03-2014 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

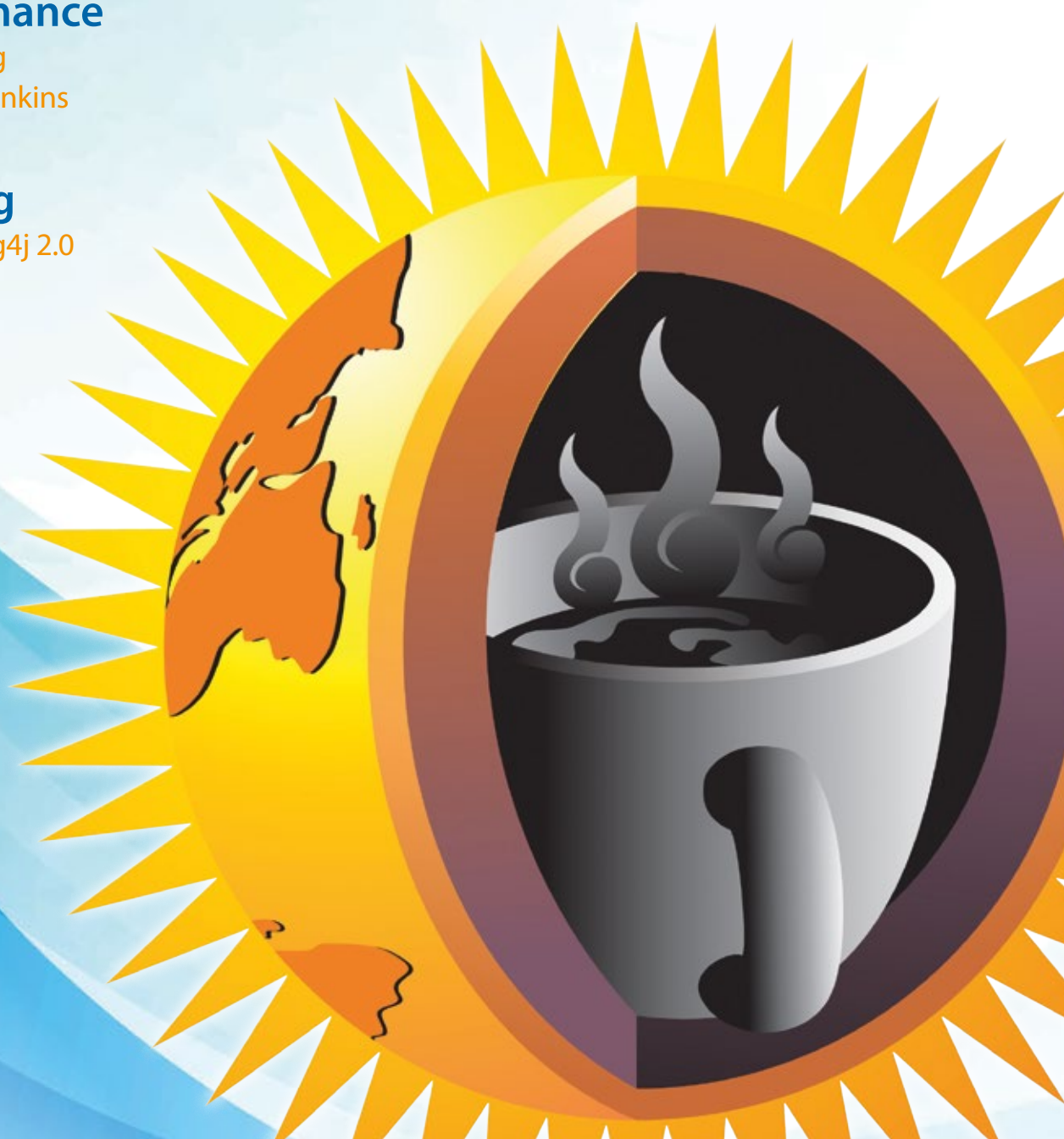
Java im Mittelpunkt

Aktuell
NoSQL für Java

Performance
Java-Tuning

Pimp my Jenkins

Logging
Apache Log4j 2.0



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

iJUG Verbund

JAVA FORUM stuttgart 2014

Donnerstag, 17. Juli 2014,
17. Java Forum Stuttgart
JFS

Eventpartner:

exXcellent solutions · THALES · TNG TECHNOLOGY CONSULTING

Die führende non-profit-Konferenz und Ausstellung rund um das Thema Java mit einem breiten Spektrum aktueller Themen der Software-Entwicklung

Vorträge

49 Vorträge in 7 parallelen Tracks vermitteln ein breites Spektrum der aktuellen Java-Technologie.

Ausstellung

An etwa 40 Ausstellungsständen informieren Sie namhafte Firmen über Leistungen und Produkte.

www.java-forum-stuttgart.de

Eine Veranstaltung der:

JUGS java user group stuttgart
www.jugs.org

Mittwoch, 16. Juli 2014
Workshop-Tag
„Java für Entscheider"

Die eintägige Überblicksveranstaltung zeigt Begrifflichkeiten und wichtige Technologien aus der seit Jahren in der Industrie etablierten Plattform „Java". Ausgehend von strategischen Gesichtspunkten wie Bedeutung und Verbreitung wird der Blick über das Client-seitige Java (Java SE) und die wesentlichen Entwicklungswerkzeuge hinüber zum Server-seitigen Java (Java EE) gelenkt.

Dort stehen dann die Bedeutung von Java als Integrationsplattform und die verschiedenen Technologien im Mittelpunkt.
Weiterhin wird noch der Einsatz von Java in den immer wichtiger werdenden mobilen Lösungen (Android, iOS) gestreift, um dann den Bogen von der Softwareentwicklung hin zum Betrieb zu schlagen.

Eine Veranstaltung der:

JUGS java user group stuttgart
www.jugs.org

EFS EXPERTEN FORUM STUTTGART 2014

Freitag, 18. Juli 2014
11. Experten-Forum-Stuttgart
EFS

- 12 halbtägige Workshops in 6 parallelen Tracks
- brandaktuelle Themen aus den Bereichen **Softwareentwicklung** und **IT-Projektmanagement**
- hochklassige Workshops mit **Top-Referenten**
- kleine Gruppen, die einen **intensiven Austausch** gewährleisten
- Austausch mit den **SENS-Experten** und anderen Teilnehmern, auch in den Pausen
- sämtliche **Unterlagen aller Workshops** werden allen Teilnehmern elektronisch zur Verfügung gestellt

www.experten-forum-stuttgart.de

Eine Veranstaltung des:

SENS

Wolfgang Taschner
Chefredakteur Java aktuell

Ein Traumstart für das JavaLand

Jatumba – das Motto der Community-Konferenz hallt mir immer noch in den Ohren. Als einer von mehr als 800 begeisterten Teilnehmern aus 18 Ländern war ich am 25. und 26. März auf der JavaLand 2014 im Phantasialand Brühl dabei. Die Stimmung war hervorragend, die Leute lebten den Community-Gedanken. In einigen Jahren werden etliche von uns davon schwärmen, von Anfang an dabei gewesen zu sein.

Zu den Fakten: Die JavaLand-Veranstaltung zählt auf Anhieb zu den größten Java-Konferenzen in Europa. Mit der Location Phantasialand Brühl bot sie den Teilnehmern ein innovatives Konzept. Neben mehr als 100 Vorträgen in sieben parallelen Streams gab es die ganze Zeit über im Hackergarten ein Labor zum Experimentieren sowie einen Ort für Diskussionen und zum Kennenlernen.

Fast 95 Prozent der Teilnehmer hielten den Umfang des Vortragsprogramms und 93 Prozent die Themenvielfalt des Programms für gut oder sehr gut. Fast gleichmäßig aufgeteilt war das fachliche Themeninteresse. Angefangen bei Core Java, Java 8, Java EE und Java FX über Software-Architektur, Methoden und Tools bis hin zu JVM-Sprachen und Security war alles vertreten. Auch die fachliche Relevanz der Vorträge (85 Prozent) und die rhetorische Qualität der Vortragenden (87 Prozent) erhielten eine positive Zustimmung.

97 Prozent bewerteten die Organisation und 85 Prozent das Preis-Leistungs-Verhältnis als gut oder sehr gut. Fast drei Viertel hoben die Networking-Möglichkeiten als positiv hervor. So war es dann auch kein Wunder, dass sich mehr als 80 Prozent der Teilnehmer ausreichend informiert fühlten und 88 Prozent die Konferenz weiterempfehlen werden.

Der große Erfolg der Konferenz ist das Ergebnis der gemeinsamen Arbeit aller Beteiligten. Die DOAG Dienstleistungen GmbH hatte die Organisation übernommen und die Konferenz gemeinsam mit dem Heise Zeitschriften Verlag präsentiert. Die Java-Community in Form der mittlerweile 20 (heute 21) unter dem Dach des Interessenverbunds der Java User Groups e.V. (iJUG) vertretenen Java User Groups gestaltete das Vortrags- und Rahmenprogramm.

Auf den Seiten 8 bis 11 finden Sie ein Stimmungsbild rund um die Community-Veranstaltung.

Ich freue mich schon auf ein Wiedersehen auf der JavaLand 2015.
Ihr

JavaLandConf 2014 Twitterwall

JavaFX– das neue Gesicht für Java-Anwendungen

3 Editorial

5 Das Java-Tagebuch
Andreas Badelt,, Leiter der DOAG SIG Java

8 #JavaLandConf 2014

12 NoSQL für Java-Entwickler
Kai Spichale

16 In Memory Grid Computing mit Oracle Coherence und WebLogic Server 12*c*
Michael Bräuer und Peter Doschkinow

22 JavaFX– das neue Gesicht für Java-Anwendungen
Frank Pientka und Hendrik Ebbers

26 Java Performance-Tuning
Kirk Pepperdine

29 Logging und Apache Log4j 2.0
Christian Grobmeier

32 WSO2 App Factory: Die Industrialisierung der Software-Entwicklung
Jochen Traunecker

36 Database-DevOps mit MySQL, Hudson, Gradle, Maven und Git
Michael Hüttermann

41 Entweder ... oder Fehler
Heiner Kücker

44 Connectivity-as-a-Service für Cloud-basierte Datenquellen
Jesse Davis

48 JSF-Anwendungen performant entwickeln
Thomas Asel

52 Pimp my Jenkins
Sebastian Laag

57 Contexts und Dependency Injection – Geschichte und Konzepte
Dirk Mahler

61 Unbekannte Kostbarkeiten des SDK Heute: HTTP-Server
Bernd Müller

63 Die Softwerkskammer
Markus Gärtner

65 Java Forum Stuttgart
Tobias Frech

66 Inserentenverzeichnis

66 Impressum

Connectivity-as-a-Service für Cloud-basierte Datenquellen

Unbekannte Kostbarkeiten des SDK

Das Java-Tagebuch

Andreas Badelt, Leiter der DOAG für Java-Themen

Das Java-Tagebuch gibt einen Überblick über die wichtigsten Geschehnisse rund um Java – in komprimierter Form und chronologisch geordnet. Der vorliegende Teil widmet sich den Ereignissen im ersten Quartal 2014.

27. Januar 2014

Internet of Things at User Group Leader Summit

Aus einem „Java Source"-Blog-Eintrag: Beim „Oracle International User Group Leader Summit" gab es die unausweichliche Podiumsdiskussion zum Internet of Things (IoT). Wenig überraschend kamen drei der fünf Experten aus dem Java-Bereich: Stephen Chin („Java Evangelist" bei Oracle), Bruno Souza (Präsident der größten brasilianischen JUG „SouJava") sowie Jai Suri (Oracle IoT und Java Embedded Product Manager). „Unsichtbarkeit" ist laut Bruno Souza das Neue am IoT: „Sie lässt Rechner unsichtbar werden". Das kann aber auch negative Konsequenzen haben – nach dem Motto „aus den Augen, aus dem Sinn". Dies verdeutlichte Stephen Chin mit zwei Fragen an die Zuschauer: „Wie viele haben zu Hause einen Wireless Router?" (Viele Hände gehen nach oben). „Wie viele haben in den letzten sechs Monaten die Firmware aktualisiert?" (Die meisten Hände gehen wieder herunter). „Eine der Stärken von Java im Internet of Things", so Jai Suri, „sei das Abstraktionslevel, das eine höhere Sicherheit und schnellere Updates ermögliche, gerade für Geräte, die jahrelang draußen im Einsatz sind". Dies wird nicht der letzte Tagebuch-Eintrag zum Thema „IoT" sein - oder mit den Worten der Experten: „You are completely underestimating how big and how fast the IoT wave is coming."
http://ift.tt/1iDmx0C

6. Februar 2014

NightHacking Live – Die „Raspberry Pi"-Wand

Ein weiterer Einsatzbereich für die günstigen Mini-Rechner, präsentiert auf der Jokus-Konferenz: Ein Team bei CodeMint hat sich eine „Raspberry Pi Wall" für Lasttests gebaut. Für Kosten von insgesamt 3.000 Dollar für 48 Pis, Kabel, Netzteile und einen High-End-Switch konnten sie damit über Wochen das zu testende System mit 90.000 simultanen Verbindungen befeuern. Die Gegenrechnung: Eine vergleichbare Kapazität, etwa in der Amazon Cloud, hätte pro Monat rund 15.000 Dollar gekostet. Der Raspberry Pi als Private Cloud – auch wenn die Skalierung noch etwas Handarbeit verlangt – klingt damit recht überzeugend. Das Video ist online verfügbar.
http://nighthacking.com/performance-testing-with-a-raspberry-pi-wall-running-java

10. Februar 2014

JavaLand – die neue Konferenz im Phantasialand

Die Vorfreude auf JavaLand 2014 steigt – auch das Java-Team bei Oracle hat gesehen, dass die deutsche Java-Community hier etwas Großes auf die Beine stellt, und erwähnt die Konferenz mit einem „Java Source"-Blog-Eintrag.
https://blogs.oracle.com/java/entry/javaland_new_java_conference_in

14. Februar 2014

Java ME Embedded 8 Early Access 2

Es ist fraglich, ob Java ME Embedded 8 wie geplant gleichzeitig mit SE 8 und SE Embedded 8 im März freigegeben wird. Aber es nähert sich seiner Fertigstellung; heute wurde die Version „Early Access 2" veröffentlicht. Verbesserungen zur Version „Early Access 1" gab es in vielen Bereichen, so wurde die Vereinheitlichung der APIs und Sprachstandards mit SE/SE Embedded weiter vorangetrieben. Auf der anderen Seite hat man die Anpassungsmöglichkeiten erweitert, sodass jetzt Hardware ab 128 KB RAM und 1 MB Flash-/ROM-Speicher unterstützt werden soll. Das Thema „REST" steht auch und gerade für die Kleinen auf der Tagesordnung (neue APIs unter anderem für JSON und „async HTTP").
http://terrencebarr.wordpress.com/2014/02/14/announcing-java-me-8-early-access-2

18. Februar 2014

New Oracle Learning Library Course: Develop Java Applications Using a Raspberry Pi

Oracle startet einen „Massive Open Online Course" zum Thema „Java-Applikations-Entwicklung mit dem Raspberry Pi". Der Kurs wird von den bekannten Java-Evangelisten wie Jim Weaver und Simon Ritter geleitet, die erste Auflage startet Ende März. Noch genug Zeit, das speziell für den Kurs zusammengestellte Hardware-Paket bei adafruit.com zu bestellen – für immerhin 150 Dollar. Ob Oracle wohl an Adafruit Industries beteiligt ist? Die Hardware macht zumindest einen guten Eindruck und der Kurs an sich ist kostenlos. Das Szenario für den Kurs ist die Aufzeichnung und Weitergabe von Informationen aus und über Container, während sich diese im Transport befinden (Temperatur; Standort; Zeiten, zu denen die Türen geöffnet werden etc.). Das Internet of Things zum Anfassen.
https://blogs.oracle.com/java/entry/develop_java_applications_using_a

25. Februar 2014

Vert.x: „A Project to Watch"

Wenn das offizielle Oracle-Java-Blog ein außerhalb des Standards „Java EE" stehendes Framework für „Mobile, Web und

Enterprise Applications“ anpreist, dann heißt das etwas. Die letzten Einträge zu Spring, an die ich mich erinnern kann, hießen typischerweise „Spring zu Java EE Migration leicht gemacht“ oder so ähnlich. Andererseits ist das leichtgewichtige, Event-getriebene „vert.x“ inzwischen recht populär geworden und einer der Kandidaten, die zeigen könnten, wo es bei Java EE momentan hakt – genau die Rolle, die Spring viele Jahre innehatte.
https://blogs.oracle.com/java/entry/vert_x_a_project_to
Nachtrag: Aktuell laufen die ersten Überlegungen zu vert.x 3.0
https://groups.google.com/forum/#!topic/vertx/nLXpM5WM9qQ

26. Februar 2014

Java EE 8 Community Survey, Auswertung Teil 1 und 2
Die Auswertung der ersten beiden offiziellen Umfragen zu Java EE 8 ist da. Die wichtigste Frage im ersten Teil war sicher die zu den JSRs: Von einer deutlichen Mehrheit wurden JSON Binding, Server-Sent Events, JCache und das Configuration API als wichtig eingestuft. Weniger klar sieht es bei dem Identity API und Data Grids aus, State Management fand sogar nur 37 Prozent Unterstützer, 12 Prozent sahen es als „nicht wichtig“ an, 51 Prozent waren sich „nicht sicher“; wobei man es wohl tendenziell als negativ einstufen muss, wenn sich Entwickler bezüglich der Wichtigkeit einer Spezifikation nicht sicher sind. Ein Standard-API für JavaScript auf dem Server schnitt mit 42 Prozent Befürwortern, aber 31 Prozent direkten Gegenstimmen sogar noch schlechter ab. Generell äußerten einige Teilnehmer Bedenken sowohl wegen der Gesamt-Anzahl von Spezifikationen als auch wegen drohender Überlappung; gewünscht wird vielmehr eine bessere Abstimmung einzelner JSRs untereinander, gerade auch in Bezug auf CDI. Recht uneinheitlich war das Bild bei vielen der weiteren Fragen: MVC-Support parallel zu JSF wurde beispielsweise von einer Mehrheit unterstützt, allerdings sah nur jeder vierte Teilnehmer hier einen De-facto-Standard, an dem man sich orientieren könnte, innerhalb dieser Gruppe führten jedoch 42 Prozent Spring oder einen vergleichbaren Ansatz auf. Relativ viele Nein-Stimmen gab es bei Fragen zum generellen Einsatz von NoSQL beziehungsweise zum Standardisierungsbedarf.

Im zweiten Teil, der mehr auf die strategische Ausrichtung von Java EE abzielte, gab es knappe Mehrheiten dafür, dass SaaS beziehungsweise PaaS reif für eine Standardisierung sind (jeweils knapp 30 Prozent waren „nicht sicher“). Knappe bis deutliche Mehrheiten gab es auch bei den verschiedenen Fragen zur Standardisierung des Logging, wobei viele Kommentare auch eine generelle Überarbeitung forderten. Ähnlich sah es bei den Themen „Security“ und „Testability Support“ (Embedded Container) aus. Die Standardisierung von Multimandanten-Fähigkeit innerhalb einer Applikations-Instanz fand hingegen (noch?) keine Mehrheit. Die Cloud scheint weiter nicht zu den Prioritäten zu gehören. Insofern hat Oracle es wohl richtig gemacht, Cloud- Unterstützung nicht in den Java-EE-Standard hineinzupressen, sondern weiter abzuwarten. Viele weitere Details können hier noch aufgelistet werden. Die gesamte Auswertung ist online verfügbar.
https://java.net/downloads/javaee-spec/JavaEE8_Community_Survey_Results.pdf

9. März 2014

Java SE 8: Spezifikation vollständig
Am 25. März soll Java SE 8 freigegeben werden. Die drei neuen, ursprünglich allerdings mal für SE 7 geplanten JSRs in diesem Release („Annotations on Java Types“, „Date and Time API“ und „Lambda Expressions“) sowie der „Mantel-JSR“ selbst sind nun endgültig vom JCP Executive Committee abgesegnet und als final veröffentlicht worden; das Gleiche gilt für eine Reihe von Maintenance Releases für ältere JSRs. Es ist ja immer beruhigend, wenn die Spezifikation fertig ist, bevor eine Software live geht.

9. März 2014

Das selbstgebaute Fahrzeug-Informationssystem
Ich hatte es ja schon angekündigt: Irgendwie bleibe ich momentan immer bei den IoT-Themen hängen – vermutlich auch dem Angebot geschuldet … Simon Ritters low-cost „Carputer“ auf Basis eines Raspberry und eines OBD2-Diagnosegeräts (zusammen unter 100 Euro) ist nicht ganz neu – er hat das Projekt schon vor einem Jahr auf Konferenzen vorgestellt. Jetzt ist ein ausführliches Vortragsvideo online verfügbar.
https://blogs.oracle.com/java/entry/video_is_it_a_car

18. März 2014

Java SE 8 freigegeben
Der offizielle Launch ist erst nächste Woche, auch das JavaLand wird mit Stephen Chin und einer Podiumsdiskussion an der Live-Schaltung beteiligt sein. Heruntergeladen werden kann das offizielle JDK 8 Release 8 aber jetzt schon. Nicht, dass es keine „known bugs“ mehr enthalten würde – in den letzten Monaten bis zum Release wurden nur noch „Showstopper“ beseitigt, um den Termin zu halten. Aber bei der deutlich erhöhten Aufmerksamkeit und Beteiligung der Java-Community in Form von vielen freiwilligen (oder aus Erfahrung genötigten) Testern sollten doch zumindest alle wirklichen Probleme identifiziert sein – damit uns ein ähnliches Chaos wie mit SE 7 erspart bleibt, als einige Open-Source-Projekte wie Apache Lucene und Solr aufgrund von Änderungen an Compiler-Optimierungs-Flags nicht mehr stabil funktionierten. Neben den neuen Sprach-Features wie Lambdas sind zwei der Highlights sicherlich die neue JavaScript-Engine Nashorn und das Entfernen der „PermGen“ (Alle, die sich schon einmal mit dem Sizing von Java VMs herumplagen – sorry: beschäftigen mussten – dürfen jetzt in Jubel ausbrechen). Gleichzeitig ist mit SE 8 wie geplant auch SE Embedded 8 freigegeben worden. Die Version ME Embedded 8 benötigt offenbar noch ein wenig mehr Zeit – sie ist weiterhin nur als „Early Access“-Version verfügbar.

25. März 2014

JavaLand startet
JavaLand 2014 ist endlich eröffnet. Mehrere Dutzend Personen verschiedener deutscher Java User Groups haben fast ein Jahr daran gearbeitet – jetzt löst sich die Anspannung. Immerhin 800 Teilnehmer haben sich registriert, das ist schon ein grandioser Erfolg; sie verlieren sich allerdings fast im Phantasialand, das für zwei Tage seinen Namen geändert hat. Das Vortrags-Programm – vier Haupt-Streams und mehrere Firmen-Tracks

– ist überwältigend und ich mache mir ein bisschen Sorgen, dass die „Community Area", in der wir auf Interaktion setzen, ein bisschen zu kurz kommt. Die Sorge ist jedoch offensichtlich unbegründet. Das Java Innovation Lab zieht die Technik-Freaks automatisch an, alle wollen mal die 3D-Brille aufsetzen, um das Verschwinden der Grenzen zwischen echter und virtueller Realität („augmented reality") zu erleben; der Hackergarten ist sowieso etabliert und hat echte Fans, die immer wieder kommen, egal wo er gerade Station macht; und mit ein bisschen spontanem Marketing füllt sich auch die „Early Adopters Area", in der die Teilnehmer direkt bei JSRs und beim OpenJDK mitmischen können.
http://www.javaland.eu

Java SE 8 offizieller Launch und NetBeans 8
Zeitgleich zum Launch von Java SE 8, per Live-Stream aus Kalifornien, ist auch NetBeans 8 offiziell freigegeben worden. Neben der offensichtlichen Unterstützung für das neueste JDK sind unter anderem direktere Unterstützung der Entwicklung für „Embedded Devices", Tomcat 8.0 und TomEE-Integration sowie Debug-Unterstützung für die Nashorn-JavaScript-Engine zu nennen.

26. März 2014

JavaLand: zweiter Tag
Zweiter und schon wieder letzter Tag des JavaLands. Es gab keine der bei einer Erstauflage befürchteten größeren Pannen. Das Setting in einem Freizeitpark ist ein Volltreffer – auch wenn ich es in kein Karussell geschafft habe, aber es muss ja noch Steigerungsmöglichkeiten für das nächste Jahr geben. Das Hotel Matamba hat mehr gehalten, als die drei Sterne versprochen haben. Alle Teilnehmer, Referenten und Organisatoren, mit denen ich gesprochen habe, waren ebenfalls begeistert. Ein besonderes Kompliment von Anatole Presch, einem unserer Experten in der „Early Adopters Area": „Diese Konferenz war für mich genauso wichtig wie die JavaOne." Fazit: Keine Frage - wir kommen nächstes Jahr wieder!
http://www.ijug.eu/home-ijug/aktuelle-news/article/800-java-enthusiasten-fuellen-javaland-mit-leben-und-ideen-auf.html

27. März 2014

Das Gemalto-Board
Nachdem ich so viel über den Raspberry Pi geschrieben habe, will ich die Konkurrenten nicht verschweigen: Etwa das Gemalto-Board – immerhin mit eingebautem 2G- und 3G-Support. Im Java-Source-Blog gibt es einen Eintrag mit Videos, die einen Überblick über das Board zeigen.
https://blogs.oracle.com/java/entry/gemalto_concept_board2

31. März 2014

Java EE 8 Community Survey, Auswertung Teil 3
Im dritten und letzten Teil der offiziellen Umfragen zu Java EE 8 durften Teilnehmer 100 Punkte auf die insgesamt 13 Bereiche verteilen, die in den ersten Teilen eine deutliche Unterstützung erfahren haben (mindestens 60 Prozent). Deutlicher Gewinner mit 13,7 Prozent ist „JSON Binding", gefolgt von Security Simplification und JCache mit 11,1 Prozent beziehungsweise 9,5 Prozent. Auf den nächsten Plätzen folgen unter anderem Security Interceptors für CDI, MVC-Support und Configuration. Weit abgeschlagen sind „Pruning" (das Entfernen veralteter Spezifikations-Teile) und EJB-Timer mit rund 4 Prozent.
https://blogs.oracle.com/ldemichiel/entry/results_from_the_java_ee

8. April 2014

Zweite Auflage des Embedded-Applications-Kurses
Wie schon angekündigt, startet Oracle eine weitere Auflage des „Massive Open Online Course" zur Embedded-Applikationsentwicklung unter anderem mit dem Raspberry Pi.
https://blogs.oracle.com/java/entry/oracle_massive_open_online_course

14. April 2014

Neues vom JCP
Im Rahmen der JCP-Next-Initiative zur Generalüberholung des Java-Community-Prozesses sind zwei JSRs („Transparenz" und „Zusammenlegung der Executive Committees") schon seit Längerem umgesetzt. Nummer drei (JSR 358: „A major Revision to the Java Community Process"), der sich insbesondere mit den rechtlichen Problemen rund um das „Java Specification Participation Agreement" (JSPA) und „Intellectual Property" beschäftigt, ist allerdings ins Stocken geraten – wie vielleicht zu erwarten, wenn die Juristen ins Boot kommen. Um Fortschritte in der Zusammenarbeit mit individuellen JCP-Mitgliedern und Java User Groups zu erzielen, für die andere Voraussetzungen gelten als für das juristisch größere Minenfeld der Beteiligung von Firmen am JCP, ist nun ein vierter JSR quasi aus JSR 358 ausgegründet worden: JSR 364 „Broadening JCP Membership". Bis zum 12. Mai 2014 wird nun abgestimmt, ob dieser JSR tatsächlich auf die Reise gehen soll – wobei eigentlich keine Ablehnung zu erwarten ist, da durch die besondere Entstehungsgeschichte praktisch alle EC-Mitglieder von Anfang an involviert gewesen sein sollten.
https://www.jcp.org

17. April 2014

Java Secure Coding Guidelines überarbeitet
Die Java Secure Coding Guidelines sind für das neue JDK 8 überarbeitet worden, unter anderem, um Patterns mithilfe von Lambda-Konstrukten zu vereinfachen. Nichts spektakulär Neues, aber dringend empfohlene Lektüre für alle, die sich noch nicht intensiv mit dem Thema auseinandergesetzt haben.
http://www.oracle.com/technetwork/java/seccodeguide-139067.html

Andreas Badelt
Leiter der DOAG SIG Java

Andreas Badelt ist Senior Technology Architect bei Infosys Limited. Daneben organisiert er seit 2001 ehrenamtlich die Special Interest Group (SIG) Development sowie die SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V.

#JavaLandConf

Eindrücke von der JavaLand2014 im Phansialand in Brühl

JavaLandConf @JavaLandConf
The #Java Community Developers Conference now has a twitter handle! Welcome :)
8. Januar 2014

adoptajsr @adoptajsr
We have a nice conversation yesterday with @jcp_org, & organisers of #Javaland - good progress so far! Looking fwd to next meeting!
11. Januar 2014

Markus Eisele @myfear
Talking about the #java8 launch with @java @steveonjava @speakjava and jug leaders! Be excited! pic.twitter.com/F1z1Lr1v4p
23. Januar 2014

Tobias Frech @TobiasFrech
Will try to discover #JavaLand today. Rumors are it's located somewhere near cologne. Heading there ...
6. Februar 2014

Anton Epple @monacotoni
just received my #javaland ticket :-). pic.twitter.com/vyHCzEg9Q5
11. Februar 2014

Oracle Tech Network @oracleotn
#JavaLand: New Java Conference in Germany
11. Februar 2014

Andres Almiray @aalmiray
hey JavaLanders! don't miss your chance to have fun at the #hackergarten
13. Februar 2014

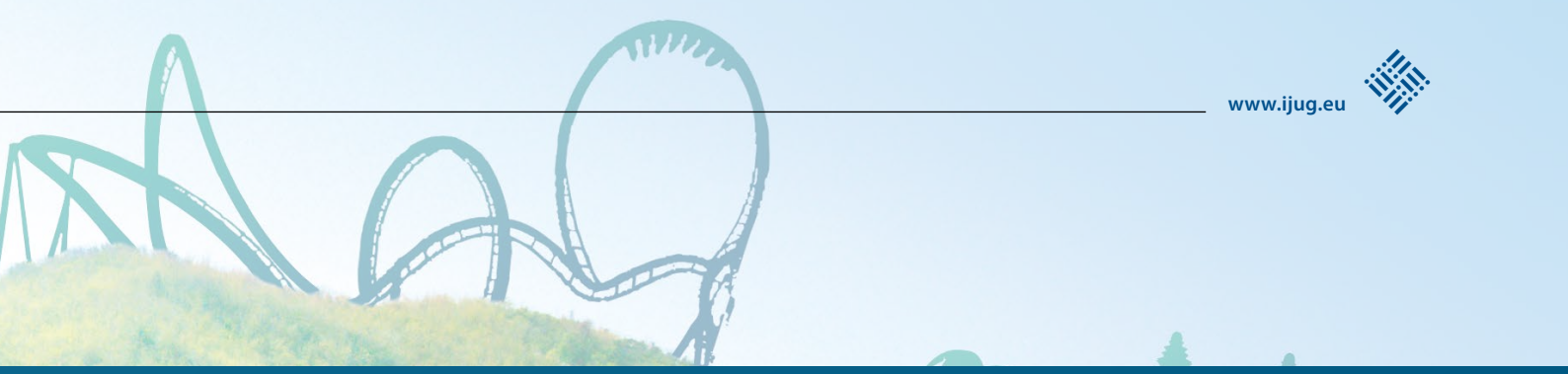
Alex Soto @alexsotob
I am proud to take over @mojavelinux at #JavaLand to speak about #JRuby and #asciidoctor, ty so much.
14. Februar 2014

André Sept @andre_sept
experience the oculus rift at #JavaInnovationLab
More informations comming soon
21. Februar 2014

edburns @edburns
I am plugging #javaland during my #devnexus talks. There will be a strong community presence for Adopt-A-JSR.
24. Februar 2014

Tobias Frech @TobiasFrech
With the stages for @EclipseFdn, @codecentric, @IrianTeam, @Oracle and @JavaLandConf there will be 7 streams for 2 days
5. März 2014

Oliver White @TheOTown
Wow, excited about @JavaLandConf - many friends coming!
7. März 2014



Heather VanCura @heathervc
Happy International Women's Day to all of the women in the #java community.
9. März 2014

Arun Gupta @arungupta
@myfear #JavaLand content is already thrilling, this will only accentuate that ;-)
10. März 2014

Paul Bakker @pbakker
With talks at #JavaLand, #DevNation and #GeeCon it already turns out to be an interesting conference year.
20. März 2014

Simon Ritter @speakjava
Going to be at #JavaLand next week with @sjmaple, @arungupta, @hansolo_ @myfear and others. Java will be discussed, beer will be drunk.
21. März 2014

tomitribe @tomitribe
If you are attending @JavaLandConf this week, catch @dblevins talk on Apache TomEE, Java EE Web Profile and More on Tomcat
24. März 2014

JavaLandConf @JavaLandConf
Bus shuttle information http://www.javaland.eu/location.html
24. März 2014

Simon Ritter @speakjava
@JavaLandConf today. #Carputer at 10 and 55 New Features in Java SE 8 at 12. Just in time for the official webcast, http://bit.ly/1hPepYw
25. März 2014

Arun Gupta @arungupta
Come join us at #JavaEE7 #hackergarten at #JavaLand, lots of fun around #WildFly #Arquillian and philosophical discussions ;-)
25. März 2014

Simon Olofsson @solofs
Perfektes Wetter für die @JavaLandConf pic.twitter.com/5EQPfhEaKh
25. März 2014

André Sept @andre_sept
Oculus rift live at 2 pm at the community hall with @aronbinienda moderated by @steveonjava #JavaLand @JavaLandConf pic.twitter.com/IJRdg6pt9R
25. März 2014

Joachim Weinbrenner @weinbrenner
My talk went quite well, lots of visitors!
25. März 2014

JATUMBA!

Das Motto der JavaLand 2014

Arun Gupta @arungupta
#javaland countless people, days, nights, weekends go in creating a new conference. Many thanks to team driving show
25. März 2014

Alex Heusingfeld @goldstift
The catering concept at #javaland is really great! I wonder if this is more expensive than the concept of other conferences.
25. März 2014

Bruno Borges @brunoborges
Why is @JavaLandConf trending? Well add the # of experts reunited since last @JavaOneConf and the perfect timing launch of #Java8. There. :)
25. März 2014

Bert Ertman @BertErtman
So far @JavaLandConf is a very enjoyable conference. Kudos to the team!
25. März 2014

NightHacking @_nighthacking
Huge crowd at #JavaLand for the #Java8 launch!
25. März 2014

Dominik Sandjaja @dadadom
Großartige Veranstaltung bisher! Man merkt nicht, dass es eine Premiere ist! Weiter so!
25. März 2014

Anatole Tresch @atsticks
@JavaLandConf is one of the geekiest conferences I have been: simply great, with awesome atmosphere and awesome talks.
25. März 2014

Neues von InterFace @InterFaceAG
Wir wünschen den Gewinnern des J-Pardy viel Spaß mit den Raspberry Pi's!
26. März 2014

Daniel Bryant @taidevcouk
Great talk on #TomEE by @dblevins at @JavaLandConf The more I see of the JEE stuff, the more I like it.
26. März 2014

André Sept @andre_sept
Full house at keynote with @heinzkabutz #JavaLand @JavaLandConf pic.twitter.com/m7X2d0dZ95
26. März 2014

JavaLandConf @JavaLandConf
RT @hseeberger: "@exit666: Sehr gut gemachte Einführung in Scala von @hseeberger in #JavaLand" << Danke!
26. März 2014

Stephen Chin @steveonjava
#NightHacking interview with @noctarius2k at @JavaLandConf on keeping your memory off the heap: http://nighthacking.com
26. März 2014

Simon Olofsson @solofs
Wow @JavaLandConf was a #great #conference
26. März 2014

Bruno Borges @brunoborges
Glad @JavaLandConf is over. My suffering can stop now... :P
26. März 2014

Rabea Gransberger @rgransberger
Goodbye @JavaLandConf . Thx to the organisers and people who helped to create a new conference experience. Safe travels home everybody.
26. März 2014

Matthias Herlitzius @mherlitzius
Spent two amazing days in #JavaLand and looking forward to coming back next year. Thanks for the great organization.
26. März 2014

Dominik Dorn @domdorn
Had an awesome time at @javalandconf and rode the black mamba 8 times - head is still spinning :) looking forward to next year!
26. März 2014

Alex Soto @alexsotob
Dear @Devoxx you have got a serious competitor called @JavaLandConf . Proud to have these two amazing conferences in Europe
27. März 2014

Marcus Kröger @subft1
Had a great time @JavaLandConf and a good talk about @RabbitMQ . Finally could help others to sort out their problems. That's what's it about
27. März 2014

Simon Maple @sjmaple
My time in Germany is almost up. Had a really enjoyable visit and had time to speak with friends, old and new. @JavaLandConf rocked :)
28. März 2014

Amelia Eiras @ameliaeiras
Thank you @myfear @JavaLandConf!!! What an AMAZING & LEGENDARY time. With @dblevins, we loved seeing old friends & meeting new ones!
28. März 2014

Arun Gupta @arungupta
#JavaLand evangelist ? Used the joggers shirt for workout, wearing the hoodie since 2 days, telling people how great the conference was!
28. März 2014

See you again next year at JavaLand 2015
Find out more on www.javaland.eu soon.

NoSQL für Java-Entwickler

Kai Spichale, adesso AG

Relationale Datenbanken gehören für viele Entwickler zur täglichen Arbeit und Technologien wie JDBC, JPA und SQL zählen seit Langem zum Standardrepertoire. NoSQL-Datenbanken funktionieren jedoch grundsätzlich anders. Dieser Artikel zeigt, welche Möglichkeiten diese neuen Datenbanken bieten und wie sie für Java-Anwendungen genutzt werden können.

NoSQL-Datenbanken sind im Vergleich zu RDBMS eine sehr inhomogene Gruppe. Es gibt zum Beispiel Datenbanken, die ihre Daten in Graphen organisieren, und andere, die einfache Key-Value-Paare nutzen. Daher ist die Einteilung in vier Kategorien gebräuchlich. Für jede dieser Kategorien soll in diesem Artikel je ein Vertreter betrachtet werden, um die Vielseitigkeit von NoSQL zu berücksichtigen:

- Redis (http://redis.io/) ist ein Key-Value-Store, der alle Daten im Arbeitsspeicher vorhält und deswegen sehr schnell ist. Das Besondere an Redis ist, dass die Daten regelmäßig auf Festplatte geschrieben werden können. Ein weiterer spezieller Vorteil ist die Unterstützung von komplexeren Datentypen wie Hash, List und Set, sodass nicht nur einfache Key-Value-Paare auf String-Basis von Redis verwaltet werden können. Ein weiteres interessantes Feature ist Publish/Subscribe Messaging. Datenmodell und Abfragesystem sind trotzdem vergleichsweise einfach.
- Cassandra (http://cassandra.apache.org/) ist ein Wide Column Store, der durch redundante Verteilung der Daten auf mehrere Server Hochverfügbarkeit und Skalierbarkeit bietet. Die Daten sind in tabellenähnlichen Spaltenfamilien organisiert, die weder JOIN-Operationen noch Aggregationen erlauben, aber unterschiedlich viele Spalten enthalten können, um semistrukturierte Daten zu speichern.
- Noch flexibler ist das Datenmodell von MongoDB (http://www.mongodb.com/). Anwendung finden hier JSON-Dokumente, die kein starres Schema einhalten müssen und sehr flexibel aufgebaut werden können. Zu den Stärken von MongoDB gehören ebenfalls die umfangreichen Abfragemöglichkeiten, zu denen ein Aggregationsframework und MapReduce zählen. Mithilfe des Aggregationsframeworks können verschiedene Aggregationsmethoden in einer Pipeline miteinander kombiniert werden, um das gewünschte Endergebnis zu berechnen. MapReduce bietet ein alternatives Aggregationskonzept, das bei MongoDB vor allem dann zum Zug kommt, wenn die bereitgestellten Aggregationsmethoden nicht ausreichend sind.
- Neo4j (http://www.neo4j.org/) ist eine Graphendatenbank, die mit den Graphen ein äußerst universelles Datenmodell bietet. Das Traversieren der Graphen ist effizient umgesetzt und erlaubt auch komplexe Abfragen.

Key-Value-Store: Redis

Ein wichtiger Redis-Client, dessen Weiterentwicklung sehr aktiv ist, ist Jedis (https://github.com/xetorthio/jedis). Listing 1 zeigt,

```
// Initialisierung eines Pools
JedisPool pool = new JedisPool(new JedisPoolConfig(), "localhost");
...
Jedis jedis = pool.getResource();
try {
  Transaction t = jedis.multi();
  t.set("key", "value");
  Response<String> response1 = t.get("key");
  jedis.zadd("vehicles", 0, "car");
  jedis.zadd("vehicles", 0, "bike");
  Response<Set<String>> response2 = jedis.zrange("vehicles", 0, -1);
  // Transaktion ausführen
  t.exec();
  String value = response1.get();
  int size = response2.get().size();
} finally {
  // Ressourcen freigeben
  pool.returnResource(jedis);
}
...
// Pool schließen wenn die Applikation beendet wird
pool.destroy();
```

Listing 1: Datenbankzugriff mit Jedis

wie Daten in Redis mithilfe von Schlüsseln abgelegt und abgerufen werden können. Die zentrale Klasse des Jedis-API ist redis.clients.jedis.Jedis. Da diese Klasse nicht Thread-sicher ist, wird die Verwendung mehrerer Instanzen für eine Multithread-Anwendung empfohlen. Ein Pool dient der Verwaltung der Jedis-Instanzen. Es ist jedoch auch möglich, direkt mit dem Konstruktor Jedis-Instanzen zu erzeugen. Das Beispiel zeigt ebenfalls die Verwendung eines Sets, mit dem unter einem Schlüssel mehrere Werte gespeichert werden können. Mit der Methode zrange(„vehicles", 0, -1) werden alle im Set enthaltenen Werte abgerufen.

Jedes Kommando in Redis ist atomar. Falls mehrere Kommandos zu einem atomaren Vorgang verbunden werden sollen, können diese innerhalb einer Redis-Transaktion ausgeführt werden. Dabei ist zu beachten, dass die Response-Objekte erst nach Ausführung von t.exec() die abgerufenen Daten enthalten.

Eine Redis-Transaktion garantiert, dass alle Kommandos der Redis-Transaktion in der vom Client vorgegebenen Reihenfolge nacheinander ausgeführt werden. Redis garantiert auch, dass keine Kommandos anderer Clients zwischenzeitig ausgeführt werden. Nach dem Alles-oder-nichts-Prinzip sind entweder alle Kommandos erfolgreich oder keines.

Einen anderen Ansatz zur Ausführung von Operationen bietet das Pipelining. Hierbei werden mehrere Operationen abgesendet, ohne auf deren einzelne Responses zu warten, sodass ein Performance-Vorteil entsteht. Die Ergebnisse können, analog zu den Transaktionen, erst nach Aufruf der abschließenden Synchronisation abgerufen werden.

Jedis kann keine kompletten Entitäten speichern, wenngleich es natürlich möglich ist, Java-Objekte zu serialisieren, um diese als String oder Bytearray zu speichern. Praxistauglicher ist allerdings die Speicherung von Objekten im JSON-Format oder in noch kompakteren Formaten wie MessagePack (http://msgpack.org/). Einen ganz anderen Ansatz verwenden Objekt-Hash-Mapper. Diese benötigen jedoch bei großen oder komplexen Objekten viele Zugriffe, um ein Objekt zu laden, da jedes Attribut mit einem separaten Key-Value-Paar erfasst wird.

Wide Colum Store: Cassandra

Cassandra ist bekannt für gute Skalierung und hohen Datendurchsatz. Mit der Cassandra Query Language (CQL), einer SQL-ähnlichen Sprache, erfolgt sowohl die Schemaverwaltung als auch der Datenzugriff. Moderne Client-Bibliotheken wie Cassandra-JDBC (http://tinyurl.com/n56jafa) und Kundera (http://tinyurl.com/k2dx6lq) bieten Unterstützung für CQL in der aktuellen Version 3.0. Mithilfe von CQL können beispielsweise SELECT-Abfragen mit WHERE-Klauseln formuliert werden.

```
SELECT * FROM User WHERE usr_id IN (4, 7) ORDER BY usr_name ASC
USING CONSISTENCY QUORUM;
```

Listing 2: CQL-Abfragen mit Konsistenzstufe Quorum

```
@Entity
@Table(name = "user", schema = "example@example_pu")
public class User {
  @Column(name="usr_id")
  private String id;

  @Column(name="usr_name")
  private String name;
  ...
}
```

Listing 3: Objekt-Mapping mit Kundera

Die Abfrage in Listing 2 selektiert alle Datensätze in Tabelle User mit den Ids 4 und 7 aufsteigend sortiert nach name. JOIN-Operationen und Aggregationen sind nicht möglich. Dies ist ein Kompromiss zugunsten der skalierbaren Architektur. Cassandra arbeitet mit Multi-Master-Replikation, was bedeutet, dass es keine zentrale Komponente gibt, die entscheidet, welches Replikat führend ist. Schreib- und Leseoperationen werden von allen Replikaten akzeptiert. Der Client bestimmt die Konsistenzstufe der durchgeführten Operation. Die Angabe USING CONSISTENCY QUORUM in Listing 2 bedeutet, dass mehr als die Hälfte aller Replikate antworten müssen, um diese Abfrage erfolgreich abzuwickeln. Mit diesen Einstellungen kann der Client den Kompromiss aus Konsistenz und Performance bestimmen, der für den jeweiligen Anwendungsfall passend ist.

Mit der bereits genannten Bibliothek Cassandra-JDBC können CQL-Abfragen in Java-Anwendungen ausgeführt werden. Dank des JDBC-Charakters können DataSources und JDBC-Verbindungen analog zu relationalen Datenbanken definiert werden.

Client-Bibliotheken wie Kundera können Java-Objekte mit einem Mapper auf das Cassandra-Datenmodell abbilden. Im Gegensatz zu den einfacheren Key-Value-Stores ist dieser Komfort hier durchaus sinnvoll, da die Spalten einer Zeile in einer Tabelle sequenziell gelesen und geschrieben werden können. Zudem ist das Lesen und Schreiben auch mehrerer Spalten einer Zeile eine atomare Operation.

Listing 3 zeigt ein Objekt-Mapping-Beispiel mit der Entität User, die mit ihren Instanzvariablen id und name auf die Tabelle user abgebildet ist. Das Beispiel erinnert nicht nur an das Java Persistence API (JPA), sondern ist es auch. Selbst die Java Persistence Query Language (JPQL) ist mit Einschränkungen von der Kundera-Bibliothek umgesetzt worden. Auch Relationen zwischen Objekten, wie beispielsweise OneToOne und ManyToMany, werden unterstützt. Die Verknüpfungen müssen jedoch im Client aufgelöst werden, da Cassandra nativ keine JOIN-Operationen bietet. Ein intensiver Gebrauch dieser Relationen ist mit Cassandra daher nicht empfehlenswert. Ob JPA als Ansatz für Cassandra sinnvoll ist, sollte kritisch hinterfragt werden. ACID-Transaktionen werden ebenfalls nicht von Cassandra unterstützt.

Eine andere Client-Bibliothek, Firebrand OCM, unterstützt nur OneToOne und OneToMany-Beziehungen, denn die verknüpften Objekte können eingebettet in der Zeile des

Ursprungsobjekts gespeichert werden. Dies geschieht jedoch in einem proprietären Format, das nicht von anderen Client-Bibliotheken oder Werkzeugen gelesen werden kann.

In Cassandra kann potenziell jede Tabellenzeile andere Spalten enthalten. Komplexere Datenstrukturen, die etwa durch Schachtelung von Key-Value-Paaren entstehen, sind nicht möglich. Falls dies erforderlich ist, sollte der Einsatz einer Dokumentendatenbank in Erwägung gezogen werden.

Dokumentendatenbank: MongoDB

MongoDB nutzt JSON-Objekte zum Datenaustausch zwischen Client-Anwendung und Datenbank. Die JSON-Dokumente haben den Vorteil, dass sie nicht nur aus einfachen Key-Value-Paaren aufgebaut sind, sondern tief geschachtelte Datenstrukturen enthalten können. Diese Flexibilität macht MongoDB zu einer der populärsten NoSQL-Datenbanken. Listing 4 bis 6 veranschaulichen den Einsatz von Spring Data MongoDB. Mit diesem Framework kann die Dokumentendatenbank in Java-basierte Anwendungen integriert werden. Im gezeigten Beispiel soll erneut eine Entität User auf das nicht-relationale Datenmodell abgebildet werden. Der Unterschied zum Beispiel in Listing 3 sind die eingebetteten Adressen, die als JSON-Array gespeichert werden. Neben dem Primärschlüssel id hat User das Attribut name, das dank eines Sekundärindizes effizient in Abfragen verwendet werden kann.

Spring Data reichen schon wenige Zeilen Quellcode, um ein Repository zu definieren. Das Repository kapselt Details über den Datenbankzugriff. Standardmäßig enthält das Repository Methoden zum Speichern und Laden von Entitäten. Eine Implementierung des Repository-Interface muss nicht geschrieben werden, denn Spring Data erzeugt zur Laufzeit eine passende Implementierung und injiziert diese an den gewünschten Stellen.

Das Interface in Listing 5 enthält eine benutzerdefinierte Finder-Methode zur Suche nach Benutzern mit einem bestimmten Namen. Die Suchparameter werden mit der Annotation @Query direkt im Interface definiert.

Mit Listing 6 wird das Spring-Data-Beispiel komplettiert. Das Repository für die User-Dokumente kann in typischer Spring-Manier in die Service-Klasse mit @Autowired injiziert werden.

Operationen auf einzelnen Dokumenten sind in MongoDB atomar. Durability (das „D“ in ACID) kann durch Aktivierung des Journaling sichergestellt werden. Mit Journaling können nicht ordnungsgemäß gespeicherte Daten wiederhergestellt werden, falls mongod, der primäre Daemon-Prozess eines MongoDB-Systems, abrupt beendet wurde. In Abhängigkeit vom konfigurierten Write Concern meldet MongoDB Fehler bei Schreiboperationen. Der Client kann so bestimmen, ob beispielsweise die Bestätigung der Schreiboperation auch auf den Replikaten erforderlich ist.

Spring Data MongoDB ist Teil des umfassenden Spring-Data-Projekts (http://projects.spring.io/spring-data/), das ein konsistentes, Spring-basiertes Programmiermodell für unterschiedliche Datenbanken bietet. Die Hauptprojekte umfassen JPA, Redis, JDBC Extensions, MongoDB, Gemfire, Neo4j und Hadoop.

Mit den drei bisher beschriebenen NoSQL-Datenbanken wurden auch drei sehr unterschiedliche Datenmodelle vorgestellt. Trotz der vielen Unterschiede gibt es eine konzeptionelle Gemeinsamkeit: Die Abwesenheit von Relationen zwischen Objekten. Referenzielle Integrität und Normalformen, wie man sie aus der Welt der relationalen Datenbanken kennt, sind hier Fremdwörter. Zwar bietet auch MongoDB sogenannte „DBRefs“, um Referenzen zwischen Dokumenten zu speichern. Diese Referenzen sind jedoch keine Fremdschlüssel, denn weder JOIN-Operationen noch Constraints sind damit möglich.

Welchen Vorteil haben diese Datenmodelle ohne Relationen? Die gespeicherten Objekte können auf verschiedene Server des Datenbanksystems verteilt werden. Wären die Objekte durch Relationen miteinander verbunden, wäre das Lesen und Schreiben eines Objekts möglicherweise von anderen Objekten auf anderen Servern abhängig. Ohne die Relationen können die Objekte anhand ihrer Schlüssel leicht verteilt werden.

Graphendatenbank: Neo4j

Die vierte NoSQL-Kategorie, die der Graphendatenbanken, fällt etwas aus dieser Reihe, denn hier sind die gespeicherten Objekte durch ein komplexes Netzwerk

```
@Document
public class User {
  @Id private String id;
  @Indexed private String name;
  private Collection<Address> addresses;
  ...
}
```

Listing 4: Objekt-Mapping mit Spring Data MongoDB

```
public interface UserRepo extends MongoRepository<User, String> {
  @Query("{ name: ?0 }")
  List<User> findByUserName(String name);
}
```

Listing 5: Repository für Entität User

```
public class UserService {
  @Autowired UserRepo repo;

  public void anonymizeUserData(User user) {
    repo.save(anonymize(user));
  }
  ...
}
```

Listing 6: Speichern von Dokumenten mit dem Repository

```
GraphDatabaseService graphDb = new GraphDatabaseFactory().
newEmbeddedDatabase(DB_PATH);

DynamicRelationshipType friend = DynamicRelationshipType.
withName("FRIEND");

Transaction tx = graphDb.beginTx();
try {
 Node user1 = graphDb.createNode();
 user.setProperty("name", "user1");
 Node user2 = graphDb.createNode();
 alex.setProperty("name", "user2");
 user1.createRelationshipTo(user2, friend);
 tx.success();
} finally {
 tx.finish();
}
```

Listing 7: Speichern eines Graphen mit Spring Data Neo4j

```
@NodeEntity
public class User {
  @GraphId private Long id;

  @RelatedTo(direction = Direction.BOTH, elementClass = User.
class, type = "FRIEND")
  private Set<User> friends;
  ...
}
```

Listing 8: Objekt-Mapping mit Spring Data Neo4j

aus Relationen miteinander verbunden. Die Objekte können aus diesem Grund nicht wie bei den anderen drei genannten NoSQL-Vertretern auf mehrere Maschinen verteilt werden. Die zentrale Datenhaltung hat jedoch den Vorteil, dass ACID-Transaktionen unterstützt werden. Für die Integration mithilfe eines Java-Clients eignet sich das schon erwähnte Spring Data Neo4j, das in Listing 7 vorgestellt wird.

Das Beispiel in Listing 7 startet Neo4j „embedded", was bedeutet, dass Neo4j in derselben JVM wie die Applikation ausgeführt wird. Der Zugriff erfolgt über den Dienst GraphDatabaseService. Innerhalb der Transaktion wird ein kleiner Graph, bestehend aus zwei Knoten, die durch eine Kante verbunden sind, erzeugt. Dies entspricht user1 und user2 sowie der Relation FRIEND. Alternativ kann auch ein Objekt-Mapping, wie dies Listing 8 zeigt, verwendet werden. Die Relation FRIEND wird durch die Annotation @RelatedTo definiert.

Fazit

Die NoSQL-Datenbanken bieten vielseitige Alternativen für die etablierten relationalen Datenbanksysteme. Das Spektrum reicht von Key-Value-Stores bis zu Graphendatenbanken. In diesem Artikel wurde gezeigt, dass für Java-Entwickler zahlreiche Client-Bibliotheken zur Auswahl stehen. Die hier vorgestellten NoSQL-Datenbanken werden in zahlreichen anspruchsvollen Gebieten eingesetzt und sind fit für den Einsatz in Enterprise-Systemen.

Kai Spichale
kai.spichale@adesso.de
http://spichale.blogspot.de

Kai Spichale ist Senior Software Engineer bei der adesso AG. Sein Tätigkeitsschwerpunkt liegt in der Konzeption und Implementierung von Java-basierten Software-Systemen. Er ist Autor verschiedener Fachartikel und hält regelmäßig Vorträge auf Konferenzen.

In Memory Grid Computing mit Oracle Coherence und WebLogic Server 12*c*

Michael Bräuer und Peter Doschkinow, ORACLE Deutschland B.V. & Co. KG

Im vierten Beitrag der Artikelreihe zum Oracle WebLogic Server 12c (12.1.2) verlassen wir das reine Java-EE-Umfeld. Sollen Daten und Zustand in Form von Objekten aus verschiedenen JVMs wie Application-Server-Knoten heraus gemeinsam genutzt werden, stellt sich die Frage, wie und wo diese Datenverwaltung erfolgt.

Dieser Artikel erklärt zunächst den Begriff „In Memory Data Grid" und stellt dann eine spezielle Implementierung, nämlich Oracle Coherence, und deren Verwendung aus Entwicklerperspektive vor. Im Anschluss daran wird auf die Integration von Oracle Coherence und WebLogic Server eingegangen, um zu zeigen, wie man Coherence in Java-EE-Anwendungen nutzen kann, die auf dem WebLogic Server laufen. Abschließend wird noch ein Blick auf die Coherence-Unterstützung im „Oracle Enterprise Pack for Eclipse" geworfen.

Java In Memory Grid Computing

Um den Einsatz eines In Memory Data Grid zu motivieren, stelle man sich eine Java-EE-basierte Shop-Anwendung vor, mit der ein Anwender in der Lage sein soll, einen Warenkorb mit Produkten zu füllen und die bestellten Produkte per Rechnung zu bezahlen. Zudem soll der Anwendungsbenutzer selbst in der Lage sein, die eigenen Benutzerdaten zu ändern, also etwa die Lieferadresse anzupassen.

Es gibt in diesem Fall mindestens drei Arten von Daten, mit denen die Anwendung umgehen muss: Bestelldaten, Benutzerprofile und Produktkatalogdaten. Ein erster Entwurf könnte vorsehen, alle Daten innerhalb einer Datenbank persistent abzulegen. Das hat jedoch den Nachteil, dass bei wiederholtem Zugriff auf dieselben Objekte diese immer wieder neu gelesen und instanziiert werden müssen. Deshalb wird man darüber nachdenken, zum Beispiel die Benutzerprofile und auch den gesamten Produktkatalog aus Gründen des schnellen Lesezugriffs in jeder JVM vorzuhalten, um ein zu verschiedenen Zeitpunkten wiederkehrendes Erzeugen von Objekten zu vermeiden. Das wirft jedoch Fragen auf:

- Muss man auf ein Caching verzichten, wenn die Menge der Daten, die vorgehalten werden soll, die Kapazität einer JVM übersteigt? Zum Beispiel, wenn der anfangs sehr kleine Produktdatenkatalog in seiner Größe auf ein Vielfaches der Heap-Size anwächst.
- Aufgrund des Benutzeraufkommens wird ein Mehrknoten-Cluster auf Application-Server-Ebene benötigt. Wie kann man sicherstellen, dass der Benutzer, dessen Anfragen auf irgendeiner JVM bearbeitet werden, sein Benutzerprofil in dieser JVM vorliegen hat, ohne dass die Benutzerdaten über die Application-Server-Knoten hinweg repliziert werden müssen? Anders gefragt: Kann man eine Replikation der Daten aus obigen Kapazitätsgründen vermeiden, die Daten intelligent verteilt ablegen und auf diese von überall aus zugreifen?
- Wie kann man Abfragen und Berechnungen (wie Aggregationen) über die vorgehaltenen Daten über JVMs hinweg durchführen? Sind diese gegebenenfalls parallel möglich?
- Was passiert, wenn das Datenvolumen weiter wächst? Wie kann man Datenverarbeitungskapazität hinzufügen und bei Bedarf wieder verringern?

Um diese und weitere Fragen zu beantworten, liegt es nahe, benötigte verteilte Datenhaltung zu delegieren und spezielle Funktionalität innerhalb der Anwendung einzuführen, die sich um das intelligente Vorhalten, die Verwaltung und die Verarbeitung von Anwendungsobjekten kümmert. Diese Funktionalität können In Memory Data Grids bereitstellen. Angelehnt an [1] sind sie wie folgt charakterisierbar: „Ein In Memory Data Grid ist ein verteiltes [In-Memory]-Datenmanagement-System zum Verwalten und Verarbeiten von Anwendungsobjekten. Es zeichnet sich durch eine geringe Latenz bei Zugriffen, einen hohen Durchsatz, vorhersehbare Skalierbarkeit, hohe Verfügbarkeit und ein robustes Verhalten aus."

Am Markt gibt es verschiedene Java-basierte In-Memory-Data-Grid-Implementierungen. Auch zwei JSRs beschäftigen sich mit diesem Thema, wobei sich JSR-107 „JCACHE - Java Temporary Caching API" mit (verteiltem) Caching – einem Aspekt eines In Memory Data Grid – auseinandersetzt [2]. Während es dieser JSR nach 13 Jahren im März 2014 zur finalen Version 1 geschafft hat, befindet sich der JSR-347 „Data Grids for the Java Platform" noch im Formierungsprozess (Stand 20. März 2014, [3]).

Oracle Coherence

Im Jahr 2007 übernahm Oracle die Firma Tangosol, Pionier im Bereich Java In Memory Data Grid Computing, die schon im Jahr 2001 die erste Version einer Software namens „Tangosol Coherence" am Markt positionierte. Bei Oracle Coherence handelt es sich somit um ein über viele Jahre am Markt bewährtes und ausgereiftes Produkt.

Ein Coherence-Cluster besteht aus mehreren Coherence-Knoten, die üblicherweise

```
/u01/app/oracle/jdk/jdk1.7.0_25/bin/java -server -showversion -cp
/u01/app/oracle/fmw/coherence12.1.2/coherence/lib/coherence.jar -Xms4096m -Xmx4096m
-Dtangosol.coherence.cluster=simplecluster1
-Dtangosol.coherence.distributed.localstorage=true
-Dtangosol.coherence.cacheconfig=../config/my-cache-config.xml
-Dtangosol.coherence.override=../config/my-tangosol-coherence-override.xml com.tangosol.net.DefaultCa-
cheServer 1> ../logs/node_0_simplecluster1_pid2267.log 2>&1 &
```

Listing 1: Starten eines Coherence Cache Servers (Storage-Knoten) per Kommandozeile

über mehrere Maschinen verteilt sind. Jeder Coherence-Knoten ist eine JVM, die beim Start mit gewissen Konfigurationsinformationen versehen wird und die Bibliothek „coherence.jar" im Klassenpfad beinhaltet (siehe Listing 1). Die Konfiguration kann per XML und/oder Java-System-Eigenschaften erfolgen und es erfolgt bei Angabe solcher Konfigurationsinformationen ein Überschreiben der Default-Eigenschaften, die in „coherence.jar" definiert sind.

Bei der Cluster-Kommunikation kommt Tangosol Cluster Member Protocol (TCMP) zum Einsatz. Es basiert auf einer Kombination von UDP und TCP; wahlweise kann Multicast oder Unicast verwendet werden. Spezielle Coherence-Knoten, die als Proxy-Server konfiguriert sind, ermöglichen auch, dass Client-Knoten nur über TCP mit dem Cluster-Verbund kommunizieren. Das ist beispielsweise bei Remote-Clients notwendig, wenn diese in einem anderen Subnetz oder vor einer Firewall liegen.

Ein Coherence-Knoten kann Kapazität halten, also Objekte aktiv verwalten („Storage- Knoten"), oder selbst keine Kapazität innehaben und nur am Cluster partizipieren („Client-Knoten", definiert durch die Java-Systemeigenschaft „tangosol.coherence.distributed.localstorage=false"). Die letztere Form findet häufig beim Application-Server-Cluster Anwendung, dessen Instanzen dann als Coherence-Clients konfiguriert sind.

Neben Java-Knoten können auch .Net- oder „C++"-basierte Clients mit einem Coherence-Cluster interagieren. Dazu gibt es entsprechende APIs. Um überhaupt Anwendungsobjekte auch in .NET und C++ verarbeiten zu können, ist ein sprachübergreifender Serialisierungsmechanismus erforderlich. Coherence stellt diesen Mechanismus in Form des Portable-Object-Formats (POF) bereit. Verglichen zum Standard-Java-Serialisierungsmechanismus ermöglicht POF neben der Portierbarkeit eine effizientere Art der Serialisierung bezüglich der Größe der zu serialisierenden Objekte. Der Vollständigkeit halber sei erwähnt, dass auch ein Zugriff über REST-Dienste möglich ist.

Ein Coherence-Cluster verwaltet eine Menge von Caches und stellt Services für deren Verwaltung und für die verteilte Verarbeitung bereit. Das Anwendungsobjekt wird zusammen mit einem Schlüssel als (Schlüssel/Wert)-Paar in einem Cache abgelegt. Voraussetzung dafür ist, dass sowohl der Schlüssel als auch das Anwendungsobjekt (im Allgemeinen ein Teil eines Objektgraphen) serialisierbar sind.

Die Organisation der Objekte in einem über mehrere Knoten in Coherence aufgespannten Cache ist durch die Cache-Topologie bestimmt (siehe auch [4]). Eine der Basis-Topologien ist die sogenannte „Distributed-Cache-Topologie", auch „Partitioned Cache" genannt. Sie zeichnet sich dadurch aus, dass die serialisierten Objekte in Partitionen verwaltet werden, die auf verschiedene Coherence-Storage-Knoten transparent verteilt sind. Jeder Coherence-Knoten behält dabei dieselbe logische Sicht und den Zugriff auf alle im Cache stehenden Einträge (siehe Abbildung 1).

Für Fehlertoleranz sorgen Backup-Einträge, die entweder synchron oder asynchron auf Basis des Primärobjekts angelegt sind. Coherence versucht dabei, die Backups aufgrund von angegebenen Konfigurationseigenschaften abhängig von der JVM-Zugehörigkeit zu einer Maschine, einem Rack (mehrere Maschinen) oder einer Lokation (mehrere Racks) so zu verteilen, dass der höchste Robustheitsgrad erreicht wird [6].

In der Praxis ist oft der Ausfall einer Maschine tolerierbar. Das bedeutet, es wird zumindest die Eigenschaft „tangosol.coherence.machine" definiert und eine weitere zusätzliche Maschine von Anfang an in der Größenplanung berücksichtigt. Fehlertole-

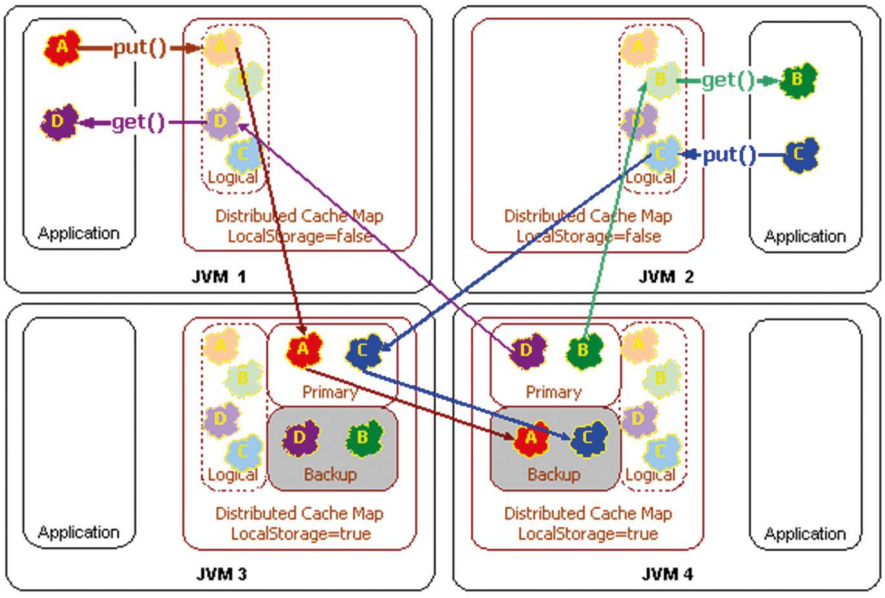


Abbildung 1: Coherence-Knoten, entnommen aus [5]

ranz erfordert jedoch Redundanz auf allen Ebenen. So muss auch die gesamte Hardware einschließlich Netzwerk-Komponenten redundant gemäß der gewünschten Fehlertoleranz ausgelegt sein.

Weitere Kapazität für partitionierte Caches im Cluster wird einfach durch Starten weiterer Storage-Knoten erreicht. Das Rebalancieren des Inhalts übernimmt Coherence im Hintergrund. Gleiches gilt bei geplantem oder ungeplantem Ausfall einer oder mehrerer JVMs.

Ein Blick auf das Java-API

Neben dem Schlüssel-basierten Zugriff auf einen Cache sind unter anderem auch paralleles Verarbeiten, Abfragen über Filter (ähnlich zu einer SQL-Where-Klausel), Berechnungen wie Aggregationen sowie die Nutzung von Events, Triggern und eigenen Eviction Policies möglich. Das zentrale Interface für den Zugriff auf einen Cache ist „com.tangosol.net.NamedCache" [7]. Es ist Subinterface für folgende wichtige Interfaces:

- *java.util.Map*
- *com.tangosol.util.QueryMap*
  Diese spezielle Map erlaubt das Abfragen eines Cache auf Basis von Filtern. Dazu liegt eine ganze Reihe vorbereiteter Filter („com.tangosol.util.filter.*") vor, es können aber auch eigene Filter genutzt werden. Mittels „ValueExtractoren" ist es möglich, Indizes zu erzeugen, um Abfragen zu optimieren. Ähnlich wie in einer Datenbank kann man sich den Ausführungsplan eine Abfrage anzeigen lassen und mithilfe der Indizes Einfluss auf die Abfrage nehmen.
- *com.tangosol.util.InvocableMap*
  Dieses Interface ermöglicht die Delegation von Ausführungscode auf Storage-Knoten, beispielsweise durch sogenannte „EntryProcessoren" oder „EntryAggregatoren". Das bedeutet, dass zu verändernde Daten nicht zum aufrufenden Code gebracht werden, sondern die Verarbeitungslogik an die Storage-Knoten weitergereicht und dort lokal angewendet wird, wo die Daten liegen. Das ermöglicht eine Parallelisierung der Verarbeitung über die verschiedenen Storage-Knoten hinweg.

  Ergebnisse von Aggregationen werden auf dem Knoten, der die Aggregation initiiert, zusammengeführt. Welche der Cache-Einträge dabei verarbeitet werden, lässt sich unter anderem über Filter definieren. Auch hier stehen für den Entwickler verschiedene Prozessoren und Aggregatoren bereit („com.tangosol.util.processor.*", „com.tangosol.util.aggregator.*"). Man kann jedoch auch eigene nutzen.
- *com.tangosol.util.ObservableMap*
  Dieses Interface erlaubt die Registrierung eines „MapListeners", um clientseitige Benachrichtigungen („MapEvents") nach Veränderungen im Cache beziehungsweise an Cache-Einträgen zu empfangen. Im Gegensatz dazu werden „MapTrigger" vor einer Veränderung ausgeführt.
- *com.tangosol.util.ConcurrentMap*
  Dieses Interface ermöglicht ein explizites Locking von Cache-Einträgen bei der konkurrierenden Verarbeitung. Aufgrund des Aufwands bei der Lock-Verwaltung sind „EntryProcessoren" einem explizitem Locking vorzuziehen.

Ein sehr umfangreiches, öffentlich zur Verfügung stehendes Tutorial [8] bietet die Möglichkeit, sich anhand von Beispielen mit Coherence und speziell mit dem API vertraut zu machen.

Die Benutzung eines Coherence-Cache erfolgt meistens nicht transparent, ein Anwendungsentwickler muss das API aus der Anwendungslogik heraus nutzen. Ebenso ist während der Modellierung der Domänenobjekte zu berücksichtigen, welcher Serialisierungsmechanismus benutzt wird (die Benutzung von POF erfordert das Implementieren von Serialisierungslogik, etwa durch Implementierung des Interface „PortableObject". Entscheidet sich der Entwickler für diesen zusätzlichen Aufwand, bekommt er aber auch vollständige Kontrolle über die Serialisierung). Zudem muss geklärt werden, für welche Objekttypen ein eigener Cache vorgesehen wird und ob Datenaffinität bei Ablage von Eltern- und Kind-Objekten in verschiedenen Caches durch Kollokation in derselben Partition sinnvoll ist [9].

Hat man die zusätzliche Anforderung, dass Daten einer persistenten Quelle (etwa eine Datenbank) mit Oracle Coherence verwendet werden sollen, muss der Anwendungsentwickler einen „CacheLoader" beziehungsweise „CacheStore" implementieren. Hier sind verschiedene Möglichkeiten des Zugriffs und des Schreibens möglich: „Read-Through" (Nachladen bei Cache-Miss im Falle eines schlüsselbasierten Zugriffs mittels „get(myKey)), „Write-Through" (synchrones Schreiben) und „Write-Behind" (asynchrones Schreiben).

Die Erweiterbarkeit der Coherence-Funktionalität ist nicht nur auf das Schreiben eigener Filter und „EntryProcessoren" begrenzt, es gibt darüber hinaus noch eine ganze Reihe von Erweiterungspunkten. Dies umfasst beispielsweise die Implementierung eigener „Eviction"-Policies, Indizes, „BackingMapListener" (serverseitige Listener), „Live Event"-Interceptoren bis hin zur Implementierung eigener „BackingMaps" oder die Erweiterung der Konfiguration durch sogenannte „Custom Namespaces" (Beispiel siehe [10]).

Um bei gewissen Fragestellungen das Rad nicht neu erfinden zu müssen, kann man auf die Open-Source-Projekte der Coherence-Community zurückgreifen [11]. Dazu gehören zum Beispiel der Coherence-Incubator [12] und das Oracle-Tools-Projekt [13]. Beim Coherence Incubator handelt es sich um zahlreiche Code-Beispiele, Utilities und Implementierungen von im Zusammenhang mit verteiltem Rechnen interessanten Software-Pattern. Zudem ist ein Plug-in für JVisualVM bereitgestellt. Das Oracle-Tool-Projekt liefert unter anderem Werkzeuge zum Testen in verteilten Coherence-Umgebungen.

Integration mit WebLogic Server

Grundsätzlich kann Coherence mit allen Java-Application-Servern arbeiten, jedoch gibt es eine sehr enge Integration mit Oracle WebLogic Server, insbesondere ab der Version 12.1.2. Das zeigt schon die Installation, denn bei der WebLogic-Server-Installation ist Coherence dabei. Abbildung 2 zeigt eine typische Topologie, die den Einsatz von Coherence und eines Application-Servers wie Oracle WebLogic Server darstellt. Die Application-Server-Knoten halten hier selbst keine Datenkapazität, Anwendungen können aber als Coherence-Clients Zugriff auf alle Daten haben. Diese Konfiguration ist eine Best-Practice und hat den Vorteil, dass beide Schichten getrennt wachsen und schrumpfen können. Außerdem vermeidet man hiermit negative Auswirkungen auf

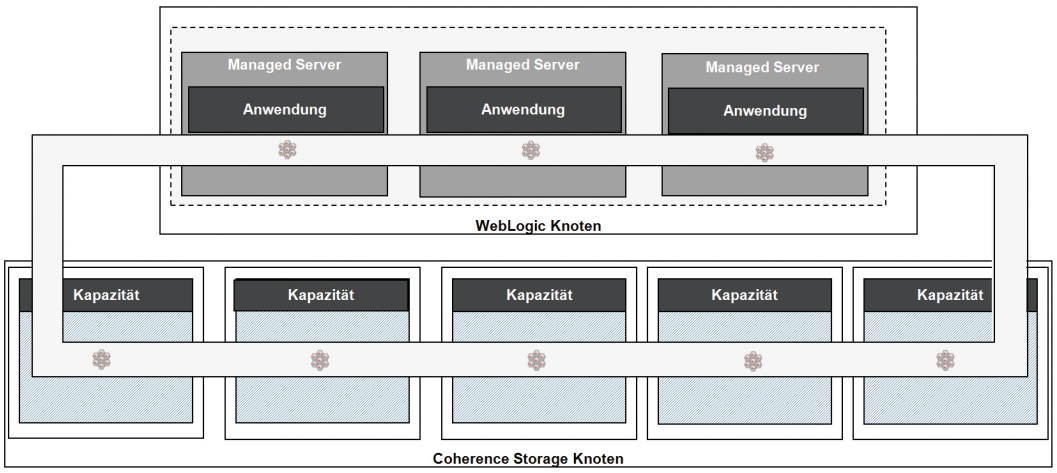


Abbildung 2: Einsatz von Coherence und WebLogic Server

die Application-Server-Knoten durch eine potenziell zu groß gewählte Heap-Size, verursacht durch Datenkapazität, die von diesen JVMs gehalten werden würde.

Die Konfiguration und Administration von Coherence-Knoten kann bei einer großen Anzahl von JVMs, die eher die Regel als die Ausnahme darstellt, sehr aufwändig sein. Hier liegt es nahe, die Coherence-Knoten in das Konfigurations- und Verwaltungsmodell von WebLogic aufzunehmen. Die Coherence-Knoten sind ab sofort vollständig in die WebLogic-Domänen-Architektur integriert, sie können also in gleicher Weise zu den herkömmlichen Managed Servern in einer Domäne konfiguriert und administriert werden. Das kann zum Beispiel per WebLogic-Administrations-Konsole erfolgen. Es ist möglich, dedizierte WebLogic-Knoten als Coherence Managed Server zu konfigurieren, indem man diesen Managed Server einem zuvor erstellten Coherence-Cluster zuordnet (siehe Abbildung 3).

Web-Logic-Scripting-Tool als Alternative

Gewisse clusterweite Konfigurationseigenschaften (siehe Abbildung 4) werden dann den zugehörigen Managed-Server-Knoten vererbt; knotenspezifische Coherence-Eigenschaften können angepasst werden. Als Alternative zur manuellen, GUI-basierten Konfiguration steht das WebLogic-Scripting-Tool (WLST) zur Verfügung. Zusätzlich ist es möglich, die Coherence Managed Server über den Node-Manager zu verwalten und zu überwachen. Diese Harmonisierung ist für den Betrieb sehr hilfreich. Doch auch für Entwickler sind diese Konfigurationsvereinfachungen von großem Vorteil, denn Coherence-basierte Anwendungen sollten schon sehr früh auf Mehrknoten-Umgebungen getestet werden, um das Verhalten in verteilten Umgebungen

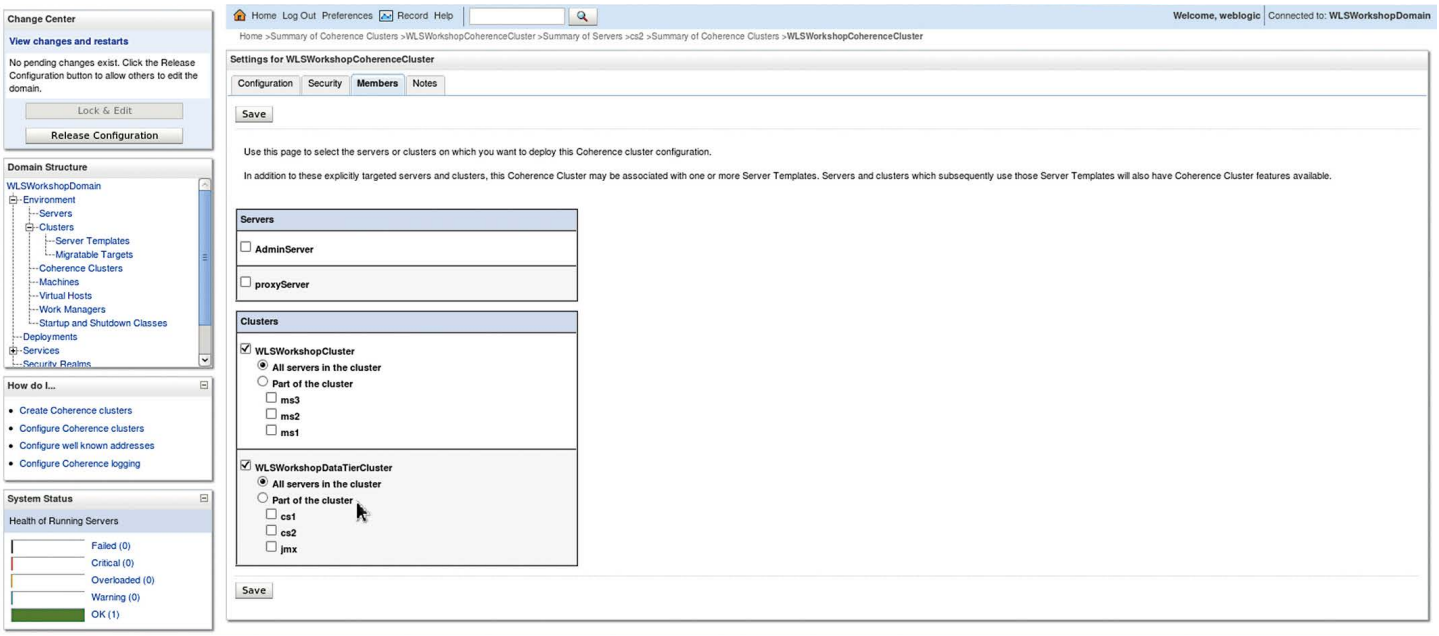


Abbildung 3: WebLogic-Knoten als Coherence Managed Server konfigurieren

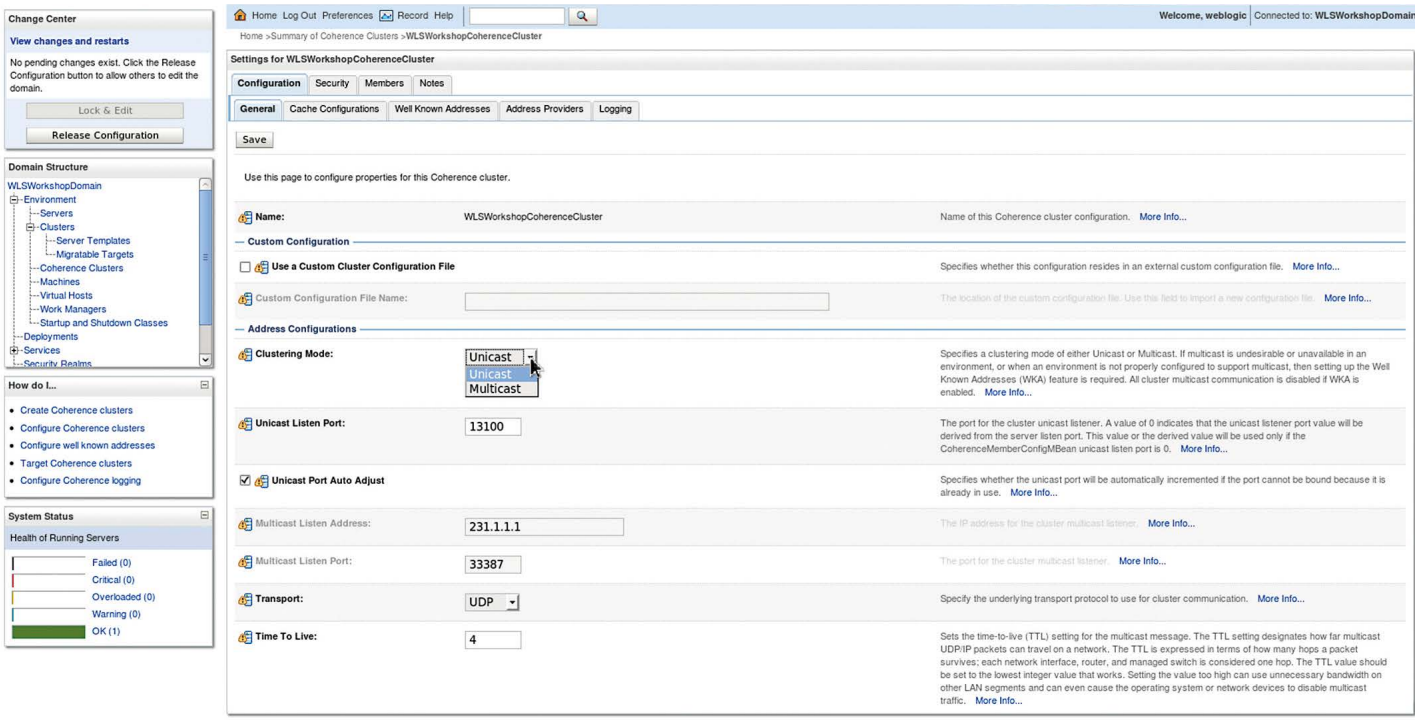


Abbildung 4: Vererbung clusterweiter Konfigurationseigenschaften

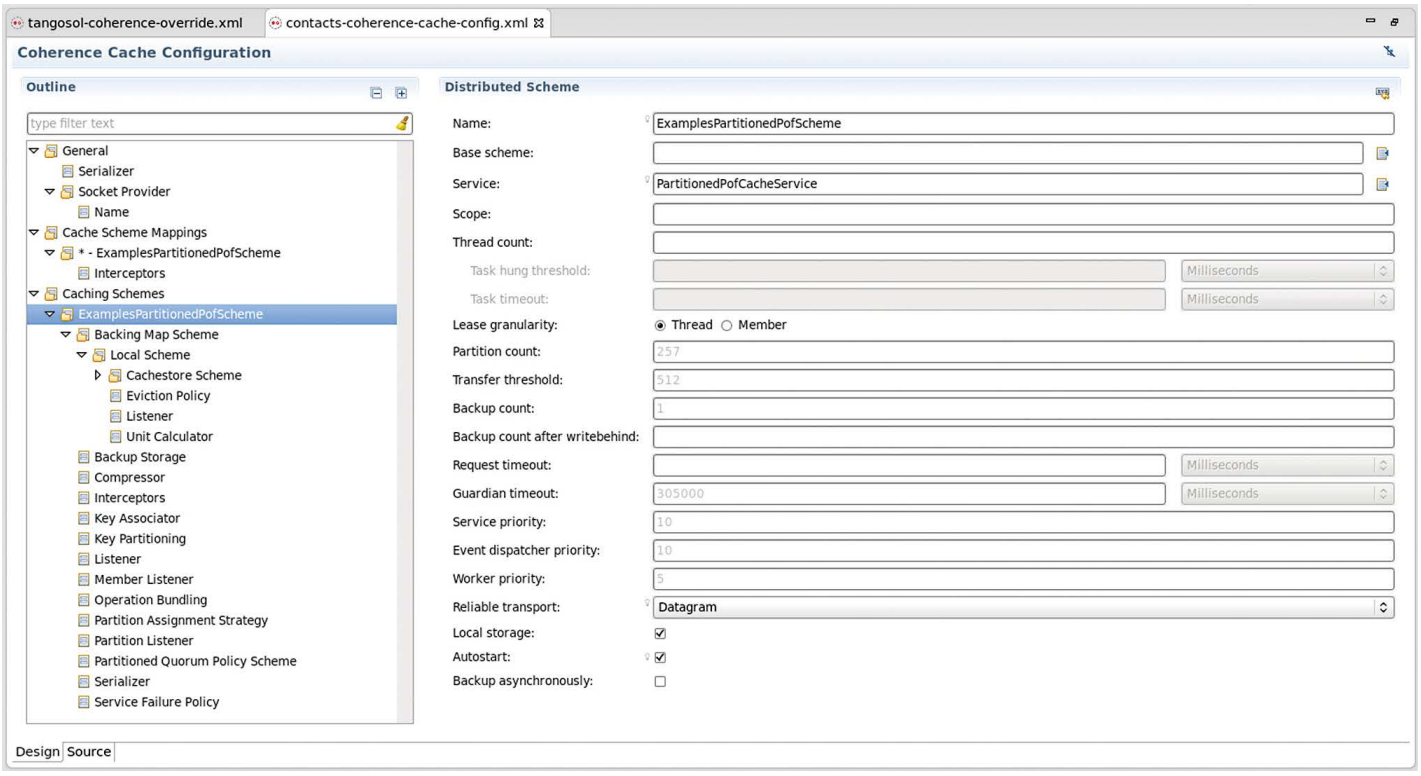


Abbildung 5: Editieren der XML-Dateien

bewerten zu können. Einen hohen Automatisierungsgrad, der für DevOps unumgänglich ist, ermöglicht der Einsatz von Maven in Zusammenarbeit mit den WebLogic und Coherence Maven Goals.

Für die Paketierung von Coherence-Anwendungen steht eine neuer Archiv-Typ namens „Grid Archive“ (GAR) zur Verfügung. Dieser hat einen archivspezifischen Verzeichnisaufbau mit eigenem Deployment Deskriptor namens „coherence-application.xml“. In der Paketierung sind neben den benötigten Java-Klassen ebenfalls Konfigurationsdateien für POF und Caches zu finden. Eine GAR-Paketierung kann auf

den Knoten die Kapazität halten, zum Laufen gebracht oder innerhalb von anderen Paketierungen wie einem EAR für das Deployment auf WebLogic untergebracht werden.

Tool-Unterstützung

Das „Oracle Enterprise Pack for Eclipse" (OEPE) bietet seit geraumer Zeit eine sehr gute Tool-Unterstützung für Coherence an. Nachfolgend wird die Version 12.1.2.1 betrachtet [14]. Um Coherence in einem Eclipse-Projekt zu nutzen, muss man die Coherence-Facette unter Angabe der Coherence-Bibliothek auswählen. Ist dies geschehen, kann man sich die XML-Konfigurationsfiles (etwa für die Coherence-Operational-Override-Konfiguration oder die Cache-Topologie-Konfiguration) Wizard-basiert erzeugen lassen.

Alternativ dazu besteht die Möglichkeit, ein Projekt vom Typ „Oracle Coherence Application" anzulegen. Das letztere erzeugt eine Verzeichnisstruktur inklusive „META-INF/coherence-application.xml", um die Projektdateien in ein GAR zu exportieren. Im Project-Wizard muss die zu verwendende Coherence-Bibliothek ausgewählt werden und es besteht die Wahl, ein POF-Konfigurationsfile oder ein Cache-Topologie-Konfigurationsfile im Projekt zur Verfügung zu stellen. Sowohl für das Cache-Topologie-Konfigurationsfile als auch das Coherence-Operational-Override-Konfigurationsfile steht ein grafischer Editor bereit, um ein manuelles Editieren der XML-Dateien zu vermeiden (siehe Abbildung 5).

Um Coherence-Knoten direkt in der IDE mit gewünschten Konfigurationseigenschaften zu starten, stehen vordefinierte Konfigurationen bereit (siehe Abbildung 6). An dieser Stelle lassen sich alle für die in einer Entwicklungsumgebung erforderlichen Konfigurationseigenschaften bequem und übersichtlich setzen.

Fazit

Ein In Memory Data Grid kann eine geeignetere Alternative zu anderen Datenhaltungsmechanismen für von verschiedenen JVMs gemeinsam genutzte Daten sein. Es löst grundsätzliche Probleme hinsichtlich verteiltem Zugriff und Kapazität.

Oracle Coherence ist eine sehr reife und in der Praxis bewährte Implementierung eines Java-basierten In Memory Data Grid. Viele der sehr umfassenden Funktionalitäten und auch Alleinstellungsmerkmale konnten an dieser Stelle nicht aufgeführt werden. Dazu zählen der Umgang mit „Stale"-Caches (Oracle Coherence Golden Gate HotCache), Continuous-Query-Caching, Elastic Data (elastisches Auslagern von Heap auf externe Medien in sehr großen Clustern), TopLink Grid (Shared L2-Cache mit Coherence und Nutzung von Coherence mit dem JPA-Programmiermodell), Coherence*Web (ersetzt den Standard-Http-Session-Verwaltungsmechanismus von Application Servern durch Nutzung von Coherence) etc. Deshalb die Empfehlung der Autoren: Einfach mal ausprobieren.

Referenzen und Links

[1] http://www.jroller.com/cpurdy/entry/defining_a_data_grid
[2] https://jcp.org/en/jsr/detail?id=107

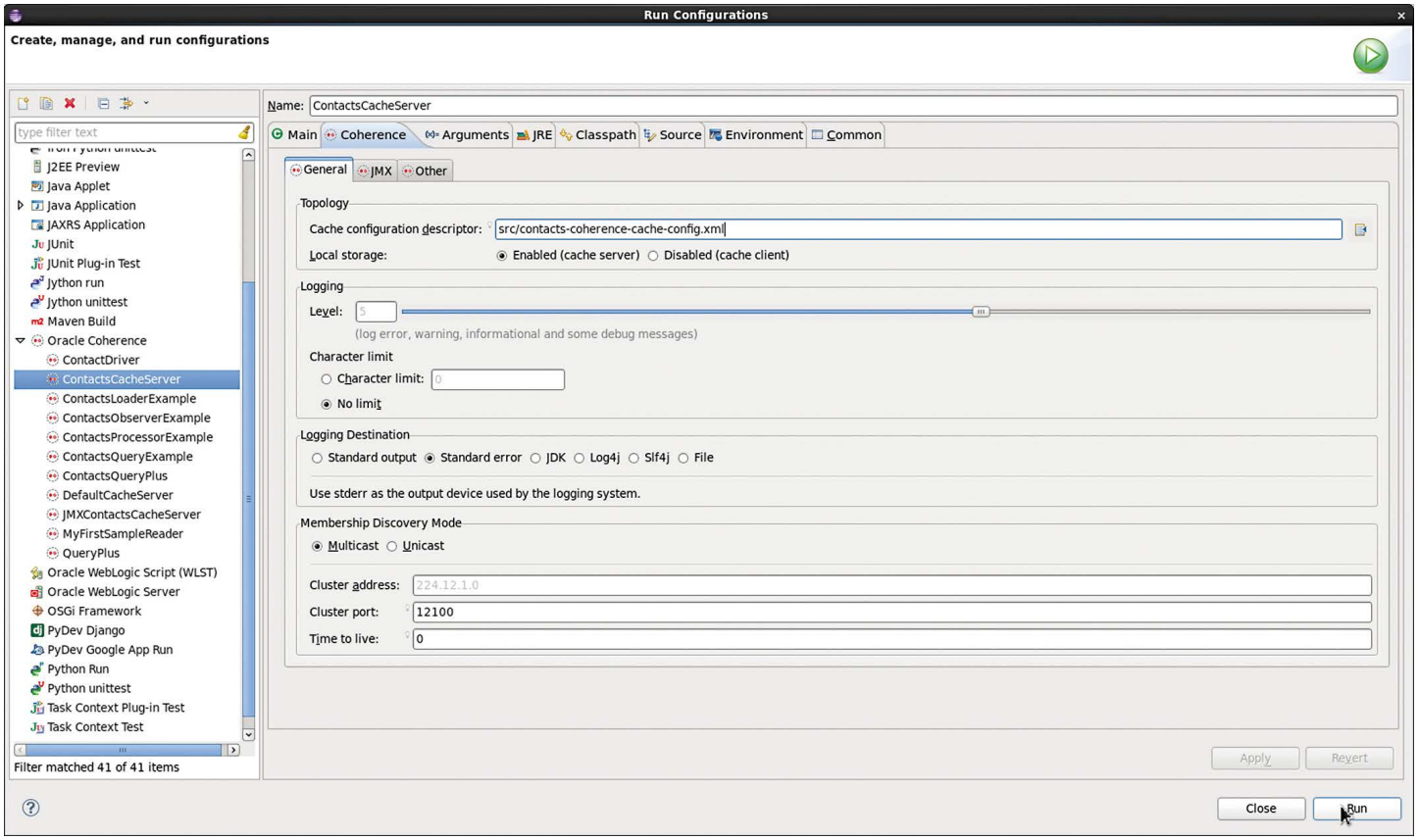


Abbildung 6: Vordefinierte Konfiguration

[3] https://jcp.org/en/jsr/detail?id=347
[4] Coherence-Cache-Topologien: http://docs.oracle.com/middleware/1212/coherence/COHDG/cache_intro.htm#COHDG5049
[5] Generelle Oracle-Coherence-12.1.2-Dokumentation: http://docs.oracle.com/middleware/1212/coherence/index.html
[6] http://coherencedownunder.wordpress.com/2013/08/29/site-and-rack-safety-in-coherence-12-1-2
[7] http://docs.oracle.com/middleware/1212/coherence/COHJR/toc.htm
[8] http://docs.oracle.com/middleware/1212/coherence/COHTU/index.html
[9] siehe Abschnitt Domain Model, Relationship Patterns in http://coherence.oracle.com/display/COH35UG/Manage+an+Object+Model
[10] "Add Functionality To Oracle Coherence Services": http://thegridman.com/coherence/extending-oracle-coherence-services/
[11] https://github.com/coherence-community
[12] https://java.net/projects/cohinc/ und https://github.com/coherence-community/coherence-incubator
[13] https://java.net/projects/oracletools
[14] http://www.oracle.com/technetwork/developer-tools/eclipse/overview/index.html

Michael Bräuer
michael.braeuer@oracle.com

Michael Bräuer ist leitender Systemberater der ORACLE Deutschland B.V. & Co. KG. Er nutzt seine langjährige Berufserfahrung in den Bereichen Anwendungsintegration, Middleware und Java EE, um kritische Geschäftsanwendungen auf das passende technologische Fundament zu stellen.

Peter Doschkinow
peter.doschkinow@oracle.com

Peter Doschkinow arbeitet als Senior Java Architekt bei Oracle Deutschland. Er beschäftigt sich mit serverseitigen Java-Technologien und Frameworks, Web Services und Business Integration, die er in verschiedenen Kundenprojekten erfolgreich eingesetzt hat. Vor seiner Tätigkeit bei Oracle hat er wertvolle Erfahrungen als Java Architect and Consultant bei Sun Microsystems gesammelt.

JavaFX™ – das neue Gesicht für Java-Anwendungen

Frank Pientka und Hendrik Ebbers, Materna GmbH

Bisher konnte Java nur wenig mit ansprechenden Oberflächen glänzen. Mit JavaFX, das Bestandteil der aktuellen Java-Version ist, wird eine große Anzahl von Plattformen unterstützt, sodass sich Java-Anwendungen nicht mehr verstecken müssen, sondern vom Desktop über Smartphone bis zu Embedded-Geräten viele neue Einsatzgebiete erschließen und eine gute User-Experience ermöglichen.

Obwohl Java von Anfang an mit Oberflächen-Funktionen wie AWT und Swing ausgerüstet ist, konnten sich Java-Oberflächen auf dem Desktop nicht durchsetzen. Hinzu kommt, dass die Java-Oberflächen-Bibliotheken von Oracle selbst nicht mehr weiterentwickelt werden und nur über externe Zusatzbibliotheken zeitgemäße Oberflächen möglich sind. Etwas anders sieht es im Bereich der Web-Oberflächen aus, bei dem Java zumindest auf dem Server sehr oft eingesetzt wird und auch eine große Anzahl von Web-Frameworks existiert. Zusätzlich stellen sich durch das immer stärker werdende Aufkommen von „Mobile & Embedded"-Devices ganz neue Anforderungen an die Gestaltung und Bedienung von Anwendungen.

Da Java gerade für diesen Bereich ursprünglich entworfen wurde, ist zu begrüßen, dass Oracle mit JavaFX eine neue Plattform für moderne Rich-Client-Oberflächen zur Verfügung stellt. Da Oracle JavaFX in der aktuellen JDK-8-Version mit ausliefert,

wird es einfacher, Rich-Client-Anwendungen zu entwickeln und diese auch für viele Plattformen zur Verfügung zu stellen. Mit dem JavaFX-Packager kann die Ablaufumgebung mit ausgeliefert werden, sodass der Anwender gar nicht merkt, dass es sich um eine Java-Anwendung handelt. Selbst so kostengünstige Umgebungen wie der Einplatinen-Computer Raspberry Pi sind unterstützt.

Warten auf die große Acht

Die Versuche von Sun mit JavaFX Script waren wenig erfolgversprechend. Doch seit JavaFX Bestandteil der Java-6-Ablaufumgebung ist und sich mit den Nachfolgeversionen einiges getan hat, besteht Hoffnung, dass sich Java wieder stärker bei Client-Oberflächen, vor allem im Embedded-Bereich, platzieren kann. Als Endgeräte kommen hier beispielsweise Smartphones, Kiosksysteme, Set-Top-Boxen, Einplatinen-Computer, Desktop-Computer oder Blu-ray-Disc-Abspielgeräte in Frage (siehe Abbildung 1).

Da eine Migration bestehender Oberflächen eher schwierig ist, bietet es sich mehr für Neuprojekte an. Es gibt jedoch mit SwingNode einen Migrationspfad, um Swing-Anwendungen schrittweise auf JavaFX umzustellen. Da das mittlerweile veraltete Swing-Toolkit auch von Oracle nicht weiterentwickelt wird, ist die Migration vor allem für weiterhin langläufige Projekte aber ein interessanter und gangbarer Ansatz. Vor allem, wenn man auf dem Markt weiterhin mit modernen Tools konkurrieren möchte, sollte man auf einige Features von JavaFX nicht verzichten.

Mobile Anwendungen können über den WebView-Node Anwendungen mit Web-Technologien entwickeln und in JavaFX-Anwendungen integriert werden. Dabei ist es von Vorteil, dass JavaFX „Cascading Style Sheets“ (CSS) zur grafischen Gestaltung verwendet und für die Web-Darstellung die WebKit Rendering Engine „PRISM“ einsetzt. Bei Bedarf können bei diesem hybriden Ansatz trotzdem die leistungsfähigeren nativen Funktionen genutzt werden. Durch diesen Technikansatz ist es beispielsweise problemlos möglich, interaktive Google-Maps-Darstellungen in Desktop-Anwendungen zu integrieren.

Neben der WebView gibt es noch ein paar weitere Komponenten, die über die allgemein üblichen Darstellungsformen (Tabellen, Buttons etc.) hinausgehen: Mit der MediaView lassen sich verschiedenste Medienformate wie Videos oder Sound-Files nahtlos in eine JavaFX-Anwendung integrieren. Zusätzlich bietet JavaFX mehrere Komponenten zur Darstellung von Business-Diagrammen an. Diese können leicht durch die angebotenen Funktionen angepasst und sogar über CSS gestaltet werden.

Mit Java 8 wird JavaFX auch eine volle 3D-Unterstützung bieten. Hierdurch können Daten in der dritten Dimension visualisiert sowie Lagepläne, komplexe Diagramme und Ähnliches deutlich besser dargestellt werden. Hierbei unterstützt JavaFX nicht nur 3D-Modelle, sondern auch dynamische Lichtquellen und verschiedene Textur-Formate etc., wie sie auch in modernen 3D-Anwendungen vorkommen (siehe Abbildung 2).

Abbildung 2: Darstellung von 3D-Modellen

Den endgültigen Einzug in die Java-Welt wird JavaFX mit Java 8 erreichen. Mit dieser Version ist es komplett im Standard von Java integriert und alle relevanten Bestandteile für „Business & Multimedia“-Anwendungen sind vorhanden und erprobt. Interessierte Entwickler können sich das Ganze aber bereits heute über die Preview Version von Java 8 anschauen. Das Final ist Anfang 2014 erschienen.

JavaFX goes Enterprise

Der JavaFX-Nutzung in „Business- & Enterprise“-Anwendungen steht somit nichts mehr im Wege. Durch die starke Community und Entwicklung in diesem Bereich gewinnen sowohl die Entwickler als auch die Endkunden. Eine sehr gute Werkzeug-Unterstützung sowie moderne Features erlauben es den Entwicklern, deutlich einfacher intuitive und ansehnliche Oberflächen zu erstellen. Spezielle UI- und Workflow-Wünsche des Kunden können so erheblich einfacher und besser umgesetzt werden. Nachfolgend sind einige dieser Features vorgestellt, die sowohl für Entwickler als auch für Benutzer einen beträchtlichen Mehrwert bringen.

Das GUI-Komponenten-Toolkit von JavaFX ist so modular aufgebaut, dass es jederzeit einfach um neue Komponenten oder neue Erscheinungsformen vorhandener Komponenten erweitert werden kann. Beispiele hierfür lassen sich unter anderem in der Enzo-Bibliothek (siehe https://github.

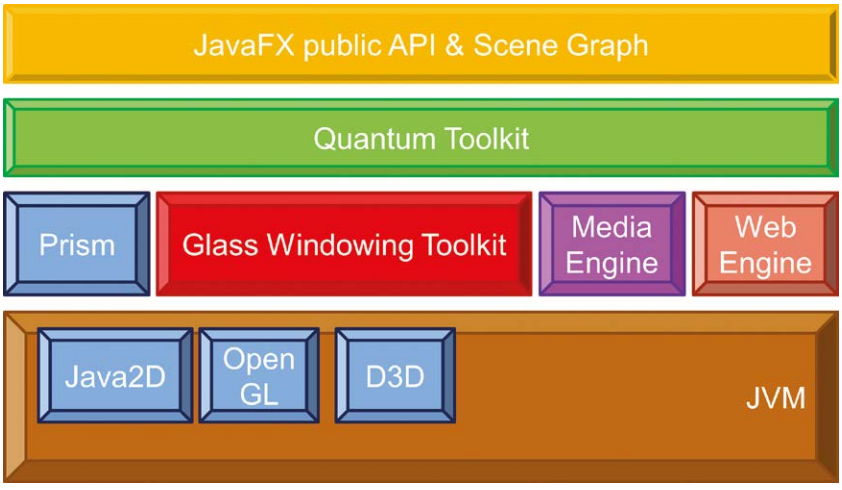


Abbildung 1: Die JavaFX-Architektur

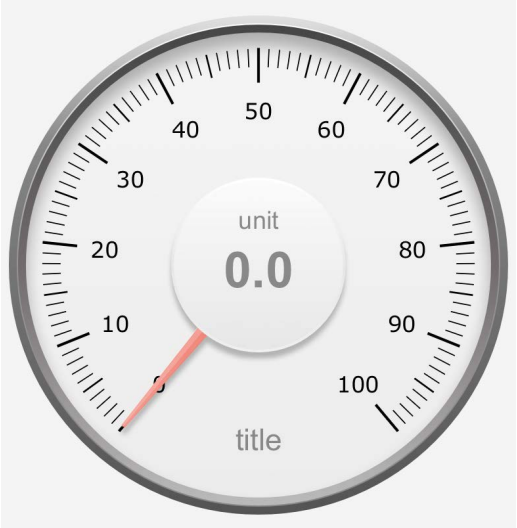


Abbildung 3: Beispiel-Tool aus der Enzo-Bibliothek

com/HanSolo/Enzo) finden, die JavaFX um mehrere grafische Komponenten zur Visualisierung von Daten erweitert. Als Vorlage für die Komponenten dienen hier verschiedenste Messgeräte (siehe Abbildung 3).

Ein weiteres Beispiel ist AquaFX (siehe www.aquafx-project.com), das ein auf Apple Mac OS basierendes Design für alle Standard-Komponenten von JavaFX bereitstellt. Durch Nutzung dieser Bibliothek kann man eine JavaFX-Anwendung nicht mehr von einer nativen Mac-Anwendung unterscheiden. Eine noch bessere Integration in das Betriebssystem des Kunden kann durch Nutzung nativer Installer erreicht werden. Durch ein Build-Tool von Oracle kann jedes JavaFX-Programm als native Anwendung ausgeliefert werden. Unter Windows bekommt der Kunde so zum Beispiel einen eigenen Installer geliefert, der mit Wizard-Dialogen durch die Installation der Anwendung leitet. Dabei kann man keinen Unterschied mehr zwischen der Java- und einer reinen Windows-Anwendung (Office etc.) feststellen. Ein großer Vorteil ist, dass die unterlagerte Java-Umgebung fester Bestandteil der nativen Anwendung ist und eine vorherige Java-Installation beziehungsweise eine nachgelagerte Administration komplett wegfällt. Die ersten JavaFX-Programme sind sogar bereits im Mac AppStore von Apple zu finden (siehe Abbildung 4). Dieser Spezialfall funktioniert momentan zwar noch über Umwege, wird aber mit hoher Wahrscheinlichkeit zum Erscheinen von Java 8 noch deutlich verbessert und ausgebaut.

JavaFX ist aber nicht nur auf klassischen Desktop-Rechnern, sondern auch noch auf vielen anderen Plattformen lauffähig. Durch die Unterstützung von Embedded Devices und den ARM-Chipsatz kann JavaFX auf kleinster Hardware ausgeführt werden und bietet so beispielsweise eine sehr elegante Lösung bei Kiosksystemen und Home Automation. Vor allem in diesen Bereichen war man bisher gezwungen, auf elegante oder hochauflösende grafische Darstellungen zu verzichten oder direkt auf ein PC-System, etwa in Form eines kleinen Cubes, zurückzugreifen. Durch die in den letzten Jahren extrem gestiegene Performance von ARM-Prozessoren und die neue Möglichkeit, JavaFX als Oberflächensprache auf diesen laufen zu lassen, ergeben sich jedoch heute völlig neue Möglichkeiten. Allein die Kosten- und Stromersparnis im Bereich der Hardware ist enorm.

Auch für moderne Arbeitsgeräte wie Windows Surface mit Windows RT ist eine komplette Integration vorhanden. Gerade auf diesen Devices zeigen sich weitere Stärken wie die MultiTouch-Fähigkeit von JavaFX. Zoom- und Swipe-Gesten, wie man sie sonst nur von Geräten wie dem iPad kennt, werden von JavaFX voll unterstützt und sind programmatisch leicht umzusetzen. Hierdurch ist ein Einzug dieser State-of-the-Art-Interaktionen in moderne Business-Anwendungen endlich möglich. Wer hat sich schließlich nicht schon oft

Abbildung 4: Native Mac-Installation Look&Feel

gewünscht, durch große Diagramme, Grafiken oder Lagerpläne einfach mithilfe von einfachen Touch-Gesten zu navigieren.

Aber auch das bereits angesprochene iPad bleibt nicht unbeachtet. Zwar sind sowohl iOS als auch Android bisher noch keine offiziell unterstützten Plattformen für JavaFX, aber das Ganze wird sicherlich nicht mehr lange auf sich warten lassen. So ist es etwa erst vor Kurzem gelungen, mithilfe der Open-Source-Lösung RoboVM JavaFX-Anwendungen auf einem iPad zu bereitzustellen (siehe http://blog.robovm.org/2013/05/JavaFX-openjfx-on-ios-using-robovm.html).

Neben den beschriebenen UI-Gestaltungen und -Interaktionen bietet das JavaFX-Ökosystem aber noch eine Fülle anderer wichtiger Features und Tools (siehe Abbildung 5). Durch die konsequente Nutzung von CSS und die hierdurch mögliche Trennung von Entwickler- und Design-Teams wird der Entwicklungsprozess deutlich vereinfacht. Als Schnittmenge zwischen

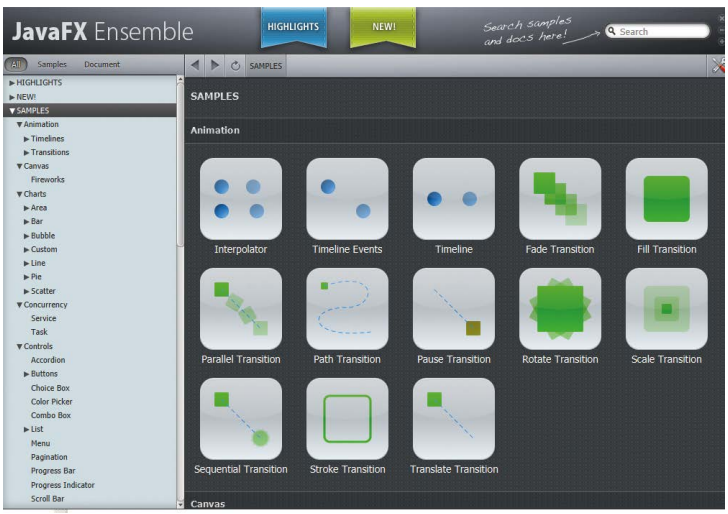


Abbildung 5: Masken für Kiosksysteme

diesen beiden Instanzen kann der Scene Builder genutzt werden, mit dem sich Dialog-Layouts so einfach wie in einem modernen Publisher-Programm erstellen lassen. Während die Designer diese Layouts ausrichten und gestalten, kann das gesamte Entwickler-Team bereits eine An- und Einbindung dieser Dialoge in das Programm vornehmen. Für eine saubere und schnelle Anbindung an vorhandene Enterprise- & SOA-Systeme ist auch gesorgt. Projekte wie DataFX (siehe http://www.JavaFXdata.org/) und OpenDolphin (siehe http://open-dolphin.org) nehmen einem Entwickler hierbei einen Großteil der Arbeit bereits ab und erlauben eine Integration und Darstellung von WebServices-Daten.

Fazit

JavaFX bringt Java bei modernen Oberflächen wieder ins Spiel und verbindet dabei die Welten Desktop, Mobile, Embedded und Web auf eine einzigartige Weise. Aufgrund der breiten Plattformbasis und der State-of-the-Art-UI-Features von JavaFX lohnt sich ab jetzt also wieder ein zweiter Blick für Kunden, die alte Oberflächen ablösen oder auf mobile Umgebungen umstellen wollen.

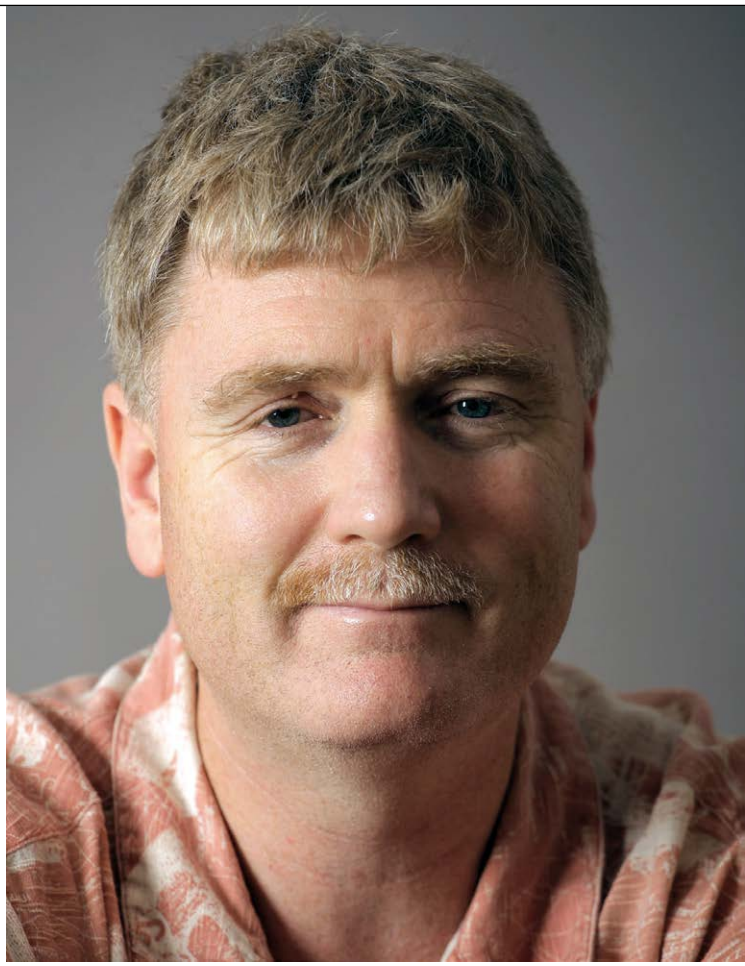
Frank Pientka
Frank.Pientka@materna.de

Hendrik Ebbers
hendrik.ebbers@materna.de

Frank Pientka und Hendrik Ebbers sind Software-Architekten bei Materna. Hendrik Ebbers arbeitet aktiv an mehreren OpenSource-JavaFX-Projekten mit und ist Leiter der Java User Group Dortmund.

Java Performance-Tuning

Wenn überhaupt, dann gibt es nur wenige Leute, die mehr über Java Performance-Tuning wissen als der Java-Champion Kirk Pepperdine, derzeit beschäftigt als Consultant für Kodewerk, einem Unternehmen, das Dienste, Trainings und Werkzeuge rund um das Thema „Performance" anbietet. Pepperdine ist hoch angesehen für seine Workshops und Artikel und hat bereits weltweit auf Konferenzen sein Wissen über Performance-Tuning weitergegeben. In einer Welt, in der sich Hardware und Software so schnell verändern, ist seine Meinung es wert, gehört zu werden. Ein Interview über die Bedeutung der Cloud, die steigende Rechenleistung von Mehrkern-Systemen und die Lösung von Performance-Problemen in Enterprise-Applikationen.



Java-Champion Kirk Pepperdine

Welche Tipps zum Java Performance-Tuning, die noch vor fünf Jahren galten, sind heute nicht aktuell?

Pepperdine: Die Liste ist lang. Die Technologie ändert sich ständig und mit dieser Änderung müssen wir neu evaluieren, was noch funktioniert und was nicht. Die Java Virtual Machine zum Beispiel durchlebt gerade eine Minirevolution bezogen auf das, was sie für ihre Anwender leisten kann. Es gibt einen neuen Garbage Collector, der gerade Einzug hält, und die anderen Collectoren werden immer öfter aussortiert. Die Adaptive Size Policy wurde für JDK 8 komplett neu geschrieben. Jede Einschätzung bezüglich Garbage Collection, die vor Java SE 8 gemacht wurde und Adaptive Sizing verwendet hat, wird sich ändern, weil sich die Implementierung geändert hat. Ein häufiger Rat ist, die minimale Heap-Größe auf den Wert der maximalen Heap-Größe zu setzen, manchmal ist das auch sinnvoll. Aber heutzutage sollte man das oft auch nicht tun, da es die adaptiven Fähigkeiten der JVM blockiert. Als die Technologie noch nicht so ausgereift war, hat das durchaus Sinn gemacht, da die JVM sich oft nicht optimal anpassen konnte. Heute ist sie viel besser darin sich anzupassen und man sollte dieser Optimierung nicht entgegenwirken. Wenn man explizit etwas tut, das gegen die Optimierungen der JVM-Entwickler arbeitet, dann ist das der eigenen Anwendung nicht zuträglich.

Welche Hardware-Änderungen sollten Entwickler berücksichtigen?

Pepperdine: Die größte Änderung sind sicherlich die Mehrkern-Prozessoren, was bedeutet, dass wir jetzt viel mehr Rechenleistung zur Verfügung haben als zuvor. Diese basiert nicht nur auf der erhöhten Taktfrequenz, sondern auf unserer Fähigkeit zu skalieren. Die größte Herausforderung in der Software-Entwicklung ist es, einen Weg zu finden, diese zusätzliche Rechenleistung zu nutzen. Sieht man sich einen Computer an, so hat man eine Kiste voll mit Dingen vor sich, die sich nicht gemeinsam nutzen lassen. Der Bus, der Speicherzugriff, die CPU, der Framebuffer, die Grafikkarte etc. lassen sich nicht gemeinsam nutzen. Das bedeutet, die Threads konkurrieren um die Nutzung dieser Ressourcen. Mit Mehrkern-Prozessoren können wir Programme schreiben, die anders mit geteilten Ressourcen umgehen. Also müssen wir anfangen, Hardware zu entwerfen oder sie anders zu konfigurieren, um sicherzustellen, dass wir auch die Ressourcen bereitstellen können, die unsere Applikation benötigt. Darüber hinaus enthüllen die Änderungen der Hardware Fehler in bestehenden Implementationen, die zumeist auf falsche Annahmen bezüglich des Zusammenspiels einzelner Komponenten zurückzuführen sind. Oft zeigen sich diese nur als schwer aufzuspürende Race Conditions innerhalb der Implementation.

Können Sie uns einige der größten Missverständnisse nennen, die Ihnen im Zusammenhang mit Java Performance begegnen?

Pepperdine: Viele verstehen die Just-in-Time-Technologie noch nicht sehr gut. Das ist ein semantisches Problem, da unser Bildungssystem Dinge wie die Omega-Notation hervorgebracht hat, mit der Anwender sich einen Algorithmus ansehen können und Annahmen über dessen Laufzeitverhalten machen können. In statischen

Umgebungen mag das auch funktionieren. Sobald man den Algorithmus jedoch in der JVM in einer JIT-Umgebung laufen lässt, ist alles möglich. JIT kann den Code soweit verändern, dass der resultierende und ausgeführte Assembler-Code zwar die gleiche Funktion erfüllt, aber überhaupt nicht mehr nach dem ursprünglichen Code aussieht.

Wenn Entwickler Code betrachten, machen sie eine statische Analyse, um den Ressourcen-Verbrauch abzuschätzen. Sie lassen jedoch außer Acht, was JIT mit dem entsprechenden Code anstellen wird. Das hat zur Folge, dass ihre auf Code-Analyse basierenden Laufzeit-Abschätzungen so ziemlich alle falsch sind.

Sie werden mit den Worten zitiert, dass es der größte Fehler beim Identifizieren von Performance-Problemen sei, nicht das System als Ganzes zu betrachten. Sie behaupten, dass wichtige Hinweise übersehen werden und manchmal nicht die richtigen Werkzeuge zum Einsatz kommen. Bitte erläutern Sie dieses Thema.

Pepperdine: Ich betrachte Software als Ressourcen-Management. Das bedeutet, man kann viele wirtschaftliche Grundsätze anwenden, die einem helfen zu verstehen, was vor sich geht. Die Wirtschaftlichkeit der Situation wird durch die Hardware geregelt, was das Ganze wirklich werden lässt. Hardware gibt reale Kapazitäten und Durchsatz vor, die nicht gemeinsam nutzbar sind. Man muss wissen, welche Ressourcen die eigene Anwendung nutzt und wie diese Ressourcen verwaltet werden.

Das gibt die besten Hinweise darauf, was die Applikation gerade macht und was man tun kann, um eine bessere Auslastung der zur Verfügung stehenden Ressourcen zu erzielen. Eine ausgewogene Auslastung der Ressourcen führt zu maximaler Performance. Werkzeuge helfen einem herauszufinden, womit der Computer gerade beschäftigt ist. Mit ein bisschen Glück verraten einem die richtigen Werkzeuge, welche der zur Verfügung stehenden Ressourcen von welchen Teilen der Applikation genutzt werden. Daraus lässt sich ableiten, was an der Applikation geändert werden muss, um die Hardware besser auszulasten und so die beste Performance für den Endanwender zu erreichen.

Außerdem ist es wichtig zu verstehen, dass die Hardware-Auslastung stark von dem abhängt, was die Applikation gerade macht, was wiederum davon abhängt, wie der Endanwender das Programm nutzt. Unter Umständen nutzt der Anwender das Programm in einer Art und Weise, die zur Verringerung der Last auf der JVM führt und dann entsprechend weniger Hardware-Ressourcen benötigt. Man muss also die Auslastungsprofile der Hardware berücksichtigen. Wenn die Nutzer die Art und Weise, wie sie die Software einsetzen, ändern, ändert sich auch die Ressourcen-Auslastung. Man muss also das gesamte System betrachten, um zu verstehen, was vor sich geht.

Was war das schwierigste Problem, das Ihnen beim Performance-Tuning bisher begegnet ist?

Pepperdine: Ich mag schwierige Probleme – sie machen Spaß. Die schwierigsten Probleme haben mit niedriger Latenz zu tun. Vor kurzem hatte ich mit einer Anwendung zu tun, bei der wir mit einer 60-Hertz-Taktung auskommen mussten. Mit anderen Worten: Die Aufgabe musste in 16,7 Millisekunden erledigt sein. Die Herausforderung war zu verstehen, was wir zu erreichen versuchten und wie schnell die Aufgabe bearbeitet werden kann. Im Fall von Java muss man wissen, wann die JVM die Garbage Collection auslösen wird, wie lange der Garbage-Collection-Zyklus dauern wird und ob man dadurch den geforderten Takt einhalten kann. Wir wollten nach den Bereichen im Code suchen, an denen Ausreißer in der Laufzeit auftraten und die einen Großteil der 16,7 Millisekunden ausmachten. Das war sehr herausfordernd. Wir versuchten verschiedene Technologien, um herauszufinden, welche uns die stabilsten Antwortzeiten liefern konnte. Wir waren in diesem Fall nicht an der durchschnittlichen, sondern an der primären Antwortzeit interessiert und wollten herausfinden, mit welchen Schwankungen wir zu rechnen hätten und wie diese sich auf die Verlängerung der Antwortzeiten auswirken.

Wie hat sich Cloud Computing auf das Performance Tuning ausgewirkt?

Pepperdine: Cloud Computing hat das Gesamtbild verändert. So lange man skalieren kann, sollte die Performance stabil gehalten werden können. Clouds bieten die Möglichkeit, in Richtung der virtualisierten Hardware zu skalieren, eine Möglichkeit, die viele in ihren Rechenzentren bisher nicht hatten. Es bedeutet aber auch, dass Monitoring wesentlich wichtiger geworden ist, denn das Monitoring muss dem Betreiber nicht nur sagen, was passiert, in der Cloud muss es außerdem möglich sein zu sagen, wann die Infrastruktur erweitert werden muss oder verkleinert werden kann. Mit anderen Worten, wenn man die Anpassungsfähigkeit der Cloud automatisieren möchte, benötigt man die entsprechenden Werkzeuge, die diese Aufgabe unter Berücksichtigung des Performance Managements erfüllen können.

Die Werkzeuge müssen also intelligenter werden und den Anwendern helfen zu verstehen, wann sie wie skalieren müssen. Das Gesamtbild hat sich also geändert. Bevor wir die Cloud hatten, musste im Ausnahmefall ein Team mobilisiert werden, um den Fehler zu lokalisieren, zu beheben und das System wiederherzustellen. Jetzt wird man ein neues virtuelles System hochfahren, in dem die Applikation läuft, und damit den fehlerhaftes Knoten einfach ersetzen. Wir möchten dem Endanwender eine gute, stabile und konsistente Performance bieten, sodass er nicht merkt, wie viel Chaos hinter den Kulissen gerade herrscht.

Wie ist Ihre Meinung zum Thema „Cloud" im Allgemeinen?

Pepperdine: Die Cloud ist ein Weg, Hardware-Ressourcen zu virtualisieren und sie einsetzbar und überschaubar zu machen. Vom administrativen und operativen Standpunkt aus gesehen, bietet eine Cloud-Umgebung viele Vorteile. Oft gehen mit einer anpassungsfähigen Umgebung auch Kostenvorteile einher. In der alten Welt und in der Welt, in der sich die meisten von uns gerade befinden, ist es so, dass man den Einkauf fragt, wenn man neue Hardware braucht, und diese dann zu seinem Cluster hinzufügt. Dann skaliert man die Anwendung, sodass auch die neue Hardware genutzt wird.

In einer Cloud-Umgebung kann man seine Applikationen elastisch machen, das heißt, wenn mehr Kapazität benötigt wird, wird sie bereitgestellt, was letztendlich auf das Bereitstellen neuer Hardware hinausläuft. Solange dieser Mechanismus zur Ver-

fügung steht, kann hoch- und runterskaliert werden, je nachdem wie viel Performance die Anwender gerade benötigen. Ich habe verschiedene Kunden besucht, die das bis zum Äußersten getrieben haben. Ihre Dienste laufen bei Amazon oder einem anderen der populären Cloud-Anbieter, sie haben aber ihre Anwendungen genommen und sie so elastisch wie möglich gestaltet. Ihre Applikationen können abhängig von der Last hoch- oder runterskalieren. Wenn sie also zusätzliche Kapazität benötigen, dann erhalten sie diese – aber auch nur dann zahlen sie dafür. Wenn die Kapazität nicht benötigt wird, geben sie sie frei und geben kein Geld für ungenutzte Ressourcen aus. Die Cloud verändert die Landschaft insofern, als dass wir Systeme entwickeln müssen, die mit dieser neuen Situation umgehen können. Nichtsdestotrotz ist die Cloud nicht die sprichwörtliche Wunderwaffe. Hat man nicht die Hardware, die unterstützt, was man tut, werden auch virtualisierte Umgebungen nicht helfen.

Pierre-Hugues Charbonneau, ein Java-EE-Consultant, hat eine Liste der Hauptgründe für Performance-Probleme in Java-Enterprise-Anwendungen aufgestellt. Wir werden Sie mit einigen davon konfrontieren. Als erstes: „Mangel an korrekter Kapazitätsplanung".

Pepperdine: Ja, das begegnet mir oft. Ich weiß nicht, ob es an der Planung der Gesamtkapazität liegt oder an der Dimensionierung einzelner Komponenten, aber es kommt sicherlich oft vor.

Wie ist es mit „Exzessive Java VM Garbage Collection"?

Pepperdine: Das ist ein Problem, das ganz oben steht. Hier gibt es einige Probleme: Falsche Heap-Konfiguration – entweder interne Größen oder die Gesamtgröße – oder vielleicht ist die Applikation selbst nicht speichereffizient. Wir haben da schon in beide Richtungen gearbeitet und manchmal auch in beide gleichzeitig.

Weiter mit „Zu viel oder zu wenig Integration in externe Systeme"?

Pepperdine: Das ist ein interessantes Problem. Die meisten Entwickler bauen auf eine simple „Absenden und Warten"-Strategie, was manchmal einfach nicht die adäquate Herangehensweise ist. Man muss die Nutzung des externen Systems in etwas einbetten, um eine vernünftige Fehlerbehandlung zu ermöglichen, also geplante Fehlerbehandlung und Wiederherstellung.

Was ist mit „Mangel an SQL-Tuning und genügend Datenbank-Ressourcen"?

Pepperdine: Oh ja, das ist ein großes Problem. Hier hilft es, in einen guten DBA zu investieren. Wir empfehlen immer, sich einen guten DBA zu besorgen, mit ihm zusammenzuarbeiten und wirklich zu prüfen, was vor sich geht. Ein nicht geringer Teil unseres Consultings besteht im Prüfen der Interaktion mit Datenbanken.

Wie steht es mit Java-EE-Middleware-Tuning-Problemen?

Pepperdine: Ja, ein Teil unserer Kunden möchte nur, dass wir ihre Middleware tunen, damit diese besser funktioniert. Manchmal müssen dazu lediglich Threadpool-Größen angepasst oder andere Dinge optimiert werden. Letztendlich hängt das vom eingesetzten Produkt und dessen Optimierungsmöglichkeiten ab. Was wir üblicherweise bei Middleware-Problemen beobachten, ist, dass virtualisierte Umgebungen mit Netzwerk-Kapazitäten sparen. Wie gut die Middleware funktioniert, ist sehr von der Netzwerk-Kapazität und -Latenz abhängig.

Wie stehen Sie zu „unzureichendes proaktives Monitoring"?

Pepperdine: Das spielt eine große Rolle. Viele warten bis zum Crash, bevor sie feststellen, dass ein Problem existiert; und dann rennen sie umher und versuchen es zu reparieren.

Zuletzt „Ausgelastete Hardware in gewöhnlicher Infrastruktur"?

Pepperdine: Ja, dies sieht man immer häufiger, insbesondere wenn Virtualisierung im Spiel ist. Von der Hardware-Perspektive aus gesehen, gibt es so etwas wie Threads und Prozesse nicht. Es gibt keines der Konstrukte, an die Entwickler üblicherweise denken. Es gibt lediglich Befehls- und Daten-Ströme. Zum Beispiel braucht nur eine aus dem Ruder laufende Applikation die gesamte Bandbreite zu belegen und der Server ist für alle anderen darauf laufenden Applikationen, die das Netzwerk benötigen, gestorben. Man muss immer die Gesamtauslastung der verfügbaren Ressourcen betrachten. Hinzu kommt, dass immer mehr auf Network Attached Storage gesetzt wird. Das kann ein gewöhnlich konfiguriertes System völlig aus dem Gleichgewicht bringen.

Hinweis: Das Interview führte Janice J. Heiss, Java Acquisitions Editor bei Oracle und Technologie-Redakteurin für das Java Magazine. Ins Deutsche übersetzt von Daniel van Ross von der Java User Group Deutschland e.V.

Neues NetBeans-Release mit umfassender Java-8- und HTML5-Unterstützung

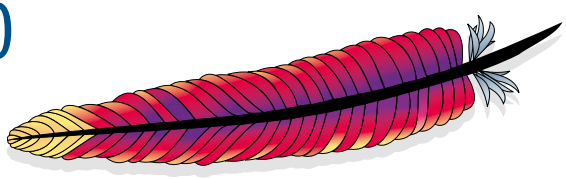
Die Verbesserungen im NetBeans Integrated Development Environment (IDE) 8.0 sorgen für Effizienzsteigerung im Umgang mit HTML5 und bringen zusätzlich sämtliche Java-SE-8-Funktionen. Daneben bietet die aktuelle NetBeans-Version zahlreiche Tools, Vorlagen und Musterbeispiele, um Entwicklern einen direkten Einstieg in die neueste Java-Version zu erleichtern. NetBeans IDE 8.0 unterstützt sowohl Java Standard Edition 8 (Java SE 8), Java Micro Edition 8 (Java ME 8) sowie deren Versionen für Embedded Devices, Java SE Embedded 8 und Java ME Embedded 8.

Java SE 8 Support erlaubt zudem die Entwicklung, Installation und das Debugging von Java-SE-Applikationen für Geräte wie den Einplatinen-Computer Raspberry Pi direkt im NetBeans-IDE. Die neue Nashorn JavaScript Engine ist ebenfalls integriert und ermöglicht die Entwicklung von Java- und JavaScript-Applikationen und deren gemeinsames Debugging.

Zu den HTML5-Features gehören eine verbesserte Code-Komplettierung für AngularJS, Knockout und weitere JavaScript-Frameworks, Karma-Test-Runner-Integration, Grunt-Integration, Web-Preview und Chrome-Developer-Tool-Integration sowie Support von On-Device-HTML5-Hybrid-Applikationen für die iOS- und Android-Plattformen. Auch der Java- und PHP-Editor wurden verbessert und die Unterstützung für Java EE ausgebaut.

Logging und Apache Log4j 2.0

Christian Grobmeier, grobmeier.de



Wir schreiben das Jahr 2014. Warum wird eigentlich immer noch über Logging diskutiert? Weil es wichtig ist. Und tatsächlich, es gibt einiges zu bereden. Warum braucht es ein Log4j 2.0? Und warum werden wir auch damit noch weiterdiskutieren müssen? In diesem Artikel geht es um die Geschichte des Java-Logging, um das neue Log4j und um eine Idee, wohin die Reise gehen könnte.

Im Dezember 2003 gründeten sechs Entwickler das Projekt „Apache Logging Services" (LS) unter dem Deckmantel der Apache Software Foundation (ASF) [1]. Darin enthalten ist das allseits bekannte Apache Log4j, das damit seinen 10. Geburtstag feiert. Das erste Log4j-Release war eigentlich im Jahr 1999, allerdings im Rahmen des Apache-Jakarta-Projekts.

Log4j hat das Loggen einfacher gemacht. Überhaupt, Log4j ist es zu verdanken, dass der Begriff „Logging" in das Bewusstsein vieler Entwickler gedrungen ist. Durch seinen flexiblen Aufbau wurde es zur Standard-Lösung in zahllosen Projekten. Einer (nicht-repräsentativen) Umfrage von „Zero Turnaround" zufolge verwenden heute etwa 52 Prozent aller Entwickler Log4j [2]. Vor einigen Jahren dürften es noch wesentlich mehr gewesen sein und Log4j hätte der König der Logging-Frameworks werden können. Doch es gab Probleme im Projekt, deren Nachwirkungen wir heute noch spüren.

Die Logging-Kriege

Diese Probleme fanden im Jahr 2005 ihren Höhepunkt. Jeder kann sich selbst ein Bild davon machen, wenn er die öffentlichen Mail-Archive des Projekts einsieht. Letztlich hat einer der Gründer von Apache Log4j, Ceki Gülcü, das Projekt gegabelt, überarbeitet und unter dem Namen „QOS Logback" veröffentlicht. Gülcü hat Apache LS nie verlassen und ist nach wie vor Mitglied des Project Management Committee (PMC). In seinem Blog führt er seine Kritik an der ASF aus [3].

Durch Wahl wird man in einem ASF-Projekt zunächst zu einem „Committer", dann später möglicherweise sogar Mitglied im PMC. Im Prinzip hat jedes gewählte Projekt-Mitglied eine Stimme bei Entscheidungen und jede Stimme zählt gleich. Nur in wenigen Situationen zählt die Stimme eines PMC-Mitglieds mehr. Es ist möglich, dass eine Person eine technische Änderung mit einem Veto blockiert, wenn sie nicht damit einverstanden ist.

Ein Veto führt normalerweise zu Diskussionen, die in vielen Fällen mit einem Konsens oder zumindest mit einem Kompromiss abschließen. Wenn die Meinungen allerdings zu unterschiedlich sind, kann dies auch zu einer Art „Deadlock" führen. Gülcü bevorzugt ein anderes Modell [4], in dem eher Stimmrechte nach Anzahl der Code-Modifikationen vergeben werden. Damit gäbe es immer eine starke Person, die das Sagen hätte. Dieser Ansatz ist aber mit der ASF nicht vereinbar. Außerdem gibt es auch gute Gründe, den ASF-Ansatz zu wählen, doch mehr dazu später.

Damit haben technische Unstimmigkeiten und die Unfähigkeit, einen Konsens zu finden, zum Beinahe-Ende von Log4j und der Gründung von Logback geführt. Logback konnte sich in relativ kurzer Zeit etablieren und hat einige interessante Änderungen eingeführt, die es in Log4j Version 1 nicht gab. Zudem wurden die ersten Versionen von „slf4j" veröffentlicht. Es handelt sich dabei um einen Wrapper für Logging-Frameworks, ebenfalls aus dem Hause QOS. Wer diesen verwendet, kann sein Logging-Framework auswechseln (zumindest theoretisch).

Die Idee ist nicht neu: Apache-Commons pflegt eine Komponente, die genau das Gleiche anbietet. Allerdings spielt bei Commons-Logging ein modernes API keine wesentliche Rolle. Der Fokus wird auf Stabilität gesetzt. Dieser eigentlich lobenswerte Ansatz rächt sich jedoch in diesem Fall: „slf4j" ist ebenfalls sehr stabil und bietet einfach mehr. Dementsprechend gibt es nur wenige Projekte wie zum Beispiel Apache Tomcat, die auf Commons-Logging setzen.

Nicht dass zwei Logging-Engines und zwei Logging-Facades genug wären – auch das JDK liefert bekanntermaßen seine eigene Logging-Engine mit: „java.util.logging" (JUL). Der Autor kenne keine Fans von JUL, jedoch gibt es tatsächlich einige Entwickler, die es benutzen. „Zero Turnaround" nennt in seiner Befragung etwa sieben Prozent Marktanteil.

Damit ist das Chaos perfekt. Als Entwickler will man eigentlich nur loggen. Aber womit? In großen Projekten kann es passieren, dass jedes Framework, das man verwendet, seine eigenen Logging-Abhängigkeiten mitnimmt. Im schlimmsten Fall hat man nicht nur die oben genannten, sondern möglicherweise auch noch ein paar Exoten und vielleicht sogar verschiedene Versionen im Einsatz. Letzten Endes ist mühseliges Konfigurieren angesagt, was einem schon mal den Geduldsfaden reißen lassen kann.

Einfach nur loggen

Logging ist ein mühseliges Thema, mit dem sich niemand gern beschäftigt. Beim Begriff „Log4j 2.0" schlägt einem oft eine Mischung aus Misstrauen und Stirnrunzeln entgegen. „Warum kann man nicht an einem Strang ziehen?", lautet der Vorwurf. Während man sich sein Web-Framework sehr genau ansieht und prüft, ob Leistung und Konzept zur eigenen Vision passen, interessiert sich kaum jemand für seine Logging-Engine. Obwohl Logging enorm wichtig ist, dürften die wenigsten wissen, was ein Receiver oder der Watchdog ist. Oft weiß man nicht einmal, welche Arten

von Logs es zu schreiben gibt. In vielen Projekten reicht eine Datei, die man zum Entwickeln beschreibt – und für den Live-Betrieb leer lässt.

Ein Logging-Framework ist eines der wenigen Projekte, die sich eigentlich nicht weiterentwickeln dürfen. Von log4j Version 1 wird erwartet, das man Upgrades nicht spürt und alles abwärtskompatibel bleibt. Erst seit dem Jahr 2012 bietet log4j 1.x keine Unterstützung mehr für Java 1.3. Wer einmal auf Logback gewechselt hat, erwartet dasselbe auch hier. Seit einiger Zeit arbeitet der Autor an Log4j 2.0. Wer sich mit dem Projekt outet, erntet oft eine hochgezogene Augenbraue: „Noch ein Framework?"

Aber haben wir tatsächlich noch ein neues Framework? Irgendwie „ja" und irgendwie „nein". Es ist eine neue Version, die fundamentale Probleme von Log4j Version 1 und Logback behebt. Log4j 2.0 hat allerdings nicht mehr viel mit Log4j Version 1 oder Logback zu tun, wenn man sich den Code ansieht. Wie gesagt: Die Architektur wurde stark überarbeitet. Nur Konzepte und Ideen wurden verwendet und daraus eine verbesserte, stabilere Engine gebaut.

Wer auf Log4j Version 2 wechselt, muss eine neue Abhängigkeit einbinden. Die Neuigkeiten wollen erlernt, die neuen Features erforscht sein. Eigentlich ist das spannend und macht auch Spaß. Aber die Begeisterung hält sich in Grenzen, wenn es sich um ein Logging-Framework handelt.

Auch Logging-Frameworks müssen sich entwickeln und aus ihren Fehlern lernen. Wir Entwickler müssen akzeptieren, dass wir uns auch ums Logging kümmern und mit den dortigen Entwicklungen Schritt halten müssen. Wären die bekannten Logging-Frameworks Web-Frameworks, würde man schon längst über deren Bewegungslosigkeit jammern.

Die Bedingungen für unsere Anwendungen verändern sich. Logging passiert nicht mehr nur auf dem Desktop (wenn überhaupt noch). Seit ein paar Jahren ist Logging in der Cloud wichtig. Logging auf mobilen Geräten. Mit den heutigen, teils fast unglaublichen Cloud-Applikationen haben wir eine neue Dimension von Logging erreicht – eine Art „Big Data"-Logging. Vergessen wir nicht: Egal in welcher Umgebung wir uns befinden, unsere Engines sind die Basis aller Logger.

Warum Log4j wiederbelebt wurde

Warum sich eigentlich die Mühe machen und Log4j 2.0 schreiben? Das ist unglaublich viel Aufwand, und immerhin gab es ja schon Logback. Ein nicht unwesentlicher Grund ist natürlich, wie bereits erwähnt, die technische Sicht. Einige Dinge, die in Log4 1.x, aber auch in Logback schlecht liefen, wurden aus Sicht der Entwickler verbessert. Beispielsweise ist es möglich, Log4j Version 2 als Audit-Framework einzusetzen. Logback verliert möglicherweise Log-Events, wenn es neu konfiguriert wird – Log4j Version 2 nicht.

Die neue Architektur basiert auf dem Concurrency-API von Java 5. Damit synchronisiert sich Log4j auf einem möglichst atomaren Level. Log4j Version 1 kämpft mit zahlreichen Deadlocks. Logback ist hier schon etwas besser, arbeitet an einigen Stellen trotzdem nicht atomar genug. So schwerwiegend diese Argumente auch bereits sein mögen: Technik allein war nicht der ausschlaggebende Punkt.

Anfangs wurde schon angesprochen, dass es ein Problem sein kann, wenn jeder eine Stimme hat. In einem harmonischen Team kann das jedoch auch ein unschlagbarer Vorteil sein. Die Motivation, jede Änderung zu sichten und auf die Waagschale zu legen, ist enorm hoch. Gerade große Umbauten müssen gut begründet werden, damit das Team sie akzeptiert. Dadurch steigert sich die Qualität enorm.

Das Projekt bleibt außerdem ständig in Bewegung. Nachdem es keinen Leader gibt, der alle Änderungen absegnen muss, gibt es auch kein Problem, wenn dieser einmal im Urlaub oder krank ist. Gerade wenn es ein kritischer Bug ist, kann schnell reagiert werden. Mit gleichen Rechten kann sich eine starke Community bilden. Genau dies ist geschehen.

Aber auch das gesamte Umfeld ist von Bedeutung. Apache Log4j wird im Rahmen der ASF entwickelt. Damit bleibt der Code immer frei verfügbar. Die Lizenz wird auch immer die Apache-Lizenz bleiben. Als Benutzer kann man davon ausgehen, dass die Software frei von den Ansprüchen Dritter ist, und wer möchte, kann auch mitwirken. Es kann sich zwar niemand einkaufen, aber engagierte Personen werden eingeladen und erhalten eine Stimme im Projekt. Es ist wichtig, diesen Punkt zu erwähnen. Nicht alle Projekte, die Open Source vermarkten, sind auch wirklich bereit, Änderungen von außen anzunehmen oder gar einen Entwickler mit Stimmrecht zu versehen. Dies kann man oft in Zusammenhang mit großen Projekten sehen, die von Firmen betrieben werden.

Die ASF ist also mehr als nur ein großes Repository. Sie bietet eine Plattform für die Community, freie Software-Entwicklung und Schutz für Benutzer und Entwickler.

Das größte Risiko besteht darin, dass ein Projekt verwaist und stillsteht. Allerdings können auch von Firmen getriebene Projekte verwaisen. Wer Software der ASF benutzt, macht sich nicht von einem Anbieter, der möglicherweise sehr restriktiv mit seinem Projekt umgeht, abhängig.

```
@Plugin(
   name = "Sandbox",
   type = "Core", elementType = "appender")
public class SandboxAppender extends AppenderBase {
    private SandboxAppender(String name, Filter filter) {
        super(name, filter, null);
    }

    public void append(LogEvent event) {
        // event.getMessage().getFormattedMessage());
    }

    @PluginFactory
    public static SandboxAppender createAppender(
         @PluginAttr("name") String name,
         @PluginElement("filters") Filter filter) {
         return new SandboxAppender(name, filter);
    }
}
```

Listing 1

Neben der verbesserten Technik, dem schnellen Entwicklungszyklus mit hoher Code-Qualität und dem Schutz der ASF hat jeder der Entwickler eigene Motive aufzuweisen. In meinem Fall: der enorme Spaß- und Lernfaktor, den die Zusammenarbeit mit so talentierten, hochqualifizierten Kollegen mitbringt.

```
<Configuration>
   <Appenders>
      <Sandbox name="myname" />
   </Appenders>
</Configuration>
```

Listing 2

Was neu an Log4j 2.0 ist

Ein paar Neuerungen wurden ja schon genannt. Glücklicherweise spickt das Projekt nicht nur bei den anderen und macht es dann besser, sondern versucht auch selbst Innovationen zu implementieren. Als Beispiel seien hier die neuen asynchronen Logger genannt, die eine sehr gute Performance aufweisen. Diese neuartigen Logger setzen auf den LMAX-Disrupter [5] auf, was ihnen zu einer Super-Power verhilft. Die Performance der Log4j-2-Logger ist um den Faktor zehn besser, verglichen mit den Altbekannten aus Log4j Version 1 und Logback [6]. Ohne diese Logger ist die Performance vergleichbar zu Logback.

Als weitere Neuerung stellt Log4j 2.0 einen Plug-in-Mechanismus bereit. Im Laufe der Zeit wurde klar, dass die Logging-Bedürfnisse manchmal nicht mit dem mitgelieferten Standard-Set an Funktionalität abgedeckt werden können. Zu vielfältig sind die Datenbanken und Use Cases von heute. Log4j 2.0 kann schnell und sauber erweitert werden. Mit folgender Anweisung in der Konfigurationsdatei werden Plug-ins im angegebenen Package gesucht: „<configuration … packages="de.grobmeier.log4j2.plugins">".

Das Plug-in selbst (in diesem Fall ein Appender) ist eine Klasse, die von „AppenderBase" ableitet und mittels Annotations konfiguriert wird. Damit legt man fest, wie das Plug-in später zu verwenden ist. Zu guter Letzt muss noch eine statische Factory-Methode angelegt werden, die das Plug-in erzeugt (siehe Listing 1).

Die Konfiguration hat sich außerdem verändert. Zunächst kann man nun die Namen der Elemente direkt angeben, wie im Beispiel von Listing 2. Neben XML ist nun auch JSON möglich. Das Plug-in könnte etwa so konfiguriert sein.

Ein besonders nettes Detail: Konfigurationen können automatisch neu geladen werden, ähnlich wie in Logback, nur mit dem Unterschied, dass beim Reload der Konfiguration, wie bereits erwähnt, keine Events verloren gehen. Weil wir gerade bei Logback sind: In Log4j 2.0 kann man quasi alle modernen Features erwarten, die auch in Logback gang und gäbe sind. Dazu gehören das Filtern von Log-Events (wie Marker) oder auch die folgende Schreibweise: „logger.debug("Logging with {} is fun!", log4j.getName());". Übrigens kann Log4j 2.0 auch perfekt mit slf4j verbunden werden. Log4j bietet zwar auch ein eigenes API, aber der Bedarf an slf4j-Unterstützung ist einfach nicht zu leugnen.

Houston, wir haben noch immer ein Problem

Wir haben ein freies Logging-Framework der ASF. Es gibt trotzdem noch ein Problem. Immer mehr und mehr Entwickler setzen auf slf4j, um die Logging-Engine darunter austauschbar zu machen. slf4j wird von QOS entwickelt. Jeder Benutzer macht sich damit von der Firma abhängig. Eine Alternative gibt es nicht. Obwohl der Autor selbst im Apache-Commons-Team aktiv ist, kann er Commons-Logging nicht ruhigen Gewissens empfehlen. Logj4 2.0 bietet auch ein API. Aber die ASF kann schwerlich genau das anbieten, was man für so ein wichtiges API benötigt: einen Standard.

Das JDK sollte die Interfaces für das Logging bereitstellen. Apache Log4j und alle anderen Anbieter wären damit austauschbar, weil die Interfaces vom JDK mitgeliefert würden; eine Abhängigkeit fiele weg. Die Unterstützung für diese Interfaces wäre vermutlich enorm.

Wenn es zu so einem Aufwand kommt, sollte auch geklärt werden, ob die Konfiguration standardisiert wird. Und wenn wir gerade dabei sind: Es kann dann auch gleich geklärt werden, wie Logging im Java-EE-Umfeld aussieht.

Tatsächlich gibt es bereits eine Initiative für dieses Szenario, und zwar im Rahmen eines Java.net-Projekts, das hoffentlich irgendwann einmal in einen JSR mündet [7].

Wenn wir erreichen, dass es zu einer Änderung im JDK kommt, sind vermutlich 95 Prozent aller Entwickler ein für alle Mal vom Logging-Ballast befreit. Und die restlichen fünf Prozent könnten sich in aller Ruhe nach dem für sie am besten passenden/geeigneten Logging-Framework umsehen.

Weiterführende Links

[1] http://logging.apache.org/charter.html
[2] http://zeroturnaround.com/rebellabs/the-state-of-logging-in-java-2013/
[3] Ceki Gülcü: Confusing intent and outcome. http://ceki.blogspot.de/2011/11/confusing-intent-and-outcome.html
[4] Ceki Gülcü: Commitocracy as an alternative for conflict resolution in OSS projects. http://ceki.blogspot.de/2010/05/committocracy-as-alternative-for.html
[5] http://lmax-exchange.github.io/disruptor
[6] http://logging.apache.org/log4j/2.x/performance.html
[7] https://java.net/projects/newlogging

Christian Grobmeier
cg@grobmeier.de

Christian Grobmeier ist Chair des Apache-Logging-Services-Projekts und auch in vielen anderen Apache-Projekten aktiv. Ab und zu befüllt er seinen Blog (www.grobmeier.de) oder schreibt an irgendeinem Buch. Manchmal findet er dann auch noch Zeit, seinem eigentlichen Job als freier Software-Entwickler nachzugehen. In seiner wenig vorhandenen Freizeit beschäftigt er sich mit Psychologie und der japanischen Bambuslängsflöte Shakuhachi.

WSO2 App Factory: Die Industrialisierung der Software-Entwicklung

Jochen Traunecker, gridsolut GmbH + Co. KG

Die Entwicklung und der Betrieb komplexer Software-Systeme kann durch die WSO2 App Factory unterstützt werden. Dieser Artikel beschreibt eine typische Deployment Pipeline des Continuous Deployments und betrachtet darauf aufbauend die WSO2 App Factory und WSO2 Private Platform as a Service näher.

Die Prinzipien der Industrialisierung halten zunehmend Einzug in die Domäne der Informations- und Kommunikationstechnologien (IKT) und stellen die Entwicklung und den Betrieb (DevOps) von Softwaresystemen vor neue Herausforderungen: Die Automatisierung und Standardisierung schreitet stetig voran und manifestiert sich in vielfältigen „X-as-a-Service"-Angeboten wie Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS) oder Software-as-a-Service (SaaS). Solche Service-Landschaften begünstigen die Etablierung von serviceorientierten Architekturen (SOA) und die darauf aufbauende Komposition komplexer Anwendungen.

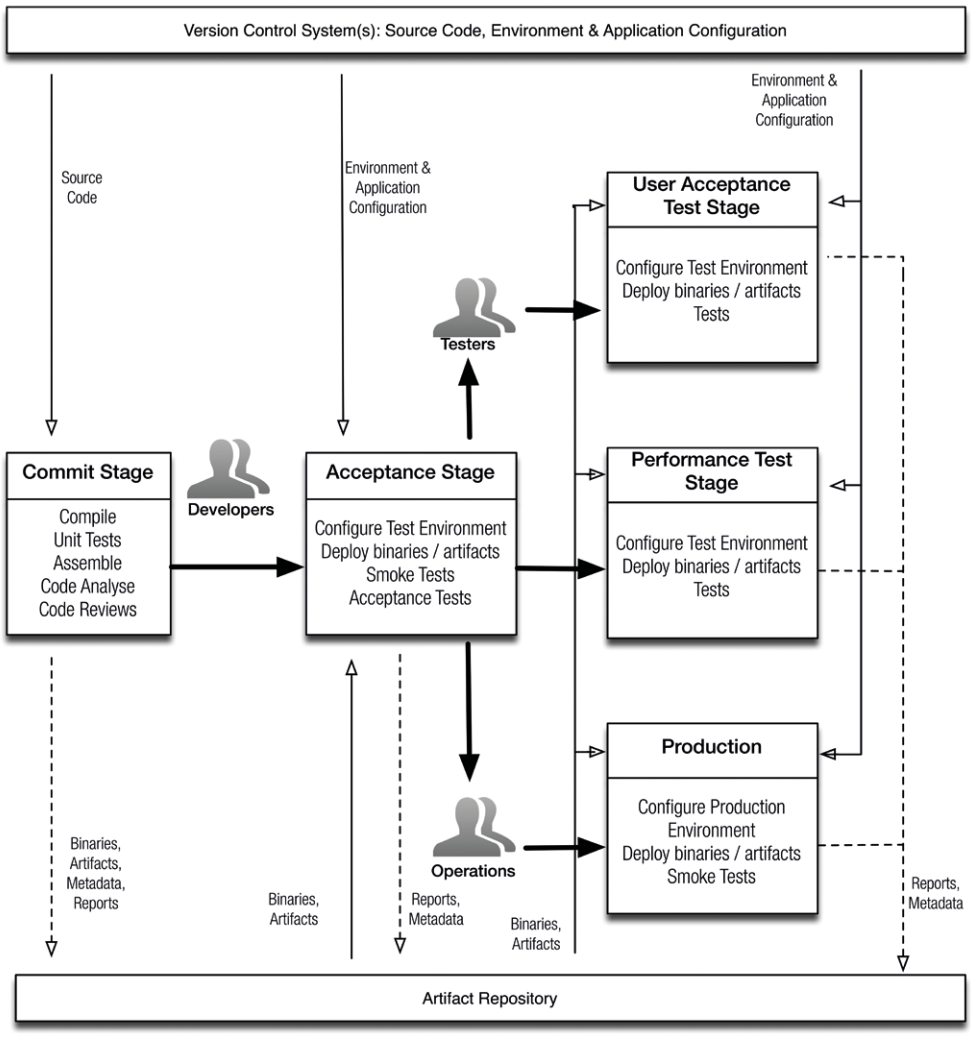


Abbildung 1: Deployment Pipeline nach [3]

Mit der losen Kopplung von Software-Komponenten durch eine SOA kann ein kontinuierlicher Verbesserungsprozess gelebt werden, da man einzelne Bausteine einfach anpassen, entfernen oder hinzufügen kann. Schließlich sollten sich die Entwickler von informationsverarbeitenden Systemen auf ihre Kernkompetenz fokussieren können und dabei möglichst viele Ressourcen für fachliche Aufgaben verwenden. Die allgemein bekannten Gesetze des Soziologen C. Northcote Parkinson [1] gelten gerade auch für die IKT.

Das Internet der Dinge nimmt zunehmend an Fahrt auf und immer mehr Informationsquellen liefern eine gewaltige Masse an Daten. Doch das Sammeln dieser Daten allein bringt noch keinen Mehrwert, sie müssen durch Software-Systeme verarbeitet, interpretiert und kontextabhängig verfügbar gemacht werden. Auf neue Erkenntnisse müssen Fachbereiche schnell reagieren können und ihre Prozesse entsprechend angleichen. Dies geht meist mit Anpassungen der beteiligten Software-Systeme einher, wobei die fachlichen Änderungen häufig wenig spektakulär sind. Gerade hier müssen sich die Software-Entwickler und die Verantwortlichen des Betriebes der Herausforderung stellen, den administrativen Aufwand zum Ausrollen neuer fachlicher Funktionen zu minimieren, ohne dabei die Stabilität des Gesamtsystems zu gefähr-

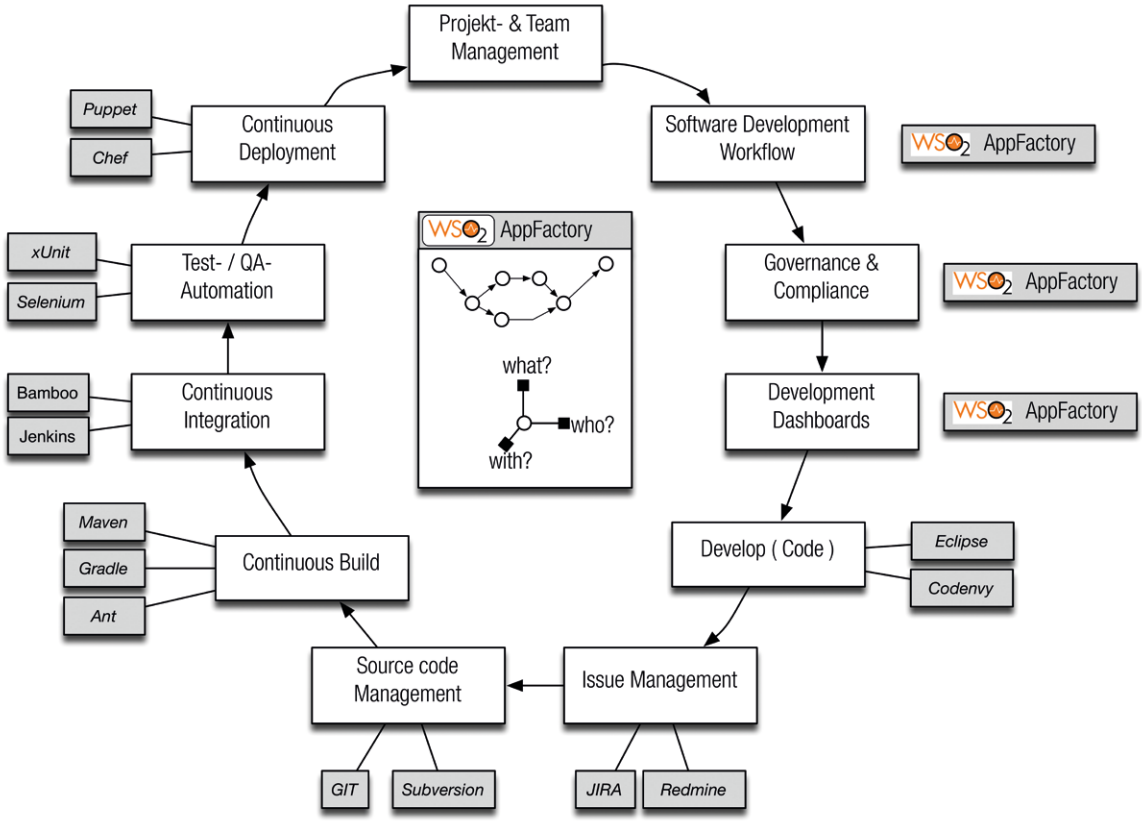


Abbildung 2: Integration von Werkzeugen

den. Hierbei kann die WSO2 App Factory Platform sowohl die Entwickler als auch den Betrieb unterstützen. Sie deckt dabei ein breites Spektrum fachlicher Anwendungen ab, das sich von Web-Anwendungen über Integrationsaufgaben bis hin zu komplexen Workflows erstrecken kann.

Die WSO2 App Factory definiert sich selbst als geteilte, elastische „Self Service Enterprise Platform", die alle Aspekte der Entwicklung und Verwaltung von Geschäftsanwendungen über deren gesamten Lebenszyklus hinweg abdeckt. Sie unterstützt die nebenläufige Entwicklung von Anwendungen durch mehrere Teams und fördert deren Zusammenarbeit durch Bereitstellung vielfältiger Kollaborationswerkzeuge. Wie alle WSO2-Produkte stehen auch die App Factory und deren Sourcecode unter der Apache-2.0-Lizenz und können für erste Gehversuche in einer öffentlich zugänglichen Preview als Software-as-a-Service ausprobiert werden [2a,2b].

Prinzipien des Continuous Deployments

Um die WSO2 App Factory besser einordnen zu können, sollen zunächst die von Jez Humble und David Farley beschriebenen fundamentalen Prinzipien, Methoden und Konzepte des Builds, des Tests und des Deployments von Software-Artefakten näher betrachtet werden [2]. Abbildung 1 skizziert die Phasen (Stages) einer typischen Deployment Pipeline (Deployment-Prozessmodell), die in [2] ausführlich diskutiert wird: Phasenübergreifend werden dabei Sourcecode und Konfigurationen versioniert und die entstehenden Artefakte werden in einem Repository abgelegt und wieder verfügbar gemacht.

Sobald die Entwickler der Überzeugung sind, dass der entwickelte Sourcecode ausgerollt werden kann, wird eine neue Deployment Pipeline (Instanz des Prozessmodells) abgearbeitet. Fehlgeschlagene Tests stoppen die Abarbeitung der aktiven Deployment Pipeline. Nach der Fehlerkorrektur beginnt das Spiel von vorn mit der Instanziierung einer neuen Deployment Pipeline:

- *Commit Stage*
  In dieser Phase erfolgt die eigentliche Software-Entwicklung. Der Code wird kompiliert, Unit Tests werden abgearbeitet und der Sourcecode einer Code-Analyse und einem Review unterzogen. Sind alle Tests und Analysen zufriedenstellend erfolgt, können Softwarepakete zusammengestellt werden (Assemble).
- *Acceptance Stage*
  Die Testumgebung wird aufgesetzt und konfiguriert, die zuvor zusammengestellten Softwarepakete werden in die Testumgebung ausgebracht und vielfältige Tests durchgeführt, die das fehlerfreie Zusammenspiel der jeweiligen Softwarepakete zum Ziel haben.
- *User Acceptance Stage*
  Die Testumgebung wird aufgesetzt und konfiguriert, sodass fachliche Testfälle abgearbeitet werden können.
- *Performance Test Stage*
  Auch hier wird eine Testumgebung aufgesetzt und nicht funktionale Tests im Hinblick auf Performance werden abgearbeitet.
- *Production*
  Die Produktivumgebung wird konfiguriert und die zuvor getesteten Softwarepakete werden ausgerollt.

Integration von Werkzeugen

Durch die WSO2 App Factory kann ein automatisierter Software-Entwicklungsprozess implementiert werden, der ein breites Spektrum an bekannten Werkzeugen integriert. In Abbildung 2 werden typische Phasen der Software-Entwicklung im Hinblick auf die eingesetzten Werkzeuge dargestellt und eine Auswahl bekannter Tools exemplarisch genannt. Die App Factory kann prinzipiell jedes Tools einbinden, sofern dieses über Schnittstellen verfügt. Im Grunde ist die Implementierung eines Software-Entwicklungsprozesses basierend auf existierenden Werkzeugen nichts anderes als ein typisches Integrationsprojekt mit all den bekannten Herausforderungen: Single Sign-on, anwendungsübergreifendes Monitoring, konsistente und einheitliche Benutzerschnittstellen etc.

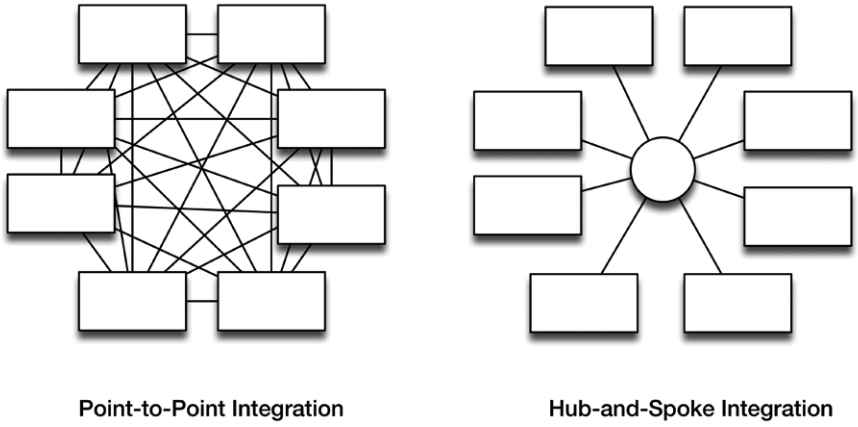


Abbildung 3: Integrationsmuster

Der WSO2 App Factory selbst steht für diese Integrationsaufgaben das gesamte WSO2-Middleware-Portfolio zur Verfügung („eat your own dog food") – angefangen auf der einen Seite beim vom WSO2 Enterprise Service Bus über den WSO2 Identity Server bis hin zum WSO2 Business Process Server auf der anderen Seite.

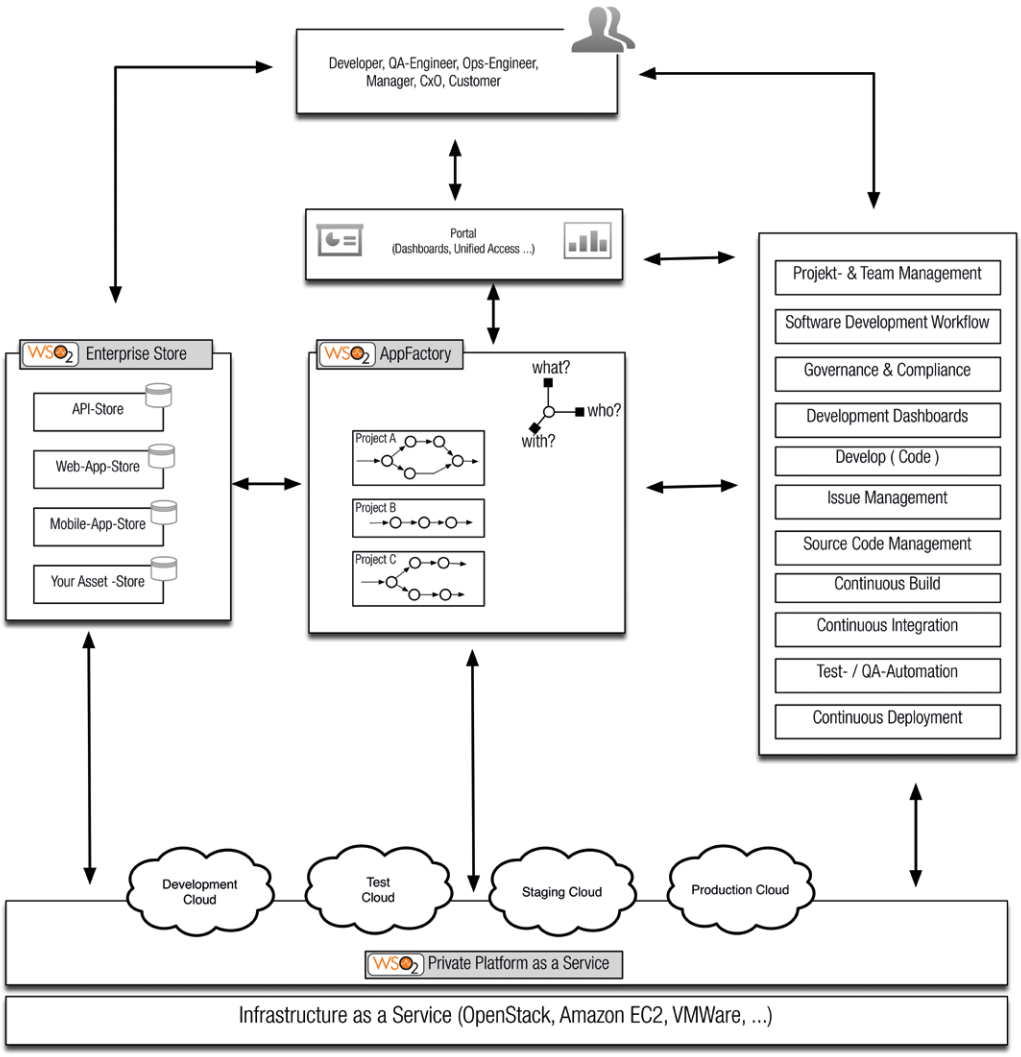


Abbildung 4: WSO2 App Factory im Kontext

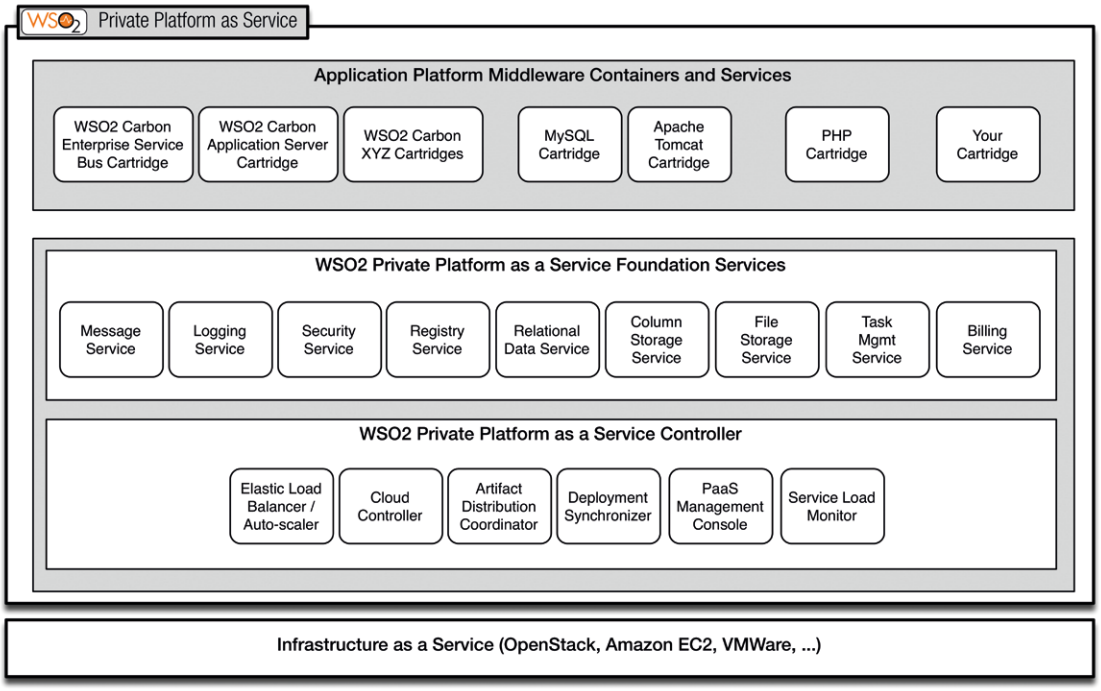


Abbildung 5: WSO2 Private Platform as a Service

Besonderes Augenmerk ist auf die in Abbildung 3 abstrakt dargestellten Integrationsmuster („Point to Point" und „Hub and Spoke") bei der Implementierung komplexer Software-Entwicklungsprozesse zu richten: Schnell schleicht sich eine direkte Abhängigkeit zwischen zwei Tools ein, die besser durch eine Indirektion über die App Factory dargestellt sein sollte. Gerade die Vielzahl an verfügbaren Plug-ins mächtiger Continuous-Integration-Lösungen verlockt zur direkten Integration.

Laufzeit-Umgebung

Ihr ganzes Potenzial kann die WSO2 App Factory im Zusammenspiel mit der WSO2 Private Platform as a Service (WSO2 PPaaS) ausspielen (siehe Abbildungen 4 und 5). Sie dient zum Aufbau privater Plattformen, die selbst auf IaaS-Lösungen aufbauen [4]. Die WSO2 PPaaS ist in drei Horizonte strukturiert:

- *Service Controller*
  Mit Komponenten wie Load Balancer, Auto-Scaler, Deployment Synchronizer und Monitoring werden die vielfältigen Applikationen dynamisch und bedarfsgerecht bereitgestellt.
- *Foundation Services*
  Diese Schicht bietet ein breites Spektrum an Basisdiensten zum Entwickeln von Anwendungen an. Zum Beispiel gibt es Message-Services zum Nachrichtenaustausch, Relational-Data- oder Column-Storage-Services als Persistent-Lösung.
- *Middleware Containers and Services*
  Diese Schicht bietet WSO2-Middleware-Komponenten als konsumierbare Dienste an (Enterprise Service Bus, Application Server etc.). Darüber hinaus können weitere Container in Form von Cartridges verfügbar gemacht werden.

Alle Komponenten der WSO2 PPaaS können sowohl über eine Web-Benutzeroberfläche administriert und konfiguriert werden als auch über APIs. So lässt sich zum Beispiel durch die APIs der Services-Controller-Schicht eine relationale Datenbank instanziieren und mit einem Datensatz initial betanken, der selbst wiederum vom File-Storage-Service bereitgestellt wird.

Die WSO2 PPaaS kann im Zusammenhang mit der WSO2 App Factory zum Bereitstellen einer Development, Test, Staging und Production Cloud genutzt werden. Die App Factory stellt durch die verfügbaren APIs der WSO2 PPaaS beliebig komplexe Test-Umgebungen bedarfsgerecht bereit und gibt die belegten Ressourcen nach erfolgten Testläufen wieder frei.

Für die jeweiligen Benutzer (Developer, QA-Engineer etc.) bietet die App Factory ein zentrales Portal, das eine einheitliche erste Sicht auf die eingesetzten Werkzeuge und deren Zustände bietet. Die jeweiligen Werkzeuge und deren spezifische Eigenschaften werden durch die WSO2 App Factory jedoch nicht verdeckt. Die Benutzer arbeiten weiterhin mit diesen Werkzeugen direkt.

Die entstandenen Artefakte und Softwarepakete können im WSO2 Enterprise Store abgelegt und über diesen verfügbar gemacht werden. Dieser hat eine zentrale Bedeutung im Application Lifecycle Management, da durch ihn das Beziehungsgeflecht zwischen Anbieter und Konsument (API, Web-App, Mobile App etc.) bekannt ist und entsprechend verwaltet werden kann. In [5] werden exemplarisch das API-Management und der API-Store ausführlich vorgestellt.

Fazit

Die WSO2 App Factory kann als flexibler Baukasten zum Aufbau individueller DevOps- und Application-Lifecycle-Management-Lösungen herangezogen werden. Sie stellt keine in sich abgeschlossene Blackbox dar, sondern bietet vielfältige Anpassungs- und Erweiterungsoptionen an. Durch diese Flexibilität können sich

die Entwicklungs- und Betriebsabteilungen auf die gemeinsame Definition, Implementierung und laufende Verbesserung der Build-, Test- und Deployment-Prozesse konzentrieren. Und im Idealfall bedingt die Entwicklung einer neuen fachlichen Anwendung für den Betrieb lediglich eine höhere Auslastung der vorhandenen Infrastruktur ohne nennenswerte Mehrarbeit.

Quellen

[1] http://de.wikipedia.org/wiki/Parkinsonsche_Gesetze
[2a] https://appfactorypreview.wso2.com
[2b] http://wso2.com/cloud/app-factory
[3] Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation; Humble and Farley, Addison-Wesley 2010; ISBN 0-321-60191-2
[4] Suse Cloud: https://www.suse.com/de-de/products/suse-cloud
[5] Prinzipien des API-Managements; T. Unger und J. Traunecker, Java aktuell, Ausgabe 1/2014

Jochen Traunecker
jochen.traunecker@gridsolut.de

Database-DevOps mit MySQL, Hudson, Gradle, Maven und Git

Michael Hüttermann

DevOps, ein Kofferwort aus Development (Entwicklung) und Operations (Betrieb), beschreibt die optimierte Zusammenarbeit eben dieser Funktionen in IT-Unternehmen und -Projekten. In der Software-Entwicklung liegen Datenbanken häufig auf einem kritischen Pfad. Dieser Artikel liefert eine Definition von DevOps und erläutert, wie Datenbank-DevOps mit Konzepten und Werkzeugen konkret aussehen kann.

Konflikte zwischen Entwicklung und Betrieb können viele Ursachen haben, etwa unterschiedliche Ziele (mehr Veränderungen in kurzer Zeit für die Entwicklung, wenig Veränderungen und Stabilität in der Produktion für den Betrieb), unterschiedliche Prozesse und Konzepte (pragmatische Ansätze für Entwicklung, mehr Fokus auf Rückverfolgbarkeit für den Betrieb) und unterschiedliche Werkzeuge (Entwicklungswerkzeuge für Entwicklung, produktionstaugliche Ansätze für den Betrieb). Diese Unterschiede reißen oft Lücken zwischen den Abteilungen beziehungsweise begünstigen Silos, die als Nächstes erläutert werden.

Verschiedene Teams

In den traditionellen Konstellationen umfasst der Begriff „Entwicklung" die Programmierer im Team. Tester sind traditionell dedizierte Projekt-Rollen, die einen Großteil ihrer Aktivitäten erst starten, nachdem die Programmierer ihre Arbeit beendet haben. Dank der agilen Software-Entwicklung überlagern sich diese Rollen und deren Aktivitäten zunehmend, sodass mittlerweile ein inkrementell/iterativer Ansatz mit einer Verzahnung von Programmierung und Test mehr die Regel als die Ausnahme ist. Der „Betrieb" umfasst Rollen wie Datenbank-, System- und Netzwerk-Administratoren sowie weitere Administratoren, die Server und Systeme einrichten und warten. Der Betrieb begleitet die Änderungen auf der sogenannten „letzten Meile", bis diese dem Benutzer in der Produktion zur Verfügung gestellt sind. In einer barrierelastigen Umgebung formen Entwicklung und Betrieb lokal optimierte Gruppen mit jeweils eigenen Zielen, Prozessen und Werkzeugen (siehe Abbildung 1).

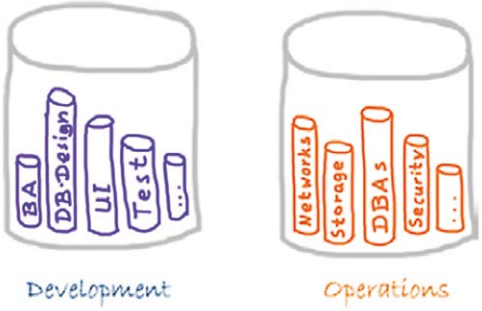


Abbildung 1: Entwicklung und Betrieb, zwei unterschiedliche Teams

Unterschiedliche Ansätze

Es ist nicht immer einfach, die richtige Software (Effektivität) richtig (Effizienz) zu entwickeln. Insbesondere die Entwicklung von Datenbanken ist recht häufig eine Herausforderung. Warum? Traditionell stellt die Entwicklung ihre Änderungen via Check-in in ein Versionskontrollsystem den Kollegen und anderen Teams zur Verfügung. Fachlicher Anwendungscode wird lokal getestet und verfolgt, ohne dass diese Arbeiten mit der Arbeit von Kollegen direkt interferieren würden.

Deployments auf Zielumgebungen werden häufig über einen zentralen Build-Server kanalisiert und regelmäßig durchgeführt. Auf der anderen Seite fokussiert sich die Datenbank-Entwicklung häufig auf eine zentrale Ressource, auch wenn Entwickler über eigene Datenbanken oder eigene Schemata auf einer zentralen Datenbank verfügen. Ferner ist Datenbank-Entwicklung kein Prozess von schlichtem „Copy & Paste", vor und zurück. So lässt sich zum Beispiel eine Datenbank-Tabelle nicht einfach löschen und mit einer neuen Struktur wiederherstellen.

Selten sind zwei Status tatsächlich gleich, da entweder die Quelle oder das Ziel von früheren Installationen und Entwicklungen geändert wurde. Als Ergebnis ergeben sich aus all diesen Aspekten große Unterschiede zwischen Entwicklung der eigentlichen Fachanwendung sowie der Datenbank beziehungsweise zwischen Entwicklung und Betrieb.

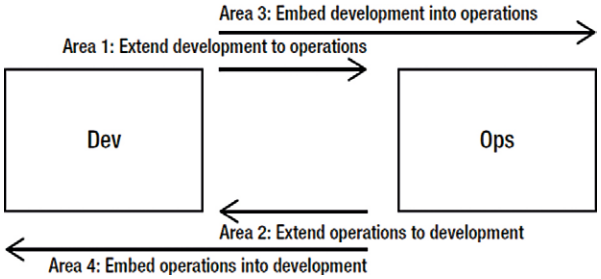


Abbildung 2: Die DevOps-Matrix

DevOps in aller Kürze

DevOps beschreibt Praktiken, die den Softwarebereitstellungsprozess ganzheitlich betrachten und optimieren. DevOps möchte Ziele, Prozesse und Werkzeuge von Entwicklung und Betrieb zueinander ausrichten und damit die Zusammenarbeit verbessern. Durch einen verbesserten Fluss soll Software schneller und in besserer Qualität vom Programmierer zum Nutzer gelangen. DevOps umfasst zahlreiche Aktivitäten und Aspekte wie die folgenden:

- *Culture*
  Software wird von Menschen für Menschen gemacht. Jedes Projekt und jedes Unternehmen hat eine eigene Kultur.
- *Automation*
  Automatisierung ist essenziell, um schnelle Rückkopplungen zu erhalten.
- *Measurement*
  DevOps wählt bestimmte Ansätze zur Definition und Messung von Qualität.
- *Sharing*
  Eine Kollaboration durch Teilen von Wissen und Erfahrungen ist unumgänglich.

DevOps geht von einem umfassenden Ansatz aus, der den kompletten Kernprozess begleitet. Ein erster Schritt in diese Richtung kann ein Value-Stream-Mapping liefern, um die Wertschöpfungskette ganzheitlich zu erheben, noch besser zu verstehen und Flaschenhälse zu identifizieren. Aufbauend darauf können auch kausale Abhängigkeiten erkannt und Feedbackschleifen eingeflochten werden. Dabei ist allen Teilnehmern genug Platz zum Experimentieren einzuräumen. Damit ist kein sinnfreies Basteln gemeint, sondern das zielgerichtete, doch experimentelle, explorative Arbeiten mitsamt Pufferzeiten, um auf Änderungen auch zukünftig möglichst flexibel eingehen zu können und Fallstricken bei der Automation zu begegnen, siehe dazu auch Kapitel 3 in „DevOps for Developers", Hüttermann, Apress, 2012.

```
michael@michael-
VirtualBox:~/talk/project/devops/src/main/resources/db/migration$
ls -la
total 16
drwxrwxr-x 2 michael michael 4096 Sep 22 11:16 .
drwxrwxr-x 3 michael michael 4096 Sep 22 11:16 ..
-rw-rw-r-- 1 michael michael 112 Sep 18 07:59 V1__Create_person_table.sql
-rw-rw-r-- 1 michael michael 149 Sep 18 07:28 V2__Insert_persons.sql
```

Listing 1

```
create table PERSON (
ID int not null auto_increment,
NAME varchar(80) not null,
primary key (ID)
);
```

Listing 2

```
INSERT INTO PERSON (NAME) VALUES ('Peter Meyer');
INSERT INTO PERSON (NAME) VALUES ('Peter Bonnd');
INSERT INTO PERSON (NAME) VALUES ('Klara Korn');
```

Listing 3

```
<profile>
   <id>db</id>
   <build>
      <plugins>
         <plugin>
            <groupId>com.googlecode.flyway</groupId>
            <artifactId>flyway-maven-plugin</artifactId>
            <version>2.1.1</version>
            <configuration>
               <user>fly</user>
               <password>way</password>
               <driver>com.mysql.jdbc.Driver</driver>
               <url>jdbc:mysql://localhost:3306/mydb</url>
               <baseDir>db/migration</baseDir>
            </configuration>
         </plugin>
      </plugins>
   </build>
   <dependencies>
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <version>5.1.25</version>
      </dependency>
   </dependencies>
</profile>
```

Listing 4

```
[INFO] --- flyway-maven-plugin:2.1.1:migrate (default-cli) @ de
vops ---
[INFO] Creating Metadata table: `mydb`.`schema_version`
[INFO] Current version of schema `mydb`: << Empty Schema >>
[INFO] Migrating schema `mydb` to version 1
[INFO] Migrating schema `mydb` to version 2
[INFO] Successfully applied 2 migrations to schema `mydb` (execu-
tion time
00:00.234s).
```

Listing 5

```
mysql> show tables;
+----------------+
| Tables_in_mydb |
+----------------+
| PERSON         |
| schema_version |
+----------------+
```

Listing 6

```
mysql> select * from PERSON;
+----+-------------+
| ID | NAME        |
+----+-------------+
| 1  | Peter Meyer |
| 2  | Peter Bonnd |
| 3  | Klara Korn  |
+----+-------------+
3 rows in set (0.00 sec)
```

Listing 7

Bei einer Implementierung von Datenbank-DevOps kann man die DevOps-Matrix heranziehen (siehe Abbildung 2). Bereich 1 der Matrix beschreibt die Ausweitung agiler Verfahren von der Entwicklung in Richtung Betrieb. Ein häufiger Anwendungsfall im Datenbank-Kontext ist das Pflegen von Datenbank-Skripten über ein Versionskontrollsystem und die Verwendung von Werkzeugen wie Flyway, etwas später in diesem Artikel weiter thematisiert. Bereich 2 ist die Ausweitung von Praktiken in die andere Richtung, also vom Betrieb in Richtung Entwicklung. Für Datenbank-DevOps kann das bedeuten, Infos über Live-Traffic oder Datenbank-Ressourcenkonflikte auch der Entwicklung zur Verfügung zu stellen.

Bereich 3 bettet die Entwicklung in den Betrieb ein. Damit ist keine organisatorische Umverteilung gemeint, sondern vielmehr die Ausrichtung von Zielen oder die Aufnahme von nicht-funktionalen Anforderungen sehr früh in die Entwicklung. Gemeinsame Ziele können sein:

- 80 Prozent der Datenbank-Abfragen liefern Ergebnisse auf dem Bildschirm in weniger als zwei Sekunden (ein gemeinsames Performance-Ziel)
- Kein Gebrauch von Technologien, die einen späteren Wechsel auf andere Technologien ausschließen oder erschweren, wie Datenbank-proprietäre SQL-Routinen (ein gemeinsames Portabilitäts-Ziel)
- 250 Milliarden Datensätze können mit der gegebenen Hardware gespeichert werden, bei gleichzeitiger Einhaltung der Performance-Ziele (ein gemeinsames Kapazitäts-Ziel)
- Automatisierte Tests existieren für alle Komponenten einschließlich Infrastruktur-Code (ein gemeinsames Wartbarkeits-Ziel)

Schließich beschreibt Bereich 4 die Einbettung des Betriebs in die Entwicklung. Dies kann zum Beispiel durch die Bereitstellung von Informationen innerhalb der Entwicklung erreicht werden, ohne bei jedem Informationszugriff auf einen DBA angewiesen zu sein, oder die Aufnahme eines Betrieblers zu 10 Prozent seiner Kapazität in ein Scrum-Team der Entwicklung. Nachdem wir nun konkrete Beispiele anhand der DevOps-Matrix durchleuchtet haben, vertiefen wir nachfolgend die Bereiche, allen voran Bereich 1, mit weiteren technischen Details.

Datenbank-DevOps

Um die Zusammenarbeit zwischen Entwicklung und Betrieb zu optimieren, verbindet eine robuste Lösung für Datenbank-DevOps traditionelle Lösungen des Betriebs mit Ansätzen wie Datenbank-Versionskontrolle, Continuous Integration und Automatisierung. Die folgenden Muster können helfen, Datenbank-DevOps einzuführen:

- Legen Sie alle Code-und Datenbank-Elemente (sowie alle Änderungssätze) in die Versionskontrolle.

```
+----------------+----------------------------+---------------------+---------+
| Version        | Description                | Installed on        | State   |
+----------------+----------------------------+---------------------+---------+
| 1              | Create person table        | 2013-09-19 20:52:32 | Success |
| 2              | Insert persons             | 2013-09-19 20:52:32 | Success |
| 3              | InsertUpdate persons       | 2013-09-19 20:55:52 | Success |
+----------------+----------------------------+---------------------+---------+
```

Listing 8

```
apply plugin: 'java'
apply plugin: 'flyway'

flyway {
   user = 'fly'
   password = 'way'
   driver = 'com.mysql.jdbc.Driver'
   url = 'jdbc:mysql://localhost:3306/mydb'
   baseDir = 'db/migration'
}

buildscript {
   repositories {
      mavenCentral()
   }

dependencies {
   classpath "com.googlecode.flyway:flyway-gradle-plugin:2.2"
   classpath "mysql:mysql-connector-java:5.1.25"
   }
}

repositories {
   mavenCentral()
}

dependencies {
   testCompile 'junit:junit:4.11'
}
```

Listing 9

```
UPDATE PERSON SET NAME='Peter Bond' WHERE ID=2;
DROP PROCEDURE IF EXISTS AddPerson;
delimiter //
CREATE PROCEDURE AddPerson (IN myvalue VARCHAR(80))
BEGIN
INSERT INTO PERSON (NAME) VALUES (myvalue);
END //
delimiter ;
CALL AddPerson(,Donald Luck');
```

Listing 10

- Erstellen Sie SQL (Structured Query Language)-Skripte, die angewendet werden müssen, um zur nächsten Version migrieren zu können (Roll-Forward). Prüfen Sie, ob Sie gegebenenfalls Roll-Backward vollständig wegoptimieren können.
- Erstellen Sie für jede logische Einheit von Änderungen (Change-Sets) eine Datei, die die Änderungsskripte beinhaltet. Geben Sie der Datei einen eindeutigen Namen einschließlich einer Nummerierung, die sich kardinal skalieren lässt.
- Erstellen Sie Baselines, frieren Sie alle Konfigurationselemente ein, einschließlich Anwendung, Middleware und Datenbank.
- Setzen Sie bei Migrationen auf diese Baselines auf. In Fällen einer vollständigen Installation wenden Sie alle Change-Sets aufbauend auf der Baseline an. Bei inkrementeller Installation führen Sie ausschließlich die jeweils fehlenden Change-Sets aus.
- Platzieren Sie Ihre Datenbank-Migrationsskripte in entsprechende Verzeichnisse.
- Berücksichtigen Sie Maßnahmen zur Überwachung, um „Mean Time To Repair" (MTTR) und „Mean Time To Detect" (MTTD) zu minimieren, sowie Smoke-Tests.
- Speichern Sie die Datenbank-Version. Ein Ansatz ist, eine dedizierte Datenbank-Tabelle fortzuschreiben, die Metadaten beinhaltet, insbesondere darüber, in welchem Zustand sich die Datenbank aktuell befindet.

Eine exemplarische Werkzeugkette

Sie können selbst eine Lösung entwickeln und die weiter oben aufgeführten Rezepte vollständig implementieren oder Sie nutzen ergänzend ein Werkzeug, das dabei hilft, diese Konzepte umzusetzen. Ein solches Werkzeug ist Flyway (siehe http://flywaydb.org). Im Folgenden werden wir ein Beispiel skizzieren, in dem wir Werkzeuge wie Flyway, Maven, Gradle, Git und Hudson miteinander integrieren. Beginnen wir mit einem Blick in den Verzeichnisbaum, in dem die Migrationsskripte liegen (siehe Listing 1).

Im Moment liegen zwei Dateien im Verzeichnis. Eine Datei beinhaltet ein Change-Set, also einen Satz von nativen SQL-Statements, die auf eine Datenbank angewendet werden sollen. Auch das ist DevOps: Definieren Sie Change-Sets ausschließlich in nativem SQL und kompilieren Sie beispielsweise diese SQL-Statements nicht in eine höhere Programmiersprache hinein. Die erste Datei ist „V1__Create_person_table.sql". Es handelt sich um eine

DDL-Datei (Data Definition Language); Listing 2 zeigt ihren Inhalt.

Die zweite Datei ist eine DML-Datei (Data Manipulating Language). Die Datei „V2__Insert_persons.sql" besteht aus drei Anweisungen zum Anlegen von drei Sätzen in der zuvor erstellten Tabelle (siehe Listing 3).

Da wir automatisieren möchten, schauen wir zunächst auf eine Maven-POM, die die Migration anstoßen soll (Weitere Infos zu Maven siehe http://maven.apache.org). Die POM ist der Ort, an dem wir beispielsweise Datenbank-Verbindungsparameter hinterlegen können, auch parametrisiert. Achtung: Abhängig von konkreten Anforderungen ist natürlich nicht immer ein Build-Skript der beste Ort, um Migrationsskripte anzustoßen. In unserem Fall ist die Logik in einem Maven-Profil hinterlegt. Listing 4 zeigt den relevanten Ausschnitt.

Flyway arbeitet mit Classpath-Scanning. Bei der Nutzung aus Maven heraus müssen Migrationsskripte auf das Ziel kopiert werden, also zum Beispiel der Maven-Zielordner „target". Der Maven-Anruf kann dann so aussehen: „clean install flyway:migrate –Pdb". Um den Aufruf noch komfortabler zu gestalten, von informativer Visualisierung zu profitieren oder auch eine umfangreiche Delivery-Pipeline aufzubauen, integrieren wir auch Hudson, den Build Server (siehe http://hudson-ci.org). Ein automatischer Aufruf des Maven-Profils via Hudson kann dann folgende Konsolenausgabe zur Folge haben (siehe Listing 5).

```
git add V3__InsertUpdate_persons.sql
git commit -m "change" .
git push
```

Listing 11

```
[INFO]
[INFO] --- flyway-maven-plugin:2.1.1:migrate (default-cli) @
devops ---
[INFO] Current version of schema `mydb`: 2
[INFO] Migrating schema `mydb` to version 3
[INFO] Successfully applied 1 migration to schema `mydb` (execu-
tion time
00:00.069s).
```

Listing 12

In unserem Fall war die Datenbank zunächst leer, es wurden also noch keine Migrationen angewendet. Nach etwas Bootstrapping (Erstellen von Meta-Informationen) wendet Flyway die beiden Migrationen auf die Datenbank an, da wir zwei SQL-Dateien in das Verzeichnis gelegt haben. Listing 6 zeigt, was ein schneller Gegen-Check auf der Datenbank liefert.

Werfen wir nun einen kleinen Blick in die Tabelle „Person". Die Tabelle hat drei Sätze, da die beiden Migrationsskripte die Tabelle anlegen und drei Sätze einfügen (siehe Listing 7).

Auch das verlief erfolgreich. Flyway bietet die Möglichkeit, durch eine „flywayInfo" den Stand der Migrationen zu zeigen. An

```
mysql> select * from PERSON;
+----+-------------+
| ID | NAME        |
+----+-------------+
| 1  | Peter Meyer |
| 2  | Peter Bond  |
| 3  | Klara Korn  |
| 4  | Donald Luck |
+----+-------------+
4 rows in set (0.00 sec)
```

Listing 13

das Ergebnis kommen wir beispielsweise auch durch Gradle (siehe http://www.gradle.org) und den Aufruf „gradle flywayInfo" (siehe Listing 8).

Listing 9 zeigt die Gradle-Build-Datei, die funktional vergleichbar ist mit der

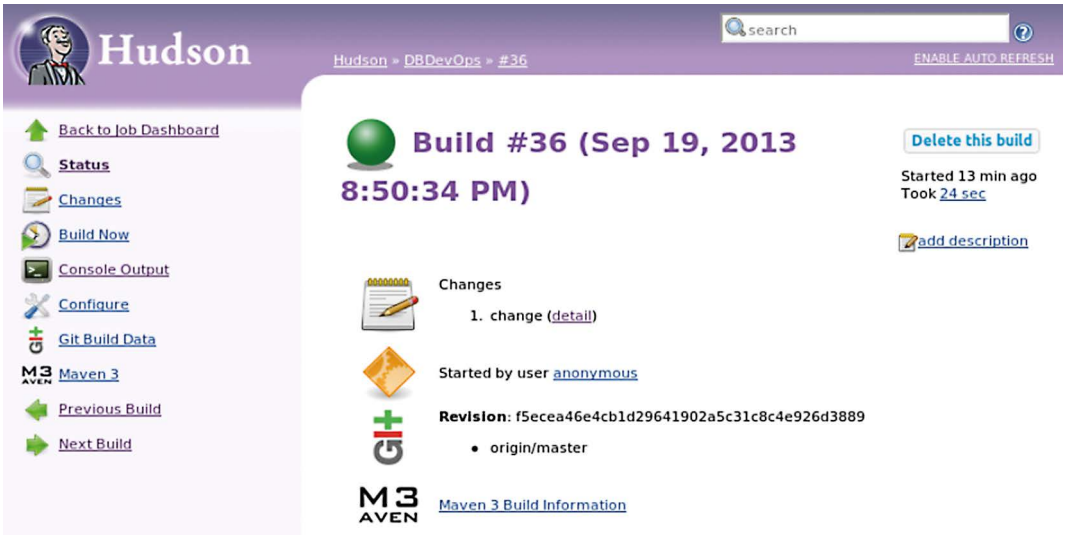


Abbildung 3: Hudson lauscht auf Änderungen in Git und löst Datenbank-Konvertierungen aus

Maven-POM. Die Datei ist ebenfalls Teil des Quellcodes im Versionskontrollsystem.

Wir nehmen nun die nächste Migration in Angriff. Eine weitere Datei, unser drittes Migrationsskript mit dem Namen „V3__InsertUpdate_persons.sql", enthält ein Update auf einen vorhandenen Datensatz sowie die Erstellung einer Stored Procedure. Diese wird dann schließlich genutzt, um einen weiteren Datensatz einzufügen (siehe Listing 10).

Wir können die neue Datei nun in das Git-Repository einspeisen und so eine Datenbank-Migration auf einem Testsystem anstoßen (siehe Listing 11). Hudson erkennt die Änderungen in Git und baut das Projekt (siehe Abbildung 3).

Die Werkzeugkette liest den aktuellen Zustand der Datenbank und erkennt, dass nun ein neues Migrationsskript vorliegt, das angewendet werden soll. Die aktuelle Version der Datenbank ist „2". Die neu verfügbare Migration hat die Nummer „3" (siehe Listing 12).

Wir haben anschließend einen vierten Datensatz in unserer Tabelle „PERSON" sowie eine aktualisierte Spalte, offenbar ein Fix eines zuvor eingeführten Rechtschreibfehlers. Der neue Tabelleneintrag wurde durch den Aufruf der frisch erstellten Stored Procedure eingefügt. Wir schauen in die Tabelle, um uns vom ordnungsgemäßen Ablauf zu überzeugen (siehe Listing 13). Nun ist die Datenbank in den gewünschten Zielzustand überführt.

Fazit

Dieser Artikel führte in DevOps ein, eine moderne Herangehensweise, die sich eine Angleichung von Zielen, Prozessen und Werkzeugen zwischen Entwicklung und Betrieb zum Ziel gesetzt hat. Aufbauend auf einem konkreten Beispiel haben wir eine kleine Rundreise durchgeführt durch relevante Praktiken und Werkzeuge.

Michael Hüttermann
michael@huettermann.net

Java-Champion Michael Hüttermann (http://huettermann.net) ist freiberuflicher Delivery Engineer und Experte für DevOps, Continuous Delivery und SCM/ALM. Er schrieb die ersten Bücher zu „Agile ALM" (Manning, 2011) und DevOps („DevOps für Entwickler", Apress, 2012).

Entweder ... oder Fehler

Heiner Kücker, Freiberufler

Es ist nicht sicher, ob Scala eines Tages Java ablösen wird, aber in Scala und seinen Libs sind Pattern/Idiome manifestiert, die der Alltags-Java-Programmierung durchaus guttun würden.

Der Autor hatte mal ein Problem im Programm eines Kollegen übernommen. Darin gab es eine Suchfunktion, die ein recht merkwürdiges Verhalten hatte. Der Suche wurde eine Liste von Nummern, beispielsweise Artikel-Nummern, übergeben. War diese Liste „null", wurde im SQL-Select-Statement keine „WHERE IN"-Klausel erzeugt. War die Liste leer, wurde die Suche nicht ausgeführt, was aber kein Problem war, da die Suche Teil eines SQL-Union-Befehls war und im weiteren Java-Code noch andere Objekte zur Ergebnisliste hinzugefügt wurden. War die Liste nicht leer, wurde eine „WHERE IN"-Klausel erzeugt. Nochmal ganz langsam:

- *Liste null*
  Alle Werte erlaubt
- *Liste leer*
  Keine Werte erlaubt
- *Liste mit Werten*
  Werte erlaubt

„null" ist also eine magische Nummer für das Ignorieren des Such-Parameters. Die zweite Variante, die leere Liste, würde ungültiges SQL erzeugen, das aber sowieso keine Ergebnisse liefern würde. Die dritte Variante, die „nicht leere" Liste mit Werten, wird in SQL als Selektions-Kriterium eingebaut. Logisch, dass hier nichts kommentiert war.

Das Ganze wurde noch verschleiert durch ein Parameter-Objekt mit Gettern und Settern, in dessen Member-Initialisierung eine leere Liste initialisiert wurde, die irgendwo im Code kommentarlos mit „null" überschrieben wurde, also ein globaler veränderlicher Zustand. Keine Ahnung, warum Eclipse bei der Referenz-Suche von Membern nicht die Getter und Setter mit einbezieht; für den Autor noch ein Grund, unsinnige Getter und Setter zu entfernen.

Parameter-Objekt – auch so ein Anti-Pattern (wie Observer, sollte vielleicht „Obscurer" genannt werden), wenn es zu einer Krabbelkiste von irgendwo einmal benutzten Membern ausartet, die vielleicht

absichtlich oder doch fehlerhaft „null“ sind oder gar nicht benötigt werden. Man soll ein Parameter-Objekt verwenden, wenn die Anzahl der Parameter zu groß wird, aber wie viel ist zu viel?

Magie

Schon mal gesehen? Die Kunden-Nummer „99999“ bedeutet Eigenbedarf, so etwas nennt man eine magische Nummer. Der Autor würde Magie als einen Mechanismus definieren, der im Quellcode nicht auftaucht, aspektorientierte Programmierung (AOP) oder das, was Hibernate mit den Beans macht. So gesehen sind magische Nummern eigentlich nicht magisch; sie werden im Programm explizit behandelt, nur auf dem Transportweg sind sie magisch, da ihre eigentliche Funktion verschleiert bleibt. Früher, als die Programmiersprachen noch keine leistungsfähigen Typ-Systeme hatten und noch Bits gezählt werden mussten, war dies ein übliches Codier-Mittel. Heutzutage haben wir Java und Besseres: Einsparen von Speicher und Rechenleistung ist nicht so wichtig wie Korrektheit. Aber Gewohnheiten sind hartnäckig, so mancher altgedienter technischer Projektleiter setzt solche Lösungen heutzutage noch durch.

Evolution der Typ-Sicherheit

Am Anfang gab es Strings oder „int“-Werte. Im fünften Teil der Java-Geschichte kamen Enums dazu. Schon besser, nun kann man den Typ des Werts per Methoden-Parameter einschränken. Aber wie kann man absichern, dass alle Ausprägungen beachtet werden? Für die vollständige Abdeckung von Enums in Switches bietet Eclipse eine Warnung, aber man schalte mal in einem Projekt die Warnungen ein – 100.000 Warnungen sind keine Seltenheit. Zudem sind Enums Singletons. Unsere Nummern-Liste vom Such-Beispiel können wir dort nicht hineinpacken, eine Member wäre eine einzige globale Variable. In Scala gibt es „sealed case“-Klassen, „option“ und „either“.

In einer Diskussion kam ein Kollege darauf, dass „checked“-Exceptions wie Enums verwendet werden könnten und für jede Ausprägung ein spezieller „catch“-Zweig notiert sein müsste. Dies ist nur eine theoretische Möglichkeit, da diese Exceptions beziehungsweise deren Oberklasse an allen beteiligten Methoden-Signaturen notiert werden müssten und man nicht verhindern kann, dass die Oberklasse der Exceptions gefangen wird. Aber es ist sinnvoll, sich der Übereinstimmungen zwischen Enums und checked-Exceptions bewusst zu sein.

Nun richtig

Statt Enums verwenden wir eine eigene Klasse. In Java benötigt ein eigener Typ stets eine Klasse, eigene Primitive sind nicht möglich. Alle erlaubten Ausprägungen besitzen eine abstrakte Oberklasse, die die Java-Datei als „public“-Klasse auch für unser Konstrukt zur Verfügung stellt. Ein privater Konstruktor der abstrakten Oberklasse begrenzt ihre Ableitungs-Klassen (Ausprägungen) auf Klassen in der entsprechenden Java-Datei, ähnlich zu Scalas „sealed case“-Klassen. Die Ausprägungs-(Implementierungs-)Klassen sind „private“ und so kommt man nicht über „instance of“ und „cast“ an die Information über den tatsächlichen Wert heran.

Zum Erzeugen des jeweils konkreten Suchparameters dienen statische Factory-Methoden in der abstrakten Oberklasse. Zum Auspacken und Behandeln des tatsächlichen Werts steht eine innere, nicht-statische, öffentliche, abstrakte „Either“-Klasse zur Verfügung, die anhand des übergebenen Suchparameter-Objekts instanziiert ist. Durch die zu implementierenden abstrakten Methoden wird über den Compiler erzwungen, jede mögliche Ausprägung des Suchparameters zu behandeln (siehe Listing 1).

Im Gegensatz zu „if else“-Kaskaden beziehungsweise „switch“ wird die Implementierung jedes Zweigs erzwungen. Ändert sich die Ausprägung und damit die „Either“-Klasse, markiert der Compiler alle zu ändernden Code-Stellen als „fehlerhaft“. Auf dem Transportweg bleibt der Wert transparent (magisch), hier wird die abstrakte Oberklasse notiert.

Eine konkrete Klasse je Variante unseres Suchparameters macht die darin ausgedrückte Variante im Gegensatz zu einer magischen Nummer explizit. Die jeweilige Variante bekommt einen Namen, ist beim Debuggen, Loggen und im StackTrace sichtbar. Eine ordentliche (kollegiale) Entwicklerin kann die codierte Absicht mit Javadoc ausführlich dokumentieren. Seltsamerweise ist das Prinzip der „expliziten Codierung“ in den Clean-Code- und OOD-Prinzipien nicht zu finden. Da müsste mal nachgearbeitet werden.

Quellcode-Distanz

Je weiter die Stellen entfernt sind, die den codierten Wert erzeugen und auswerten, desto wichtiger ist eine sichere und dokumentierte Codierung. Was an Ungenauigkeit innerhalb einer Methode oder auf einer Bildschirmseite noch tolerierbar ist, wird bei mehreren Klassen, Packages oder Komponenten zum Problem. Wenn mehrere Personen am Projekt beteiligt sind, ist die Dokumentation solcher Codierungen notwendig. Sehr angenehm, wenn der Compiler dabei mithilft.

Unsere „Either“-Klasse wirkt auf eine mehr oder weniger große Quellcode-Distanz, was Unit-Tests als Alternative zur expli-

```
// Konstruktor für nicht-statische
// innere Klasse
suchParam.new Either<Sql>()
    // konkrete anonyme innere Klasse
{
    public Sql all() {
        ... alle Werte erlaubt ...
    }

    public Sql nothing() {
        ... keine Suche ...
    }

    public Sql values() {
        ... Suche nach Werte-Liste ...
    }
}.result;
```

Listing 1

ziten Codierung per Definition ausschließt. Das Problem der Codierung und deren korrekte Behandlung lassen sich eher durch Interaktions- oder Integrations-Tests abdecken. Bei Interaktions-Tests kommen häufig Mocks zum Einsatz. Letztendlich kann man auf Tests nicht verzichten, aber gerade beim Anwendungsfall für unsere „Either"-Klasse ist die frühe Warnung durch den Compiler bequemer.

Wie bei Interfaces und abstrakten Klassen beschränkt sich der Vertrag auf eine Methoden-Signatur; was im Methoden-Body gemacht wird, kann der Compiler nicht prüfen. Hier sind wieder Unit-Tests angebracht. Böswilligerweise kann man diesen Mechanismus mit Reflection umgehen, was nicht zu vermeiden ist und in einem normalen Projekt sicher niemand tun wird.

Sicherheit durch Unveränderlichkeit

Ein globaler Status kann an allen möglichen Codestellen geändert werden. Dadurch ist das Funktionieren einer Programm-Eigenschaft von vielen Code-Stellen abhängig. Eine Änderung im Code kann ein Problem an einer ganz anderen Stelle erzeugen. Üblicherweise nennt man eine globale Zustandsänderung einen „Seiteneffekt", manche Kollegen benutzen diesen Begriff aber auch für globale Verhaltensänderungen im Code nach einer lokalen Änderung, also einen Seiteneffekt im Code. Der Autor strebt an, veränderlichen globalen Status zu vermeiden (siehe Listing 2).

Der Java-Compiler sichert in diesem Beispiel ab, dass der Such-Parameter genau ein einziges Mal zugewiesen und dann nicht mehr verändert wird. Durch die einmalige und unveränderliche Initialisierung des jeweiligen Werts, die natürlich auch durch Unveränderlichkeit in dessen Klasse abgesichert sein muss, wird die Veränderung des Wertes auf genau diese „if else"-Kaskade eingeschränkt. Eine (wahrscheinlich irrtümliche) Änderung des Wertes ist nicht mehr möglich. Fast jedes Programm mit globalem Status lässt sich in diesem sicheren Stil umbauen. Es gehört einfach zum guten Stil, alle Parameter einer Methode mit dem „final"-Modifier auszustatten.

Ist eine veränderliche temporäre Variable erforderlich, die anhand eines Parameters initialisiert werden muss, kann man den Parameter leicht in diese Variable umkopieren. Auch lokale Variable sollten den „final"-Modifier besitzen und einen möglichst kleinen Sichtbarkeitsbereich haben. Lokale Variable sollten nicht für unterschiedliche Zwecke wiederverwendet werden, sie sind keine begrenzte Ressource.

Kann man seine lokale Variable aufgrund der Unveränderlichkeit nicht wiederverwenden, so muss man sich über einen besseren Namen Gedanken machen und vielleicht drängt sich sofort das Refactoring auf, also das Herausziehen einer eigenen Methode für diese Aufgabe. Unveränderliche lokale Variable machen den Code unabhängig von seiner Reihenfolge. Wenn ein notwendiger Wert nicht vorhanden ist, meldet sich der Compiler.

Optimierung durch Vorberechnung, Wiederverwendung oder Caching ist leicht einzubauen. Veränderliche Werte sind nach Meinung des Autors nur bei Zählern und Akkumulatoren erlaubt, die sowieso einen lokalen Sichtbarkeitsbereich haben. Schade nur, dass es im JRE keine unveränderlichen Collections gibt, eine Alternative sind hier die Collections der Google-Guava-Bibliothek.

Programmier-Automat

Beim Anlegen unserer „Either"-Klasse gibt es eine Menge zu beachten; das ist eine langweilige und fehleranfällige Arbeit. Deshalb hat der Autor dafür einen Code-Generator mit einer „main"-Methode geschrieben [1]. Die Klassen-Namen der abstrakten Oberklasse und der konkreten Ausprägungs-Klassen sind als Strings festgelegt, weil diese Klassen erst durch das Generieren entstehen. Die Member der Ausprägungs-Klassen werden als Name-Klasse-Tupel übergeben, da deren Klassen bereits existieren sollten. Die abstrakte Oberklasse und damit alle konkreten Ausprägungsklassen sowie die Member können mit Typ-Parametern genauer spezifiziert werden.

In manchen Projekten ist das Einchecken generierten Codes in die Versionsverwaltung strikt verboten. Für eventuell zahlreiche „Either"-Klassen Generierungs-Anweisungen im Build zu hinterlegen, ist ein eigenes nicht triviales Verwaltungsproblem. Passende Lösungen hat der Autor nicht parat.

Code-Generator

[1] http://heinerkuecker.de/SimulateOptionEither.html

Heiner Kücker
mail@heinerkuecker.de

```
final SuchParam suchParam;

if ( ...Bedingung1... ) {
    suchParam = ...Wert1...;
}
else if ( ...Bedingung2... ) {
    suchParam = ...Wert2...;
}
else {
    // dieser Zweig darf nie durchlaufen werden
    throw new UnreachableCodeException();
    ...oder...
    suchParam = defaultWert;
}

callSuche( suchParam );
```

Listing 2

Heiner Kücker ist seit dem Jahr 2000 freiberuflicher Java-Entwickler und freut sich, wenn das Java-Typ-System ihn vor seinen Flüchtigkeitsfehlern (macht er ständig) schützt. Zur Rationalisierung seiner Arbeit schreibt er sich selbst kleine Main-Methoden-Tools. Außerdem interessiert er sich für fortschrittliche Programmiertechniken wie funktionale Programmierung.

Connectivity-as-a-Service für Cloud-basierte Datenquellen

Jesse Davis, Progress DataDirect

Mit Verbreitung der Cloud-Technologien hat sich auch die Zahl der Datenquellen rasant vermehrt. Die Vielzahl der Schnittstellen von Point-to-Point-Verbindungen und Versionen macht deren Pflege äußerst aufwändig. Abhilfe schafft Connectivity-as-a-Service, ein standardbasierter Zugang zu heterogenen Datenquellen in der Cloud.

Cloud Computing hat massive Auswirkungen auf die Art und Weise, wie Applikationen entwickelt und betrieben werden. Schätzungen von Gartner zufolge wird sich rund die Hälfte der unabhängigen Softwarehersteller (Independent Software Vendor, ISV) innerhalb der nächsten drei Jahre auf die Entwicklung von Cloud-Anwendungen konzentrieren. Gleichzeitig wird die Zahl sogenannter „Hybrid-Anwendungen" steigen, die sowohl auf im Rechenzentrum vor Ort vorliegende als auch auf in der Cloud befindliche Daten zugreifen. Das heißt, die Variationsbreite von Daten, die in Applikationen zu berücksichtigen ist, nimmt weiter zu.

```
Connection conn = DriverManager.getConnection("<myurl>");
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getColumns(null, "SFORCE", "ACCOUNT", null);
while (rs.next()) {
   System.out.println("COLUMN  NAME = " + rs.getString("COLUMN_NAME"));
   System.out.println("TYPE NAME = " + rs.getString("TYPE_NAME"));
   System.out.println("COLUMN SIZE = " + rs.getInt("COLUMN_SIZE"));
   System.out.println("");
}
Statement stmt = conn.createStatement();
stmt.execute("SELECT ROWID, NAME FROM ACCOUNT");
rs = stmt.getResultSet();
while (rs.next()) {
   System.out.println("ROWID = " + rs.getString(1));
   System.out.println("NAME = " + rs.getString(2));
}
rs.close();
stmt.close();
conn.close();
```

Listing 1: JDBC-Code

Die Vielzahl der Datenquellen ist jedoch nur eine der Dimensionen, mit denen es Entwickler heute zu tun haben. Hinzu kommt die rasante Geschwindigkeit, mit der neue Daten produziert werden, beispielsweise in den vielfältigen Social-Media-Kanälen oder in der Geolokation. Der dritte Faktor ist das gigantische Datenvolumen. Marktforscher schätzen, dass 90 Prozent der heute vorhandenen Daten erst in den letzten beiden Jahren entstanden sind. Zur Charakterisierung des massiven Datenwachstums, der zunehmenden Datenvarietät sowie der hohen Produktionsgeschwindigkeit hat sich im allgemeinen Sprachgebrauch der Begriff „Big Data" durchgesetzt. Diesen strukturellen Rahmen müssen Entwickler heute in ihren Applikationen berücksichtigen.

```
ConnectConfig config = new ConnectorConfig();
config.setUsername(username);
config.setPassword(password);
config.setAuthEndpoint("https://login.salesforce.com/services/SOAP/c/27.0/");
EnterpriseConnection connection = new EnterpriseConnection(config);
DescribeGlobalResult dgr = connection.describeGlobal();
DescribeGlobalObjectResult[] sobjectResult = describeGlobalResult.getSobjects();
int i;
for (i=0; i < sobjectResult.length; i++){
   if (sobjectResult[i].getName().equals("Account"))
       break;
}
Field[] fields = sobjectResult[i].getFields();
for (int j=0; j < fields.length; j++) {
   System.out.println("FIELD NAME = " + field.getName());
   System.out.println("LENGTH = " + field.getLength());
   System.out.println("TYPE = "+ field.getType());
   System.out.println("");
}
QueryResult qr = connection.query("SELECT Id, Name FROM Account");
boolean done = false;
while (!done) {
   Sobject[] records = qr.getRecords();
   for (int k=0; k < records.length; k++ ) {
       Account account = (Account) records[k];
       System.out.println("ID = " + account.getId());
       System.out.println("NAME = " account.getName());
   }
   if (qr.isDone()) {
       done = true;
   }
   else {
       qr = connection.queryMore(qr.getQueryLocator());
   }
}
```

Listing 2: Salesforce-Code

Evolution der Datenbank-Zugriffsmethoden

Der drastische Wandel, der sich hier in den letzten fünf bis zehn Jahren vollzogen hat, lässt sich am besten durch einen kurzen Blick zurück auf die Evolution der Datenbank-Zugriffsmethoden veranschaulichen. Als sich relationale Datenbankmanagement-Systeme im Verlauf der 1990er-Jahre immer stärker in den Unternehmen durchsetzten, nutzten Applikationen zunächst proprietäre Database Client Libraries oder APIs (zum Beispiel SQL*NET für Oracle oder DB Library für Sybase), wie sie die Datenbank-Hersteller zur Kommunikation mit der Datenbank bereitstellten. In einer zweiten Phase wurden die proprietären Schnittstellen weitgehend durch standardbasierte ersetzt. Entwickler waren damit unabhängig von herstellerspezifischen Lösungen und gleichzeitig verbesserte sich die Interoperabilität zwischen unterschiedlichen Datenbanken. Einer der ersten Standards in diesem Umfeld war ODBC, das immer noch eingesetzt wird. Eine hohe Verbreitung haben in der Zwischenzeit ADO.NET und JDBC.

Wie JDBC-Code funktioniert, lässt sich an einem stark vereinfachten Beispiel zei-

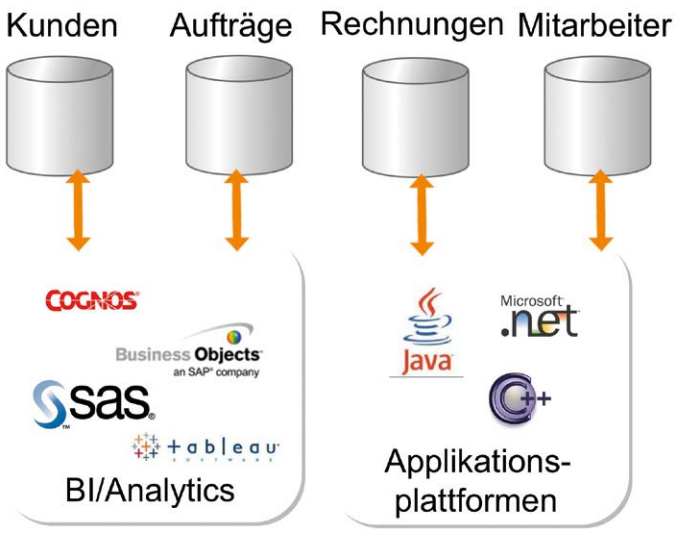


Abbildung 1: Im Datacenter vor Ort sind die zentralen Datenquellen mit standardbasierten Datenbanktreibern ohne großen Aufwand zugänglich

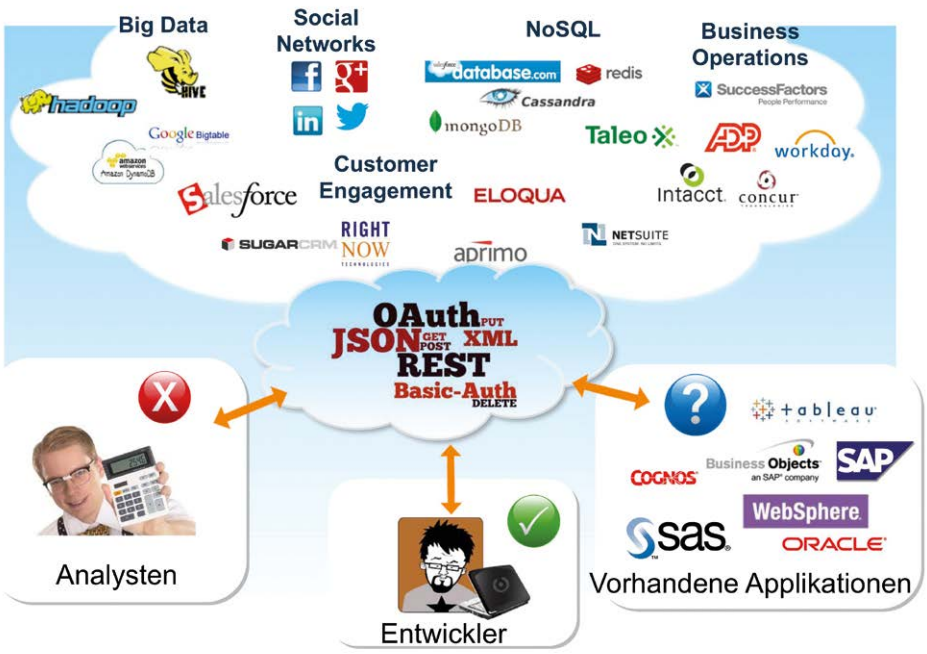


Abbildung 2: Mit Point-to-Point-Verbindungen entsteht eine enorm komplexe Access-Infrastruktur

gen. Der Code ist kurz, knapp, leicht verständlich und ermittelt einige Metadaten sowie Informationen aus einer Rechnungstabelle. Das alles ist reiner, standardbasierter JDBC-Code, der nach einer leichten Modifikation von Table Name etc. auch mit Microsoft Dynamics, einer CRM-Lösung wie RightNow oder einer anderen JDBC-fähigen Datenquelle funktionieren würde (siehe Listing 1).

Deutlich komplizierter wird es bereits beim Datenzugriff auf die Cloud-Lösung salesforce.com mithilfe von deren SOAP-API. Das zeigt das folgende Beispiel. Im Vergleich zum oben stehenden JDBC-Code ist der saleforce.com-Code schwieriger nachzuvollziehen – vor allem für Entwickler, die aus der relationalen Datenbankwelt kommen (siehe Listing 2).

Während sich der JDBC-Code mit einem geringen Aufwand auch für den Zugriff auf andere JDBC-fähige Datenquellen anpassen lässt, gilt das für den Salesforce-Code nicht. Wollte ein Entwickler in seiner Applikation RightNow-Daten verwenden, müsste er neuen Code schreiben. Das gleiche gilt, wenn zusätzlich zu salesforce.com auch Daten aus Microsoft Dynamics gefragt sind. In beiden Fällen wäre ein erheblicher Codierungsaufwand erforderlich. Das demonstriert die Herausforderung, vor der Entwickler heute stehen, wenn sie Applikationen erstellen, die auf eine Vielzahl interner und Cloud-basierter Datenquellen zugreifen sollen.

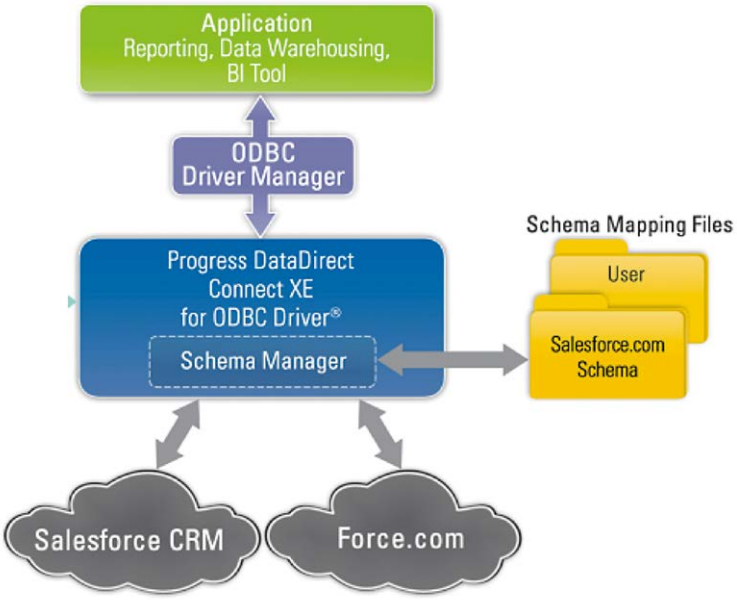


Abbildung 3: Auf einen Blick: Die wichtigsten Komponenten eines SaaS SQL Driver

Verdeutlichen lässt sich das am fiktiven Fall einer europaweit tätigen Fast-Food-Kette, die ein Loyalitätsprogramm zur Kundenbindung starten möchte. Viele grundlegende Daten auf Basis der täglichen Verkaufstransaktionen liegen bereits vor. Sehr nützlich wäre aber die Einbeziehung weiterer Informationsquellen: Von wem stammen die Likes auf der Facebook-Seite des Unternehmens? Welche positiven, negativen oder anderen inhaltlichen Kommentare werden hier oder auf anderen Chat-Plattformen abgegeben? Wer twittert was über das Unternehmen? etc.

Wichtig ist es hier, in einem ersten Schritt die vorhandenen Daten auszuwerten: Welche Produkte werden in welcher Filiale, in welcher Region oder in welchem Land bevorzugt gekauft? Gibt es Unter-

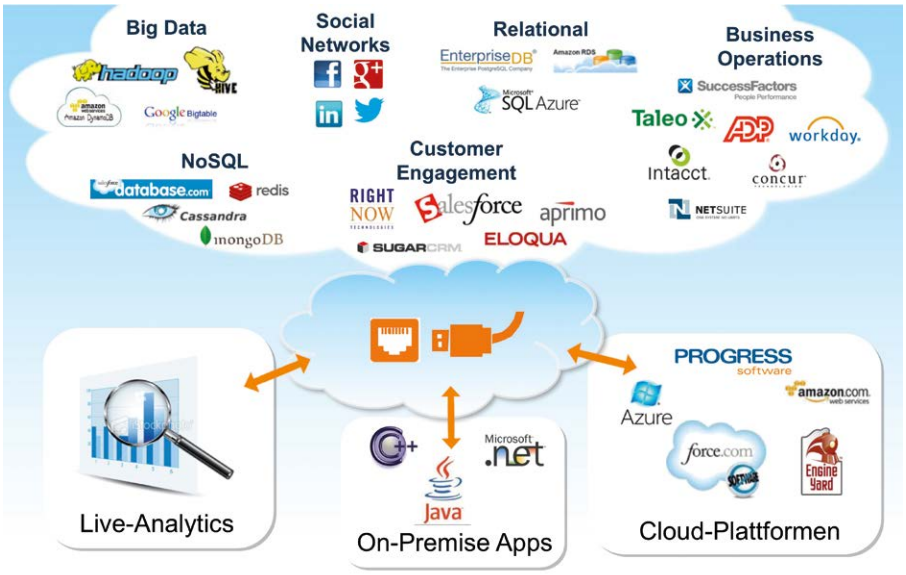


Abbildung 4: Wichtige Einsatzszenarien von Cloud Connectivity Services im Überblick

schiede an einzelnen Wochentagen? Sind saisonale Schwankungen zu beobachten? Interessant wird es bereits dann, wenn es darum geht, aus Daten der Vergangenheit, kombiniert etwa mit Informationen der Kundenkarten, auf künftiges Kaufverhalten zu schließen. Der Zugriff auf die intern in weit verbreiteten Formaten vorliegenden Daten dürfte in aller Regel kein Problem darstellen.

Spannend wird es, wenn darüber hinaus auch Daten aus den unterschiedlichen Social-Media-Kanälen berücksichtigt werden sollen. Mit einer Point-to-Point-Verbindung, bei der für jede Datenquelle deren proprietäre APIs in einer Applikation genutzt werden müssen, lässt sich das Problem kaum lösen, etwa bei Big-Data-Quellen wie dem Hadoop-Distributed-File-System, der Data-Warehouse-Lösung Hive, dem CRM-System salesforce.com oder sugarCRM, Facebook und Twitter sowie NoSQL-Datenbanken wie cassandra und mongoDB etc. Hier gibt es zu viele unterschiedliche APIs und Versionsstände der APIs, die ein Entwickler in einer Applikation berücksichtigen müsste. Zusätzlich verschärft wird die Situation, wenn die externen mit den internen Datenquellen zu kombinieren sind.

SQL-Zugriff auf Cloud-basierte Datenquellen

Auch wenn eine Point-to-Point-Verbindung über eine proprietäre API-Schnittstelle oder einen Webservice zur Verfügung stehen mag: Einen einheitlichen Zugang für zahlreiche unterschiedliche Programmierschnittstellen zu schaffen, ist sehr komplex. Die Vielzahl dieser Schnittstellen und ihrer verschiedenen Versionen macht die Pflege von Anwendungen äußerst aufwändig.

Mit DataDirect Cloud bietet Progress Software eine Lösung für das Connection Management in der Cloud, die auch On-Premise-Apps mit einbezieht. Als Connectivity-as-a-Service stellt der Dienst einen auf Standards basierenden SQL-Zugang zu Daten aus den verschiedenen Quellen bereit. Entwickler müssen dazu keine Schnittstellen anpassen oder neue Libraries hinzuziehen. DataDirect Cloud kann dazu den gleichen, oben erwähnten JDBC-Code nutzen: Das einzige, was sich ändert, ist die URL. Wer JDBC beherrscht, kann damit auch den gesamten Leistungsumfang von DataDirect Cloud nutzen (siehe Listing 1).

Werden die SQL-Queries ausgeführt, verwaltet der Dienst selbstständig die Schnittstellen und deren Versionen. Statt einer Vielzahl verschiedener APIs gibt es dann nur noch einen auf der DataDirect-Technologie basierenden ODBC- und JDBC-Treiber, der für die meisten der weit verbreiteten Applikationen unabhängiger Softwarehersteller passt, etwa für Einsatzgebiete wie Business Intelligence, Enterprise Service Bus, Business Process Management, Data Integration ETL oder Java EE. Mithilfe Browser-basierter Werkzeuge können Entwickler Queries erstellen und aktualisieren, die eine Interaktion mit den unterschiedlichen Datenquellen ermöglichen, ohne dass dafür On-Premise-Software nötig ist. Ein wichtiger Vorteil: Neue Datenquellen können problemlos hinzugefügt werden, ohne dass dazu Änderungen an der Applikation erforderlich sind.

Jesse Davis
jesse.davis@progress.com

Jesse Davis ist Senior Director of Engineering bei Progress DataDirect. Er ist verantwortlich für die Entwicklung neuer Produkte und für zukunftsweisende Forschung. Jesse Davis entwickelt seit mehr als 16 Jahren Lösungen für den Datenzugriff im Unternehmen und ist Mitglied in verschiedenen Expertengremien.

JSF-Anwendungen performant entwickeln

Thomas Asel, Orientation in Objects GmbH

Die Leistungsfähigkeit einer Anwendung lässt sich auf unterschiedliche Weisen beschreiben. Je nach Anwendungsklasse und Szenario existiert eine Vielzahl von Kenngrößen, die den diffusen Begriff „Performance" quantifizierbar machen sollen. Bezogen auf Web-Anwendungen wird mit Performance häufig die vom Benutzer wahrgenommene Reaktionsgeschwindigkeit des Gesamtsystems bezeichnet. Der Artikel zeigt, welche Möglichkeiten speziell den Entwicklern von Java-Server-Faces-Anwendungen (JSF) zur Verfügung stehen, um ein in diesem Sinne performantes Laufzeitverhalten zu erzielen.

Je früher im Software-Entwicklungsprozess Anforderungen adressiert werden, desto günstiger ist die Umsetzung. Die im Folgenden vorgestellten Aspekte sollten daher bei der Neu-Implementierung von JSF-basierten Anwendungen berücksichtigt werden, lassen sich nach Auffassung des Autors jedoch auch mit weitgehend vertretbarem Aufwand nachträglich in bestehende Systeme integrieren. Bevor konkrete Ansatzpunkte besprochen werden, widmen wir uns zunächst den grundlegenden Prinzipien von JSF.

Component Tree und View State

Einzelne Seiten (Views) werden in JSF deklarativ in einem XHTML-Dokument (Facelet) beschrieben. UI-Komponenten sind durch XHTML-Tags repräsentiert und über Tag-Attribute oder verschachtelte Tags weiter parametrisiert. Bei eingehenden Requests wird die baumartige Struktur, die eine View beschreibt, traversiert und Instanzen der jeweiligen JSF-Komponenten erzeugt. Der resultierende Objektgraph heißt „Komponentenbaum" (Component Tree). Er enthält alle Instanzen von JSF-Komponenten auf einer Seite sowie deren Zustand. Im Falle einer komplexen Tabelle zählen beispielsweise die Sortierreihenfolge einzelner Spalten oder der Index der ausgewählten Zeile zum Zustand der Komponente.

JSF verwaltet sowohl den Component Tree (Struktur) als auch den Zustand der Komponenten (State) getrennt voneinander. Struktur des Komponentenbaums und Zustand der Komponenten bezeichnet man als „View State". Da die Struktur des Komponentenbaums für eine View nicht statisch sein muss, ist es notwendig, den View State zu einem späteren Zeitpunkt bei eintreffendem Request wiederherstellen zu können. Dazu hält JSF eine bestimmte Anzahl von View States vor, von denen jeder durch einen Identifier eindeutig referenzierbar ist. Auf diese Weise kann für jeden eingehenden Request bestimmt werden, auf welchen View State er sich bezieht.

Standardmäßig wird der View State serverseitig als Session-Attribut gehalten, was zulasten des verfügbaren Arbeitsspeichers gehen kann. Alternativ kann der View State clientseitig gehalten werden. In diesem Fall wird der Zustand aus jedem eintreffenden Request zunächst de-serialisiert und nach Bearbeitung wieder serialisiert in den Response geschrieben. Serverseitiges State Saving ist somit in erster Linie speicherintensiv, während clientseitiges State Saving mehr Rechenzeit (wegen De-/Seri-

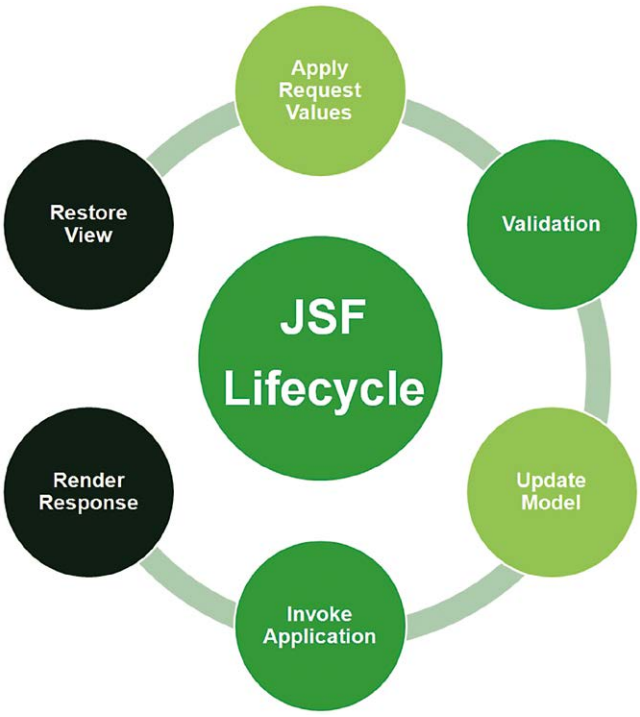


Abbildung 1: Der JSF-Lifecycle

alisierung) und mehr Bandbreitenauslastung (da der Zustand mit jedem Request übertragen wird) zur Folge hat.

Lifecycle

Die Bearbeitung eines Request untergliedert sich in sechs Phasen – der Lebenszyklus („Lifecycle“) einer Anfrage (siehe Abbildung 1):

1. In der ersten Phase („Restore View“) wird ermittelt, welche View angefragt ist. Handelt es sich um die erste Anfrage einer View, wird der Component Tree aus der Struktur des Facelet erzeugt; bei einer erneuten Anfrage einer View durch den gleichen Benutzer wird der existierende View State anhand des State-Identifier wiederhergestellt.
2. In der Phase „Apply Request Values“ werden die übermittelten Werte von Eingabefeldern in die entsprechenden Komponenten übertragen.
3. In der Process-Validation-Phase werden die Eingaben zunächst konvertiert und anschließend, falls nötig, validiert.
4. Wenn die vorangegangene Phase erfolgreich durchlaufen wurde, werden die zwischenzeitlich konvertierten und validierten Eingaben in der Update-Model-Phase schließlich in das Modell übertragen.
5. In der Phase „Invoke Application“ werden der eigentliche Controller-Code ausgeführt und Navigationsziele festgelegt.
6. Schließlich werden in der letzten Phase („Render Response“) der Komponentenbaum durchlaufen und der Response erzeugt.

Mit dieser kurzen Erläuterung von Lifecycle, View State und Component Tree sind die grundlegenden Konzepte vorgestellt, die JSF klar von anderen Web-Frameworks abgrenzen.

Mojarra oder MyFaces?

JSF ist eine Spezifikation, die im Rahmen eines offenen Standardisierungsprozesses entwickelt wird [1]. Um die generelle Anwendbarkeit der Spezifikation zu garantieren, existiert unter dem Projektnamen „Mojarra“ die von Oracle bereitgestellte Referenz-Implementierung. Zusätzlich wird unter dem Dach der Apache Foundation eine freie Implementierung der Spezifikation mit dem Namen „MyFaces“ entwickelt.

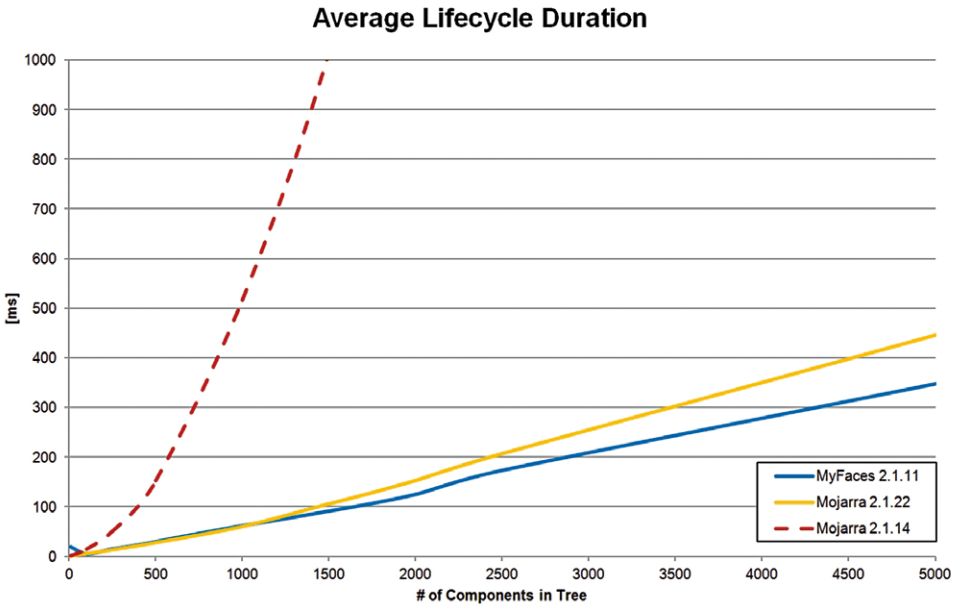


Abbildung 2: Vergleich von JSF-Implementierungen

Sowohl Mojarra als auch MyFaces stellen Implementierungen derselben Spezifikation. Die Implementierungen gelten somit als prinzipiell austauschbar, der Wechsel von einer Implementierung auf die andere gestaltet sich meist erst dann schwierig, wenn die eigene Anwendung Abhängigkeiten zu Implementierungsdetails der verwendeten JSF-Implementierung aufweist.

Auf die Größe kommt es an

Aus der Beschreibung des Lebenszyklus ist ersichtlich, dass für jeden Request der Komponentenbaum entweder erzeugt oder wiederhergestellt und in den folgenden Phasen potenziell mehrfach traversiert werden muss. Somit stellt sich die Frage, welche Auswirkung die Größe eines Komponentenbaums auf die Abarbeitungsdauer eines Request hat. Da beide Implementierungen in ihrer internen Realisierung teilweise erheblich voneinander abweichen, lassen sich auch Unterschiede im Laufzeitverhalten beider Implementierungen ausmachen.

Abbildung 2 zeigt die durchschnittliche Abarbeitungsdauer des Lifecycle von verschiedenen Versionen der JSF-Implementierungen bei unterschiedlich großen Komponentenbäumen. Besonders dramatisch erscheint das Delta zwischen Mojarra 2.1.11 und 2.1.22: Die Referenz-Implementierung litt bis einschließlich Version 2.1.21 unter einer ineffizienten Handhabung des Komponentenbaums, die sich bei steigender Anzahl von Komponenten immer deutlicher bemerkbar machte.

Ein häufiger Fehler mit potenziell negativen Auswirkungen auf die Performance ist der inflationäre Gebrauch von Compo-

Composite Components	
greeting.xhtml (2)	form:foo:greeting form:greeting
foo.xhtml (1)	form:foo
Total number of components:	**3**

Abbildung 3: JSFInspector macht den Komponentenbaum sichtbar

site-Komponenten zur Schaffung wiederverwendbarer Einheiten. Dabei stellt JSF mit benutzerdefinierten Tags, Dekoratoren und Includes die Möglichkeit bereit, Views modular zu gestalten, ohne auf Komponenten zurückgreifen zu müssen. Der Unterschied liegt auf der Hand: Während auch für Composite-Komponenten eine Instanz erzeugt und dem Baum hinzugefügt wird, gilt dies für die alternativen Mechanismen nicht. Der Komponentenbaum bleibt schlank und der Overhead bei der Traversierung des Baums hält sich in Grenzen. [9] behandelt diesen Gesichtspunkt detailliert.

Es lohnt sich, die Größe des Komponentenbaums zur Entwicklungszeit immer im Blick zu haben. Da die Verwendung einer einzelnen Komponente durchaus dazu führen kann, dass weitere Komponenten erzeugt werden und in den Baum wandern, kann nicht zuverlässig von der View-Definition auf die Größe des resultierenden Baums geschlossen werden. Abhilfe schafft hier JSFInspector [8], ein Tool, das die Größe des Komponentenbaums ausgibt und die verwendeten Komponenten sichtbar macht (siehe Abbildung 3).

Daraus ergibt sind folgende Zusammenfassung:

- Hinsichtlich der Performance-Optimierung von JSF-Anwendungen sollte zuallererst die eingesetzte Version betrachtet und gegebenenfalls ein Update durchgeführt werden.
- MyFaces zeigt im direkten Vergleich das bessere Verhalten hinsichtlich Abarbeitungszeit des Lifecycle.
- Die Größe des Komponentenbaums hat direkten Einfluss auf die Performance.
- Es müssen nicht immer Composites sein.

Caching von EL-Ausdrücken

Implementierungsunterschiede zwischen Mojarra und MyFaces lassen sich nicht nur im Detail beobachten: MyFaces verfügt über Funktionalitäten, die über den Umfang der JSF-Spezifikation hinausgehen. Es bietet die Möglichkeit, Expression-Language-Ausdrücke (EL) zu cachen. Facelets enthalten typischerweise eine Vielzahl von EL-Ausdrücken. Für jeden EL-Ausdruck wird eine ValueExpression-Instanz erzeugt und ausgewertet. Dieser Prozess kostet Rechenzeit und Speicher-Kosten, die bei gleichlautenden EL-Ausdrücken eingespart werden können. EL-Expression-Caching lohnt bei Views mit vielen sich wiederholenden EL-Ausdrücken. Weitere Hinweise zur Konfiguration finden sich in [7].

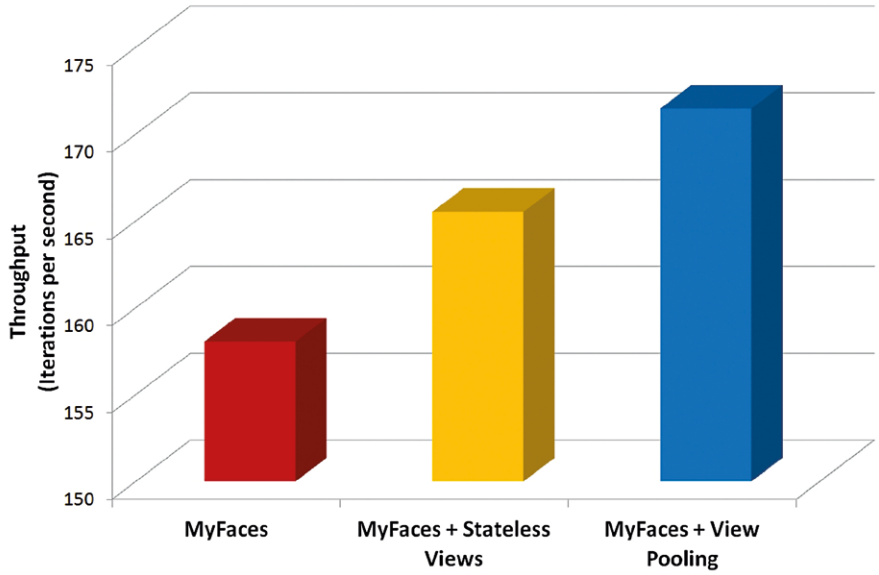


Abbildung 4: View Pooling und Stateless Views im Vergleich

JSF 2.2 bringt Performance-relevante Neuerungen

Mit Stateless Views steht seit JSF 2.2 eine neue Funktionalität zur Verfügung, die sich potenziell positiv auf die Performance von JSF-Anwendungen auswirken kann. Die generelle Funktionsweise des Features besteht darin, dass einzelne Views als transient, also zustandslos („stateless“) markiert werden können. Diese werden bei jeder Anfrage aus der Facelet-Definition neu erzeugt und die Verwaltung des View State entfällt, was einen positiven Effekt auf die Abarbeitungsdauer des Lifecycle haben kann.

Prinzipiell könnte JSF durch Stateless Views für stark verteilte, hochskalierbare und meist Cloud-basierte Szenarien attraktiver werden. Der Nutzen des Features ist jedoch nicht unumstritten. Wie ein Blick auf die MyFaces-Mailingliste zeigt [4], basiert das Komponentenmodell von JSF doch grundlegend darauf, dass Komponenten zustandsbehaftet sind.

Ein Verzicht auf die Zustandsverwaltung hat beinahe zwangsläufig unerwünschte Seiteneffekte bei mehr oder minder stark formularbasierten Views zur Folge. Aus diesem Grund stellt MyFaces ab der Version 2.2.0 einen alternativen, nicht standardisierten Ansatz bereit, das „View Pooling“ [5]. Statt die View bei jedem Request erneut aus der Facelet-Definition zu generieren, basiert View Pooling darauf, erzeugte View-Instanzen in einem Pool vorzuhalten. Der Komponentenbaum muss dadurch nicht immer wieder neu erzeugt werden. Nach vollständiger Abarbeitung des Lifecycle werden der Zustand der View zurückgesetzt und die Instanz an den Pool zurückgegeben.

Sowohl Stateless Views als auch View Pooling stellen kein Allheilmittel dar, im Gegenteil: Es sollte sehr sorgfältig geprüft werden, ob das abweichende Handling des View State zu Problemen auf den jeweiligen Seiten führt. Typische Kandidaten für beide Maßnahmen sind stark frequentierte Seiten mit möglichst einfach strukturierten Formularen wie Login- und Einstiegs-Seiten. Abbildung 4 zeigt eine Vergleichsmessung zwischen beiden Betriebsarten. Die Daten basieren auf einer von MyFaces-Committer Leonardo Uribe durchgeführten Betrachtung [6].

Effizientes Resource-Loading kann entscheidend sein

Neben der rein serverseitigen Bearbeitungsdauer einer Anfrage sind Anzahl und Größe der nachzuladenden Ressourcen von entscheidender Bedeutung. Praktisch jedes an den Client übermittelte

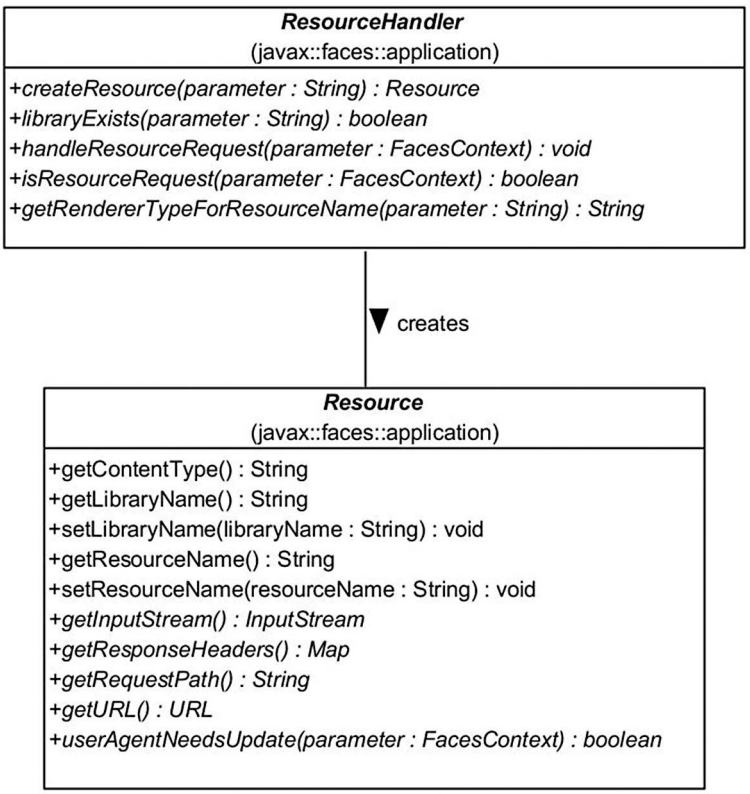


Abbildung 5: Resource und ResourceHandler

Dokument enthält Verweise auf Bilder, Stylesheets und JavaScript-Dateien. Diese werden vom Browser noch während des Renderings der Seite nachgeladen. Während die HTTP-Spezifikation vorschlägt, dass Clients nicht mehr als zwei gleichzeitige Connections zu einer Domain halten sollen, umgehen moderne Browser diese Einschränkung und laden bis zu acht Ressourcen gleichzeitig [10].

Wenn mehr Ressourcen zu laden sind, werden diese sequenziell geladen. Dies führt beim Benutzer – unabhängig von der verfügbaren Bandbreite – zu dem Eindruck, die Verbindung sei langsam. Effizientes Ressource-Handling ist in jeder Web-Anwendung unbedingt erforderlich, um beim Benutzer eine positive Wahrnehmung zu erzielen. Unabhängig von den eingesetzten Technologien kann dies durch grundlegende Maßnahmen gefördert werden:

- Verringerung der Anzahl von zu ladenden Ressourcen
- Caching von Ressourcen
- Auslagern statischer Ressourcen

Im Folgenden wird beschrieben, wie diese Maßnahmen konkret mit JSF umgesetzt werden können.

Weniger ist mehr

Weniger referenzierte Ressourcen bedeuten weniger simultane Connections, geringere benötigte Bandbreite und weniger vom Server zu verarbeitende Requests. Es liegt also nahe, möglichst viele Skripte und Stylesheets zu jeweils einer Ressource zusammenzufassen. Dreh- und Angelpunkt des Resource-Handlings sind die Klassen „Resource“ und „ResourceHandler“. Sie stellen einen Erweiterungspunkt dar, der bei Bedarf durch abgeleitete Klassen angepasst werden kann.

Alternativ kann auf existierende Funktionalität in Bibliotheken zurückgegriffen werden: Die Komponenten-Bibliotheken „RichFaces“ [11] und „ICEFaces“ [12] sowie „OmniFaces“ stellen entsprechende Features bereit. Natürlich gilt auch hier: Einsatz immer mit Augenmaß. Auf vielen Seiten verwendete Skripte wie „jQuery“ sollten nicht mit anderen kombiniert werden, um ein effizientes Caching nicht zu unterlaufen. Speziell OmniFaces bietet eine elegante Lösung, bei der sich auch Ausnahmen sehr einfach konfigurieren lassen. Auf diese Weise lassen sich einzelne Ressourcen von der Zusammenfassung ausschließen (siehe Abbildung 5).

Caching

Um zu entscheiden, wann eine ausgelieferte Ressource nicht mehr gültig ist, wird standardmäßig der Expires-Header auf das aktuelle Datum + 7 Tage gesetzt. Dieses Intervall kann durch einen Kontextparameter (Mojarra „com.sun.faces.defaultResourceMaxAge“, MyFaces „org.apache.myfaces.RESOURCE_MAX_TIME_EXPIRES“) für alle Ressourcen festgelegt werden. Eine Unterscheidung für einzelne Ressourcen ist jedoch nicht möglich. Sollen HTTP-Header auf Ressourcen-Ebene angepasst werden, ist dazu wieder der ResourceHandler anzupassen. OmniFaces bietet auch hierfür eine mögliche Alternative [14].

Statische Ressourcen auslagern

Wie bereits angesprochen, empfiehlt die HTTP-Spezifikation, pro Domain nur zwei Connections zu verwenden. Moderne Browser lassen mehr Verbindungen zu, oftmals sind jedoch mehr Ressourcen simultan zu laden, als der Browser pro Domain erlaubt. Eine naheliegende Lösung besteht daraus, die Ressourcen auf mehrere (Sub-) Domains zu verteilen. Für statische Ressourcen könnten eigene Subdomains eingerichtet sein, etwa „images.example.com“, „js.example.com“ und „css.example.com“. Pro Subdomain können jetzt mehrere Ressourcen parallel bezogen werden.

Darüber hinaus bietet es sich an, statische Ressourcen generell von einem anderen Server zu beziehen – auf diese Weise muss der Application-Server beziehungsweise Servlet-Container sich nur um Anfragen von JSF-Seiten kümmern. Die statischen Ressourcen selbst können von einem reinen Web-Server ausgeliefert werden. Statt eigene Server für statische Ressourcen zu betreiben, empfiehlt sich die Verwendung eines Content-Delivery-Networks (CDN). In diesem Fall werden die statischen Ressourcen komplett ausgelagert. Ein CDN besteht aus mehreren geografisch verteilten Servern, die Ressourcen bereitstellen. Das CDN übernimmt Aufgaben wie Skalierung, Caching und

Verteilung der Ressourcen selbstständig. Durch leistungsfähige Anbindungen und die geographische Verteilung bieten CDN meist bessere Antwortzeiten für die Auslieferung statischer Ressourcen als die eigenen Application-Server. Werden diese inhouse betrieben, fällt außerdem der Bandbreitenbedarf für die Auslieferung der Ressourcen weg, was zusätzlich die eigene Anbindung entlastet.

Sollen statische Ressourcen einer JSF-Anwendung ausgelagert werden, muss wieder in den Ressource-Handling-Mechanismus eingegriffen werden. Natürlich ist es wünschenswert, das gewohnte Programmiermodell beibehalten zu können und Ressourcen über JSF-Komponenten wie „<h:graphicImage>“ oder „<h:ouputStylesheet />“ einzubinden, unabhängig davon, von wo diese ausgeliefert werden. Außerdem muss beim Deployment der Anwendung darauf geachtet werden, statische Ressourcen auf die dafür vorgesehenen Webserver oder das CDN zu kopieren. Letzteres kann unabhängig von JSF über Build-Tools wie Maven oder Ant erfolgen. Allerdings stellt sich die Frage, wie JSF-Komponenten wie „<h:graphicImage />“ korrekte Links auf Ressourcen in entfernten Systemen setzen können. Auch dafür bietet OmniFaces mit „CDNResourceHandler“ eine leicht zu konfigurierende Lösung an [15].

Fazit

Die Performance von JSF-basierten Anwendungen lässt sich durch eine Vielzahl von Maßnahmen positiv beeinflussen. Die vorgestellten Techniken und Methoden können relativ leicht mit Bordmitteln oder durch den Einsatz von Bibliotheken umgesetzt werden. Insbesondere OmniFaces ist zu einem wertvollen Werkzeug für den fortgeschrittenen JSF-Entwickler avanciert. Die vorgestellten Methoden sollten als Vorschläge betrachtet werden – eine universelle Lösung für Performance-Probleme stellen sie freilich nicht dar. Wie immer gilt: Regelmäßige Analyse des Verhaltens unter Last bereits zur Entwicklungszeit bewahrt vor bösen Überraschungen im späteren Produktivbetrieb.

Links

[1] https://jcp.org/en/jsr/detail?id=344
[2] https://javaserverfaces.java.net
[3] http://myfaces.apache.org
[4] http://bit.ly/stateless-views-in-myfaces
[5] http://bit.ly/myfaces-view-pooling
[6] http://bit.ly/myfaces-view-pooling-analysis
[7] https://cwiki.apache.org/confluence/display/MYFACES/Cache+EL+Expressions
[8] http://tasel.github.io/jsfinspector
[9] http://bit.ly/jsf-alternatives-to-composites
[10] http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections
[11] http://bit.ly/richfaces-resource-optimization
[12] http://www.icesoft.org/blog/?p=864
[13] http://bit.ly/omnifaces-combined-resource-handler
[14] http://showcase.omnifaces.org/filters/CacheControlFilter
[15] http://showcase.omnifaces.org/resourcehandlers/CDNResourceHandler

Thomas Asel
thomas.asel@oio.de

Thomas Asel ist Trainer, Berater und Coach bei der Orientation in Objects GmbH. Schwerpunktmäßig beschäftigt er sich mit Architektur und Optimierung von webbasierten Systemen.

Pimp my Jenkins

Sebastian Laag, adesso AG

Continuous-Integration-Server werden heutzutage von vielen Projekten genutzt, um Qualitätsstandards der eigenen Software dauerhaft sicherzustellen. Jenkins [1] ist eines der populärsten Produkte in diesem Bereich.

Jenkins kann um mehrere Hundert Plug-ins erweitert werden [2], um den meisten Anforderungen eines kontinuierlichen Build-Prozesses gerecht zu werden. Fast alle Erweiterungen stehen frei zur Verfügung. Die meisten sind Open Source, sodass eine Mitarbeit oder eigene Weiterentwicklung möglich ist. In den Kategorien „Benutzeroberfläche“, „Benachrichtigungen“, „Build-Trigger“ oder „Integration externer Sites“ stehen Erweiterungen zur Verfügung. Je nach Projekt sind zum Teil unterschiedliche Plug-ins erforderlich, doch einige der hier vorgestellten sollte jeder Administrator einer Jenkins-Umgebung verwenden.

Ausgehend von den Erfahrungen des Autors bei der Wartung und Neuinstallation verschiedener Jenkins-Server stellt dieser Artikel nützliche Plug-ins zur Produktivitätssteigerung und vereinfachten Wartung vor. Dies ist der erste Artikel einer Serie über interessante Jenkins-Plug-ins. Mit einigen der Erweiterungen lässt sich auch viel Zeit bei der Benutzung und Administration von Jenkins einsparen. Die Reihenfolge der nachfolgend beschriebenen Erweiterungen ist rein zufällig und stellt keinerlei Wertung dar.

Project name	Jobs	Builds all	Builds locked	Workspace
ChildChildJobA	130 KB	116 KB	7 KB	-
ChildJobA	63 KB	53 KB	-	-
ParentJob	43 KB	34 KB	-	-
ChildJobB	28 KB	18 KB	-	-
MultiJob	-	-	-	-
Total	**265 KB**	**222 KB**	**7 KB**	**-**

Abbildung 1: Speicherverbrauch aller Jobs

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <project>
3   <actions/>
4   <description>
5 +-</description>
6   <keepDependencies>false</keepDependencies>
7   <properties>
8     <jenkins.advancedqueue.AdvancedQueueSorterJobProperty
plugin="PrioritySorter@2.6">
```

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <project>
3   <actions/>
4   <description>Das ist ein Beispiel für die
JobConfigHistory</description>
5   <keepDependencies>false</keepDependencies>
6   <properties>
7     <jenkins.advancedqueue.AdvancedQueueSorterJobProperty
plugin="PrioritySorter@2.6">
```

Restore old version

Abbildung 2: Anzeige einer veränderten Konfiguration mit dem JobConfigHistory-Plug-in

Disk Usage

Je nach verwendetem Rechner oder virtueller Maschine (VM) kann Speicherplatz sehr schnell knapp werden. Das Plug-in [3] bietet eine gute Möglichkeit, speicherintensive Jobs zu erkennen und entsprechend reagieren zu können. Es zeigt für jeden Job den Speicherverbrauch von Workspace und von gesammelten Builds separat an (siehe Abbildung 1). Zudem existiert eine Job-Übersicht, sodass schnell klar wird, welcher Job viele Artefakte in seinem Arbeitsbereich ablegt. Eine weitere Möglichkeit ist die Speicherung zu vieler Builds, da diese nicht automatisch ab einer gewissen Anzahl oder nach einer bestimmten Zeit entfernt wurden (Log-Rotate).

Job Config History

Bei vielen Jenkins-Installationen ist es allen Benutzern möglich, Jobs zu konfigurieren. So kann es je nach Know-how des Anwenders schnell zu einer fehlerhaften Konfiguration kommen. Das Plug-in [4] zeigt jede Veränderung eines Jobs direkt neben dem jeweiligen Folge-Build in einer Übersicht an (siehe Abbildung 2). So kann bei einem fehlgeschlagenen Build schnell erkannt werden, ob dieser mit einer Veränderung der Konfiguration zusammenhängt. So lässt sich bei der Fehleranalyse eines Builds unter Umständen viel Zeit sparen.

Green Balls

Standardmäßig bietet Jenkins nur die Anzeige des Status von Builds in den Farben blau (erfolgreich), gelb (Unit-Tests fehlgeschlagen) und rot (Fehler im Build) an. Der Begründer von Jenkins, Kohsuke Kawaguchi, ist Japaner und dort wird die Farbe blau mit etwas Positivem verbunden. Zudem war das heute grüne Ampelsignal in Japan in früheren Jahren noch blau, bis festgestellt wurde, dass die Farbe aus der Ferne schlecht erkennbar ist. Aufgrund dieser Umstände ist ein erfolgreicher Build standardmäßig blau [5]. Das Plug-in [6] verändert die Anzeige der erfolgreichen Builds nach Grün. Wer also einen erfolgreichen Build mit der Farbe Grün verbindet, sollte dieses Plug-in verwenden.

Abbildung 3: Konfigurationsmöglichkeiten des Copy-Artifact-Plug-ins

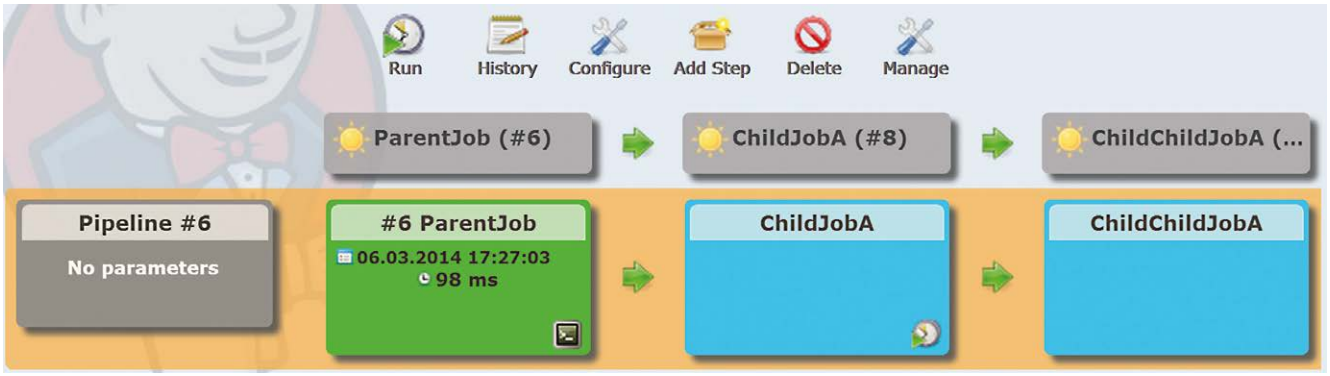


Abbildung 4: Ansicht einer Build-Pipeline mittels Plug-in

Copy Artifact

Zur übersichtlichen Gestaltung der Jobs ist man häufig gezwungen, Abhängigkeiten zwischen ihnen zu erstellen. Sollen dann auch noch Artefakte zwischen den Builds ausgetauscht werden, kann das Plug-in [7] sehr nützlich sein. Es wird als einzelner Build-Schritt in einen Job integriert. In der Konfiguration wird der vorgelagerte Job angegeben, von dem ein Artefakt (etwa eine TXT-Datei) benötigt wird (siehe Abbildung 3). Mittels regulären Ausdrucks werden die zu kopierenden Artefakte und das jeweilige Zielverzeichnis angegeben. Während der Job-Ausführung werden die Dateien in den Workspace kopiert und stehen danach wie gewünscht zur Verfügung.

Die Verwendung des Plug-ins sollte nur dann erfolgen, wenn es bei den kopierten Artefakten nicht sinnvoll ist, diese auf einem zentralen Server wie beispielsweise Nexus einzusetzen. Bei Nexus kann dieses direkt mit dem integrierten Maven-Plug-in und einem eigenen Build-Schritt erfolgen. Wenn andere Artefakte zentral in Jenkins verwaltet werden sollen, kann das Artifact-Deployer-Plug-in [8] eingesetzt werden.

Build Pipeline

Nach der Installation des Plug-ins [9] kann man eigene Views zur Visualisierung der vor- und nachgelagerten Jobs eines ausgewählten Jobs erstellen (siehe Abbildung 4). Zusätzlich wird eine konfigurierbare Anzahl von Build-Ergebnissen angezeigt. Je nach Einstellung ist eine manuelle Aus-

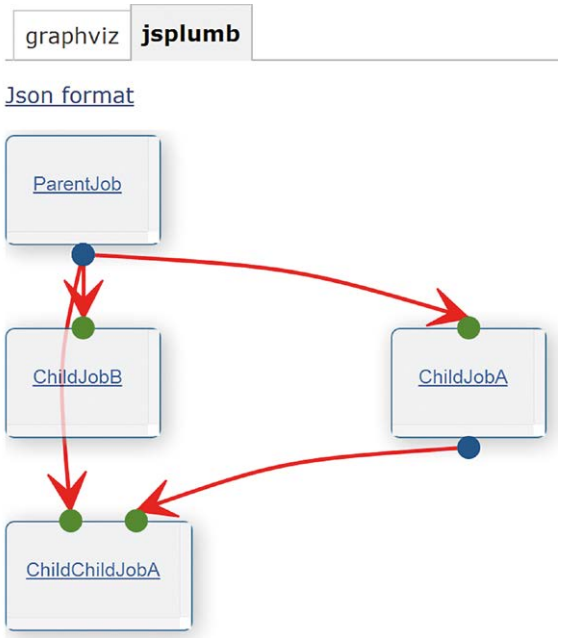


Abbildung 5: Grafische Ansicht der Job-Abhängigkeiten

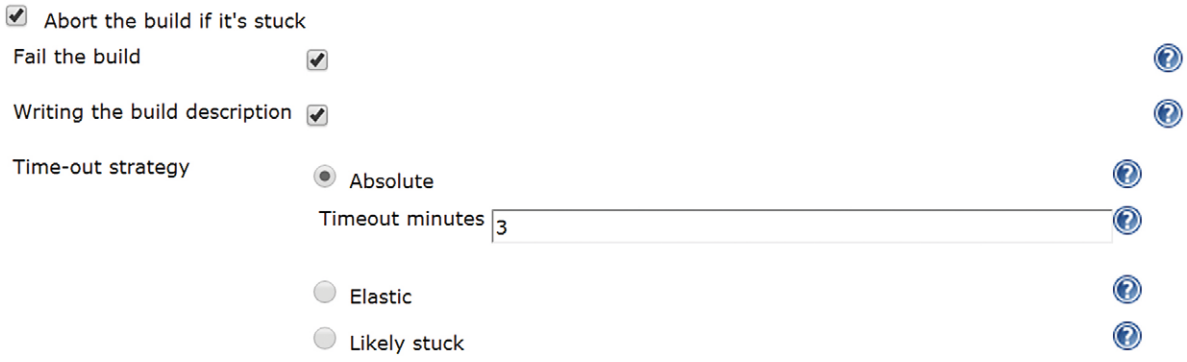


Abbildung 6: Konfiguration des Build-Timeout-Plug-ins für einen Job

führung der einzelnen Schritte einer Build Pipeline möglich.

Das Plug-in eignet sich hervorragend zum Erstellen und Verwalten komplexer Build-Pipelines mit manuellen Zwischenschritten, sodass zum Beispiel ein Deployment, angefangen vom Build über das Ausbringen und Testen in der Testumgebung bis zur Produktion, ausgeliefert werden kann.

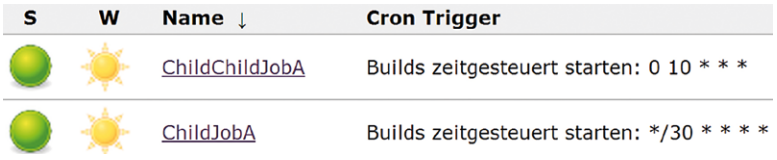


Abbildung 7: Zusätzliche Spalte für die Anzeige des Cron Triggers

Dependency Graph View

Das Plug-in [10] bietet eine grafische Übersicht inklusive aller Abhängigkeiten der Jobs, die in der Jenkins-Installation vorhanden sind (siehe Abbildung 5). Die grafische Ansicht eignet sich für die Analyse der eigenen Jobs, um etwa Zyklen in der Aufruf-Struktur oder unnötig komplexe Builds aufzudecken. Das Plug-in verfügt über das „graphviz"- und das „jsplumb"-Format sowie über eine JSON-Repräsentation zur Anzeige. Je nach Anzahl der Jobs kann die Dauer bis zur Anzeige der Übersicht mehrere Minuten betragen.

Configuration Slicing

Je nach Anzahl der Jobs in einer Jenkins-Installation kann eine Veränderung der Konfiguration mehrerer Jobs zur gleichen Zeit mit viel Aufwand verbunden sein. Mit dem Plug-in [11] können Parameter wie verwendetes JDK, genutzter Slave, Log-Rotate-Einstellung, die Maven-Version und einiges mehr für die gewünschten Jobs gleichzeitig angepasst werden. Viele Plug-ins bringen die passende Erweiterung für das Configuration Slicing direkt mit, sodass per Plug-in hinzugefügte Parameter ebenfalls für mehrere Jobs gleichzeitig angepasst werden können.

Locks and Latches

Wenn mehrere Jobs gleichzeitig auf eine bestimmte Ressource zugreifen möchten, kann es zu ungewünschtem Verhalten beim Ausführen von Builds oder Tests kommen, etwa wenn ein Build in eine Datenbank-Tabelle schreibt, während ein Test diese gerade ausliest. Um dies zu vermeiden, lassen sich innerhalb der Jenkins-Installation Locks festlegen und den jeweiligen Jobs zuordnen. Bevor ein Job ausgeführt wird, prüft das Plug-in [12], ob der zugeordnete Lock gerade frei ist. Ist dies der Fall, kann die Ausführung beginnen, ansonsten muss weiter gewartet werden. Leider wird das Plug-in seit dem Jahr 2010 nicht mehr weiterentwickelt. Das Throttle-Concurrent-Builds-Plug-in stellt eine Alternative dar, die jedoch umständlicher zu verwenden ist.

Build Timeout

Das Plug-in [13] ermöglicht die Angabe eines Timeouts für einen bestimmten Build. Beim Überschreiten des angegebenen Werts bricht der Job automatisch ab. Die absolute Zeitangabe ist allerdings nur eine von drei verschiedenen Timeout-Strategien (siehe Abbildung 6). Mit der Strategie „Elastic" lässt sich die prozentuale Zeitüberschreitung über die letzten Builds angeben, nach der ein Job abgebrochen wird. Als dritte Möglichkeit kommt ein heuristischer Ansatz zur Erkennung eines festhängenden Builds zum Einsatz. Diesen stellt Jenkins selbst bereit und das Plug-in nutzt ihn. Die drei Alternativen begrenzen die Laufzeit einzelner Jobs oder regulieren problematische Builds, die beispielsweise häufig stehenbleiben und einen Ausführungsplatz (Executor) des Jenkins-Servers blockieren.

Cron Column

Um eine Übersicht über Ausführungszeiten der eigenen Jobs zu bekommen, kann das Plug-in [14] eingesetzt werden. Es er-

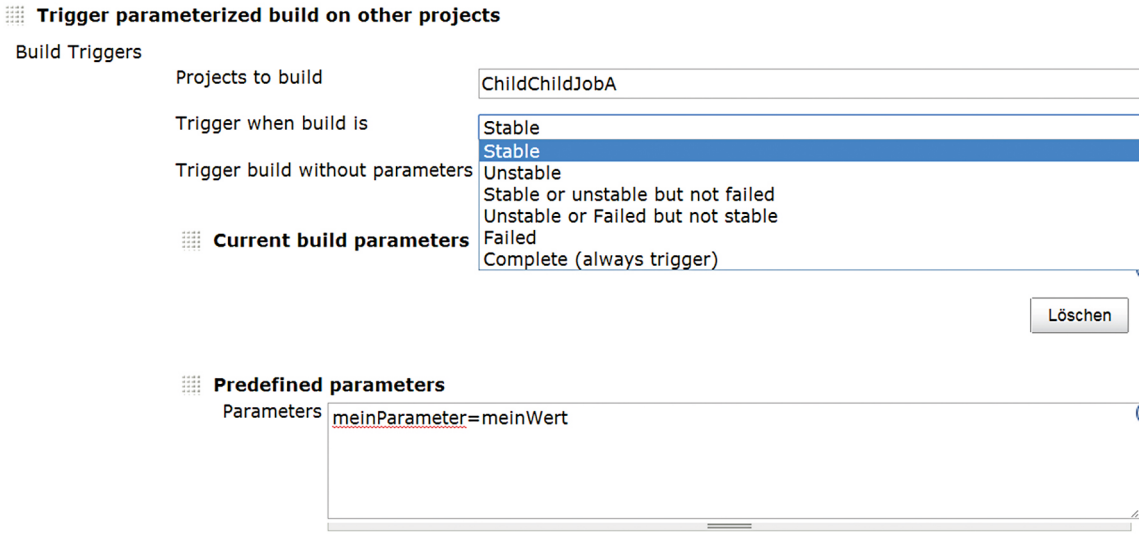


Abbildung 8: Konfigurationsmöglichkeiten des Parametrized-Trigger-Plug-ins

weitert die Standard-Views von Jenkins um die Anzeige der Startzeit jedes einzelnen Builds (siehe Abbildung 7). So lässt sich auf einen Blick feststellen, welche Jobs ungewünscht häufig, zu selten oder gar nicht ausgeführt werden.

Parametrized Trigger

Sollen mehrere Jobs nacheinander ausgeführt werden, kann es oftmals notwendig sein, Parameter zu übergeben. Nach der Installation des Plug-ins [15] lassen sich nachfolgende Jobs mit verschiedenen Parametern nachgelagert aufrufen. Der Aufruf kann dazu noch an den Status („erfolgreich", „instabil", „fehlerhaft" etc.) des aktuellen Jobs gebunden sein. Auch die übergebenen Parameter lassen sich verschiedenartig einrichten. Dabei können einfache boolesche Parameter, Parameter aus einer Properties-Datei oder auch die aktuelle SVN-Revision übergeben werden (siehe Abbildung 8). Der nachfolgende Job kann diese Parameter dann dementsprechend verarbeiten oder ebenfalls weiterreichen.

Multi Job

Mit dem Plug-in [16] können komplexe Build-Pipelines konfiguriert und ausgeführt werden. Dabei lassen sich Jobs in einzelne Schritte gruppieren, für eine parallele oder serielle Ausführung einrichten und nacheinander abarbeiten. Je nach Ergebnis einer einzelnen Phase kann der komplette Multijob abgebrochen oder einfach weiter ausgeführt werden (siehe Abbildung 9). Zudem kann man je nach Ergebnis eines einzelnen Jobs die weitere Abarbeitung der zugehörigen Phase unterbinden und direkt die nächste Phase ausführen.

Um einen Multijob einzurichten, ist ein neuer Job vom Typ „Multijob" erforderlich.

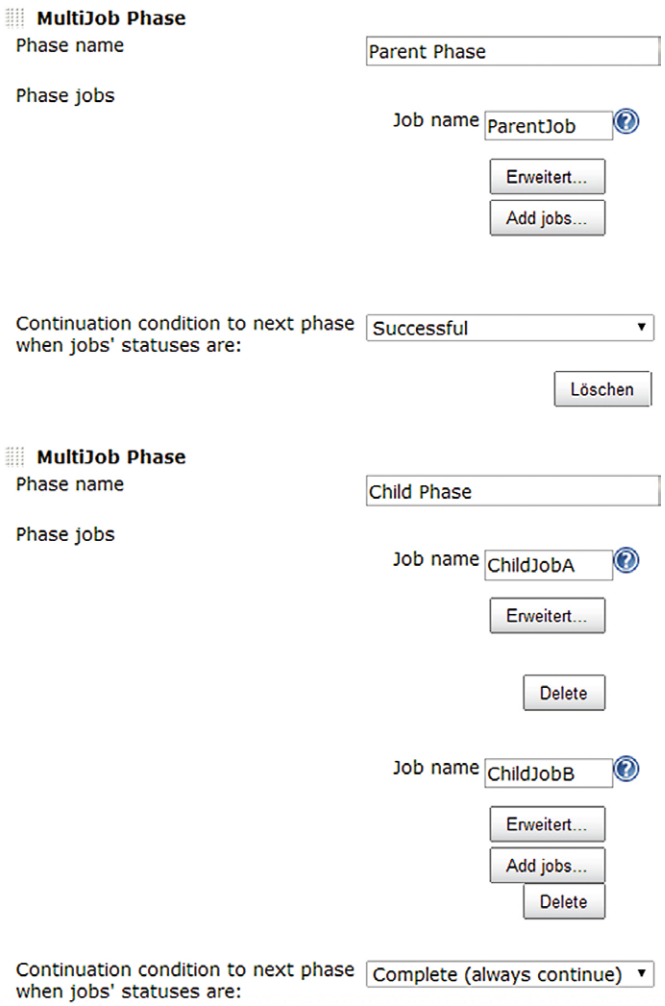


Abbildung 9: Konfiguration verschiedener Phasen mittels Multi-Job-Plug-in

Die einzelnen Phasen werden dann als einzelne Build-Schritte der Art „Multijob-Phase" hinzugefügt. Im Gegensatz zum Build-Pipeline-Plug-in lassen sich hier keine manuellen Schritte konfigurieren. Zudem ist die Ausführung einzeln ausgewählter Phasen nicht möglich.

Priority Sorter

Mit dem Plug-in [17] ist es möglich, Jobs zu priorisieren. So können diese sich in der Build-Warteschlange vorbeischieben und werden bevorzugt ausgeführt. Die Anzahl zur Verfügung stehenden Prioritäten und die Standard-Priorität lassen sich in der Jenkins-Verwaltung einrichten. Soll ein Job bevorzugt oder benachteiligt zu allen anderen ausgeführt werden, so ist dies Job-spezifisch zu konfigurieren.

DropDown ViewsTabBar

Um Jobs zu gruppieren, kommen bei Jenkins Ansichten („Views") zur Anwendung. Diese können zum Beispiel durch einen regulären Ausdruck anhand der Job-Namen erstellt werden. Ab einer gewissen Anzahl von Views kann die Übersichtlichkeit der Anzeige leiden, da Jenkins alle Views in Form von Tabs nebeneinander in einer Reihe anzeigt. Zum Wechsel der View im Browser muss dann horizontal gescrollt werden.

Das Plug-in [18] ermöglicht die Anzeige der Views in einer Dropdown-Liste. Somit bleibt die Übersichtlichkeit erhalten. Die Jobs lassen sich nun bequem in der Liste ohne störendes horizontales Scrollen auswählen. Bei einer großen Anzahl von Views kann allerdings auch die Dropdown-Liste sehr unübersichtlich werden.

Fazit

Unzählige Plug-ins erweitern die Funktionalität von Jenkins. Sie werden stetig weiterentwickelt und mit neuen Features angeboten. Dabei gibt es Erweiterungen für die verschiedensten Themengebiete. Die Erfahrung des Autors hat gezeigt, dass der Jenkins-Server durch die Installation vieler Plug-ins langsamer beim Start der Applikation wird. Zudem leiden auch die Antwortzeiten im Web-Browser bei der Navigation durch die einzelnen Jobs. Also sollte man nur wirklich notwendige Plug-ins installieren. Um eine Erweiterung zu überprüfen, kann die Einrichtung eines Test-Jenkins sinnvoll sein.

Einige der Plug-ins haben Abhängigkeiten zu anderen Erweiterungen, sodass durch eine Aktualisierung Inkompatibilitäten entstehen können. Deshalb sollten Updates hauptsächlich bei bekannten Problemen oder sicherheitskritischen Aktualisierungen vorgenommen werden. Die Wiki-Seiten des jeweiligen Plug-ins zeigen zumeist die Änderungen, die mit jeder Version vorgenommen wurden.

Die in diesem Artikel genannten Plug-ins sind nur einige von sehr vielen sinnvollen und beliebten Erweiterungen für den vermutlich am weitesten verbreiteten Build-Server. Ohne die vielen Plug-ins der Community wäre der Funktionsumfang von Jenkins eingeschränkt.

Im nächsten Artikel werden weitere nützliche Erweiterungen wie das Artifact-Deployer-Plug-in, Plug-ins zur Unterstützung anderer JVM-Sprachen wie Groovy oder Ruby und auch das Sonar- und das Gradle-Plug-in vorgestellt.

Weitere Informationen

[1] http://jenkins-ci.org
[2] https://wiki.jenkins-ci.org/display/jenkins/Plugins
[3] https://wiki.jenkins-ci.org/display/jenkins/Disk+Usage+Plugin
[4] https://wiki.jenkins-ci.org/display/jenkins/JobConfigHistory+Plugin
[5] https://issues.jenkins-ci.org/browse/ jenkins-ocusedCommentId=120769&page=com.atlassian.jira.Plugin.system.issuetabpanels:comment-tabpanel#comment-120769
[6] https://wiki.jenkins-ci.org/display/jenkins/Green+Balls
[7] https://wiki.jenkins-ci.org/display/jenkins/Copy+Artifact+Plugin
[8] https://wiki.jenkins-ci.org/display/jenkins/ArtifactDeployer+Plugin
[9] https://wiki.jenkins-ci.org/display/jenkins/Build+Pipeline+Plugin
[10] https://wiki.jenkins-ci.org/display/jenkins/Dependency+Graph+View+Plugin
[11] https://wiki.jenkins-ci.org/display/jenkins/Configuration+Slicing+Plugin
[12] https://wiki.jenkins-ci.org/display/jenkins/Locks+and+Latches+Plugin
[13] https://wiki.jenkins-ci.org/display/jenkins/Build-timeout+Plugin
[14] https://wiki.jenkins-ci.org/display/jenkins/Cron+Column+Plugin
[15] https://wiki.jenkins-ci.org/display/jenkins/Parameterized+Trigger+Plugin
[16] https://wiki.jenkins-ci.org/display/jenkins/Multijob+Plugin
[17] https://wiki.jenkins-ci.org/display/jenkins/Priority+Sorter+Plugin
[18] https://wiki.jenkins-ci.org/display/jenkins/DropDown+ViewsTabBar+Plugin

Sebastian Laag
sebastian.laag@adesso.de

Sebastian Laag (Dipl. Inf., Univ.) ist als Senior Software-Engineer bei der adesso AG in Dortmund tätig und arbeitet dort als Entwickler an Java-basierten Web-Anwendungen.

Contexts und Dependency Injection – Geschichte und Konzepte

Dirk Mahler, buschmais GbR

Die ersten beiden Teile dieser Reihe über Contexts und Dependency Injection (CDI) haben die Geschichte des Standards und seine grundlegenden Konzepte beleuchtet. Im Folgenden sollen weitere Aspekte aufgegriffen und vertieft werden: Wie sich Querschnittsaspekte abbilden lassen und wie die Unterstützung für die serverseitige Anwendung des Observer-Patterns aussieht.

Der historische Abriss des ersten Teils zeigt im Abschnitt „Industrialisierung: Fabriken", dass eine Factory beziehungsweise ein Container anstelle einer angeforderten Instanz Stellvertreter (Proxys) ausliefern kann, mit deren Hilfe sogenannte „Querschnittsaspekte" (Cross Cutting Concerns) abgebildet werden können. Dies funktioniert über das Abfangen von Methoden-Aufrufen und die Anreicherung zusätzlicher Logik.

Im zweiten Teil wurde die Abgrenzung zwischen CDI-Beans und Enterprise Java Beans (EJBs) so beschrieben, dass Letztere ebenfalls als normale Beans aufgefasst werden können, die aber zusätzliche Funktionalitäten zur Verfügung stellen. Bei genauerer Betrachtung fällt auf, dass es sich dabei um die Realisierung beziehungsweise das Management von Querschnittsaspekten handelt, also:

- *Transaktionen*
  Beginn und Commit/Rollback vor beziehungsweise nach dem Aufruf einer Methode
- *Security*
  Überprüfung der Authentifizierung/Autorisierung eines Nutzers bzw. seiner Rollen vor dem Aufruf einer Methode
- *Concurrency*
  Synchronisierung konkurrierender Zu-

```
@Interceptor
@Trace
public class TracingInterceptor {

   private static final Logger LOGGER = …
   @AroundInvoke
   public Object invoke(InvocationContext context) throws Exception {
       Method method = context.getMethod();
   LOGGER.debug("Entering " + method.toString());
       try {
           return context.proceed();
       } finally {
           LOGGER.debug("Entering " + method.toString());
   }
}
```

Listing 1

griffe durch verschiedene Threads auf zustandsbehaftete EJBs mittels Read- beziehungsweise Write-Locks

Die genannten Aspekte haben folgende gemeinsame Eigenschaften:

- Sie sollen unabhängig vom Interface der EJB auf alle aufrufbaren Methoden einer Bean angewendet werden können
- Die Ausführung lässt sich durch Annotationen an den EJBs beziehungsweise deren Methoden steuern

Das Prinzip des Abfangens von Methoden-Aufrufen steht dabei nicht nur dem Container selbst zur Verfügung, auch eine Anwendung, die im Container läuft, kann von diesem Programmiermodell profitieren. Dies wurde bereits mit Java EE 5 ermöglicht und ein paar Jahre später durch CDI etwas verfeinert.

Abgefangen: Interceptor

Ein typischer Querschnittsaspekt ist das Tracing von Methodenaufrufen, also das Logging von Ein- und Austritt. Die Realisierung erfolgt durch einen „Interceptor", der exemplarisch in Listing 1 umgesetzt ist.

Es fällt auf, dass die Klasse kein Interface implementiert, anstelle dessen aber zwei Annotationen trägt: „@Interceptor" und „@Trace". Erstere markiert die Klasse als Interceptor-Implementierung, der Container erwartet daraufhin das Vorhandensein einer mit „@AroundInvoke" annotierten Methode mit der gegebenen Signatur. Diese setzt die gewünschte Funktionalität um, dabei kann sie über die „InvocationContext"-Instanz weitere Informationen über die Ursprungs-Instanz, Parameter und die ursprünglich aufgerufene Methode beziehen, also gegebenenfalls vorhandene Annotationen auswerten. Dies ist im Beispiel nicht realisiert, stattdessen wird grundsätzlich zunächst der Eintritt geloggt sowie mittels „context.proceed()" der Aufruf an die ursprüngliche Methode delegiert, um anschließend im „finally"-Block den Austritt über den Logger auszugeben.

Genaugenommen ist die Beschreibung noch nicht ganz korrekt: Oftmals handelt es sich nicht nur um einen, sondern um eine Kette von Interceptoren, die durchlaufen werden, bevor die eigentliche Bean-Methode erreicht wird beziehungsweise nachdem sie verlassen wurde. Es gibt darüber hinaus weitere Eigenschaften, die nicht verschwiegen werden sollen: So ist eine Interceptor-Instanz immer an eine Bean-Instanz gebunden, der Interceptor übernimmt also den Lebenszyklus dieser Bean. Darüber hinaus kann „@Inject" verwendet werden, um Zugriff auf von ihm benötigte Dienste zu erhalten (siehe Listing 2).

Die Bedeutung der Annotation „@Trace" wurde bisher verschwiegen. Sie wird als „Interceptor-Binding" bezeichnet und ebenso wie die Interceptor-Implementierung durch die Anwendung definiert. Ihre Aufgabe besteht darin, Beans beziehungsweise deren Methoden zur Verwendung mit einem Interceptor zu markieren. Ihre Umsetzung ist in Listing 3 zu finden, relevant ist das Vorhandensein der (Meta-)Annotation „@InterceptorBinding".

Die Annotation kann nun auf durch den Container verwaltete Beans beziehungsweise Methoden angewendet werden. Dies ist in Listing 4 dargestellt; für „ServiceA" ist eine konkrete Methode, für „ServiceB" sind alle Methoden der Klasse zur Verwendung des Interceptors markiert.

Eine letzte Frage ist noch ungeklärt: Interceptoren treten oft in Ketten auf und es ist dabei nicht unwahrscheinlich, dass Abhängigkeiten zwischen ihnen, also eine Aufrufreihenfolge sichergestellt sein muss. Denkbar ist beispielsweise die Notwendigkeit der Synchronisierung konkurrierender Threads (Concurrency-Management), bevor ein zustandsbehafteter Interceptor durchlaufen wird. Aus diesem Grund erzwingt CDI die Deklaration zu aktivierender Interceptoren für ein Bean-Archive (also eine JAR- beziehungsweise WAR-Datei) in der Datei „beans.xml". Damit gewinnt diese neben ihrer Marker-Funktion eine weitere Bedeutung. Der Inhalt wird im Listing 5 veranschaulicht, der „TracingInterceptor" wird dabei um einen „SecurityInterceptor" gewickelt, das Tracing findet also vor der Sicherheitsprüfung statt.

```
@Interceptor
@Trace
public class TracingInterceptor {

  @Inject // In this case a producer for the type Logger must be
available
  private Logger logger;
  …
}
```

Listing 2

```
@InterceptorBinding
@Retention(RUNTIME)
@Target({TYPE, METHOD})
public @interface Trace {
}
```

Listing 3

```
public class ServiceA {
   @Trace
   public int add() { …}
   public int subtract() {…}
}
@Trace
public class ServiceB {
   public int add() { …}
   public int subtract() { …}
}
```

Listing 4

Elegant verziert: Decorator

Der Interceptor-Ansatz eignet sich vor allem für die Umsetzung technischer Aspekte. Diese zeichnen sich dadurch aus, dass sie auf beliebige Methoden anwendbar sind. Manchmal kann es jedoch gewünscht sein, gezielt ein fachliches Problem in eine eigene Implementierung auszulagern, es ist also ein Bean-Interface vorhanden, an dem Methoden-Aufrufe gezielt abgefangen und behandelt werden sollen. Ein Beispiel hierfür könnte fachliches Auditing darstellen, bei dem neben dem Logging des Aufrufs auch gezielt übergebene Parameter ausgewertet werden sollen. Durch CDI wird hierfür das Konzept des Decorators angeboten (siehe Listing 6).

Bei der Injizierung von Instanzen des Typs „MyService" soll automatisch ein „AuditService" involviert sein. Dieser Aspekt wurde aber aus „MyServiceImpl" ausgelagert, da beispielsweise auch andere Implementierungen existieren könnten. Vielmehr wird die Klasse „MyServiceDecorator" eingeführt, die das gleiche Interface implementiert und als „@Delegate" die Original-Instanz injiziert bekommt, an die Methodenaufrufe nach der Auditierung weitergeleitet werden. Besonders interessant sind folgende Eigenschaften der Decorator-Implementierung:

- Die Klasse ist abstrakt. Damit wird ermöglicht, dass nur die Methoden implementiert werden müssen, in denen eine explizite Behandlung (im konkreten Fall Auditierung) stattfinden soll.
- Das Feld „delegate" ist mit der Qualifier-Annotation „@Any" versehen, sodass die Verwendung beliebig qualifizierter Implementierungen von „MyService" möglich ist.

Wie im Falle des Interceptor-Konzepts ist die vorliegende Beschreibung ebenfalls nicht ganz korrekt: Auch Decorator-Instanzen können als Ketten auftreten und deshalb ist auch hierfür eine Aktivierung mit Reihenfolgendefinition in der Datei „beans.xml" vorgesehen (siehe Listing 7).

Ereignisgetrieben: Observer-Methoden

Eine häufig benötigte Funktionalität innerhalb einer Anwendung ist die Möglichkeit, gezielt eintretende Ereignisse zu überwachen und entsprechend zu reagieren. Das Mittel der Wahl ist das Observer-Pattern: Interessierte Konsumenten – also Observer – registrieren sich an einer zentralen Stelle, um eine entsprechende Nachricht zu erhalten. Ein Problem bei der Umsetzung stellt jedoch oft die große Dynamik der Observer-Instanzen dar: Sie werden erzeugt und zerstört. Erfolgt ihre Registrierung beziehungsweise Deregistrierung im Anwendungscode, können bei fehlerhafter Implementierung (etwa bei der Überwachung ihres Lebenszyklus) Instanzen hängenbleiben, die zu teils schwer nachvollziehbaren Problemen beziehungsweise Memory-Leaks führen. Da ein CDI-Container sowieso über den Lebenszyklus von Beans informiert ist, liegt es nahe, eine Container-seitige Implementierung dieses Musters anzubieten, die sich transparent um dieses Problem kümmert. Listing 8 demonstriert die Nutzung dieses Konzepts.

Das zu überwachende Ereignis („Event") stellt eine User-Instanz dar, die persistiert werden soll. Die Klasse „User" ist keine Bean im Sinne der Verwaltung des Lebenszyklus ihrer Instanzen durch CDI, sie repräsentiert schlicht ein Datenobjekt. In die Klasse „UserDAO" kann nun eine Instanz vom Typ „Event<User>" injiziert werden, wobei der Typ „Event" vom Container zur Verfügung gestellt wird und eine „fire()"-Methode an-

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
       http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
   <interceptors>
       <class>com.buschmais.cdi.interceptor.TracingInterceptor</class>
       <class>com.buschmais.cdi.interceptor.SecurityInterceptor</class>
   </interceptors>
</beans>
```

Listing 5

```
public interface MyService {
   int add(int a, int b);
   int subtract(int a, int b);
}

public class MyServiceImpl implements MyService {
   public int add(int a, int b) { return a+b; }
   public int subtract(int a, int b) { return a-b; }
}

@Decorator
public abstract class MyServiceDecorator implements MyService {
   @Inject
   @Any @Delegate
   private MyService delegate;

   @Inject
   private AuditService auditService;

   public int add(int a, int b) {
       auditService("Adding values " + a + ":" + b);
       return delegate.add(a, b);
   }
}
```

Listing 6

```
<?xml version="1.0" encoding="UTF-8"?>
<beans …>
  <interceptors>
     …
  </interceptors>
  <decorators>
     <class>com.buschmais.cdi.decorator.MyServiceDecorator</class>
     <class>com.buschmais.cdi.decorator.AnotherServiceDecorator</class>
  </decorators>
</beans>
```

Listing 7

bietet, die im vorliegenden Fall den persistierten User als Parameter erhält.

Eine derartige Instanz ist als Ereignis allerdings recht unspezifisch: Ein Observer würde lediglich die Information erhalten, dass irgendetwas mit einem User passiert ist. Aus diesem Grund können Ereignisse qualifiziert werden; hierfür wird durch die Anwendung die Qualifier-Annotation „@Created" definiert und am Injection-Point des Events („userCreated") verwendet. Wie bereits im zweiten Teil der Artikelserie beschrieben, sind diese Qualifier über mehrere Typen hinweg verwendbar, „@Created" könnte also ebenso Events qualifizieren, die das Persistieren der Instanzen anderer Typen (etwa „Group") anzeigen.

Das Konsumieren von Events erfolgt in sogenannten „Observer-Methoden", die in Container-verwalteten Beans deklariert sind und sich durch Parameter auszeichnen, die mit „@Observes" und gegebenenfalls gewünschten Qualifier-Annotationen – im konkreten Fall „@Created" – versehen sind. Im Anwendungscode ist dabei keine Stelle enthalten, die sich explizit um die Registrierung oder Deregistrierung der Observer kümmert – dies wird vollständig und transparent durch den Container übernommen.

```
public class UserDAO {
  @Inject
  private EntityManager em;

  @Inject @Created
  private Event<User> userCreated;

  public void create(User user) {
     em.persist(user);
     userCreated.fire(user);
  }
}

@Qualifier
@Retention(RUNTIME)
public @interface Created { }

@SessionScoped
public class UserManager {

  public void onUserCreated(@Observes @Created User user) {
     …
  }
}
```

Listing 8

```
@SessionScoped
public class UserManager {
  public void onUserCreated(@Observes(notifyObserver = Reception.IF_EXISTS, during = TransactionPhase.AFTER_SUCCESS) @Created User user) {
    …
  }
}
```

Listing 9

Beim Feuern eines Events werden durch ihn automatisch alle Beans synchron benachrichtigt, die eine passende Observer-Methode implementieren. Standardmäßig entstehen dabei auch Instanzen dieser Beans in ihren jeweils deklarierten Scopes. Dieses Verhalten kann durch Parametrisierung von „@Observes" dahingehend angepasst werden, dass nur bereits existierende Instanzen („notifyObserver=IF_EXISTS") benachrichtigt werden (siehe Listing 9).

Es wurde bereits erwähnt, dass der Aufruf der Observer-Methoden im Normalfall synchron erfolgt, ein Aufruf von „Event<User>.fire(user)" arbeitet also der Reihe nach alle bekannten Observer ab und kehrt erst danach zurück. Listing 9 zeigt darüber hinaus den Einsatz des Parameters „during", der es ermöglicht, Ereignisse in Abhängigkeit vom Zustand der umgebenden Transaktion zu einem späteren Zeitpunkt zu konsumieren – in diesem Fall nach erfolgreichem Abschluss.

Richtig spannend wird CDI jedoch erst durch seine Erweiterungsschnittstellen: Neben der Definition eigener Scopes ist es beispielsweise möglich, während des Container-Starts Ereignisse wie die Registrierung von Bean-Typen zu überwachen und zu beeinflussen. Beispiele dazu gibt es im vierten und letzten Teil dieser Serie.

Dirk Mahler
dirk.mahler@buschmais.com

Dirk Mahler ist als Senior Consultant auf dem Gebiet der Java-Enterprise-Technologien tätig. In seiner täglichen Arbeit setzt er sich mit Themen rund um Software-Architektur auseinander, kann dabei eine Vorliebe für den Aspekt der Persistenz nicht verleugnen.

Unbekannte Kostbarkeiten des SDK

Heute: HTTP-Server

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir stellen in dieser Reihe derartige Features des SDK vor: die unbekannten Kostbarkeiten.

Das SDK enthält einen einfachen HTTP-Server, der ausreicht, um einfache Web-Seiten ausliefern zu können. Der HTTP-Server wird durch Möglichkeiten für sicheres HTTP, Benutzer-Authentifizierung und Filter erweitert.

Der HTTP-Server

Der HTTP-Server ist nicht im öffentlichen und damit offiziellen API des SDK enthalten, sondern im Package „com.sun.net.httpserver". Bei seiner Verwendung muss man sich über die Konsequenzen im Klaren sein: Es besteht keine Garantie dafür, dass der Server auch im nächsten Release enthalten und die Anwendung über die verschiedenen SDKs hinweg portabel ist.

Zumindest das zweite Manko kann man etwas entkräften: Sowohl das OpenJDK als auch die von Oracle distribuierte Erweiterung des OpenJDK enthalten den HTTP-Server. Oracles JRockit sowie IBMs JDK umfassen ihn ebenfalls.

Das Server-Package

Da das Package kein offizielles Java-API darstellt, ist das API-Doc nicht in der Distribution des Standard-SDK enthalten. Für eigene Versuche mit dem Package ist auf [1] verwiesen. Wir zitieren die API-Dokumentation des Package: „Provides a simple high-level Http server API, which can be used to build embedded HTTP servers."

Das Package enthält als zentrale Bestandteile die Klassen „HttpServer" und „HttpExchange" sowie das Interface „HttpHandler", die wir in unserem Beispiel gleich verwenden werden. Daneben sind noch weitere Klassen wie „Authenticator", „HttpPrincipal", „HttpsServer" und „Filter" enthalten, deren Namen den Leser auf weitere Realisierungsmöglichkeiten aufmerksam machen. Das Package enthält insgesamt ein Interface und siebzehn Klassen.

Die genannte Gefahr, dass das Package in zukünftigen Releases nicht enthalten ist, ist zumindest für den Package-Namen sehr real. Der Autor geht davon aus, dass alle „com.sun"-Artefakte über kurz oder lang auf OpenJDK-, Oracle- oder JCP-Bezeichner migriert werden, wie dies bereits in den XML-Namensräumen von Java-EE kürzlich erfolgt ist [2].

Eine einfache Server-Anwendung

Listing 1 zeigt eine einfache Server-Anwendung, die beispielsweise für eine Java-Client-Anwendung das Handbuch der Anwendung in Form von Web-Seiten publizieren kann. Die HTTP-Server-Anwendung wird durch eine Implementierung des Interface „HttpHandler" realisiert. Dieses Interface enthält die einzige Methode „handle()", die als Callback-Methode für eingehende HTTP-Requests fungiert. Die Klasse „SimpleHandler" implementiert „HttpHandler" und realisiert damit unseren einfachen HTTP-Server.

Die Methode „handle()" bekommt als einzigen Parameter einen „HttpExchange" übergeben. Diese Klasse kapselt empfangene HTTP-Requests und die zu erzeugende HTTP-Response. Sie enthält daher Methoden, um auf Details des Request und der Response zuzugreifen. Im Beispiel wird die HTTP-Methode abgefragt und nur bei „GET" eine Antwort generiert. Ebenfalls wird der URI des Request abgefragt und über einen einfachen Mechanismus eins zu eins auf ein Verzeichnis des Dateisystems abgebildet. Weitere Methoden erlauben beispielsweise den Zugriff

```
class SimpleHandler implements HttpHandler {
    private static final String ROOT = "...";
    public void handle(HttpExchange exchange) throws IOException {
        Headers responseHeaders = exchange.getResponseHeaders();
        String requestMethod = exchange.getRequestMethod();
        if (requestMethod.equalsIgnoreCase("GET")) {
            File file = new File(new File(ROOT), exchange.getRequestURI().toString());
            try (InputStream is = new FileInputStream(file);
                OutputStream responseBody = exchange.getResponseBody(); ) {
                responseHeaders.set("Content-Type", "text/html");
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_OK, file.length());
                byte[] buffer = new byte[1024];
                int len;
                while ((len = is.read(buffer)) != -1) {
                    responseBody.write(buffer, 0, len);
                }
            } catch (FileNotFoundException e) {
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_NOT_FOUND, 0);
            } catch (IOException e) {
                exchange.sendResponseHeaders(HttpURLConnection.HTTP_INTERNAL_ERROR, 0);
            }
        }
    }
}
```

Listing 1

auf das Principal-Objekt eines authentifizierten Benutzers, den Request-Header oder auf Client- und Server-IP-Adressen.

Das Beispielprogramm realisiert sogar eine einfache Fehlerbehandlung und benötigt trotzdem nur wenige Zeilen. Um das Beispiel zu vervollständigen, fehlt noch die Instanziierung der „HttpServer"-Klasse und die Bindung des oben definierten Handlers für eingehende Requests. Dies erfolgt in den Zeilen, die in der Main-Methode aufgerufen werden können (siehe Listing 2). Das Beispiel ist auf den vier genannten SDKs lauffähig, wobei uns JRockit nur für Java 6 vorliegt, sodass die Try-With-Resources-Anweisung umformuliert wurde.

Das Kompilieren mit „javac" gelingt ohne Probleme. Die Kompilate des Package sind in „rt.jar" vorhanden, sodass „javac" sie ohne weitere Konfiguration findet. Bei der Verwendung von Eclipse ist das Package zunächst nicht im Klassenpfad und man muss über „Window" –> „Preferences, Java" –> „Compiler" –> „Errors/Warnings", unter „Deprecated and Restricted API" den Eintrag „Forbidden Reference" auf „Ignore" oder „Warning" stellen.

```
InetSocketAddress addr = new InetSocketAddress(8080);
HttpServer server = HttpServer.create(addr, 0);
server.createContext("/", new SimpleHandler());
server.setExecutor(Executors.newCachedThreadPool());
server.start();
```

Listing 2

Fazit

Das SDK enthält einen einfachen HTTP-Server, der mit wenig Aufwand zu einem funktionsfähigen Web-Server ausgebaut werden kann. Obwohl der Server nicht im öffentlichen API enthalten ist und das Package mit dem Präfix „com.sun" beginnt, ist es in allen dem Autor zugänglichen SDKs (OpenJDK, Oracle, IBM, JRockit) enthalten.

Literatur

[1] http://docs.oracle.com/javase/6/docs/jre/api/net/httpserver/spec/com/sun/net/httpserver/package-summary.html.

[2] https://blogs.oracle.com/theaquarium/entry/java_ee_schema_namespace_moved.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0" und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).

Die Softwerkskammer

Markus Gärtner, Coach und Berater für it-agile GmbH

Ende des Jahres 2008 kam auf einer Mailingliste zum ersten Mal das Thema „Software Craftsmanship" in größerem Maße zur Sprache. Aus diesen Diskussionen und einigen lokalen Treffen in Chicago entstand das Manifest der Software-Craftsmanship-Bewegung [1]. Diese ist spätestens seit der ersten „SoCraTes – Software Craftsmanship and Testing (Un-)Conference" [2] im Jahr 2011 in Rückersbach auch in Deutschland angekommen.

In einer der Sessions am letzten Tag der SoCraTes kamen verschiedene Leute mit ganz unterschiedlicher Motivation zusammen. Sie hatten so viel über Software Craftsmanship in anderen Ländern gehört und über die zwei Tage viel Energie für Software-Entwicklung aufgebracht, dass sie sich fragten: „Soll es das jetzt gewesen sein – bis zum nächsten Jahr?" Sie stellten sich abhängig von der örtlichen Herkunft im Raum auf. Auf einmal standen sieben Leute um den Autor herum. Sie beschlossen, dass sie diese Energie nach Hause weitertragen wollten, um den Gedanken an eine bessere (Software-) Welt aufrechtzuerhalten.

Ein Jahr später gab es zehn User Groups zum Thema „Software Craftsmanship" in Deutschland. Sie kamen zur zweiten SoCraTes zusammen und waren bereits zu einer großen Familie herangewachsen. In den folgenden Jahren konnten sie die Community stetig weiter ausbauen, sodass sie jetzt auf eine große deutschsprachige Gemeinde blicken können. Auch in Wien, Linz und Zürich gibt es mittlerweile lokale Gruppen.

Darüber hinaus haben sie es über die Jahre immer wieder geschafft, sowohl Inspiration von außen zu holen als auch die gleiche Inspiration nach außen zu transportieren. So gibt es seit dem Jahr 2013 in UK ebenfalls eine SoCraTes-Konferenz im selben Format. Dabei fand im Jahr 2009 die erste Software-Craftsmanship-Konferenz überhaupt in London bei der BBC statt.

User Groups und Code Retreats

Beim ersten Treffen der Softwerkskammer [3] User Group in Münster, Osnabrück und Bielefeld waren es zehn Teilnehmer. Sie hatten sich auf eine rotierende Location geeinigt. Außerdem wollten sie mit ganztägigen Treffen alle drei Monate beginnen und wählten OpenSpace als Format. Ein Vierteljahr später kamen zum Treffen in Osnabrück bereits dreißig Teilnehmer. Was war zwischenzeitlich geschehen? Die kurze Antwort: Es wurde Werbung gemacht. Die lange Antwort: Martin Klose hatte kontinuierlich seit dem Global Day of Code Retreat in Bielefeld Code Retreats auf monatlicher Basis mit verschiedenen Formaten organisiert. Diese eintägigen Events verhalfen der Softwerkskammer dazu, auf das nächste Treffen aufmerksam zu machen.

Nach einem Tag Code Retreat sagten beispielsweise drei Studenten aus Paderborn, dass ihr Professor ihnen niemals so viel an einem Tag hätte beibringen können. Ein anderer Teilnehmer meinte, er würde sich am Montag einen neuen Job suchen, weil er keine Chance mehr darin sah, seine Kollegen zu „konvertieren". Code Retreats sind wahrlich ein Format, das diesen Eindruck nach nur einem Tag intensiven Programmierens vermitteln kann.

Die Erfahrungen der anderen User Groups in Deutschland sind teilweise vergleichbar, obwohl sich andere Formate etablierten. In Karlsruhe gab es monatliche Treffen, genau wie in Hamburg. In Hamburg rotierte das Format zwischen Vortrag, OpenSpace und einer Coding Session. In München kombinierte man beide Formate von Beginn an.

Neben der Diversität war auch der Zulauf in den einzelnen Gruppen über die Jahre ganz verschieden. Es gibt einige sehr große Gruppen mit mehr als hundert Mitgliedern; daneben kleinere Gruppen, die sich weniger häufig, aber mit einem kleinen Kern treffen.

Zum Teil ergeben sich auch schon Synergie-Effekte. Ein regelmäßiger Teilnehmer in Münster/Osnabrück/Bielefeld startete kürzlich eine eigene Gruppe in Hannover. Letztes Jahr reisten Delegierte aus mehreren lokalen Gruppen nach Wiesbaden für den ersten OpenSpace der Rhein-Main-Gruppe.

Auf der Agenda der Meetings stehen vielfältige Themen: Zum einen gibt es Coding Dojos und Coding Competitions, bei denen es darum geht, seine Fähigkeiten zu zeigen und von anderen lernen zu können. Hinzu kommen praktische Themen wie Erfahrungsberichte zu Domain-Driven Design, Command Query Responsibility Segregation (CQRS) und Event Sourcing, Event Storming und Clean-Code-Developer-Erfahrungen im eigenen Unternehmen. Dann sind da noch grundlegende Themen wie eine Einführung in Scrum, Kanban, Test-Driven Development (TDD) etc. Darüber hinaus

kommen auch Themen wie der beliebteste Short-Cut und die persönliche Weiterbildung zur Sprache, damit alle voneinander lernen und gemeinsam wachsen können.

SoCraTes

Die jährliche Konferenz SoCraTes bot in der Vergangenheit den lokalen Gruppierungen immer wieder die Möglichkeit, sich über neue Themen auszutauschen. Die Vertreter der London Software Craftsmanship User Group sind längst gern gesehene Stammgäste und geben Impulse auch aus anderen Ländern. Im letzten Jahr haben sie zudem das Format der SoCraTes mit auf die Insel genommen und veranstalten dort ihren eigenen OpenSpace. Über die Jahre kamen auch Teilnehmer aus Frankreich, Finnland und Israel hinzu. In diesem Jahr startet der Versuch, auf der SoCraTes vom 7. bis 10. August in Soltau nahe bei Hamburg bis zu 200 Plätze zur Verfügung zu stellen.

Und was noch?

Seit einigen Monaten haben sich alle zwei Wochen kleinere Treffen etabliert. Unter den Begriffen „Code 'n' Cake" oder „Code 'n' Coffee" kommen regelmäßig Leute zusammen, um voneinander zu lernen. Dabei werden verschiedene Themen zwischen den größeren Treffen aufgerollt. Unter dem Mantra „bring your own code" oder in Open-Source-Projekten, die händeringend Hilfe benötigen, sind die Softwerker zu vielen Taten bereit.

Darüber hinaus gibt es einige engagierte Softwerker, die nach noch mehr Erfahrungen streben. Unter diesem Aspekt haben vor einiger Zeit Obtiva und 8thLight in den USA einen sogenannten „Craftsman Swap" [4] gestartet. Dabei arbeitet ein Entwickler aus der einen Firma für eine gewisse Zeit in der anderen und ein Entwickler der anderen Firma bei der ersteren. Anschließend kehren beide zu ihrem Arbeitgeber zurück und bringen vielfältige Eindrücke aus dem jeweils anderen Unternehmen mit, die sie dann wiederum bei ihrem Arbeitgeber weiter verbreiten.

Die beiden Firmen fanden diese Erfahrung so überzeugend, dass sie das Experiment über die Jahre wiederholt haben. Anfangs konsultierten sie dazu ihren Rechtsbeistand [5], doch fanden sie schnell heraus, dass das gemeinsame Vertrauen jeden Vertrag überwiegen kann.

Natürlich hat man sich gefragt, wie man das System in Deutschland nachempfinden kann. Im letzten Jahr konnte Daniel Temme verschiedene Firmen ausfindig machen, die ihn für eine bestimmte Zeit mitarbeiten ließen. So konnte er in kurzer Zeit ziemlich viele Eindrücke über verschiedene Domänen und Technologien mitnehmen. Dieses Modell hatte bereits Corey Haines [6] vor einiger Zeit über mehr als ein Jahr verfolgt, als er arbeitslos war. Heute ist er nicht nur der Mann hinter Code Retreats, sondern auch einer der Gurus in der Software-Craftsmanship-Bewegung.

Auch wenn wir in Deutschland vielleicht noch nicht bereit sind für Craftsman Swaps, so existieren bereits Erfahrungen in verschiedenen Bereichen. Beispielsweise ist ein „One Way"-Swap denkbar, wie ihn Daniel Temme getätigt hat. Ein Entwickler arbeitet für einen Zeitraum von zwei Wochen in einem anderen Unternehmen, um von ihm zu lernen. In größeren Unternehmen sind auch interne Swaps denkbar, bei denen ein Entwickler aus dem einen Produktbereich in einem anderen für einen Zeitraum von zwei oder mehr Wochen arbeitet. Vielleicht können wir uns so langfristig auch mit der Idee anfreunden, dass wir mal einen Entwickler an ein anderes Unternehmen ausleihen. Es muss ja nicht immer die direkte Konkurrenz sein.

Eine Community of Professionals

In den frühen Tagen der Software-Craftsmanship-Bewegung dachte man unter anderem auch darüber nach, ein Äquivalent zu den Prinzipien hinter agiler Software-Entwicklung zu haben. Man nannte sie die „Ethiken" der Software-Craftsmanship-Bewegung:

- Wir sorgen uns um unseren Code, damit wir nicht nur heute den besten Mehrwert schaffen können, sondern auch noch morgen
- Wir üben, damit wir täglich die beste Arbeit leisten können
- Wir lernen, damit wir mit technologischen Entwicklungen standhalten können
- Wir teilen, damit wir von anderen lernen und andere lehren können

Fünf Jahre, nachdem diese Essenz der Craftsmanship-Bewegung festgehalten wurde, sind diese Prinzipien in die deutschsprachige Bewegung, die Softwerkskammer, eingeflossen.

Fazit

Die Mitglieder der Community sehen, dass sie ihren Code ständig verbessern können. Deshalb lernen sie von anderen, wie diese dabei vorgehen, und lernen ihre Praktiken in der täglichen Arbeit durch Swaps von Softwerkern. Sie lernen bei den Community-Events auf mehreren Ebenen von anderen Experten außerhalb ihres Umfelds. Außerdem üben sie regelmäßig, damit sie auch morgen noch ihrem Arbeitgeber den besten Mehrwert für ihren Lohn bieten können. Und all dieses Wissen teilen sie miteinander und mit der ganzen Welt.

Der Vater des Autors war Automechaniker. Vor vielen Jahren sagte er einmal zu ihm, er habe einmal geglaubt, dass mit der Lehre das Lernen aufhöre. Über die Jahre hat er jedoch erfahren, dass nach der Lehre das Lernen erst beginnt. Seitdem weiß er, dass er lebenslang lernen wird. Was für einen einfachen Automechaniker so einsichtig ist, sollte für Softwerker, die die Software schaffen, auf denen vieles basiert, selbstverständlich sein.

Literatur

[1] Manifest für Software Craftsmanship: http://manifesto.softwarecraftsmanship.org

[2] SoCraTes Conference: http://www.socrates-conference.de

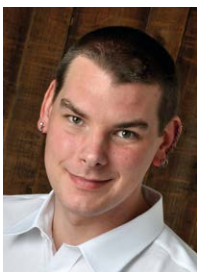
[3] Softwerkskammer: http://www.softwerkskammer.org

[4] Dave Hoover - Craftsman Swap: http://nuts.redsquirrel.com/post/80855433/craftsman-swap

[5] http://articles.chicagotribune.com/2009-06-15/news/0906140126_1_software-8th-light-perspectives

[6] http://blog.coreyhaines.com

Markus Gärtner
mgaertne@gmail.com



Markus Gärtner arbeitet als agiler Trainer, Coach und Berater für it-agile GmbH, Hamburg. Er schrieb unter anderem „ATDD by Example – A Practical Guide to Acceptance Test-Driven Development", erhielt im Jahr 2013 den „Most Influential Agile Testing Professional Person"-Award und steuert zur Softwerkskammer, der deutschen Software-Craftsmanship-Bewegung, bei. Er bloggt regelmäßig unter http://www.shino.de/blog.

JAVA FORUM stuttgart 2014

Tobias Frech, Java User Group Stuttgart

Die Java User Group Stuttgart e.V. veranstaltet am 17. Juli 2014 im Kultur- & Kongresszentrum Liederhalle (KKL) in Stuttgart wieder das Java Forum Stuttgart.

Wie im Vorjahr werden rund 1.500 Teilnehmer erwartet. Geplant sind 49 Vorträge in sieben parallelen Tracks. Zudem werden bis zu 35 Aussteller vor Ort sein, darunter auch der Interessenverbund der Java User Groups e.V. (iJUG). An der Community-Wand stehen sowohl offene White-Boards als auch BoF-Boards (Bird-of-a-Feather). Abends gibt es die Gelegenheit, sich bei verschiedenen BoF-Sessions mit Gleichgesinnten zu treffen, um über ein bestimmtes Thema zu diskutieren und sich auszutauschen. Darüber hinaus wird es wieder eine Jobbörse/Karriereecke für die Besucher geben.

Workshop „Java für Entscheider"

Die eintägige Überblicksveranstaltung am Vortag (16. Juli 2014) zeigt Begrifflichkeiten und wichtige Technologien aus der seit Jahren in der Industrie etablierten Plattform "Java". Ausgehend von strategischen Gesichtspunkten wie Bedeutung und Verbreitung wird der Blick über das Client-seitige Java (Java SE) und die wesentlichen Entwicklungswerkzeuge hinüber zum Server-seitigen Java (Java EE) gelenkt. Dort stehen dann die Bedeutung von Java als Integrationsplattform und die verschiedenen Technologien im Mittelpunkt. Darüber hinaus wird noch der Einsatz von Java in den immer wichtiger werdenden mobilen Lösungen (Android, iOS) gestreift, um dann den Bogen von der Software-Entwicklung hin zum Betrieb zu schlagen.

Experten-Forum Stuttgart

Am 18. Juli 2014 findet wieder im Anschluss an das Java Forum Stuttgart ein Experten-Forum Stuttgart statt. Auf dem Programm stehen zwölf halbtägige Workshops in sechs parallelen Tracks. Die Workshops in kleinen Gruppen mit maximal 25 Teilnehmern ermöglichen einen intensiven Austausch zwischen Trainer und Zuhörern.

Anmeldung und weitere Informationen zum Java Forum Stuttgart unter www.java-forum-stuttgart.de

Die iJUG-Mitglieder auf einen Blick

Java User Group Deutschland e.V.
http://www.java.de

DOAG Deutsche ORACLE-Anwendergruppe e. V.
http://www.doag.org

Java User Group Stuttgart e.V. (JUGS)
http://www.jugs.de

Java User Group Köln
http://www.jugcologne.eu

Java User Group Darmstadt
http://jugda.wordpress.com

Java User Group München (JUGM)
http://www.jugm.de

Java User Group Metropolregion Nürnberg
http://www.source-knights.com

Java User Group Ostfalen
http://www.jug-ostfalen.de

Java User Group Saxony
http://www.jugsaxony.org

Sun User Group Deutschland e.V.
http://www.sugd.de

Swiss Oracle User Group (SOUG)
http://www.soug.ch

Berlin Expert Days e.V.
http://www.bed-con.org

Java Student User Group Wien
www.jsug.at

Java User Group Karlsruhe
http://jug-karlsruhe.mixxt.de

Java User Group Hannover
https://www.xing.com/net/jughannover

Java User Group Augsburg
http://www.jug-augsburg.de

Java User Group Bremen
http://www.jugbremen.de

Java User Group Münster
http://www.jug-muenster.de

Java User Group Hessen
http://www.jugh.de

Java User Group Dortmund
http://www.jugdo.de

Java User Group Hamburg
http://www.jughh.de

iJUG Verbund

Der iJUG möchte alle Java-Usergroups unter einem Dach vereinen. So können sich alle Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne unter office@ijug.eu melden.

www.ijug.eu

Impressum

Herausgeber:
Interessenverbund der Java User Groups e.V. (iJUG)
Tempelhofer Weg 64, 12347 Berlin
Tel.: 030 6090 218-15
www.ijug.eu

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

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

Redaktionsbeirat:
Ronny Kröhne, IBM-Architekt;
Daniel van Ross, NeptuneLabs;
Dr. Jens Trapp, Google; André Sept, InterFace AG

Titel, Gestaltung und Satz:
Alexander Kermas,
DOAG Dienstleistungen GmbH
Titel © sandal / Fotolia.com
Titel © antishock / 123rf.com
Foto S. 44 © Jozsef Bagota / 123rf.com
Foto S. 61 © alexmit / 123rf.com

Anzeigen:
Simone Fischer
anzeigen@doag.org

Mediadaten und Preise:
http://www.doag.org/go/mediadaten

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

Inserentenverzeichnis

aformatik Training und Consulting GmbH & Co. KG, www.aformatik.de — *S. 3*

cellent AG www.cellent.de — S. 25

Heise-Verlag www.heise.de — *U 4*

Java Forum Stuttgart www.java-forum-stuttgart.de — *U 2*

Quality First Software GmbH www.qfs.de — *S. 15*

Source Talk Tage www.sourcetalk.de/2014 — *U 3*

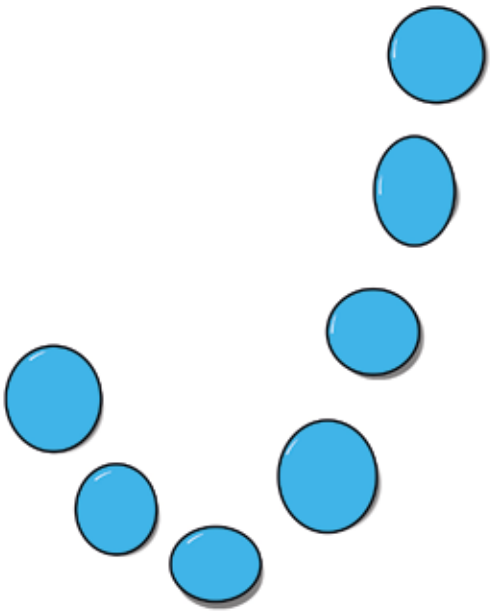
Source Talk Tage

Die Source Talk Tage versorgen Mitarbeiter von Firmen vor allem aus der Region Göttingen, Kassel, Hannover, Braunschweig mit aktuellen Handouts und Anregungen im Bereich Java und Entwicklungssysteme. Der regionale Bezug hält Zeitaufwand und Kosten gering.

Die Konferenz schafft ein Forum für Studierende aus ganz Deutschland und potentielle Arbeitgeber, um sie auf einer technischen Ebene jenseits von Jobmessen zusammenzubringen. Das bedeutet: Die Vortragenden zeigen, mit welch interessanten technischen Projekten sich ihre Firma beschäftigt und welche Lösungsmöglichkeiten sie gefunden haben. Damit präsentiert sich ein Unternehmen als spannender Arbeitgeber. Die Techniker werden zu Botschaftern ihres Unternehmens - ein einmaliges Konzept.

Dienstag 26. & Mittwoch 27. August 2014

Mathematisches Institut der Universität Göttingen



weitere Informationen unter

http://www.sourcetalk.de/2014

Die Technologie-Konferenz der Java User Group Deutschland e.V.

Die Java User Group Deutschland ist Mitveranstalter der Source Talk Tage.
Weitere Information zur Java User Group Deutschland unter http://www.java.de

Mobilität, Mensch, Maschine

Unternehmen und IT im Wandel

heise Events-Konferenz

Foto: © zentilia + tanatat – Fotolia.com

Der nächste Schritt zum mobilen Arbeitsplatz der Zukunft

Smartphones, Tablets und ultraportable Notebooks sind aus dem Alltag nicht mehr wegzudenken. Der rasante Wandel zwingt Unternehmen zur Neugestaltung von Arbeit und Arbeitsplätzen. Dabei greift der herkömmliche Ansatz, mobile Geräte noch immer wie stationäre Clients zu behandeln – restriktiv, zentral organisiert und abgesichert –, zu kurz.

Auf der heise Events-Konferenz erhalten Sie Denkanstöße & Best Practices für den nächsten Schritt zur Entwicklung des mobilen Arbeitsplatzes der Zukunft.

Unsere Experten erläutern Ihnen dabei anhand von Fallbeispielen einen ganzheitlichen Ansatz vom Device zum Workplace Management. Dieser eröffnet nicht nur neue Chancen, durch optimale Arbeitsbedingungen auch die Attraktivität des Unternehmens für die Mitarbeiter zu erhöhen, sondern fördert auch das Potenzial der Mitarbeiter und eine zukunftsfähige IT-Architektur zum strategischen Nutzen der Firma zu verbinden.

Zielgruppe: Entscheider Strategie- und Unternehmensentwicklung; Technische Entscheider, IT-Manager- und -Berater

Teilnahmegebühr: 475,- Euro

Programmschwerpunkte:

- **Arbeitstypen der Zukunft**
  Dipl.-Psych. Jürgen Wilke, Fraunhofer-Institut für Arbeitswirtschaft
- **Cyber Physical Systems – Mobil und wissensbasiert**
  Prof. Dr.-Ing. Thorsten Schöler, Fakultät für Informatik der Hochschule für angewandte Wissenschaften Augsburg
- **Die Firma auch nach Feierabend in der Hosentasche**
  Peter Meuser, iTlab Consulting

Ihre Benefits:

- Hochkarätige Referenten
- Praxisrelevanz der Vorträge
- Networking und Erfahrungsaustausch
- Begleitende Ausstellung mit Informationen über die neuesten IT-Lösungen & -Produkte

Goldsponsoren: CORTADO, F-Secure, magic OUTPERFORM THE FUTURE

Silbersponsoren: Check Point SOFTWARE TECHNOLOGIES LTD., Samsung KNOX

Organisiert von: heise Events Conferences, Seminars, Workshops

In Zusammenarbeit mit: brainworks COMPUTER TECHNOLOGIE GMBH

Weitere Informationen und Anmeldung unter: **www.heise-events.de/momema2014**
Krypto-Kampagne: **www.ct.de/pgp**