EE. For example, you can easily link CDI events to a WebSocket or stream data from an HTTP client response to a servlet response.
    - **Third-Party Libraries**: It also facilitates integration with third-party libraries, making it easier to connect Jakarta EE APIs with external components.
4. **Example Scenario**: Imagine subscribing to a queue using a Reactive Streams-compatible messaging API in Jakarta EE. You could use Akka Streams to handle the stream and save message content asynchronously using a database API.

While support for Java 9 reactive streams in Jakarta EE hasn't been explicitly planned, the transition from Java EE to Jakarta EE may eventually include the Reactive Streams API. Additionally, Jakarta EE aims to accelerate cloud-native Java by adding new reactive and stream-based messaging APIs.

> https://www.slideshare.net/slideshow/stoyanchev-servletvsreactivestacks/77275448
>
> https://docs.spring.io/spring-boot/reference/web/index.html

Understanding the shift from Java EE to Jakarta EE:

The transition from **Java EE** to **Jakarta EE** marks a pivotal moment in the history of enterprise Java. Let's explore why it's important:

1. **Origins and Recognition**:
   - **Java EE** (formerly known as J2EE) kick-started the industry for using Java on the server. It's recognized for its Java Servlet specification and servlet containers like Tomcat and Jetty.
   - Over the years, Java EE expanded to include specifications for persistence (Java Persistence API [JPA]), REST (JAX-RS), WebSocket, and more.
2. **Java EE 8 and Transfer to Eclipse Foundation**:
   - The last official release of **Java EE** was **Java EE 8** in August 2017.
   - Oracle decided to transfer Java EE fully to an open-source foundation, specifically the **Eclipse Foundation**.
   - This transfer involved moving the API, implementation code, and the Technology Compatibility Kit (TCK) to Eclipse.
3. **Why the Transition?**:
   - The shift to **Jakarta EE** reflects a broader industry trend toward community-driven development and open standards.
   - It ensures a more dynamic and responsive future for enterprise Java.
   - **Jakarta EE** aims to align with contemporary development practices, especially in the era of microservices and cloud-native applications.
4. **Key Differences**:
   - Every API in **Jakarta EE** must transition from `javax` to `jakarta`.
   - Specifications need updates to reflect the API changes.
   - Implementations must adjust to handle the new APIs.

The move to **Jakarta EE** signifies more than just a name change—it sets the stage for a robust and community-driven future for enterprise Java.

> https://blogs.oracle.com/javamagazine/post/transition-from-java-ee-to-jakarta-ee?s=21

New features in Jakarta EE 10 for cloud-native Java applications:

Jakarta EE 10 brings exciting enhancements for building modernized, simplified, and lightweight cloud-native Java applications. Here are some key features:

1. **Jakarta Contexts and Dependency Injection (CDI) 4.0**:
   - Includes CDI-Lite, enabling build-time extensions.
2. **Jakarta Security 3.0**:
   - Supports OpenID Connect for secure authentication.
3. **Jakarta Servlet 6.0**:
   - Simplifies programming and improves security.
4. **Jakarta Faces (JSF) 4.0**:
   - Modernizes the API using CDI.

5. **Jakarta JSON Binding (JSON-B) 3.0**:
    - Adds new support for polymorphic types.
6. **Jakarta RESTful Web Services**:
    - Standardizes a Java SE Bootstrap API and multipart/form-data support.
7. **Jakarta Persistence**:
    - Standardizes UUID as a Basic Type and extends the Query language and Query API.
8. **Jakarta Concurrency 3.0**:
    - Moved to the Web Profile, enhancing parallel and reactive programming models.

Breaking changes between **Jakarta EE 9** and **Jakarta EE 10**:

1. **Namespace Changes**:
    - Jakarta EE 10 uses the `jakarta.*` namespace instead of the `javax.*` namespace for APIs. This transition ensures compatibility with the Eclipse Foundation's stewardship.
    - Developers need to update their imports and dependencies to use the new namespace.
2. **Removed APIs**:
    - Some APIs deprecated in Jakarta EE 9 have been removed in version 10. For example, the `javax.xml.registry` API is no longer available.
    - Developers should review their codebase and replace deprecated APIs with their Jakarta EE equivalents.
3. **Java SE Version Requirement**:
    - Jakarta EE 10 requires Java SE 11 or later. Applications targeting Jakarta EE 10 must run on a compatible JDK.
    - Developers upgrading from older versions need to ensure their environment meets this requirement.
4. **Servlet API Changes**:
    - The Servlet API has evolved. Developers should review their servlet-based applications for any adjustments needed due to changes in Servlet 5.0.
5. **JSP Removal**:
    - Jakarta EE 10 removes support for JavaServer Pages (JSP). Developers relying on JSP should migrate to other view technologies like Jakarta Server Faces (JSF) or Thymeleaf.
6. **Security Enhancements**:
    - Jakarta Security 3.0 introduces improvements, but some features may require adjustments in existing security configurations.