JEE5 Features

Momentum for organizations to adopt Java™ Platform, Enterprise Edition 5 (Java EE 5) is steadily increasing as the platform's container support, developer tools, educational resources, and developer-community experience all improve. Get a high-level view of the new productivity features and API enhancements in Java EE 5 and work through a Web service example that demonstrates its simplified development models.

Introduction

Java EE technology is an extension to the Java language platform that enables you to create scalable, powerful, and portable enterprise applications. It defines four types of containers for application components: Web, Enterprise JavaBean (EJB), application client, and applet. These containers, and the Java APIs each must support, are detailed in an application-server specification that encourages a competitive marketplace for Java EE products while guaranteeing server portability for applications that adhere to the specification (see A brief history of Java EE).

The latest version of the platform, Java EE 5, was released in May 2006. Primarily focused on developer productivity, Java EE 5 brings simpler programming models without sacrificing any of the platform's power and richness. Two mechanisms — Java annotations and better defaults — are responsible for most of its simpler development models. Major functional improvements include enhanced Web services support and incorporation of JavaServer Faces (JSF) and the Java Standard Tag Library (JSTL) into the platform.

A brief history of Java EE

Java EE 5 was released by the Java Community Process as Java Specification Request 244, an "umbrella" JSR that refers to other specifications that detail its constituent technologies (see Resources). Specification Lead Bill Shannon of Sun Microsystems shepherded an Expert Group whose 31 members ranged from IT industry heavyweights to individual experts. Java EE 5's predecessors are:

- **J2EE 1.2** (released December 1999): This first J2EE release accompanied the release of Java 2 Standard Edition (J2SE). Its 10 specifications and APIs addressed common Web-tier, business-logic, persistence-layer, and messaging services needed by enterprise applications.
- **J2EE 1.3** (released September 2001): Released as JSR 58, this version improved half of the specifications in J2EE 1.2 and introduced an XML API, connector architecture, and security framework.
- **J2EE 1.4** (released November 2003): This version improved on 9 of the 13 technologies in J2EE 1.3 and introduced new support for Web services and security.

Java EE 5 immediately follows J2EE 1.4 because Sun has dropped "2" from the name (a relic from when Java 1.2 was rebranded as "Java 2"), and now uses the word "Java" rather than "J" in the technologies' shorthand names. The standard edition is now Java SE 6 (rather than J2SE 1.6), and the enterprise edition is Java EE 5 (rather than J2EE 1.5).

This article explores the features in Java EE 5, highlighting what's changed since the last J2EE release. (Enhancements beyond this article's scope include the addition of the StAX API — a pull-parsing API for XML — and small- to medium-sized improvements across many of the APIs.) I'll give you a comprehensive look at four feature categories: Web services, Web application, enterprise, and management and security. Then you'll "kick the tires" a bit, examining a small application that uses Java EE 5 features that demonstrate simpler development models for Web applications that use a service-oriented architecture (SOA).

The article is aimed at readers familiar with enterprise-level software systems who want a comprehensive, high-level view of Java EE 5. For the feature overview, a working knowledge of Java programming and J2EE is helpful but not necessary. The discussion of the example application dives into more-technical details that are probably more relevant to those with a J2EE (or at least Java programming) background.

Web services technologies

The introduction of annotations into Java EE 5 makes it simple to create sophisticated Web service endpoints and clients with less code and a shorter learning curve than was possible with earlier Java EE versions. Annotations — first introduced in Java SE 5 — are modifiers you can add to your code as metadata. They don't affect program semantics directly, but the compiler, development tools, and runtime libraries can process them to produce additional Java language source files, XML documents, or other artifacts and behavior that augment the code containing the annotations (see Resources). Later in the article, you'll see how you can easily turn a regular Java class into a Web service by adding simple annotations.

A leap forward in Web services support

The cornerstone of Web services support in Java EE 5 is JAX-WS 2.0, which is a follow-on to JAX-RPC 1.1. Both of these technologies let you create RESTful and SOAP-based Web services without dealing directly with the tedium of XML processing and data binding inherent to Web services. Developers are free to continue using JAX-RPC (which is still required of Java EE 5 containers), but migrating to JAX-WS is strongly recommended. Newcomers to Java Web services might as well skip JAX-RPC and head right for JAX-WS. That said, it's good to know that both of them support SOAP 1.1 over HTTP 1.1 and so are fully compatible: a JAX-WS Web services client can access a JAX-RPC Web services endpoint, and vice versa.

The advantages of JAX-WS over JAX-RPC are compelling. JAX-WS:

- Supports the SOAP 1.2 standard (in addition to SOAP 1.1).
- Supports XML over HTTP. You can bypass SOAP if you wish. (See the article "Use XML directly over HTTP for Web services (where appropriate)" for more information.)
- Uses the Java Architecture for XML Binding (JAXB) for its data-mapping model. JAXB has complete support for XML schema and better performance (more on that in a moment).
- Introduces a dynamic programming model for both server and client. The client model supports both a message-oriented and an asynchronous approach.
- Supports Message Transmission Optimization Mechanism (MTOM), a W3C recommendation for optimizing the transmission and format of a SOAP message.
- Upgrades Web services interoperability (WS-I) support. (It supports Basic Profile 1.1; JAX-WS supports only Basic Profile 1.0.)
- Upgrades SOAP attachment support. (It uses the SOAP with Attachments API for Java [SAAJ] 1.3; JAX-WS supports only SAAJ 1.2.)

You can learn more about the differences by reading the article "JAX-RPC versus JAX-WS."

The `wsimport` tool in JAX-WS automatically handles many of the mundane details of Web service development and integrates easily into a build processes in a cross-platform manner, freeing you to focus on the application logic that implements or uses a service. It generates artifacts such as services, service endpoint interfaces (SEIs), asynchronous response code, exceptions based on WSDL faults, and Java classes bound to schema types by JAXB.

JAX-WS also enables high-performing Web services. See Resources for a link to an article ("Implementing High Performance Web Services Using JAX-WS 2.0") presenting a benchmark study of equivalent Web service implementations based on the new JAX-WS stack (which uses two other Web services features in Java EE 5 — JAXB and StAX) and a JAX-RPC stack available in J2EE 1.4. The study found 40% to 1000% performance increases with JAX-WS in various functional areas under different loads.

Back to top

Web application technologies

Java EE 5 welcomes two major pieces of front-end technology — JSF and JSTL — into the specification to join the existing JavaServer Pages and Servlet specifications. JSF is a set of APIs that enable a component-based approach to user-interface development. JSTL is a set of tag libraries that support embedding procedural logic, access to JavaBeans, SQL commands, localized formatting instructions, and XML processing in JSPs. The most recent releases of JSF, JSTL, and JSP support a unified expression language (EL) that allows these technologies to integrate more easily (see Resources).

JSF 1.2

JSF has built-in support for common UI concerns such as component state management, event handling, navigation, user-input validation, and internationalization. Expert developers can create customized, powerful, reusable components or create custom renderers for client devices other than a Web browser. Less-technical users can reuse custom components, including the default JSF tag library for HTML interfaces, in visual programming environments such as Sun Java Studio Creator. This puts the creation of sophisticated Web presentation layers within reach of novice programmers.

A growing landscape of third-party JSF components exists in both the open source community and the licensed-software realm. You'll turn up dozens of options by searching for "JSF components" or "JSF component libraries" on the Web. Many of these components rely on Asynchronous JavaScript + XML (Ajax) techniques, which are a driving force behind the "Web 2.0" experience. Web programmers can use them to create applications that offer a better user experience than traditional Web applications, without needing to fuss with the details of writing Ajax components from scratch.

JSP 2.1

JSP technology has been around since J2EE 2.1. It leverages the Java Servlet specification to enable declarative programming of a UI. It supports programming UIs as documents that are translated into Java servlets, compiled, and invoked by the Web application container to service requests. These documents typically mix JSP directives and scriptlets with a presentation markup language such as HTML. JSPs can use an older syntax that relies on special tags that start with <% and end with %>, or on a newer syntax that is well-formed XML. They typically serve as the "View" part of a Model-View-Controller (MVC) UI framework.

JSP 2.1 and JSF 1.2 are more mutually compatible than prior releases, primarily because of the integration of their EL syntax into the unified EL. EL enables operations such as:

- Accessing properties of JavaBeans in the request, session, and application contexts.
- Performing logical tests that determine such choices as whether to hide or show a particular element.
- Performing calculations that affect numbers and strings that appear in the UI.

In the past, there were differences in the JSP and JSF EL syntax and how the container evaluated them. The unified EL resolves these differences while also adding features such as:

- A pluggable framework to enable customization of EL interpretation.
- Support for deferred expressions that can be executed as needed by a JSP tag handler.
- Support for assignment operations that can, for example, use an EL expression to set a property on a JavaBean from within JSP code.

A boon for JSP tag-library developers is that tag handlers now support resource injection using annotations, so tags that once required resource configuration and code to perform Java Naming and Directory Interface (JNDI) lookups can now be greatly simplified.

JSTL 1.2

JSTL has been around for years, but until Java EE 5, it existed outside of the Java EE umbrella. JSTL tags provide support for embedding the following types of elements in JSPs:

- **Procedural logic**, such as loops and `if/else` switches.
- **Access to JavaBeans** that can provide dynamic data to the UI or be modified by UI code.
- **SQL commands** to perform database access.
- **Formatting instructions** that format UI output according to specific locales.
- **XML processing**, such as Document Object Model (DOM) parsing or Extensible Stylesheet Language (XSL) transformations.

JSTL 1.2 is a maintenance release that supports the unified EL and resolves issues that used to arise from trying to mix JSF tags and JSTL iteration tags in the same JSP page.

Java Servlet 2.5

The Java Servlet specification — the backbone of Java Web-tier technology — is as old as Java EE technology itself. The specification is designed to provide a high-performing, component-based approach to building Web applications that are portable across any servers that implement the specification.

The Servlet 2.5 specification required by Java EE 5 is a maintenance release that includes minor improvements over the 2.4 release. It introduces a dependency on the Java 5 platform along with some annotations that reduce the need for configuration in the Web application deployment descriptor (web.xml) configuration file. Additional configuration conveniences have been added, such as more-flexible servlet configuration with wildcarding and multiple `url-pattern` elements.

Back to top

Enterprise application technologies

A lot of technologies fall under the enterprise application umbrella, many of which have not changed with the Java EE 5 release or are too detailed fall within this article's scope. I'll focus here on the two major improvements: ease of EJB development and new persistence features.

EJB 3.0

The EJB specification is core to the Java EE platform. It defines a way to encapsulate an application's business logic and distribute it in a highly scalable, secure, and transaction-aware way so that simultaneous data access doesn't result in corrupt data.

EJBs are of three basic types:

- **Session beans** come in two flavors: *stateless* and *stateful*. Stateless session beans are used for business-logic tasks that service a single request from client code. Stateful session beans maintain a "conversational state" with the client and are good for sets of related tasks that span multiple client requests. Session beans cannot be shared among clients. Session beans typically operate on one or more entity beans.
- **Entity beans** represent persistent data that is typically loaded from a database. Entity beans can be shared across clients, and the EJB specification provides transaction-safety mechanisms that ensure that entity beans can safely serve multiple, concurrent client requests without becoming corrupted. An entity bean can manage its own persistence or let the container manage it (*container-managed persistence*or CMP).
- **Message-driven beans** (MDBs) serve requests from client code without forcing the client to wait for the response. They typically interact with a Java Message Service (JMS) queue — another enterprise application technology in Java EE 5 — but can also serve asynchronous clients, even ones not written in Java code, by other means.

In the past, EJB development has been complex and unwieldy, often forcing developers to rely on tools to manage the complexity of all the interfaces and deployment descriptors required to implement EJBs. It was invasive on the business-logic code, mandating that certain classes be extended or certain interfaces be implemented. Obtaining a simple reference to an EJB involved a lot of boilerplate code. These issues garnered EJB a bad reputation in the development community — in many cases, deservedly so.

EJB 3.0 represents a vast improvement in the EJB programming model and is one of the biggest potential sources of increased productivity for Java EE 5 developers. An EJB can now be an annotated "plain old Java object" (POJO) that doesn't need to extend a certain class. It only needs to implement a remote interface that you define or allow your IDE to create automatically. Deployment descriptors are no longer required because the EJB container can extract all it needs to know from the annotations on an EJB.

This article's Kicking the tires: The RideSynergy application section presents example code that provides concrete examples of these improvements. For readers thirsty for deeper details, you'll find links in Resources to two articles that offer compelling examples of how much EJB development has improved with the latest release.

Java Persistence API (JPA 1.0)

JPA introduces an object-relational mapping (ORM) framework for persisting Java objects. It was developed with EJBs in mind, but it can be used for any Java objects. Using annotations, you indicate which objects and fields are persistent and which database tables and columns they map to. JPA supports a rich query language that resembles SQL. The query language enables:

- Definition of parameterized queries that can take parameters as an ordered list, referred to by index numbers, or as named parameters referred to by name.
- Queries that traverse the relationships between persistent entities without requiring `JOIN` statements (though you can use `JOIN` statements if you prefer).
- Typical SQL-like features to form search criteria (comparison operators, `LIKE` statements, `BETWEEN` statements, and so on) and define how to treat the result set (using operators such as `DISTINCT`, `ORDER BY`, `GROUP BY`, and so on.)

JPA brings new functionality to the Java EE platform, relieving many of the past headaches of roll-your-own or container-specific persistence approaches. The article "Design enterprise applications with the EJB 3.0 Java Persistence API" is a good starting point for learning more.