Java EE 7: auf halber Strecke

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

Schlüsselworte:
Java EE, Java EE 7, PaaS

Einleitung
Die Java EE 7 Plattform Spezifikation ist seit ihrer einstimmigen Annahme in März 2011 durch das Executive Committee für Java SE/EE in Arbeit. Ihre Fertigstellung ist für die Mitte des nächsten Jahres geplant. Was wurde in der Zwischenzeit erreicht? Wie wurden die anspruchsvollen Ziele zur Unterstützung einer PaaS und der weiteren Verbesserung und Harmonisierung der Java EE API adressiert? Worauf muss verzichtet werden? Die Antworten auf diesen Fragen, die bereits in den frühen Draft-Versionen der Java EE 7 selbst(JSR-342) und der einzelnen Teilspezifikationen enthalten sind, werden im Vortrag zusammengefasst. Es werden auch die Fortschritte in der Java EE 7 Referenzimplementierung GlassFish 4.0 demonstriert.

Die Evolution von Java EE in Richtung einer PaaS
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. Der Höhepunkt der Java EE Entwicklung ist die aktuelle Version Java EE 6. Ihr Riesenerfolg ist in der deutlich erhöhten Produktivität, Flexibilität und Verschlankung der Plattform begründet und in zahlreichen(nach heutigem Stand 15) Implementierungen sichtbar (http://www.oracle.com/technetwork/java/javaee/overview/compatibility-jsp-136984.html).

Mit den Fortschritten von Virtualisierungs- und Clustering-Technologien und der Entstehung von verschiedenen Cloud-Service Modellen wie IaaS, PaaS und SaaS hat sich natürlich die Frage gestellt, wie die Java EE Plattform für Cloud-Umgebungen angepasst werden kann, damit ihre Benutzer von den typischen Vorteilen des Cloud-Computing wie wirtschaftlichere Ressourcennutzung, Elastizität, garantierte QoS, etc. profitieren können. Die Java EE abstrahiert die zugrunde liegende Runtime, bietet über standardisierte API diverse Services an und als Anwendungsplattform ähnelt am meisten einer PaaS. Was sie von einer PaaS unterscheidet ist, dass sie in erster Linie die Bedürfnisse der Entwickler addressiert. Die Bedürfnisse der Betreiber bleiben außen vor. Sie müssen sich selber darum kümmern die Dienste, die eine Java EE Anwendung benötigt, zu provisionieren und zu konfigurieren und für ihre Skalierbarkeit, Hochverfügbarkeit und Sicherheit zu sorgen. Eine PaaS adressiert per Definition die Anforderungen von Entwickler und Betreiber. Auch wenn die ersten PaaS auf dem Markt nicht für Java-Anwendungen ausgelegt waren, so häufen sich in letzter Zeit die Angebote für Java- und insbesondere Java EE-Anwendungen. Zu den Anbietern gehören

- klassische Middleware Hersteller wie Oracle(Java Cloud Service), Microsoft(Azure), RedHat(OpenShift)
- Cloud-Services Anbieter wie Amazon(Beanstalk), Google(AppEngine)
- innovative Start-Ups wie Jelastic und CloudBees

Auffälig ist, dass diese Angebote so unterschiedlich in Bezug auf ihre funktionale und nicht-funktionale Merkmale sind, dass auch abgesehen von ihren Zahlungs-Modellen ein Vergleich kaum möglich ist.

Vor diesem Hintergrund ist es leicht nachvollziehbar, dass der Wunsch nach der Standardisierung einer PaaS für Java EE Anwendungen in der Java EE Community stark war. Deshalb wurde im JSR-342 die Anpassung der Java EE Plattform und ihre Überführung in eine PaaS als Haupt-Fokus für Java EE 7 festgelegt.

Die Java EE 7 Expert Group hat recht früh das neue Rollen-Modell der Java EE PaaS in einem Dokument aufgestellt und es darin anhand von typischen Anwendungsfällen erläutert.

Im nächsten Schritt gab es Überlegungen, neue Metadaten für Java EE Anwendungen einzuführen, die sich auf die von ihnen benutzten Services beziehen. Diese Metadaten sollten die schon spezifizierten Annotationen für Java EE Ressourcen ergänzen, so dass der Java EE PaaS Provider genug Informationen hat, um alle Services, auf die eine Java EE Anwendung angewiesen ist, zum Zeitpunkt ihrer Installation zu provisionieren, konfigurieren und letztendlich der Anwendung zur Verfügung zu stellen. Es sollte Metadaten geben, die die verschiedenen Eigenschaften der Services beschreiben, wie z.B. parametrisierte Verbindungsdaten, ihre Qualität im Sinne von Hochverfügbarkeit, Elastizität und ob sie exklusiv oder auch von anderen Anwendungen benutzt werden können. Es würde auch sinnvolle Default-Werte für Services geben, für die keine Metadaten im Anwendungsarchiv vorliegen, aber die offensichtlich benötigt werden. Überhaupt das Konzept von einem Service bekommt bei einer Java EE PaaS eine neue Dimension, weil er nicht nur über API für den Entwickler zugänglich ist, sondern auch vom IT-Betrieb in seinem gesamten Lebenszyklus verwaltet werden muss. Die gesamte Plattform kann somit als eine Sammlung von Managed Services angesehen werden: Java EE Service, JMS-Broker-Service, Database-Service, Load-Balancer-Service etc. Um Ressourcen effizienter nutzen zu können und eine höhere Computing-Dichte zu erreichen, wird eine Java EE PaaS sicherlich auf einer Virtualisierungs-Service-Schicht aufsetzen, die über Service-Provider Interfaces(SPI) den Umgang mit unterschiedlichen Virtual Machine Hypervisor abstrahiert. Somit kristallisieren folgende Service Schichten in der Architektur einer Java EE PaaS heraus: Infrastruktur-Services, PaaS-Level Services und die klassischen Java EE Level Services, wie in dem folgenden Bild verdeutlicht:

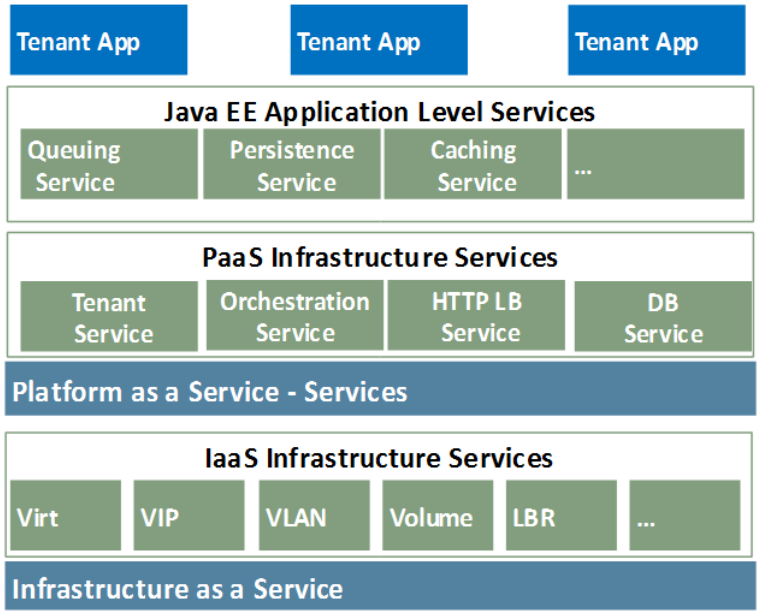

Abb. 1: Java EE PaaS Service Architecture

Vom Anfang an gab es Bestrebungen, Java EE Anwendungen Mandanten-fähig zu machen. Es war zunächst klar, dass für jeden Mandanten eine separate, teilweise konfigurierbare Anwendungs-Instanz

laufen würde, ggf. im selben Java EE Container. Deshalb war die Absicht verbreitet, eine eingeschränkten Unterstützung des SaaS Modells anzubieten. Für alle Ressource-Manager API wie JPA, JDBC, JMS, JCA hätte dies zur Folge, 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 Anwendungsrequests auf Mandanten abzubilden und ihre Identität über verschiedene Ressource-Manager und Tiers zu propagieren.

Verschiebung der Unterstützung für PaaS auf Java EE 8
Aus heutiger Sicht(August 2012) ist der Stand der Spezifikation der PaaS in Java EE 7 recht mager:

- Rollendefinitionen und ihre Klarstellung anhand von Use Cases
- Ansätze der Spezifikation von Metadaten für Services im Sinne von Ressource-Definition und Konfiguration(DataSourceDefinition, JMSConnectionFactoryDefinition, JMSDestinationDefinition, MailSessionDefinition, ConnectorResourceDefinition)
- Anforderung zur Bereitstellung von Default-Services/Resources(java:comp/defaultDataSource, java:comp/defaultJMSConnectionFactory)
- Spezifikation der Bereitstellung vom Mandanten-Identifier(java:comp/tenantId)

Deshalb wurde der Vorschlag vom Java EE 7 Spec-Lead Linda DeMichiel Ende August, die Unterstützung für PaaS auf Java EE 8 zu verschieben, von der Expert-Group mit Erleichterung aufgenommen.

Der Hauptgrund dafür ist die noch fehlenden Erfahrungen im Umgang mit Provisionierung, Mandantenfähigkeit, Elastizität und Deployment von Anwendungen in Cloud-Umgebungen. Im Gegensatz zu der Vorgehensweise bei den frühen Ausgaben von Java EE, als zunächst spezifiziert und dann implementiert wurde, soll das Thema der Standardisierung der PaaS konservativ behandelt werden. Erst werden gut funktionierende Implementierungen abgewartet, dann werden die Best-Practices konsolidiert und in einen Standard gegossen. Ähnlich wie bei der JPA-Spezifikation, in deren Vorfeld verschiedene Persistenz-Lösungen wie Hibernate und TopLink gereift haben. An der Strategie, Java EE in Richtung PaaS zu bewegen, ändert sich nichts. Nur der Zeitpunkt verschiebt sich, um eine bessere und allgemein akzeptierte Standardisierung zu erreichen und den Zeitplan für Java EE 7 mit allen ihren Teilspezifikationen einhalten zu können.

Ein weiterer Grund für diese Entscheidung ist eine andere Initiative für die Programmiersprachen-unabhängige Standardisierung von Anwendungs-Management in PaaS-Umgebungen mit dem Ziel, eine Interoperabilität der Self-Service Interfaces der PaaS-Anbieter herbeizuführen. Sie wurde im August unter dem Namen CAMP(Cloud Application Management for Platforms) ins Leben gerufen und wird von Oracle, Red Hat, CloudBees, Cloudsoft, Huawei, Rackspace und der Software AG unterstützt. Die initiale Spezifikation wurde bereits OASIS zur Weiterentwicklung vorgelegt. Wie Jevgeni Kabanov, Mitglied der Java EE 7 Expert Group, in einer Email zur Unterstützung der Verschiebung des PaaS-Thema auf Java EE 8 anmerkt, gibt es momentan viele Cloud-Aktivitäten, aber welche Technologien, Methodologien oder Philosophien sich dabei durchsetzten, wird die Zukunft vielleicht erst in 2-3 Jahren zeigen.

Die Java EE 7 Teilspezifikationen

Ein wichtiges Ziel von Java EE 7 ist die weitere Vereinfachung der Plattform APIs und ihre Nutzung, ein Trend, der in Java EE 5 angefangen und in Java EE 6 sehr erfolgreich fortgesetzt wurde. Beispiele für die Umsetzung dieser Absicht sind die Vereinfachungen im JMS 2.0 API, die Ausdehnung der Benutzung von CDI wie in JSF 2.2 und Servlet 3.1 und die neuen Service Metadaten, wie im obigen Abschnitt beschrieben. Weiterhin, eine Harmonisierung der API von Java EE 6 ist notwendig. CDI 1.0 wurde z.B. sehr spät im Prozess der Java EE 6 Spezifikation aufgenommen und deshalb wurde von

vielen anderen Teilspezifikationen nicht berücksichtigt. Das unschöne Ergebnis ist, dass wir uns mit verschiedenen ManagedBeans in JSF 2.1, ManagedBeans 1.0 und CDI 1.0 mit unterschiedlicher und teilweise überlappender Semantik auseinandersetzen müssen. Momentan wird gerade eine Diskussion auf der Mailing-Liste der Java EE 7 Expert Group geführt, wo überall noch CDI als Injektion-Mechanismus übernommen werden kann. Direkt angesprochen wird JAX-RS 2.0, das seinen Context mit @Context injiziert. Andererseits soll JAX-RS 2.0 auch alleinstehend, ohne CDI eingesetzt werden können, was gegen ein @Inject spricht.

Im Early-Draft von Java EE 7 ist bereits das Pruning, d.h. das optionale Entfernen von veralteten und überholten Spezifikationen für folgende API festgeschrieben: EJB Entity Beans, JAX-RPC, JAXR und Deployment. Zur Freude der Entwickler wird in der Vorversion vom Web Profile, neben den Abgleich mit den entsprechenden Versionen von Java EE 7 Technologien, die Aufnahme des JAX-RS API gefordert.

Auf dem folgenden Bild sind die unveränderten(rosa), überarbeiteten(blau), die massiv überarbeiteten(orange) und die neuen, noch aufzunehmenden(grün) API der Java EE 7 Plattform dargestellt:

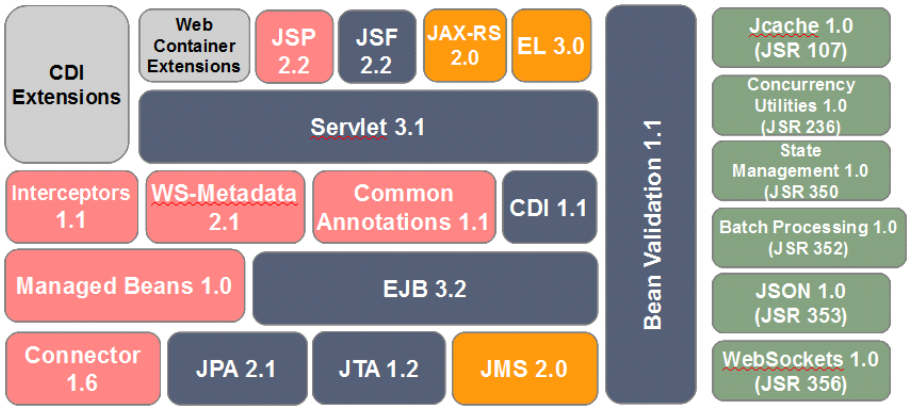

Abb. 2: Java EE 7 neue und modifizierte API

Die spannendsten Neuerungen stehen im Zusammenhang mit den neuen HTML5 Web Technologien und dazu zählen JAX-RS, WebSockets, JSON, JSF, Servlet. Zu der Liste von lang ersehnten Features zählen noch die ausgedehnte Integration von Bean Validation 1.1 in die Plattform mit der neuen Möglichkeit der Method-Validierung und das radikal vereinfachte JMS 2.0 API.

Da eine detailliertere Behandlung der Neuigkeiten in den Java EE 7 Teilspezifikationen den Rahmen dieses Artikels sprengen würde, wird in der folgenden Tabelle versucht, die wichtigsten Informationen der jeweiligen API mit einem Link auf den entsprechenden Projekt-Seiten(Spezifikation, ggf. Referenzimplementierung) zusammenzufassen:

API /JSR Nummer	Spec Lead	Status/Features	Projekt-Seiten
Java EE 7/342	Linda DeMichiel (Oracle) Bill Shannon (Oracle)	Early Draft Support für HTML5 Standards Ease of Development API Harmonisierung	http://javaee-spec.java.net http://glassfish.java.net

API /JSR Nummer	Spec Lead	Status/Features	Projekt-Seiten
JPA 2.1/338	Linda DeMichiel (Oracle)	Early Draft Stored procedures Bulk update/delete with Criteria User-defined functions	http://jpa-spec.java.net http://www.eclipse.org/eclipselink/
JMS 2.0/343	Nigel Deakin (Oracle)	Early Draft New simplified API JMS provider to implement P2P and Pub-Sub Many clarifications	http://jms-spec.java.net http://mq.java.net
JTA 1.2/907	Paul Parkinson (Oracle)	In process	http://jta-spec.java.net/
EJB 3.2/345	Marina Vatkina (Oracle)	Early Draft Alignment with JMS 2.0 More features for EJB 3.2 Lite Cautious usage of Java I/O allowed	http://ejb-spec.java.net http://glassfish.java.net
CDI 1.1/346	Pete Muir (RedHat)	Early Draft Stand-alone usage Global ordering of interceptors/decorators	http://cdi-spec.java.net https://github.com/jboss/cdi/wiki
Servlet 3.1/340	Shing Wai Chan (Oracle) Rajiv Mordani (Oracle)	Early Draft More support for async processing File upload clarifications	http://servlet-spec.java.net http://glassfish.java.net
JAX-RS 2.0/339	Santiago Pericas-Geertsen (Oracle) Marek Potociar (Oracle)	Early Draft Client-side API Filters and Interceptors Async support	http://jax-rs-spec.java.net http://jersey.java.net
JSF 2.2/344	Edward Burns (Oracle)	Early Draft CDI integration, OSGi support better Portlet integration HTML5 Forms, Page Actions	http://jsf-spec.java.net http://jsf.java.net
EL 3.0/341	Kin-man Chung(Oracle)	Early Draft Operations on collections Lambda expressions	http://el-spec.java.net
Bean Validation 1.1/349	Emmanuel Bernard(RedHat)	Early Draft Method validation Integration with CDI	http://beanvalidation.org/
WebSocket 1.0/356	Danny Coward (Oracle)	Early Draft Annotation driven model and API proposed	http://websocket-spec.java.net
JSON 1.0/353	Jitendra Kotamraju (Oracle)	Early Draft API for JSON representation Streaming API to produce/consume JSON	http://json-processing-spec.java.net http://jsonp.java.net
Batch 1.0/352	Chris Vignola (IBM)	Early Draft	http://java.net/projects/jbatch

API /JSR Nummer	Spec Lead	Status/Features	Projekt-Seiten
		RI available Batch domain language	
State 1.0/350	Mitch Upton (Oracle)	In progress	http://java-state-managemen.java.net
Concurrency 1.0/236	Anthony Lai (Oracle)	Early Draft Candidate	http://concurrency-ee-spec.java.net
JCache 1.0/107	Gregory Luck Brian Oliver (Oracle) Cameron Purdy (Oracle)	Early Draft RI available	https://github.com/jsr107/jsr107spec https://github.com/jsr107/RI

Abb. 3: Modifizierte und neu aufzunehmende API in Java EE 7

Ausblick

Nach gut einem Jahr intensiver Arbeit nimmt die Java EE 7 Plattform Gestalt an. Der Umfang der Verbesserungen vieler Teilspezifikationen aus Java EE 6 wurde weitgehend festgelegt. Die entsprechenden JSRs wurden gestartet, ihre Early-Draft-Versionen liegen bereits zur öffentlichen Diskussion vor. Es wird noch zu entscheiden sein, welche der neuen API, die auch gut vorangekommen sind, in die Plattform aufgenommen werden. Da die Fertigstellung der Java EE 7 Spezifikation auf die Mitte nächsten Jahres terminiert wurde, ist es wahrscheinlich, dass einige ihrer angestrebten Features aus Zeitgründen auf die nächste Java EE 8 Version verschoben werden müssen. Wie das Thema PaaS, wobei hier die noch geringe Erfahrung auf dem Markt eine wichtigere Rolle spielt. Für Java EE 8 bleiben die Themen PaaS, Modularisierung, OSGi Interoperabilität und die Ausarbeitung eines neuen Programmiermodells, das die Anwendungsentwicklung auf der Basis der entstehenden HTML5-Standards besser unterstützt.

Es ist noch möglich, sich an der Entwicklung der Java EE 7 Plattform oder der einzelnen API 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 sehr 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