Java EE: Wohin die Reise geht

Peter Doschkinow, Wolfgang Weigend
ORACLE Deutschland B.V. & Co. KG
München

Schlüsselworte:
Java EE, Java EE 6, PaaS

Einleitung

Java EE 6 hat eine Reihe von neuen Features und Technologien mit sich gebracht, die die Java EE Plattform flexibler machen und die Erstellung und den Betrieb von standards-basierten Web-Anwendungen wesentlich vereinfachen. Java EE 6 wird heute auch deshalb schneller als sonst angenommen, weil sich in vielen Fällen der Einsatz von Third-Party-Frameworks wie Spring als überflüssig erweist. Obwohl die Java EE eine gewisse Unterstützung für Virtualisierung anbietet (Container, Security, Ressourcen-Management), mehr ist nötig, um die Plattform für Cloud-Umgebungen salonfähig zu machen. Genau das ist das Hauptthema von Java EE 7: eine Evolution in Richtung Cloud-Computing, neben weitere Verbesserungen der Entwickler-Produktivität. Das bedeutet, dass die Java EE versionierte, gut von einander isolierte und multimandanten-fähige Anwendungen unterstützen und eine Standardisierung von Management- und Monitoring Interfaces vornehmen muss. Neue effektive Web-Technologien wie HTML5, WebSockets und Not-Only-SQL DBMS müssen berücksichtigt werden. Java EE 7 wird sich auch um die Bereinigung und das bessere Zusammenspiel der beteiligten Spezifikationen und Komponenten-Modelle kümmern.

Java EE Evolution

Die Java EE Plattform ist seit mehr als zehn Jahren erfolgreich auf dem Markt und bietet für tausenden von Firmen eine standardisierte Anwendungsplattform, die die Entwickler in komplexen Infrastruktur-Themen wie Support für Transaktionen, Persistenz, Web Services, Messaging und vieles mehr unterstützt und ihnen ermöglicht, sich auf die Business-Logik und die Mehrwerte ihrer Anwendung zu konzentrieren:

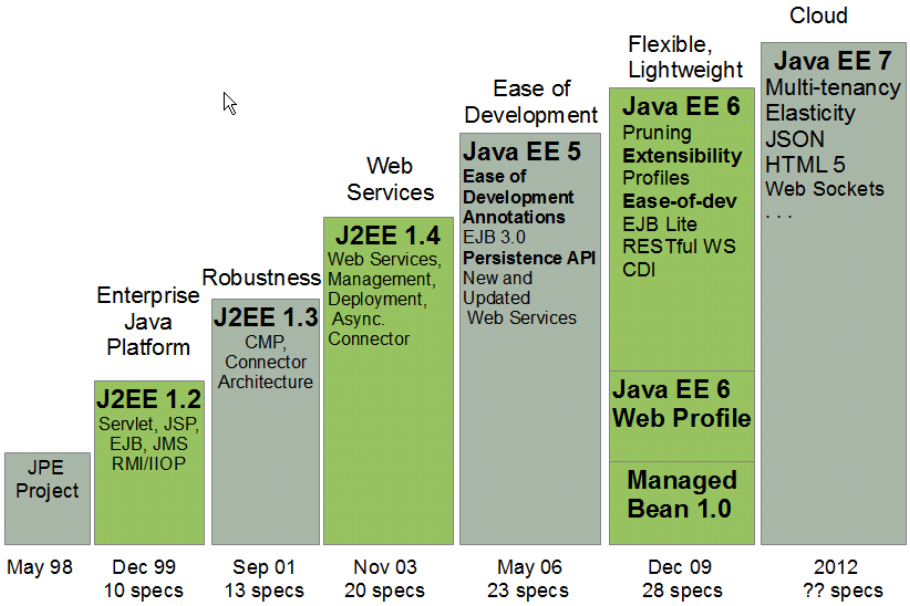

Abb. 1: Java EE Evolution

Am Anfang wurden die meisten Bemühungen auf den Ausbau der Plattform gerichtet, bis sie mit J2EE 1.4 funktional fast vollständig war. Bedauerlicherweise ist sie aber dabei auch extrem komplex und schwierig zu benutzen geworden. Mit Java EE 5 wurde zum ersten mal das Thema Entwickler-Produktivität adressiert und in den Mittelpunkt gerückt. Die Verwendung von Annotationen, die Vereinfachung vom EJB-Modell und seine Umstellung auf POJOs, sowie die Einführung von JPA haben hierzu eine entscheidende Rolle gespielt. Mit Java EE 6 wurde dieser Trend fortgesetzt, jedoch die Akzente wurden jetzt auf die Flexibilität und die Leichtgewichtigkeit der Plattform gesetzt, in erster Linie durch die Einführung von Profiles. Entsprechend der Notwendigkeit und der Nachfrage nach Unterstützung für Cloud-Computing, wird der Haupt-Fokus für Java EE 7 die Anpassung der Plattform und ihre Überführung in eine PaaS, sowie eine Harmonisierung der beteiligten API.

Die Haupt-Themen von Java EE 6
Die neuen Features und API-Modifikationen in Java EE 6 lassen sich in drei Kategorien unterteilen: Flexibilität und Leichtgewichtigkeit, Erweiterbarkeit und Entwickler-Produktivität.

Flexibilität und Leichtgewichtigkeit
Flexibilität und Leichtgewichtigkeit wird in erster Linie durch die Einführung von dem Konzept von Profiles herbeigeführt. Profiles sind definierte Untermengen von Java EE 6 API, die für eine bestimmte Klasse von Anwendungen vorgesehen sind. So gibt es bei Java EE 6 neben dem Full-Profile, der Gesamtheit aller Java EE 6 Teilspezifikationen, das Web-Profile. Das Web-Profile umfasst nur die API, die typischerweise für die Erstellung und Betrieb von modernen Web-Anwendungen ausreichen und gut aufeinander abgestimmt sind, nicht alles was eine Java EE Anwendung benötigen könnte. Das führt dazu, dass eine Web-Profile Implementierung einen geringeren Footprint in Bezug auf Speicher und Rechen-Ressourcen hat und deutlich schlanker als die Gesamtplattform ist. Ein Entwickler oder Kunde kann leicht und schlank mit dem Web-Profile anfangen und später, wenn er Funktionalitäten benötigt, die nicht im Web-Profile enthalten sind, auf das Full-Profile umsteigen.

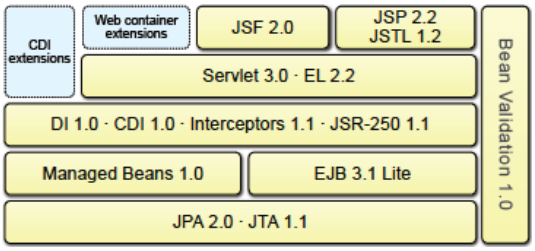

Abb. 2: Java EE 6 Web Profile

Ein neuer organisatorischer Mechanismus, das Pruning, wird künftig dafür sorgen, dass Spezifikationen und API, die sich nicht durchgesetzt haben oder durch modernere überholt wurden, als optional bestimmt werden können, so dass sie Java EE Hersteller nicht implementieren müssen und dadurch die Plattform entlastet wird.

Erweiterbarkeit
Ein wichtiges Design-Ziel von Java EE 6 war vom Anfang an die Erweiterbarkeit der Plattform, um mit der fortlaufenden Innovation im Umfeld von Web-Anwendungsframeworks Schritt halten zu können. Das Konzept vom web-fragment.xml sieht vor, dass Web-Frameworks, die in einer Java EE 6 Umgebung eingesetzt werden sollen, alle benötigten Artefakte wie Servlets, Listener, Filter und Context-Parameter in einem eigenen web-fragment.xml registrieren und zusammen mit dem Code

und den benötigten Ressourcen in einen autonomen, in sich geschlossenen und wiederverwendbaren Jar-Archiv packen. Dadurch ist es möglich Web-Frameworks durch Drag-and-Drop in das standard WEB-INF/lib Verzeichnis in Web-Anwendungen bereitzustellen. Weitere mächtige Mittel zur Erweiterung der Java EE 6 Plattform bieten die neuen Servlet- und Context and Dependency Injection (CDI) API.

Entwickler-Produktivität
Die Verwendung von Annotationen in Java EE 5 wurde in Java EE 6 konsequent weitergeführt. Die Servlet- und Java Connection Architecture API wurden komplett für die Verwendung von Annotationen angepasst, die Annotationen für JPA wurden erweitert und bei den neuen Spezifikationen wie Managed Beans, Dependency Injection und CDI kamen sie out-of-the-box. Obwohl Annotationen durch entsprechende XML-Elemente in Deployment-Deskriptoren überschrieben werden können, in vielen Fällen können nun Java EE Anwendungen komplett ohne Deployment-Deskriptoren auskommen.

Durch die Einführung von Managed Beans wurde zum ersten mal ein POJO basiertes und anwendungszentrisches Komponenten-Modell in Java EE eingeführt, analog zu Java Beans in Java SE. Managed Beans sind POJO mit Unterstützung von einer minimalen Anzahl von Diensten: Lebenszyklus-Verwaltung und Injektion. Als Entwickler kann man nun für seine Business-Objekte zunächst Managed Beans verwenden. Wenn man dann später mehr Dienste vom Container für sie benötigt, wie z.B. Support für Transaktionen oder SOAP-, bzw. REST Web Service Interfaces, fordert man sie einfach an durch das Hinzufügen von den entsprechenden Annotationen.

CDI nimmt dem Entwickler die Zustandsverwaltung seiner Anwendung ab und überträgt sie dem Container. Außerdem kann man durch CDI Annotationen und API den Container veranlassen, auf eine typensichere und lose gekoppelte Weise ein ganzes Netzwerk voneinander abhängiger Objekte zu erzeugen.

Marktverbreitung
Im Vergleich zu früheren Java EE Spezifikationen hat sich Java EE 6 rasch auf dem Markt zum Vorteil von Entwicklern, Herstellern und Kunden verbreitet. Neben GlassFish gibt es momentan Java EE 6 Full-Profile Implementierungen von IBM (WebSpere 8.0), Fujitsu (Fujitsu Interstage Application Server) und TmaxSoft (TMAX JEUS 7), sowie Web-Profile Implementierungen von RedHat (JBoss AS7) und Caucho (Caucho Resin 4.0.17). Weitere Hersteller haben auch in naher Zukunft Java EE 6 Support angekündigt – Oracle (WebLogic), Apache (Geronimo), MechSoft (Siwpas). Die Folge davon ist, dass Kunden durch die Wahl der Java EE 6 als Anwendungs-Plattform von ihren Stärken und hohen Produktivitätsverbesserungen profitieren, ohne sich in eine Hersteller-Bindung zu begeben, während Entwickler von Java EE 6 Anwendungen eine Vielzahl von Java EE 6 Application Server Implementierungen adressieren können.

Java EE 7
Das Hauptthema von Java EE 7 ist die verbesserte Unterstützung für Cloud-Umgebungen mit dem Ziel die Java EE Plattform nach dem PaaS Model auszurichten. Das würde dazu führen, dass Anwendungen, die für einen Kunden auf Java EE 7 PaaS installiert werden, neben solchen für andere Kunden laufen würden und von den typischen Vorteilen des Cloud-Computing wie wirtschaftlichere Ressourcennutzung, Elastizität, garantierte QoS, etc. profitieren können. Cloud-Anbieter wären in der Lage eine standardisierte Plattform effizienter und kostengünstiger zu betreiben.
Was bei Java EE 7 sicher beachtet werden muss sind die neuen Herausforderungen, die sich generell für den Betrieb von Anwendungen auf einer PaaS ergeben. Das heißt, dass die Java EE als PaaS mandantenfähige Anwendungen unterstützen muss, die besser von einander getrennt sind, damit sie

keine Auswirkungen auf andere parallel laufende haben und nur die Daten ihrer Mandanten sehen und manipulieren können. Für alle Ressource-Manager API wie JPA, JDBC, JMS, JCA bedeutet dies, dass sie aufgearbeitet werden müssen, um gleichzeitig und sicher von mehreren Anwendungen und Mandanten benutzt werden zu können. Es wäre dann die Aufgabe vom Container einzelne Anwendungsequests auf Mandanten abzubilden und ihre Identität über verschiedene Ressource-Manager und Tiers zu propagieren.

Das folgende Bild stellt die angestrebte High-Level Architektur von Java EE 7 als kontrollierte Cloud-Plattform PaaS dar.

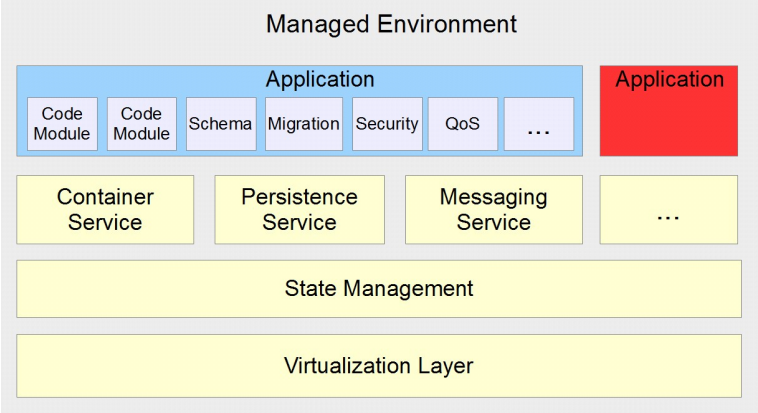

Abb. 3: Java EE 7 als PaaS

Die Basis bildet die Virtualisierungsschicht, die neu ist und beim Deployment, Betrieb, Management und Monitoring der Laufzeitumgebung zum Einsatz kommt. Sie abstrahiert die Schnittstellen zu Virtualisierungsprovider wie Xen, KVM, VirtualBox und öffentliche Cloudprovider wie Amazon EC2 oder Microsoft Azure. Außerdem standardisiert sie den Umgang mit Application Server Cluster und passt ihn an die Cloud-Umgebung an.

In der darüber liegenden Schicht werden der Zustand und die Auslastung der Anwendungen und der Application Server Instanzen überwacht und verwaltet. Hier werden bei Bedarf Virtual Machines(VM) instantiiert, provisioniert und dem Cluster hinzugefügt, sowie Anwendungszustände repliziert.

Die Funktionalität der Java EE Plattform manifestiert sich in der Services-Schicht. Zu ihr gehören Dienste wie JavaEE Container(für EJB, Servlets, CDI), Message-Queue- und Persistence-Services. Sie werden mehr abgekoppelt und können gegebenenfalls sogar in einer eigenen VM laufen.

Die oberste Schicht bilden die Anwendungen. Der JSR-342 sieht vor, dass Anwendungen Metadaten in einem erweiterbaren Metadata-Deskriptor mit sich führen. Darin bestimmen sie explizit, ob sie für eine PaaS-Umgebung geeignet sind, und wenn ja, welche Anforderungen sie an die PaaS haben. Zum Beispiel ob sie mandantenfähig sind, Elastizitätsanforderungen wie minimale/maximale Anzahl von Instanzen, Sicherheits- und QoS-Informationen, Abhängigkeiten von anderen Anwendungen, Ressourcen und Services und ob und welche Dienste sie selbst anbieten. Die Plattform wird die Versionierung von Anwendungen und die Koexistenz mehrerer Versionen der gleichen Anwendung unterstützen. Die rot gefärbte Anwendung soll verdeutlichen, dass es durchaus Anwendungen mit besonders hohen Sicherheitsansprüchen geben kann, die keine Dienste und Ressourcen mit anderen teilen wollen und komplett abgeschottet sein sollen.

Modularität
Der Bedarf an Modularität für Java EE Anwendungen war schon immer groß, zumal sie oft mit ihrem

Footprint denjenigen vom zugrundeliegenden Application Server übertreffen. Da Java EE 7 auf Java SE 7 aufsetzt und die Modularisierung der Java Plattform auf Java SE 8 verschoben wurde, wird es leider keine echte standardisierte Modularisierung für Java EE vor Java EE 8 geben. Plattform-Anbieter, die in der Architektur von ihrem Application Server eine Abstraktionsschicht über dem eigenen Modulsystem vorgesehen haben(wie das HK2-Modulsystem über OSGi in GlassFish) werden künftig vermutlich leichter ihre Produkte auf die Modularität von Java SE 8 umstellen können. Auch wenn Java EE 7 noch keine Modularisierung anbieten wird, so wird zumindest angestrebt, sich darauf vorzubereiten. Zum Beispiel durch die Verwendung von Metadaten, die Abhängigkeiten und Versionen explizit beschreiben, und durch Modifikationen der Classloader und der Art ihrer Verwendung. Das Ziel ist schon mit Java EE 7 typische Anwendungsfälle der Nutzung von Modularität zu unterstützen. So ein Fall wäre der Support für Anwendungen, die aus versionierten Modulen bestehen.

Unterstützung der neuesten Web-Standards
Um mit den neuen technologischen Entwicklungen Schritt zu halten und die Wettbewerbsfähigkeit der Plattform zu erhöhen, wird die Unterstützung für eine Reihe von neu entstehenden Web-Standards wie WebSockets und HTML5 angestrebt. Offensichtlich ist JSF die natürliche Heimat für den HTML5-Support. Im März wurde der JSR-344 für JSF 2.2 gestartet, in dem HTML5 Funktionalitäten wie HTML5 Forms und einige der Inhaltsmodelle der HTML5 Elemente unterstützt werden sollen.
Mit der wachsenden Verbreitung von NoSQL Architekturen im Web soll auch das Thema der Abbildung von Java-Modellen auf nicht-relationale Datenbanken untersucht werden. Eine Erweiterung des JPA API erscheint auf Grund seiner starken Abhängigkeit von relationalen Datenbanken nicht als sinnvoll, so dass vermutlich ein neues API zur Anbindung von NoSQL Backend-Systemen notwendig sein wird.

Vorläufiger Inhalt
Wenn man den beabsichtigten Umfang von Java EE 7 nach aktuellem JSR-Status kategorisiert, ergibt sich folgende Tabelle:

Beantragt und abgestimmt	Noch zu beantragen	Andere (wieder aufgenommen, kleine Änderungen)
JPA 2.1 – JSR 338	DI 1.1	JCache – JSR 107
JAX-RS 2.0 – JSR 339	JSON 1.0	Concurrency Utilities – JSR 236
Servlets 3.1 – JSR 340		Common Annotations 1.2
EL 3.0 – JSR 341		JAX-WS 2.3
Java EE 7 – JSR 342		JTA 1.2
JMS 2.0 – JSR 343		JSP 2.3
JSF 2.2 – JSR 344		Connector 1.7
EJB 3.2 – JSR 345		
CDI 1.1 – JSR 346		
Data Grids – JSR 347		
Bean Validation 1.1 - JSR 349		
Java State Management – JSR 350		

Abb. 4: Vorläufiger Inhalt von Java EE 7

Manche der API haben im Fokus die Entwicklung zu vereinfachen. Dazu zählt das JMS 2.0 API, das komplett modernisiert wird. Geplant ist die Einführung von Annotations, ggf. den Übergang zu „Connectionless API“ (um das übliche Bootstrapping ConnectionFactory → Connection → Session → … zu eliminieren) und die Standardisierung der Integration mit Application Server und verbreiteter JMS Hersteller-Erweiterungen. JAX-RS, das serverseitige API für REST-basierte Architekturen, wird in seiner neuen Version 2.0 zwei Client API definieren, ein Low-Level API unter Verwendung vom Factory-Pattern und ein High-Level API, das auf das erste aufsetzt. Weiterhin wird JAX-RS 2.0 eine MVC Architektur spezifizieren, die mit dem JAX-RS Programmiermodel kompatibel ist und mit verschiedenen Viewing-Technologien wie JSP, FreeMaker oder StringTemplate zusammenarbeiten kann. Eine gute Ergänzung für JAX-RS wird die neue in Betracht gezogene JSON API. Weiterhin in Richtung Vereinfachung der Entwicklung ist die Überlegung, wie das Managed Bean Model verfeinert und erweitert werden kann, um die Überlappungen und Inkonsistenzen unter Managed Beans, EJB, Servlets, JSF, CDI und JAX-RS zu eliminieren.

Andere der API werden die Funktionalität der Plattform deutlich erweitern. JCache zum Beispiel standardisiert das Caching von Java Objekten, auf die von mehreren Threads im laufenden Prozess zugegriffen wird. Das Caching kann erheblich die Performance und Skalierbarkeit von Web Anwendungen erhöhen. Es wird gerade diskutiert, wie die Funktionalität des JCache API durch das neu angenommene Data Grids API auf verteilte Caches mit Unterstützung für Transaktionen und asynchrone Zugriffe ausgedehnt werden kann. Das würde Applicatin Server Cluster und ihre Elastizitätsfähigkeit mit einem separaten hochverfügbaren Cache-Tier ergänzen. Das Concurrency Utilities for Java EE API wird auf dem java.util.concurrent API von Java SE basieren und Nebenläufigkeit auf der Anwendungsebene in der kontrollierten Umgebung von Java EE ermöglichen.

Ausblick
Für Java EE 7 hat die Reise in die Wolken begonnen. Sie wird sicherlich spannend und nicht leicht sein. Auf der einen Seite sind die Inhalte umfangreich, anspruchsvoll und die Wunschliste geforderter Funktionalitäten ist lang. Auf der anderen Seite gibt es viele Produkte auf dem Markt die als Vorbild und Anregung bei der Standardisierung dienen können. So kann TopLink als Beispiel genommen werden, wie die Unterstützung von Mandantenfähigkeit bei relationalen Datenbank-Ressourcen (Erweiterungen für mandantenspezfische Entity-Customization, @Multitenant Annotation) und die Anbindung an Data-Grids und NoSQL-DBMS (Interceptoren zur Anbindung an Coherence oder NoSQL-Ressourcen) implementiert werden kann. Da die Fertigstellung der Spezifikation auf das Ende nächsten Jahres terminiert wurde, ist es wahrscheinlich, dass einige ihrer angestrebten Features aus Zeitgründen auf die nächste Java EE 8 Version verschoben werden müssen.

Es ist noch möglich, sich an der Entwicklung der Java EE 7 Plattform oder der einzelnen API federführend zu beteiligen. Voraussetzung ist die Registrierung als JCP Member und die Unterzeichnung eines Java Specification Participation Agreement (JSPA). Unter http://jcp.org/en/participation/membership ist der Prozess genau beschrieben. Anregungen und Feedback sind auf den Mailing-Listen der Java EE 7 Projektseite willkommen.

Kontaktadresse:
Peter Doschkinow, Wolfgang Weigend
ORACLE Deutschland B.V. & Co. KG
Riesstr. 25
D-80992 München
Telefon: +49 (0) 1802672253
Fax: +49 (0) 1802672329
E-Mail peter.doschkinow@oracle.com, wolfgang.weigend@oracle.com
Internet: www.oracle.com