Java Enterprise Edition 6

Markus Eisele
msg systems ag
Ismaning

Schlüsselworte:

Java, Java EE 6

Einleitung

Es hat mal wieder mal sehr lange gedauert. Nach den zwei Jahren Spezifikationszeit für die Vorgängerversion Java EE5 hat es diesmal sogar zwei ein halb Jahre gedauert, die neue Version der Java Enterprise Edition (Java EE) zur Freigabe zu bringen. Nach einem fulminanten Fehlstart als Java Specification Request (JSR) 313 im April 2007 formierte sich die Expert Group bereits im Juli unter dem JSR-316 neu und startete mit der Arbeit. Der ursprüngliche Plan, bereits 2008 mit einer finalen Version fertig zu sein kann zu dokumentarischen Zwecken auf der Webseite des Java Community Process (JCP) noch nachgelesen werden.

Versionspflege?

Vieles bleibt auf den ersten Blick mehrheitlich beim Alten. Die mit der Vorgängerversion Java EE 5 betretenen Pfade werden konsequent weiter ausgebaut. Das Entwicklungsmodell wird Spezifikationsübergreifend mehr und mehr Metadaten (Annotation) getrieben und durch ein geeignetes Standard-Verhalten der entsprechenden Container (Web, EJB, etc.) ergänzt. Somit werden die gehassten XML Deployment Beschreibungen vielfach vollkommen unnötig. Am Beispiel der Servlets läßt sich dies anschaulich machen. Wurden mit Java EE 5 und früheren Versionen entsprechende Einträge in dem Webapplikations-Deployment Deskriptor web.xml notwendig, lässt sich ein Java EE 6 konformes Servlet durch die Verwendung von Annotationen konfigurieren.

```
@WebServlet(„/doag")
public class DoagServlet extends HttpServlet {…}
```

Sind bestimmte Anforderungen an die Ausführungsumgebung des Servlets nicht explizit im Code bzw. an der Annotation formuliert, übernimmt der Container mit seinen Standard Einstellungen und führt Aufrufe an dieses Servlet beispielsweise synchron aus. Ist dies nicht gewünscht, kann alternativ immer noch über den Deployment Deskriptor eingegriffen werden und das Laufzeitverhalten verändert werden. Dieses Vorgehen ist nunmehr der Standardfall für viele Spezifikationen.
Abseits dieser Vereinfachungen sind auch einige größere Versions- und damit auch Funktionssprünge in der Java EE 6 enthalten.
Die Enterprise Java Beans (EJB) Spezifikation liegt nun in der Version 3.1 vor. Technische Höhepunkte sind hier vor allem der „no interface" Local view, welcher die Umsetzung von EJBs als einfache, Plain Old Java Objects (POJOs) ermöglicht. Bisher benötigte eine lokale Session Bean immer ein lokales Business-Interface. Mit EJB 3.1 gelten Session Beans ohne Interface automatisch als Local Session Beans, deren komplette Methoden von Client ausgerufen werden können.
Singleton Beans werden jetzt ebenfalls unterstützt. Dabei handelt es sich um Enterprise Java beans, von denen nur genau eine Instanz existiert. Bisher waren solche Konstrukte in der Java EE nur mit Herstellerspezifischen Funktionen abzubilden. Ein Singleton Bean wird mit der Annotation `@Singleton` versehen. Der EJB Container übernimmt auch hier die Erzeugung und Verwaltung.

Angelehnt an die Concurrency API aus Java SE 5 werden nun auch wie asynchrone Methodenaufrufe angeboten. Bisher ein klassischer Fall für die Verwendung des vergleichsweise schwergewichtigen "Java Messaging Service" (JMS). Auch hier wird durch das hinzufügen einer Annotation (`@Asynchronous`) eine Bean Methode zur asynchronen Ausführung vorgesehen.
Gänzlich neu ist die EJB „Lite“. Ein Subset, welches EJB Funktionalität für das Web Profil zur Verfügung stellt. Damit EJB Komponenten auch außerhalb von Java EE Containern ausgeführt werden können ist eine Embeddable EJB API ebenfalls Teil der Spezifikation geworden. Durch letztere lassen sich vor allem Testszenarien deutlich vereinfachen, da für die Ausführung kein vollständiger Container mehr benötigt wird. In Summe wird das Deployment von EJBs vereinfacht. Seit dieser Version können EJBs auch direkt in Web-Archiven (WAR-Dateien) verpackt und deployed werden.
Abgekoppelt aber verbunden mit EJBs ist die Java Persistance API. Sie hat mit der Versionsnummer 2.0 eine Überarbeitung erfahren. Vordringlich wurden hier die Wünsche der Java Community bedient und die Vorgängerversion entsprechend abgerundet. Offensichtliche Lücken (Collection Support, Cache Control, Criteria API, Advanced Locking, etc.) sind geschlossen und bestehende Dinge erweitert (bspw. JP QL). Auch hier wurde der Schulterschluss zu anderen JSRs gelebt. So bietet die JPA jetzt auch eine Validation API.
Die Java Server Faces liegen ebenfalls in Version 2.0 vor. Neben einer breiten Unterstützung für Asynchrones Javascript und XML (AJAX) wurden auch die sogenannten Facelets in die Spezifikation aufgenommen. Ebenfalls hinzugenommen wurde eine umfangreiche Ressourcen Verwaltung. Das hinzufügen von View Parametern, sowie Partial State Saving ergänzen die JSF 2.0 ebenso sinnvoll wie das Konzept der Composite Components. Abseits dieser Höhepunkte hat die JSF Spezifikation wohl die beeindruckendste Änderungs- und Erweiterungsliste vorzuweisen.

Das wirklich Neue

Neben der Versionspflege der etablierten Teile, sind vor allem ein paar neue Technologien dazu gekommen. Besonders erwähnenswert ist hierbei die Tatsache, dass noch kurz vor der finalen Veröffentlichung der Java EE 6 die Aufnahme von zwei neuen Spezifikationen bekannt gegeben wurde, um die es sehr lange Diskussionen gab. Sowohl der JSR-330 (Dependency Injection for Java) als auch der JSR-299 (Web Beans) wurden erst im August 2009 aufgenommen und sind seitdem fester Bestandteil der Java EE 6. Die Umstellung aller beteiligten Spezifikationen erfolgte in rekordverdächtigen zwei Monaten. Der auch als Web Beans bekannt gewordene JSR-299 bietet ein vollständig neues Set an Funktionen für die komplette Java EE Plattform. Grundkonzept ist dabei eine Menge von definierten Lebenszyklus Kontexten (RequestScoped,ApplicationScoped, SessionScoped, ConversationScoped und Dependent), welche die Zusammenarbeit von Java EE Komponenten regeln. Diese ermöglicht das einheitliche Binden von Objekten (JSF Managed Beans, EJBs, etc.) an definierte Kontexte. Diese können auch via Dependency Injection (DI) lose gekoppelt werden. Die im JSR-299 definierten DI Funktionalitäten basieren dabei auf den Annotationen, welche durch den JSR-330 definiert wurden. Die Funktionalitäten beider JSRs sind aktuell vergleichsweise gut aufeinander abgestimmt. Die Abstimmungsprotokolle im Java EE 6 JSR zeugen aber von einer bewegten Vergangenheit beider Spezifikationen. Nicht zu Letzt sind auch die kritischen Stimmen der Java Community noch nicht verstummt, welche die Praxistauglichkeit beider anzweifeln.
Mit der Java API für RESTful Web Services (JAX-RS, JSR-311) kommt endlich auch die REST Unterstützung in die Plattform. Weitere Versionsupdates unter anderem der Java API for XML-Based Web Services (JAX-WS) und auch der Java Architecture for XML Binding (JAXB) runden den Web Service Technologie Stack ab.
In Summe haben 15 Spezifikationen ein Versionsupdate erfahren. Inklusive dem Umbrella JSR sind davon fünf Major Updates verfügbar. Bei dem Rest handelt es sich um Minor Versionen. Genau zehn neue Technologien sind seit Java EE 5 in die Plattform aufgenommen worden.

Ein paar lange diskutierte Kandidaten haben es nicht in die Spezifikation geschafft. Dazu gehört das Thema Portlets (JSR-168, JSR-286, JSR-301) genauso wie die Java Business Integration (JBI, JSR-208). Ebenso vermissen wird man die Content Repository for Java technology API (JSR-170).

Profiles und Pruning – die Jagd nach den Pfunden

Blickt man auf die Entwicklung der Spezifikation, dann sind alle Bemühungen, die Komplexität zu Reduzieren bisher nur im Umfeld der Entwicklung einzelnen Technologien zu bemerken. Das Gesamtwerk erreicht in der Version 6 die wahnsinnige Zahl von mehr als 6000 A4-Seiten Spezifikation. Dabei ist bei dieser Zahl noch einiges an Release-Notes unberücksichtigt geblieben. Im Vergleich mit der Java EE 5 sind nochmal knapp 900 Seiten an Umfang dazu gekommen. Im Vergleich mit den Vorgängerversionen sieht die Entwicklung noch dramatischer aus (vgl. Grafik). Noch nicht enthalten sind hier die durchaus auch notwendigen Handbücher für den Applikationsserver der Wahl. Auch die gewünschte Entwicklungsumgebung bringt noch die eine oder andere Seite Dokumentation mit. Auch auf den zweiten Blick kein leichter Einstieg in die Plattform für Enterprise Java. Kritisch hinterfragt wird klar, dass die Java EE 6 noch viele Technologien enthält, welche den sogenannten Kompatibilitäts-Modus zu früheren Spezifikationen ermöglichen sollen. Trotz kaum mehr praktischer Relevanz finden sich dennoch sowohl Container Managed Persistence (CMP) als auch JAX-RPC oder auch die JAXR API noch immer im Standard. Alle drei sind durchaus Kandidaten für die Entfernung aus dem Standard. Nachdem sich die Java Enterprise Edition durchaus als Industrie Standard bezeichnen kann verbirgt sich in diesem Vorhaben durchaus Sprengstoff. Die Java EE Expert Group möchte sich deshalb zukünftig eines „pruning“ genannten Verfahrens bedienen um alten Ballast los zu werden. In Anlehnung an das bei der Java SE 6 eingeführte Vorgehen sollen dabei zukünftig als überflüssig identifizierte Umfänge von der Expert Group vorgeschlagen und dokumentiert werden. Die darauf folgende Expert Group kann diese dann tatsächlich aus dem Standard nehmen oder der darauf folgenden zur Entscheidung überlassen.
Allein die Tatsache, dass bisher nur drei Kandidaten für das Pruning vorhanden sind, lässt erahnen, dass das eigentliche Ziel der Vereinfachung auf diesem Weg nicht erreicht werden kann. Einen deutlich größeren Schritt tut die Java EE 6 dabei mit der Einführung von sogenannten Profilen. Dies bündeln Technologie Pakete entlang der Nutzerprofile. Für die durchschnittliche Webanwendung ist damit nicht mehr die komplette Java EE 6 Spezifikation notwendig, will man beispielsweise trotzdem EJBs einsetzen, sondern es kann direkt auf ein optimiertes Profil zurückgegriffen werden. Zusammen mit der Java EE 6 definiert die Expert Group ein Profil. Das sogenannte Web Profil vereint genau die typische Technologie Kombination, welche eine leichtgewichtige Entwicklung verspricht. Weitere Profile sollen von den Herstellern kommen. Je nach Bedarf sind hier nahezu beliebige Kombinationen denkbar. Einzig die Verbindung zur übergreifenden Spezifikation soll eindeutig sein. Es wird also nicht nur ein Web Profil geben, sondern zukünftig ein entsprechendes Profil pro Spezifikationsversion. Das Webprofil reduziert die Komplexität, gemessen am Gesamtumfang der notwendigen Dokumentation um mehr als die Hälfte. Mit gut 3000 Seiten bleibt es noch immer ein anständiges Paket. Zum Vergleich: Das „leichtgewichtige“ Spring kommt hier mit grade einmal 1000 Seiten aus.
Die Zukunft von Enterprise Java
Mit der Java EE 6 geht es in die richtige Richtung. So oder ähnlich formulieren es Kritiker und Befürworter des Java Enterprise Standards. Zumindest auf die Technologie bezogen. Leichtgewichtiger sollte sie werden. Dieses Ziel wurde mithilfe der Profile durchaus erreicht. Die großen Applikationsserver Hersteller werden ausgehend von dieser Steilvorlage beginnen ihre eigenen Profile auf den Markt zu bringen. In wie weit sich diese von den nächsten Umbrella Spezifikationen einfangen lassen, bleibt abzuwarten. Ein grundlegender Trend lässt sich bei allen Herstellern erkennen. Es geht hin zum DM (Dynamic Modules) Server. Das mittlerweile geläufige Kürzel für einen auf OSGI Standards basierten Applikationsserver trägt bisher zwar offiziell nur der Spring DM

Server. Erste Präsentationen in denen beispielsweise von einem Weblogic DM Server die Rede ist, sind aber auch schon im Netz verfügbar. Erst ein zumindest zu OSGI vergleichbarer Mechanismus ermöglicht die nahezu freie Kombination von verschiedenen Containern zu einem Gesamten. Auch wenn immer Abhängigkeiten zwischen einzelnen Technologie-Paketen bestehen bleiben, so ebnet diese bedarfsgerechte Kombinationsmöglichkeit doch den Weg für die Zukunft. Wem OSGI als Lösung für diese Probleme nicht behagt, mag sich über den JSR 294 freuen. Was als Improved Modularity Support in the Java Programming Language bereits im April 2006 gestartet wurde, soll nun mit der Java SE 7 verfügbar sein.
Egal, welcher Weg gegangen wird, Tatsache bleibt, dass nicht nur die Standards kleiner werden müssen. Vor allem ihre Ablaufumgebungen (Java EE Container) müssen zugunsten von Geschwindigkeit und Handhabbarkeit optimiert werden. Laut Roberto Chinnici (Spezification Lead Java EE 6) soll es zukünftig möglich sein, die verfügbaren EE Container nahezu beliebig auf Ebene von einzelnen Objekten zu verwenden. Dazu müssen noch viele weitere Grenzen fallen.

Kontaktadresse:

Markus Eisele
msg systems ag
Robert-Bürkle-Str.1
D-85737 Ismaning

Telefon:	+49 (0) 89-96 101 0
Fax:	+49 (0) 89-96 101 2315
E-Mail	markus.eisele@msg-systems.com
Internet:	www.msg-systens.com
Internet:	blog.eisele.net