



Das iJUG Magazin

Java aktuell

Herbst 2011

Java aktuell
Magazin der Java-Community

Java überall

- Neu: Java SE7
- Interview mit Patrick Curran, Vorsitzender des JCP
- Für Android entwickeln
- Testen mit Arquillian
- Suchen mit Apache Solr



iJUG
Verbund

Erfahrungen, Ideen und Lösungen für Java-Entwickler

www.ijug.eu D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

Sonderdruck

04/2011



- 3 Editorial
- 5 Die lange Reise von Java 7
Markus Eisele, msg systems ag
- 8 „Unbedingt die Specs lesen und Feedback geben ...“
Interview mit Patrick Curran, Vorsitzender des JCP
- 11 Das Java-Tagebuch
Andreas Badelt, DOAG Deutsche ORACLE-Anwendergruppe e.V.
- 15 Java überall
Oliver Szymanski, Source-Knights.com, stellv. Vorstandsvorsitzender des iJUG
- 17 „Java muss sich neuen Einsatz-Szenarien wie Cloud und Mobile Computing stellen ...“
Interview mit Dr. Mark Little, Red Hat
- 19 Arquillian
Frederik Mortensen
- 22 Suchen mit Apache Solr
Peter Karich, Pannous GmbH
- 26 „Ich denke, die Java-Community ist wie eine große Familie ...“
Interview mit Michael Hüttermann, Java User Group Köln
- 28 Android – Java macht mobil
Andreas Flügge, object systems GmbH
- 31 Hibernate im Projekteinsatz
Dirk Mahler, buschmais GbR
- 34 Semantisch-orientierte Programmierung mit Java
Oliver Böhm
- 37 Slice – Unterstützung für verteilte, partitionierte und heterogene Datenbanken mit OpenJPA
Bernd Müller, Ostfalia Hochschule für angewandte Wissenschaften, sowie Harald Wehr, MAN Truck & Bus AG
- 40 NetBeans Platform 7
gelesen von Jürgen Thierack
- 41 XPages – Ein neues Framework zur Entwicklung von Web-Anwendungen
Dr. Rolf Kremer, PAVONE AG
- 45 „Bereits jetzt zählen wir zu den führenden Java-Magazinen im deutschsprachigen Raum ...“
Interview mit Fried Saacke, Vorstandsvorsitzender des iJUG
- 46 Leichtgewichtige Authentifizierung mit OpenID
Sebastian Glandien, Acando GmbH
- 51 Java-Problem-Determination mit der IBM Support Assistant Workbench
Marc Bauer, IBM Deutschland GmbH
- 54 Java EE 7 – eine Reise in die Wolken
Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG
- 57 Varianten-Entwicklung in 3D mit Object Teams
Dr. Stephan Herrmann, GK Software AG
- 60 Unbekannte Kostbarkeiten des SDK Heute: Der Service-Loader
Bernd Müller, Ostfalia
- 62 Rich Client Frontends für umfangreiche Unternehmensanwendungen
Björn Müller, CaptainCasa
- 61 Inserenten
- 53 Impressum



Kleiner Exkurs darüber, wo wir Java direkt oder indirekt überall antreffen, Seite 15

Dies ist ein Sonderdruck aus der Java aktuell. Er enthält einen ausgewählten Artikel aus der Ausgabe 04/2011. Das Veröffentlichen des PDFs bzw. die Verteilung eines Ausdrucks davon ist lizenzfrei erlaubt. Weitere Informationen unter www.ijug.eu



Arquillian

Frederik Mortensen

Der Artikel befasst sich mit Unit-Tests im Enterprise-Application-Bereich unter Verwendung von Arquillian. Dazu werden Quellcode-Beispiele aufgeführt, die Maven als Build-Management-Tool sowie JBoss und Glassfish als mögliche Beispiel-Application-Server verwenden. Grundlegende Vorkenntnisse im Bereich der serverseitigen Java EE-Entwicklung (Enterprise Java Beans EJB3.1) sowie eines Unit-Test-Frameworks sind daher empfehlenswert.

Um den Erfolg und die Notwendigkeit von Arquillian verstehen zu können, muss man sich mit der Geschichte von Java EE im Allgemeinen befassen. Dessen Final-Release wurde am 10. Dezember 2009 veröffentlicht. Betrachtet man die vielen Veränderungen und Verbesserungen, die im Laufe der Versionen Einzug in die Plattform hielten, wird klar, dass vieles aus dem Wunsch heraus entstanden ist, Java noch weiter zu vereinfachen und zu verbessern. Die Praxis hat gezeigt, dass der übermäßige Einsatz von Interfaces (siehe EJB 2.1), Deployment-Deskriptoren und abstrakten Bean-Klassen bald auch belastend und unübersichtlich werden kann und einfach unnötig ist. Nicht zuletzt beeinflusst durch Lösungsansätze wie das Spring-Framework entstand mit Java EE 6 eine leistungsfähige, stabile, aber vor allem auch leicht handhabbare Plattform, die sich sehen lassen kann. Frei nach dem Motto „Convention over Configuration“ promoten Java-Champions wie Adam Bien auf Veranstaltungen, wie einfach es ist, Transaktions-Handling, Persistenz, Security oder RESTful-Web-Services einzubinden. Dazu sind oft nur wenige Annotationen erforderlich. Java EE 6 wurde schnell von der Community akzeptiert. Mehr und mehr Anwendungen werden migriert, der TIBOE-Index meint es gut mit Java.

Die Frage nach dem Warum

Dass Java EE 6 sich schnell zu einer Erfolgsgeschichte entwickelt hat, wissen wir nun.

Doch ist es perfekt? Es stellt sich die Frage: „Welche Lücke füllt Arquillian?“

Keine professionelle Software wird heute noch ohne eine absichernde Teststrategie auskommen. Idealerweise macht man sich zu Anfang Gedanken über die Schnittstelle und schreibt dann zuerst einen Unit-Test. Das Implementieren der wirklichen Geschäftslogik folgt erst danach. Ist die Entwicklung abgeschlossen, lässt man die Unit-Tests laufen und sieht recht schnell, ob das gewünschte Ergebnis auch erreicht wurde. Grundgedanke dieses Testansatzes war immer: Funktionieren die einzelnen Bestandteile, so funktioniert auch die Software als Ganzes. Eine möglichst hohe Abdeckung von Quellcode mit Tests war und ist wünschenswert. Doch hielt man sich bisher an diesen guten Brauch, merkt man in Enterprise-Anwendungen schnell, dass hier etwas nicht stimmt. Es schleicht sich das Gefühl ein, dass die bisherigen Testmethoden unzureichend sind. Es entsteht ein Code, der nicht ohne Weiteres testbar ist. Themen wie „Dependency Injection“ und „Session Beans“ machen schnell deutlich, dass mit einfachen Mockup-Objekten hier nicht gearbeitet werden kann. Wo in klassischen Client-Anwendungen einfache Unit-Tests ausreichen, muss man sich in einer verteilten Architektur mit der Testbarkeit sowie dem Zusammenspiel verschiedener Komponenten Gedanken machen. Es bestehen Abhängigkeiten untereinander. Genau hier kommt Arquillian ins Spiel.

Die Einbürgerung der Integration

„Der Begriff „Integrationstest“ bezeichnet in der Software-Entwicklung eine aufeinander abgestimmte Reihe von Einzeltests, die dazu dienen, verschiedene voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander zu testen. Die erstmals im gemeinsamen Kontext zu testenden Komponenten haben jeweils einen Modultest erfolgreich bestanden und sind für sich isoliert fehlerfrei funktionsfähig“ (aus Wikipedia).

Die Referenz-Dokumentation beschreibt Arquillian mit der Überschrift „An integration testing framework for Containers“. Es geht also um sogenannte „Integrationstests“. Der verwendete Begriff macht deutlich, dass die angewandte Testmethode sich von herkömmlichen Unit-Tests unterscheidet. Im Gegensatz zum Testansatz einzelner Klassen werden hier ganze Abläufe simuliert durchlaufen. Es findet eine Kombination von Modulen statt und die erwähnten Abhängigkeiten entstehen. Die bei Unit-Tests getroffene Annahme, dass das Gesamtbild passt, wenn die Einzeltests funktionieren, trifft hier also nicht zu. Die Kernkompetenz von Integrationstests besteht darin, Erfolg oder Misserfolg im Zusammenspiel der Module zu prüfen. Nimmt man beispielsweise in einer Web-

Hinweis: Die Listings zu diesem Artikel finden Sie unter <http://src.ijug.eu/ja/1104/arquillian.zip>



anwendung den Vorgang des Registrierens eines neuen Benutzers als Vorlage, so ist dieser Vorgang erst dann erfolgreich abgeschlossen, wenn folgende Bedingungen erfüllt sind:

- Die Benutzereingabe validiert
- Der Benutzer persistiert
- Eine E-Mail an den Benutzer versandt
- Die entsprechenden Rechte gesetzt

Dabei können für das Versenden der Nachricht und das Persistieren des Benutzers schon verschiedene Services beansprucht worden sein. Ist einer dieser Services fehlerhaft oder nicht verfügbar, scheitert die Registrierung im Ganzen. Ein Integrationstest würde hier also bereits mehrere Abhängigkeiten einfordern.

Hello Arquillian

Die graue Theorie klingt vielleicht etwas herausfordernd. Nun der Schritt in die Praxis: Ein Beispielprojekt veranschaulicht, wie man mit Arquillian als Framework sein Ziel recht einfach erreicht. Dabei spielt die fachliche Logik eine eher untergeordnete Rolle. Die angeführten Beispiele sind mehr als technische Vorlage gedacht und zu verstehen. Das Beispiel demonstriert einen ersten Einsatz unter Verwendung von JBoss. Der Applications-Server liegt derzeit als Release-Candidate in Version 7.0.0.CR1 vor, wir nutzen jedoch vorerst noch die 6.0.0.Final-Version. Downloadbar ist er unter <http://www.jboss.org/jbossas/downloads>. Als IDE wurde NetBeans 7.0 verwendet, die Beispiele sind aber natürlich auch in jeder anderen IDE lauffähig (Maven vorausgesetzt). Sind IDE und Application-Server also installiert, muss ein EJB3 Projekt angelegt werden. Listing 1 zeigt eine Beispiel-POM.

Viele Geheimnisse birgt diese Konfiguration erst einmal nicht. Unter den Eigenschaften (properties-Block) wird die aktuelle Arquillian-Version 1.0.0-Alpha5 als Variable hinterlegt. Demnächst erscheint der Release-Candidate 1. Ist dies der Fall, ist nur in dieser Eigenschaft die Versionsnummer zu ändern, um die neue Abhängigkeit herunterladen zu lassen. Als Nächstes sind die benötigten Repositories zum Download der Bibliotheken angegeben. Darauf folgend sind die Dependencies für das

erste Beispiel gelistet. Im vorliegenden Fall wird ein Test mittels des JUnit-Frameworks auf einem vorinstallierten JBossAS per Remote-Aufruf ausgeführt. Daraus ergeben sich die gelisteten, zu verwendenden Abhängigkeiten. Beim ersten Clean und Build des Projekts kann es etwas dauern, bis alle Abhängigkeiten aus dem entsprechenden Repository heruntergeladen worden sind. Um erfolgreich den Remote-Aufruf absetzen zu können, wird eine weitere Datei benötigt, die wie folgt aussieht und im „src/test/resources“-Ordner abgelegt sein muss:

```
java.naming.factory.initial=org.jnp.interfaces.  
NamingContextFactory  
java.naming.factory.url.pkgs=org.jboss.  
naming:org.jnp.interfaces  
java.naming.provider.url=jnp://localhost:1099
```

„Localhost“ ist unter der Annahme eingetragen, dass der zu nutzende Applications-Server auf dem Entwicklungs-PC lokal installiert ist. Zuletzt sind noch die zu testende Java-Klasse und deren Testklasse erforderlich, hinterlegbar jeweils unter „src/main/java“ beziehungsweise „src/test/java“. Um den Fokus erstmal auf Arquillian zu belassen, fällt das Beispiel sehr simpel aus, es werden nur zwei Zahlen addiert und das Ergebnis zurückgegeben (siehe Listings 2 und 3).

Das erste Mal findet hier Arquillian Erwähnung, wenn man es der JUnit-Annotation „@RunWith“ als „TestRunner“-Parameter übergibt. Eine Instanz der zu testenden Klasse „IntCounter“ wird dann per „@Inject“ unter Zuhilfenahme von CDI instanziiert. Richtig spannend wird es mit der Annotation „@Deployment“. Hier kommt ein erster Vorgeschmack auf die Arbeitsweise von Arquillian auf. Mittels des „ShrinkWrap“ genannten Deployment-Mechanismus (ebenfalls ein Unterprojekt von jboss.org) ist es möglich, programmatisch und voll automatisiert ein .JAR-, .WAR- oder .EAR-Archiv zu erzeugen, um dieses dann in einem eingebetteten Server zu nutzen oder per Serialisierung remote auf den Zielsever zu übertragen. Wichtig hierbei ist, anzugeben, welche Klassen letztendlich im Archiv aufgenommen werden sollen und wie der Archivname lauten soll. In fortgeschrittener Nutzung wird es auch möglich

sein, ganze Archive statt einzelner Klassen einzubinden. Sinnvoll ist dies etwa in Maven-Projekten, wo man eventuell sein „domain model“ (also alle Entity-Klassen) als Ganzes übergeben möchte. Die ebenfalls angelegte, leere Datei „beans.xml“ wird für die Verwendung von CDI benötigt und sollte so übernommen werden. Ein erster Testlauf des anzustoßenden Unit-Tests sollte folgenden Consolen-Output liefern:

```
Tests run: 1, Failures: 0, Errors: 0, Time  
elapsed: 7,438 sec
```

Die embedded-Variante mit EJB

Die oben erwähnten 7,438 Sekunden für eine einfache Addition an sich sind vielleicht nicht unbedingt überzeugend. Man sollte jedoch bedenken, dass im Hintergrund unter anderem das Erzeugen und Löschen des Testarchives per „ShrinkWrap“ stattfindet sowie der Remote-Aufruf auf dem Application-Server. Wenn wir jetzt auf einen Embedded-Test umstellen, wird sich die Zeit für einen einzelnen Test sogar noch erhöhen, weil das Hochfahren und Beenden des Servers dann noch zur Laufzeit hinzukommt. Allerdings handelt es sich ja, wie bereits erwähnt, um Integrationstests, die im Gegensatz zu Unit-Tests nicht bei jedem Build-Vorgang angestartet werden müssen. Ein Zeitgewinn entsteht dann, wenn in einer Testklasse mehrere Einzeltests hintereinander erfolgen. Dabei ist das Hochfahren des Embedded-Servers natürlich nur einmal am Anfang des ersten Tests nötig und der Server wird nach Durchlaufen des letzten Tests schließlich wieder beendet.

Ein zweites ähnliches Beispiel zeigt die vollen Vorzüge Arquillians. Denn wer möchte schon dauerhaft einen Testserver laufen lassen, wenn Arquillian das Starten, Deployen und Stoppen des Servers komplett für einen übernehmen kann. Versprochen wurde auch bereits, dass EJBs (allein und als Abhängigkeiten untereinander) ebenfalls testbar sind.

Ziel dieses Beispiels ist das Einbinden von EJB3 sowie der Einsatz von Glassfish im Embedded-Modus. Wichtig hierbei ist, dass im Gegensatz zum vorherigen Beispiel keine „jndi.properties“-Datei angelegt werden darf. Listing 4 zeigt die POM.



Wie man sehen kann, ist ein neues Repository hinzugekommen. Grund ist die neue, benötigte Abhängigkeit zu Glassfish. Neben der POM-Datei werden nun noch die EJB- und die Test-Klasse erstellt (siehe Listings 5 und 6). Der angelegte TimeService gibt die aktuelle Jahreszahl als String zurück. Die Annotation „@EJB“ wurde – wie bei Session-Beans üblich – dazu verwendet, um den Service mittels Dependency Injection zu laden. Der schlanke Quellcode lässt erahnen, wie viel Arbeit einem Arquillian erspart und übernimmt. Ist man durch diese ersten Beispiele neugierig geworden, findet man unter <https://github.com/arquillian/arquillian-examples> weiterführende, eingetragene Quellcode-Beispiele.

Die Einfachheit der Tests bedeutet keinesfalls, dass man als Entwickler hiermit schon an die Grenzen des Machbaren stößt. Mittels hinterlegbarer Konfigurationsdateien wie „arquillian.xml“ oder „domain.xml“ (unter GlassFish) kann man noch einiges an Feinjustierung vornehmen.

Fazit

Arquillian hat eine vernünftige Basis geschaffen. Der Einsatz ist durchaus auch in größeren, produktiven Umgebungen möglich. Trotzdem sollte man natürlich beachten, dass sich das Produkt noch in der Alpha-Phase befindet. Erfreulich ist es daher, zu hören, dass ein erster Release-Candidate schon in Arbeit ist. Die aufgezeigten Quellcodes können natürlich nur einen Vorgeschmack auf das liefern, was Arquillian an Funktionalität bietet. Es lohnt sich, tiefer in die Materie einzudringen, indem man sich die Referenz-Dokumentation näher ansieht. Neben diversen Konfigurationsmöglichkeiten für die möglichen Applikations-Server findet man dort Informationen über Remote-Debugging, erweiterte Anwendungsfälle, Hintergründe und Tipps zu ShrinkWrap, dem Erstellen von Archiven, Hinterlegen von DataSources und mehr. Das Ausprobieren lohnt sich. Die volle Unterstützung von JPA2 beispielsweise in Unit-Tests zu sehen, lässt einen den anfänglichen Einarbeitungsaufwand schnell vergessen. Es wäre schön,

wenn dieser Artikel einen ersten Anreiz zum Umdenken in so manchem Projekt führen könnte, und es bleibt zu hoffen, dass Arquillian weiterhin Anwendung findet.

Weiterführende Quellen

<http://docs.jboss.org/arquillian/reference/1.0.0.Alpha5/en-US/html/>
<http://de.wikipedia.org/wiki/Integrationstest>

*Frederik Mortensen
 mortensen.frederik@googlemail.com*

Frederik Mortensen ist gelernter Anwendungsentwickler, Sun-zertifiziert (SCJP) und seit zehn Jahren im Bereich der Java-Entwicklung in verschiedenen Tätigkeitsfeldern aktiv. Derzeit ist er mitverantwortlich für die Weiterentwicklung des Java-Backends einer Domain-Verwaltungssoftware der Firma InternetX in Regensburg. In seiner Freizeit beschäftigt er sich neben der Entwicklung von Web-Anwendungen auf JSF2- und EJB3.1-Basis auch mit der Entwicklung eines Open-Source-Projekts im Java-3D-Bereich.



(G) G E B I T Solutions

**Potential
 zum
 wachsen!**

www.gebit.de/jobs

SW-Entwickler/Berater (m/w)

Java Profis suchen Gleichgesinnte!

Auf Sie warten

anspruchsvolle Projekte und interessante Aufgaben, hochqualifizierte und nette Kollegen, State-of-the-Art Technologien und Werkzeuge, sehr gute Lern- und Entwicklungsmöglichkeiten.

Was Sie mitbringen!

Sehr gute OO-/Java-Kenntnisse und Erfahrung in der Entwicklung/Konzeption server- bzw. clientseitiger Anwendungen, Informatikstudium oder entsprechende Praxiserfahrung, kundenorientiertes und selbstverantwortliches Arbeiten im Team, Begeisterung für Neues, Motivation zur permanenten Weiterbildung

Lust auf neue Herausforderungen?

Neugierig geworden und Lust auf Zusammenarbeit mit Kollegen, die ihr Handwerkzeug verstehen? Wir freuen uns auf Ihre Online-Bewerbung!

Mehr Infos: www.gebit.de/jobs



Bestellen eines kostenlosen Exemplares der Zeitschrift Java aktuell

Anschrift:

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

ggf. Rechnungsanschrift

E-Mail

Telefonnummer

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

Jetzt Abonnement sichern:

- Abonnement Newsletter: Java aktuell – der iJUG-Newsletter, kostenfrei
- Java aktuell – das iJUG-Magazin Abo: vier Ausgaben zu 18 Euro im Jahr

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG News und zwei Ausgaben im Jahr Business News. Weitere Informationen unter www.doag.org/shop/

Senden Sie das ausgefüllte Formular an:

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

oder faxen Sie es an:

0700 11 36 24 39

oder bestellen Sie online:

go.ijug.eu/go/abo

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechnung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Widerrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.

Impressum

Herausgeber:
Interessenverbund der Java User
Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 0700 11 36 24 38
www.ijug.eu

Verlag:
DOAG Dienstleistungen GmbH
Fried Saacke, Geschäftsführer
info@doag-dienstleistungen.de

Chefredakteur (VisdP):
Wolfgang Taschner,
redaktion@ijug.eu

Chefin von Dienst (CvD):
Carmen Al-Youssef,
office@ijug.eu

Titel, Gestaltung und Satz:
Claudia Wagner,
DOAG Dienstleistungen GmbH

Anzeigen:
CrossMarkeTeam, Ralf Rutkat,
Doris Budwill
redaktion@ijug.eu

Mediadaten und Preise:
[http://www.ijug.eu/images/
vorlagen/2011-ijug-mediadaten_
java_aktuell.pdf](http://www.ijug.eu/images/vorlagen/2011-ijug-mediadaten_java_aktuell.pdf)

Druck:
adame Advertising and Media
GmbH Berlin
www.adame.de

Java aktuell – das Abo
4 Ausgaben für 18 Euro