An Oracle White Paper
June 2013

Introduction to Java Platform, Enterprise Edition 7

Executive Overview

Java Platform, Enterprise Edition 7 (Java EE 7) offers new features that enhance HTML5 support, increase developer productivity, and further improves how enterprise demands can be met. Java EE 7 developers will write less boilerplate code, have better support for the latest Web applications and frameworks, and gain access to enhanced scalability and richer, simpler functionality. Enterprises will benefit from new features that enable portable batch processing and improved scalability.

Introduction

Java EE initially evolved as an enterprise application deployment platform that focused on robustness, Web services, and ease of deployment. Continually shaped by feedback through the Java Community Process (JCP), Java EE represents a universal standard in enterprise IT, facilitating the development, deployment, and management of multi-tier, server-centric applications. Beginning with Java EE 5, focus shifted to increasing developer efficiency with the introduction of annotations, the Enterprise JavaBeans (EJB) 3.0 business component development model, new and updated Web services, and improvements to the persistence model. Java EE 6 further streamlined the development process and increased the flexibility of the platform, thus enabling it to better address lightweight Web applications. This is in part due to the introduction of the Web Profile as a subset of the Java EE specification targeted to Web applications. In addition, Java EE 6 embraced open source frameworks with hooks for more seamless integration, and began the process of pruning less relevant technologies.

Java EE 6 was particularly successful:

- As of May 2013, there have been over 50 million downloads of Java EE components, from Oracle and other industry vendors.
- It is the #1 choice for enterprise developers.
- It is the #1 application development platform.
- It has had the fastest adoption of any Java EE release with 18 compliant application server vendors.

Java EE 7 extends the benefits of Java EE 6 by leveraging the more transparent JCP process and community participation to deliver new and updated features, excelling in the areas expressed in Figure 1:

Java
ENTERPRISE EDITION
DEVELOPER PRODUCTIVITY
Java EE 7
HTML
MEETING ENTERPRISE DEMANDS

Figure 1

- Java EE 7 enables developers to deliver HTML5 dynamic scalable applications. New to the platform, WebSockets reduce response time with low latency bi-directional data exchange while standard JSON support simplifies data parsing for portable applications. JAX-RS has been improved to deliver asynchronous, scalable, high performance RESTful Services. And much more.
- Java EE 7 increases developer productivity in multiple ways. It offers a simplified application architecture with a cohesive integrated platform; increased efficiency with reduced boiler-plate code and broader use of annotations; and enhanced application portability with standard RESTful Web service client support.
- Java EE 7 meets the most demanding enterprise requirements by breaking down batch jobs into manageable chunks for uninterrupted OLTP performance; easily defines multithreaded concurrent tasks for improved scalability; and delivers transactional applications with choice and flexibility.

Java EE 7 was developed with the most extensive Java community participation of any Java EE release, with vendors, organizations and individuals all contributing to Expert Groups. The Java community has been an engaged partner, offering reviews and feedback in the ongoing development of the specifications. The Java Community Process (JCP) has refined its processes in ways that have facilitated greater openness and more accessible participation among stakeholders. 19 Java User Groups (JUGs) throughout the world, ranging from North America to Europe, South America, and Asia, have participated in the Adopt-a-JSR program, reviewing platform proposals and developing several applications in an effort to explore, test, and create code samples for proposed features.

One of these applications is now a part of the Java EE 7 SDK, an all-in-one bundle with API documentation, samples, tutorials, and GlassFish Server Open Source Edition 4.0 that is used to teach developers how to become proficient with Java EE 7. While the SDK is a way to get started with Java EE, the platform is a multi-vendor and community technology with wide cross-industry investment represented by 19 Java EE 6 implementations from multiple vendors and many Java EE 7 implementations are forthcoming.

This White Paper provides a technical overview of Java EE 7 and specifies ways in which its new features and functionalities enable Java EE developers to work with greater efficiency and productivity.

Introducing Java Platform, Enterprise Edition 7

Since its inception, the Java EE platform has been targeted for offloading common infrastructure tasks through its container-based model and abstraction of resource access so developers can focus on business logic. In recent releases, the platform has considerably simplified the APIs for access to container services while broadening the range of services available. Java EE 7 continues this trend with enhanced simplification and productivity while further extending the range of the platform to encompass support for emerging Web technologies.

Deliver Dynamic Scalable HTML5 Applications

HTML5 is accelerating the ability of developers to create applications that are highly interactive and dynamic, alongside client-side technologies like JavaScript and CSS3. These applications can deliver live data feeds such as sport scores, stock news and quotes, application notifications, twitter and chat feeds, and more, all in a highly interactive manner. HTML5 enables these applications to be written once and render properly on a range of devices like smart phones, tables, and desktops. These highly dynamic applications, combined with the ability to access them at any time from anywhere, are driving the need to scale the services that feed application data to the client. Java EE 7 lays the foundation for dynamic HTML5 applications with new JSRs like WebSockets and JSON Processing, along with updates to existing JSRs like JAX-RS 2.0, Java Server Faces 2.2, and Servlet 3.1 NIO.

Low Latency Data Exchange Using the Java API for WebSocket 1.0

A growing number of Web applications rely on timely updates from central servers. The Java developer community has expressed considerable interest in WebSockets because they offer a solution to the inherent problems of latency and bi-directional communication that come with HTTP-based solutions like polling, long-polling and HTTP-streaming.

WebSockets seamlessly support low latency, bi-directional client-server data exchange over a single TCP connection. This can be exemplified by a whiteboard application, where multiple participants can be drawing on a shared whiteboard, seeing each other's work simultaneously. The WebSocket API, at its most basic level, is an annotated plain old Java object (POJO) as shown here:

```
@ServerEndpoint("/whiteboard”)
public class WhiteboardServer {
   @OnOpen
   public void onOpen(…) { }
   @OnClose
   public void onClose(…) { }

   @OnMessage
   public void message(String message, …) { }
}
```

Defining an endpoint to a socket is as simple as specifying the URI with the @ServerEndpoint annotation. The annotation-based callback API responds to specific events, such as when a client connects, a message is received, and a client disconnects. The WebSocket API, at its most basic level, supports sending and receiving simple text and binary messages. The simplicity of the API enables developers to get started quickly.

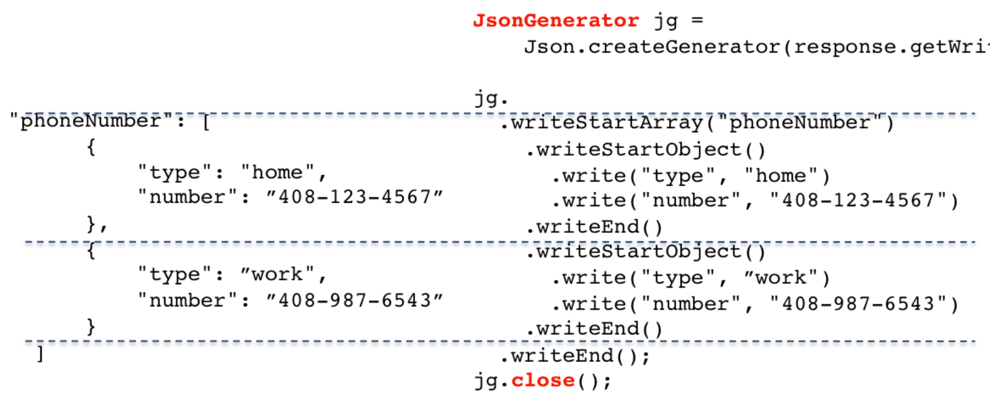
Of course, feature-rich applications have more complex needs, and for those the WebSocket API supports programmatic endpoints that allow control of the protocol handshake and message exchange. In addition, WebSockets leverage the existing web container security model for authentication, authorization, and transport guarantee, so secure communication can be established with little effort.

Simplify Data Parsing for Portable Applications with Java API for JSON Processing 1.0

JSON (JavaScript Object Notation), a lightweight data-interchange format, is used by many popular Web services to invoke and return textual data. Many popular online services, like Twitter, Facebook, and Pinterest, expose RESTful services that exchange JSON objects. Prior to Java EE 7, Java applications have used different implementation libraries to produce and consume JSON from RESTful services. However, this is no longer the case.

With the Java API for JSON Processing 1.0, JSON processing is standardized into a single API so that applications that use JSON need not bundle 3rd party implementation libraries. As a result, applications will be smaller in size and more portable. However, the API includes support for plugging in any parser/generator implementation, so developers have the option to use the best implementation for the job at hand.

The Java API for JSON Processing will look familiar to JAXP developers. It can produce and consume JSON text in a streaming fashion similar to StaX's XMLStreamReader, a pull parser. The API also supports a Java Object Model representation of the JSON, similar to the DOM API. The image below shows a simple JSON-formatted object on the left, with the Java Object Model representation to the right, with the dashed guide to facilitate the comparison. Generating a JSON object using the API is more straightforward and less error prone than manually creating JSON.



```
"phoneNumber": [
      {
          "type": "home",
          "number": "408-123-4567"
      },
      {
          "type": "work",
          "number": "408-987-6543"
      }
  ]
```

```
JsonGenerator jg =
    Json.createGenerator(response.getWriter());

jg.
  .writeStartArray("phoneNumber")
    .writeStartObject()
      .write("type", "home")
      .write("number", "408-123-4567")
    .writeEnd()
    .writeStartObject()
      .write("type", "work")
      .write("number", "408-987-6543")
    .writeEnd()
  .writeEnd();
jg.close();
```

Scalable RESTful Services with Java API for RESTful Web Services 2.0 – JAX-RS 2.0

JAX-RS 2.0 adds asynchronous response processing, which is critical for scaling to meet the demands of data-hungry HTML5 clients. Asynchronous processing is a technique that enables a better and more efficient use of processing threads. On the server side, a thread that is processing a request should avoid blocking while waiting for an external task to complete so that other requests arriving at the server during that period of time can be attended. Accessing remote RESTful resources from Twitter or Facebook, for example, can now occur without blocking other clients while the request is being serviced.

Similarly, on the client side, a thread that issues a request will block while waiting for a response, impacting the performance of the application. The new JAX-RS 2.0 asynchronous client API enables the client's call to a RESTful service to execute in parallel with other client activity. The client can poll for a response or use a callback API to be notified of a response. The benefit of the asynchronous client API is that a client can invoke multiple backend services simultaneously, reducing the client's overall latency to the request originator.

To easily enhance a RESTful service, JAX-RS 2.0 developers can use Filters and Entity Interceptors. JAX-RS 2.0 filters are similar to Servlet filters. They execute before and after request and response processing, and are primarily used to modify or process incoming and outgoing request or response headers. Like Servlet filters, they can be chained together so multiple filters can inspect requests. The JAX-RS 2.0 Entity Interceptor API allows framework developers to intercept request and response processing, just as the Interceptor API does for Java methods. The Interceptors operate on the request and response message bodies instead of headers. This API allows framework developers to transparently add orthogonal concerns like authentication, caching, and encoding without polluting application code. Interceptors are intended to be applicable to any kind of JAX-RS service. Prior to JAX-RS 2.0, many JAX-RS providers like RESTEasy, Jersey, and Apache CXF wrote their own proprietary Filter and Entity Interceptor frameworks to deliver various features in their implementations. Developers can now utilize the standard APIs for these features, enabling more portable applications.

Enhanced Ease of Development with JavaServer Faces 2.2

JavaServer Faces is the standard, component-oriented Java EE framework for building portable Web application user interfaces. It maximizes the productivity of Web application development for graphical IDEs, while simultaneously minimizing the complexity of maintenance of the Web application during its production lifetime. With this release, JSF adds support for HTML5.

JavaServer Faces 2.2 offers HTML5-friendly markup, enabling page authors to write “pure” HTML markup that can be viewed in an HTML tool, or simply rendered in a browser page as HTML-formatted code without any clunky XML markup. As illustrated below, any JSF attributes that are preceded by “jsf:” are ignored by the browser and passed on to the server.

```
<button type=“submit” jsf:id=“submitbutton”
                      jsf:action=“#{bean.action}”>
 Click Me!
</button>
```

JSF 2.2 includes a new feature called "pass-through elements". HTML5 adds a series of new attributes for existing elements, like "tel", "range", and "date" for input elements. Unfortunately, existing JSF components do not recognize these new attributes, so JSF applications would ignore these attributes and be unable to use them, or proprietary or one-off workarounds will have to be created. With pass-through elements, the JSF renderer ignores the elements, and instead just passes them to the HTML5-enabled browser, which renders them properly -- enabling existing JSF components to "utilize" HTM5 features.

JSF introduces a new pass-through namespace http://xmlns.jcp.org/jsf/passthrough that maps to "p:". Any arbitrary name/value pair in a component can be prefixed with "p:" and passed through to the browser.

```
<h:inputText
      value="#{bean.color}"
      p:type="color" />
```

In this case, HTML5 "type=color" is passed through to the browser, without any interpretation by the JSF component.

Improved Request Processing with Servlet 3.1 NIO

HTML5 applications are inherently more dynamic, and drive many more requests to the server for information updates. In Java EE 6, Servlet Asynchronous I/O enabled many more concurrent requests by removing the "thread per request" limitation, enabling a thread to handle multiple concurrent requests. This can help deliver necessary data to an HTML5 client in a scalable manner. However, if the server can read data faster than the client can send it, perhaps due to a slow client connection, the thread will block until more data is available, therefore limiting scalability. With Servlet 3.1 NIO, reading data from a client is non-blocking when using the new event-driven API. When data is available, a Servlet thread can read and process just that data, and then move on to another request.

Increased Developer Productivity

Beginning with Java EE 5, a tremendous amount of focus has been placed on developer productivity. This is important to Java developers because it makes Java EE more enjoyable to work with, and more importantly, helps meet management deadlines. Developer productivity is important to business because it can deliver new services in less time, and new services drive new revenue opportunities.

Java EE 7 delivers significant developer productivity improvements. First, it removes the amount of boilerplate code required to write core business logic. Next, the platform continues its convention over configuration approach to development by introducing more annotated POJOs that use little to no XML configuration. Last, the technologies that are delivered in Java EE 7 are more tightly integrated, offering a more seamless developer experience.

Reduce Boilerplate Code

Java EE 7 goes a long way towards reducing the amount of “boilerplate” code, which is a set of required steps in code that must be executed before the core business logic can run. The top three areas that reduce boilerplate code include default resources, JMS 2.0, and the JAX-RS client API.

Default resources is a new feature that requires the platform provider to pre-configure a default data source and a default JMS connection factory, for example, that maps to an underlying database and JMS runtime respectively. This option eliminates the need for developers to define these resources since they can rely on default resources. This provides a simpler out-of-the-box developer experience for building sample applications.

JMS has gone through significant improvements as well, and JMS 2.0 is the first update to the JMS 1.1 API since 2003. JMS is used in countless production deployments, and the fact that it has been meeting enterprise needs for ten years proves that it is a well-defined specification. The JMS 1.1 API was fairly verbose due to available Java SE and Java EE capabilities at the time. For example, the JMS 1.1 API required 13 lines of boilerplate code just to send a message. However, modern features in Java SE and Java EE, combined with a refreshed API, have enabled JMS 2.0 to significantly simplify the API for developers. For example, JMS 2.0 introduces a JMSContext interface to reduce two separate JMS 1.1 classes down to a single interface; utilizes Java EE 7 default resources with the default connection factory; supports AutoCloseable; uses runtime exceptions; and chains method calls together. Together, all of these reduce the lines of code required to send a message down to one, as shown below.

```
@Inject
JMSContext context;

@Resource(lookup = "java:global/jms/demoQueue”)
Queue demoQueue;

public void sendMessage(String payload) {
    context.createProducer().send(demoQueue, payload);
}
```

13 lines down to 1 line!

JAX-RS 2.0 has not just gone through scalability improvements, it also addresses the most commonly requested feature: a client API. Many if not all JAX-RS 1.1 implementations provide some degree of client API support. However, until JAX-RS 2.0 they were all different, impacting application portability. To develop portable applications, developers would have to use a rather unproductive approach with the HTTPUrlConnection class, custom error checking, manage data bindings, and more. While the approach is not difficult, it requires a lot of boilerplate code just to invoke a RESTful service and get a response. As shown below, the JAX-RS 2.0 client API uses a builder pattern where developers can chain together method calls to build the RESTful client invocation, requiring only two lines of code!

```
// Get instance of Client
Client client = ClientBuilder.newClient();

// Get customer name for the shipped products
String name = client.target("../orders/{orderId}/customer")
                    .resolveTemplate("orderId", "10")
                    .queryParam("shipped", "true")
                    .request();
```

More Annotated POJOs

Thanks to annotations, Java EE has become more about programming with Java objects, and less about configuration. For example, beginning with Java EE 6 the web.xml became optional thanks to the ability to provide metadata with annotations, and annotated EJBs could be packaged with .war files.

Java EE 7 continues the move towards a POJO development model. As shown earlier, a WebSocket is represented by an annotated POJO, exposed externally at @ServerEndpoint and whose @OnOpen, @OnClose, and @OnMessage methods are called by the runtime when the respective event is fired.

JAX-RS 2.0 Interceptors and Filters are defined as POJOs, are annotated with @Provider, and implement specific interfaces. Interceptors and Filters annotated with @Provider are globally enabled for all resources by default – no configuration required. Applying Interceptors and Filters to specific resources is accomplished by using type-safe annotations similar to CDI Qualifiers.

With feedback from the Java EE developer community, CDI is now enabled by default and no longer requires beans.xml file just to use CDI. Developers can simply use @Inject to inject virtually any Java object with no configuration required. This includes the new @JMSDestinationDefinition and @MailSessionDefinition resource annotations, which enable developers to specify resource metadata in source code, simplifying the DevOps experience.

Cohesive, Integrated Platform

Java EE 6 introduced Managed Beans 1.0 as the first step towards aligning EJBs, JSF Managed Beans, and CDI beans. With Java EE 7, managed bean alignment continues. For example, JSF Managed Beans has begun the pruning process in favor of CDI Beans.

Java EE 7 also brings the ease of use of EJB container managed transactions to the platform as a whole, using a more general solution based on CDI interceptors so that these can be used by CDI managed beans and other Java EE components. Applying the @Transactional annotation to any CDI bean or a method of any managed component makes that method transactional.

Bean Validation is more widespread in Java EE 7, and can now be used for method-level validation, including both built-in and custom constraints. Constraints can be applied to method parameters as well as return values. Constraints can also use the Java EE Expression Language for flexible rendering and string formatting of constraint violations.

Bean validation also extends to JAX-RS 2.0. Constraining annotations can be specified in public constructor parameters, method parameters, fields and bean properties. In addition, they can also decorate resource classes, entity parameters, and resource methods. For example, constraints can be applied to JAX-RS method parameters to validate form data submitted through @POST and @PUT.

Simplifying Java EE by Pruning Old Technologies

While many new features have been added in Java EE 7, others have been made optional as older features have been replaced by simpler ones or have simply been removed. Java EE 6 introduced a formal process for deprecating obsolete technologies, and targeted features for eventual pruning -- Java EE Management (JSR-77); Application Deployment (JSR-88); JAXR, for interfacing with UDDI registries (JSR-93); JAX-RPC, for XML-based RPC (JSR-101); and EJB 2.x Container Managed Persistence, which is effectively replaced by the Java Persistence API (JSR-338). These specifications, while removed from the current release, remain optional for vendors in the event that demand for them persists among customers. They will, however, be removed in Java EE 8.

Meeting the Demands of the Enterprise

Java EE has been addressing enterprise demands for over a decade with connectivity to backend systems using the Java Connector Architecture, support for transactions with the Java Transaction Service, and communication between many IT systems using the Java Message Service. Today, enterprises want to leverage their developers' existing Java skills to write batch applications that use a standard API and are portable across multiple runtimes.

Enterprises also need to build highly scalable applications to meet higher demand for their services and to also drive higher utilization of existing assets. Concurrency Utilities for Java EE enable developers to write scalable applications so that they cleanly integrate with the Java EE runtime in a secure, reliable manner.

Greater Efficiency with Batch Applications for the Java Platform

While the vast majority of Java EE applications are online user-driven systems, there is an expanding class of server-side applications that require batch processing -- especially with a renewed need for off-line analytics and ETL (extract, transform, load) tasks. These batch-oriented applications are best suited for non-interactive, bulk-oriented and long-running tasks that are computationally intensive, can execute sequentially or parallel, and may be initiated ad hoc or through scheduling. Batch processing also effectively utilizes computing resources by shifting processing times to when resources are typically idle.

Previously, no standard Java programming model existed for batch applications. Batch Applications for the Java Platform provides such a model and creates a lingua franca for well understood batch processing concerns such as jobs, steps, repositories, the reader-processor-writer pattern, chunking, checkpoints, parallel processing, flow, split, transactions, retries, sequencing and partitioning.

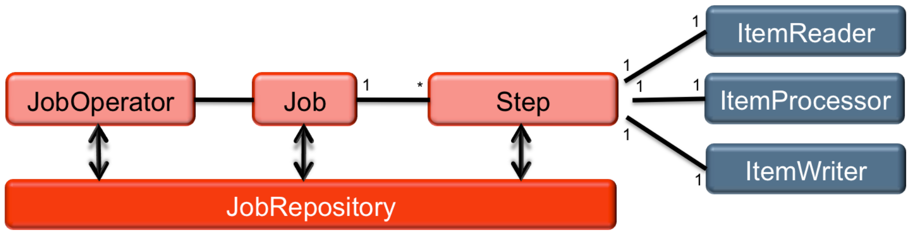
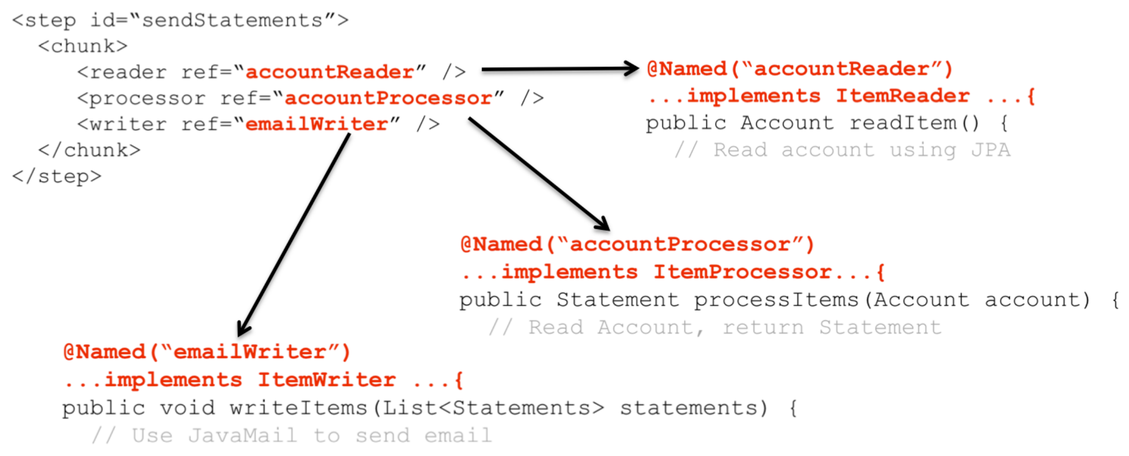


FIGURE 2 - A JOB REPRESENTED AS STEPS

As illustrated in Figure 2, a *job* represents a series of closely related *steps* that, taken together, perform a discrete business process. Steps may be executed in sequence or in parallel. Steps may also be optional, with the decision to execute or skip them conditioned on the outcome of prior steps in the same *workflow*. Steps can be *checkpointed* and retried if needed and are generally transactional. A *repository* stores information about the current jobs, such as the last time a job executed. Jobs can be listed, started, stopped, paused and cancelled through the *operator*. The operator is typically invoked in a scheduled or ad-hoc fashion. The entire batch process is put together through a *Job Specification Language* (JSL) written in XML.

Although the JSR codifies these robust concepts, the programming model is kept very simple as can be seen in the following example:



```
<step id="sendStatements">
  <chunk>
     <reader ref="accountReader" />
     <processor ref="accountProcessor" />
     <writer ref="emailWriter" />
  </chunk>
</step>
```

```
@Named("accountReader")
...implements ItemReader ...{
public Account readItem() {
  // Read account using JPA
```

```
@Named("accountProcessor")
...implements ItemProcessor...{
public Statement processItems(Account account) {
  // Read Account, return Statement
```

```
@Named("emailWriter")
...implements ItemWriter ...{
public void writeItems(List<Statements> statements) {
  // Use JavaMail to send email
```

The example illustrates a simple step to send emailed bank statements as a batch process. The step is composed of a reader, processor, and writer. The *reader* reads the next account, conceivably from a database using JPA. The *processor* processes the account and creates a corresponding statement. The writer conceivably uses JavaMail to email the set of statements. Each chunk is executed within a transaction, and there is also automatic checkpointing.

The JSR allows for batch applications that may execute in either a Java SE or Java EE environment. There may be different qualities of service among each environment -- for example more robust JTA transactions in a Java EE environment.

Simplified Concurrency and Enhanced Portability with Concurrency Utilities for Java EE

Java EE has long provided a rich set of features geared towards asynchronous, parallel, and background task execution such as EJB `@Asynchronous`, Servlet async and JAX-RS async. However, in some cases developers need to use lower level concurrency facilities such as the ones provided by Java SE threads. Java EE discourages the direct use of underlying JVM threads as the container cannot manage important quality of service metrics such as reliability, scalability, and security in unmanaged threads. In the past, these challenges were addressed by vendor-specific services that allowed low-level threads to be executed in a managed fashion through the container. The Concurrency Utilities for Java EE standardizes this concept by providing managed versions of the Java SE threading APIs available in the `java.util.concurrent` package such as `ExecutorService`.

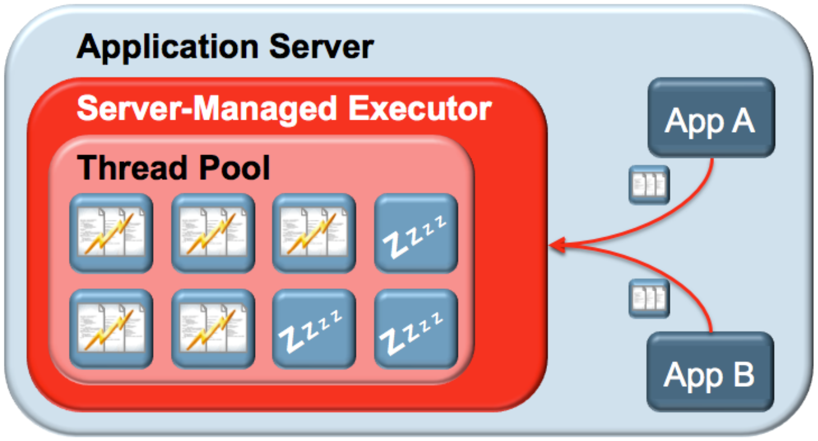


Figure 3. Submitting Tasks to a Server-Managed Executor

As Figure 3 and the following code demonstrate, the API allows developers to write tasks that implement the `java.lang.Runnable or java.util.concurrent.Callable` interfaces and submit the tasks to a managed executor service that guarantees the tasks scalability, utilizing the container thread pool and run inside a reliable security and naming context.

```
public class TestServlet extends HTTPServlet {
  @Resource(name="concurrent/MyExecutorService")




                .
    pub
      ... // Task logic
    }
  }
}
```

Both simple common and advanced concurrency patterns are supported such as scheduled execution, custom thread factories, execution notification, and manual container context management.

Java EE Web Profile Enhancements

The Java Enterprise Edition Web Profile was introduced in Java EE 6, and is targeted at developers of dynamic Web applications. Most Web applications have significant requirements in the areas of transaction management, security, and persistence. Such requirements can be readily addressed by established Java EE technologies such as the Enterprise JavaBeans (EJB) Lite technology, and the Java Persistence API, Java Transaction API, but are not supported by standalone servlet containers. By incorporating many of these APIs, the Web Profile raises the bar for the development of Web applications using the Java platform with pre-installed, pre-integrated, fully tested Web infrastructure features.

The Java EE 7 Web Profile adds support for HTML5 with WebSockets, JSON, JAX-RS 2.0, and more. While the Web Profile is feature rich, it strives for simplicity by leaving out many of the enterprise connectivity APIs that are part of the full Java EE platform. If enterprise connectivity is required at a later date, developers can simply redeploy their applications to a full Java EE platform.

GlassFish Server Open Source Edition 4.0

GlassFish Server Open Source Edition 4.0 is a compatible, production implementation of the Java EE 7 platform specification built using an open source license. As with Java EE 5 and 6, the Reference Implementation (RI) of Java EE 7 is derived from Project GlassFish. As the RI, GlassFish Server is always up to date with the latest Java EE specifications. For developers, GlassFish Server offers a lightweight runtime that starts in seconds, and enables rapid iterative development with Active Redeploy that saves session state when an application is redeployed. For IT Operations, GlassFish Server offers a feature-rich web console for manual operations, and a feature-equivalent command line utility for automated environment. GlassFish Server also features centralized administration and high availability clustering.

GlassFish can be downloaded from http://glassfish.org.

Java EE 7 SDK

For developers that are coming up to speed on Java EE, or simply want to quickly get started on new Java EE 7 features, the Java EE 7 SDK (SDK) is an all-in-one bundle for doing just that. The SDK includes the First Cup Java EE 7 introduction, the full Java EE 7 tutorial, sample applications that can be built with Maven, Java EE 7 javadocs, GlassFish Server Open Source Edition 4.0, and (optionally) JDK 7. The Java EE 7 SDK has been tested with the NetBeans IDE, although the samples should work in any IDE that supports Maven.

The Java EE 7 SDK can be downloaded from http://www.oracle.com/javaee.

Integrated Development Environments

Leading IDEs such as NetBeans and Eclipse can be used to develop applications and other components for Java EE 7. Such IDEs support virtually all the Java EE capabilities described in this paper. NetBeans 7.3.1 and later provide comprehensive support of the Java EE 7 platform and bundles GlassFish so you can get started quickly. NetBeans also includes wizards to rapidly create JAX-RS 2.0

Interceptors and Filters, JSF 2.2 Faces Flow, WebSocket Endpoints, and more. NetBeans can be downloaded from http://www.netbeans.org.

In addition, the Eclipse Kepler release will include support for Java EE 7, and the Oracle Enterprise Pack for Eclipse (OEPE) 12.1.2 in the Eclipse Marketplace hosts the GlassFish plugin. You can learn more about Eclipse and other IDE support from https://glassfishplugins.java.net/.

Conclusion

The Java EE platform offers enterprise developers the opportunity to deliver today's Web applications with the greatest efficiency, flexibility, and ease of development. After 13 years offering business-critical applications for thousands of companies, Java EE remains ahead of the pack as an enterprise application and deployment platform. As the industry standard for enterprise computing, Java EE enables developers to take advantage of the emerging usages, patterns, frameworks, and technologies of the enterprise space.

Developing enterprise applications has never been easier.

Appendix 1: References

Java EE 7 contains 14 new and updated JSRs. Java specifications are available at http://www.jcp.org.

- JSR 236: Concurrency Utilities for Java EE 1.0
- JSR 338: Java Persistence API 2.1
- JSR 339: Java API for RESTful Web Services 2.0
- JSR 340: Java Servlet 3.1
- JSR 341: Expression Language 3.0
- JSR 342: Java Platform, Enterprise Edition 7
- JSR 343: Java Message Service 2.0
- JSR 344: JavaServer Faces 2.2
- JSR 345: Enterprise JavaBeans 3.2
- JSR 346: Contexts and Dependency Injection for Java EE 1.1
- JSR 349: Bean Validation 1.1
- JSR 352: Batch Applications for the Java Platform 1.0
- JSR 353: Java API for JSON Processing 1.0
- JSR 356: Java API for WebSocket 1.0

Oracle is committed to developing practices and products that help protect the environment

White Paper Title
June 2013

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200

oracle.com

Copyright © 2013, Oracle and/or its affiliates. All rights reserved.

This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. 0113

Hardware and Software, Engineered to Work Together