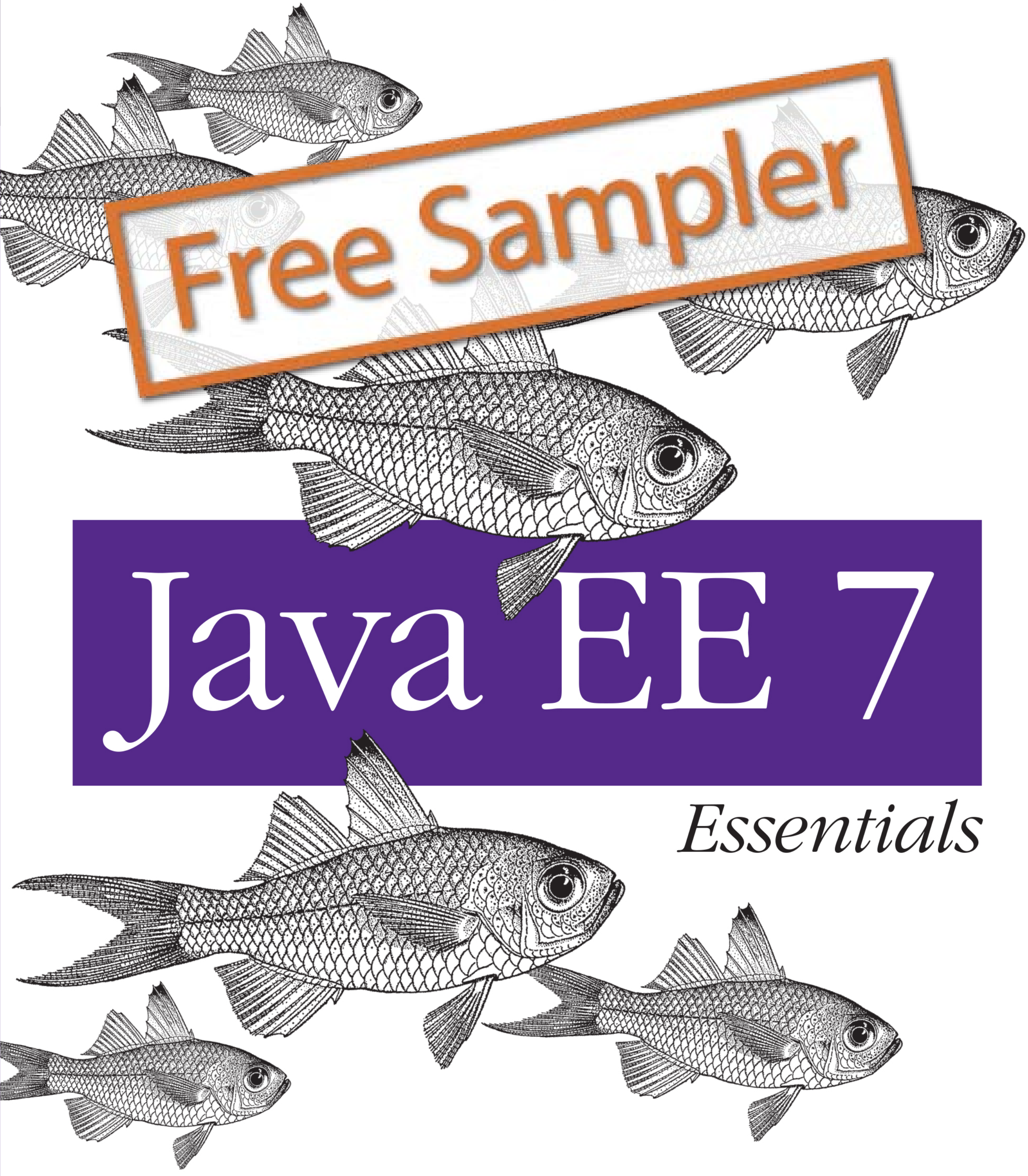
Enterprise Developer Handbook



Java EE 7 *Essentials*

O'REILLY®

Arun Gupta

Programming/Java

Java EE 7 Essentials

Get up to speed on the principal technologies in the Java Platform, Enterprise Edition 7, and learn how the latest version embraces HTML5, focuses on higher productivity, and provides functionality to meet enterprise demands. Written by Arun Gupta, a key member of the Java EE team, this book provides a chapter-by-chapter survey of several Java EE 7 specifications, including WebSockets, Batch Processing, RESTful Web Services, and Java Message Service.

You'll also get self-paced instructions for building an end-to-end application with many of the technologies described in the book, which will help you understand the design patterns vital to Java EE development.

- Understand the key components of the Java EE platform, with easy-to-understand explanations and extensive code samples
- Examine all the new components that have been added to the Java EE 7 platform, such as WebSockets, JSON, Batch, and Concurrency
- Learn about RESTful Web Services, SOAP XML-based messaging protocol, and Java Message Service
- Explore Enterprise JavaBeans, Contexts and Dependency Injection, and the Java Persistence API
- Discover how different components were updated from Java EE 6 to Java EE 7

Arun Gupta, a Java evangelist working at Oracle, has several years of experience working with the Java platform. He has been with the Java EE team since its inception and has contributed to all of the releases.

US $39.99 CAN $41.99
ISBN: 978-1-449-37017-6
53999
9 781449 370176

Java EE 7 Essentials

Arun Gupta

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Java EE 7 Essentials

by Arun Gupta

Copyright © 2013 Arun Gupta. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Mike Loukides and Meghan Blanchette	**Indexer:** Angela Howard
Production Editor: Kara Ebrahim	**Cover Designer:** Randy Comer
Copyeditor: Rachel Monaghan	**Interior Designer:** David Futato
Proofreader: Linley Dolby	**Illustrator:** Rebecca Demarest

August 2013: First Edition

Revision History for the First Edition:

2013-08-08: First release

See *http://oreilly.com/catalog/errata.csp?isbn=9781449370176* for release details.

ISBN: 978-1-449-37017-6

[LSI]

Table of Contents

CHAPTER 1
Java Platform, Enterprise Edition

Introduction

The Java Platform, Enterprise Edition (Java EE), provides a standards-based platform for developing web and enterprise applications. These applications are typically designed as multitier applications, with a frontend tier consisting of a web framework, a middle tier providing security and transactions, and a backend tier providing connectivity to a database or a legacy system. These applications should be responsive and capable of scaling to accommodate the growth in user demand.

The Java EE platform defines APIs for different components in each tier, and also provides some additional services such as naming, injection, and resource management that span across the platform. These components are deployed in containers that provide runtime support. Containers provide a federated view of the underlying Java EE APIs to the application components. Java EE application components never interact directly with other Java EE application components. They use the protocols and methods of the container for interacting with each other and with platform services. Interposing a container between the application components and the Java EE services allows the container to transparently inject the services required by the component, such as declarative transaction management, security checks, resource pooling, and state management. This container-based model and abstraction of resource access allows the platform to offload the developer from common infrastructure tasks.

Each component of the platform is defined in a separate specification that also describes the API, javadocs, and expected runtime behavior.

Java EE 7 was released in June 2013 and provides a simple, easy-to-use, and complete stack for building such web and enterprise applications. The previous versions of the platform, starting with Java EE 5 and continuing with Java EE 6, took the first steps in providing a simplified developer experience.

The Java EE 7 platform built upon the previous version with three main goals:

Embracing HTML5

The WebSocket protocol, developed as part of the collection of technologies that make up HTML5, brings a new level of ease of development and network efficiency to modern, interactive web applications. It provides a two-way, full-duplex communication channel between a client and a server over a single TCP (transmission control protocol) channel. Java EE 7 defines a new standard API to develop and deploy WebSocket clients and endpoints.

JSON is the lingua franca of the Web for lightweight data exchange. Until now, developers were required to bundle third-party libraries for JSON processing. Java EE 7 defines a new portable API to parse, generate, transform, and query JSON using a Streaming API or Object Model API.

JavaServer Faces (JSF) introduces pass-through attributes and elements that allow near-total control over the user experience of each individual element in the view. This allows HTML5-friendly markup to be easily embedded in a page.

Higher productivity

The JMS API has been greatly simplified. `JMSContext` provides the unified functionality of `Connection` and `Session` objects. In addition, several JMS interfaces implement `Autocloseable` and thus are automatically closed after use. Finally, correct error handling, runtime exceptions instead of checked exceptions, method chaining on `JMSProducer`, and simplified message sending are further examples of features that the JMS API has simplified.

Without the Client API (introduced in JAX-RS 2), developers are required to use basic `HttpUrlConnection` APIs and write all the surrounding code.

More defaults for the application's use—such as a preconfigured `DataSource` for accessing databases in operational environments, a preconfigured JMS `ConnectionFactory` for accessing a JMS provider, and a preconfigured `ManagedExecutorService`—provide a seamless out-of-the-box experience for new developers starting with the platform.

The Contexts and Dependency Injection (CDI) specification is now a core component model, and is enabled by default. This makes the platform a lot more cohesive and integrated. CDI interceptors are now more widely applicable to beans. `@Transactional` annotation brings transactional semantics to POJOs (plain old Java objects), outside of an EJB (Enterprise JavaBean). Bean Validation allows automatic validation of method arguments and results using interceptors.

Less boilerplate text, more defaults, and a cohesive integrated platform together boost developers' productivity when building applications using the latest version of the platform.

Enterprise demands

Batch Applications for the Java Platform is a new functionality in the platform and very important for enterprise customers. It allows developers to easily define non-interactive, bulk-oriented, long-running jobs in an item- or task-oriented way.

Concurrency Utilities for Java EE, another functionality new to the platform, is an extension of the Java SE Concurrency Utilities API, for use in the Java EE container-managed environment so that the proper container-managed runtime context can be made available for the execution of these tasks.

This functionality in the platform allows the developer to leverage the standard APIs and reduces the dependency on third-party frameworks.

Prior to Java EE 7, the Java EE 6 platform improved upon the developer productivity features and added a lot more functionality.

Deliverables

The Java EE 7 platform was developed as Java Specification Request (JSR) 342 following JCP 2.9. The JCP process defines three key deliverables for any JSR:

Specification

A formal document that describes the proposed component and its features.

Reference Implementation (RI)

Binary implementation of the proposed specification. The RI helps to ensure that the proposed specifications can be implemented in a binary form and provides constant feedback to the specification process.

The RI of Java EE is built in the GlassFish community.

Technology Compliance Kit (TCK)

A set of tests that verify that the RI is in compliance with the specification. This allows multiple vendors to provide compliant implementations.

Java EE 7 consists of the platform specification that defines requirements across the platform. It also consists of the following component specifications:

Web technologies

- JSR 45: Debugging Support for Other Languages 1.0
- JSR 52: Standard Tag Library for JavaServer Pages (JSTL) 1.2
- JSR 245: JavaServer Pages (JSP) 2.3
- JSR 340: Servlet 3.1
- JSR 341: Expression Language 3.0
- JSR 344: JavaServer Faces (JSF) 2.2

- JSR 353: Java API for JSON Processing (JSON-P) 1.0
- JSR 356: Java API for WebSocket 1.0

Enterprise technologies
- JSR 236: Concurrency Utilities for Java EE 1.0
- JSR 250: Common Annotations for the Java Platform 1.2
- JSR 316: Managed Beans 1.0
- JSR 318: Interceptors 1.2
- JSR 322: Java EE Connector Architecture (JCA) 1.7
- JSR 330: Dependency Injection for Java 1.0
- JSR 338: Java Persistence API (JPA) 2.1
- JSR 343: Java Message Service (JMS) 2.0
- JSR 345: Enterprise JavaBeans (EJB) 3.2
- JSR 346: Contexts and Dependency Injection (CDI) for the Java EE Platform 1.1
- JSR 349: Bean Validation 1.1
- JSR 352: Batch Applications for Java Platform 1.0
- JSR 907: Java Transaction API (JTA) 1.2
- JSR 919: JavaMail 1.5

Web service technologies
- JSR 93: Java API for XML Registries (JAXR) 1.0 (optional for Java EE 7)
- JSR 101: Java API for XML-based RPC (JAX-RPC) 1.1 (optional for Java EE 7)
- JSR 109: Implementing Enterprise Web Services 1.4
- JSR 181: Web Services Metadata for the Java Platform 2.1
- JSR 222: Java Architecture for XML Binding (JAXB) 2.2
- JSR 224: Java API for XML Web Services (JAX-WS) 2.2
- JSR 339: Java API for RESTful Web Services (JAX-RS) 2.0

Management and security technologies
- JSR 77: J2EE Management API 1.1
- JSR 88: Java Platform EE Application Deployment API 1.2 (optional for Java EE 7)
- JSR 115: Java Authorization Contract and Containers (JACC) 1.5

- JSR 196: Java Authentication Service Provider Inteface for Containers (JASPIC) 1.1

The different components work together to provide an integrated stack, as shown in Figure 1-1.

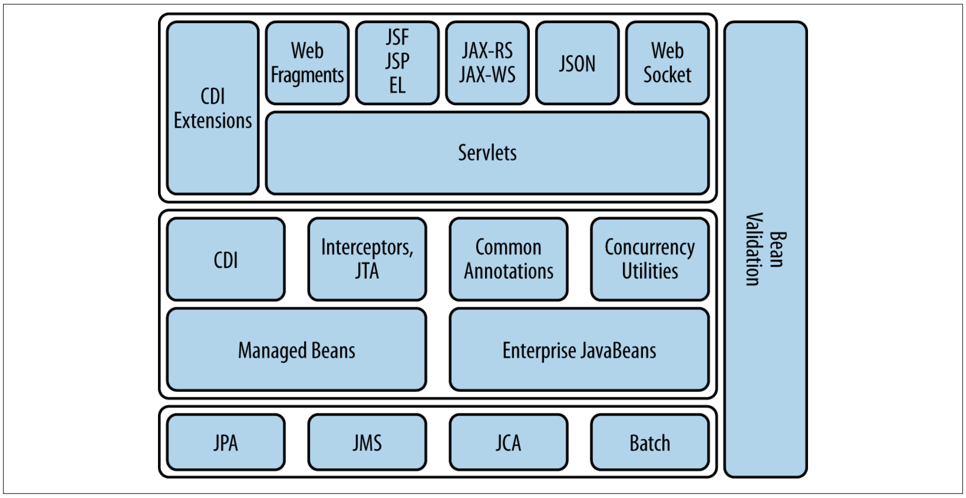


Figure 1-1. Java EE 7 architecture

In Figure 1-1:

- Different components can be logically divided into three tiers: backend tier, middle tier, and web tier. This is only a logical representation, and the components can be restricted to a different tier based upon the application's requirements.
- JPA and JMS provide the basic services such as database access and messaging. JCA allows connection to legacy systems. Batch is used for performing noninteractive, bulk-oriented tasks.
- Managed Beans and EJB provide a simplified programming model using POJOs to use the basic services.
- CDI, Interceptors, and Common Annotations provide concepts that are applicable to a wide variety of components, such as type-safe dependency injection, addressing cross-cutting concerns using interceptors, and a common set of annotations. Concurrency Utilities can be used to run tasks in a managed thread. JTA enables Transactional Interceptors that can be applied to any POJO.
- CDI Extensions allow you to extend the platform beyond its existing capabilities in a standard way.

- Web Services using JAX-RS and JAX-WS, JSF, JSP, and EL define the programming model for web applications. Web Fragments allow automatic registration of third-party web frameworks in a very natural way. JSON provides a way to parse and generate JSON structures in the web tier. WebSocket allows the setup of a bidirectional, full-duplex communication channel over a single TCP connection.
- Bean Validation provides a standard means to declare constraints and validate them across different technologies.

JAX-RPC (JSR 101), JAXR (JSR 93), EJB Entity Beans (part of JSR 153), and Java EE Application Deployment (JSR 88) are pruned in this version of the platform.

The RI of Java EE 7 is built in the GlassFish Community. The GlassFish Server Open Source Edition 4.0 provides a full Java EE 7–compliant, free, and open source application server. It is also available in a Web Profile distribution and can be downloaded from *http://glassfish.org*. The application server is easy to use (ZIP installer and NetBeans/Eclipse/IntelliJ integration), lightweight (downloads starting at 37 MB, small disk/memory footprint), and modular (OSGi-based, containers start on demand).

Prior to Java EE 7, GlassFish Server Open Source Edition 3.1.2.2 provides a Java EE 6–compliant version application server. It also provides clustering with high availability and centralized administration using CLI, Web-based administration console, and REST management/monitoring APIs. The Oracle GlassFish Server is Oracle's commercially supported GlassFish server distribution and can be downloaded from *http://oracle.com/goto/glassfish*. As of this writing, there are 18 Java EE 6–compliant application servers.

The TCK is available to all Java EE licensees for testing their respective implementations.

What's New in Java EE 7

Some new specifications have been added to improve the functionality and richness of the platform. Several existing component specifications were revised to make them simpler and easier to use.

The main features of the new specifications are described as follows:

Java API for WebSocket

- Enables a WebSocket client and server endpoint to be defined declaratively via annotations on a POJO, or programmatically via interface implementation.
- Provides server-specific configuration, such as mapping that identifies a WebSocket endpoint in the URI space of the container, subprotocols supported by the endpoint, and extensions required by the applications.
- Offers client-specific configurations such as providing custom configuration algorithms.

- Enables packaging and deployment on JDK or web containers.
- Allows for integration with existing Java EE technologies.

Java API for JSON Processing

- The streaming API provides a way to parse and generate JSON in a streaming fashion.
- The Object Model API creates a random-access, tree-like structure that represents the JSON data in memory.

Batch Applications for Java Platform

- Allows for description of a Batch Job using Job Specification Language defined by an XML schema. It defines a complete execution sequence of the jobs.
- Features the Batch Programming Model using interfaces, abstract classes, and field annotations.
- Offers the Chunked and Batchlet job-processing styles.

Concurrency Utilities for Java EE

- Provides concurrency capabilities to Java EE application components, without compromising container integrity.
- Defines managed objects: `ManagedExecutorService`, `ManagedScheduledExecutorService`, `ContextService`, and `ManagedThreadFactory`.

The main features of the updated specifications are described as follows:

Java API for RESTful Web Services

- Offers a new Client API that can be used to access web resources and provides integration with JAX-RS providers.
- Supports asynchronous processing in both the Client API and the Server API.
- Defines Message Filters and Entity Interceptors as extension points to customize the request/response processing on the client and server side.
- Supports new server-side content negotiation using `qs` factor.
- Enables declarative validation of fields, properties, and parameters injected using `@HeaderParam`, `@QueryParam`, etc. Resource classes may be annotated with constraint annotations.

Java Message Service

- Several changes have been made to make the API simpler and easier to use. For example, `Connection`, `Session`, and other objects with a `close` method now implement the `java.lang.Autocloseable` interface to allow them to be used in a Java SE 7 try-with-resources statement. New methods have been added to create a session without the need to supply redundant arguments. A new method, `getBody`, has been added to allow an application to extract the body directly from a `Message` without the need to cast it first to an appropriate subtype.

- A message producer can now specify that a message must not be delivered until after a specified time interval.
- New send methods have been added to allow an application to send messages asynchronously.
- JMS providers must now set the `JMSXDeliveryCount` message property.

Expression Language

- Expression Language (EL) is a specification of its own. It can be configured and used outside of a Java EE container using `ELProcessor`.
- The lambda syntax is now included in EL. Using lambdas, a complete set of collection operations, such as map and filter is now supported.
- In addition to the usual arithmetic and comparison operators, new operators—such as an assignment operator and string concatenation operator—have been added to make EL more expressive.

Enterprise JavaBeans

- Support for EJB 2.1, EJB QL, and JAX-RPC-based Web Service endpoints and client view is now optional.
- Enhanced message-driven beans (MDBs) contract with a no-method message listener interface to expose all public methods as message listener methods. This will allow custom Resource Adapters for future MDBs.
- EJB API Groups have been defined with clear rules for an EJB Lite Container to support other API groups. This will help define how EJB features beyond EJB Lite can be officially added to a product that does not support full Java EE Profile.
- Asynchronous session bean invocations and nonpersistent EJB Timer Service are included in EJB Lite.
- An option has been added to disable passivation of stateful session beans.

Servlets

- Defines a standard mechanism to upgrade existing HTTP connection to a different protocol using `HttpUpgradeHandler`.
- Offers nonblocking request and response processing for async servlets.
- Defines rules for which HTTP methods are covered by `<security-constraint>`.

JavaServer Faces

- Faces Flow provides an encapsulation of related views/pages with application-defined entry and exit points.
- Resource Library Contracts enable developers to apply facelet templates to an entire application in a reusable and interchangeable manner.
- HTML5-friendly markup allows near-total control over the user experience of each individual element in the view.
- Stateless Views mean developers no longer have to save the `UIComponent` state. This allows applications with JavaScript components to manage the state instead of JSF doing it for them.

Java Persistence

- Database schema and tables may be generated by using `javax.persistence.schema-generation.*` properties.
- Unsynchronized Persistence Contexts mean a persistence context does not have to be enlisted in a transaction. Such persistence contexts can explicitly join a transaction.
- Bulk update/delete is supported via `Criteria` API.
- Predefined and user-defined functions can be invoked using `FUNCTION`.
- Stored procedures can be invoked using `StoredProcedureQuery` and `@NamedStoredProcedureQuery`.

Interceptors

- Associating interceptors using `InterceptorBinding` is now part of this specification, instead of CDI.
- `@AroundConstruct` designates an interceptor method that receives a callback when the target class constructor is invoked.
- Method-level interceptors can be extended to life-cycle callbacks, adding constructor-level interceptors.
- Priority ranges can be dedicated for ordering interceptors using interceptor binding.

Contexts and Dependency Injection

- Allows for automatic enabling of CDI for beans with a scope annotation, and EJBs, in Java EE.
- Supports global ordering and enabling of interceptors, decorators, and alternatives using the `@Priority` annotation.
- Adds the `@Vetoed` annotation, allowing easy programmatic disabling of classes.

Bean Validation

- Validation constraints can be applied to the parameters and return values of arbitrary methods and constructors.
- Integration points with CDI have been increased and reworked.
- The targeted group can be altered when validation cascading is happening.

Java Transaction

- `@Transactional` provides the application to declaratively control transaction boundaries on CDI-managed beans, as well as classes defined as managed beans by the Java EE specification, at both the class and method level where method-level annotations override those at the class level.
- `@TransactionScoped` is a new CDI scope that defines bean instances whose life cycle is scoped to the currently active JTA transaction.

JavaMail

- `@MailSessionDefinition` and `@MailSessionDefintions` defines `MailSession` to be registered with JNDI.

Java EE Connector Architecture

- Provides `@AdministeredObjectDefinition`, `@AdministeredObjectDefintions`, `@ConnectorFactoryDefinition`, and `@ConnectorFactoryDefinitions` to define a connector-administered object and factory to be registered in JNDI.