

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java wächst weiter



Microservices

Schnell und einfach implementieren

Container-Architektur

Verteilte Java-Anwendungen mit Docker

Java-Web-Anwendungen

Fallstricke bei der sicheren Entwicklung



ijug
Verbund

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





JUG SAXONY DAY



JUG SAXONY DAY 2016

am 30. September 2016 in Radebeul bei Dresden

www.jug-saxony-day.org



- Premium-Sponsoren des JUG Saxony Days 2016 -



Javaaktuell



- Sponsoren des JUG Saxony Days 2016 -



Eine Veranstaltung des JUG Saxony e.V.
Mit freundlicher Unterstützung durch



Wolfgang Taschner
Chefredakteur Java aktuell

Die Sache mit JavaFX

Tobias Frech, stellvertretender Vorstand des Interessenverbands der Java User Groups e.V. (IJUG), greift das Thema im November letzten Jahres im Rahmen der IJUG-Mitgliederversammlung auf. Er macht sich Sorgen um die Zukunft von JavaFX. Hintergrund ist, dass Oracle den Support für die Version 8u33 von JavaFX auf der ARM Embedded Platform eingestellt und zudem den offiziellen Download der JavaFX Scene Builder Binaries beendet hat (letztere sind jetzt von der Firma Gluon unter „<http://gluonhq.com/open-source/scene-builder>“ bereitgestellt). Es folgt ein offener Brief an Oracle mit der Forderung nach einem klaren Bekenntnis zu JavaFX.

Worum geht es bei JavaFX? Dahinter steckt eine in Java geschriebene Bibliothek, um Plattformübergreifend Rich-Client- und Desktop-Applikationen zu entwickeln. Die optische Gestaltung einer Anwendung ist über Stylesheets definiert, was den Vorteil hat, dass Gestaltung und funktionaler Code voneinander getrennt sind. Der eingangs genannte Scene Builder dient zur intuitiven Erstellung von Benutzeroberflächen mit JavaFX.


Dr. Michael Paus von der JUG Stuttgart organisiert im März auf der JavaLand 2016 einen Workshop zum Thema „Ist JavaFX reif für den Business-Alltag?“. Eine Umfrage unter den rund 50 Teilnehmern ergibt, dass immerhin die Hälfte JavaFX produktiv und 29 Prozent in kommerziellen Projekten einsetzt. Ebenfalls 29 Prozent sind auch bereit, für JavaFX-Support zu bezahlen.

Markus Karg, Software-Entwickler und Autor der Java aktuell, ist bei dem Workshop ebenfalls anwesend und stellt das Projekt „TeamFX“ vor. Es handelt sich um einen losen Zusammenschluss von rund fünfzehn JavaFX-Experten, die sich in irgendeiner Form um JavaFX verdient gemacht und direkt oder indirekt vom weiteren Erfolg der JavaFX-Plattform abhängig oder zumindest sehr stark daran interessiert sind, JavaFX voranzubringen. Über das Einbringen von Ideen und Arbeitsleistung, vom reinen Melden von Bugs über die Bewertung von RFCs bis hin zur Entwicklung komplett neuer Features soll ein Community-Projekt à la Linux oder Debian für OpenJFX entstehen. TeamFX übernimmt dabei die Kommunikation mit Oracle. Die Aktivitäten sind unter „<http://www.freelists.org/archive/teamfx>“ dokumentiert.

Auch Oracle hat sich bewegt und JavaFX als festen Bestandteil in den Java-SE-8-Funktionsumfang aufgenommen. JavaFX unterliegt damit der Java-SE-Roadmap, was Support bis zum Jahr 2028 bedeutet.

Es zeichnet sich also ein Hoffnungsschimmer auf dem JavaFX-Himmel ab. In diesem Sinne wünsche ich allen viel Erfolg bei ihren JavaFX-Projekten.

Ihr





Neues von den letzten Releases



Sich schnell einen Überblick über bestehende Strukturen und deren Beziehungen verschaffen

3	Editorial	28	Weiterführende Themen zum Batch Processing mit Java EE 7 <i>Philipp Buchholz</i>	53	Profiles for Eclipse – Eclipse im Enterprise-Umfeld nutzen und verwalten <i>Frederic Ebelshäuser und Sophie Hollmann</i>
5	Das Java-Tagebuch <i>Andreas Badelt</i>	34	Exploration und Visualisierung von Software-Architekturen mit jQAssistant <i>Dirk Mahler</i>	57	JAXB und Oracle XDB <i>Wolfgang Nast</i>
8	Verteilte Java-Anwendungen mit Docker <i>Dr. Ralph Guderlei und Benjamin Schmid</i>	38	Fallstricke bei der sicheren Entwicklung von Java-Web-Anwendungen <i>Dominik Schadow</i>	61	Java-Enterprise-Anwendungen effizient und schnell entwickeln <i>Anett Hübner</i>
12	JavaLand 2016: Java-Community-Konferenz mit neuem Besucherrekord <i>Marina Fischer</i>	43	Java ist auch eine Insel – Einführung, Ausbildung, Praxis <i>gelesen von Daniel Grycman</i>	66	Impressum
14	Groovy und Grails – quo vadis? <i>Falk Sippach</i>	44	Microservices – live und in Farbe <i>Dr. Thomas Schuster und Dominik Galler</i>	66	Inserentenverzeichnis
20	PL/SQL2Java – was funktioniert und was nicht <i>Stephan La Rocca</i>	49	Open-Source-Performance-Monitoring mit stagemonitor <i>Felix Barnsteiner und Fabian Trampusch</i>		
25	Canary-Releases mit der Very Awesome Microservices Platform <i>Bernd Zuther</i>				



Daten in unterschiedlichen Formaten in der Datenbank ablegen

Das Java-Tagebuch

Andreas Badelt, stellv. Leiter der DOAG Java Community

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 dritten Quartal 2016.

2. Februar 2016

Browser ohne Java

Für Otto Normalverbraucher spielt Java im Browser wohl schon länger keine Rolle mehr. Einigen Firmen mit vielen Applets im Einsatz dürfte der auslaufende Java-Support (Stichwort: „NPAPI“) aber noch ein größeres Problem werden. Für die dermaßen geplagten Entwickler, Administratoren und Projektleiter hat Oracle zumindest mal ein knappes Whitepaper herausgegeben, in dem die Alternativen erläutert sind: Web-Start, Native Bundles und „inverted browser control“, also ein in Java integrierter Browser beispielsweise mithilfe von JavaFX WebView/WebFX. Denjenigen, die sich bereits im Detail mit dem Thema auseinandergesetzt haben, wird das Whitepaper nichts Neues mehr bringen – zumindest gegenüber dem Management ist es jedoch gut, mal ein offizielles Dokument von Oracle in der Hand zu haben.

<http://www.oracle.com/technetwork/java/javase/migratingfromapplets-2872444.pdf>

18. Februar 2016

Java EE Security – eine griechische Göttin soll aufräumen

Mit dem Thema „Java EE Security“ beschäftigen sich die meisten Entwickler nicht so gerne – was auch damit zu tun hat, dass man in der Vergangenheit einen Großteil der nötigen Arbeiten in proprietäre Implementierungen ausgelagert hat. Jeder Applikationsserver kocht sein eigenes Süppchen, das ist in der Cloud natürlich umso problematischer. JSR-375 „Java EE Security API“ ist mit dem Versprechen angetreten, die Security-Konfiguration stark zu vereinheitlichen und gleichzeitig zu vereinfachen (neben weiteren Aspekten wie Unterstützung für CDI und Lambdas). Einzelne Mitglieder der Exper-

tengruppe sind mit viel Einsatz unterwegs – Ivar Grimstad und David Blevins werden auch auf der JavaLand vertreten sein, um für den JSR zu werben beziehungsweise um Feedback einzusammeln. Der aktuelle Stand der Implementierung ist unter „<https://github.com/javaee-security-spec>“ zu sehen, insbesondere die Referenz-Implementierung Soteria, benannt nach der griechischen Göttin der Sicherheit und Rettung. Wer nach übergreifenden Beispielen für die API-Nutzung in MVC oder JSF sucht, wird bei Ivar fündig. <https://www.agilejava.eu/2016/02/15/java-ee-security-api-moving-forward>

18. Februar 2016

Strings im JDK9

Strings waren bislang für mich – nun ja, halt Strings. Allgegenwärtige, aber langweilige Basis-Objekte. Bis ich auf einen Blog-Eintrag gestoßen bin. Dort plaudert Aleksey Shipilev aus dem OpenJDK-Entwickler-Team sichtlich enthusiastisch über die Herausforderungen bei der Überarbeitung von Strings im JDK 9 und führt die dramatischen Verbesserungen hinsichtlich Laufzeit und Allocation (also GC-Laufzeit) zwischen frühen JDK-9-Builds und der neuen Implementierung vor. Dann kann ich ja den StringBuffer und StringBuilder wegwerfen.

https://blogs.oracle.com/java/entry/compact_string_in_java_9

19. Februar 2016

„Dr. JUG“ analysiert Java User Groups

Der brasilianische JUG Leader und Java Champion Daniel deOliveira alias „Dr. JUG“ beschäftigt sich als Doktorand an der Universität Kent mit der Dynamik von Java User Groups. In einem Blog Post hat er einige Erkenntnisse seiner Arbeiten zusammen-

gefasst. Vieles davon ist sicher nicht spezifisch für JUGs, sondern ähnlich in jedem vergleichbaren System anzutreffen, etwa die Struktur von 10 Prozent Kern- und weiteren 15 Prozent aktiven Mitgliedern sowie 75% „Lauerern“ („lurker“), die sich nicht sichtbar beteiligen, sowie Außenstehenden („outsider“), externen Referenten und sonstigen Besuchern, etwa von anderen JUGs. Gesunde JUGs, so die Analyse, schaffen es insbesondere, dass die Mitglieder an der Peripherie sich heimisch und wohl fühlen, weil sie in der Regel Innovationen in die Gruppe tragen. Soweit, so nett.

Interessanter finde ich die folgende Zahl: DeOliveira hat versucht, alle JUGs weltweit zu zählen (ich kann mir allerdings nicht vorstellen, dass er alle gefunden hat), und kam auf 367 Gruppen mit 894.000 Mitgliedern. Ich möchte jetzt nicht pedantisch hinterfragen, wann genau jemand als Mitglied zählt, da das ja sehr verschieden gehandhabt wird, sondern einfach die Zahl als sehr beeindruckend hinnehmen. Also, liebe JUGs da draußen: Sorgt weiter für den Wohlfühlfaktor und arbeitet als Katalysator für Innovationen – vielleicht wird dann ja irgendwann die Million geknackt.

<https://community.oracle.com/docs/DOC-994258>

24. Februar 2016

Java-9-Playlist auf YouTube

Auf YouTube gibt es eine Java-9-Playlist mit einer Zusammenstellung von Vorträgen zum JDK 9 und Jigsaw, insbesondere von Mark Reinholds (Platform Chief Architect). Wer ein bisschen Zeit hat: Vielleicht ist das mal eine Alternative zu „House of Cards“ oder „Game of Thrones“.

<https://www.youtube.com/watch?v=I1s7R85GF1A&list=PLX8CzqL3ArzXT8zZ5EvdOcl6F1LyLFVh>

4. März 2016

Reza Rahman verlässt Oracle

Ein weiterer prominenter Java-Evangelist verlässt Oracle, um sich nach eigener Aussage den rein Community-getriebenen Anstrengungen um Java EE zu widmen. In seinem privaten Blog erklärt Reza Rahman die Gründe für sein Ausscheiden: Dass er zuerst skeptisch gewesen sei, als er vor einigen Jahren das Angebot von Oracle annahm, es jedoch unter anderem wegen Cameron Purdy trotzdem getan habe. Zunächst sei auch alles gut gewesen – nach dessen erzwungenem Abgang sei die Skepsis nun allerdings umso größer. Sein Blog-Eintrag ist ein Aufruf an die Community, die Dinge wieder stärker selbst in die Hand zu nehmen und denjenigen bei Oracle zu helfen, denen wirklich an der Zukunft von Java gelegen ist.
<http://blog.rahmannet.net>

7. März 2016

„JAX-RS und Hypermedia“-Vortrag bei der JUG Dortmund

Zu einem erfreulicheren Bereich der Interaktion Oracle/Community: Neues von der Nighthacking-Tour. Kurz vor der JavaLand macht Lead Java Community Manager Stephen Chin auf seiner Europa-Tour in Dortmund halt. Neben seiner Demo (Bauen einer Retro-Spielekonsole mit dem Raspberry Pi) hält Sebastian Daschner einen Vortrag zum Thema „Putting Hypermedia Back in REST with JAX-RS“. Wer sich mit dem Thema noch nicht intensiv auseinandergesetzt hat, für den könnte das Video von der Tour richtig sein.

https://blogs.oracle.com/java/entry/jax_rs_and_hypermedia

8. März 2016

JavaLand 2016: erster Tag

Die JavaLand 2016 kann im Vergleich zum Vorjahr eine deutliche Bevölkerungszunahme vorweisen. Mit der JavaLand4Kids ist bereits am Vortag der Konferenz auch wieder der Nachwuchs dabei. Neben den zahlreichen guten Vorträgen finden auch die Community-Aktivitäten in der großen Quantum-Halle großen Anklang. Viele lassen sich von den humanoiden Robotern Nao und Pepper anlocken oder wollen das VR-Formel-Eins-Auto fahren. Aber auch Code

Golf ist sehr beliebt – ein spannender und fordernder Wettbewerb, den die Teilnehmer jedoch hoffentlich nicht für die Praxis verinnerlichen; ich möchte nicht gezwungen sein, ständig Code zu lesen, der eine Aufgabe mit der kleinstmöglichen Anzahl von Zeichen erfüllt ...

Allzu viel Zeit habe ich sowieso nicht, um mir andere Aktivitäten anzuschauen. In der Early Adopters' Area (meinem eigenen Steckenpferd) haben sich wieder viele Experten zusammengefunden, die zusammen ein „Who's Who“ des Java Community Process bilden könnten. Zentrale Java-EE-8-JSRs wie MVC 1.0, JAX-RS, Security und CDI sind vertreten. Die Experten präsentieren ihre Projekte in kleinen, interaktiven Sessions oder diskutieren und programmieren gemeinsam an den Tischen. Es gibt Kontaktaufnahmen von anderen JavaLand-Teilnehmern, allerdings nicht so viele, wie man vermuten würde – schließlich bietet sich hier schon eine ziemlich einmalige Chance: Auf der JavaOne laufen sicher noch mehr Gurus herum, aber einfacher anzusprechen als im JavaLand sind sie nirgendwo. Im Vorfeld der nächsten Konferenz werden wir das wohl noch besser kommunizieren müssen. Dafür nutzen die Experten umso intensiver untereinander die Chance, einmal nicht nur per E-Mail/Telefon oder höchstens im Video-Chat zusammenzukommen. Mein persönliches Highlight am Ende des ersten JavaLand-Tages ist dann auch ein spontanes Meeting von einem Dutzend Expertengruppen-Mitgliedern sowie JCP Program Manager Heather vanCura. Wir besprechen die aktuellen Probleme mit dem Fortschritt von Java EE 8 kontrovers, aber konstruktiv (insbesondere die Tatsache, dass viele Specification Leads von Oracle überlastet scheinen und kaum auf Anfragen reagieren). Ich hoffe, es hilft, Lösungen anzustoßen, auch wenn ich befürchte, dass es noch ein sehr langer Weg ist. Neben diesen „Meta-Diskussionen“ wird auch viel an den Spezifikationen und (Referenz-) Implementierungen gefeilt.

<https://www.javaland.eu>

9. März 2016

JavaLand 2016: zweiter Tag

Nach der JavaLand-Party – die Karussells habe ich diesmal nicht genutzt, aber die Live-Band war wieder super – klingelt der Wecker viel zu früh. Die Motivation ist allerdings groß genug und der Tag geht dann (leider) auch

im Flug vorbei. Die Early Adopters' Area bietet Sessions zum Java Community Process selbst, dem EE Security JSR, JSF und Servlet 4.0, sowie Agorava (Social Media Framework für Java EE). Aber auch rundherum findet sich wieder eine überwältigende Anzahl an hervorragenden Präsentationen. Und dann gibt es auch noch Aktivitäten wie das JavaLand-Café, wo Interessierte die an der JavaLand beteiligten JUGs näher kennenlernen können; wir nutzen die Gelegenheit, um die JUGs mit Heather vom JCP zu vernetzen. Abends gibt es dann noch einige Workshops. Im Architektur-Kata-Workshop, in einem Nebenraum der Community Hall, drängen sich mehrere Gruppen von sechs oder sieben Teilnehmerinnen und Teilnehmern mit je einem Coach um die Flipcharts und diskutieren eifrig über ihr Solution Design zu einer Liste von vorgegebenen Requirements. Das sieht sehr interessant aus, allerdings ist der Workshop komplett ausgebucht, also keine spontane Teilnahme ... Aber auch so hat sich der Aufwand wieder absolut gelohnt und ich freue mich schon auf die JavaLand 2017. Mit mehr als 1.200 Teilnehmern war es eine deutliche Steigerung von 2015 auf 2016 – hoffentlich reichen im nächsten Jahr die Räume.

<https://www.javaland.eu>

10. März 2016

JavaLand 2016: dritter Tag

Im Freizeitpark kehrt wieder Ruhe ein. Dafür rauchen in den Konferenzräumen die Köpfe. Sieben Schulungsanbieter vermitteln den ganzen Tag ihr Wissen, darunter Ed Burns und Oliver Szymanski zu „JSF 2.X, HTML5, WebSocket und JSON“ oder Niko Köbler über „JavaScript-Programmierung auf der Java Virtual Machine“. Fast alle Schulungen sind komplett ausgebucht. Parallel dazu tagt die Mitgliederversammlung des Interessenverbands der Java User Groups e.V. (iJUG) und stellt bereits die Weichen für die JavaLand 2017. Ein weiteres wichtiges Thema ist die Sorge der Community um den Fortbestand von Java FX. Hier wird von Oracle ein klares Signal erwartet.

<https://www.javaland.eu>

20. März 2016

Java EE Guardians

Reza Rahman und einige Mistreiter haben die „Java EE Wächter“ gegründet. Anders

als der Name vielleicht vermuten lässt, geht es hier nicht um quasi-religiöse Bestrebungen und das Sichern des „einen, wahren Java“ – sondern im Gegenteil darum, Java aus der Community heraus wieder voranzutreiben – ähnlich wie bei Adopt-a-JSR und Adopt-OpenJDK. Deren Problem ist nur: Wenn etwa wichtige JSRs nicht vorankommen, weil die zentralen Rollen mit Oracle-Mitarbeitern besetzt sind und diese aus welchen Gründen auch immer ihre Arbeit nicht erledigen, dann kann ein Adopt-a-JSR-Programm auch nicht mehr viel machen. Die Gruppe hat ein Twitter Handle („@javaee_guardian“) und eine schnell wachsende Google-Gruppe, zu der bereits eine ganze Reihe sehr prominenter Java-Experten gehören – auch einige Leute aus der spontanen Runde in der JavaLand tauchen hier wieder auf.

<http://blog.rahmannet.net>

17. März 2016

DukeScript und „WORA“

„Write Once, Run Anywhere“ (WORA) – dieses Versprechen ist mit neuen Plattformen und dem Abschied von Applets für Java nicht mehr ganz gültig. An diesem Punkt will Toni Epple mit seinem DukeScript-Projekt ansetzen. DukeScript ist nicht ganz neu (es hat bereits im Jahr 2014 den „Duke’s Choice Award“ für innovative Projekte gewonnen), aber auf jeden Fall einen Blick wert. Es besteht aus einer dünnen Schicht, die eine JVM mit einem HTML5-Renderer verbindet. In einem Beitrag zu Stephen Chins Nighthacking-Tour erklärt Toni, wie es funktioniert.

<https://www.youtube.com/watch?v=JX4gePMOpVA>

28. März 2016

Modules System in JDK 9 integriert

Chief Platform Architect Mark Reinholds hat verkündet, dass das Java-Modulsystem (Projekt „Jigsaw“) jetzt in das JDK 9 integriert ist (ab dem Early Access-Build 111). Alle Applikationen, die keine internen Packages, sondern nur Java-SE-Standard-APIs benutzen und auf JDK 8 laufen, „sollten“ mit dem JDK 9 einfach weiter laufen. Nutzt man interne Packages (wie „sun.*“), werden sie nach der tiefen Umstrukturierung des JDK wahrscheinlich erst mal nicht laufen, ein Workaround bietet sich jedoch erst mal

mit dem Flag „-XaddExports“ an. Fröhliches Testen.

https://blogs.oracle.com/java/entry/module_system_in_jdk_9

6. April 2016

JCP Executive Committee ist besorgt

Auf dem Executive Committee Meeting des Java Community Process haben die London Java Community und andere Vertreter ihre Sorge darüber zum Ausdruck gebracht, dass Oracle an einem Fortschritt von Java EE vielleicht nicht mehr interessiert sei (siehe die vorigen Tagebuch-Einträge über Java EE Guardians etc. zu den Argumenten). Einen interessanten Aspekt möchte ich aber erwähnen: Bruno Souza von der brasilianischen JUG SouJava stößt eine Diskussion darüber an, ob der JCP auch unabhängig von Oracle existieren und die Standardisierung weiter-treiben könne; dies sei doch schließlich sein Zweck, und Aufgabe des Executive Committee sei es, dies zu ermöglichen. Die Diskussion wird, wie zu erwarten, ohne konkretes Ergebnis vertagt – sie könnte allerdings noch sehr interessant werden.

<https://jcp.org/about/java/communityprocess/ec-public/materials/2016-04-05/April-2016-Public-Minutes.html>

6. April 2016

Java EE und Spring – ein Versöhnungsversuch

Java EE und Spring – das sind doch zwei parallele Welten, richtig? Josh Long und Markus Eisele haben sich für einen Vortrag auf der O’Reilly Software Architecture Conference die Aufgabe gestellt, sie miteinander zu verbinden, weil dies zum Beispiel für Migrationsszenarios erforderlich sein könnte oder weil Spring häufig Features anbietet, die Java-EE-Standards (noch) nicht abdecken.

<https://github.com/myfear/javaee-spring-tutorial>

28. April 2016

Das Ende von java.net

Für alle normalen Java-Entwickler: keine Panik (fürs Erste). Es geht natürlich nicht um das Package. Für alle Expertengruppen-Mitglieder und Unterstützer von JSRs sowie andere, die Projekte auf der Site „java.net“ haben: Bitte geraten Sie jetzt in Panik, bevor

es zu spät ist! Die „java.net“-Forge, also die Software-Projekte, sollen gemeinsam mit dem noch auf „kenai.com“ stehenden Content zum 28. April 2017 verschwinden. Das ist nach dem Chaos um verloren gegangene Einträge beim Umzug der Blogs auf „community.oracle.com“ der nächste Nackenschlag. Sogar ein deutlich gewaltigerer, weil er insbesondere die Java Specification Requests mit all ihren Daten betrifft. Es gibt wohl keinen generellen Plan für eine Nachfolge, die einzelnen Projekte sollen sich selbst eine neue Heimat suchen. Hat Oracle nicht mehr genug Platz in der Cloud? Leider ist das Thema nicht so lustig. Zwar können die Betroffenen einmalig eine Kopie der Daten und einen Redirect auf eine neue Site beantragen. Aber selbst mit den aktuellen Plänen (die sicherlich nicht mehr eingehalten werden) wäre das wohl vor der Fertigstellung von Java EE 8 – also in-flight – und somit eine weitere Belastung für das Programm. Zumindest stellt Mark Reinhold schon mal klar, dass das OpenJDK nicht betroffen ist. Na dann ...

<http://bit.ly/1Z2pLfc>

Andreas Badelt

Leiter der DOAG SIG Java



Andreas Badelt ist ein freiberuflicher eCommerce Solution Architect („www.badelt.it“). Daneben organisiert er seit 2001 ehrenamtlich die Special Interest Group (SIG) Development sowie die SIG Java der DOAG Deutsche ORACLE-Anwendergruppe e.V. Daneben war er von 2001 bis 2015 ehrenamtlich in der Development Community und ist seit 2015 in der neugegründeten Java Community der DOAG Deutsche ORACLE-Anwendergruppe e.V. aktiv.



Verteilte Java-Anwendungen mit Docker

Dr. Ralph Guderlei und Benjamin Schmid, eXXcellent solutions GmbH

Docker ist in aller Munde; gleichzeitig hört man vom Tod der klassischen Java Application Server. Der Artikel beleuchtet die Funktionsweise von Docker und zeigt, wie mit Docker Anwendungen strukturiert werden können.

Docker [1] ist eine Plattform zur Bereitstellung und Ausführung von Anwendungen. Die Ausführung erfolgt in einer Sandbox, isoliert von anderen Prozessen auf demselben Rechner. Innerhalb eines Containers sind die übrigen Prozesse nicht sichtbar. Jede Sandbox verfügt über ein eigenes Dateisystem und eine eigene Netzwerk-Verbindung. Die Basis von Docker ist nicht neu. Es werden Funktionalitäten zur Betriebssystem-Virtualisierung verwendet, die schon seit längerer Zeit im Linux-Kernel enthalten sind. Do-

cker setzt damit auch Linux für den Einsatz voraus, kann aber innerhalb von virtuellen Maschinen auch unter Windows und OSX betrieben werden.

Die wirklichen Neuerungen von Docker sind das portable Paketformat (sogenannte „Images“), gute Tools und ein Cloud-basiertes Verzeichnis („DockerHub Registry“ [2]) für fertige Images. Ein Image ist nichts Weiteres als eine paketierte, schreibgeschützte Zusammenstellung von Dateien, die als Blaupause für eine von Docker bereitgestellte

System-Umgebung dient. Diese enthält sowohl die Anwendung als auch sämtliche Abhängigkeiten der Anwendung. Ein Container ist eine solche System-Umgebung. Er teilt sich mit dem zugrunde liegenden Betriebssystem nur den Kernel. Alle weiteren Dateien (auch Kommandozeilen-Werkzeuge) kommen aus dem basierenden Image (siehe Abbildung 1).

Der Artikel erläutert die Arbeit mit Docker und dessen Funktionsweise Schritt für Schritt anhand eines Beispiels. Als Erstes

wird eine Java-Anwendung in ein Image verpackt und als Container gestartet. Danach wird gezeigt, wie sich diese Anwendung mit einer Datenbank verbinden lässt. Am Ende ist ein einfacher Cluster von Java-basierten Microservices entstanden.

Meine Anwendung als Docker-Image

Der erste Schritt in der Arbeit mit Docker besteht darin, die eigene Anwendung als Docker-Image zu verpacken. Dazu gibt es zwei Möglichkeiten: Entweder man baut das Image manuell auf oder das Image wird automatisiert über ein sogenanntes „Dockerfile“ erstellt. Letztere Variante ist zu bevorzugen. Als Basis für ein eigenes Image dient dabei ein bereits fertiges Image („Base-Image“). Dieses wird normalerweise von der DockerHub-Registry bezogen. Angeboten sind dort Images für Betriebssysteme (Debian, CentOS etc.), Laufzeit-Umgebungen (Java, Node.js) oder Anwendungen (Datenbanken, Webserver etc.).

Um eine Java-Anwendung in ein Docker-Image zu verpacken, bietet sich das offizielle Java-Image als Base-Image an. *Listing 1* zeigt das minimale Dockerfile für unsere Anwendung.

Die erste Zeile nennt das zu verwendende Base-Image. Danach wird vermerkt, wer für dieses Dockerfile verantwortlich ist. In den folgenden Zeilen wird das Dateisystem für das Image bearbeitet, in diesem Fall durch das Hinzufügen eines JAR und einer Konfigurations-Datei. Die letzten beiden Zeilen enthalten Informationen, die das Laufzeitverhalten eines auf diesem Image basierenden Containers beschreiben – zum einen über die Netzwerk-Ports, die der Container verwenden wird, und zum anderen darüber, welcher Befehl beim Start des Containers ausgeführt werden soll.

```
FROM java:8 # base image
MAINTAINER Ralph Guderlei
WORKDIR /opt
# add JAR & config file to image
ADD target/rest-microservice-1.0.0.jar app.jar
ADD src/main/resources/example.yml app.yml
EXPOSE 8080 8081 # announce exported ports
ENTRYPOINT java -jar app.jar server # run the JAR on start
```

Listing 1

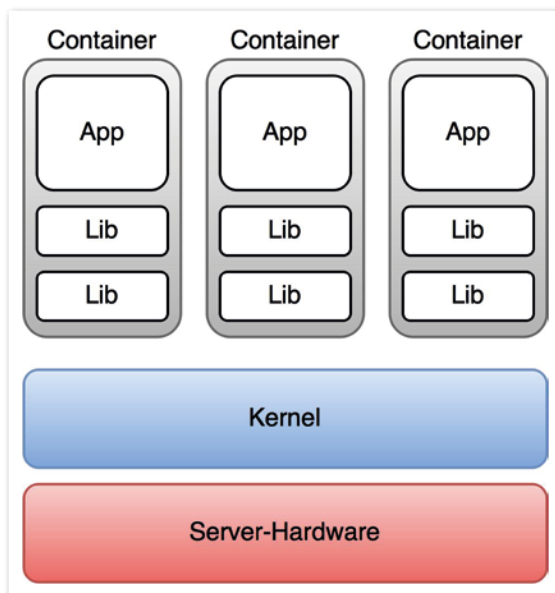


Abbildung 1: Schematischer Aufbau von Docker

Der Befehl „docker build -t my/microservice“ erzeugt nun das Image und legt es in der lokalen Registry ab. Ein Image sollte immer so einfach wie möglich gehalten sein und nur die absolut notwendigen Dateien enthalten. Entsprechend sollte ein Image auch nur einen einzigen Prozess enthalten, insbesondere sollte kein SSH-Server laufen. Das Thema „Inspektion“ ist nachfolgend behandelt. Container werden quasi als Wegwerf-Ware betrachtet. Sie sollten beliebig gestartet und angehalten sowie möglichst einfach durch eine neuere Version ersetzt werden können.

Arbeiten mit Containern

Nachdem das Image erstellt wurde, lässt sich mit „docker run -d -p 8080:8080 --name web1 my/microservice“ ein neuer Container erzeugen und starten. Der Parameter „-d“ sorgt für einen Start des Containers im Hintergrund und der Parameter „-p“ bindet den

Port 8080 des Containers an den Port 8080 des Hosts. Auf unseren Microservice kann nun über „http://localhost:8080“ zugegriffen werden. Es empfiehlt sich, den Containern explizit Namen zu geben (Parameter „--name“), um Container leichter identifizieren zu können.

Existierende Container können mit „docker start <container>“ beziehungsweise „docker stop <container>“ gestartet beziehungsweise gestoppt werden. Der Befehl „docker ps“ zeigt alle laufenden Container an und der Parameter „-a“ auch alle gestoppten.

Ein Blick in den Container

Jeder Container ist in Isolation zum Host-Betriebssystem und verfügt deshalb über eine eigene Netzwerk-Verbindung. Dafür erhält jeder Container eine eigene private IP-Adresse und ist über diese vom Host-Betriebssystem aus erreichbar. Der Befehl „docker inspect <container>“ kann die IP-Adresse und weitere Metadaten zu einem Container ermitteln.

Da der Container abgeschottet vom Host-Betriebssystem läuft, stellt sich die Frage, wie Informationen über den laufenden Container beziehungsweise die im Container laufende Anwendung ermittelt werden können, beispielsweise zur Untersuchung von Problemen. Wenn mehrere Instanzen eines Containers laufen, kann so gezielt auf einen bestimmten Container zugegriffen werden. Um die Diagnose einer Anwendung zu er-

leichtern, bieten Frameworks wie Dropwizard oder Spring Boot Health-Checks und Metriken an, um den Zustand einer Anwendung abzufragen. So lässt sich zum Beispiel einfach feststellen, ob eine Datenbank-Verbindung funktioniert oder nicht.

Nicht jede Anwendung ist jedoch so gebaut, dass alle für eine Diagnose notwendigen Informationen über das Netzwerk abgegriffen werden können. Docker bietet deswegen den Befehl „docker logs <container>“, um auf die Log-Ausgabe („stdout“ und „stderr“) des Containers zugreifen zu können. Wenn die Anwendung also ihre Logs auf die Standard-Ausgabe schreibt, können die Logs mit Docker-Mitteln einfach abgegriffen werden.

Wenn auch das nicht reicht, bietet Docker noch die Möglichkeit, mit „docker exec“ weitere Prozesse im Container zu starten. Mit „docker exec web1 bin/bash“ kann man eine (Root-)Shell im Container starten und damit beliebig innerhalb des Containers arbeiten, vergleichbar mit einem SSH-Zugriff auf eine normale virtuelle Maschine.

Datenhaltung in Containern

Wie bereits erwähnt, sind Images schreibgeschützt. Sie bestehen aus einem oder mehreren Layern. Layer sind Zusammenstellungen von Dateien und in etwa mit einem Commit-Objekt einer Versionsverwaltung wie Git vergleichbar. Docker enthält also eine Versionsverwaltung, die die Änderungen an einem Image nachvollziehbar machen. Wie Images sind auch Layer schreibgeschützt. Wird also eine Datei geändert oder gelöscht, so wird ein neuer Layer mit den Änderungen erstellt; die ursprüngliche Version ist aber noch im alten Layer enthalten und wird nur von der neuen Version überblendet. Dieses Verhalten sollte man im Blick behalten, insbesondere wenn die Größe eines Image eine Rolle spielt. Legt man beispielsweise zuerst eine große Datei an und löscht diese später wieder, so ist die Datei noch im Image enthalten und vergrößert es entsprechend.

Container erweitern das Image um einen schreibbaren Layer. Dieser ist an den Lebenszyklus des Containers gebunden. Wenn also der Container entfernt wird, werden auch die Dateien gelöscht, die der Container erzeugt hat.

Bei zustandslosen Services ist das kein Problem. Sobald jedoch Daten dauerhaft gehalten werden sollen, ändert sich das. Das einfachste Beispiel dafür ist eine Datenbank in einem Container. Wenn diese auf eine

```
haproxy:
  image: haproxy:1.5
  ports:
    - 8080:80
  volumes:
    - ./etc/haproxy.cfg:/usr/local/etc/haproxy/haproxy.cfg
  links:
    - web1
    - web2
    - web3
web1:
  image: exxcellent/docker_talk
  links:
    - db
web2:
  image: exxcellent/docker_talk
  links:
    - db
web3:
  image: exxcellent/docker_talk
  links:
    - db
db:
  image: postgres:9
  volumes:
    - ./data:/var/lib/postgresql/data
  environment:
    - POSTGRES_PASSWORD=mysecretpassword
```

Listing 2

neue Version aktualisiert werden soll, dann geschieht das normalerweise nicht durch Aktualisieren des Containers, sondern durch Verwerfen des alten Containers und Erzeugen eines neuen mit der neuen Version der Datenbank. Wenn die Datenbank-Daten im Container abgelegt sind, gehen sie mit dem Verwerfen des Containers verloren.

Um dieses Problem zu umgehen, bietet Docker die Möglichkeit, Daten in sogenannten „Volumes“ abzulegen. Diese haben ihren eigenen Lebenszyklus und können unabhängig von einem Container verwaltet werden. Es gibt zwei Möglichkeiten, Volumes bereitzustellen: das Einbinden von lokalen Dateisystemen in einen Container oder die Erzeugung eines speziellen Volume-Containers.

Der Befehl „docker run -d --name db -v /share/data:/var/lib/postgresql/data v postgres:9“ startet eine PostgreSQL-Datenbank. Diese legt ihre Daten im Verzeichnis „/share/data“ des Hosts ab. Der Parameter „-v“ definiert dieses Volume.

Mehrere Container verbinden

Eine Anwendung besteht üblicherweise nicht nur aus einem Teil: Load Balancer, Application Server und Datenbank sind gebräuchliche Bestandteile. Sie sollten in separaten Containern laufen und damit stellt sich das Problem, wie diese Teile miteinander kommunizieren sollen. Die IP-Adressen der Container

werden dynamisch vergeben und lassen sich damit schwer an andere Container übergeben. Docker bietet dazu sogenannte „Links“, um Container über ihre Namen miteinander zu verknüpfen. Über den Parameter „--link“ beim Aufruf von „docker run“ können Container miteinander verbunden werden: „docker run -d -p 8080:8080 --name balancer --link web1:web1 my/loadbalancer“.

Im Beispiel wird der neue Load-Balancer- mit dem Microservice-Container verbunden. Der Load-Balancer-Container kann damit auf sämtliche Netzwerk-Ports des Microservice-Containers zugreifen. Der Hostname „web1“ wird im Load-Balancer-Container mit der IP-Adresse des Microservice-Containers aufgelöst und eine Reihe von Umgebungsvariablen für Zugriffs-Informationen auf den Microservice-Container bereitgestellt. Falls der „web1“-Container neu gestartet wird, werden die Informationen im Load-Balancer entsprechend aktualisiert.

Um die Zusammenstellung von Containern zu einer Anwendung einfacher zu gestalten, gibt es das Werkzeug Docker Compose [3]. Über eine Konfiguration werden die einzelnen Container und deren Verbindungen definiert. Ein einziger Befehl startet dann alle Container gemeinsam. Listing 2 zeigt die Konfiguration für unser System aus Load Balancer, drei Microservices und Datenbank.

Ein einfaches „docker-compose up“ startet nun das ganze System.

Docker und Java

Bisher sind im Artikel nur wenige Annahmen zu der genutzten Java-Anwendung getroffen worden. Normalerweise sind keine Anpassungen an einer Java-Anwendung notwendig, um diese in einem Docker-Container betreiben zu können. Nur bei der Erstellung des Anwendungs-Image ist davon ausgegangen worden, dass der Java-Microservice als Fat Jar ausgeliefert wird.

Prinzipiell lassen sich auch herkömmliche Application-Server-Anwendungen in Docker betreiben. Die DockerHub-Registry bietet fertige Base-Images für Open-Source-Container wie Tomcat oder WildFly. Für kommerzielle Container muss ein eigenes Image erstellt werden, was recht aufwändig sein kann. Die Einfachheit des Ansatzes geht dabei dann teilweise verloren. Fat Jars haben gegenüber Application-Servern normalerweise auch deutlich kürzere Startup-Zeiten. Die Erzeugung lässt sich in die üblichen Build-Tools (Maven, Ant etc.) integrieren und kann so leicht im Rahmen eines Continuous-Integration-Prozesses durchgeführt werden.

Wenn man aber beabsichtigt, eine Java-Anwendung mit Docker zu betreiben, sollte man ein paar Punkte beachten: Wichtig ist zunächst ein möglichst einfaches Erstellen des Image. Im besten Fall reicht es aus, ein JAR in das Docker-Image zu kopieren und die Anwendung mit einem einfachen „java -jar“ zu starten. Aktuelle Frameworks wie Dropwizard [4], Spring Boot [5] oder Vert.X [6] sind auf genau diese Art der Bereitstellung ausgerichtet und unterstützen oft kein Deployment im Servlet-Container mehr.

Die Konfiguration der Anwendung sollte komplett externalisiert sein und das Image keine Konfiguration enthalten. Die Konfiguration sollte dem Container beim Start mitgegeben werden. Am besten eignen sich Konfigurationsdateien oder Umgebungsvariablen zur Konfiguration. Dateien können einfach per Volume in den Container geladen werden, Umgebungsvariablen können ebenfalls einfach beim Start übergeben werden.

Der Zugriff auf Inhalte eines Containers ist immer etwas umständlich. Deswegen sollte die Anwendung Log-Informationen immer auf die Standard-Ausgabe schreiben, um sie mit „docker logs“ einfach abgreifen zu können. Noch besser ist es, den Zustand

der Anwendung über das Netzwerk (also REST-Service) abfragbar zu machen. Dropwizard und Spring Boot bieten dazu sogenannte „Health Checks“ an. Diese geben dann Informationen darüber, ob die Anwendung erwartungsgemäß funktioniert oder ob Probleme vorhanden sind, beispielsweise ob die Datenbank erreichbar ist oder nicht. Health Checks können dann auch vom Load Balancer genutzt werden, um defekte Anwendungs-Instanzen aus dem Cluster zu nehmen.

Ausblick: Docker und Cluster-Betrieb

Der Artikel hat sich bisher ausschließlich mit dem Betrieb einer Anwendung auf einem einzelnen Host beschäftigt. Die Funktionalität von Docker beschränkt sich auch auf dieses Szenario. Das gilt insbesondere für Container-Links. Um eine Docker-basierte Anwendung auf einem Cluster von Rechnern zu betreiben, müssen weitere Werkzeuge eingesetzt werden. Sie sorgen für eine Verteilung der Anwendung über die einzelnen Rechner. Für diese Aufgabe gibt es eine Reihe von Lösungen. Zu erwähnen sind hier Apache Mesos/Marathon [7], Kubernetes [8] und Docker Swarm [9]. Letzteres befindet sich noch in der Beta-Phase und wird noch nicht für den produktiven Einsatz empfohlen. Die Beschreibung der Tools und Vorgehensweisen für deren Einsatz ist mit einer gewissen Komplexität verbunden und würde den Rahmen dieses Artikels sprengen.

Fazit

Docker ist nur ein Werkzeug für die Bereitstellung von Anwendungen. Auch Java-Anwendungen lassen sich mit Docker problemlos betreiben. Docker lässt sich einfach bedienen und lässt sich auch in die übliche Toollandschaft integrieren. Allerdings bringt Docker auch eine gewisse Komplexität mit und wirft neue Fragen zu Sicherheit und Verantwortlichkeiten im Betrieb auf, die bei einem Einsatz zu berücksichtigen sind. Docker ist also sicher kein Allheilmittel für alle Probleme, kann aber eine interessante Alternative zu herkömmlichen Vorgehensweisen sein.

Links

- [1] <https://www.docker.com>
- [2] <https://hub.docker.com>
- [3] <https://www.docker.com/docker-compose>
- [4] <http://www.dropwizard.io>
- [5] <http://projects.spring.io/spring-boot>
- [6] <http://vertx.io>

[7] <http://mesos.apache.org>

[8] <http://kubernetes.io>

[9] <https://www.docker.com/docker-swarm>

Dr. Ralph Guderlei
ralph.guderlei@excellent.de

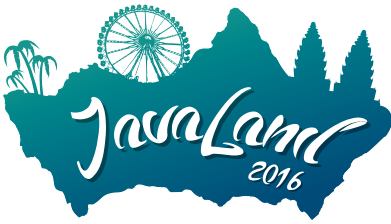


Dr. Ralph Guderlei ist Technology Advisor bei der eXcellent solutions GmbH in Ulm. Neben der Arbeit als Architekt/Projektleiter in unterschiedlichen Kundenprojekten berät er Teams in technologischen und methodischen Fragestellungen. Seine Schwerpunkte dabei sind zukunftsfähige Software-Architekturen, Web-Entwicklung und alternative JVM-Sprachen.

Benjamin Schmid
benjamin.schmid@excellent.de



Benjamin Schmid ist Technology Advisor bei der eXcellent solutions, dort primärer Ansprechpartner in allen technologischen und methodischen Fragestellungen und zudem auf der stetigen Suche nach innovativen, soliden und nachhaltigen Lösungen. Seine praxisnahen Erfahrungen und Aha-Momente rund um Java, Web und .NET gibt er dabei immer wieder gerne auf Konferenzen und in Fachartikeln weiter.



JavaLand 2016: Java-Community-Konferenz mit neuem Besucherrekord

Marina Fischer, DOAG Online

Zur dritten Auflage der Java-Konferenz vom 8. bis 10. März 2016 kamen mehr als 1.200 Teilnehmer, um sich im Phantasialand Brühl bei Köln über die neuesten Trends im Java-Umfeld zu informieren und auszutauschen. Die JavaLand gehört damit zu den größten Java-Konferenzen in Europa.

Am Dienstagmorgen fiel der Startschuss: Mit einem lauten „Jatumba“ rief die Java-Community zu einer erneuten Besiedelung des JavaLandes auf. Für drei Tage verwandelte sich das Phantasialand Brühl in das Zentrum der Java-Community. Auf die Besucher wartete ein umfangreiches Konferenzprogramm mit mehr als 100 Vorträgen, acht Streams und einer Vielzahl von Community-Aktivitäten. Beim anschließenden Schulungstag am 10. März konnten die Teilnehmer in sieben Seminaren ihr Wissen vertiefen. Organisiert wurde die Konferenz von der DOAG Dienstleistungen GmbH in Zusammenarbeit mit Heise Medien.

Ein Highlight der Veranstaltung: Das zweitägige Vortragsprogramm mit zahlreichen internationalen und nationalen Top-Speakern, darunter die in der Java-Szene sehr umtriebigen Bruno Borges, David Blevins, Kirk Pepperdine, Hendrik Ebberts, Stephen Chin und Holly Cummins. Das Programm deckte die sieben Themenbereiche „Core Java und JVM-basierte Sprachen“, „Architektur & Security“, „Enterprise Java & Cloud“, „IDEs & Tools“, „Frontend & Mobile“, „Internet der Dinge“ sowie „Container und Microservices“ ab.

Für frischen Wind im Konferenzprogramm sorgte in diesem Jahr der neu geschaffene Newcomer-Stream: Zehn Entwickler, die bisher noch nie auf einer Konferenzbühne standen, feierten am Dienstag ihr Debüt und gaben erstmals ihr Know-how zum Besten. Betreut wurden sie von namhaften Mentoren der Java-Szene wie Charles Nutter, Roland Huß und Anton Arhipov.

Fünfzehn Community-Aktivitäten, die zusammen mit zahlreichen Java User Groups

gestaltet wurden, boten an den ersten beiden Veranstaltungstagen viel Spaß beim Mitmachen und Ausprobieren: Ob in den Workshops „Architektur Kata“, „Coding Dojo“ oder „Code Shrink“, beim „Java Innovation Lab“ oder der „Early Adopters' Area“ – bei diesen und weiteren Sessions war für jeden etwas dabei. Das gegenseitige Lernen voneinander und der Austausch standen dabei an erster Stelle.

Viele Gelegenheiten zum Netzwerken bot neben der begleitenden Ausstellung mit rund 30 Ausstellern insbesondere die traditionelle Abendveranstaltung am Dienstagabend, bei der eine Vielzahl von Fahrgeschäften exklusiv für die Besucher geöffnet waren.

JavaLand4Kids: Mehr als nur YouTube und Smartphones

Kinder spielerisch an das Programmieren heranzuführen und sie fit für den Umgang mit Computern und neuen Technologien machen – den begeisterten Stimmen der kleinen Teilnehmer und ihrer Mentoren nach geht das Konzept der JavaLand4Kids mehr als auf. Rund zwanzig Grundschüler im Alter von neun und zehn Jahren nutzten auch bei der zweiten Ausgabe der Veranstaltung im Phantasialand in Brühl bei Köln die Gelegenheit, viel Neues zu erforschen und auszuprobieren.

Der kleine „Nao Roboter“ sorgte bei den Kindern schon gleich zu Beginn für viel Begeisterung, als er sich persönlich vorstellte. Mit seinen vielen Sensoren am Körper und einem eingebauten Sonar bot der menschlich wirkende Roboter viele Möglichkeiten der visuellen Programmierung. Die Kinder lernten schnell, die Zusammenhänge zwischen den einzelnen Aktionen des Roboters und

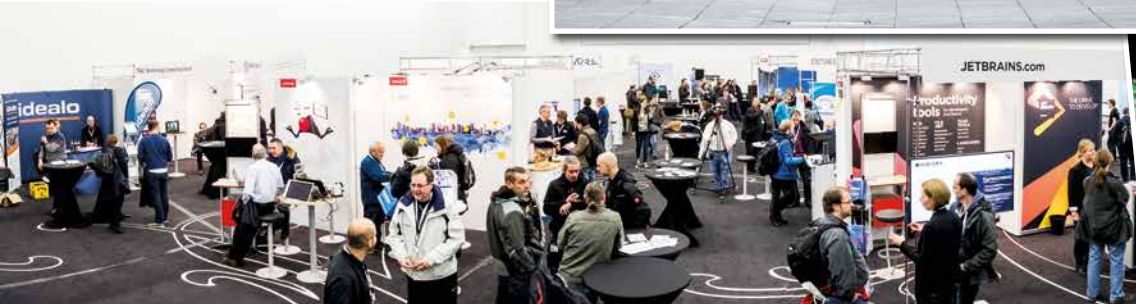
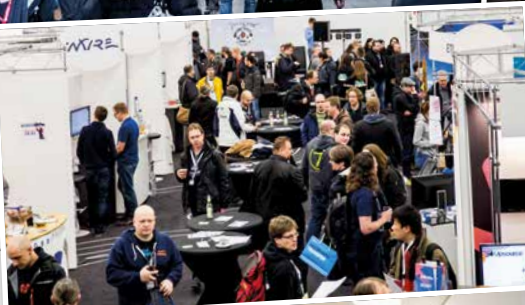
der zugrunde liegenden Programmierung zu verstehen und konnten schon bald mit dem Roboter interagieren sowie verschiedene Choreografie-Übungen absolvieren.

Im Workshop „Jumping Sumo“ brachten die Kinder der Drohne eine Reihe von Kunststücken bei. Neben 80 cm hohen Sprüngen programmierten sie mithilfe eines Routenplans auch eine Abfolge von Aktionen und konnten so die Drohne gezielt steuern. „Wichtig ist uns bei der JavaLand4Kids vor allem, dass die Kinder den kreativen Umgang mit Computern lernen und einen Einblick bekommen, wie die Technik funktioniert, anstatt sie nur zu benutzen“, betont Uwe Sauerbrei, der die JavaLand4Kids mitorganisiert hat und sich als einer der Leiter der Java User Group Hamburg auch für die Interessen der erwachsenen Java-Programmierer einsetzt.

Das Datum für die JavaLand 2017 steht bereits fest: Sie findet vom 28. bis 30. März 2017 an gewohnter Stätte im Phantasialand statt.

Marina Fischer
marina.fischer@doag.org





Groovy und Grails – quo vadis?

Falk Sippach, OIO Orientation in Objects GmbH



Die Turbulenzen des Jahres 2015 haben Groovy und Grails zwar gut überstanden, inhaltlich gingen die Neuerungen der letzten Releases allerdings etwas unter. Dabei hat sich trotz aller Probleme wieder eine Menge getan.

Darum wollen wir in diesem Artikel einen Blick auf die Änderungen werfen,

zunächst aber die typischen Eigenschaften zusammentragen und auch Kritikpunkte diskutieren.



GRAILS

Groovy hat seit jeher eine treue Fangemeinde (siehe Artikel in der letzten Ausgabe). Ein Großteil der Java-Entwickler meidet die Sprache aber, weil es sich ja „nur“ um eine dynamisch typisierte Skriptsprache handelt. Für Irritation hat in Jahr 2009 auch die Aussage von James Strachan [1], einem der Gründer, gesorgt: „I can honestly say if someone had shown me the Programming in Scala book by Martin Odersky, Lex Spoon & Bill Venner's back in 2003 I'd probably have never created Groovy.“

Man darf diese Aussage allerdings nicht falsch verstehen und sollte sich den gesamten Kontext des Blog-Beitrags anschauen. James Strachan hatte damals über einen Nachfolger von Java philosophiert. Groovy stand natürlich auch auf der Kandidatenliste. Durch die statische Typisierung und die mächtigen Typ-Inferenz-Mechanismen war Scala zu diesem Zeitpunkt für ihn aber die bessere Alternative zu Java. Das damals noch rein dynamisch typisierte Groovy konnte er sich nicht als kompletten Ersatz für Java vorstellen.

Auch heute lassen sich Groovy und Scala nur schwer vergleichen, auch wenn es Gemeinsamkeiten gibt. Letztlich verfolgen sie jedoch ganz unterschiedliche Ansätze und haben sich jeweils eine Nische in der Welt der alternativen JVM-Sprachen erkämpft. Als Ersatz oder Nachfolger für Java hat sich keine der beiden Sprachen durchsetzen können. Immerhin haben sie durch ihre mächtigere Syntax und den größeren Funktionsumfang Java vor sich hergetrieben. Das mündete letztendlich mit Version 8 im größten Update der Java-Geschichte. Und auch James Strachan sieht man in letzter Zeit wieder häufiger wohlwollend über sein frü-

heres Baby (Groovy) twittern. Groovy lässt sich auf vier Hauptprinzipien reduzieren: Feature-Reichtum, Java-freundlich, dynamisches Groovy und robuste Plattform als Basis.

Feature-Reichtum: Auf der einen Seite bietet Groovy eine sehr ausdrucksstarke Syntax. Hat man sie einmal kennen und lieben gelernt, vermisst man sie schnell in reinen Java-Projekten. Zu nennen wären da stellvertretend die native Unterstützung für Collections inklusive der Wertebereiche (Ranges), die Verarbeitung von regulären Ausdrücken, GStrings, mehrzeilige Zeichenketten, der Property-Support, das Überladen von Operatoren und vieles mehr. Ebenfalls sehr praktisch sind die eingebauten Verarbeitungsmöglichkeiten für XML und JSON sowie die einfachen Datenbank-Zugriffsmöglichkeiten. Die Laufzeit-Meta-Programmierung (MOP), die Compiletime-Meta-Programmierung mit AST-Transformationen und Traits (Mehrfachvererbung) ermöglichen es, bestehenden Code sehr flexibel zu erweitern. Eine der grundlegenden Eigenschaften sind die anonymen Funktionen (Closures), auf denen viele andere Sprach-Konstrukte aufbauen. Sie geben der Sprache zudem einen funktionalen Anstrich.

Java-freundlich: Durch die große Ähnlichkeit mit der Java-Syntax lässt es sich mit Groovy sehr schnell loslegen, sei es in Tests, Skripten, als DSL (Spock, Gradle), zur Android-Entwicklung oder natürlich in Enterprise-Anwendungen (Grails). Als Superset der Java-Syntax ist gerade für Java-Entwickler die Lernkurve flach und die Migration von bestehenden Anwendungen einfach möglich. Groovy integriert sich nahtlos in Java und wird auch zu Bytecode kompiliert. Zu-

dem können wechselseitig Klassen verwendet werden (Aufruf, Vererbung, Relationen). Dadurch kann Groovy als Basis das JDK wiederverwenden, erweitert es im GDK (Groovy Development Kit) aber auch um viele nützliche Funktionen.

Dynamisches Groovy: Bekannt wurde Groovy als dynamische Skriptsprache. Dieses Image hat in den letzten Jahren allerdings mehr geschadet, denn die dynamische Typisierung und die darauf aufbauende Laufzeit-Metaprogrammierung können so manchen Java-Entwickler überfordern, wenn er nur statisch getypten und damit vom Compiler geprüften Code gewohnt ist.

Als Alternative gibt es mittlerweile aber auch Traits und die Compiletime-Meta-Programmierung mit AST-Transformationen. Damit werden die Erweiterungen schon im Kompilervorgang hinzugefügt, was sich positiv auf die Laufzeit-Performance und die Stabilität gerade von großen Projekten auswirkt. Zudem kann auch noch die statische Typprüfung aktiviert und, sofern gewünscht, der Sourcecode statisch kompiliert werden.

Aber auch die Runtime-Meta-Programmierung hat weiterhin ihre Berechtigung, ermöglicht sie doch gerade für Domain Specific Languages (wie in Grails, Gradle, Spock) oder in kleinen Skripten das Schreiben von kompaktem und Boiler-Plate-freiem Code, der dynamisch zur Laufzeit ganz flexibel um neue Funktionalitäten erweitert wird. Die Builder (für XML, JSON, Swing etc.) oder das Überladen von Operatoren sind typische Beispiele dafür.

Robuste Plattform als Basis: Da der Sourcecode zu Java-Bytecode übersetzt wird, profitiert Groovy als JVM-Sprache von allen Vorteilen der Java-Laufzeit-Umgebung. Das

bedeutet auch, dass die Java-Klassenbibliothek (JDK) und überhaupt alle Java-Bibliotheken einfach wiederverwendet werden können. Als Entwickler muss man zudem nicht umdenken, da das gleiche Speicher-, Threading-, Vererbungs- und Sicherheitsmodell zur Anwendung kommt. Alle lieb gewonnenen Werkzeuge (Build-Tools, IDEs, Profiler, Debugger ...) können meist „1:1“ oder mit kleinen Einschränkungen weiterverwendet werden.

Kritikpunkte

Da wo Licht ist, ist natürlich auch Schatten. Denn gerade wenn man Groovy als General Purpose Language in großen Projekten einsetzt, stößt man schnell auf einige Probleme. Wie so oft gilt: „There is no free lunch“ (Herb Sutter). Man kann nicht die Vorteile von Groovy nutzen, ohne auch einige Kompromisse in Kauf zu nehmen. Diese sind nicht unbedingt als Nachteil zu werten, man muss sich aber mit den möglichen Konsequenzen auseinandersetzen und vor allem verstehen, warum es sich so verhält. Bei Java geht es uns nicht anders, auch da leben wir mit gewissen Nachteilen. Am häufigsten wird bei Groovy die dynamische Typisierung mit all ihren Folgeerscheinungen kritisiert:

- Fehler erst zur Laufzeit
- Fehlende Tool-Unterstützung bei Code-Vervollständigung
- Unsicherheit beim Refactoring
- Schlechtere Performance

Kommt man aus der Java-Welt, ist man natürlich gewohnt, dass der Compiler bereits viele Fehler entdeckt. Beim Refactoring erhält man so direktes Feedback, wenn sich der Code nicht mehr übersetzen lässt. Zudem können durch die statische Kompilierung Optimierungen vorgenommen werden, die die Performance positiv beeinflussen. Aber auch wenn die dynamische Typisierung zunächst ungewohnt ist, so stellt sie doch eine der wichtigsten Funktionen von Groovy dar. Dierk König sagt dazu: „In OO, static types limit your choice. That makes dynamic languages attractive ...“

Gerade bei kleineren Aufgaben (wie Skripten) helfen optionale Typen und Meta-Programmierung dabei, die Intention des Codes zu präzisieren. Vom Ballast befreiter Quellcode ist zudem viel besser verständlich. Das ist nicht zu unterschätzen, wenn man bedenkt, dass man ihn zu 80 Prozent der Zeit liest. Außerdem fällt es einfach leichter, gewisse Design- und Clean-Code-

Prinzipien einzuhalten beziehungsweise umzusetzen.

Die Groovy-Entwickler haben die kritischen Stimmen vernommen und so gibt es seit einiger Zeit Alternativen zur dynamischen Typisierung und zur Laufzeit-Meta-Programmierung. Mit den AST-Transformationen „@TypeChecked“ und „@CompileStatic“ kann man den Groovy-Compiler zu strengen Prüfungen und sogar zur statischen Kompilierung zwingen. An den annotierten Codestellen lassen sich dann allerdings keine dynamischen Fähigkeiten nutzen, die beiden Vorgehensweisen können jedoch innerhalb einer Anwendung kombiniert werden. Und für die Meta-Programmierung stehen mit Traits (Mehrfachvererbung) und AST-Transformationen Compiler-basierte Alternativen zur Verfügung.

Will man weiterhin mit dynamisch typisierten Anteilen umgehen, empfiehlt sich zur Qualitätssicherung natürlich auch eine gute Testabdeckung. In Java-Programmen haben wir diesen Anspruch doch ebenfalls. Groovy zwingt uns durch die fehlende Compiler-Unterstützung also nur stärker zu dieser Vorgehensweise. Gleichzeitig lassen sich in Groovy die Tests einfacher schreiben. Dadurch wird der Test-Code kürzer, ausdrucksstärker und besser verständlich, zudem sind Mocking-Funktionalitäten bereits eingebaut und Tools wie Spock oder Geb verwenden sowieso Groovy-DSLs. Durch eine hohe Testabdeckung werden auch die Fehler aufgedeckt, die in einer statisch getypten Sprache sonst durch den Compiler gefunden worden wären. Zusätzlich werden aber auch viele Szenarien getestet, bei denen der Compiler nicht helfen würde. Das klingt doch irgendwie nach einer Win-Win-Situation.

Durch die statische Kompilierung entfallen auch die Performance-Nachteile, da der erzeugte Bytecode ähnlich optimiert werden kann wie der von Java. Aber auch bei dynamisch typisiertem Groovy-Code wurden in den letzten Versionen immer wieder kleinere Optimierungen vorgenommen, sodass sich die Situation in den vergangenen Jahren stetig verbessert hat. Der größte Geschwindigkeitszuwachs wird die noch ausstehende Überarbeitung des Meta Object Protocol auf Basis von Invoke Dynamic bringen. Leider ist aktuell nicht klar, ob und wann dieser Umbau erfolgen wird. Allerdings geht es beim Großteil der Anwendungsfälle selten um das Herauskitzeln der letzten Millisekunden; die Flaschenhalse liegen oftmals an ganz anderen Stellen (wie Datenbank-Abfragen) und für die Implementierung von Real-Time-

Systemen wird man Groovy eher nicht verwenden.

Groovy hebt sich von Java nicht nur durch die dynamischen Fähigkeiten, sondern auch durch viele kleine nützliche Funktionen ab. Man könnte jede Menge nennen, stellvertretend sind hier die inoffiziellen Top 5 kurz an Beispielen illustriert (siehe Listing 1):

- Mehrzeilige Strings und Zeichenketten mit Platzhaltern
- Elvis-Operator
- Vereinfachte Objekt-Navigation und sicheres De-Referenzieren
- Eingebauter Support für das Lesen und Schreiben von Datenformaten (XML, JSON etc.)
- Power-Asserts

Das war jedoch nur die Spitze des Eisbergs, genauso erwähnenswert wären zum Beispiel:

- Multiple Assignments
- Null Object Pattern
- Spaceship-Operator
- Diverse GDK-Methoden und -Operatoren
- Diverse AST-Transformationen
- Dynamische Method-Interception
- Default-Parameter in Methoden

Eine Aufstellung dieser Hidden Features steht unter [2]. Aufgrund der Mächtigkeit der Sprache darf ein Punkt jedoch auch nicht vergessen werden: „With great power comes great responsibility“. Man kann Dinge machen, von denen Java-Entwickler nur träumen können. Aber gerade deshalb sollte man Vorsicht walten lassen, insbesondere in gemischten Teams mit reinen Java-Entwicklern. Was für einen Groovy-Fan elegant aussieht, ist für einen anderen Kollegen ein Buch mit sieben Siegeln. Deshalb sollte man zum Beispiel einheitliche Code-Konventionen festgelegt. Über den Einsatz statischer Code-Analyse-Tools (CodeNarc) und die statische Typprüfung empfiehlt es sich ebenfalls nachzudenken.

Neuerungen der letzten Releases

Dass Groovy kein One-Hit-Wonder ist, sieht man an den regelmäßig veröffentlichten Versionen (mindestens einmal pro Jahr). Da die Sprache mittlerweile mehr als zwölf Jahre alt ist, darf man allerdings keine bahnbrechenden Neuerungen mehr erwarten. Es gibt jedoch stetig kleine Fortschritte, um die Verwendung einfacher und effizienter zu machen. Das letzte große Release war

```
String s = """"Dieser GString
reicht über mehrere Zeilen und
kann Platzhalter verwenden: ${person.foo}""""

// von if/else zum Elvis-Operator
def username = getUsernameFromAnywhere()
if (username) { username } else { 'Unknown' }
username ? username : 'Unknown'
username ?: 'Unknown'

// Null-Safe Dereferenz Operator
bank?.kunden?.konten?.guthaben?.sum()

// XML parsen
def text = '''
<list>
  <technology>
    <name>Groovy</name>
  </technology>
  <technology>
    <name>Grails</name>
  </technology>
</list>'''

def list = new XmlSlurper().parseText(text)
assert list.technology.size() == 2

// Power Asserts
groovy> a = 10
groovy> b = 9
groovy> assert 89 == a * b
Exception thrown

Assertion failed:

assert 89 == a * b
      |  |  |  |
      | 10 | 9
      |  |  |
      false
```

Listing 1

Version 2.4 Anfang 2015. Danach gab es nur noch kleinere Bugfix-Releases, bei denen sehr viel Aufwand in die Erfüllung der Bedingungen der Apache Software Foundation gesteckt werden musste. Die markantesten Neuerungen der letzten drei Minor-Releases waren:

- Neue AST-Transformationen
- Traits
- Android-Support

Der Groovy-Compiler bietet bereits seit der Version 1.6 die Möglichkeit, in den Kompilierungsprozess einzugreifen, um beispielsweise zusätzliche Anweisungen einzufügen oder zu löschen („AST“ steht für „Abstract Syntax Tree“). Damit können klassische Querschnittsprobleme wie Autorisierung, Tracing oder die Transaktionssteuerung gelöst werden, ohne den Sourcecode mit unnötigem Ballast aufzublähen. Vergleichbar ist dieser Mechanismus mit der Vorgehensweise der Java-Bibliothek Lombok, bei Groovy ist er allerdings direkt in die Sprache integriert.

Regelmäßig kommen neue Implementierungen von AST-Transformationen hinzu. Eigene zu schreiben, ist auch möglich. Man unterscheidet zwischen globalen und lokalen AST-Transformationen, bei letzteren werden die anzupassenden Codestellen durch Annotationen markiert. Der Compiler kann dann an diesen Stellen zusätzliche Informationen in den Bytecode generieren. Der Sourcecode bleibt so schlank, also gut les- und wartbar.

```
@Builder
class Person {
  String firstName, lastName
  int age
}

def person = Person.builder()
  .firstName("Dieter")
  .lastName("Develop")
  .age(21)
  .build()

assert person.firstName == "Dieter"
assert person.lastName == "Develop"
```

Listing 2

Da die generierten Informationen im Bytecode landen, können aber auch andere (Java-) Klassen diese Funktionalitäten aufrufen. Es folgt mit dem Builder Design Pattern ein Beispiel für eine lokale AST-Transformation. Die Implementierung der Builder-Klasse existiert dabei nur im Bytecode (siehe Listing 2).

Neben dem Builder gibt es eine große Anzahl von fertigen AST-Transformationen [3] und es kommen ständig neue dazu. Die Transformationen können in verschiedene Kategorien einsortiert werden: Code-Generation („@ToString“, „@EqualsAndHashCode“ etc.), Klassen-Design („@Immutable“, „@Singleton“ etc.), Logging-Erweiterungen („@Log“, „@Log4j“ etc.), deklarative Nebenläufigkeit („@Synchronized“ etc.), Compiler-Hinweise („@TypeChecked“, „@CompileStatic“, „@PackageScope“ etc.), Auflösen von Abhängigkeiten (Dependency Management mit „@Grab“) und viele mehr.

In der Dokumentation finden sich weitere Informationen. Auch Grails macht mittlerweile starken Gebrauch davon und kann seine Magie so sehr stabil und auch performant umsetzen. Das Schreiben eigener Transformationen ist übrigens nicht ganz trivial, es gibt jedoch einige Tools (wie AST-Browser) und mittlerweile auch Tutorials, die bei der Umsetzung unterstützen.

Traits wurden Anfang 2014 in Version 2.3 in Groovy eingeführt. Seit Version 8 gibt es mit Default-Methoden in Java eine vergleichbare, aber nicht ganz so mächtige Alternative. Letztlich handelt es sich um einen Mehrfachvererbungs-Mechanismus; in Groovy gab es mit der AST-Transformation „@Mixin“ bereits seit Version 1.6 eine Vorgänger-Umsetzung. Diese hatte allerdings einige Probleme und so wurde das Konzept mit Traits direkt in die Sprache integriert. Das Ziel ist es, einzelne Funktionalitäten in eigene Interfaces herauszuziehen („Sing-

```
trait Fahrbar {
    int geschwindigkeit
    void fahren() {
        "println Fahren mit ${geschwindigkeit} km/h!"
    }
}
```

Listing 3

```
class Bobbycar implements Fahrbar {
}
new Bobbycar(geschwindigkeit:100).fahren()
```

Listing 4

le Responsibility“-Prinzip) und später über Vererbung wieder zu kombinieren.

Bis zur Einführung der Default-Methoden war in Java keine Mehrfachvererbung möglich – genau genommen auch nicht gewollt. In Java 8 wurde sie aus der Not heraus hinzugefügt. Durch das Stream-API mussten die Interfaces des Collection-Frameworks erweitert werden. Ohne Default-Methoden wären alle bisherigen Implementierungen kaputtgegangen, die Abwärtskompatibilität wäre gebrochen gewesen. Traits können im Gegensatz zu Java-Interfaces nicht nur Methoden, sondern auch Zustände enthalten (siehe Listing 3). Subklassen können dann von beliebig vielen Traits ableiten und erben sowohl die Variablen als auch die Methoden (siehe Listing 4).

Wie bei Interfaces üblich, dürfen Traits abstrakte Methoden definieren, sodass Subklassen diese überschreiben müssen. Das kann beispielsweise für die Implementierung des Template Method Pattern interessant sein. Dass in Java lange Zeit keine Mehrfachvererbung angeboten wurde, hing in erster Linie mit der Angst vor Konflikten in der Vererbungshierarchie zusammen. Das allgemein bekannte Diamond-Problem wird in Groovy aber relativ einfach umgangen (siehe Listing 5).

Es gewinnt nämlich automatisch das zuletzt in der „implements“-Liste stehende Trait, wenn die aufzurufende Methode ermittelt wird. Möchte man die Auswahl steuern, kann man entweder die Reihenfolge anpassen oder explizit den Super-Aufruf implementieren (siehe Listing 6).

Traits können übrigens auch ganz dynamisch zur Laufzeit von Instanzen einer Klasse implementiert werden. Dann ist man zwar wieder bei der langsameren Runtime-Meta-Programmierung, kann aber sehr flexibel bestehende Objekte um neue Funktio-

nalitäten ergänzen. Übrigens kann ein Trait mit einer abstrakten Methode (SAM-Type) auch über eine Closure instanziiert werden, ähnlich wie das bei Java 8 mit Lambda-Ausdrücken und Functional Interfaces der Fall ist. Weitere Beispiele finden sich unter [4].

Seit Anfang 2015 (Version 2.4) lassen sich Android-Apps mit Groovy entwickeln. Android-Entwickler profitieren dabei auch wiederum von Groovys prägnanter Syntax und den Meta-Programmierungsmöglichkeiten. In Java entwickelte Android-Anwendungen enthalten typischerweise viel Boiler Plate Code. Zudem stehen die modernen Errungenschaften aus Java 8 noch nicht zur Verfügung. Groovy bringt mit der SwissKnife-Bibliothek zudem einige AST-Transformationen mit, sodass häufig wiederkehrende Funktionen (Behandeln von UI-Events, Verarbeitung in Background-Threads) sehr schlank implementiert werden können.

Wir haben gesehen, dass es in Groovy in den vergangenen Jahren keine technischen Revolutionen gab. Das liegt hauptsächlich an der Ausgereiftheit der Sprache, die nur wenige Wünsche offenlässt. Trotzdem wurden evolutionsartig kleine Verbesserungen und Neuerungen eingebaut. Wie sich das unter der Obhut von Apache ohne kommerzielle Firma im Rücken entwickeln wird, muss sich noch zeigen. Ganz anders ist es bei Grails verlaufen. Da gab es mit Version 3.0 im Frühjahr 2015 geradezu ein Feuerwerk an Änderungen, damals noch unter der Ägide von Pivotal. Aber durch die Übernahme durch die Firma OCI ist weiterhin Zug in dem Open-Source-Projekt, wie auch das Release 3.1 und die in der Roadmap [5] geplanten Versionen beweisen.

Im Jahre 2005 erblickte Grails das Licht der Welt und war letztlich die Antwort der Groovy-Welt auf das Web-Application-Framework Ruby on Rails. In Anlehnung da-

ran wurde ursprünglich der Name gewählt („Groovy auf Schienen“), durch die Abkürzung ist man nun aber eher auf der Suche nach dem heiligen Gral.

Grails ist geprägt von einigen Prinzipien für sauberes, objektorientiertes Software-Design wie „DRY“ (Don't Repeat Yourself), „SRP“ (Single Responsibility) und „CoC“ (Convention over Configuration). Das heißt zum Beispiel, dass statt einer komplexen Konfiguration einfach Konventionen für die Namensgebung von Objekten einzuhalten sind, aus denen sich das Zusammenspiel zur Laufzeit automatisch ergibt. Diese Eigenschaften ermöglichen eine rasche Umsetzung von Anforderungen, wodurch Grails seit jeher ein beliebtes Prototyping-Framework ist, das zudem auch optisch sehr ansprechende Ergebnisse liefert. Vor mehr als zehn Jahren hatten die klassischen Java-Webanwendungen einige Tücken:

- Aufwändiges Editieren von Konfigurationsdateien
- Notwendige Anpassungen des Deployment-Deskriptors („web.xml“)
- Aufwändige Konfiguration der Abhängigkeiten
- Umständliche Build-Skripte
- Primitive Template- und Layout-Mechanismen
- Neustarten der Anwendung beziehungsweise des Webcontainers nach Änderungen

```
trait A {
    String exec() { 'A' }
}

trait B extends A {
    String exec() { 'B' }
}

trait C extends A {
    String exec() { 'C' }
}

class D implements B, C {}

def d = new D()
assert d.exec() == 'C'
```

Listing 5

```
class D implements B, C {
    String exec() { B.super.exec() }
}

def d = new D()
assert d.exec() == 'B'
```

Listing 6

Mit Grails wurde den Entwicklern die Arbeit deutlich erleichtert. Aus dem ursprünglichen Einsatzzweck für CRUD-Webanwendungen hat es sich mittlerweile zu einem gestandenen Fullstack-Web-Application-Framework gemauert. Zu den Kernprinzipien zählen die gesteigerte Produktivität, die einfachere Entwicklung, die Erweiterbarkeit und die ausgereiften Bibliotheken als stabile Basis.

Gesteigerte Produktivität

Grails eignet sich wunderbar für das Bauen von Prototypen. Durch sinnvoll gewählte Grundeinstellungen und einfache Konventionen lassen sich Redundanzen vermeiden und in kurzer Zeit beachtliche Ergebnisse erzielen. Hinzu kommen die mittlerweile auf Gradle aufbauende Entwicklungskonsole, die einheitliche Projektstruktur, die durch Metaprogrammierung hineingemixten Enterprise-Funktionalitäten und die Verwendung von Groovy.

Die Einfachheit ergibt sich durch die prägnantere Syntax von Groovy, die einheitliche Persistenz-Schnittstelle (GORM, ursprünglich für Objekt-relacionales Mapping erfunden, mittlerweile aber auch für NoSQL-Datenbanken geeignet), die wohldefinierte Projektstruktur mit hohem Wiedererkennungseffekt, Verzicht auf unnötige Konfigurationen, die automatische Injektion von fertig mitgelieferten Enterprise-Services und die Validierung von Daten anhand flexibler und leicht zu erweiternder Regeln. Grails folgt dem KISS-Prinzip („keep it simple and stupid“) mit dem Ziel, unnötigen Boiler-Plate-Code zu vermeiden. Zudem kann per Scaffolding statisch oder dynamisch ein initiales Code-Gerüst generiert werden, das bereits die CRUD-Funktionalitäten vom Frontend bis zur Datenspeicherung abdeckt und sogar Restful-Schnittstellen anbietet. Man kommt somit in wenigen Minuten zu einer lauffähigen Applikation. Nur bei der Geschäftslogik kann und muss letztlich noch Hand angelegt werden.

Die Erweiterbarkeit zeigt sich durch den modularen Aufbau von Grails mit der Austauschbarkeit von einzelnen Komponenten (Hibernate/RDBMS \Leftrightarrow NoSQL, Tomcat \Leftrightarrow Jetty etc.). Zudem kann der Kern von Grails über Plug-ins beliebig um weitere Funktionalitäten ergänzt werden. Auch die eigene Anwendung lässt sich über Plug-ins sauber strukturieren und einzelne Teile sind entsprechend leichter wiederverwendbar. Aufbauend auf den gleichen Ideen wie dem Java EE Standard (Integration von Spring und Hi-

bernate) bringt es zudem seine Ablaufumgebung in Form eines Webcontainers direkt mit und war durch die Entwicklungskonsole seit jeher quasi eine Art Vorgänger der heute so stark beworbenen Self-Contained-Systems. Da die aktuelle Version auf Spring Boot aufsetzt, können mittlerweile auch direkt ausführbare Fat-JARs erzeugt werden.

Stabile Basis

Grails ist in erster Linie eine Menge Gluecode um bestehende mächtige Bibliotheken wie Spring, Hibernate und Sitemesh. Durch die Entwicklung in Groovy kann man den syntaktischen Zucker von Grails voll ausnutzen. Als Build-Artefakt wird ein klassisches Java WAR erzeugt, das in Webcontainern eingesetzt beziehungsweise mittlerweile als Fat-JAR einfach ausgeführt werden kann. Leider ist nichts perfekt und so werden auch an Grails immer wieder Kritikpunkte laut:

- Problematisch in großen Projekten
- Gebrochene Abwärtskompatibilität und großer Aufwand bei Upgrades
- Aktualität der Plug-ins
- Unleserliche Stack-Traces

Auch wenn einige Erfolgsgeschichten für große Projekte existieren, eignet sich Grails eher für überschaubare Projekte mit kleineren und möglichst homogenen Teams (etwa für Microservices). Gerade ein Mix von erfahrenen Java- und enthusiastischen Groovy-Entwicklern kann schnell zu Konflikten und auf Dauer zu Unverständnis führen. Aus eigener Erfahrung empfiehlt es sich auf jeden Fall, projektübergreifende Code-Konventionen (Typ-Angaben hinschreiben, Verwendung von „return“ etc.) festzulegen, damit auch die Java-Entwickler den Groovy-Code lesen und ändern können. Zudem ist eine gute Testabdeckung wichtig, um die Auswirkungen der dynamischen Funktionalitäten abfangen zu können und bei Refactorings wenigstens durch die Testläufe Feedback zu erhalten.

Letztlich gilt wie so oft das Pareto-Prinzip. 80 Prozent der Anwendung (zumeist CRUD) lassen sich in Grails sehr einfach umsetzen. Die letzten 20 Prozent brauchen dann jedoch auf einmal einen Großteil der Zeit und erscheinen im Vergleich umständlicher, aber natürlich nicht unlösbar. In Java fällt das nur nicht so auf, weil schon die ersten 80 Prozent deutlich mehr Aufwand bedeuten.

Die mangelnde Abwärtskompatibilität schlägt sich meist darin nieder, dass man ei-

nigen Aufwand investieren muss, um bestehende Anwendungen auf die neueste Grails-Version zu aktualisieren. Geschuldet ist dies dem Innovationsdrang der Grails-Entwickler, die neuesten Techniken und Ideen rasch zu integrieren. Das wird umso aufwändiger, je länger man wartet. Darum sollte man immer frühzeitig updaten. Die Grails-Dokumentation unterstützt durch einen Migrationsguide. Und hat man doch mal einige Versionsupdates ausgelassen, empfiehlt es sich, alle Änderungen Schritt für Schritt und von Version zu Version abzuarbeiten.

Der modulare Aufbau von Grails hat zu einem riesigen Plug-in-Ökosystem geführt [6, 7]. Leider schwankt die Qualität sehr stark. Guten Gewissens kann man die Plug-ins der Core-Entwickler einsetzen, sie werden bei neuen Grails-Versionen zeitnah angepasst und regelmäßig gewartet. Erkennen kann man sie über positive Bewertungen des Voting-Systems und durch eine große Anzahl von aktiven Nutzern. Bei anderen, oft von Einzelkämpfern erstellten Modulen gilt es zunächst, einen Blick auf die Aktualität des Sourcecodes (letzte Commits), die Antworthaftigkeit bei Fragen und die Bearbeitung der Tickets zu werfen. Leider existieren nämlich einige Karteileichen, die durch die hochfrequente Update-Politik mit den aktuellen Versionen gar nicht mehr funktionieren. Da bleiben dann bei Interesse nur der Fork und die eigene interne Pflege des Plug-ins oder natürlich eine Weiterentwicklung des Originals, sofern man sich das zutraut.

Aufgrund des großen Funktionsumfangs fällt es schwer, die wichtigsten Grails-Features zu benennen. Es gibt einfach zu viele nützliche Eigenschaften. Zudem setzt jeder Entwickler auch andere Prioritäten. Folgende Punkte sollen aber stellvertretend für viele andere stehen:

- Automatische Dependency Injection (etwa durch Namenskonventionen)
- Validierung der Domänen-Klassen (ähnlich wie Java-EE-Validation-API)
- Sehr einfache, aber trotzdem mächtige Erweiterung durch Tag-Libs
- Automatisch generierte CRUD-Methoden
- Viele Persistenz-Abfrage-Varianten wie die gut lesbaren „Where“-Queries

Bei jedem dieser Punkte könnte man wieder ins Detail gehen. An dieser Stelle sei aber auf die Dokumentation verwiesen. In-

interessanter sind die neuesten Funktionen des letzten Release. Neue Minor-Versionen kommen ähnlich wie bei Groovy mindestens einmal pro Jahr heraus. Das letzte Major-Update (3.0) wurde im Frühjahr 2015 veröffentlicht. Die Liste der Neuerungen ist lang und beeindruckend. Die Maßnahmen versprechen eine verbesserte Stabilität und leichtere Integration in bestehende Systemlandschaften:

- Spring Boot als Basis
- Neues Interceptor-API
- Anwendungsprofile
- Gradle als Build-System
- API-Redesign

Über Spring Boot braucht man mittlerweile nicht mehr viel zu schreiben. Viele seiner Eigenschaften hatte Grails in ähnlicher Form schon sehr lange im Portfolio. Um sich aber den unnötigen Pflegeaufwand zu sparen, baut man nun auf dem vielfach bewährten Framework auf. Von Spring Boot profitiert Grails besonders durch die Einbettung des ausführenden Containers, der in einem lauffähigen JAR inkludiert wird. Damit kann man Grails-Anwendungen tatsächlich überall laufen lassen, es muss nur Java installiert sein. In der Entwicklung ist man zudem nicht mehr auf eine speziell angepasste IDE angewiesen, da eine Grails-Anwendung nun wie jede beliebige Java-Anwendung gestartet, auf Fehler überprüft und mit Profilen eingerichtet werden kann.

Das neue Interceptor-API ergänzt beziehungsweise ersetzt die Grails-Filter bei der Entwicklung von Querschnittsbelangen (Security, Tracing, Logging etc.). Gewisse Basis-Funktionalitäten erben Interceptoren von einem Trait. Im Gegensatz zu den Filtern, die ähnlich Servlet-Filtern über Muster auf bestimmte URLs reagieren, werden Interceptoren in der Regel über Namenskonventionen jeweils einem Controller zugeordnet. Gemeinsam ist beiden Ansätzen, dass sie verschiedene Callbacks („before“, „after“ und „afterView“) anbieten, über die Querschnittsfunktionalität implementiert wird. Interceptoren sind dazu noch kompatibel mit der statischen Kompilierung („@CompileStatic“), ein nicht zu verachtender Performance-Vorteil. Schließlich werden sie potenziell bei jedem Request aufgerufen. Weitere Informationen finden sich unter [8].

Eine andere Neuerung ist der Support für Anwendungsprofile. Gemeint ist hier die Struktur, die beim Anlegen eines neuen

Grails-Projekts gleich auf das gewünschte Szenario (Web-, REST-, RIA-Anwendung etc.) zugeschnitten werden kann. Es ist ein bisschen vergleichbar mit den Java-EE-Profilen („Web“ und „Full“), geht aber noch einige Schritte weiter. So kapselt ein Profil die aufrufbaren Kommandos, Plug-ins, Skeletons/Templates und Ressourcen. Der Default ist übrigens weiterhin die klassische Web-Anwendung. Man kann auch eigene, firmenspezifische Profile anlegen, die dann in einem Repository (im User-Verzeichnis) gespeichert werden.

Grails hatte seit jeher ein auf Groovy, Ant und Ivy basierendes, selbst entwickeltes Build-Management, auf dem auch die Kommandos für die Entwicklungskonsole aufgebaut haben. Aber gerade die Integration in eine bestehende Systemlandschaft mit klassischen Java-Build-Tools wurde dadurch unnötig verkompliziert. Zudem war das Bauen mit Gant immer etwas problematisch. Seit Grails 3 wird als Build-System Gradle verwendet. Somit ist man auch hier nicht mehr abhängig von speziell integrierten Funktionen der IDEs. Das erleichtert den Abschied von der Eclipse-basierten „Groovy und Grails Toolsuite“, die seit dem Rückzug durch Pivotal nicht mehr weiterentwickelt wird. Es reicht theoretisch wieder ein einfacher Text-Editor mit der Kommandozeile, so wie Grails bereits vor mehr als zehn Jahren gestartet ist.

In Grails passieren viele Dinge automatisch, manchmal magisch. Das wurde in den älteren Versionen vor allem durch Laufzeit-Meta-Programmierung erreicht, war jedoch immer wieder die Quelle für Probleme und Instabilitäten im Entwicklungsprozess. Mittlerweile setzt man bei der Erweiterung von Domain-Klassen, Controllern, Services, Tag-Libs und Tests verstärkt auf Compiletime-Meta-Programmierungsmechanismen mit AST-Transformationen und auf Traits. Im Rahmen dieser Umbauarbeiten wurden gleich das API umstrukturiert und eine saubere Trennung in öffentliche („grails.*“) und private/interne Pakete („org.grails.*“) vorgenommen.

Die stetig steigenden Downloadzahlen zeigen das wachsende Interesse. Wer nach dem hier Gelesenen (erneut) Lust auf Groovy und Grails bekommen hat, sollte sich selbst ein Bild machen, die neuesten Versionen installieren und einfach mal loslegen. Die Online-Dokumentation hilft beim Starten und zudem gibt es mittlerweile jede Menge Literatur und Tutorials zum Thema.

Referenzen

- [1] Blog James Strachan: <http://macstrac.blogspot.de/2009/04/scala-as-long-term-replacement-for.html>
- [2] Groovy Hidden Features: <http://stackoverflow.com/questions/303512/hidden-features-of-groovy>
- [3] Available AST transformations: http://groovy-lang.org/metaprogramming.html#_available_ast_transformations
- [4] Traits: <http://blog.oio.de/2014/07/04/unterstuetzung-von-traits-ab-groovy-2-3/>
- [5] Grails Roadmap: <https://github.com/grails/grails-core/wiki/Roadmap>
- [6] Grails-Plug-ins bis 2.x: <https://grails.org/plugins>
- [7] Grails-Plug-ins ab 3.0: <https://bintray.com/grails/plugins>
- [8] Grails Interceptor: <http://www.ociwweb.com/resources/publications/sett/september-2015-grails-3-interceptors>

Falk Sippach
falk.sippach@oio.de



Falk Sippach hat mehr als fünfzehn Jahre Erfahrung mit Java und ist bei der Mannheimer Firma OIO Orientation in Objects GmbH als Trainer, Software-Entwickler und Projektleiter tätig. Er publiziert regelmäßig in Blogs, Fachartikeln und auf Konferenzen. In seiner Wahlheimat Darmstadt organisiert er mit Anderen die örtliche Java User Group. Falk Sippach twittert unter @sipsack.

PL/SQL

PL/SQL2Java – was funktioniert und was nicht

Stephan La Rocca, PITSS GmbH



Immer mehr Projekte beschäftigen sich mit der Migration von Oracle Forms hin zu ADF oder vielleicht auch zu Open-Source-basierten Java-Lösungen. Um den Invest in die jahrelange Forms-Entwicklung sichern zu können, wird immer wieder diskutiert, mögliche PL/SQL-basierte Logik in die Datenbank zu transferieren. Doch dieser Ansatz ist nicht unumstritten.

Es betrifft nicht nur die Forms-Applikationen – viele Oracle-basierte Lösungen tragen einen hohen Invest von PL/SQL in sich und der Feingeist eines puristischen Java-Architekten verlangt in einem MVC-Pattern diese Implementierung nicht im Datastore, sondern vielmehr im Model-Projekt der Applikation. Nun stellt sich die Frage, ob eine Re-Implementierung mit allen Aufwänden für die Modellierung, das Schreiben des Codes und die Testverfahren sowie Hürden in Qualität und Verständnis dem Puristen am Ende noch immer ein Lächeln entlockt, wenn er neben dem Code den Blick auf das Budget wirft.

Was kann helfen?

Gehen wir zunächst davon aus, dass mehr als 80 Prozent der fachlichen Anforderungen und der Funktionalität der Anwendung erhalten bleiben sollen, sodass Hilfsmittel bei der Migration sinnvoll erscheinen. Eine Überführung von PL/SQL nach Java muss berücksichtigen, dass es sich bei den beiden Sprachen um Vertreter völlig differenter Programmier-Paradigmen handelt: die strukturierte, modulare Herangehensweise von PL/SQL auf der einen und der objektorientierte Ansatz von Java auf der anderen Seite.

Eine Annäherung für eine Überführung kann nun auf der syntaktischen und auf der Modellierungsebene stattfinden. Bei der syntaktischen Überführung werden wir im weiteren Verlauf erkennen, dass ein Automatismus sehr gut helfen kann. Die Modellierungsebene hingegen beschäftigt sich mit dem Vergleich, wie Daten und die darauf angewandten Methoden in PL/SQL und Java abgebildet werden.

Wie Modellierungsebenen zusammenkommen

Allein schon einen gemeinsamen Wort für die Verteilung der Implementierung innerhalb von PL/SQL und Java zu finden, ist schwierig und zeigt den grundlegenden Unterschied. Sind einem in PL/SQL Prozeduren, Funktionen, Trigger und Packages vertraut, wird auf der anderen Seite in Eigenschaften, Metho-

den, Vererbung und Polymorphie gedacht. Dazwischen existiert keine Transformation, die das PL/SQL-Modell in Java überführt.

Das schließt allerdings nicht aus, dass in PL/SQL bereits objektorientiert entwickelt werden kann. Durch die konsequente Verwendung von „TYPE“-Objekten in der Implementierung können die Paradigmen zu Objekten, Instanzen, Methoden und Vererbung konsequent genutzt werden. Allerdings ist dieser Ansatz nicht weit verbreitet und führt an manchen Stellen doch zu Performance-Problemen, da nicht immer ein effizienter, mengenorientierter Ansatz für Select- oder DML-Statements zur Anwendung kommen kann.

Um auf einem tabellenbasierten Datenmodell ohne Objekt-Typen in der Datenbank dennoch die Struktur von Instanzen und Methoden zu empfinden, besteht die Möglichkeit, durch Packages diese Struktur zu simulieren. Auf erster Ebene wird ein Package („Table“-API) erstellt, das die grundlegenden DML-Statements und Getter-Funktionen anbietet. Parameter sind in der Regel entweder der Oracle-„ROWTYPE“ der Tabelle oder ausschließlich der Primärschlüssel. So entsteht für jede Tabelle ein eigenes Package. In darauf aufbauenden Packages werden dann semantische Methoden auf den Daten implementiert. Mit dieser Trennung der Packages haben alle Prozeduren und Funktionen in einem Package eine klar umrissene Aufgabe und stellen nicht länger ein Sammelsurium von Prozeduren dar.

Diese Vorüberlegungen helfen, in dem Migrationsansatz die verschiedenen Betrachtungsweisen der Programmiersprachen zusammenzubringen. Ein Refakturieren nach diesen Prämissen kann ein wertvoller Schritt in der Vorbereitung einer Migration sein.

Clientseitiges PL/SQL

Der Fokus soll allerdings nicht allein auf datenbankseitigen PL/SQL-Code beschränkt bleiben, sondern auch die Möglichkeit aufzeigen, clientseitige Komponenten zu betrachten. Als prominentester Vertreter für clientseitige PL/SQL-Applikation sei hier die

Entwicklungsplattform Oracle Forms herangezogen. Auch hier können in Triggern, Funktionen, Prozeduren und Packages PL/SQL-Artefakte erstellt werden.

Betrachtet man auf dieser Seite einen Architektur-Vergleich, so ist bei den Java-basierten Web-Anwendungen ein Model-View-Controller-Konzept zu berücksichtigen, das Oracle Forms fremd ist. Da innerhalb von Forms bei der Programmierung von PL/SQL keinerlei Unterscheidung unterstützt wird, ist es nicht unüblich, in einem Programmblock Statements zu finden, die später in allen drei verschiedenen Ebenen des MVC-Pattern positioniert werden müssten. Vereinzelt, etwa bei Transaktions-Triggern, ist hier eine Zuordnung einfach möglich, bei der Großzahl der Units wird jedoch eine manuelle Zuordnung notwendig sein.

Der Versuch eines Refactoring in diesem Prozess muss allerdings nicht nur den Kontext des aktuellen PL/SQL-Blocks berücksichtigen, sondern auch die zur Verfügung stehenden Klassen und Methoden innerhalb des Ziel-Kontextes. Bei einer geplanten Migration nach ADF gibt es im Model-Projekt bestehende Klassen und Methoden, beispielsweise um über das Application Model auf Entitäten zugreifen zu können, oder im ViewController-Projekt existieren etablierte Verfahren, um auf die Bindings der DataControls zuzugreifen.

Ein anschließendes Refakturieren im ADF Workspace, also ein Verschieben einer doch datenintensiveren Methode aus dem ViewController-Projekt in das Model-Projekt, geht einher mit der Verwendung anderer Framework-Extensions und Utility-Klassen innerhalb des Frameworks. Wie wir später noch sehen werden, kann dieser Schritt durch einen Parser, der den Zielkontext berücksichtigt, unterstützt werden.

Die syntaktische Überführung

Nach diesen vorgelagerten Überlegungen zur Architektur und der Idee möglicher Refakturierungsschritte in der Legacy-Applikation betrachtet der zweite Teil mehr den syntaktischen Teil der Migration. An erster Stelle bei einer Zerlegung einer Programmiersprache in

ihren Sprachschatz und die Grammatik steht die Verwendung eines Parsers, der in der Lage ist, bestehende Code-Fragmente über eine Metasprache zu beschreiben.

Für PL/SQL gibt es eine Reihe von verfügbaren Parsern, die diese Aufgabe erledigen können. Für eine komplette Analyse allerdings des Forms-seitigen Codes muss der Parser einen deutlich größeren Sprachschatz berücksichtigen. Dazu gehört neben PL/SQL – und damit verbunden natürlich auch SQL – ebenso die Möglichkeit, Oracle-Forms-Built-ins und darin Embedded PL/SQL-Konstrukte zu analysieren. Ergebnis einer solchen Analyse ist eine semantische Zerlegung der Operatoren und Operanden in einen Graphen (siehe Listing 1 und Abbildung 1).

Mit dieser Zerteilung als Graph besteht nun die Möglichkeit, sich den einzelnen Komponenten und deren Transformation von PL/SQL nach Java zu widmen. Der erste Schritt betrachtet die Transformation der verwendeten Datentypen von PL/SQL nach Java. Hier kann eine einfache Gegenüberstellung der verfügbaren PL/SQL-Datentypen, also jener, die als mögliche Variablen-Typen zum Einsatz kommen können, und den Repräsentanten in Java getroffen werden.

Bei den Statements sind alle SQL- und PL/SQL-Statements berücksichtigt. Schwierig und daher nicht im Fokus waren Verwendungen von „GOTO“-Statements. Diese sinnvoll in eine Java-Methode zu überführen, war aufgrund der hoffentlich geringen Anzahl dieses „Pattern“ nicht im Verhältnis zum Aufwand.

Spannend in der ersten Betrachtung eines kleinen Code-Fragments ist die unterschiedliche Behandlung von „NULL“-Statements. Während es in PL/SQL gang und gäbe ist, Null-Werte für Variablen zuzulassen, und jede Operation mit einem Null-Wert automatisch auch in einen „NULL“-Wert für das Ergebnis führt, lässt sich ein überführter Java Code zwar kompilieren, wirft aber dann in der Ausführung eine „Null Pointer Exception“. Listing 2 würde in einer Konvertierung zu Listing 3. Um diesen „NullPointer“ zur Laufzeit zu vermeiden, besteht die Möglichkeit, vor jedem Operanden automatisch eine explizite Null-Überprüfung vorzuschalten (siehe Listing 4).

Aufgrund der Unübersichtlichkeit und der Tatsache, dass viele solcher Situationen durch Daten aus Datenbank-Tabellen in PL/SQL-gesteuert werden und damit beim Lesen des Source Code die Inhalte der Variablen nicht bekannt sind, wird auf eine durchgängige und sinnvolle Fehlerbehandlung innerhalb der Java-Applikation verwiesen.

```

- - Example 4.9 EXIT Statement
DECLARE
  x NUMBER := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE
      ('Inside loop: x = ' || TO_CHAR(x));
    x := x + 1;
    IF x > 3 THEN
      EXIT;
    END IF;
  END LOOP;
END;
    
```

Listing 1

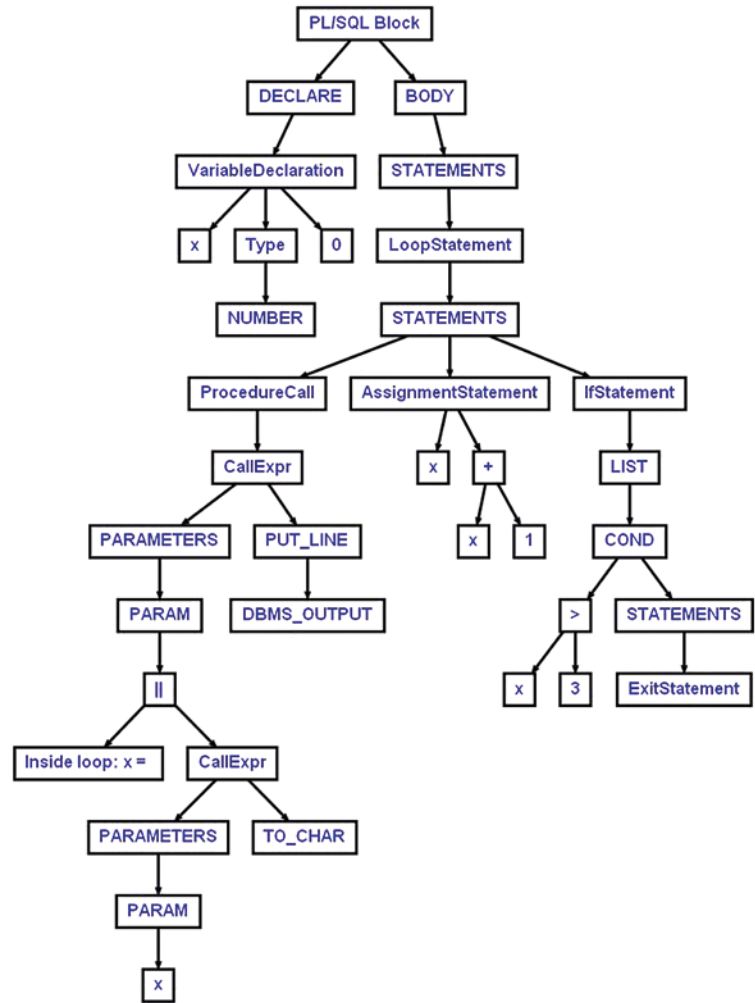


Abbildung 1: Darstellung als Graph

Ein weiterer Aspekt im atomaren Bereich der Statement-Überführung sei an dieser Stelle ebenfalls erwähnt. Während in PL/SQL in der Regel Funktionen mit Rückgabewerten und Parametern erstellt sind, sind Pendanten dazu in Java häufig Methoden des passenden Objekts. Dazu ein Beispiel: Aus Listing 5 wird nach einer Transformation Listing 6. Der implementierte Transformationsprozess überführt also die vorhandenen PL/SQL-Statements in sinnvolle Java-Konstrukte.

DML-Statements

Wenden wir uns im nächsten Schritt den SQL-Statements innerhalb der Programmierung zu. Im Gegensatz zu den einfachen PL/SQL-Statements ist bei den DML-Statements und auch schon bei einzelnen SQL-Statements zu berücksichtigen, dass die spätere Java-Applikation schon in einem Framework eingesetzt wird, das außerhalb der eigentlichen Java-Klasse für Transaktions-Kontext, Datenbank-Sessions, Connection-Verwaltung etc. sorgt.

```

declare
  num1 PLS_INTEGER:=NULL;
  num2 PLS_INTEGER:=5;
  res PLS_INTEGER;
begin
  res:=num1*num2;
  DBS_OUTPUT.PUT_LINE('Result is:'||res);
end;

```

Listing 2

```

public class YourNamingConvention{
  public void anonBlock() {
    Integer num1 = null;
    Integer num2 = 5;
    Integer res;
    res= num1 * num2;
    System.out.println("Result is : " + res);
  }
}

```

Listing 3

```

public class YourNamingConvention{
  public void anonBlock() {
    Integer num1 = null;
    Integer num2 = 5;
    Integer res;
    if (num1 == null || num2 == null) {
      res = null;
    } else {
      res= num1 * num2;
    }
    System.out.println("Result is : " + res);
  }
}

```

Listing 4

```

declare
  str VARCHAR2(80) := 'Hallo DOAG';
  res PLS_INTEGER;
begin
  res:=LENGTH(str);
  DBS_OUTPUT.PUT_LINE('Length is:'||res);
end;

```

Listing 5

```

public class YourNamingConvention{
  public void anonBlock() {
    String str ="Hallo DOAG";
    Integer res;
    res= str.length();
    System.out.println("Length is : " + res);
  }
}

```

Listing 6

```

declare
  dept_no NUMBER(4) := 270;
BEGIN
  DELETE FROM DEPARTMENTS WHERE department_id = dept_no;
  IF SQL%FOUND THEN
    INSERT INTO DEPARTMENTS VALUES (270,'Personnel',200,1700);
  END IF;
END;

```

Listing 7

Für das eigentliche DML-Statement wird also in der Java-Klasse keine Datenbank-Verbindung erstellt, sondern bereits vorhandene Sessions genutzt. Mehr als das, wenn in einem Framework wie ADF bereits bekannt ist, dass das DML-Target bereits als Implementierung im Model-Projekt vorliegt, kann direkt darauf verwiesen werden. Auch dazu noch ein kleines abstraktes Beispiel, das die Vorgehensweise bei der Transformation aufzeichnen soll: *Listing 7* zeigt die simple Ausgangslage, unter der Annahme, Oracle ADF sei Zielkontext mit einem Model-Projekt unter Einhaltung von Namenskonventionen für ViewObjekte und EntityObjekte. *Abbildung 2* zeigt, was daraus erzeugt werden kann.

Die Methode für das Insert-Statement ist ebenfalls in dieser Klasse implementiert, aber aus Platzgründen im Screenshot nicht aufgeführt. Man kann anhand des generierten Codes erkennen, dass auf Basis der Namens-Konvention zur Tabelle „DEPARTMENTS“ ein Objekt „DepartmentsImpl“ innerhalb des Model-Projekts angenommen wird. Diese Namens-Konventionen sind Teil eines Customizing-Prozesses für die Transformation. Dazu später noch weitere Möglichkeiten. Zudem kann man dem Code-Fragment in der Klasse entnehmen, dass auf die ADF-Methoden wie „getWhereClause()“, „executeQuery()“ etc. zurückgegriffen wird.

Im Rahmen der im ersten Abschnitt besprochenen architekturellen Eingliederung von PL/SQL-Sourcen in eine objektorientierte, nach einem MVC-Pattern aufgesetzte Projekt-Landschaft, kann man diese Klasse, oder auch nur die notwendigen Methoden aus dieser Klasse, beispielsweise entweder in die View-Implementierung des passenden View-Objekts überführen oder in die Implementierung des übergeordneten Applikation-Moduls.

Von der Datenbank nach Java

Bei der Überführung dieser Methoden in das Model-Projekt ist es zunächst irrelevant, ob es sich um ursprünglichen, clientseitigen PL/SQL-Code etwa in einer Forms-Applikation handelt oder um datenbankseitigen PL/SQL-Code. Betrachtet man die Möglichkeit, über Table-APIs sowie semantisch zusammenhängende Prozeduren und Funktionen in Oracle-Datenbank-Packages bereits die Zugriffe auf das Datenmodell zu kapseln, so können diese nach der Überführung in ein Java-basiertes Model-Projekt ideal genutzt werden. In einem solchen Environment greifen sowohl alle Konzepte des Session-Pooling als auch die Mög-

```

public class Dummy {
    public void anonBlock() {
        BigDecimal deptNo = BigDecimal.valueOf(270);
        DepartmentsImpl departments = getDepartments();
        int rowsAffected = departments.deleteDepartments(dept_no);
        if (rowsAffected > 0) {
            rowsAffected = departments.insertDepartments(270, "Personnel", 200, 1700);
        }
    }

    public int deleteDepartments(Integer dept_no) {
        String oldWhereClause = getWhereClause();
        setWhereClause("department_id=:dept_no");
        defineNamedWhereClauseParam("dept_no", null, null);
        setNamedWhereClauseParam("dept_no", dept_no);
        executeQuery();
        int rowCount = 0;
        for (Row row = getCurrentRow(); null != row; row = next()) {
            row.remove();
            rowCount++;
        }
        removeNamedWhereClauseParam("dept_no");
        setWhereClause(oldWhereClause);
        return rowCount;
    }
}

```

Abbildung 2: ADF-Implementierung

lichkeiten des Caching auf Application-Model-Ebene. Zudem können darauf idealerweise Web-Services exponiert werden.

Bei den Transformationen ganzer Datenbank-Packages bietet es sich an, aus dem Package eine eigene Java-Klasse zu erstellen und die einzelnen Prozeduren und Funktionen als Methoden anzubieten. Auch in dieser eigenen Java-Klasse werden der Ziel-Kontext ADF und die Namens-Konvention berücksichtigt, sodass man später diese Klasse als eigene Bibliothek in ihrem ADF-Model-Projekt wiederverwenden kann.

Effiziente Anpassungen

Bei einer Transformation steht zu Beginn immer eine Ausgangslage, die in der Regel individuellen Mustern folgt. Bleiben wir in dem Bild von Sprache und Grammatik, so ist der Wortschatz nicht in allen Projekten gleich ausgenutzt und diverse Slangs prägen die geschriebene Geschichte der Implementierung. Diese Slangs lassen sich in einer Analyse der Grundgesamtheit erkennen

und idealerweise zu einem neuen Pattern in der Java-Notation überführen. Typischer Kandidat für solch eine, an der atomaren Ersetzung vorbeigeleiteten Transformation ist die Fehlerbehandlung. Sowohl in der Datenbank als auch auf der Clientseite pflegen die PL/SQL-basierten Applikationen ein festes Muster, um Exceptions abzufangen und darauf innerhalb des Programms zu reagieren. Diese Fehlerbehandlung erfährt in Java eine andere Methode, ja mitunter eine ganz andere Architektur. Daher ist es ratsam, innerhalb einer solchen Transformation diese Statements nicht auf atomarer Ebene zu ersetzen, sondern das Muster als Block zu verstehen und ganzheitlich neu zu betrachten. Listing 8 zeigt ein Beispiel einer Forms-seitigen Fehlersituation, in der die eingeführte und neu implementierte Methode die ursprünglichen Aufgaben übernimmt. Das kann an dieser Stelle wieder ein Zweizeiler sein oder aber auch etwas mehr Logik unter Verwendung von Framework-Klassen. Dem Transformationsprozess wird dann nur beiseite gestellt, dass er diesen Block

```

if STORE_xxx then
    emessage(L_ERROR_MESSAGE);
    RAISE FORM_TRIGGER_FAILURE;
end if;
würde durch eine Erkennung des wiederkehrenden Blockes (Fehlermeldung zeigen - Fehler erzeugen) ein Konstrukt werden können:
if (storeXxx) {
    raiseValidationException(LErrorMessage);
}

```

Listing 8

nach dem Muster unter Berücksichtigung von Regular Expression zu ersetzen hat.

Fazit

Eine Überführung von PL/SQL nach Java findet auf mehreren Ebenen statt. Eine architekturelle Überführung kann durch ein sinnvolles Refactoring auf PL/SQL-Seite vorbereitet werden. Dennoch ist bei der Überführung nach Java zu entscheiden, an welcher Stelle gerade die Methode sinnvoll platziert ist. Dazu sind nicht nur das MVC-Pattern, sondern auch der Namensraum und die Möglichkeiten des Ziel-Kontextes zu berücksichtigen. Im vorliegenden Fall war eine ADF-Web-Anwendung die Ziel-Plattform; das hat unmittelbar Einfluss auf die Transformation der DML-Statements. Auf der syntaktischen Ebene ist eine Transformation nahezu komplett nutzbar. Es gibt nur sehr wenige Konstrukte, die kein Pendant haben und eine individuelle Betrachtung erfordern. Gerade im Bereich der Querschnitts-Funktionalitäten von Fehlerbehandlung, Mehrsprachigkeit und Konfigurationsmanagement gibt es Pattern oder Slangs, die eine effizientere Lösung in Java ermöglichen. Diese können durch Anpassung der Transformation identifiziert und ersetzt werden.

Eine systematische Betrachtung des Ausgangszustands sichert die Qualität, dass keine Implementierung vergessen wird. Darüber hinaus unterstützen auch nur fragmentbasierte Generierungen dabei, dass eine durchgängige Qualität sehr effizient erreicht werden kann.

Stephan La Rocca
slarocca@pitss.com



Stephan La Rocca ist Consulting Manager bei der PITSS GmbH und seit mehr als 17 Jahren als Coach, Consultant und Trainer im Bereich der Software Entwicklung auf Basis von Oracle unterwegs. In rund dreihundert Projekten weltweit kam er mit mehr als zwei Dutzend Programmiersprachen und Paradigmen in Kontakt. In den letzten zwei Jahren beschäftigt er sich vorrangig mit der Modernisierung von Legacy-Applikationen aus Sicht der Architektur und den eingesetzten Frameworks.

Canary-Releases mit der Very Awesome Microservices Plattform

Bernd Zuther, codecentric AG



Immer mehr Unternehmen zerschlagen ihre Software-Systeme in kleine Microservices. Wenn das passiert, entstehen mehrere Deployment-Artefakte, was nicht nur das Deployment des Gesamtsystems komplexer macht. Um diese Komplexität beherrschen zu können und die Auslieferungsmöglichkeiten einer Software zu verbessern, ist der Einsatz von Werkzeugen zur Infrastruktur-Automatisierung unumgänglich.

Nur Unternehmen, die es schaffen, sich schnell genug an die sich ändernden Ansprüche und Anforderungen ihrer Kunden anpassen zu können, werden auf lange oder kurze Sicht überlebensfähig bleiben. Die Absatzmärkte der meisten Firmen erfahren seit einigen Jahren einen drastischen Wandel. Durch Globalisierung, Marktsättigung und das Internet haben sich stark konkurrierende und dynamische Märkte entwickelt, die bedarfsgesteuert sind. Daher sollten sich Unternehmen an die veränderten Bedürfnisse ihrer Kunden schnell anpassen: An dieser Stelle sollte von „kosteneffizienter Skalierung“ zu „Reaktionsfähigkeit“ gewechselt werden [1].

Canary-Release

Um das Risiko beim Ausrollen einer neuen Software-Version zu vermindern, benutzen viele Unternehmen eine Technik, die

sich „Canary-Release“ nennt. Dabei werden Änderungen erst einmal nur einem Teil der Benutzer zu Verfügung gestellt und es wird gemessen, wie dieser Teil auf die Änderungen reagiert, bevor man die neue Software auf der kompletten Infrastruktur ausrollt. Durch dieses Vorgehen wird eine Möglichkeit geschaffen, Software schrittweise aus fachlicher Sicht zu verbessern [2].

Abbildung 1 stellt den Aufbau eines Canary-Release dar. Es ähnelt in der Ausführung einem Blue-Green-Deployment [3], das aber das Ziel verfolgt, die Downtime einer Applikation zu minimieren. Bei beiden Verfahren werden zwei Infrastruktur-Stränge benötigt, auf die zwei unterschiedliche Software-Versionen geliefert werden. Dabei handelt es sich jeweils um die neue und die alte Version der Software. Über einen Router werden die Benutzer auf die entsprechende Version umgeleitet.

Bei einem Blue-Green-Deployment macht man einen harten Wechsel auf die neue Version der Software. Beim Canary-Release werden dagegen beispielsweise nur fünf Prozent des Traffics auf die neue Version umgeleitet. Damit soll herausgefunden werden, ob eine neue Funktionalität wirklich eine signifikante Veränderung im Verhalten der Benutzer auslöst. Diese Release-Form kann auch benutzt werden, um A/B-Testing zu implementieren. Das Durchführen von Canary-Releases ist bei einer Microservice-Architektur eine nicht triviale Aufgabe.

Microservice-Architektur

Abbildung 2 zeigt ein Verteilungsdiagramm einer Microservice-Architektur, das die Komplexität dieser Architektur-Form verdeutlichen soll. Hier sind unterschiedlichste Artefakte zu sehen, die entstehen können, wenn

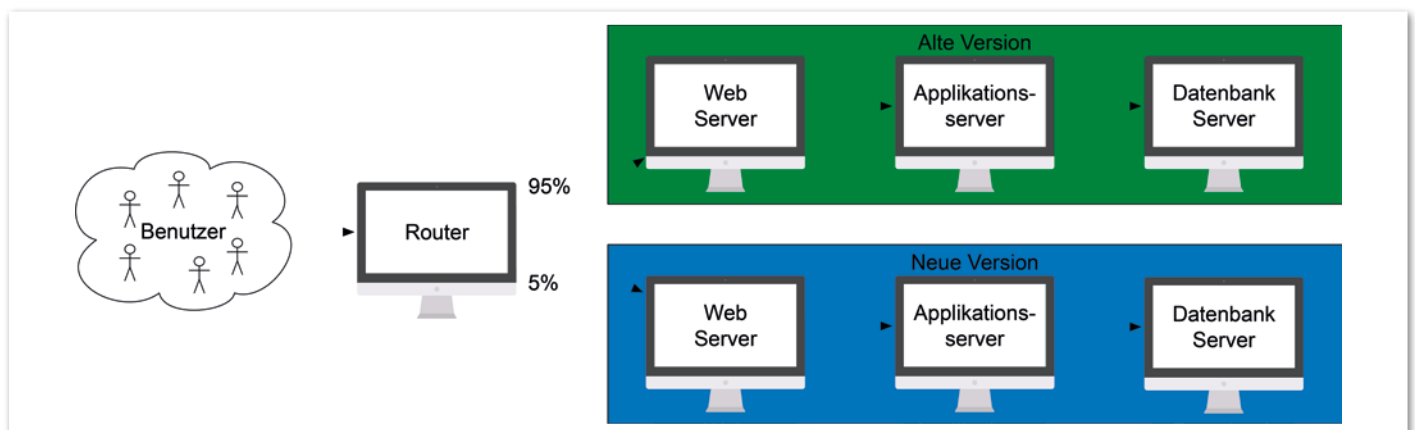


Abbildung 1: Canary-Release

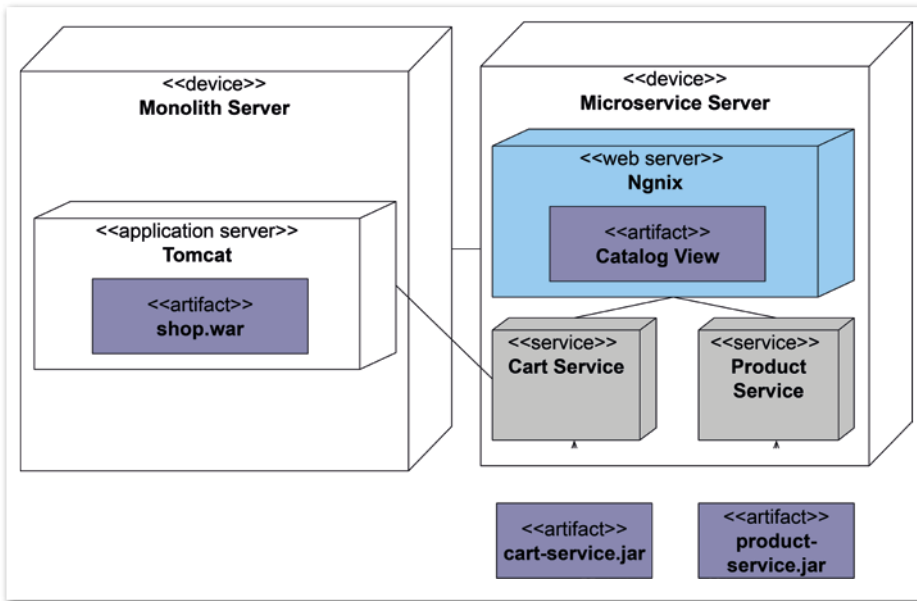


Abbildung 2: Verteilungsdiagramm einer Microservice-Architektur

man eine monolithische Anwendung in eine Microservice-Architektur zerteilt. Auf dem linken Server ist eine Shop-Anwendung zu sehen, die über eine WAR-Datei ausgeliefert wird und über einen Service mit dem rechten Server kommuniziert. Auf diesem Server sind drei verschiedene Anwendungen installiert. Eine Anwendung implementiert eine Katalog-Ansicht, die von einem Nginx ausgeliefert wird. Außerdem sind ein Produkt- und ein Warenkorb-Service dargestellt, die jeweils über eine JAR-Datei gestartet werden. Damit die Anwendung vollständig läuft, sind also vier Artefakt-Arten erforderlich, die unterschiedlich betrieben werden [4].

Docker

Docker eignet sich, um diese unterschiedlichen Artefakt-Arten aus Betriebssicht zu standardisieren. Es bietet Funktionen zum Erstellen und Pflegen von Images, die man über eine sogenannte „Registry“ sehr einfach und schnell auf andere Rechner verteilen kann. Betreibt man den Docker-Daemon auf einem Rechner, kann man aus einem solchen Image einen Container erzeugen, der einen laufenden Linux-Prozess beinhaltet. Der Docker-Daemon nutzt zum Ausführen solcher Prozesse spezielle Kernel-Funktionalitäten, um einzelne Prozesse in den Containern voneinander abzugrenzen. Container enthalten kein Betriebssystem, sondern nur die jeweils erforderlichen Linux-Anwendungen und deren benötigte Bibliotheken [5].

Listing 1 verdeutlicht einen Workflow, der mit Docker abgebildet werden kann, um Anwendungen auf beliebigen anderen Rech-

nern auszuführen. Zuerst wird mithilfe eines Docker-Files und des Build-Kommandos ein Image gebaut. Dann übermittelt man das Image mit dem Push-Kommando in eine private oder öffentliche Registry. Auf dem Zielrechner wird ein Image mit dem Pull-Kommando aus der Registry heruntergeladen und mit dem Run-Kommando gestartet. Danach läuft auf diesem Rechner ein neuer Prozess.

Docker im verteilten System

Um zwei Container miteinander per Netzwerk kommunizieren zu lassen, bietet Docker das Link-Konzept an. Beim Verlinken werden IP und exportierte Ports des Zielcontainers als Umgebungsvariable im nutzenden Container bereitgestellt. Der Container, der den Link erzeugt, kann in der Konfiguration der Anwendung auf diese Umgebungsvariablen zugreifen und muss nicht IP und Port direkt

verwenden. Listing 2 zeigt, wie diese Umgebungsvariablen aussehen.

Die Verlinkung funktioniert allerdings nur, wenn sich die Docker-Container auf dem gleichen Server befinden. Wenn Container über mehrere Server verteilt und miteinander verbunden werden sollen, kann das direkt über IP und Port passieren. Möchte man dies per Hand machen, muss man sich auf den unterschiedlichen Rechnern anmelden, die Container in einer bestimmten Reihenfolge starten und die entsprechenden Umgebungsvariablen setzen. Sollten sich IP oder Port der Zielcontainer ändern, müsste der zugreifende Container angepasst und neu gestartet werden. Dieses Vorgehen würde zu einer statischen Umgebung führen und man könnte nicht einfach neue Rechner hinzufügen oder wegnehmen [6].

Very Awesome Microservices Platform

Die Very Awesome Microservices Platform (VAMP, [7]) ist ein Werkzeug, das an dieser Stelle ansetzt. Es verwendet aktuell Mesos und Marathon als Container-Manager. Die Plattform wird von der niederländischen Firma magnetic.io [8] entwickelt. Mesos [9] sorgt für die Ressourcen-Verwaltung im Cluster und ist verantwortlich für das Starten von Docker-Containern in einem verteilten Rechner-Cluster. Marathon [10] ist ein Werkzeug, das auf Mesos aufbaut. Mit Marathon kann das Deployment von Microservice-Anwendungen durchgeführt werden. Marathon stellt einen Scheduler dar, der die Verteilung und Ausführung von Docker-Containern steuert (siehe Abbildung 3).

Vamp ist ein Dienst, der sich oberhalb eines Container-Managers wie Marathon ansiedelt und aus mehreren Komponenten besteht.

```

bz@cc1 $ docker build -t zutherb/product-service .
bz@cc1 $ docker push zutherb/product-service
bz@cc2 $ docker pull zutherb/product-service
bz@cc2 $ docker run zutherb/product-service
bz@cc2 $ docker ps
CONTAINER ID IMAGE COMMAND
87bb5524067d zutherb/product-service:latest "/product-0.6/bin/
    
```

Listing 1

```

bz@cc $ docker exec 254e40d85a33 env
MONGODB_PORT_27017_TCP=tcp://172.17.0.2:27017
MONGODB_PORT_27017_TCP_ADDR=172.17.0.2
MONGODB_PORT_27017_TCP_PORT=27017
    
```

Listing 2

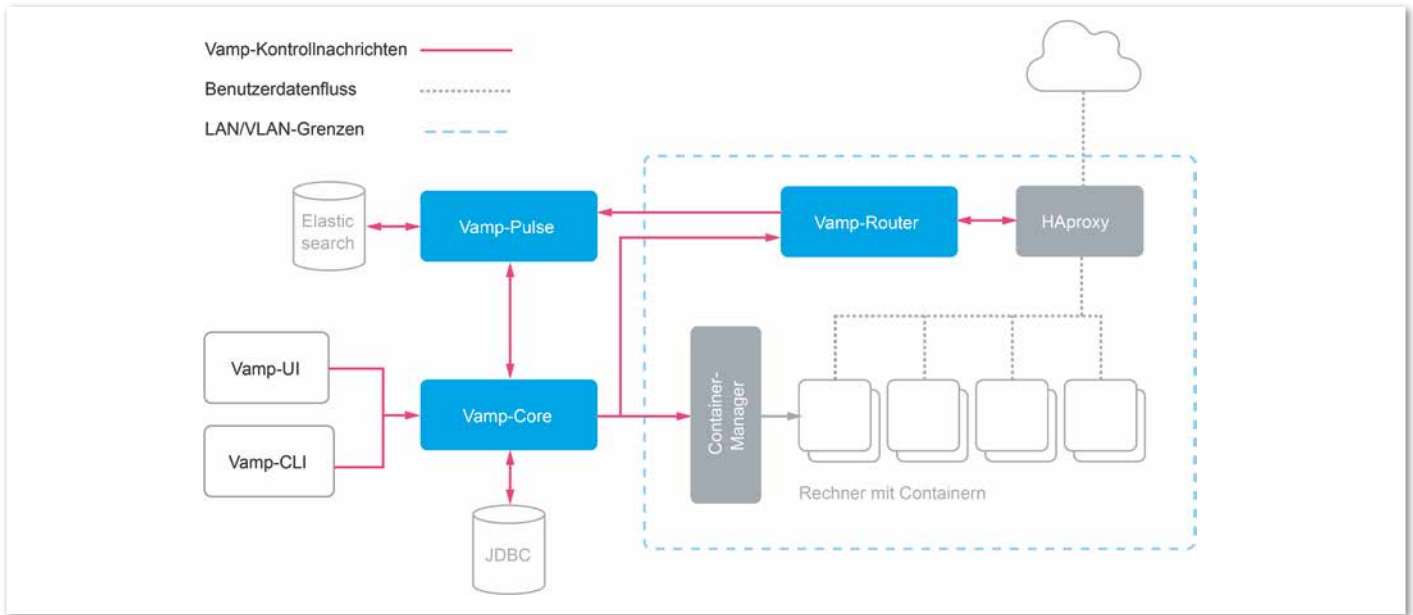


Abbildung 3: Vamp-Architektur

Vamp-Core bietet eine Plattform-agnostische DSL und ein REST-API. Die DSL kann ähnlich wie Giant Swarm [11] und Docker-Compose [12] benutzt werden, um eine Microservice-Anwendung mit all ihren Abhängigkeiten zu beschreiben und auszuführen.

Listing 3 zeigt einen Ausschnitt aus einer solchen Beschreibung [13]. Diese DSL wird von Vamp-Core benutzt, um mit Marathon die Microservices im Mesos-Cluster auszuliefern. Außerdem beinhaltet die DSL Elemente zum Beschreiben von A/B-Tests und Canary-Releases sowie zum Beschreiben von SLAs. Diese SLAs sorgen dafür, dass, wenn bestimmte Antwortzeiten unterschritten werden und noch Ressourcen im Cluster verfügbar sind, automatisch neue Docker-Container im Cluster gestartet werden.

Zum Sammeln der Metriken, die für die SLAs und A/B-Tests benötigt werden, gibt es den Metriken- und Event-Store Vamp-Pulse, in dem Daten von Core und Router in Elasticsearch gespeichert werden. Der Vamp-Router ist eine Komponente, die einen HA-Proxy steuert, der die Verbindung zwischen den laufenden Docker-Containern und den eigentlichen Benutzern realisiert [14].

Fazit

Um die Reaktionsfähigkeit durch Canary-Releases und A/B-Testing für die Produkt-Entwicklung zu verringern, die Effizienz bei der Auslieferung von Microservice-Architekturen zu verbessern und eine größere Geschwindigkeit in die Auslieferung von Software zu bekommen, ist Vamp eine Plattform, die sich sehr gut für folgende Aufgaben eignet:

- Jeden einzelnen Service in einer Microservice-Architektur erst mal nur im Rahmen eines Canary-Release mit einem kleinen Teil seiner Benutzer testen und feststellen, ob eine Änderung wirklich einen Mehrwert für seine Kunden bringt
- Software schneller und ohne Ausfallzeiten mithilfe eines Blue-Green-Deployment in Produktion bringen
- Testumgebungen einfachen bereitstellen, da auf unterschiedlichen Endpunkten unterschiedliche Anwendungskonfigurationen ausgeliefert werden können
- Bei Lastspitzen sein eigenes Datacenter um Cloud-Infrastrukturen erweitern und automatisch bei bestimmten Antwortzeiten herauf und herunter skalieren
- Daten im Metrik- und Event-System sammeln, um nachhaltige Entscheidungen über Veränderungen treffen zu können. Diese kommen aus dem Live-Betrieb und können vom Anfang bis zum Ende der Wertschöpfungskette vollständig automatisiert betrieben werden.

Quellen

- [1] <https://blog.codecentric.de/2015/08/the-need-for-speed-eine-geschichte-ueber-devops-microservices-continuous-delivery-und-cloud-computing/>
- [2] <http://martinfowler.com/bliki/CanaryRelease.html>
- [3] <http://martinfowler.com/bliki/BlueGreenDeployment.html>
- [4] Zuther, Bernd: Microservices und die Jagd nach mehr Konversion., in: Java aktuell (2015), Nr. 2, S. 35 – 40.
- [5] <https://public.centerdevice.de/00d60e2d-5490-4df6-afe0-c70824ffb14b>
- [6] <https://blog.codecentric.de/2015/08/variation-des-ambassador-pattern-von-coreos/>
- [7] <http://vamp.io/>

- [8] <http://magnetic.io/>
- [9] <http://mesos.apache.org/>
- [10] <https://mesosphere.github.io/marathon/>
- [11] <https://blog.codecentric.de/2015/05/microservice-deployment-ganz-einfach-mit-giant-swarm/>
- [12] <https://blog.codecentric.de/2015/05/microservice-deployment-ganz-einfach-mit-docker-compose/>
- [13] <https://github.com/zutherb/AppStash/blob/master/vamp/shop-ab-test.yml>
- [14] <https://blog.codecentric.de/2015/10/canary-release-mit-der-awesome-microservices-platform-vamp/>

Hinweis: Das Listing 3 finden sie online unter: www.ijug.eu/go/javaaktuell/zuther/listing

Bernd Zuther
bernd.zuther@codecentric.de



Bernd Zuther ist seit mehreren Jahren als Software-Entwickler tätig. Derzeit ist er bei der codecentric AG als Berater im Bereich Agile Software Factory beschäftigt, wo er vor allem Themen in den Bereichen „Big Data“, „Continuous Delivery“ und „Agile Software-Architekturen“ bearbeitet. Bernd verfügt über langjährige Erfahrung im Bereich E-Commerce und beschäftigt sich seit mehreren Jahren mit der Entwicklung von hochverfügbaren, individuellen Online-Systemen.

Weiterführende Themen zum Batch Processing mit Java EE 7

Philipp Buchholz, esentri AG

In der letzten Ausgabe ging es um grundlegende Dinge zum Batch Processing mit Java EE 7. Der Aufbau des Batch Processing nach JSR 352 wurde ausführlich beschrieben und anhand von Beispielen erläutert. Dieser Artikel beschreibt weiterführende Themen wie die Parallelisierung von Batch-Jobs mit Partitionen und Splits sowie Batchlets, Transition-Elemente und das Neustarten von Batch-Jobs. Darüber hinaus werden die Möglichkeiten der Batch-Verarbeitung auf dem Applikationsserver Oracle WebLogic kurz angesprochen.

Abbildung 1 zeigt als Wiederholung zum letzten Artikel den Zusammenhang zwischen `ItemReader` (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemReader.html>“), `ItemWriter` (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemWriter.html>“), `ItemProcessor` (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemProcessor.html>“) und den CheckPoint-Informationen (siehe Seite 17 der Spezifikation in Sektion 7.9), die zum Wiederaufsetzen von Batch-Jobs verwendet werden. Die Beziehung zwischen Chunk und CheckPoint-Information besteht darin, dass nach jedem erfolgreich gelesenen und geschriebenen Chunk eine CheckPoint-Information abgegeben wird. Diese sind in den Zustandsinformationen des Schrittes hinterlegt. Sobald die Schreib- beziehungsweise Lesevorgänge innerhalb der Chunks nicht erfolgreich ausgeführt werden können, lassen sich die gesicherten CheckPoint-Informationen für das Wiederaufsetzen

der Batch-Verarbeitung am letzten erfolgreichen CheckPoint nutzen.

Neustarten von Batch-Jobs

Die innerhalb der Job-XML-JobSpecificationLanguage (JSL) definierten Batch-Jobs können über das Attribut „restartable“ neustartbar gemacht werden. „True“ ist für dieses Attribut der Standardwert. Wird es auf „False“ gesetzt, lässt sich ein Batch-Job nicht neu starten.

Damit ein Batch-Job die oben erwähnten CheckPoint-Informationen für ein Wiederaufsetzen nutzen kann, muss man als Erstes die konfigurierte „ItemReader“-Implementierung anpassen. Die Implementierung erbt von der abstrakten Basisklasse `AbstractItemReader` (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/AbstractItemReader.html>“) die leeren Methodenrumpfe für alle Methoden, außer der abstrakt definierten „readItem“ (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/AbstractItemReader.html#readItem-->“). Die Methode „open“ (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/AbstractItemReader.html#open-java.io.Serializable->“) wird überschrieben, um eine „CheckPoint“-Information nach einem Wiederaufsetzen berücksichtigen zu können.

In Listing 1 wird die ID der letzten Bestellung als CheckPoint-Information verwendet und hinterlegt. Handelt es sich nicht um einen Restart, sondern um den initialen Start eines Batch-Jobs, ist die CheckPoint-Information „null“. Die „readItem“-Methode muss ebenfalls angepasst werden, um die CheckPoint-Information zu berücksichtigen. Listing 2 zeigt einen Ausschnitt aus der Methoden-Implementierung:

Hier wird die zuvor innerhalb der „open“-Methode hinterlegte ID verwendet, um die Datenbank-Abfrage entsprechend einzuschränken. Innerhalb der „readItem“-Methode ist darauf zu achten, dass die ID des zuletzt verarbeiteten Datensatzes aktualisiert wird. Diese Information ist innerhalb der Methode „checkpointInfo()“ erforderlich. Sie wird von der Laufzeitumgebung nach dem erfolgreichen Verarbeiten eines Chunk aufgerufen; der Rückgabewert wird in den Zustands-Informationen („JobRepository“) gespeichert.

Ebenso wie die „ItemReader“- ist auch die „ItemWriter“-Implementierung anzupassen. Auch hier werden die „open“- und die „checkpointInfo“-Methode entsprechend implementiert. Nachdem die Batch-Artefakte angepasst sind, kann ein Neustart eines Batch-Jobs über den `JobOperator` erfolgen (siehe Listing 3).

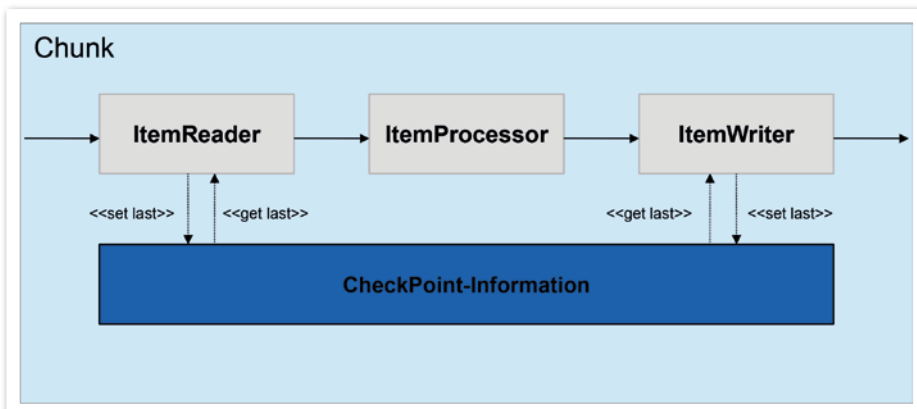


Abbildung 1: Zusammenhang zwischen `ItemReader`, `ItemProcessor` und `ItemWriter` innerhalb eines Chunk

Wichtig ist, dass nur der Neustart der aktuellsten JobExecution möglich ist (siehe auch Seite 95 der Spezifikation in Sektion 10.8 „Restart Processing“). Eine Hilfsmethode in Listing 3 ermittelt die aktuellste JobExecution. Sie lässt sich auf Basis des JobOperator schnell selbst entwickeln („findMostRecentJobExecution()“). Je nach Anwendungsdesign kann eine Basisklasse die Methoden für das Suchen der aktuellsten JobInstance beziehungsweise JobExecution anbieten.

Erreichen eines bestimmten Statuswerts

Für das Anhalten eines Batch-Jobs ist innerhalb der Job-XML das „stop“-Element (siehe auch Seite 45 der Spezifikation in Sektion 8.6.4 „Stop Element“) vorgesehen. Es wird verwendet, um einen Batch-Job nach dem Erreichen eines bestimmten Statuswerts innerhalb eines Verarbeitungsschritts anzuhalten. Das Attribut „on“ enthält den zu erreichenden Statuswert und „restart“ erlaubt das Setzen des Schrittes, bei dem die Verarbeitung nach dem Anhalten wieder startet. Listing 4 zeigt einen Schritt, der beim Erreichen des

Statuswerts „COMPLETED“ anhält und nach dem Neustarten beim Schritt „loadIncomingOrders“ wieder aufsetzt. Das Beispiel dient nur der Demonstration. Normalerweise würde das „stop“-Element innerhalb eines Entscheidungsknotens dazu dienen, die Ausführung bei bestimmten Zuständen kontrolliert anzuhalten.

Verzweigter Schrittaufbau und Entscheidungsknoten

Innerhalb der Job-XML können per „decision“-Element (siehe Seite 42 der Spezifikation in Sektion 8.5 „Decision“) Entscheidungsknoten implementiert sein. Sie werden nach dem Ausführen eines Schrittes innerhalb der Batch-Verarbeitung aufgerufen. Die Entscheidung über die weitere Ausführung ist in einem sogenannten „Decider“ implementiert. Dieser muss das Interface „javax.batch.api.Decider“ implementieren. Innerhalb der Methode „decide(stepExecutions : StepExecution[]) : String“ wird unter Verwendung der Zustands-Informationen der zuvor ausgeführten Schritte – der übergebenen StepExecutions – entschieden, welcher Schritt als

Nächstes ausgeführt werden soll. So lassen sich Verzweigungen innerhalb eines Batch-Jobs realisieren. Die Entscheidung wird als String dargestellt, der innerhalb der Job-XML für die Traversierung zum nächsten Transition-Element genutzt wird. Listing 5 zeigt das Beispiel eines konfigurierten Decider.

Innerhalb des „OrdersAvailableDecider“ wird entschieden, ob Bestellungen geladen werden konnten oder nicht. Nur wenn diese erfolgreich geladen werden konnten, kann eine Überprüfung des Inventars erfolgen. Diese Überprüfung findet dann innerhalb des Schritts „checkInventory“ statt.

Transition-Elemente

Die Transition-Elemente innerhalb der Spezifikation sind „next“, „stop“, „fail“ und „end“ (siehe Seite 43 der Spezifikation in Sektion 8.6 „Transition Elements“). Das „stop“-Element wurde bereits behandelt, das „next“-Element (siehe Seite 43 der Spezifikation in Sektion 6.6.1 „Next Element“) kam schon für die Steuerung der Schrittfolge innerhalb eines Batch-Jobs vor. Das „fail“-Element (siehe Seite 44 der Spezifikation in Sektion 8.6.2 „Fail



... more voice

-  Mitspracherecht
-  Gestaltungsspielraum
-  Hohe Freiheitsgrade

... more locations



80%

Moderate Reisezeiten –
80 % Tagesreisen
< 200 Kilometer

Aalen	Karlsruhe
Böblingen	München
Dresden	Neu-Ulm
Hamburg	Stuttgart (HQ)

... more partnership



-  Experten auf Augenhöhe
-  Individuelle Weiterentwicklung
-  Teamzusammenhalt

Unser Slogan ist unser Programm. Als innovative IT-Unternehmensberatung bieten wir unseren renommierten Kunden seit vielen Jahren ganzheitliche Beratung aus einer Hand. Nachhaltigkeit, Dienstleistungsorientierung und menschliche Nähe bilden hierbei die Grundwerte unseres Unternehmens.

Zur Verstärkung unseres Teams Software Development suchen wir Sie als

Senior Java Consultant / Softwareentwickler (m/w)

an einem unserer Standorte

Ihre Aufgaben:

Sie beraten und unterstützen unsere Kunden beim Aufbau moderner Systemarchitekturen und bei der Konzeption sowie beim Design verteilter und moderner Anwendungsarchitekturen. Die Umsetzung der ausgearbeiteten Konzepte unter Nutzung aktueller Technologien zählt ebenfalls zu Ihrem vielseitigen Aufgabengebiet.

Sie bringen mit:

- Weitreichende Erfahrung als Consultant (m/w) im Java-Umfeld
- Sehr gute Kenntnisse in Java/J2EE
- Kenntnisse in SQL, Entwurfsmustern/Design Pattern, HTML/XML/ XSL sowie SOAP oder REST
- Teamfähigkeit, strukturierte Arbeitsweise und Kommunikationsstärke
- Reisebereitschaft

Sie wollen mehr als einen Job in der IT? Dann sind Sie bei uns richtig!
Bewerben Sie sich über unsere Website: www.cellent.de/karriere



Element“) beendet nach dem Ausführen eines Schritts die Verarbeitung fehlerhaft. Der Status des Batch-Jobs ist in der Folge auf „failed“ gesetzt.

Sollte es notwendig sein, eine in mehrere Schritte zerlegte Ausführung zu einer Arbeitseinheit zusammenzufassen, die gemeinsam erfolgreich oder fehlerhaft abgeschlossen wird, kommen Flows (siehe Seite 39 der Spezifikation in Sektion 8.3 „Flow“) zum Einsatz. Die darum gruppierten Schritte lassen sich nur mit Schritten innerhalb dieses Flows verbinden. Eine Verschachtelung zur Bildung von hierarchischen Strukturen ist möglich.

Innerhalb des Beispiels wurde die Konfiguration des „ItemReader“, „ItemWriter“ beziehungsweise „ItemProcessor“ zur besseren Lesbarkeit ausgelassen. Die beiden innerhalb des Flows geschachtelten Schritte würden in diesem Beispiel als gemeinsame Verarbeitungseinheit erfolgreich oder fehlerhaft abschließen. Somit ist hier eine abgeschlossene Verarbeitungseinheit, bestehend aus zwei Schritten, entstanden.

Verwendung von Batchlets

Folgt ein Schritt innerhalb der Batch-Verarbeitung nicht dem Chunk-orientierten Ansatz, kommt ein Batchlet (siehe Seite 22 der Spezifikation in Sektion 8.2.2 „Batchlet“) zum Einsatz. Dieses muss das Interface „javax.batch.api.Batchlet“ (siehe „<https://docs.oracle.com/javaee/7/api/javax/batch/api/Batchlet.html>“) implementieren oder von der abstrakten Basisklasse „javax.batch.api.AbstractBatchlet“ (siehe „<http://docs.oracle.com/javaee/7/api/javax/batch/api/AbstractBatchlet.html>“) erben. Diese abstrakte Basisklasse bietet leere Methoden-Implementierungen, die überschrieben werden können.

Die Logik für das Processing innerhalb des Batchlet ist in der Methode „process(): String“ implementiert. Diese gibt einen String zurück, der innerhalb der Job-XML für die Entscheidung des nächsten Verarbeitungsschrittes verwendet wird. In Batchlets lassen sich beispielsweise Dateiübertragungen oder der Aufruf von einzelnen für sich stehenden Operationen sehr gut kapseln.

Wenn Verarbeitungsergebnisse für die weitere Verarbeitung relevant sind, können sie beispielsweise innerhalb des „StepContext“ (siehe „<http://docs.oracle.com/javaee/7/api/javax/batch/runtime/context/StepContext.html>“) hinterlegt sein. Innerhalb des StepContext können dafür transiente Userdaten gesetzt sein („setTransientUserData“-Methode). Diese Daten stehen den nächs-

ten Schritten zur Verfügung, sind allerdings nach einem Neustart nicht mehr vorhanden.

Parallelität

Innerhalb einer umfangreichen Batch-Verarbeitung ist es sehr wichtig, dass verfügbare Rechenleistung effizient genutzt wird.

Heute ist es normal, solche Verarbeitungen auf Servern auszuführen, die ihre Rechenarbeiten auf mehrere Prozessoren aufteilen. Im Gegensatz zur Quasi-Parallelität ist damit eine echte parallele Ausführung von

```
@Override
public void open(Serializable checkpoint) throws Exception {
    /* If this is a restart the Checkpoint information will not be null. */
    if (null != checkpoint)
        lastOrderId = (Long) checkpoint;
}
```

Listing 1

```
/* Consider if may be performing a checkpoint restart. */
Predicate checkpointInfoPredicate = null;
if (null != lastOrderId) {
    checkpointInfoPredicate = criteriaBuilder.greaterThanOrEqualTo(orderRoot,
        <Long> get("id"), lastOrderId);
}
```

Listing 2

```
JobExecution mostRecentJobExecution = this.findMostRecentJobExecution();
jobOperator.restart(mostRecentJobExecution.getExecutionId(), restartParameters);
```

Listing 3

```
<step id="sendAcknowledgmentMessage">
    <chunk item-count="1">
        <reader ref="ProcessedInvoiceReader" />
        <processor ref="MessagePreparationProcessor" />
        <writer ref="sendAcknowledgmentMessageWriter" />
    </chunk>
    <stop on="COMPLETED" restart="loadIncomingOrders" />
</step>
```

Listing 4

```
<decision ref="OrdersAvailableDecider" id="checkOrdersAvailable">
    <stop on="NO_DATA_READ" exit-status="NO_DATA_READ" />
    <next on="DATA_READ" to="checkInventory" />
</decision>
```

Listing 5

```
<flow id="loadingStage">
    <step id="detectOffsetStart">
        <batchlet ref="DetectOffsetStartBatchlet" />
        <next on="OFFSET_AVAILABLE" to="loadIncomingOrder" />
    </step>
    <step id="loadIncomingOrder" next="checkOrdersAvailable">
        ...
    </step>
    ...
</flow>
```

Listing 6

```

<step id="loadIncomingOrder">
  <chunk item-count="10">
    <reader ref="loadIncomingOrderItemReader">
      <properties>
        <property name="maxResultsPerChunkHolder" value="#{jobProperties['maxResultsPerChunkHolder']}" />
        <property name="offset" value="#{partitionPlan['offset']}" />
        <property name="logVerbose" value="#{jobProperties['logVerbose']}" />
      </properties>
    </reader>
    <processor ref="MessagePreparationProcessor" />
    <writer ref="sendAcknowledgmentMessageWriter" />
  </chunk>
  <partition>
    <plan partitions="2" threads="2">
      <properties partition="0">
        <property name="offset" value="0" />
      </properties>
      <properties partition="1">
        <property name="offset" value="5" />
      </properties>
    </plan>
  </partition>
</step>

```

Listing 7

Rechenaufgaben möglich. Um von diesen Ressourcen innerhalb einer Batch-Verarbeitung auf Basis von Java EE 7 profitieren zu können, muss es möglich sein, die einzelnen Verarbeitungsschritte auf mehrere Threads zu verteilen und parallel auszuführen.

Zur Partitionierung eines Verarbeitungsschritts ist es innerhalb von Java EE 7 möglich, einen Schritt in Partitionen zu unterteilen. Jede dieser Partitionen wird standardmäßig in einem eigenen Thread ausgeführt. Die „1:1“-Zuordnung zwischen

Partition und Thread kann jedoch innerhalb des „partition“-Elements geändert werden, das für das Anlegen eines Partitioned Step (siehe Seite 32 der Spezifikation in Sektion 8.2.6 „Step Partitioning“) zum Einsatz kommt.

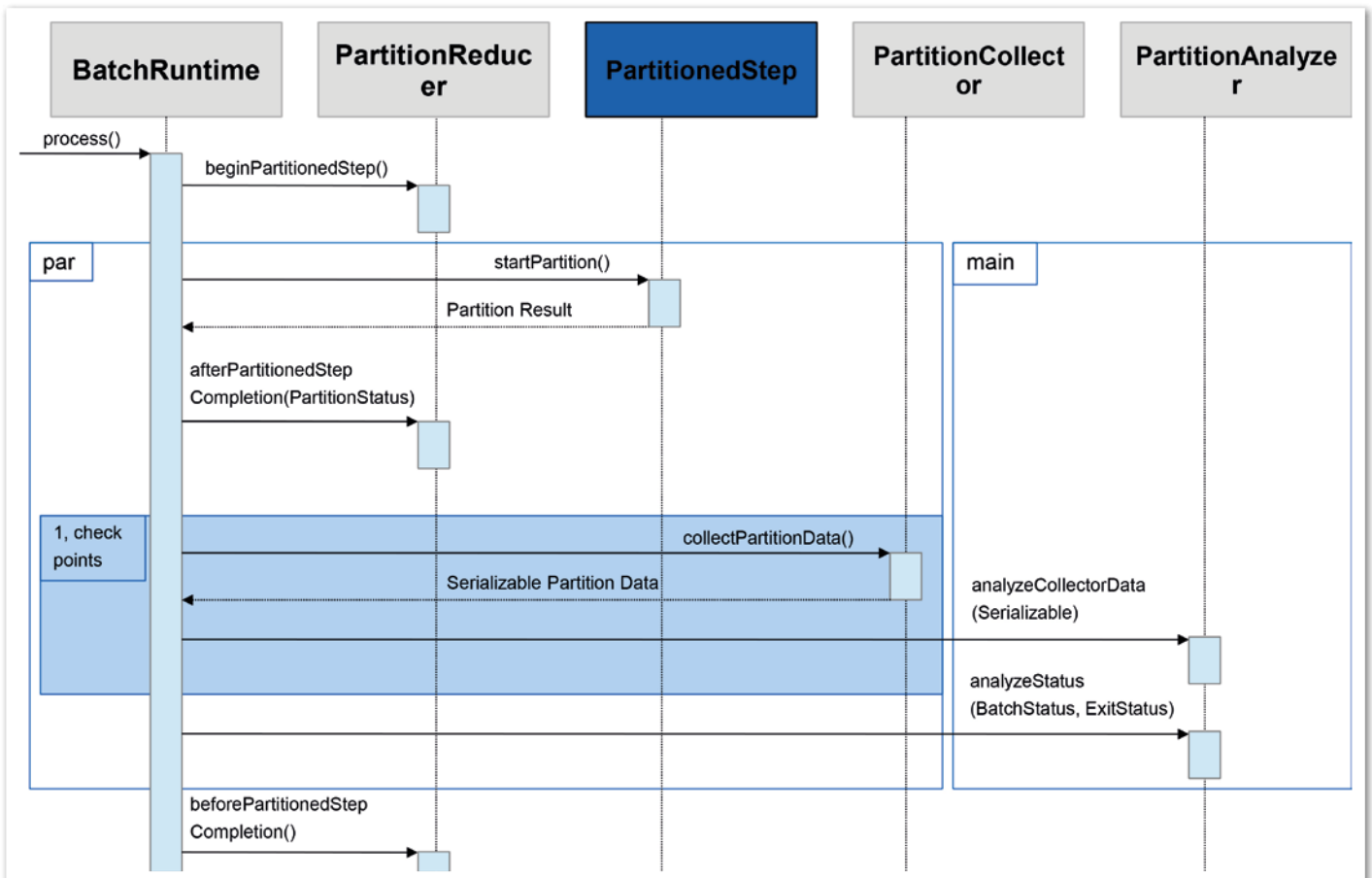


Abbildung 2: Zusammenspiel von PartitionReducer, PartitionCollector und PartitionAnalyzer

Das Beispiel in Listing 7 zeigt einen Verarbeitungsschritt, der in zwei Partitionen aufgeteilt ist. Ein Partition-Plan regelt die Aufteilung der Daten zwischen den Partitionen. Dieser ist durch das „plan-element“ (siehe Seite 33 der Spezifikation in Sektion 8.2.6.1 „Partition Plan“) dargestellt. Wenn die Definition innerhalb der Job-XML nicht flexibel genug ist, lässt sich das Interface „javax.batch.api.partition.Partitioner“ (siehe „https://docs.oracle.com/javaee/7/api/javabatch/api/partition/Partitioner.html“) implementieren.

Die Methode „mapPartitions() : Partition-Plan“ erlaubt das programmatische Erstellen eines Partition-Plans. In der Folge werden pro Thread ein „ItemReader“, ein „ItemProcessor“ sowie ein „ItemWriter“ erstellt und die Verarbeitung parallel durchgeführt. Um keine konkurrierenden Datenzugriffe zu erhalten, müssen die „ItemReader“ auf die konfigurierten Daten der jeweiligen Partition zugreifen. Die konfigurierte Property des Partition-Plans wird hierzu übergeben.

Für das Bilden eines passenden Partition-Plans sind beliebig viele Properties möglich. Im gezeigten Beispiel wird zwischen den Partitionen ein Offset definiert, das innerhalb des „ItemReader“ für das Ausschließen von überschneidenden Lesevorgängen verwendet wird.

PartitionReducer, PartitionCollector und PartitionAnalyzer

Da das Checkpointing und damit das Transaktionsverhalten bei parallelisierten Verarbeitungsschritten pro Thread unabhängig voneinander ausgeführt werden, gibt es mit den oben genannten Batch-Artefakten erweiterte Möglichkeiten, das Batch Processing zu kontrollieren (siehe Abbildung 2).

Das Sequenz-Diagramm zeigt das Zusammenspiel dieser Batch-Artefakte. Die Methodenaufrufe innerhalb der mit „par“ gekennzeichneten Regionen werden parallel in separaten Threads ausgeführt. Die mit „main“ gekennzeichnete Region stellt den Main-Thread der Anwendung dar.

Der PartitionReducer (siehe „https://docs.oracle.com/javaee/7/api/javabatch/api/partition/PartitionReducer.html“) bietet über Callback-Methoden die Möglichkeit, an bestimmten Stellen des Lifecycle in die parallele Verarbeitung einzugreifen. Die Methode „beginPartitionedStep()“ wird vor Beginn eines parallelen Schritts, die Methode „beforePartitionedStepCompletion()“ am Ende aufgerufen. Nach dem Beenden einer Partition erfolgt die Methode „afterPartitionedStepCompletion(PartitionStatus)“. Hier bietet sich die Möglichkeit, auf den resultierenden Status der Verarbeitung einer Partition zu reagieren.

Der „PartitionCollector“ (siehe „https://docs.oracle.com/javaee/7/api/javabatch/api/partition/PartitionCollector.html“) sammelt fortlaufend die Daten, die nach den Checkpoints innerhalb eines partitionierten Schritts anfallen. Diese gehen zur Analyse und weiteren Verarbeitung an einen „PartitionAnalyzer“ (siehe „https://docs.oracle.com/javaee/7/api/javabatch/api/partition/PartitionAnalyzer.html“).

Zu beachten ist, dass die Instanzen des „PartitionCollector“ im Thread der jeweiligen Partition ausgeführt werden und dadurch mehrere Instanzen vorhanden sind. Der „PartitionAnalyzer“ hingegen ist immer einmal vorhanden und wird im Main-Thread der Anwendung ausgeführt. Damit bietet dieser eine zentrale Stelle, um am Schluss eines partitionierten Schritts alle angefallenen Daten zu analysieren und zu finalisieren.

Parallele Ausführung von unterschiedlichen Schritten

Das „split“-Element (siehe Seite 40 der Spezifikation in Sektion 8.4 „Split“) bietet im Gegensatz zum Partitionieren eines Verarbeitungsschritts die Möglichkeit, unterschiedliche Flows parallel auszuführen. Damit lassen sich nicht nur die Daten eines Verarbeitungsschritts parallel prozessieren,

```
<split id="checkInventories">
  <flow id="checkFoodInventory">
    ...
  </flow>
  <flow id="checkNonFoodInventory">
    ...
  </flow>
</split>
```

Listing 8

```
<flow id="loadingStage">
  <step id="detectOffsetStart">
    <batchlet ref="DetectOffsetStartBatchlet" />
    <next on="OFFSET_AVAILABLE" to="loadIncomingOrder" />
  </step>
  ...
</flow>
```

Listing 9

```
@Dependent
@NamedNativeQuery(name = "detectOffsetStartQuery", query = "SELECT MAX(id) FROM Order WHERE orderStatus == 'NEW'")
public class DetectOffsetStartBatchlet extends AbstractBatchlet {
    ...
    @Override
    public String process() throws Exception {
        /* Read and set needed start of the offset. */
        TypedQuery<Long> detectOffsetStartQuery = entityManager.createNamedQuery("detectOffsetStartQuery", Long.class);
        Long detectOffsetStart = detectOffsetStartQuery.getSingleResult();

        if (null == detectOffsetStart)
            return "FAILED";

        stepContext.setTransientUserData(detectOffsetStart);

        return "OFFSET_AVAILABLE";
    }
}
```

Listing 10


```

<step id="loadIncomingOrder" next="checkOrdersAvailable">
  <chunk item-count="10">
    <reader ref="loadIncomingOrderItemReader">
      <properties>
        <property name="maxResultsPerChunkHolder" value="#{jobProperties['maxResultsPerChunkHolder']}" />
        <property name="offset" value="#{partitionPlan['offset']}" />
        <property name="logVerbose" value="#{jobProperties['logVerbose']}" />
      </properties>
    </reader>
    <writer ref="orderStatusItemWriter">
      <properties>
        <property name="OrderStatus" value="RECEIVED" />
      </properties>
    </writer>
  </chunk>
  <partition>
    <plan partitions="2" threads="2">
      <properties partition="0">
        <property name="offset" value="0" />
      </properties>
      <properties partition="1">
        <property name="offset" value="5" />
      </properties>
    </plan>
  </partition>
</step>
<decision ref="OrdersAvailableDecider" id="checkOrdersAvailable">
  <stop on="NO_DATA_READ" exit-status="NO_DATA_READ" />
  <next on="DATA_READ" to="checkInventory" />
</decision>
</flow>

```

Listing 11

sondern auch unabhängige Verarbeitungen nebenläufig durchführen.

In Listing 8 werden die Flows „checkFoodInventory“ und „checkNonFoodInventory“ mit ihren Verarbeitungsschritten parallel ausgeführt. Ein „split“ wird immer gemeinsam beendet, wenn alle enthaltenen Flows fertig sind. Ein komplexeres Anwendungsbeispiel könnte beispielsweise die Daten aus mehreren Datenquellen parallel laden.

Batch Processing auf Basis von Oracle WebLogic

Ab Version 12.2.1 (siehe „<https://docs.oracle.com/middleware/1221/wls/NOTES/whats-new.htm#NOTES107>“) unterstützt der Applikationsserver Oracle WebLogic Java EE 7 und damit auch das Batch Processing mit Java EE 7. Dazu wurde die Administrationsoberfläche um eine Überwachungsmöglichkeit für Batch-Jobs erweitert. Diese ist in den Haupteinstellungen der Domäne über den Reiter „Monitoring“ zu erreichen. Die Oberfläche ermöglicht standardmäßig die Überwachung der Statuswerte von laufenden und beendeten Batch-Jobs.

Im Auslieferungszustand nutzt WebLogic die integrierte Derby-Datenbank, um Zustands-Informationen wie Checkpoints abzuspeichern. Als Zustandsspeicher kann

hier allerdings jede eingerichtete Datasource zum Einsatz kommen. Auf dieser müssen nur die benötigten Tabellen eingerichtet sein. Die WebLogic-Installation liefert im Verzeichnis „ORACLE_HOME/oracle_common/common/sql/wlservices/batch/“ für diverse Datenbank-Systeme DDL-Skripte für die Erstellung der benötigten Datenbank-Strukturen mit.

Das gewählte Szenario für ein Anwendungsbeispiel beschreibt das Laden von eingehenden Bestellungen. Diese liegen in einer Datenbank vor und könnten von einem Vorsystem in dieser Datenbank stammen. Der Zugriff erfolgt per JavaPersistenceAPI (JPA). Ein erster Verarbeitungsschritt ermittelt den Start des Offset, ab dem das Laden von eingehenden Bestellungen begonnen wird (siehe Listing 9).

Die Verarbeitung ist in Flows aufgeteilt. Der erste Flow kümmert sich als geschlossene Verarbeitungseinheit um das Laden der Daten. Außerdem steht an dieser Stelle ein Batchlet, da man im initialen Schritt nicht dem Chunk-orientierten Ansatz folgt (siehe Listing 10).

Innerhalb des Batchlet steht eine JPA „NamedNativeQuery“ (siehe „<http://docs.oracle.com/javaee/7/api/javax/persistence/NamedNativeQuery.html>“), um den Start des Offset zu ermitteln. Einige Variablen-Deklarationen wur-

den aufgrund besserer Lesbarkeit ausgelassen. Beim nächsten Schritt des aktuellen Flows wird das Laden der eingehenden Bestellungen anhand des Offset auf zwei Partitionen und damit zwei Threads aufgeteilt (siehe Listing 11).

Der „Offset“ verhindert, dass beide Threads die gleichen Daten lesen. Am Ende des Flows beendet eine „Decision“ die Verarbeitung, wenn keine Daten mehr vorhanden sind. Solange es Daten gibt, wird ein Lagerverwaltungssystem nach dem aktuellen Lagerbestand der bestellten Waren befragt. Sind Waren in ausreichender Menge vorhanden, wird die Bestellung akzeptiert, ansonsten abgelehnt.

Fazit

Der Artikel zeigt, wie innerhalb des Java-EE-7-Standards auch erweiterte Konzepte der Batch-Verarbeitung realisierbar sind. Die bereitgestellten Möglichkeiten der Parallelisierung von Verarbeitungsschritten nutzen moderne Mehrprozessor-Systeme effizient aus. Das Verzweigen von Verarbeitungsschritten mit Entscheidungslogik innerhalb der Batch-Verarbeitung ab und die Transition-Elemente stoppen die Verarbeitung an bestimmten Stellen kontrolliert. Beim Einsatz eines Applikationsservers wie Oracle WebLogic lässt sich der Status von Batch-Verarbeitungen über die mitgelieferte Administrationsoberfläche einsehen und kontrollieren. Das Entwickeln einer separaten Administrationsoberfläche entfällt.

Philipp Buchholz

philipp.buchholz@esentri.com



Philipp Buchholz ist Senior Consultant bei der esentri AG. Als Architekt konzipiert und implementiert er umfangreiche Enterprise-Software-Lösungen auf Basis von Java EE und Technologien aus dem Oracle-Middleware-Stack wie WebLogic und ADF. Bei der Modellierung komplexer Software setzt er auf die Techniken des Domain-driven-Design und der objektorientierten Analyse. Zusätzlich unterstützt er die Entwicklungsteams von Kunden als Scrum-Master.



Exploration und Visualisierung von Software-Architekturen mit jQAssistant

Dirk Mahler, buschmais GbR

Für die Arbeit in umfangreichen Code-Strukturen ist es sehr hilfreich, wenn sich Entwickler oder Architekten schnell einen Überblick über bestehende Strukturen und deren Beziehungen verschaffen können. Das Werkzeug jQAssistant leistet dabei wertvolle Unterstützung.

Der Ausdruck „gewachsene Strukturen“ versinnbildlicht oft deutlich – ob mit oder ohne ironischen Unterton – den Zustand von Java-Anwendungen, die im Unternehmensumfeld über Jahre hinweg gediehen sind und viele Entwickler kommen, aber auch wieder gehen gesehen haben. Ein stets wiederkehrendes Problem für die im Projekt Verbliebenen ist die Frage, welche grundlegenden Struktur-Einheiten im Code existieren, wie diese zueinander in Beziehung stehen und in welchem Maß sie mit der Architektur-

Dokumentation (sofern vorhanden) beziehungsweise den Vorstellungen in den Köpfen der Entwickler (hoffentlich vorhanden) übereinstimmen.

Das Open-Source-Werkzeug jQAssistant bietet die Möglichkeit, bestehende Software-Strukturen zu erfassen und in einer Graphen-Datenbank abzulegen. Über Abfragen können diese Informationen um Konzepte (etwa Abstraktionen wie „Modul“) angereichert sowie Reports erzeugt werden, die in textueller oder grafischer Form

dem Nutzer Einblicke in den Status quo seiner Anwendung ermöglichen.

Eureka!

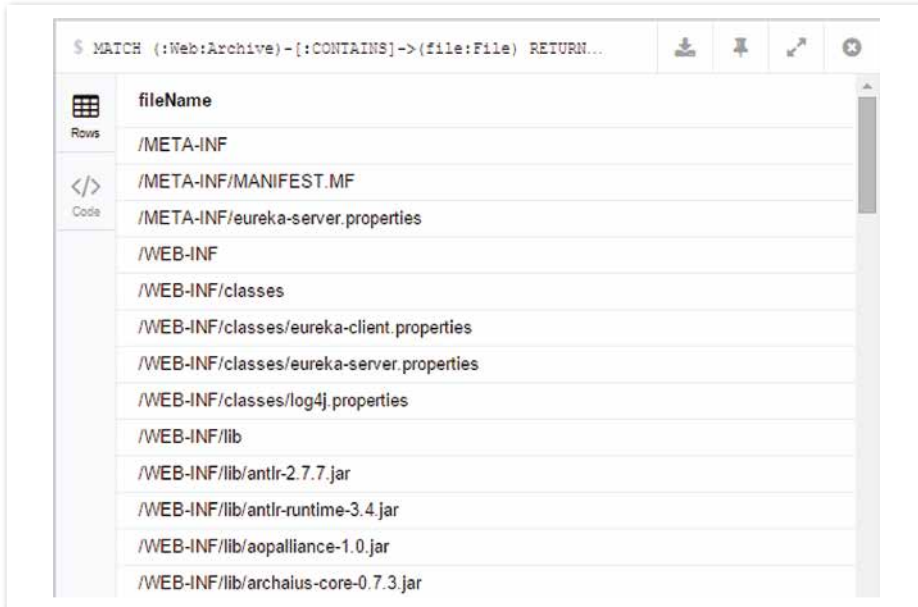
Das Unternehmen Netflix [1] stellt viele der von ihm entwickelten Lösungen unter Open-Source-Lizenzen zur Verfügung. Dazu zählt auch „Eureka“, ein Discovery-Dienst für Microservices, dessen Server als WAR-Artefakt via Maven-Central (groupId: com.netflix.eureka, artifactId: eureka-server) bezogen werden kann und analysiert werden soll. Die

```
bin jqassistant.sh scan -f eureka-server-1.4.2.war
```

Listing 1

```
MATCH (:Web:Archive)-[:CONTAINS]->(file:File) RETURN file.fileName as file-Name ORDER BY fileName
```

Listing 2

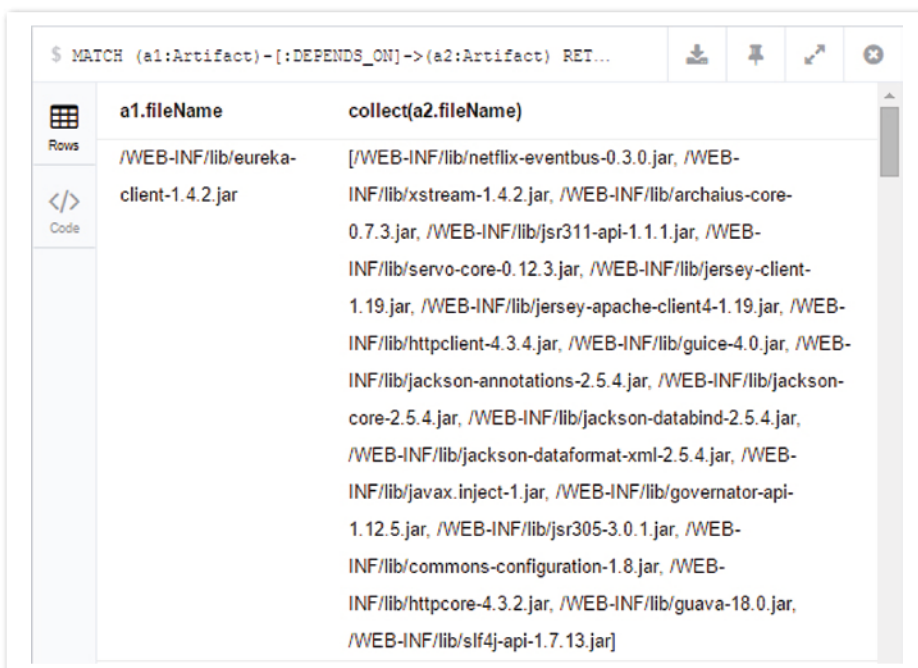


file:File	fileName
	/META-INF
	/META-INF/MANIFEST.MF
	/META-INF/eureka-server.properties
	/WEB-INF
	/WEB-INF/classes
	/WEB-INF/classes/eureka-client.properties
	/WEB-INF/classes/eureka-server.properties
	/WEB-INF/classes/log4j.properties
	/WEB-INF/lib
	/WEB-INF/lib/antlr-2.7.7.jar
	/WEB-INF/lib/antlr-runtime-3.4.jar
	/WEB-INF/lib/aopalliance-1.0.jar
	/WEB-INF/lib/archaius-core-0.7.3.jar

Abbildung 1: Dateien im WAR-Archiv

```
MATCH (a1:Artifact)-[:DEPENDS_ON]->(a2:Artifact) RETURN a1.fileName, collect(a2.fileName)
```

Listing 3



a1:Artifact	a2:Artifact
/WEB-INF/lib/eureka-client-1.4.2.jar	[/WEB-INF/lib/netflix-eventbus-0.3.0.jar, /WEB-INF/lib/xstream-1.4.2.jar, /WEB-INF/lib/archaius-core-0.7.3.jar, /WEB-INF/lib/jsr311-api-1.1.1.jar, /WEB-INF/lib/servo-core-0.12.3.jar, /WEB-INF/lib/jersey-client-1.19.jar, /WEB-INF/lib/jersey-apache-client4-1.19.jar, /WEB-INF/lib/httpclient-4.3.4.jar, /WEB-INF/lib/guice-4.0.jar, /WEB-INF/lib/jackson-annotations-2.5.4.jar, /WEB-INF/lib/jackson-core-2.5.4.jar, /WEB-INF/lib/jackson-databind-2.5.4.jar, /WEB-INF/lib/jackson-dataformat-xml-2.5.4.jar, /WEB-INF/lib/javax.inject-1.jar, /WEB-INF/lib/governator-api-1.12.5.jar, /WEB-INF/lib/jsr305-3.0.1.jar, /WEB-INF/lib/commons-configuration-1.8.jar, /WEB-INF/lib/httpcore-4.3.2.jar, /WEB-INF/lib/guava-18.0.jar, /WEB-INF/lib/slf4j-api-1.7.13.jar]

Abbildung 2: Beziehungen zwischen JAR-Dateien

jQAssistant-Distribution ist auf der Projekt-Seite [2] als ZIP-Archiv verfügbar und nach dem Entpacken kann die WAR-Datei mit dem Kommando „cd jqassistant.distribution-1.1.3“ gescannt werden (siehe Listing 1).

Als Ergebnis existiert nun im aktuellen Arbeitsverzeichnis eine Ordnerstruktur „jqassistant/store“, die die Datenbank beinhaltet. Jetzt kann mit „bin/jqassistant.sh server“ der integrierte Neo4j-Server gestartet werden. Die Oberfläche steht im Browser unter der URL „http://localhost:7474“ zur Verfügung und ermöglicht erste Explorationen mit Cypher [3], der Abfragesprache von Neo4j.

Artefakte und ihre Beziehungen

Zunächst stellt sich die Frage, welche Dateien überhaupt im WAR-Archiv enthalten sind (siehe Listing 2 und Abbildung 1). Es ist erkennbar, dass das Archiv keinerlei Klassen, dafür jedoch JAR-Dateien enthält. Es wäre interessant zu wissen, welche Abhängigkeiten zwischen diesen JARs bestehen. Diese Information ist in den erfassten Daten jedoch nicht direkt vorhanden, kann aber über ein Konzept, also eine vorgefertigte Abfrage, angereichert werden. Dies wird nach Stoppen des Servers auf der Kommandozeile mit „<Enter>“ durch den Befehl „bin/jqassistant.sh analyze -concepts dependency:Artifact“ erreicht.

Das Konzept „dependency:Artifact“ wird mit dem Java-Plug-in von jQAssistant ausgeliefert und erzeugt eine Beziehung „DEPENDS_ON“ zwischen zwei Artefakten „a1“ und „a2“ (etwa JAR-Dateien), wenn „a1“ einen Java-Typen beinhaltet, der eine Abhängigkeit zu einem Java-Typen in „a2“ besitzt. Nach erneutem Starten des Servers können diese Beziehungen über eine Abfrage ermittelt werden (siehe Listing 3 und Abbildung 2).

Die präsentierten Informationen sind zwar vollständig und korrekt, eine grafische Repräsentation wäre jedoch wesentlich anschaulicher. Hier kann das GraphML-Report-Plug-in von jQAssistant zum Einsatz kommen, um das Ergebnis eines Konzepts als GraphML-Datei zu exportieren, die wiederum durch Werkzeuge wie yEd [4] oder Gephi [5] visualisiert werden kann.

Mehr Graph bitte!

Konzepte werden in XML- oder AsciiDoc-Dateien hinterlegt, für das Beispiel im letzteren Format. Hierzu wird relativ zum aktuellen Arbeitsverzeichnis eine Datei „jqassistant/

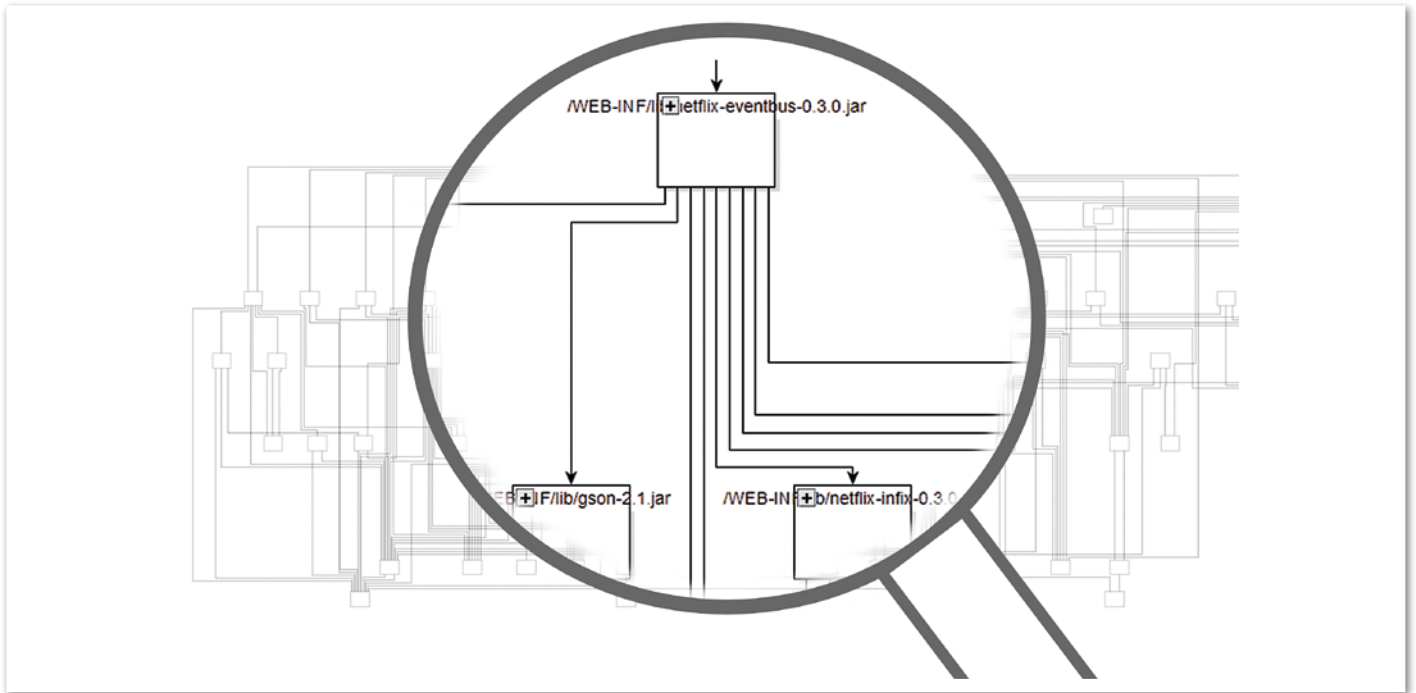


Abbildung 3: Visualisierung der Beziehungen zwischen JAR-Artefakten

rules/eureka.adoc“ [6] angelegt (siehe Listing 4).

AsciiDoc [7] ist eine einfach zu erlernende, aber gleichzeitig mächtige Markup-Language zum Erstellen von HTML- oder PDF-Dokumenten, die unter anderem von jqAssistant verwendet wird, um darin ausführbare Regeln wie Konzepte einzubetten. Im konkreten Beispiel handelt es sich um einen Cypher-Code-Block mit einer ID, einer kurzen Beschreibung sowie der Meta-Information, dass dieses Konzept die vorhergehende Ausführung des Konzepts „dependency:Artifact“ voraussetzt. Der „graphml“-Suffix der ID „artifactDependencies.graphml“ wird vom GraphML-Report-Plugin erkannt und als Hinweis darauf gewertet, dass das Ergebnis exportiert werden soll.

Der Aufruf von „bin/jqassistant.sh analyze -concepts artifactDependencies.graphml“ erzeugt im Verzeichnis „jqassistant/report“ eine Datei „artifactDependencies.graphml“, die mit yEd geöffnet werden kann und nach Anwendung eines hierarchischen Layouts die in Abbildung 3 gezeigte Visualisierung erzeugt.

Der Graph ist recht umfangreich und auch nur bedingt übersichtlich. Jedoch ist bereits erkenntlich, dass Artefakte mit ausgehenden Abhängigkeiten oben dargestellt werden (Netflix-Artefakte wie „eureka-core“ oder „netflix-eventbus“). Solche mit eingehenden Abhängigkeiten sind entsprechend weiter unten zu finden (hauptsächlich Bibliotheken und APIs wie „stax-api“ oder „http-core“).

```
= Eureka Example
== Artifact Dependencies
[[artifactDependencies.graphml]]
.Creates a GraphML report for artifact dependencies
[source,cypher,role=concept,requiresConcepts="dependency:Artifact"]
----
MATCH (:Web:Archive)-[:CONTAINS]->(artifact:Artifact)
OPTIONAL MATCH
  (artifact)-[dependsOn:DEPENDS_ON]->(:Artifact) // equivalent to left-outer-join in SQL
RETURN artifact, dependsOn
----
```

Listing 4

```
MATCH (:Web:Archive)-[:CONTAINS]->(artifact)-[:CONTAINS]->(package:Package)
RETURN artifact.fileName, collect(package.fqn)
```

Listing 5

```
[[internalArtifact]]
.Labels all artifacts containing the package "com.netflix" as "Internal".
[source,cypher,role=concept]
----
MATCH (:Web:Archive)-[:CONTAINS]->(artifact:Artifact)-[:CONTAINS]->(package:Package)
WHERE package.fqn = "com.netflix"
SET artifact:Internal
RETURN artifact
----

[[internalArtifactDependencies.graphml]]
.Creates a GraphML report for internal artifact dependencies.
[source,cypher,role=concept,requiresConcepts="dependency:Artifact,internalArtifact"]
----
MATCH (artifact:Artifact:Internal)
OPTIONAL MATCH (artifact)-[dependsOn:DEPENDS_ON]->(:Artifact:Internal)
RETURN artifact, dependsOn
----
```

Listing 6

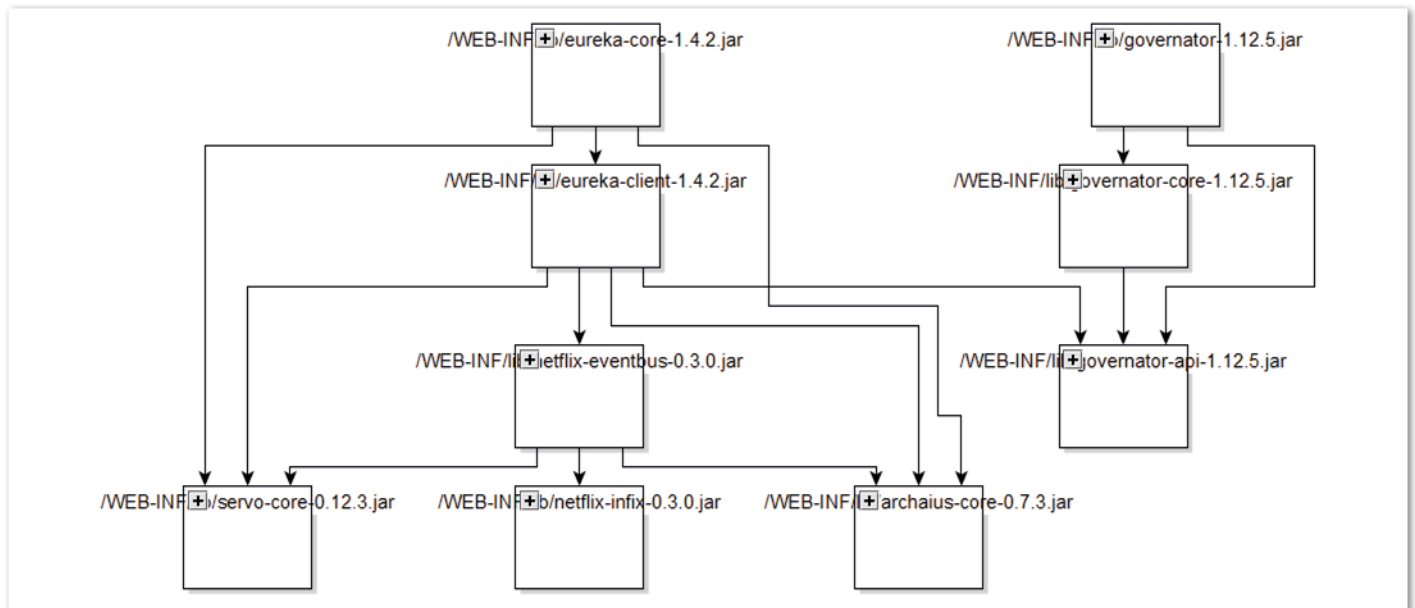


Abbildung 4: Visualisierung der Beziehungen zwischen internen Artefakten

Weniger ist manchmal mehr

Letztere spielen für eine erste Analyse meist nur eine untergeordnete Rolle – die eigenen Artefakte und deren Abhängigkeiten sind oft von höherer Bedeutung. Es wird in diesem Zusammenhang oft von internen und externen Abhängigkeiten gesprochen (etwa im Kontext von Maven-Projekten). Doch wie können diese voneinander unterschieden werden?

Im Beispiel wird der Eureka-Server von Netflix untersucht – es ist naheliegend, dass interne Artefakte Package-Strukturen mit entsprechenden Namensmustern aufweisen. In der Tat ist es so, dass im Neo4j-Server eine Abfrage für Netflix-eigene Artefakte jeweils Packages mit dem Prefix „com.netflix“ liefert (siehe Listing 5). Mit diesem Wissen lassen sich in „eureka.adoc“ zwei weitere Konzepte erstellen (siehe Listing 6).

Das Konzept „internalArtifact“ versieht alle Artefakte, die ein Package mit dem Namen „com.netflix“ enthalten, mit dem Label „Internal“. Das darauf aufbauende Konzept „internalArtifactDependencies.graphml“ nutzt dieses, um die Abfrage für den GraphML-Report filtern zu können. Letzterer wird durch den Aufruf „bin/jqassistant.sh analyze -concepts internalArtifactDependencies.graphml“ erzeugt.

Wird die erzeugte Datei „jqassistant/report/internalArtifactDependencies.graphml“ in yEd geöffnet und das hierarchische Layout angewendet, ergibt sich *Abbildung 4*, die den Ansprüchen einer Architektur-Visualisierung bereits gerecht wird.

Fazit

jqAssistant ermöglicht es, Artefakte zu scannen und über Abfragen zu untersuchen. Daraus können bei Bedarf textuelle oder graphische Reports („GraphML“) erzeugt werden. Dem Anwender wird dabei jegliche Freiheit gegeben, die Daten nach seinen individuellen Bedürfnissen anzureichern (wie interne vs. externe Abhängigkeiten) und die Ausgaben entsprechend zu filtern. Über das Anlegen sogenannter „Konzepte“ sind diese Vorgänge automatisierbar und können bei Bedarf auch in einen Build-Prozess integriert werden. Damit ist kontinuierliches Reporting über den aktuellen Zustand einer Architektur ermöglicht.

Das demonstrierte Vorgehen auf der Ebene von Artefakten stellt im Übrigen nur die sehr kleine Spitze des Eisberges dar. Es sind weitere Abhängigkeits-Analysen denkbar, etwa das Auffinden von Querschnitts-Aspekten (Logging, Injection) oder die Prüfung, welche Technologien überhaupt zum Einsatz kommen (etwa Hibernate oder JAX-RS) oder welche Auswirkungen das Anheben einer Bibliotheksversion haben könnte.

Bei Bedarf können die Abfragen auch auf Package- oder Klassenebene heruntergebrochen werden. Ein Beispiel dafür ist ein Report, der aufzeigt, welche Teile einer umfangreichen Bibliothek (etwa Guava) tatsächlich verwendet werden. Der Fantasie sind kaum Grenzen gesetzt.

Links

- [1] Netflix: <http://netflix.com>
- [2] jqAssistant: <http://jqassistant.org>

- [3] Cypher: <http://neo4j.com/docs/stable/cypher-query-lang.html>
- [4] yEd: <https://www.yworks.com/products/yed>
- [5] Gephi: <https://gephi.org>
- [6] Gist: <https://gist.github.com/DirkMahler/8b5b8551de275cbe76c1>
- [7] AsciiDoc: <http://www.methods.co.nz/asciidoc>

Dirk Mahler

dirk.mahler@buschmais.com



Dirk Mahler ist Senior-Consultant bei der buschmais GbR, einem Beratungshaus mit Sitz in Dresden. Der Schwerpunkt seiner mehr als zehnjährigen Tätigkeit liegt im Bereich Architektur und Entwicklung von Java-Applikationen im Unternehmensumfeld. Den Fokus setzt er dabei auf die Umsetzung von Lösungen, die im Spannungsfeld zwischen Pragmatismus, Innovation und Nachhaltigkeit liegen. In diesem Rahmen engagiert er sich für das Open-Source-Projekt jqAssistant.



Fallstricke bei der sicheren Entwicklung von Java-Web-Anwendungen

Dominik Schadow, BridgingIT GmbH

Bei der sicheren Entwicklung von Web-Anwendungen zeichnet sich eine ähnliche Entwicklung ab wie bei deren Entwurf: eine gewisse Standardisierung und empfohlene sowie erprobte Vorgehensweisen. Wo Empfehlungen existieren, sind die zugehörigen Antipatterns und Fallstricke allerdings nicht weit. Diese bedrohen die mühsam errungene Sicherheit einer Web-Anwendung und können die umgesetzten Maßnahmen gleich wieder zunichtemachen.

Zur sicheren Entwicklung von Java-Web-Anwendungen existieren mittlerweile zahlreiche Empfehlungen und Vorgehensweisen, etwa vom Open Web Application Security Project (OWASP) [1]. Man denke nur an das Output Escaping zur Vermeidung von Cross-Site Scripting (XSS) oder an Prepared Statements zur Verhinderung von SQL-Injections. Derartig konkrete Empfehlungen sind meist vergleichsweise einfach in vielen Web-Anwendungen umsetzbar. Sie sind stark standardisiert und mehr oder weniger überall gleich umzusetzen. Als Entwickler muss man sich nur der Bedrohung bewusst sein und die zugehörige Gegenmaßnahme kennen und umsetzen.

An anderer Stelle werden Empfehlungen rund um die sichere Entwicklung notgedrungen sehr viel allgemeiner. Beispiele hierfür sind die Benutzerverwaltung oder das Session Management. Zwar hat nahezu jede Web-Anwendung mit diesen Aufgaben zu tun; die konkrete Implementierung, Konfiguration und angebundene Systeme unterscheiden sich allerdings meist deutlich. Direkt einsetzbare Empfehlungen werden damit schwierig beziehungsweise zielen immer nur auf bestimmte Komponenten oder Frameworks ab.

Trotz dieser Einschränkung haben sich einige Muster herauskristallisiert, die man vor, während und nach der Entwicklung berücksichtigen sollte. Nur ist es – wie auch bei den normalen Design-Patterns – allzu leicht möglich, einmal falsch abzubiegen

und aus einem eigentlich sicheren Pattern ein unsicheres zu machen.

Erschwerend kommt im Umfeld der sicheren Software-Entwicklung das subjektive Sicherheitsgefühl hinzu. Als Entwickler merkt man ja durchaus, ob eine Web-Anwendung sich sicher anfühlt oder nicht. Wendet man nun ein Security-Pattern bei der Entwicklung an, verstärkt sich das Gefühl, dass man alles Notwendige zur Absicherung der Web-Anwendung getan hat. Warnzeichen – etwa eine bekanntgewordene Sicherheitslücke in einer anderen Web-Anwendung – werden in der Folge leichter ignoriert; man hat die Web-Anwendung ja sicher entwickelt. Der Schein trügt allerdings, sie ist längst nicht so sicher wie gedacht und eigentlich möglich.

Der Artikel stellt einige ausgewählte Fallen und Antipatterns vor, die häufig die Sicherheit einer Web-Anwendung bedrohen. Diese Liste ist natürlich bei Weitem nicht vollständig. Zum einfacheren Verständnis lassen sich die folgenden Antipatterns grob in die Projektphasen „Architektur“, „Implementierung“ und „Wartung“ einteilen. Da für Entwickler der Schwerpunkt auf der Implementierung einer Web-Anwendung liegt, liegt dort auch der Schwerpunkt dieses Artikels. Ganz ohne Architektur und Wartung geht es aber auch bei der Sicherheit nicht.

Architektur

Security-Antipatterns lassen sich in den meisten Projekten schon in der ersten Pha-

se finden. Was sich beim Design einer Web-Anwendung längst durchgesetzt hat – etwa UML-Diagramme oder Dokumentation mit „arc42“ – steckt bei der Planung der Sicherheit noch immer in den Kinderschuhen.

Tatsächlich wird die Sicherheit einer Web-Anwendung allzu häufig bis kurz vor dem Release aufgeschoben; Sicherheit verlangsamt schließlich die Entwicklung und macht alles unnötig kompliziert. Von einer inhärenten Sicherheit der Anwendung kann daher keine Rede sein. Stattdessen wirkt die Sicherheit wie ein zusätzlicher Layer, der die Anwendung möglichst vollständig umgeben und absichern soll. Ganz im Sinne einer Firewall, die einen Schutzkreis um das Netzwerk (in diesem Fall die Anwendung) zieht. Security Flaws, also tiefliegende Sicherheitsprobleme aufgrund mangelnder Planung, sind damit nahezu vorprogrammiert. Die Gefahr ist groß, dass man bei dieser späten Absicherung einen Eingangskanal wie etwa einen konsumierten fremden Webservice übersieht und Daten ungeprüft in die eigene Anwendung gelangen.

Dabei ist die Vermeidung dieses Antipatterns so einfach: Die Sicherheit einer Web-Anwendung muss wie deren Design vor der Implementierung geplant werden. Die UML hilft hier nur sehr begrenzt weiter, notwendig sind Threat Models [2]. Diese sind im Gegensatz zur UML wenig standardisiert; als Architekt oder Entwickler trifft man daher in jedem

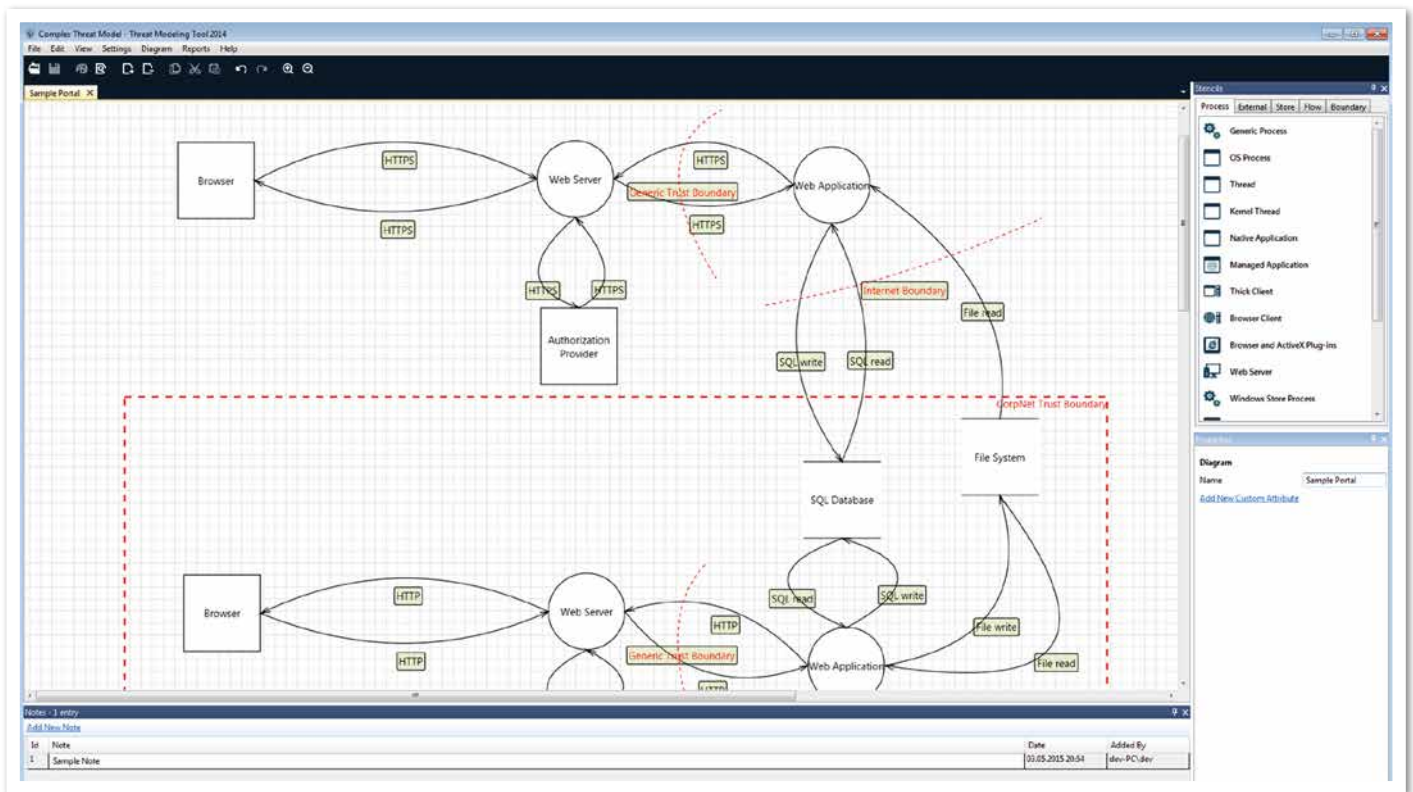


Abbildung 1: Microsoft Threat Modeling Tool 2014

Tool auf andere grafische Elemente. Auch wenn ohnehin nur wenige produktionsreife Tools zur Verfügung stehen, sollte man sich daher auf ein Tool beschränken und diesem möglichst immer treu bleiben. Das selbst für Java-Web-Anwendungen empfehlenswerte Tool ist das Microsoft Threat Modeling Tool 2014 [3] (siehe Abbildung 1).

Dieses Tool steht im Rahmen des Microsoft Secure Development Lifecycle kostenlos zur Verfügung und ist – zumindest bei der Modellierung – unabhängig von der eingesetzten Programmiersprache. Bei der Erstellung des Modells folgt man idealerweise den Daten, in einer Web-Anwendung ist der Startpunkt daher häufig der Webbrowser (ebenso wie der Endpunkt). Die vom Benutzer eingegebenen Daten wandern durch das System, kommunizieren/interagieren mit anderen Entitäten oder werden mit weiteren Informationen angereichert. Alle diese Entitäten – Datenbanken, Webserver, Benutzerverwaltung etc. – werden in das Threat Model eingetragen.

Im nächsten Schritt entstehen Trust Boundaries, im Beispiel die rot gestrichelten Linien. An diesen Stellen müssen etwa aus Benutzereingaben vertrauenswürdige Daten werden, damit diese sicher weiterverarbeitet werden können. Die Eingabedaten müssen also validiert werden. Eine weitere Trust Boundary stellt einen Kommunika-

tionsvorgang dar, an dem Daten aus dem Internet in das Intranet übertragen werden. Das Threat Model kann dadurch bei großen Web-Anwendungen mit zahlreichen externen Entitäten durchaus komplex werden

und sollte daher in mehrere Threat Models aufgeteilt werden.

Gerade an den Boundaries wird deutlich, dass zwingend alle externen Entitäten im Model erfasst werden müssen. Ein Angrei-

```
byte[] hash(char[] password, byte[] salt) throws Exception {
    SecretKeyFactory skf =
        SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
    PBEKeySpec spec = new PBEKeySpec(password, salt, 10000, 512);

    return skf.generateSecret(spec).getEncoded();
}
```

Listing 1

```
@Bean
public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder(10);
}
```

Listing 2

```
public void doFilter(ServletRequest req, ServletResponse res,
    FilterChain filterChain) throws Exception {
    response.addHeader("Strict-Transport-Security",
        "max-age=31536000; includeSubDomains");
    filterChain.doFilter(req, response);
}
```

Listing 3

fer steht andernfalls nur vor der Aufgabe, die eine vergessene (und damit ungeschützte) Entität zu finden und für einen Angriff auszunutzen.

Das vollständige Threat Model legt die Basis für die nun folgende Implementierung. Viele konkrete Bedrohungen der eigenen Web-Anwendung sind dadurch bekannt, beginnend bei XSS und SQL-Injections über die Eingabevalidierung bis hin zu applikationsspezifischen Bedrohungen. Gleich wie bei UML-Diagrammen hat der Architekt in der Regel auch hier die Aufgabe, das Model zu pflegen und bei Änderungen der Anwendung zu aktualisieren.

Implementierung

Bei der Implementierung ist es aufgrund der Vielzahl unterschiedlicher Anwendungen, Frameworks und Systeme schwierig, verbreitete Antipatterns und deren korrektes Pattern allgemeingültig aufzuzeigen. Die folgenden Abschnitte konzentrieren sich daher auf Aufgaben rund um das Login eines Benutzers bis hin zur Erstellung seiner Session. Der vollständige Quellcode steht auf GitHub [4].

Passwörter speichern

Trotz Single Sign-on (SSO) besitzen viele Web-Anwendungen ihre eigene Benutzerverwaltung. Wo eine Umstellung auf ein zentrales SSO-System nicht möglich ist, haben diese Anwendungen die Aufgabe, Benutzer-Passwörter sicher zu speichern. Die Zeiten, in denen diese im Klartext gespeichert wurden, sind zum Glück längst vorbei. Auch dass die Passwort-Verschlüsselung ein Antipattern ist, hat sich längst herumgesprochen: Ein verschlüsseltes Passwort kann schließlich wieder entschlüsselt werden, und das ist niemals notwendig. Das Hashen von Passwörtern vor dem Speichern ist daher die einzig richtige Vorgehensweise.

Nun gilt es, unter den vielen verfügbaren Hashing-Algorithmen den richtigen auszuwählen. MD5, SHA1 und die gesamte SHA2-Familie sind hierfür nicht die richtige Wahl (und wurden teilweise auch schon längst als unsicher entlarvt). Diese auf Geschwindigkeit optimierten Hashing-Algorithmen machen es einem Angreifer zu leicht, viele Millionen Passwörter pro Sekunde per Brute-Force-Angriff auszuprobieren. Stattdessen gilt es – wohl nahezu einmalig in der Software-Entwicklung –, den Hashvorgang gezielt zu verlangsamen und so einen Angriff auszubremsen. Für einen einzelnen Benutzer ist eine Verzögerung um einige Millisekunden nicht spürbar, für einen Angreifer dagegen schon.

```
@WebServlet
public class LoginServlet extends HttpServlet {
    protected void doPost(HttpServletRequest req,
        HttpServletResponse res) throws ServletException {
        request.changeSessionId();
    }
}
```

Listing 4

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.owasp</groupId>
      <artifactId>dependency-check-maven</artifactId>
      <version>1.3.3</version>
      <reportSets>
        <reportSet>
          <reports>
            <report>aggregate</report>
          </reports>
        </reportSet>
      </reportSets>
    </plugin>
  </plugins>
</reporting>
```

Listing 3

Schließlich treffen ihn die beispielsweise 100 Millisekunden Verzögerung nicht nur einmalig, sondern gleich millionenfach.

Zum sicheren Hashen von Passwörtern müssen daher die Algorithmen Password-Based Key Derivation Function 2 (PBKDF2), „bcrypt“ oder „scrypt“ verwendet werden. PBKDF2 steht ohne weitere Bibliotheken direkt in plain Java zur Verfügung und lässt sich sehr einfach einsetzen (siehe Listing 1).

„10000“ bestimmt dabei die Anzahl der Iterationen, „512“ die Hash-Länge. Gerade bei der Anzahl der Iterationen tappt man gerne in die nächste Falle. Dieser Wert muss sich mit Verbesserung der Hardware immer weiter erhöhen, damit Angriffe weiterhin wirksam aufgehalten werden können. Wichtig ist bei einer Erhöhung der Anzahl von Iterationen, dass sämtliche Benutzer-Passwörter innerhalb einer vorgegebenen Zeit aktualisiert werden.

Sobald sich ein Benutzer erfolgreich eingeloggt hat, muss sein Passwort mit einem neuen Salt mit der neuen Anzahl von Iterationen gehasht und anschließend als neuer Hash gespeichert werden. Wer sich innerhalb dieser Zeitspanne nicht anmeldet, sollte deaktiviert werden und muss seinen Benutzer später etwa per „Passwort vergessen“ erneut aktivieren. Das endlose Speichern von potenziell unsicher gehashten Passwörtern birgt ansonsten ein enormes Risiko, sollte ein Angreifer doch einmal Zugang zu den Benutzerdaten erlangen.

Vereinfachen lässt sich das sichere Speichern von Benutzer-Passwörtern mit Spring Security [5]. Spring Security unterstützt out of the box die Verwendung von „bcrypt“ und muss dazu lediglich um eine Bean in einer mit „@Configuration“ annotierten Konfigurationsklasse erweitert werden (siehe Listing 2).

Die „10“ als Konstruktor-Argument steht dabei nicht für die Anzahl von Iterationen, sondern bestimmt die Komplexität der Berechnung. „10“ ist der Default, Werte zwischen „4“ und „31“ sind möglich. Die Komplexität steigt dabei exponentiell an.

Eine weitere Alternative ist „scrypt“, das im Gegensatz zu den beiden zuvor genannten Algorithmen nicht Iterationen, sondern einen Mindest-Speicherverbrauch als Schutz vor Brute-Force-Angriffen verwendet.

Sichere Datenübertragung

Häufig verwenden Web-Anwendungen das ungesicherte HTTP zur Datenübertragung, solange keine Benutzerdaten (Benutzernamen und Passwort beziehungsweise Session ID) im Spiel sind. Selbst das Login-Formular wird noch per HTTP ausgeliefert, überträgt seine Daten aber sicher an eine HTTPS-URL. Die sichere Datenübertragung ist damit gewährleistet, allerdings verschenkt man damit die Integrität des Login-Formulars. Per „Man in the middle“-Angriff kann dann so das Formularziel manipuliert worden sein

und die Daten damit zu einem vom Angreifer kontrollierten Server übertragen werden.

Erschweren lässt sich dieser Angriff durch den konsequenten Einsatz von HTTPS. Um den Browser zu dessen Verwendung zu zwingen, sollten HTTPS-Seiten immer mit dem HTTP Strict Transport Security Header (HSTS) ausgeliefert werden. Mit Spring Security wird auch diese Anforderung automatisch für per HTTPS ausgelieferte Seiten umgesetzt [6], ohne Spring Security fügt der folgende ServletFilter diesen Header zur Response hinzu (siehe Listing 3).

Von nun an wird die gesamte Seite nach einer einzigen initialen Auslieferung per HTTPS nur noch über das sichere Protokoll ausgeliefert. Jeder moderne Browser sorgt dann dafür, dass selbst URL-Eingaben ohne Protokoll sofort auf HTTPS umgeleitet werden. Das Abfangen einer Session-ID oder anderer Informationen ist damit nicht mehr möglich.

Die Session-ID nach dem Login ändern

Unter Entwicklern hält sich hartnäckig das Gerücht, dass ein Benutzer erst dann eine

Session-ID erhält, nachdem er sich erfolgreich bei der Anwendung angemeldet hat. Richtig ist dagegen, dass ein Benutzer sie nahezu immer sofort beim Aufruf einer Seite bekommt. Bis zum erfolgreichen Login ist eine dem anonymen Benutzer zugewiesene Session-ID für einen Angreifer allerdings wertlos. Nach dem Login ist sie jedoch mit einem Benutzer verknüpft und damit wertvoll geworden. Ein Angreifer mit Kenntnis dieser ID kann sich gegenüber der Anwendung nun als dieser Benutzer ausgeben.

Mittels Session Fixation und Session Hijacking stehen einem Angreifer gleich zwei Angriffsmöglichkeiten zur Verfügung. Bei der Session Fixation schiebt ein Angreifer einem Benutzer eine ihm bekannte Session-ID unter und wartet, bis dieser sich angemeldet hat. Beim Session Hijacking stiehlt der Angreifer die Session-ID, beispielsweise bei einer ungesicherten Übertragung per HTTP.

Zur Vermeidung dieses Problems muss die Session-ID nach einem erfolgreichen Login immer geändert werden. Spring Security erledigt auch diese Aufgabe automatisch,

ganz ohne notwendige weitere Konfiguration. Ohne Spring Security hilft ein Servlet weiter, das unmittelbar nach erfolgreichem Login die Session-ID ändert (siehe Listing 4).

Wartung

In wohl kaum einer Programmiersprache ist der Einsatz von fremden Bibliotheken so verbreitet wie in Java. Mehr und mehr mitunter auch kritische Funktionalität wird in Bibliotheken ausgelagert und nicht mehr selbst entwickelt. Auf der einen Seite ist das natürlich überaus begrüßenswert, Standard-Entwicklungsaufgaben werden schließlich nur von den wenigsten Entwicklern gerne übernommen. Und sicherheitsrelevante Funktionalität überlässt man besser einer zig-fach erprobten Bibliothek, als diese selbst zu implementieren.

Auf der anderen Seite steigen die Anforderungen an die eingesetzten Bibliotheken, schließlich übernehmen diese kritische Aufgaben, die man sonst eigentlich selbst entwickelt hätte. Damit wird es wichtig, alle in einer Anwendung vorhandenen Bibliotheken aktuell zu halten und verwundbare Ver-

Community-Konferenz organisiert von Java User Groups aus dem Norden

<http://javaforumnord.de> @JavaForumNord



JAVA FORUM NORD



jügh!



JUG HANNOVER



SUG



Hannover, Donnerstag 20. Oktober 2016

sionen zeitnah durch verbesserte Versionen auszutauschen.

Bei der üblich großen Menge an eingesetzten Bibliotheken ist das allerdings gar nicht so einfach. Mit OWASP Dependency Check [7] steht zur Unterstützung ein sehr empfehlenswertes Tool zur Verfügung, das Entwickler über verwundbare Bibliotheken in der Web-Anwendung informiert. Hierfür vergleicht Dependency Check die identifizierten Bibliotheken mit Einträgen in der National Vulnerability Database (NVD) und erstellt einen entsprechenden Report. Neben dem einfachen Aufruf im Terminal und der Integration in den Maven Build (siehe Listing 5) steht eine Integration in Jenkins zur Verfügung.

In der Standard-Konfiguration prüft Dependency Check nach jedem erfolgreichen Build. Damit das nicht zu lange dauert, empfiehlt es sich, das NVD-Update in diesen Jobs durch Aktivieren von „Disable NVD auto-update“ auszuschalten (siehe Abbildung 2).

Zusätzlich richtet man einen weiteren Job ein, der regelmäßig nur ein Update der lokalen NVD durchführt. Alle anderen Jobs verwenden diese gemeinsame Datenbank und werden dadurch spürbar schneller aus-

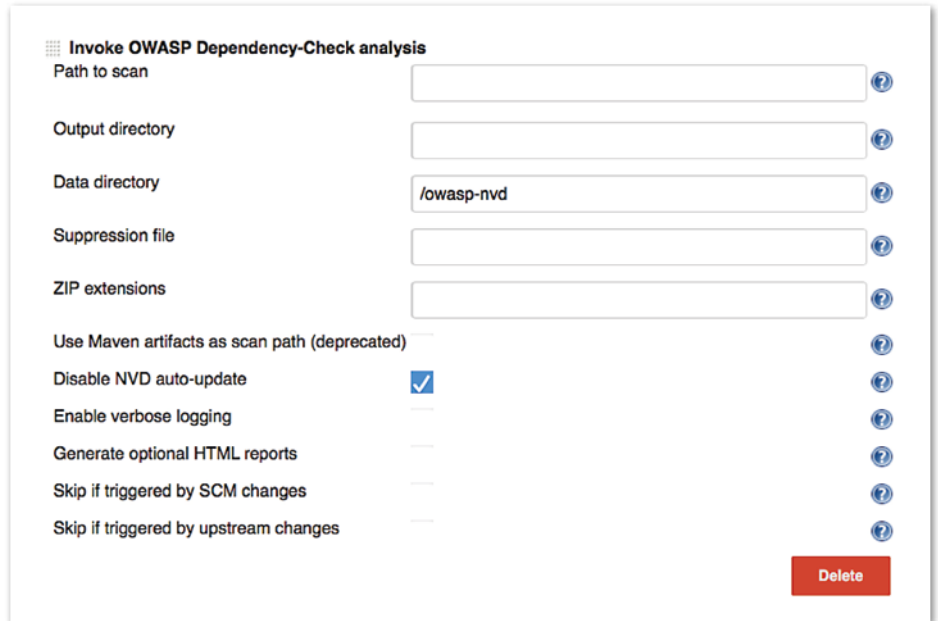


Abbildung 2: OWASP Dependency Check in Jenkins

geführt. Unabhängig von der gewählten Variante (Terminal, Maven, Jenkins) generiert Dependency Check einen HTML-Report mit den identifizierten möglichen Verwundbarkeiten (siehe Abbildung 3).

Dieser Report enthält häufig „false positives“, etwa wenn eine Bibliothek aufgrund

ihres Namens falsch erkannt wurde. Beim Durcharbeiten der Liste gilt es daher, die Meldungen zu hinterfragen – bevor man sich an das Austauschen der Bibliothek macht. Die sich an den Austausch einer Bibliothek anschließenden Unit- und Integrationstests übernimmt Dependency Check zwar nicht,

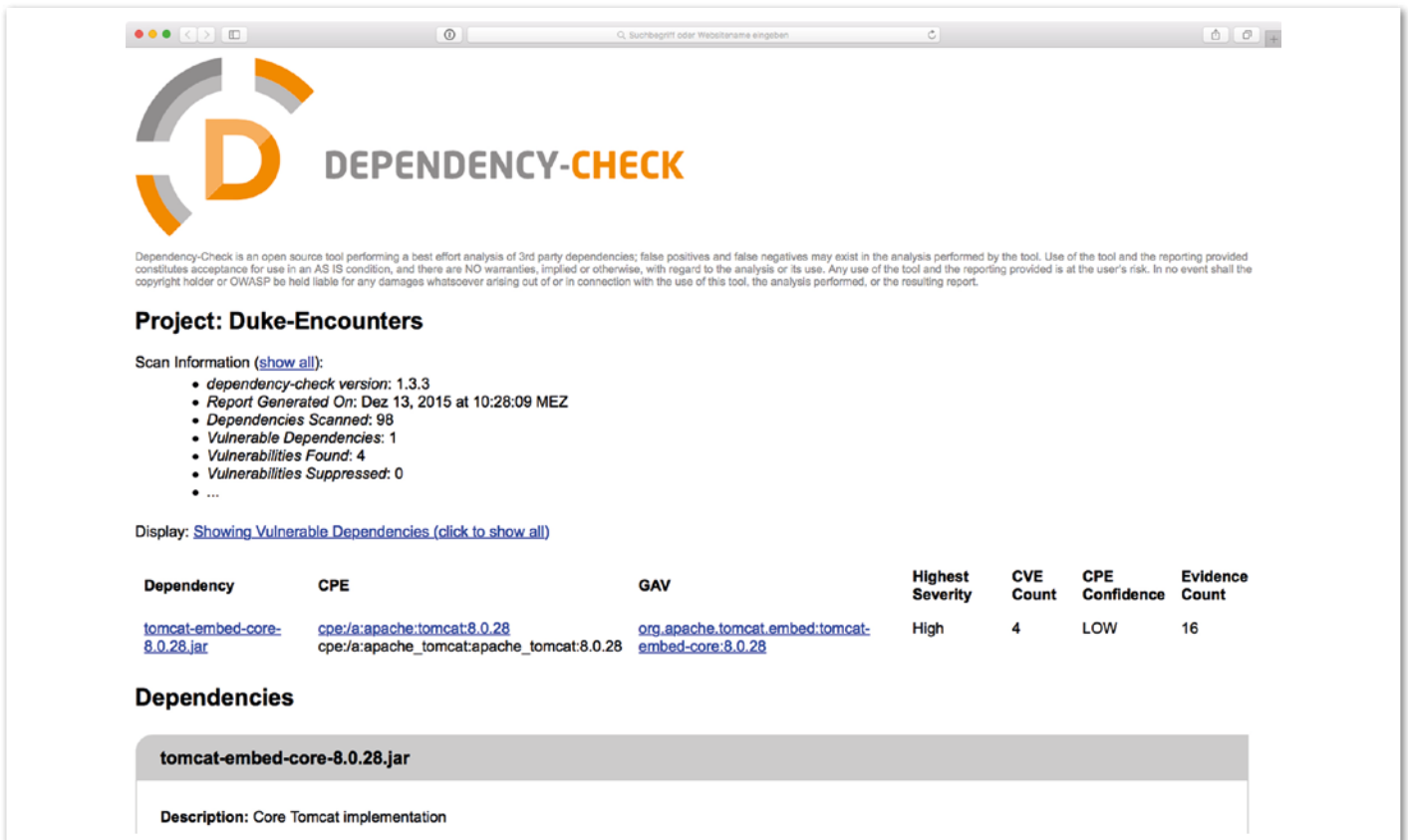


Abbildung 3: OWASP Dependency Check Report

unterstützt Entwickler aber immerhin bei der Identifikation von unsicheren Bibliotheken.

Fazit

Trotz der immer weiteren Standardisierung bei der Entwicklung von sicheren Web-Anwendungen und immer mehr empfohlenen Vorgehensweisen ist und bleibt die Sicherheit einer Web-Anwendung ein komplexes Thema. Die Empfehlungen, vor Beginn der Implementierung ein Threat Model zu erstellen und währenddessen beziehungsweise danach die Aktualität der Bibliotheken zu überwachen und gegebenenfalls auszutauschen, lassen sich ohne große Anpassungen in jedem Projekt einsetzen.

Schwieriger wird es bei konkreten Implementierungsaufgaben. Hier sind die Anwendungen einfach zu unterschiedlich, empfohlene Vorgehensweisen (und damit auch

typische Antipatterns) hängen stark von den angebundenen Systemen und eingesetzten Frameworks ab. Viele der typischen Fallstricke und Antipatterns lassen sich allerdings mit den richtigen Frameworks automatisch vermeiden. Wer diese nicht einsetzen kann oder will, muss bei der Entwicklung entsprechend stärker auf die gezeigten (und die anderen) Fallen achten.

Ressourcen

- [1] <https://www.owasp.org>
- [2] Adam Shostack – Threat Modeling: Designing for Security – ISBN 978-1118809990
- [3] <http://blogs.microsoft.com/cyber-trust/2014/04/15/introducing-microsoft-threat-modeling-tool-2014/>
- [4] <https://github.com/dschadow/JavaSecurity>
- [5] <http://projects.spring.io/spring-security>
- [6] <http://docs.spring.io/spring-security/site/docs/current/reference/html/headers.html>
- [7] https://www.owasp.org/index.php/OWASP_Dependency_Check

Dominik Schadow

dominik.schadow@bridging-it.de



Dominik Schadow arbeitet als Senior Consultant beim IT-Beratungsunternehmen bridgingIT. Er hat mehr als zehn Jahre Erfahrung in der Java-Entwicklung und -Beratung. Sein Fokus liegt auf der Architektur und Entwicklung von Java-Enterprise-Applikationen und der sicheren Software-Entwicklung mit Java. Er ist regelmäßiger Speaker auf verschiedenen Konferenzen, Buchautor und Autor zahlreicher Fachartikel.

Java ist auch eine Insel – Einführung, Ausbildung, Praxis

gelesen von *Daniel Grycman*

Mittlerweile ist die zwölfte aktualisierte Auflage des deutschsprachigen Standardwerks von Christian Ullenboom im Rheinwerk Verlag erschienen. Wie der Autor im Vorwort schreibt, sind hauptsächlich Fehlerkorrekturen vorgenommen worden. Als einzige Neuerung ist das bisherige zwölfte Kapitel aufgespalten. Im neuen dreizehnten Kapitel geht Ullenboom auf die kommende Modularisierung in Java 9 ein.

Das Buch besticht durch sein ausführliches Inhaltsverzeichnis, das die Inhalte in 23 Kapitel unterteilt. Im Vorwort bietet Christian Ullenboom eine mögliche Lernstrategie zum Lesen und Lernen an. Er bezieht sich hierbei auf die „PQ4R“-Methode. Diese soll einem Anfänger im Bereich der Java-Programmierung den Einstieg und den Wissenserwerb erleichtern.

In den ersten 13 Kapiteln wird dem Leser Java als Programmiersprache nähergebracht. Sie stellen zugleich auch den ersten Teil des Buches dar. Im zweiten Teil beschäftigt sich der Autor mit grundlegenden APIs. Allen Kapiteln ist gemein, dass an deren Ende ein Abschnitt mit dem Titel „Zum Weiterlesen“ steht. Dieser bietet dem Leser die Möglichkeit, sich mit einem bestimmten Thema weiter auseinanderzusetzen. Den Abschluss des Werkes bilden eine ausführliche Übersicht über die in Java 8 enthaltenen Pakete sowie ein Index.

Die neueste Auflage der „Insel“ stellt eine absolute Kaufempfehlung für Java-Neulinge dar. Das Buch präsentiert auf eine verständliche Art und Weise die notwendigen Techniken zur Java-Programmierung und gibt auch praxisorientierte Informationen an den Leser weiter.



Titel:	Java ist auch eine Insel – Einführung, Ausbildung, Praxis
Autor:	Christian Ullenboom
Auflage:	12. Auflage, 2016
Verlag:	Rheinwerk Verlag
Umfang:	1.312 Seiten
Preis:	49,90 Euro
ISBN:	978-3-8362-4119-9

Microservices – live und in Farbe

Dr. Thomas Schuster und Dominik Galler, esentri AG



Microservices sind derzeit stark gefragt, doch welche Konzepte verbergen sich dahinter und wie lassen sie sich realisieren? Der Artikel erläutert zuerst die Grundprinzipien und stellt dann mit Vert.x ein Framework vor, mit dem Microservices schnell und einfach implementiert werden können. Ein praxisnahes Beispiel zeigt eine Anwendung mit Quellcode-Auszügen, die vollständige Anwendung steht auf GitHub zum freien Zugriff bereit.

Der Begriff „Microservice“ zieht seit geraumer Zeit eine Menge Aufmerksamkeit auf sich [1, 2]. Microservices können als Architekturstil für den Entwurf von verteilten Software-Systemen gesehen werden. Kurz gesagt, sie sind ein Ansatz zur Implementierung eines Systems durch eine größere Menge von kleinen Diensten (Services).

Dies ist vergleichbar mit den Prinzipien einer serviceorientierten Architektur (SOA), mit Microservices sind allerdings einige zusätzliche Forderungen verbunden, die mit SOA im Allgemeinen nicht assoziiert werden. So soll jeder Dienst unabhängig ausgeführt (eigener Prozessraum) werden, seine eigenen Daten (Datenbank) vorhalten und leichtgewichtige Kommunikationsmechanismen gegenüber anderen Diensten (oft über HTTP oder HTTPS) bieten. Der Dienst soll dabei ausschließlich Funktionalitäten bieten, die einer Geschäftsfunktion („Business Capability“) zugeordnet werden können.

Aus diesem fachlich stark eingeschränkten Fokus ergeben sich die Grundprinzipien serviceorientierter Architekturen. Insbesondere werden lose Kopplung, starke Kohäsion und die Trennung unterschiedlicher Belange („Separation of Concerns“) erreicht. Microservices fördern und erfordern die folgenden Grundsätze in besonderem Maße:

- Intelligente Dienste und einfache Kommunikation („Smart Endpoints & Dumb Pipes“)
- Evolutionäres Design
- Strenge Kapselung („Shared Nothing“)
- Dezentrale Governance
- Dezentrale Datenhaltung
- Automatisierung der Infrastruktur (Build-, Test- und Deployment-Prozesse)

Im Vergleich zur klassischen SOA setzen Microservices auf einfache Kommunikationsmechanismen nach dem Konzept „Smart Endpoints & Dumb Pipes“.

Anstelle eines Enterprise Service Bus (ESB) nutzen Microservices eher das Architekturmuster „Pipes & Filters“. Daraus folgt,

dass die intelligente Verarbeitung von Nachrichten innerhalb der Dienste („Smart Endpoint“) erfolgt, während die Kommunikation lediglich auf einfachen Mechanismen aufgebaut ist („Dumb Pipe“) und oftmals per REST oder über eine leichtgewichtige, asynchrone Kommunikationsinfrastruktur umgesetzt wird. Aus diesen Gründen können Microservices sehr leicht geändert oder gegen eine neue Implementierung ausgetauscht werden (evolutionäres Design).

Nachfolgend wird am Beispiel einer Web-Applikation demonstriert, wie eine verteilte Anwendung mithilfe von Microservices umgesetzt werden kann. Als Framework zur Realisierung kommt Vert.x 3 zum Einsatz. Zunächst werden die Aufgaben (Geschäftsfunktionalität) der Web-Applikation und eine Unterteilung in mögliche Microservices vorgestellt und abschließend gezeigt, wie diese mit Vert.x realisiert werden können. Neben den Erläuterungen im Artikel ist im Blog der Autoren [3] eine ausführliche Anleitung zur Umsetzung mit Vert.x beschrieben, der Quellcode der Beispielanwendung ist über GitHub öffentlich verfügbar [4].

Vert.x 3

Vert.x ist ein leichtgewichtiges, eventbasiertes Framework, das die Entwicklung von verteilten Anwendungen unterstützt. Es ist polyglott entwickelt, unterstützt somit

die Unabhängigkeit der Entwicklungsteams durch die Möglichkeit, verschiedene Programmiersprachen zur Umsetzung einzelner Anwendungsbestandteile einzusetzen. Vert.x selbst ist in Java implementiert, die Entwicklung des Frameworks wurde dabei durch Node.js inspiriert. Vert.x sollte aber nicht als Node.js in Java verstanden werden. Der Aufbau von Vert.x ermöglicht es, schnell und effektiv Microservices zu realisieren, auch wenn Vert.x kein explizites Framework für Microservices darstellt. Der Aufbau von Anwendungen auf Basis von Vert.x entspricht typischerweise folgendem Schaubild (siehe Abbildung 1).

Ein Host kann dabei mehrere JVMs bereitstellen, die als Ausführungskontext der einzelnen Vert.x-Instanzen dienen. Jede Vert.x-Instanz beherbergt eine eigene Menge sogenannter „Verticles“. Diese wiederum sind Bereitstellungselemente, die von Vert.x ausgeführt und kontrolliert werden. In den Verticles ist die eigentliche Anwendungslogik realisiert.

In der Regel wird ein Verticle auf einen Event reagieren oder ein neues Event emittieren. Die Kommunikation zwischen den Verticles regelt der in Vert.x integrierte, verteilte Eventbus. Dies dient wiederum dazu, die Verticles voneinander zu entkoppeln (Positions- und Mobilitätstransparenz). Die Kommunikation erfolgt durch die typischen

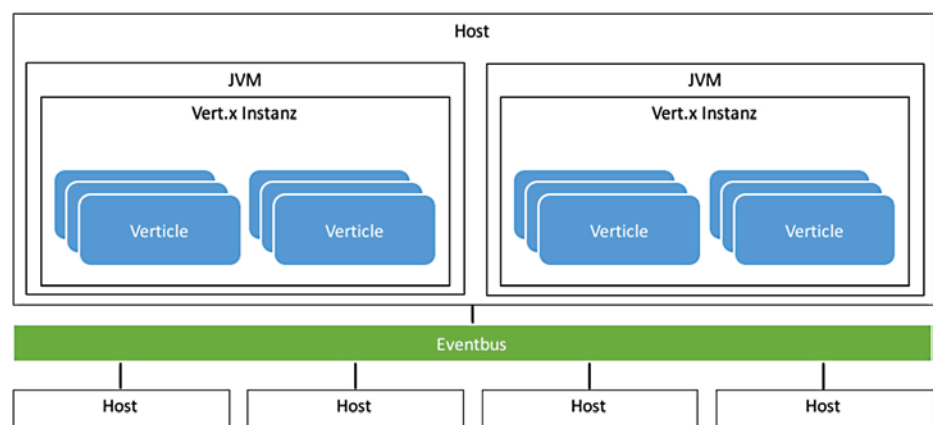


Abbildung 1: Anwendungsrealisierung mit Vert.x

Messaging Muster (Publish-Subscribe sowie Point-to-Point).

Neben der asynchronen kann auch eine synchrone Kommunikation etabliert werden. Mit Vert.x ist anstelle der eventbasierten auch eine REST-basierte Kommunikation möglich. Entsprechende HTTP-Anfragen können dann, nach Bedarf ebenfalls über den Eventbus, abgebildet und weiterverarbeitet werden. Zusätzlich sei angemerkt, dass über den Eventbus nur primitive Datentypen und Strings verschickt werden können. Dadurch wird verhindert, dass Programmiersprachen-spezifische Konstrukte genutzt werden, wodurch die Mehrsprachigkeit der verteilten Anwendung gesichert werden kann. Zur Übermittlung komplexer Objekte (Nachrichten) bietet sich das JSON-Format an.

Eine verteilte Anwendung

Als Beispielanwendung dient ein Branchenbuch. Benutzer tragen über eine Web-Schnittstelle zur Aktualität der Einträge bei. Die Anwendung soll den Benutzern entsprechende Telefonbuch-Einträge anzeigen. Angemeldete Benutzer sollen die Möglichkeit haben, die Einträge zu bearbeiten oder neue zu erstellen. Entlang dieser Geschäftsfunktionalitäten ist die Anwendung beispielhaft in folgende Komponenten zerlegt:

- UI (Nutzer-Interaktion auf Client-Seite)
- Server (zur Koordination der Benutzeranfragen)
- Verwaltung der Branchenbucheinträge
- Nutzerverwaltung (Registrierung und Verwaltung von Benutzern)

Die Komponenten sind als Microservices in Form von Verticles realisiert. Grundsätzlich kommunizieren in diesem Entwurf alle Komponenten über den Eventbus. Zusätzlich kontrolliert die Server-Komponente auch das Deployment der anderen Dienste (siehe Abbildung 2, Aufbau der Beispielanwendung), dies ließe sich selbstverständlich auch entkoppeln und extern steuern.

Benutzerschnittstelle

Die Benutzerschnittstelle kann prinzipiell in jeder beliebigen Technologie umgesetzt werden; die Autoren haben sich für JavaScript (AngularJS) entschieden, eine detaillierte Beschreibung findet sich in ihrem Blog-Artikel [5]. AngularJS wurde wegen der einfachen Anbindung des Vert.x-Eventbus gewählt. So lassen sich die JSON-Objekte,

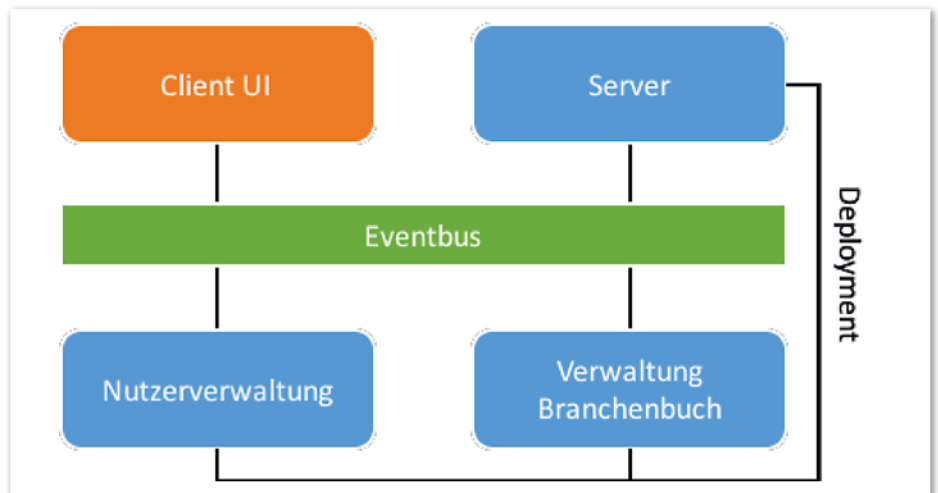


Abbildung 2: Aufbau der Beispielanwendung

```

1 public class MyVerticle extends AbstractVerticle {
2     private EventBus eb;
3
4     public void start() {
5         eb = vertx.eventBus();
6         eb.consumer("example.adress", this::exampleHandler);
7         eb.send("example.anotheradress","example message ");
8     }
9
10
11     private void exampleHandler(Message<String> message) {
12         //handle message content
13     }
14
15     public void stop() {
16         cleanup();
17     }
18
19 }
  
```

Listing 1

```

1 public void start() {
2     //...
3
4     vertx.deployVerticle("com.esentri.example.ExampleVerticle");
5
6     DeploymentOptions d0 = new DeploymentOptions();
7     d0.setWorker(true);
8
9
10    vertx.deployVerticle("com.esentri.example.ExampleWorkerVerticle", d0);
11
12    vertx.deployVerticle("com.esentri.example.AnotherVerticle",
13        rdy -> {
14            if (rdy.succeeded()) {
15                //some logic
16            }
17        });
18
19    //...
20 }
  
```

Listing 2

die über den Eventbus kommunizieren, per Data-Binding praktisch direkt zur Darstellung bringen. Zur Kommunikation benötigt die Applikation lediglich drei Bibliotheken:

- SockJS-basierte Eventbus-Bridge zur Anbindung an den Vert.x-Eventbus
- AngularJS-Wrapper für den Vert.x-Eventbus
- Bibliothek zur Kommunikation mit SockJS

```

1  {
2  "id": (String),
3  "name": (String),
4  "number": (String)
5  }

```

Listing 3

```

1  private void deleteEntryHandler(Message<String> message) {
2  String msg = message.body();
3  JsonObject json = new JsonObject(msg);
4  long id = json.getInteger("id").longValue();
5  String name = json.getString("name");
6  String number = json.getString("number");
7  Entry e = new Entry(id, name, number);
8  //delete entry
9  JsonObject reply = new JsonObject(msg);
10 reply.put("succeed", true);
11 eb.send("reply.adress", reply.toString());
12 }

```

Listing 4

Zur Entwicklung mit Vert.x 3 ist mindestens die Java-Version 1.8 erforderlich.

Verticle entwickeln

Das Muster zur Implementierung eines Verticle ist in [Listing 1](#) dargestellt. Zunächst ist die Klasse „AbstractVerticle“ zu erweitern (Zeile 1), um Zugriff auf den Eventbus und weitere Informationen der Vert.x-Instanz zu erhalten. Zur Bereitstellung wird die „start“-Methode aufgerufen (Zeile 5). Diese bildet den Ausgangspunkt für die Implementierungen der Anwendungslogik. Um Zugriff auf den Eventbus zu erhalten, registriert man einen Consumer (Zeile 7), der dann auf Events (Nachrichten) reagiert. Consumer können als anonyme Klassen per Lambda-Ausdruck oder als Klassen-Methode (Zeile 11) definiert werden.

Wird ein Verticle wieder gestoppt, also aus dem Ausführungskontext genommen, so wird seine „stop“-Methode aufgerufen (Zeile 15). Diese muss nicht zwingend überschrieben werden, es bietet sich jedoch an, Aufräumprozesse anzustoßen oder auszuführen (beispielhaft Zeile 16). Im Beispiel kann hier etwa der HTTP-Server gestoppt werden.

Verticles werden in einem Event-Loop ausgeführt. Sie warten, bis sie benötigt werden, und können dann Arbeit verrichten. Die Rechenzeit wird dabei zwischen allen Verticles aufgeteilt. Daher gilt hier der Grundsatz: Blockierender Code und langwierige Berechnungen sollten im Event-Loop vermieden werden. Um blockierende, lang andauernde Anwendungslogik realisieren zu können, kommen sogenannte „Worker“ zum

Einsatz. Noch besser ist es, auch diese zu vermeiden und auf asynchrone Lösungen zurückzugreifen. Für Datenbank-Zugriffe findet sich ein entsprechendes Beispiel auf der Vert.x-Homepage [6]. Implementierungsseitig besteht kein Unterschied zwischen normalen und Worker-Verticles, der Typ wird lediglich in den Bereitstellungsoptionen festgelegt.

Die Bereitstellung eines Verticle (siehe [Listing 2](#)) geschieht primär über die Methode „deployVerticle“ unter Angabe des voll qualifizierten Namens (Zeile 3). Durch die „DeploymentOptions“ (Zeile 4) kann das Verticle weiter konfiguriert werden. Die Optionen werden der „deployVerticle“-Methode übergeben und sorgen in diesem Beispiel (Zeile 8) dafür, dass das „ExampleWorkerVerticle“ als Worker bereitgestellt wird. Darüber hinaus ist es auch möglich, der „deployVerticle“-Methode einen Handler zu übergeben, der aufgerufen wird, sobald der Bereitstellungsprozess abgeschlossen ist (Zeile 10 ff.). Die „succeeded“-Methode des Handlers (Zeile 12) gibt an, ob die Bereitstellung erfolgreich war.

Nachdem in [Listing 1](#) bereits gezeigt wurde, wie ein Consumer auf dem Eventbus registriert werden kann, soll nun die Nachrichten-Verarbeitung anhand eines Dienstes zur Verwaltung der Branchenbuch-Einträge aufgezeigt werden. Es wird hier nur die Empfängerseite (Consumer) gezeigt. Bei der Kommunikation über den Eventbus sind, wie bereits erwähnt, nur Primitive und Strings zugelassen. Um dennoch komplexe Objekte und Nachrichten zu übertragen, bie-

tet sich JSON an. Die Nachricht an den Consumer des Branchenbuch-Dienstes, der für das Löschen eines Branchenbuch-Eintrags zuständig ist, liegt folglich in JSON-Format vor. Der Aufbau ist in [Listing 3](#) exemplarisch gezeigt.

Der Consumer ist in [Listing 4](#) beschrieben. Zunächst wird der Nachrichten-Typ als String spezifiziert (Zeile 1). In Zeile 2 wird der Nachrichten-Inhalt ausgelesen. Dieser kann dann dem Konstruktor für das in Vert.x integrierte „JsonObject“ übergeben werden. Über die „getter“-Methoden des JSON-Objekts lassen sich die einzelnen Werte auslesen (Zeile 3 ff.). Ein neues JSON-Objekt kann durch einen leeren Konstruktor erzeugt werden (Zeile 9). Attribute lassen sich wiederum mit der „put“-Methode hinzufügen (Zeile 10). Es wäre ebenso möglich, die „put“-Methoden auf dem „JsonObject“ zu verketten.

In Zeile 11 wird gezeigt, wie man eine komplexe Nachricht in JSON-Notation versenden kann. Die „toString“-Methode des „JsonObject“ behält die JSON-Syntax bei und ermöglicht so wiederum dessen Deserialisierung. Abschließend sei angemerkt, dass natürlich auch jeder andere JSON-Parser eingesetzt werden kann. Beispielsweise können mit FasterXMLs Jackson [7] DAO-Objekte durch entsprechende Annotation sehr leicht in JSON-Objekte und vice versa verwandelt werden.

Um das asynchrone Verhalten von Vert.x zu testen, wird abschließend der unter [Listing 4](#) definierte Consumer mit Vert.x-Unit-Tests getestet. Dieser löscht einen Branchenbuch-Eintrag und zeigt das Ergebnis der Operation durch einen booleschen Wert an. In Zeile 10 ff., ist zu sehen, dass das Verticle unabhängig von der Ausführung des Löschvorgangs mit „true“ antwortet. Dieses Verhalten ist so natürlich nicht wünschenswert, es wurde aber bewusst so gewählt, um dem exemplarischen Test ein wenig Leben einzuhauchen.

Die Testumgebung besteht aus einem speziellen „Runner“ für JUnit-Tests (siehe [Listing 5](#)). Neben der Bibliothek für die Vert.x-Unit-Tests ist darum auch noch die JUnit-Testbibliothek erforderlich. Nach der gewohnten Initialisierungsphase („@BeforeClass“) werden, wie bei JUnit üblich, die Tests unabhängig ausgeführt. Hier fällt auf, dass der Testmethode als Argument ein „TestContext“ übergeben wird (Zeile 13). Mit dessen Hilfe kann dann ein „Async“-Objekt erzeugt werden, das es ermöglicht, dass der Test so lange ausgeführt wird, bis die

„complete“-Methode des „Async“-Objekts den Test per Callback beendet (Zeile 19). Somit wird das asynchrone Testen ermöglicht. Bleibt der Callback zu lange aus, wird „complete“ per Timeout abgebrochen. Zusätzlich ist darauf zu achten, dass man nicht die normalen JUnit-Zusicherungen verwenden kann, sondern entsprechende Pendanten des „TestContext“-Objekts (Zeile 18) nutzt.

Der Test läuft insgesamt also folgendermaßen ab: Zunächst wird das Eventbus-Objekt bereitgestellt und ein Consumer registriert, der auf die Antwort des zu testenden Verticle reagiert (Zeile 17). Sobald dann eine Antwortnachricht emittiert ist, wird überprüft, ob die erhaltene Antwort dem Erwarteten entspricht (Zeile 19). Dazu wurde vorher die erwartete Antwort als JSON-Objekt erstellt (Zeile 16).

Schließlich wird der Test durch Aufruf der „complete“-Methode abgeschlossen. Im „TestContext“ ist nun hinterlegt, ob der Test erfolgreich war, und das Ergebnis wird schließlich, wie für JUnit üblich, protokolliert. Zum Abschluss sollten die testenden Verticles wieder aus dem Ausführungskontext genommen werden, um eine saubere Umgebung zu hinterlassen (Zeile 36 ff.). Dies ist insbesondere dann wichtig, wenn mehrere Verticles getestet und Seiteneffekte ausgeschlossen werden müssen.

Fazit

Verteilte Anwendungen können durch Microservices unter Unterstützung von Vert.x realisiert werden. Dank Vert.x kann man diese leicht konstruieren, ohne dass sich die Entwickler auf eine einzige Programmiersprache festlegen müssen. Mit Unit-Tests (Vert.x Unit) können auch asynchrone Implementierungen gut getestet werden. Vert.x stellt damit eine ernstzunehmende Alternative zu anderen, bereits etablierten Frameworks dar.

Quellen

- [1] J. Lewis und M. Fowler, *Microservices*: <http://martinfowler.com/articles/microservices.html>
- [2] C. Richardson, *Microservices, Decomposing Applications for Deployability and Scalability*, InfoQ: <http://www.infoq.com/articles/microservices-intro>
- [3] **Blog-Artikel „Vert.x – Live und in Farbe“**: <http://www.esentri.com/blog/2016/01/22/vertx-live-und-in-farbe>
- [4] <https://github.com/dominikgaller/esentrimicroserviceexample.git>
- [5] **Blog-Artikel „AngularJS in Zusammenspiel mit Vert.x“**: <http://www.esentri.com/blog/2016/01/22/angularjs-in-zusammenspiel-mit-vertx>
- [6] http://vertx.io/docs/#data_access
- [7] <https://github.com/FasterXML/jackson>

```

1  @RunWith(VertxUnitRunner.class)
2  public class DatabaseTest {
3
4      private static Vertx vertx;
5      private static final String VERTICLE =
6          "com.esentri.microserviceExample.databaseService.DatabaseVerticle";
7
8      @BeforeClass
9      public static void start(TestContext context) {
10         vertx = Vertx.vertx();
11         vertx.deployVerticle(VERTICLE);
12     }
13
14     @Test
15     public void deleteHandlerTest(TestContext context) {
16         Async async = context.async();
17         EventBus eb = vertx.eventBus();
18         JsonObject expectedReply = new
19             JsonObject().put("succeeded", true);
20         JsonObject generatedObject = generateTestObject();
21         eb.consumer("reply.adress",
22             onreply -> {
23                 context.assertEquals(expectedReply.toString(), onreply.body());
24                 async.complete();
25             });
26         eb.send("adress.to.handler",
27             generatedObject.toString());
28     }
29
30     private JsonObject generateTestObject() {
31         JsonObject json = new JsonObject();
32         /*
33          * Generate testobject here, and
34          * assure its in the database. After that
35          * return its json representation.
36          */
37         return json;
38     }
39
40     @AfterClass
41     public static void stop(TestContext context) {
42         vertx.undeploy(VERTICLE);
43     }
44 }

```

Listing 5

Dr. Thomas Schuster
thomas.schuster@esentri.com



Thomas Schuster beschäftigt sich seit Langem mit dem Entwurf und der Optimierung komplexer, verteilter Software-Architekturen und Geschäftsprozesse. Ein besonderes Anliegen sind ihm dabei Qualitätsmerkmale (Flexibilität und Skalierbarkeit) sowie die System-Ausrichtung entlang der Geschäftsfunktionalität. Als Principal Consultant bei der esentri AG koordiniert er neben Beratungsprojekten auch Innovations- und Forschungsvorhaben.

Dominik Galler
dominik.galler@esentri.com



Dominik Galler studiert Informatik am Karlsruher Institut für Technologie (KIT). Als Werkstudent unterstützt er die esentri AG bei internen Software-Entwicklungsprojekten und arbeitet derzeit außerdem aktiv im Forschungsprojekt BiE. In diesem Rahmen fertigt er auch seine Abschlussarbeit an.



Open-Source-Performance-Monitoring mit stagemonitor

Felix Barnsteiner und Fabian Trampusch, iSYS Software GmbH

In jedem Software-Projekt muss man sich früher oder später Gedanken über die Performance machen, um marktfähig zu bleiben und den Erwartungen der Nutzer zu entsprechen. Hierzu ist ein gutes Monitoring-Werkzeug unerlässlich. Kommerzielle Lösungen sind teuer und bestehende Open-Source-Projekte sind oftmals nicht speziell für Web-Anwendungen ausgelegt oder unterstützen Anwendungen im Clusterbetrieb unzureichend.

Genau hier setzt die Open-Source-Performance-Monitoring-Lösung „stagemonitor“ an. Geboten werden alle Werkzeuge, die während der Entwicklung, der Qualitätssicherung und in der Produktion nötig sind, um die Performance einer Java-Web-Anwendung überwachen zu können. Die Lösung ist speziell darauf ausgelegt, aktuelle und historische Metriken von Anwendungen im Cluster-Betrieb zu überwachen. Die Plug-in-Architektur ermöglicht es, offizielle Erweiterungen oder Plug-ins Dritter einzubinden. Auch eigene Erweiterungen können entwickelt werden.

stagemonitor während der Entwicklung

In der Praxis realisiert man oftmals erst dann, wie wenig performant eine Anwendung

ist, wenn sich ein verärgertes Kunden über eine langsame Seite beschwert. Gerade bei eCommerce-Webseiten kann dies zu erheblichen wirtschaftlichen Schäden führen. Nicht nur, dass ein verärgertes Kunden wahrscheinlich nicht nochmals einen langsamen Shop besucht, viele machen ihrem Unmut auch bei Freunden oder in sozialen Netzwerken Luft. Deshalb ist es entscheidend, Performance-Probleme so früh wie möglich im Entwicklungszyklus zu erkennen und zu beheben.

stagemonitor bietet mit dem In-Browser-Widget ein Werkzeug an, mit dem die Performance während der Entwicklung analysiert und zielgerichtet optimiert werden kann. Zudem wird der Entwickler auf Probleme hingewiesen, die sonst erst in der Produktion sichtbar werden. Das In-Browser-

Widget ist ein Monitoring-Dashboard, das automatisch in das HTML von Webanwendungen eingefügt wird. Die wichtigsten Funktionen des In-Browser-Widgets sind:

- **Call Tree**

Nach dem Öffnen wird ein Call Tree aller Methoden, SQL- und Elasticsearch-Queries angezeigt, die während der Bearbeitung des aktuellen Requests ausgeführt wurden. Hierdurch lassen sich die langsamen Methoden einer Anwendung identifizieren. stagemonitor unterstützt den Entwickler beim Beheben des Problems, indem es den zeitlichen Anteil eines Methodenaufrufs an der Anfrage darstellt. Dadurch kann der langsame Code optimiert werden, ohne dass man als Entwickler ziellos ins Blaue

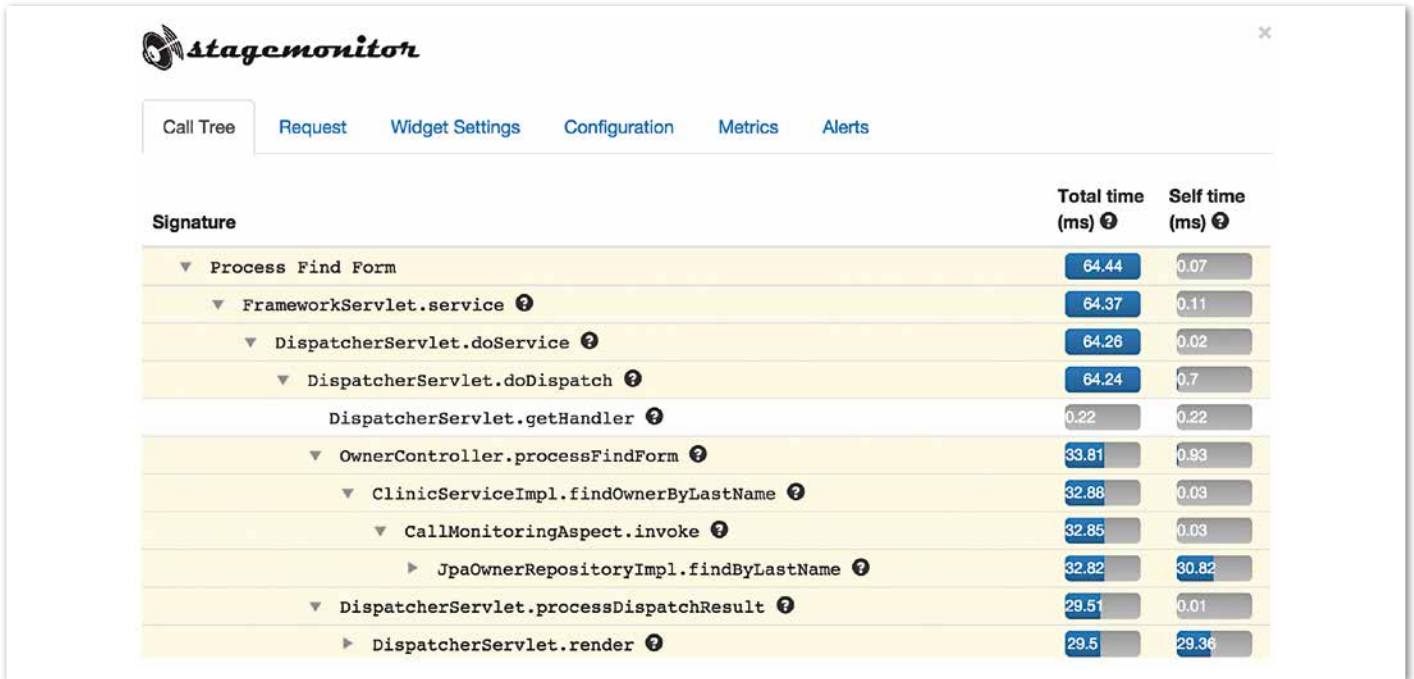


Abbildung 1: In Browser Widget - Call Tree

optimiert. Ajax Requests können ebenfalls analysiert werden (siehe Abbildung 1).

▪ **HTTP-Request-Informationen**

Im Request-Tab finden sich allgemeine Informationen zur abgesetzten Anfrage. Diese Informationen umfassen beispielsweise die Größe der Response, die Request-Header sowie Details zum User-Agent (von welchem Gerät und von welchem Betriebssystem die Anfrage abgesetzt wurde). Zudem wird die Ladezeit der Seite in die Kategorien Netzwerk, Server, DOM Processing und Page Rendering aufgeteilt.

▪ **Konfigurierbare Notifications**

Es können Grenzwerte für die Seitenladezeit und die Anzahl der SQL-Queries während einer Anfrage definiert werden. Das Widget zeigt eine Notification an, falls diese Grenzwerte nicht eingehalten wurden. Es können so besonders datenbankintensive Requests identifiziert werden.

▪ **Real-Time-Performance-Graphen**

Im Metrics-Tab befinden sich Graphen, die in Echtzeit beispielsweise Metriken der JVM, des Betriebssystems und der Antwortzeiten darstellen. Dieser Tab ist durch Plug-ins erweiterbar.

▪ **Alerts**

Im Alerts-Tab können Grenzwerte für alle gesammelten Metriken wie beispiels-

weise die Antwortzeit festgelegt werden. Exemplarisch können hiermit E-Mail-Benachrichtigungen versendet werden, falls die durchschnittliche Antwortzeit der Suchfunktion im Produktivsystem einen bestimmten Wert überschreitet. Dieser Tab ist Bestandteil des Alerting-Plugins und zeigt, wie die Tabs durch Plug-ins erweiterbar sind.

stagemonitor in der Produktion

Die Performance während der Entwicklungsphase überwachen und verbessern zu können, ist schön und gut, aber viel wichtiger ist das Monitoring in der Produktion. Wenn eine Anwendung mit einem kleinen Testdaten-Bestand und einem aktiven Nutzer schnell ist, bedeutet das nicht, dass die Anwendung auch mit mehreren Nutzern und einer größeren Datenbasis skaliert. stagemonitor hat sich für den Einsatz in der Produktion bewährt, ohne dass hierzu ein weiteres Tool notwendig ist. Damit vereinfacht sich auch die Kommunikation zwischen Entwicklern und Administratoren, da beide dieselbe Daten-Grundlage und Auswertungsmöglichkeiten haben. Bei der Entwicklung von stagemonitor lag ein Augenmerk darauf, dass die Anfragebearbeitung in der Produktion nicht spürbar verlangsamt werden darf.

Da das In-Browser-Widget in der Produktion nicht aktiv ist und nur die Daten eines Servers anzeigen kann, gibt es für den pro-

duktiven Betrieb spezielle Dashboards, in denen die Informationen für alle Server zusammengefasst dargestellt sind. Die Dashboards basieren auf den Open-Source-Projekten „Kibana“ und „Grafana“; sie lassen sich individuell anpassen. Dabei können die Daten wahlweise aggregiert (Übersicht über alle Server) oder einzeln dargestellt werden. Es ist auch möglich, historische Daten abzurufen. Deshalb ist es notwendig, eine Datenbank aufzusetzen, in der die Daten persistiert werden.

Analyse von Anfragen

Ein momentan weitverbreitetes Vorgehen, um mehr Einsicht in die Anfragen zu bekommen, die eine Applikation abarbeitet, ist die Analyse von Log-Dateien mit dem sogenannten „ELK-Stack“. Hierzu werden zunächst Logstash-Access-Logs, beispielsweise von Apache, ausgelesen und mithilfe von regulären Ausdrücken Informationen extrahiert, beispielsweise die angefragte URL oder die Verarbeitungszeit. Anschließend werden die Daten in der Key-Value-Datenbank Redis zwischengespeichert. Der nächste Schritt ist die Ablage der Daten in Elasticsearch, um eine einfache Durchsuchbarkeit und Analyse zu ermöglichen. Im letzten Schritt muss noch ein Kibana-Dashboard erstellt werden, mit dessen Hilfe die Informationen visualisiert werden können.

Setzt man stagemonitor ein, so entfallen die meisten Schritte. stagemonitor befindet sich innerhalb der Applikation und hat so

direkten Zugriff auf Informationen wie die URL, die Verarbeitungszeit, aufgetretene Exceptions und viele weitere. Diese Informationen müssen nicht erst mühsam aus Logs extrahiert werden. Zudem sind nicht gleich vier neue Systeme zu installieren und zu warten, sondern nur Elasticsearch und Kibana. Um auch diesen Schritt zu vereinfachen, wird eine Imagedocker-compose.yml Datei angeboten, in dem alle benötigten Systeme bereits aufgesetzt und vorkonfiguriert sind.

stagemonitor sendet die gesammelten Informationen auf Wunsch selbstständig zu einem Elasticsearch-Server und bietet sogar ein vorkonfiguriertes Kibana-Dashboard an, das jedoch angepasst werden kann. Durch die dynamischen Analysemöglichkeiten von Kibana können vielseitige Fragestellungen beantwortet werden:

- Welche Fehler treten besonders häufig auf und durch welche Requests werden sie verursacht?
- Wie können die Fehler reproduziert werden?
- Warum war der Request eines Kunden zu einer bestimmten Zeit langsam?
- Welchen Browser und welchen Gerätetyp setzen meine Kunden am häufigsten ein?

Diese Informationen sind für den Betrieb und für das Marketing Gold wert, um Fehler oder das Nutzerverhalten nachzuvollziehen.

Analyse von Metriken

Ein weiteres Kerngebiet von stagemonitor ist die Analyse von Metriken verteilter Anwendungen. Zwar bietet das In-Browser-Widget bereits eine Möglichkeit, die Metriken eines Servers in Graphen darzustellen, jedoch reicht dies nicht immer aus. Will man die Metriken der Vergangenheit ansehen, interessiert man sich für die Metriken mehrerer Server oder möchte man eigene Dashboards erstellen, so benötigt man die von stagemonitor bereitgestellten Grafana-Dashboards. Grafana ist ein Visualisierungstool für Zeitreihen-Daten und unterstützt mehrere Zeitreihen-Datenbanken:

- Das Request-Dashboard gibt Auskunft über das Antwortzeitverhalten und den Durchsatz der Anwendung. Zudem werden die langsamsten und fehleranfälligsten Business Transactions (wie „Suche“ oder „Artikeldetail ansehen“) hervorgehoben.
- Das JVM-Dashboard enthält Informationen über die Heap-Auslastung, das Garbage-Collection-Verhalten und die CPU-Auslastung der JVM.
- Darüber hinaus gibt es Dashboards für EhCache-, Logging-, Application-Server-, Betriebssystem-Metriken und Datenbank-Abfragen.

Mit diesen Statistiken fällt es leicht, die Frage zu beantworten, ob einzelne Caches das Antwortverhalten der Anwendung tatsächlich

verbessern oder ob die Cache-Hit-Rate zu gering ist, um einen tatsächlichen Mehrwert zu bieten. Jedes Dashboard bietet die Wahl, welche Applikation, welcher Host und welche Instanz betrachtet werden soll. Somit lassen sich wahlweise entweder einzelne Server oder der gesamte Cluster im Überblick betrachten.

stagemonitor unterstützt die Zeitreihen-Datenbanken Elasticsearch, InfluxDB und Graphite. Empfohlen ist der Einsatz von Elasticsearch, da hierbei nur eine Datenbank installiert werden muss, um sowohl die Request-Traces als auch die Metriken abzuspeichern. Dies macht die Installation und den Betrieb von stagemonitor besonders einfach. Zudem kann Elasticsearch gut mit großen Datenmengen umgehen, da es sich einfach skalieren lässt.

Der Unterschied zu vielen anderen Lösungen besteht darin, dass nicht bei jedem eingehenden Request Daten übermittelt werden müssen. Die statistischen Daten über die Antwortzeit, beispielsweise Standardabweichung, Mittelwert und verschiedene Perzentile, werden im Server berechnet und periodisch (standardmäßig jede Minute) an die Zeitreihen-Datenbank übermittelt. Damit ist es egal, wie viele Anfragen der Server bearbeiten muss. Die anfallende Datenmenge bleibt immer konstant und wird nur vom Übertragungsintervall bestimmt. Auch die Datenstruktur, die die Statistiken generiert, ist unabhängig von der Anzahl eingehender Anfragen, da immer nur eine repräsentative Menge von rund tausend

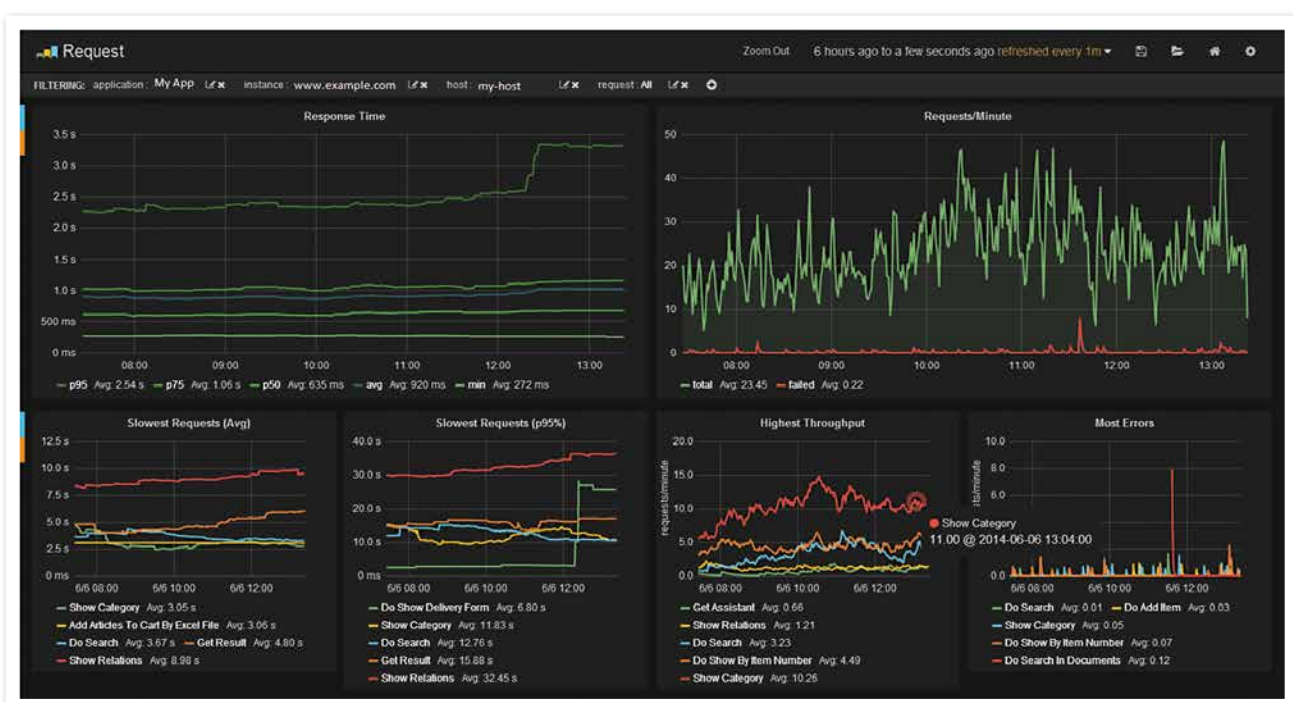


Abbildung 2: Request Dashboard

Antwortzeiten (pro Business Transaction) im Speicher gehalten wird (siehe Abbildung 2).

stagemonitor ausprobieren

Nachfolgend sind drei Möglichkeiten beschrieben, um stagemonitor schnell und einfach auszuprobieren. Für eine Live-Demo steht unter „www.stagemonitor.org“ ein Link zur „Spring PetClinic“ mit stagemonitor bereit. Das In-Browser-Widget wird durch das Icon in der unteren rechten Ecke aktiviert.

Um stagemonitor in eigener Umgebung auszuprobieren, steht als zweite Möglichkeit auf GitHub (siehe „https://github.com/stagemonitor/spring-petclinic#running-petclinic-locally-with-stagemonitor-enabled“) ein Beispielprojekt auf Basis der Beispiel-Applikation „Spring PetClinic“ bereit. Um die Anwendung zu starten, sind lediglich drei Befehle auf der Konsole erforderlich (siehe Listing 1).

Das Projekt wird durch die Befehle per „git“ aus GitHub geklont und mithilfe von Maven gebaut. Anschließend wird ein Embedded-Tomcat-Server mit der Anwendung gestartet. Unter „http://localhost:9966/petclinic“ steht nun die Beispiel-Applikation „Spring PetClinic“ mit aktiviertem stagemonitor-In-Browser-Widget zur Verfügung.

Als dritte Möglichkeit wird stagemonitor in das Spring-Beispielprojekt PetClinic (siehe „https://github.com/stagemonitor/spring-petclinic“) integriert. Hierfür muss die „pom.xml“ um den Dependency-Eintrag für stagemonitor ergänzt werden (siehe Listing 2).

Projekte, die Gradle oder Ant verwenden, können ähnlich gebaut werden. Hierfür sei auf die Dokumentation von stagemonitor („https://github.com/stagemonitor/stagemonitor/wiki/Step-1:-Prerequisites“) verwiesen. Zuletzt wird unter „src/main/resources“ die Datei „stagemonitor.properties“ angelegt. In dieser Datei können diverse Konfigurations-Parameter (siehe https://github.com/stagemonitor/stagemonitor/wiki/Configuration-Options“) für stagemonitor hinterlegt sein, die beim Start der Anwendung aktiviert sind. Die meisten lassen sich jedoch auch zur Laufzeit über das In-Browser-Widget anpassen. Als minimale Konfiguration ist „stagemonitor.instrument.include=org.springframework.samples.petclinic“ empfohlen. Der Konfigurationsschalter erlaubt die Angabe von kommagetrennten Package- und Klassennamen, die instrumentiert werden sollen. Damit ist die Integration abgeschlossen.

Das Maven-Target „tomcat7:run“ baut und startet nun die Anwendung. Sie ist unter „http://localhost:9966/petclinic“ erreichbar. In der rechten unteren Ecke befindet sich das

```
git clone https://github.com/stagemonitor/spring-petclinic.git
cd spring-petclinic
mvn tomcat7:run
```

Listing 1

```
<dependencies>
  <dependency>
    <groupId>org.stagemonitor</groupId>
    <artifactId>stagemonitor-web</artifactId>
    <version>0.22.0</version>
  </dependency>
  [...]
</dependencies>
```

Listing 2

stagemonitor-Icon für das In-Browser-Widget zum Anklicken.

Fazit

stagemonitor stellt eine praxiserprobte Lösung für das Monitoring der Anfragen und Einblicke in das interne Performance-Verhalten der Anwendung dar. Es unterstützt dabei den gesamten Lebenszyklus der Anwendung, von der Entwicklung bis zum Betrieb. Mit dem Call-Tree des In-Browser-Widgets steht ein mächtiges und einfaches Werkzeug zur Verfügung, um gezielte Optimierungen des serverseitigen Codes zu ermöglichen. Das Aufzeichnen der Request-Statistiken bietet detaillierte Informationen, um auch bei umfangreichen Web-Anwendungen langsame Funktionalitäten zu identifizieren und den langfristigen Erfolg der Optimierung messbar zu machen. Auch Fehler lassen sich mit den gesammelten Daten einfacher reproduzieren.

Auf der Roadmap steht ein Ausbau der Analyse-Funktionen, um eine leichtgewichtige Alternative zu Google Analytics oder Piwik bieten zu können. In Zukunft wird stagemonitor unter anderem Unterstützung für automatisiertes Baselineing bieten. Dies soll es ermöglichen, konkrete Kennzahlen zur Performance der Anwendung in Bezug auf unterschiedliche Releases zu liefern. Auch automatisierte Anomalie-Detektion oder Rückschlüsse auf Fehlerursachen sollen so möglich werden. Getreu dem Zitat des Informatik-Pioniers Donald Knuth „We should forget about small efficiencies, say about 97 percent of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3 percent“ hilft stagemonitor dabei, sich auf diese drei Prozent zu konzentrieren. stagemonitor steht kostenlos unter der Apache License 2.0 unter „http://www.stagemonitor.org“ zur Verfügung.

Felix Barnsteiner

f.barnsteiner@isys-software.de



Felix Barnsteiner ist Software Engineer bei der iSYS Software GmbH. Er ist der Entwickler des Open-Source-Performance-Monitoring-Projekts stagemonitor. Zudem beschäftigt er sich mit der Entwicklung von eCommerce-Systemen.

Fabian Trampusch

f.trampusch@isys-software.de



Fabian Trampusch ist Software Engineer bei der iSYS Software GmbH. Er entwickelt moderne Web-Anwendungen mit Haupt-Augenmerk auf Qualität und Benutzerkomfort. Neben der Arbeit studiert er den Master „Software Engineering“ an der Hochschule München.

Profiles for eclipse – Eclipse im Enterprise-Umfeld nutzen und verwalten

Frederic Ebelshäuser und Sophie Hollmann, Yatta Solutions

Eclipse wird geschätzt für Flexibilität, Funktionsvielfalt und Erweiterbarkeit. Außerdem ist die Entwicklungsumgebung quelloffen. Ideale Voraussetzungen für Software-Entwickler, sollte man meinen. Doch schon das Einrichten stellt besonders Entwickler im Enterprise-Umfeld oftmals vor Herausforderungen. Flexibilität führt zu Komplexität, und so verursachen Setup und Wartung der Eclipse-Entwicklungsumgebung gerade in großen Teams häufig erhebliche Kosten. Aber Eclipse wäre nicht Eclipse, gäbe es da nicht die Community und das Ökosystem: Verschiedene Lösungen für Einrichtung, Wartung und Pflege von Eclipse-Projektsetups stehen bereit. Aber welche passt zum eigenen Entwicklungsprojekt?

Wer kennt nicht dieses Bild: Ein Programmierer sitzt allein in einem schlecht beleuchteten Kellerraum, umgeben von Monitoren, alten Pizzakartons und benutzten Kaffeebechern ... So oft dieser Stereotyp auch über die Kino-Leinwände flimmern mag, er könnte kaum weiter von der Realität entfernt sein. Die Wahrheit ist, Software-Entwickler des 21. Jahrhunderts sind Teamplayer. Agile Software-Entwicklung ist nichts für Einzelgänger: Ein Team kooperiert womöglich mit weiteren Teams, deshalb stehen Kommunikation und Kooperation weit oben auf der Anforderungsliste an den Entwickler von heute. Zudem wird inzwischen auch in sehr konservativen Unternehmen verstanden, welchen Wertschöpfungsbeitrag Software – und damit auch Software-Entwickler – zum Unternehmenserfolg leisten. Für Entwicklungsabteilungen ist eine effiziente Infrastruktur für Technologie-Management und -Distribution umso entscheidender.

Die Eclipse IDE im Enterprise-Umfeld

Eclipse hat weltweit zwischen sieben und acht Millionen Nutzerinnen und Nutzer. Viele von ihnen sind im Enterprise-Umfeld tätig – als Freelancer, Auftragsentwickler oder fest angestellte Mitarbeiterinnen und Mitarbeiter. Die meisten verstehen unter „Eclipse“ die bekannte Entwicklungsumge-

bung für Java-Applikationen (RCP oder Web) – auch wenn Eclipse im PHP- oder C++-Umfeld ebenfalls weite Verbreitung findet.

Dreizehn standardisierte Packages in gängigen Konfigurationen stehen zum Download zur Verfügung. Zehn davon sind für die

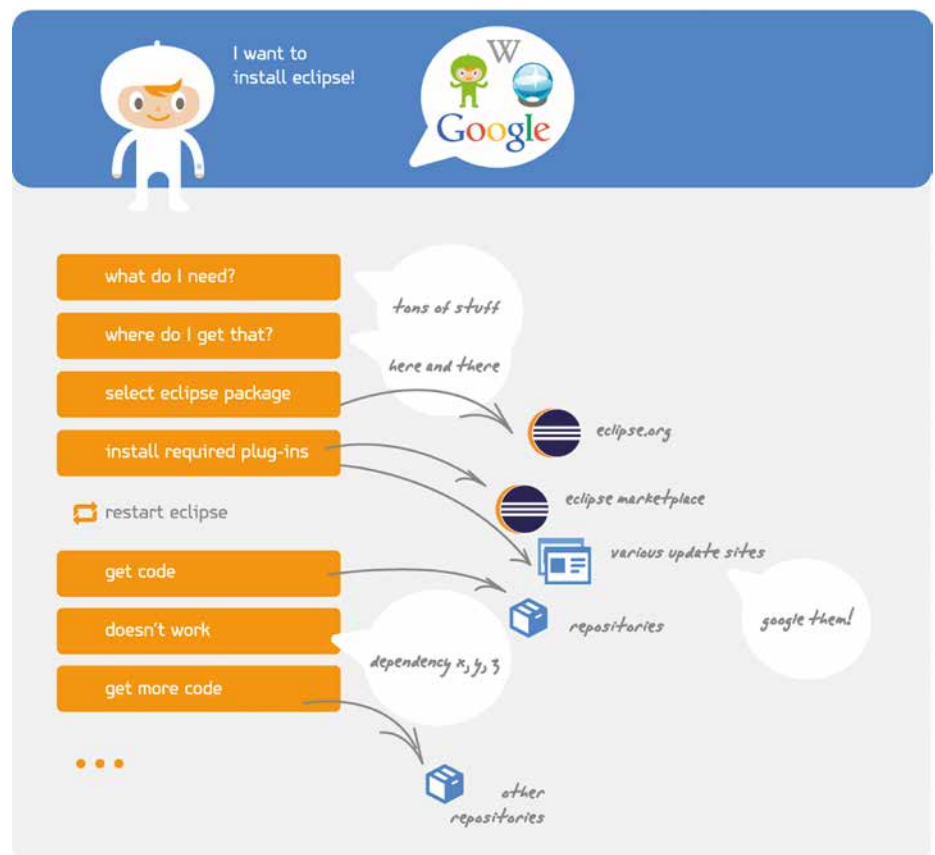


Abbildung 1: Viele Faktoren spielen bei der Erstellung eines Projekt-Setups mit Eclipse eine Rolle

Java-Softwareentwicklung bestimmt. Diese Packages verzeichnen zusammen jeden Monat etwa 2,4 Millionen Downloads. Dennoch werden die Packages nur selten ohne weitere Anpassungen eingesetzt (siehe Abbildung 1).

Das meistverbreitete Package (oder Profil) zum Beispiel ist die „Eclipse IDE for Java EE Developers“. Es besteht aus elf Features (Software-Komponenten) oder 63 Plug-ins. Sie sollen möglichst viele Use-Cases für die Anwendungsentwicklung mit Java EE abdecken. In der Regel wird jedoch nur ein Bruchteil der enthaltenen Plug-ins auch tatsächlich benötigt, dafür fehlen andere, die für ein spezielles Projekt-Setup essenziell sind. Weitere 1.500 Plug-ins stehen im Eclipse Marketplace für die Anpassung oder Erweiterung zur Auswahl; mehr als 400.000 Plug-in-Installationen monatlich verzeichnet schon allein der Eclipse Marketplace Client. Allen Bemühungen um Standardisierung zum Trotz: Software-Projekte im Enterprise-Umfeld sind heterogen. Jedes Projekt hat seine spezifischen Anforderungen.

Im Grunde kommt die Eclipse-IDE diesen Ansprüchen durch ihre Funktionsvielfalt, Offenheit und Flexibilität sehr entgegen. Eclipse ist skalierbar und dazu auch noch Open Source. Das macht die IDE zu einem äußerst mächtigen Werkzeug zur Software-Entwicklung. Um ein eigenes Projekt-Setup kommt allerdings niemand herum, und dieses erfolgt zumeist händisch – fast schon archaisch.

Nach der Auswahl eines Package müssen Entwickler Plug-ins einzeln deinstallieren oder zusätzlich installieren – und dazu Eclipse häufig neu starten. Sie müssen Quellcode- und Task-Repositories einrichten, Quellcode auschecken und den entsprechenden Projekten im Workspace zuordnen. Erst danach können sie produktiv arbeiten (siehe Abbildung 2).

Das Einrichten der Entwicklungsumgebung mit allen Abhängigkeiten und Konfigurationen nimmt nicht selten mehrere Stunden oder gar Tage in Anspruch – und das betrifft meist nicht nur den einen, neuen Entwickler. Dabei arbeiten größere Teams und Entwicklungsabteilungen häufig mit eigens standardisierten Installationen und Einstellungen von Eclipse. Diese können je nach Rolle im Team, Projektphase oder sogar bei bestimmten Problemstellungen variieren, doch schon bei der Verteilung der gemeinsamen Standardkonfigurationen hapert es. Projekt-Setups effizient zu verteilen und zu

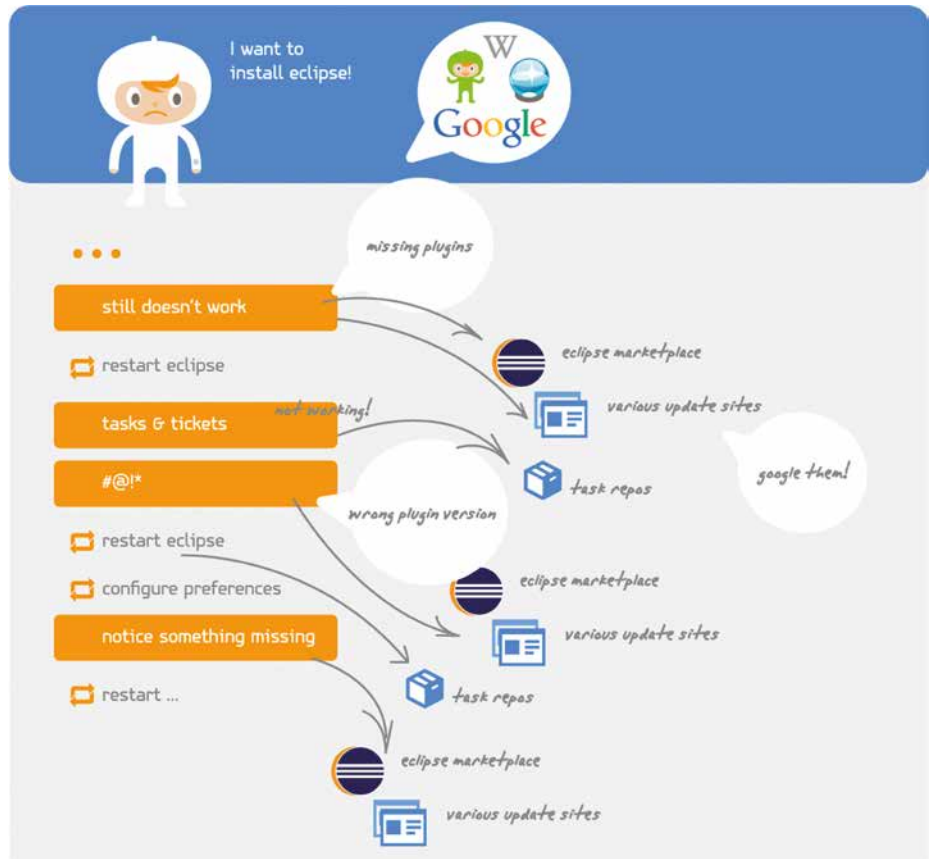


Abbildung 2: Entwickler erstellen Setups häufig zeitaufwändig manuell, bevor sie produktiv werden können

verwalten, ist immer noch eine echte Herausforderung. Die Kombination von oftmals veralteten ZIP-Files, Wikis und sonstigen Dokumenten mit Anleitungen zu Installation und Konfiguration kann nicht der Weisheit letzter Schluss sein. Und den einen, allwissenden Entwickler im Team bei jedem Setup mit Nachfragen zu überhäufen, ist ebenso wenig eine nachhaltige Lösung.

Eclipse-Infrastruktur für Teams

Für das Einrichten und Verwalten der Eclipse-Entwicklungsumgebung in Teams gibt es eine Reihe von Lösungsansätzen. Verschiedene kommerzielle Angebote von großen IT-Dienstleistern und Beratern sind im Markt verfügbar. Nachfolgend gehen wir auf eine Auswahl aus der Eclipse-Community (Mitgliedsunternehmen und Committer) selbst ein (siehe Abbildung 3).

Die minimale Lösung ist es, eines der Eclipse-Packages herunterzuladen, die benötigten Plug-ins zu installieren sowie anschließend das Installationsverzeichnis als „zip“-File zu packen und im Team zu verteilen. Dies ist ein einfacher und schneller Weg, ein Setup einem überschaubaren Personenkreis, zum Beispiel seinem SCRUM-Team,

zur Verfügung zu stellen – eventuell kombiniert mit einer Installations-Anleitung. Dabei wird jedoch der Lebenszyklus des eingesetzten Setups nicht berücksichtigt. Bei der Aktualisierung eines einzelnen Plug-ins muss der gesamte Prozess wiederholt werden. Zudem erfordert Eclipse auf unterschiedlichen Betriebssystemen (Windows, Mac OS, Linux), aber auch bei unterschiedlichen lokalen Installationspfaden oder zum Aufsetzen des Projekt-Workspace, in aller Regel noch manuelle Nacharbeit. Ein mühsamer Prozess, der für größere Teams ungeeignet erscheint.

Kommerzielle Lösungen

Das Software Delivery Center (SDC) von Genuitec ist ein Provisionssystem für große Unternehmen. Als eigene Plattform inklusive einer privaten Cloud können hier Setups erstellt, verwaltet und verteilt werden. Anlegen und Bearbeiten lassen sich Setups mit einem Generator im laufenden Eclipse sowie über eine Admin-Oberfläche durch berechtigte Projektleiter.

Interessant ist beim SDC vor allem die Möglichkeit, Policy-Enforcements für die Installation neuer Software und anderer Be-

nutzerrechte zu definieren. Dies stellt sicher, dass Entwickler zum Beispiel nur bestimmte Plug-ins aus bestimmten Quellen beziehen oder ihre Installation gar nicht selbst verändern können. Gleiches gilt für Preferences und Sourcecode-Repositories.

Über eine Admin-Oberfläche können Profile angelegt und bearbeitet werden. Insbesondere lassen sich dort Policies und Zugriffsrechte definieren. Updates können ebenfalls vom SDC aus gezielt an Mitarbeiter geschickt werden. Einschränkungen unterliegt SDC vor allem im Hinblick auf die Beschränkung als kommerzielles Angebot mit hohen Einstiegshürden.

Yoxos ist zugleich eine Eclipse-Distribution, ein Management-Werkzeug für Eclipse-Setups und ein Provisioning-Service. In der kostenlosen Version können individualisierte Setups definiert werden, die volle Funktionalität erhält man allerdings erst mit einer kostenpflichtigen Lizenz. Anlegen und Bearbeiten lassen sich Setups mit einem Generator im laufenden Eclipse sowie über eine Admin-Oberfläche, die der des SDC nicht unähnlich ist.

Bei der Konfiguration stehen dabei ausschließlich Plug-ins aus dem Yoxos-Repository zur Verfügung. So wird zwar die Kompatibilität der ausgewählten Plug-ins gewährleistet, dafür werden indes Flexibilität und Funktionsumfang von Eclipse eingeschränkt. Plug-ins aus anderen Quellen müssen von Entwicklern manuell nachinstalliert werden; eine Distribution im Setup wird – soweit ersichtlich – nicht unterstützt.

Die unterstützten Setups werden in zwei Phasen definiert: Zuerst wird die Installation

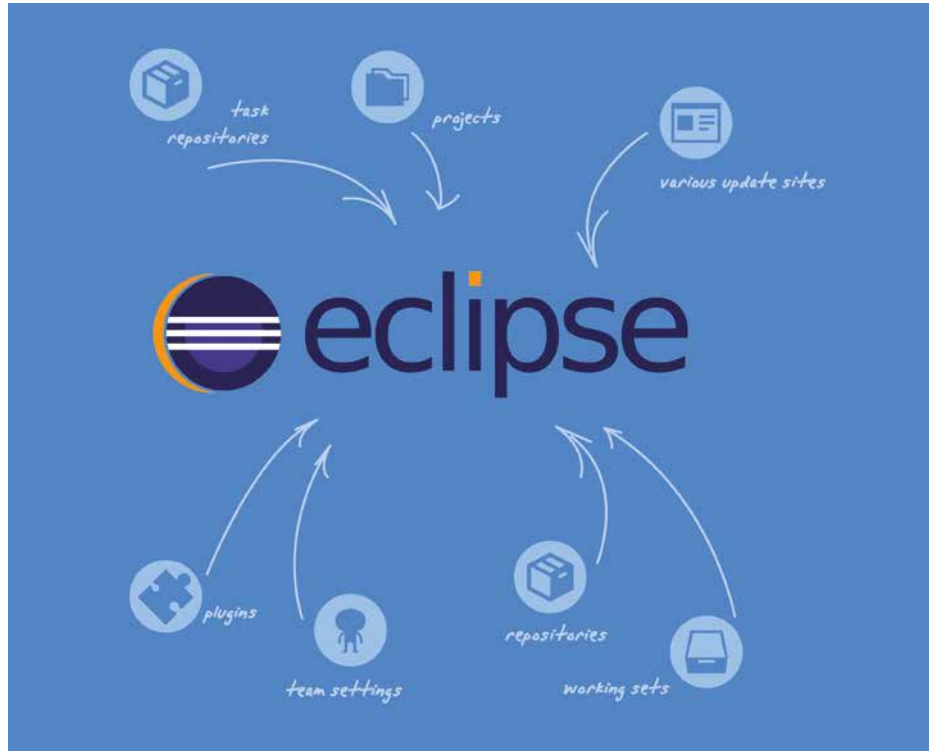


Abbildung 3: Projekt-Setups enthalten viele verschiedene Komponenten. Universelle Lösungen für eine leistungsfähige Infrastruktur für Verwaltung und Verteilung sind Mangelware.

zusammengestellt und anschließend in der laufenden IDE Zugriff auf eine spezielle Perspektive gewährt, um die Workspace-Setup-Tasks zu konfigurieren (etwa Git Repositories, Mylyn, Target Platform oder Preferences).

Community-Frameworks

Oomph ist ein Eclipse-Open-Source-Projekt und liefert als Framework die Technologie, ein

speziell auf ein Eclipse-Entwicklungsprojekt zugeschnittenes Eclipse-Setup zusammenzustellen. Dazu gehören zum Beispiel installierte Plug-ins, Einstellungen und Projekt-Spezifikationen wie Repositories und Projekte im Workspace oder Preferences. Für die Erstellung eines Setups werden verschiedene Tasks händisch verschiedenen Scopes zugeordnet; eine automatisierte Erstellung eines Profils ist hingegen nicht möglich. Dafür lässt sich die Konfiguration in allen Details festlegen, Oomph ist damit ein äußerst mächtiges Werkzeug. Es erfordert jedoch auch eine Menge an Know-how, die Funktionsweise von Eclipse und die verschiedenen Tasks betreffend.

Diese Vorgehensweise eignet sich für sehr detaillierte und tiefgreifende Konfigurationen und damit für ausgewiesene Eclipse-Experten. Eine Lösung für die Distribution oder Überwachung der erstellten Setups bietet Oomph allerdings nicht.

Free Services

Profiles ergänzt die Eclipse IDE hierfür um einen kostenfreien Service. Dieser besteht aus einer Desktop App, dem Launcher, zum Starten und Managen verschiedener Eclipse-Profiles, einem Plug-in und einem freien Webservice – dem eHub – für das Hosting der hochgeladenen, öffentlichen Profile. Mit Profiles lassen sich laufende

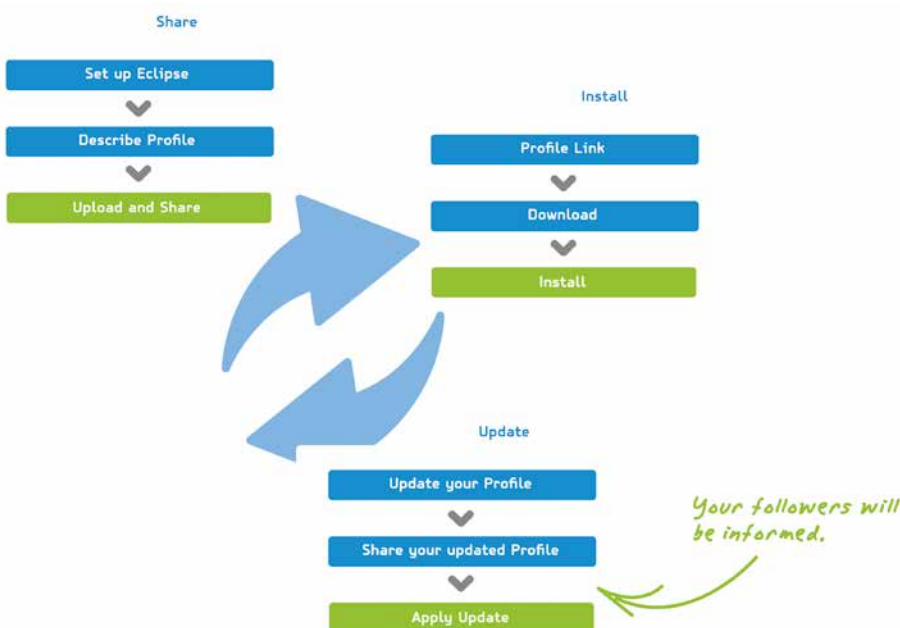


Abbildung 4: Erstellen, Teilen, Downloaden und Updaten: Profile ersparen viele Arbeitsschritte.

Eclipse-Instanzen auf Knopfdruck als Profile speichern. Zudem können sie mit Namen und Beschreibung versehen werden sowie – auf weiteren Knopfdruck – auf den eHub hochgeladen und als Link im Team verteilt werden.

Ein Profil beinhaltet dabei installierte Plug-ins, einen eingerichteten Workspace mit allen wichtigen Einstellungen (Projekt-Metadaten und -Preferences, Buildserver-Konfigurationen und Working Sets) sowie Task- und Source-Code-Repositories (Benutzernamen und Passwörter werden hingegen bewusst nicht gespeichert).

Teilen lassen sich Profile sowohl über den Launcher als auch über das Eclipse-Plug-in. Hierzu wird im Wesentlichen ein Link zu dem betreffenden Profil verteilt – zum Beispiel per E-Mail, in einem Wiki oder auch über soziale Netzwerke, falls gewünscht. Hochgeladene Profile können zudem jederzeit aktualisiert werden: Profil-Anwender werden automatisch über ein verfügbares Update informiert. Zusätzlich kann jeder User die eigenen Profile über den Launcher zentral verwalten und auch starten.

Profile sind über verschiedene Betriebssysteme hinweg einsetzbar, und es lassen sich aufeinander aufbauende Profilketten bilden. Gerade Letzteres ist für große Teams mit einem rudimentären Grund-Setup, Projekt-Setups und speziellen Anwenderprofilen oftmals unerlässlich (siehe Abbildung 4).

Zurzeit befindet sich Yatta Profiles – wenn auch stabil – noch in der Public-Beta-Phase. Einige Funktionen, insbesondere das „private sharing“ von Profilen zwischen einzelnen Usern, sind noch nicht veröffentlicht. Aktuell werden deshalb auch keine Zugangsdaten oder etwaige Passwörter von Anwendern im Profil gespeichert.

Das Hosting findet allerdings ausschließlich auf deutschen Servern statt. Das Eclipse-Mitgliedsunternehmen gewährleistet auf diese Weise volle Compliance und einen Sicherheitsstandard entsprechend dem strengen deutschen Datenschutzrecht – eine Anforderung, die bisherige Lösungen nicht zuverlässig erfüllen. Die weiter oben genannten kommerziellen Anbieter bieten Lösungen für das Enterprise-Hosting an; auch für Profiles sind entsprechende Schnittstellen und Angebote bereits in Arbeit.

Die richtige Lösung

Grundsätzlich hängt die Entscheidung für eine Lösung von den konkreten Anforderungen des Unternehmens ab. Angesichts der Heterogenität von Projekten, Vorlieben und Policies lässt sich eine allgemein gültige Empfehlung nicht aussprechen. Gemeinsam ist allen oben genannten Lösungen, dass sie zu Performance, Zuverlässigkeit und Skalierbarkeit von Eclipse im Enterprise-Umfeld erheblich beitragen. Im Einzelnen unterscheiden sie sich bei Pricing, Funktionsumfang und Komplexität jedoch erheblich.

Für Einsteiger bietet Yatta Profiles sicher einen geeigneten Service, um Eclipse schnell und einfach in Teams aufzusetzen und zu verteilen. Bei weiteren Anforderungen an Profilinhalte kann zudem ergänzend auf das Oomph-Framework zurückgegriffen werden; Profiles setzt auf Eclipse Oomph auf. Schließlich lassen sich anhand von Profiles auch Anforderungen an die Infrastruktur sukzessive erfassen, um dann zielgerichtet auf kommerzielle Angebote zurückzugreifen. Dazu ist Profiles auch gedacht: um die Eclipse Community nachhaltig und dauerhaft zu stärken, sei es durch Open-Source-, freie oder kommerzielle Angebote.

Frederic Ebelshäuser
profiles@yatta.de



Frederic Ebelshäuser ist Projektleiter und Software Engineer bei Yatta. Er entwickelte eCommerce-Lösungen als Consultant in verschiedenen Bereichen. Heute kümmert sich Frederic als Projektleiter um Eclipse-Integrationen von Yatta, darunter auch um die Entwicklung der Profiles for Eclipse. Außerdem engagiert er sich als Speaker mit Vorträgen in der Eclipse-Community. Frederic Ebelshäuser begeistert sich für kulturelle Erlebnisreisen und frühstückt am liebsten in Barcelona.

Sophie Hollmann
profiles@yatta.de



Sophie Hollmann ist technische Redakteurin und bei Yatta mit der umfassenden Dokumentation von Yatta Profiles for Eclipse betraut. Die Spezialistin in Sachen „strukturierter Text“ ist für die knackige Aufbereitung wichtiger Informationen zuständig. Die vielen verschiedenen Informationen und technischen Daten in ihrem Kopf hält sie in der Freizeit mit Cellospielen und Radtouren unter Kontrolle.

Kritische Lücke in Java SE: Oracle veröffentlicht außerplanmäßiges Update

Oracle hat ein Patch für Java SE bereitgestellt, das eine kritische Sicherheitslücke schließen soll. Der Fehler erlaubt es Angreifern, ein System vollständig zu übernehmen. Betroffen sind Oracles Java SE 7 Update 97 und Version 8 Update 73 und 74 für Win-

dows, Mac OS X, Linux und Solaris. Nach Angaben von Oracle können Angreifer die Lücke dazu nutzen, um über eine speziell präparierte Webseite Schadcode aus der Ferne und ohne Authentifizierung einzuschleusen und auszuführen. Die Schwachstelle ist auf der Skala des

umfassenden „Common Vulnerability Scoring System“ mit 9,3 Punkten bewertet, wobei 10 die höchste Stufe darstellt. Da die technischen Details der Schwachstelle bereits veröffentlicht wurden, empfiehlt Oracle allen Nutzern dringend zu einem Update.



JAXB und Oracle XDB

Wolfgang Nast, MT AG

Der Artikel handelt vom Speichern von JAXB-Daten in der Oracle-XDB 12c. Es wird gezeigt, wie man die Daten in unterschiedlichen Formaten in der Datenbank ablegen kann. Zuerst werden die unstrukturierten Speichermöglichkeiten CLOB, BLOB und XmlType vorgestellt. Hier wird auch die Übertragung für die drei Strukturen vorgestellt. Mit dem XmlType ist auch eine strukturierte Speicherung der XML-Daten möglich.

In der Ausgangslage werden die Daten in Java mit JAXB verarbeitet. Dafür kommt ein Schema zum Einsatz, das Grundlage für die Java-Klassen ist. Alternativ lässt sich auch das Schema aus annotierten Java-Klassen generieren. Listing 1 zeigt das XML-Schema. Daraus ergeben sich diese Klassen (siehe Abbildung 1).

Für die Verwendung der Oracle-XDB 12c sind die Dateien „ojdbc.jar“, „xdb.jar“ und „xmlparserv2.jar“ erforderlich. Beim Einsatz von „FastInfoset“ wird noch „FastInfoset.jar“ des Projekts „fi.java.net“ benötigt.

Speicherung als CLOB

Zuerst werden die Daten einfach gespeichert.

Dafür eignet sich der Datentyp Character Large Object (CLOB), mit dem das XML als Text hin-

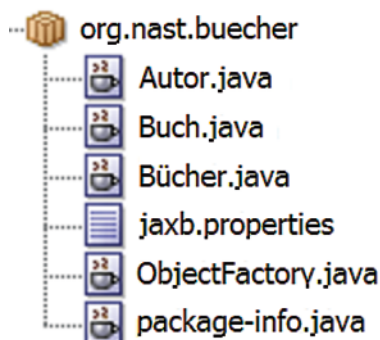


Abbildung 1: JAXB-generierte Klassen

```

<?xml version="1.0" encoding="windows-1252" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://www.nast.org/Buecher"
  targetNamespace="http://www.nast.org/Buecher"
  elementFormDefault="qualified">
  <xsd:element name="Buch">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Autor"/>
      </xsd:sequence>
      <xsd:attribute name="Titel" type="xsd:string" use="required"/>
      <xsd:attribute name="Beschreibung" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Bücher">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="Buch" minOccurs="0" maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="Id" use="required" type="xsd:integer"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="Autor">
    <xsd:complexType>
      <xsd:attribute name="Name" use="required" type="xsd:string"/>
      <xsd:attribute name="Vorname" type="xsd:string"/>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
  
```

Listing 1

terlegt ist. Zusätzlich ist noch der Schlüssel für die Daten notwendig. Dabei ist möglichst darauf zu achten, dass dieser aus dem XML genommen wird. Ist das nicht sinnvoll möglich, so ist ein technischer Schlüssel zu generieren. Hierzu wird die Tabelle „create table clobbuch (titel varchar2(80) primary key, daten clob);“ verwendet. Darin können die Daten mit folgendem Programmcode gespeichert werden (siehe Listing 2).

Mit dem Programmcode aus Listing 3 werden die Daten aus der Tabelle gelesen. Da schon ein Reader und Writer verwendet wird, ist keine Angabe einer Codierung notwendig. Dafür ist der CLOB zuständig.

Speicherung als BLOB

Als zweite einfache Speicherung sind die Daten in einem Binary Large Object (BLOB)

abgelegt. Hier sollten die XML-Daten gepackt werden. Dies geht mit GZip und FastInfoset. FastInfoset ist eine binäre Darstellung von XML und wird nicht direkt von Java unterstützt. Hier eignet sich die Implementierung von „fi.java.net“.

Anmerkung: Es empfiehlt sich, nicht die internen Klassen aus dem Package „com.sun.xml.internal.fastinfoset“ zu verwenden. Es

```
public void schreibeClobBuch(Buch buch) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Marshaller mar = cont.createMarshaller();

    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("insert into clobbuch values (?, ?)")) {
        Clob clob = con.createClob();
        mar.marshal(buch, clob.setCharacterStream(1));
        ps.setString(1, buch.getTitel());
        ps.setClob(2, clob);
        ps.executeUpdate();
    } catch (SQLException e) {
        System.out.append("Fehler beim Schreiben des Buches: ").println(e.getMessage());
    }
}
```

Listing 2

```
public Buch leseClobBuch(String titel) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Unmarshaller umar = cont.createUnmarshaller();
    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("select daten from clobbuch where titel = ?")) {
        ps.setString(1, titel);
        try (ResultSet rs = ps.executeQuery()) {
            rs.next();
            Clob clob = rs.getClob(1);
            return (Buch)umar.unmarshal(clob.getCharacterStream());
        }
    } catch (SQLException e) {
        System.out.append("Fehler beim Lesen des Buches: ").println(e.getMessage());
    }
    return null;
}
```

Listing 3

```
public void schreibeBlobBuch(Buch buch) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Marshaller mar = cont.createMarshaller();

    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("insert into blobbuch values (?, ?)")) {
        Blob blob = con.createBlob();
        try (GZIPOutputStream gout = new GZIPOutputStream(blob.setBinaryStream(1))) {
            XMLStreamWriter xw = new StAXDocumentSerializer(gout);
            mar.marshal(buch, xw);
        }
        ps.setString(1, buch.getTitel());
        ps.setBlob(2, blob);
        ps.executeUpdate();
    } catch (SQLException | IOException e) {
        System.out.append("Fehler beim Schreiben des Buches: ").println(e.getMessage());
    }
}
```

Listing 4

```

public Buch leseBlobBuch(String titel) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Unmarshaller umar = cont.createUnmarshaller();

    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("select daten from blobbuch where titel = ?") {
        ps.setString(1, titel);
        try (ResultSet rs = ps.executeQuery()) {
            rs.next();
            Blob blob = rs.getBlob(1);
            try (GZIPInputStream gin = new GZIPInputStream(blob.getBinaryStream());
                XMLStreamReader sp = new StAXDocumentParser(gin);
                return (Buch)umar.unmarshal(sp);
            }
        }
    } catch (SQLException | IOException e) {
        System.out.append("Fehler beim Lesen des Buches: ").println(e.getMessage());
    }
    return null;
}

```

Listing 5

```

public void schreibeXMLBuch(Buch buch) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Marshaller mar = cont.createMarshaller();

    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("insert into xmlbuch values (?, ?)") {
        SQLXML sxml = con.createSQLXML();
        mar.marshal(buch, sxml.setResult(StAXResult.class));
        ps.setString(1, buch.getTitel());
        ps.setSQLXML(2, sxml);
        ps.executeUpdate();
    } catch (SQLException e) {
        System.out.append("Fehler beim Schreiben des Buches: ").println(e.getMessage());
    }
}

```

Listing 6

```

public Buch leseXMLBuch(String titel) throws JAXBException {
    JAXBContext cont = JAXBContext.newInstance(Buch.class);
    Unmarshaller umar = cont.createUnmarshaller();

    try (Connection con = DriverManager.getConnection(DB_URL, USER, PWD);
        PreparedStatement ps = con.prepareStatement("select daten from xmlbuch where titel = ?") {
        ps.setString(1, titel);
        try (ResultSet rs = ps.executeQuery()) {
            rs.next();
            SQLXML sxml = rs.getSQLXML(1);
            return (Buch)umar.unmarshal(sxml.getSource(DOMSource.class));
        }
    } catch (SQLException e) {
        System.out.append("Fehler beim Lesen des Buches: ").println(e.getMessage());
    }
    return null;
}

```

Listing 7

sind die gleichen Klassen, aber nicht offiziell vorhanden. Sie können bei jedem Update von Java umbenannt oder ersetzt werden, womit die Java-Anwendung dann nicht mehr läuft.

Es kommt die Tabelle „create table blobbuch (Titel varchar2(80) primary key, daten blob);“ zum Einsatz. *Listing 4* zeigt das Programm zum

Schreiben von BLOB mit GZip und FastInfoset, *Listing 5* das Programm zum Lesen der Daten.

Dabei ist die Codierung der XML-Daten Aufgabe des Clients, da die Datenbank nur die Binär-Daten des BLOB liest und schreibt. Bei dieser Art der Speicherung ist das Format der Daten von der Java-Anwendung

vorgegeben; das führt zu Problemen, wenn man über andere Wege wie SQL*Plus an die Daten kommen möchte.

Speicherung als „XmlType“

Die dritte Möglichkeit ist der für XML definierte Daten-Typ „XmlType“, der von Oracle

unterstützt und in Java mit „SQLXML“ angesprochen wird. Man kann den „SQLXML“-Typ in den Oracle-Java-Typ „XMLType“ umwandeln (nur beim Oracle-JDBC-Treiber). Diesmal wird die Tabelle „create table xmlbuch (Titel varchar2(80) primary key, daten XMLTYPE);“ verwendet. *Listing 6* zeigt den Programmteil zum Schreiben von „XmlType“, *Listing 7* den zum Lesen von „XmlType“.

Bei der Verwendung von „XmlType“ in Java-„SQLXML“ ist bei der Codierung der Daten aufzupassen. Beim Schreiben mit „`sxml.getResult(SAXResult.class)`“ und „`sxml.getResult(StreamResult.class)`“ gibt es Probleme bei der Codierung. Es tritt kein Fehler beim Schreiben auf, doch die Daten werden mit der falschen Codierung gespeichert. Beim Lesen mit „`sxml.getSource(SAXSource.class)`“, „`sxml.getSource(StAXSource.class)`“ und „`sxml.getSource(StreamSource.class)`“ gibt es auch Probleme bei der Codierung. Diese führen zu einem sofortigen Abbruch der Verarbeitung.

„XmlType“ mit XML-Schema

In der Oracle-Datenbank können XML-Schemata registriert sein, um die Daten besser speichern und validieren zu können. Beim Daten-Typ „XmlType“ kann angegeben werden, ob es nur XML-konform sein oder einem Element im registrierten XML-Schema entsprechen muss. Hier gibt es viele Möglichkeiten, die Speicherung im „XmlType“ zu optimieren. Sie sind alle abhängig von den Daten

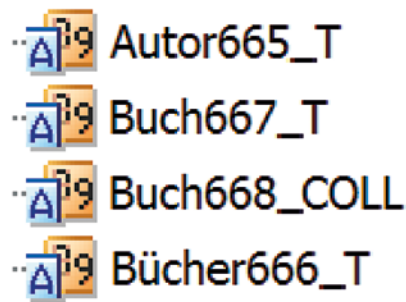


Abbildung 2: Generierte Datenbank-Objekte

und ihrer Verwendung. Diese Optimierungen haben keinen Einfluss auf die Verwendung in Java. Mit „create table xmlbuch2 (Titel varchar2(80) primary key, daten XMLTYPE) XMLTYPE COLUMN daten XMLSCHEMA "http://www.nast.org/buecher.xsd" element "Buch"“ wird die Tabelle um das Schema „Beschreibung“ ergänzt. Beim Registrieren des Schemas werden Datenbank-Typen erzeugt (siehe *Abbildung 2*).

Am Zugriff von Java ändert sich nichts, er bleibt wie vorher beschrieben. Als Element des Schemas bei XmlType ist eines der Top-level-XML-Elemente anzugeben. Bei dem Beispiel Schema sind es „Buch“, „Autor“ und „Bücher“.

Fazit

Es gibt unterschiedliche Möglichkeiten, XML-Daten mit JAXB in eine Oracle-Datenbank

zu speichern. Mit dem CLOB ist man sehr unabhängig von der Datenbank, dafür sind die Daten groß. Als Unterstützung von der Datenbank gibt es die textuelle Suche im CLOB. BLOB reduziert die Daten stark, es gibt jedoch keine Unterstützung von der Datenbank. Mit „XmlType“/„SQLXML“ kann die Datenbank die meiste Unterstützung bieten, da die XML-Strukturen auf die Strukturen der XDB von Oracle abgebildet werden.

Wolfgang Nast

wolfgang.nast@mt-ag.com



Wolfgang Nast (Dipl. Ing.) ist Senior Berater bei der MT AG und seit dem Jahr 1998 mit Java SE sowie seit dem Jahr 2006 mit Java EE in den Bereichen „WebServices“ sowie „Architektur“ beratend und umsetzend tätig.

Spannende Diskussionen & gute Laune: Das DOAG DevCamp 2016 war ein voller Erfolg

Es war nicht das, was man sich üblicherweise unter einer IT-Fachkonferenz vorstellt: Wer am 23. und 24. Februar aus der Lobby des Kameha Grand durch die verglasten Wände in Richtung Kongressräume blickte, fand ein buntes Treiben vor. Zu unterschiedlichen Zeiten strömten Gruppen aus lässig gekleideten Menschen aus den Kongressräumen, einige setzten ihre Gespräche an der Kaffeebar fort, wieder andere wechselten permanent die Räume. Auch in den Sessions erinnerte nichts an typische IT-Konferenzen: die Teilnehmer diskutierten aktiv mit den Referenten. Zu manchen Zeiten traf man auf Diskutanten, die sich wild gestikulierend vor einem Laptop-Bildschirm versammelt hatten.

Die Rede ist vom DevCamp 2016, das wie in den letzten beiden Jahren im Barcamp-Format stattfand. Die Regeln erklärte Robert Szilinski, Leiter der Development-Community, den Barcamp-Neulingen gleich zu Beginn des Events: Jeder darf sich aktiv einbringen, alle sind per du, die Sessions beginnen, wann sie beginnen und enden, wann sie eben enden. Auch die Themenfindung gestaltete sich alles andere als gewöhnlich: die Inhalte wurden am Morgen von den Teilnehmern selbst vorgeschlagen – je nach Interesse einigte man sich anschließend auf den passenden Raum. Manch einer ließ sich dabei von anderen inspirieren und schlug spontan weitere Themen vor.

Die eigentlichen Sessions deckten die gesamte Palette der Oracle-Entwicklertemen ab: Forms, PL/SQL, APEX, ADF, SOA/BPM, JET, und natürlich waren Mobile und Cloud besonders heiß diskutierte Punkte. Auch Themen wie „Teambuilding“, „Work/Life-Balance“ und „Kundenumgang“ standen auf der Agenda. In einem zusätzlichen Raum wurden neue Tools vorgestellt – im Anschluss hatten die Teilnehmer die Möglichkeit, diese in praktischen Hands-On-Sessions selbst auszuprobieren. Außerdem war das ADF Fitness Center – bisher eine separate Oracle-Veranstaltung – in das DevCamp integriert. Networking war dank des offenen Formats durchgehend möglich.



Java-Enterprise-Anwendungen effizient und schnell entwickeln

Anett Hübner, H&D International Group

Zu Beginn eines JEE-Web-Projekts ist die Erstellung einer neuen Architektur insbesondere bei kleinen Anwendungen erfahrungsgemäß mit hohem Zeitaufwand verbunden. Weitere Zeit geht bei der Realisierung und Wartung von wiederkehrenden Funktionen verloren. Das Erstellen einer nutzerfreundlichen Oberfläche ist ein Hauptkriterium für die Akzeptanz einer Software und wird oft stiefmütterlich behandelt. Schwerpunkte des hier vorgestellten Architektur-Ansatzes sind die Beschleunigung des Projektstarts, das Ermöglichen einer schnelleren Entwicklung von Standard-Funktionalitäten und ein Ansatz für die effiziente Integration von Web-Designern in ein Projekt.

Welche Eigenschaften sollte eine gute Architektur haben? „Einfach“, „klein“, „einheitlich“, „flexibel“ und „effizient“ sind sehr allgemeine Stichwörter, die in diesem Zusammenhang oft genannt werden. Die entstandene Software soll zudem nutzerfreundlich sein. Diese Eigenschaften dienen als Leitgedanken für die weiteren Betrachtungen.

„Einfach“ und „klein“: Die Entwicklung von kleinen, auf einen Anwendungsbereich spezialisierten Software-Lösungen ist unter dem Begriff „Microservice“ bekannt geworden und sorgt gerade in der letzten Zeit immer wieder für Schlagzeilen [1, 2]. Netflix, Ebay und Amazon geben Beispiele dafür, wie diese Idee realisiert werden kann. Um monolithische Systeme zu vermeiden, sind mehrere

kleine Anwendungen auch das Ziel des hier beschriebenen Architektur-Ansatzes. Geringe Komplexität führt zu einer Verringerung des Arbeitsaufwands während der Software-Entwicklung, der Wartung und des Betriebs.

„Einheitliche Prinzipien“: Ein automatisiertes Projekt-Setup, eine einheitliche Projekt-Struktur und Code-Templates über mehrere Web-Anwendungen hinweg können für stabile und dennoch flexible Architektur-Lösungen sorgen. Die Erfahrungen aus mehreren Projekten fließen in die Weiterentwicklung von Projekt-Templates ein. So kann Flexibilität mit einer bewährten einheitlichen Architektur kombiniert werden.

„Effizient“: Beim Erstellen von modernen Web-Anwendungen müssen bekannterma-

ßen immer wiederkehrende Aufgaben gelöst werden, beispielsweise das Speichern und der Zugriff auf Daten in Datenbanken. Der JEE-Technologie-Stack ermöglicht es heute, sehr leicht Daten mittels Java-Persistence-API (JPA) in Datenbanken abulegen. Dennoch muss noch immer etliches an Code geschrieben werden, damit Daten von der Nutzeroberfläche in die Datenbank gelangen können. Dem sollte durch ein entsprechendes Design entgegengewirkt werden.

„Flexibel“: Um die Architektur in mehreren Projekten einsetzen zu können, muss es trotz Templates und der vorgegebenen Rahmen möglich sein, Anpassungen an Projektstruktur und Quellcode vorzunehmen.

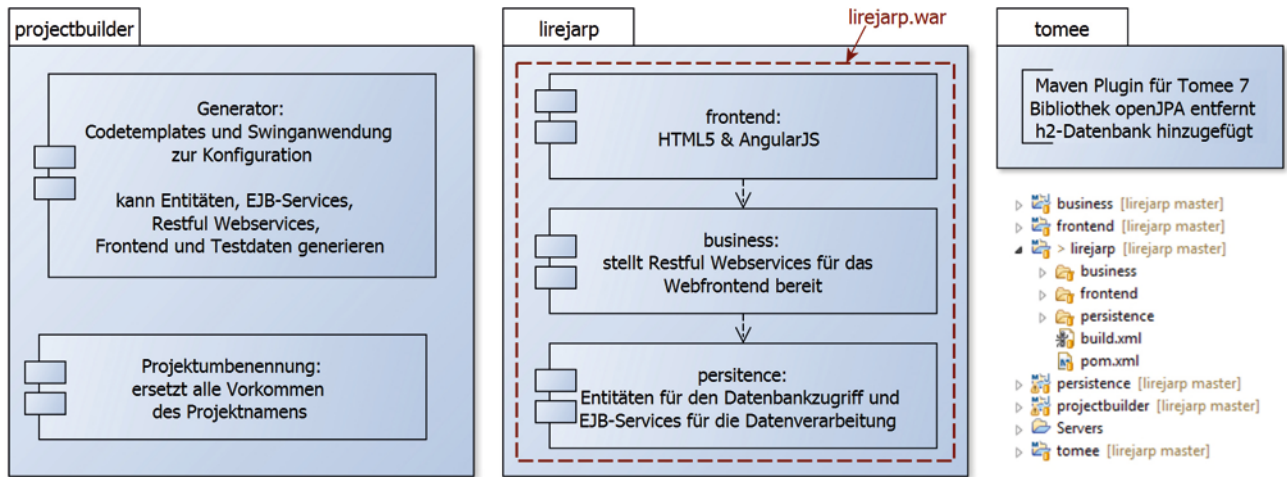


Abbildung 1: Projektaufbau als Komponenten-Diagramm (links) und Projekt-Struktur in Eclipse (rechts)

„Nutzerfreundlich“: Der Erfolg neuer Anwendungen steht und fällt mit der Akzeptanz der Anwender. Design, Nutzerführung und Ästhetik spielen dabei eine zentrale Rolle. Das effiziente Einbinden von Experten bei der Erstellung des Interaktionsdesigns ist deshalb ein wichtiges Ziel des hier vorgestellten Ansatzes. Die Möglichkeit, Designer direkt am Quellcode für die Oberflächen-Repräsentation arbeiten zu lassen, stellt sich in den meisten Java-Web-Projekten jedoch als schwierig heraus. Dies ist die Folge von Technologien und Architekturen, die die Implementierung von Nutzeroberflächen stets Entwicklern zuschreiben. Eine konsequente Trennung von Logik und Repräsentation und die Einbindung des typischen Designer-Workflows fehlen fast immer.

Projekt-Aufbau

Das Projekt „LireJarp“ kann über GitHub [3] heruntergeladen werden und besteht aus folgenden drei Modulen:

- Das Modul „persistence“ ermöglicht den Zugriff auf die Datenbank. Geschäftslogik, die transaktionsgesichert ablaufen muss, kann hier über Enterprise-Java-Beans implementiert werden
- Das Modul „business“ stellt Funktionalität des „persistence“-Moduls als RESTful-Webservices bereit und dient somit als Schnittstelle für die Oberfläche
- Im „frontend“-Modul wird die Präsentation und Steuerung der Oberfläche implementiert

Um das Projekt nicht unnötig aufzublähen, wurde bewusst auf weitere Schichten verzichtet. Zwei zusätzliche Maven-Projekte

helfen bei der Einrichtung und Entwicklung der Web-Anwendung:

- Das Projekt „tomee“ benutzt das Tomee-Maven-Plug-in, um einen vorkonfigurierten Webserver zur Verfügung zu stellen
- Der „ProjectBuilder“ unterstützt das Projekt-Setup und die initiale Erstellung der Grundfunktionalitäten durch Code-Templates

Abbildung 1 zeigt die Projektstruktur zusammenfassend. Die genaue Erläuterung der einzelnen Komponenten erfolgt in den nachfolgenden Abschnitten.

Einrichten eines Projekts

Für das Bauen und Ausführen des Projekts sind Maven 3.3.3 als Build-Tool und JDK 1.7

erforderlich. In den nachfolgenden Beispielen kommt Eclipse Mars als Entwicklungsumgebung zum Einsatz. Zunächst müssen die benötigten Maven-Projekte mit „git clone“ ausgecheckt und anschließend in Eclipse als Maven-Projekte importiert werden.

Der nächste Schritt ist das Bauen des Projekt-Templates über den Aufruf von „mvn clean install“ im „LireJarp“-Projektordner. Als Ergebnis legt Maven eine „war“-Datei in das „target“-Verzeichnis. Danach kann im „tomee“-Projekt der vorkonfigurierte Server über den Befehl „mvn toomee:build“ heruntergeladen werden. Dieser ist so konfiguriert, dass der „target“-Ordner, in dem die „war“-Datei liegt, als WebApp-Ordner genutzt wird. Damit werden neu gebaute Versionen automatisch eingerichtet.

In der Standard-Konfiguration ist „H2“ als Datenbank angelegt. Die Verwendung von

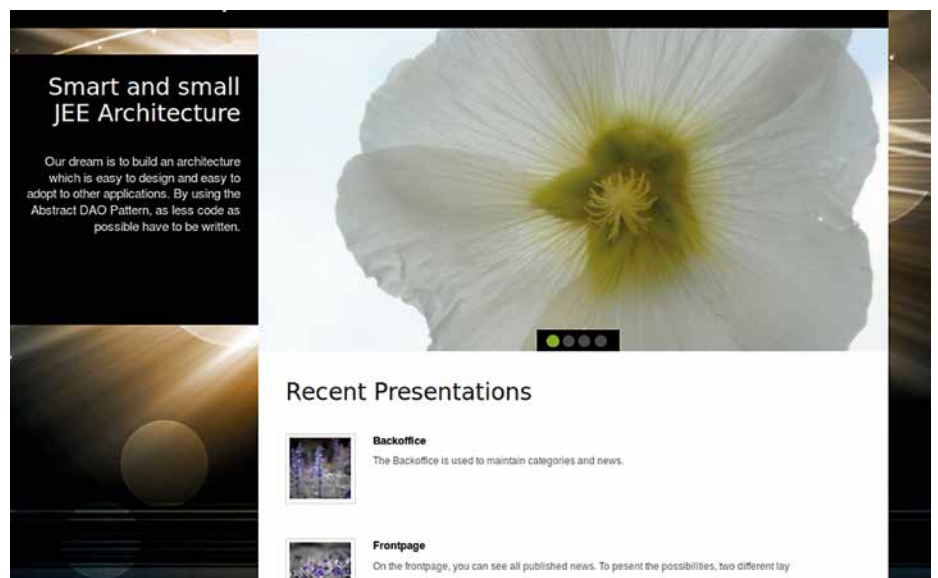


Abbildung 2: Start-Bildschirm der Beispiel-Anwendung

```
public class AbstractServiceImpl<E extends AbstractEntity>
public class CategoryServiceImpl extends AbstractServiceImpl<CategoryEntity>
```

Listing 1: Abstrakte und konkrete Klassen-Deklaration mit generischem Typ

„JPA“ kann diese leicht durch eine richtige Datenbank ersetzen. Die Web-Anwendung kann schließlich mit „<http://localhost:8080/lirejarp>“ aufgerufen werden und ein Start-Bildschirm erscheint (siehe Abbildung 2).

„ProjectBuilder“ dient als plattformunabhängiges Hilfsmittel für die Umbenennung des Projektnamens im gesamten Quellcode und aller Spezifikationsdateien. Er ist entweder per Kommandozeile oder über eine Swing-Anwendung benutzbar (siehe Abbildung 3).

Realisierung von Standard-Funktionalität mithilfe generischer Daten-Typen

Das übliche Vorgehen in heutigen Software-Projekten ist es, in der Anforderungsanalyse

zunächst die Geschäftsobjekte zu ermitteln und daraus ein Domänenmodell und die Datenbank-Struktur abzuleiten. Der Schwerpunkt des Software-Designs liegt somit auf der Fachlichkeit und erfolgt nach den Prinzipien des Domain-Driven-Designs [5]. Für die Objekte oder Entitäten im Domänenmodell werden meist Standardfunktionalitäten wie Erstellen, Anzeigen, Bearbeiten und Speichern (CRUD = Create, Read, Update, Delete) benötigt. Die Umsetzung dieser Standard-Funktionalität ist nachfolgend beschrieben.

Ein schlankes Design mit einer möglichst geringen Anzahl von Code-Duplikaten trägt viel zur Übersichtlichkeit eines Software-Projekts bei und verspricht eine höhere Wartbarkeit und geringere Fehlerrate. Ein Ansatzpunkt

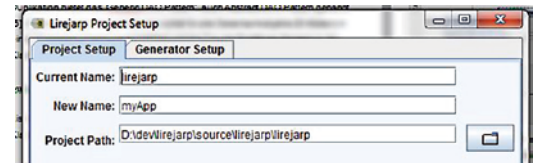


Abbildung 3: Umbenennen des Projekts mittels „ProjectBuilder“

für das Vermeiden von Duplikation bietet das „Generic DAO Pattern“, auch „Abstract DAO Pattern“ genannt [6]. Hierbei werden Standard-Funktionen für alle Datenobjekte (Entitäten) in einer abstrakten Klasse implementiert und der Typ der Entität als Generic in der Klassendeklaration übergeben (siehe Listing 1). Der Typ wird in den konkreten Implementierungen der abstrakten Klasse mitgegeben.

Überträgt man diesen Ansatz auf alle Schichten des Java-Teils der Anwendung (RESTful Webservices, Arquillian Tests [7] und Enterprise Java Beans), minimiert sich die Anzahl von Duplikaten und der Quellcode schrumpft weiter zusammen. Außerdem trägt die konsequente Verwendung von Patterns wie diesem dazu bei, dass die Nachvollziehbarkeit von und die Einarbeitung in bestehenden Code einfacher wird. Abbildung 4 skizziert den Architektur-Ansatz als Klassen-Diagramm. Die abgebildeten Schichten werden anschließend näher erläutert.

„Persistenz-Schicht“: Der Datenbank-Zugriff erfolgt nach JEE-Standard über JPA, wobei Hibernate als Implementierung benutzt wird. In den Entitäten ist somit festgelegt, welche Attribute an der Oberfläche angezeigt und in der Datenbank gespeichert werden können. Sie dienen als Domänen-Objekte. Über Bean Validation [13] kann die Prüflogik für Attribute definiert werden. Listing 2 zeigt Ausschnitte aus der „CategoryEntity“ mit entsprechenden Annotationen für Datenbank-Felder und Validierungen. Die verwendeten XML-Annotationen sind die Grundlage für die Erzeugung von JSON-Objekten, um die RESTful-Services auszuliefern.

„EJB-Services“: Über die Service-Klassen werden Funktionen für die jeweiligen Domänen-Objekte realisiert. Wie bereits beschrieben, sind auch hier die Standard-Funktionen über eine abstrakte Klasse bereitgestellt und in konkreten Service-Klassen befindet sich nur noch die Implementierung spezifischer Funktionalität.

„Integrationstests über Arquillian“: Die pro Domänen-Objekt erstellten EJB-Services werden mit JUnit getestet. Arquillian stellt einen Container für das Ausführen der Tests zur Verfügung und startet dabei den TomEE-Server im

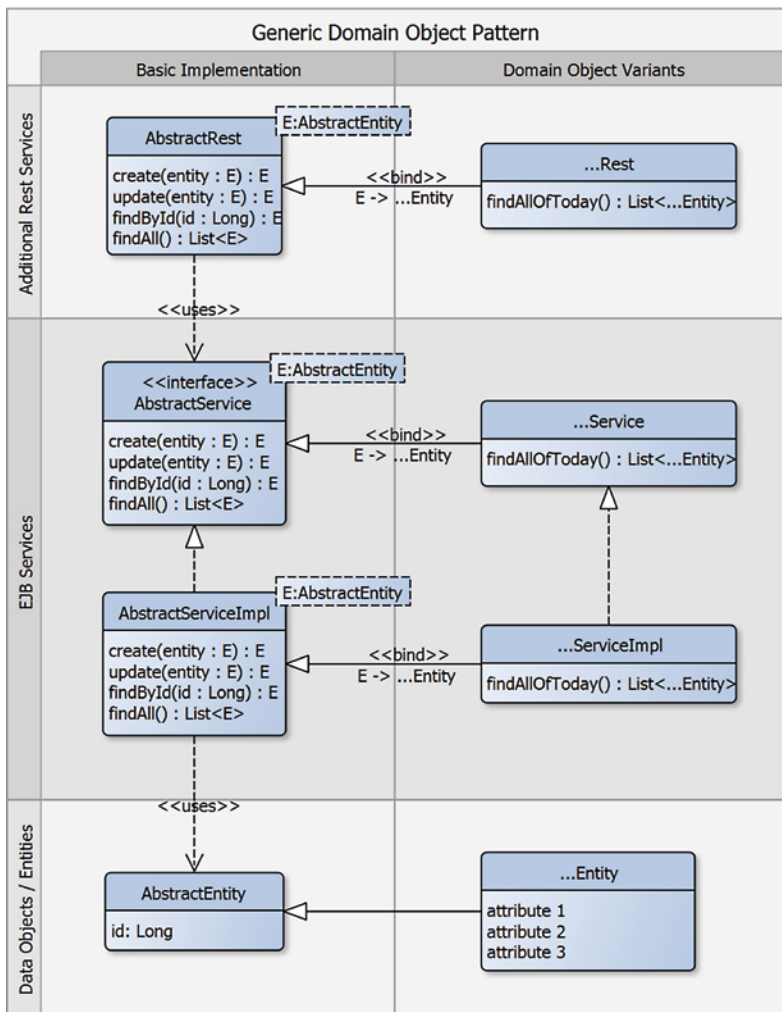


Abbildung 4: Aufbau des Java-Teils der „LireJarp“-Architektur

„embedded“-Modus. Auch in diesem Fall wird eine abstrakte Klasse mit generischen Typen verwendet. Das Einbinden von Testdaten, das Löschen eines Datensatzes und der Zugriff auf die Daten sind somit generell prüfbar. *Listing 3* und *4* zeigen die Klassen-Deklarationen und den Einsatz der generischen Typen.

„Rest-Services“: RESTful-Webservices bilden die Schnittstelle zum Frontend und ermöglichen eine saubere Trennung zwischen Oberfläche und Business-Logik.

Die an den Entitäten definierte Validierung wird an dieser Stelle aufgegriffen, durchgeführt und das Ergebnis wird für die Weiterverwendung im Browser aufbereitet. Um Redundanzen zu vermeiden, wurde bewusst auf ein Mapping in andere Daten-Objekte verzichtet. Da Entitäten über Transaktionsgrenzen hinweg verwendet werden, ist es leicht möglich, die Aktualisierung der Daten mit den Werten aus der Datenbank zu vergessen. Der Aufruf von „getManager().merge(entity)“ beseitigt dieses Problem.

Strikte Trennung von Frontend und Geschäftslogik

Für die Web-Oberfläche kommen HTML und AngularJS [8] als Oberflächen-Technologie zum Einsatz. Der Web-Designer arbeitet direkt an CSS- und HTML-Dateien des Projekts. Zum Betrachten seiner Änderungen im Web-Browser müssen die HTML-Dateien in das Deployment-Verzeichnis des Web-Browsers kopiert werden. Um plattformunabhängig zu bleiben, wurde für diesen Zweck eine Ant-Datei erstellt.

Dies bildet den typischen Workflow eines Designers ab. Die Möglichkeit, ohne langes Neubauen der Applikation Änderungen direkt anzuzeigen, führt so zu einem erheblichen Produktivitätszuwachs. Ein Detail am Rande ist die einfache Testbarkeit des Backend (gerade in der Anfangsphase) über Tools wie den RestClient für Firefox [12] oder durch simple Shellskripte mit „curl“.

Einsatz von Code-Templates

Code-Templates und ein Generator erleichtern die Erstellung neuer Geschäftsobjekte sowie deren Funktionalitäten und Oberflächen. Über eine Swing-Anwendung sind der Name des Geschäftsobjekts und dessen Attribute einstellbar. Zusätzlich kann der Ablageort der Code-Templates angegeben werden. Welche Teile des Projekts generiert werden sollen, ist ebenfalls konfigurierbar (siehe *Abbildung 5*).

Zusätzlich zu den Java-Klassen erzeugt der Generator simple UI-Strukturen, die einerseits unmittelbar zu einer lauffähigen Anwendung

mit minimaler Funktion führen und andererseits den Einstieg in die weitere Entwicklung vereinfachen. Als Template-Engine wurde Freemarker [9] aufgrund der einfachen Lesbarkeit und der guten Dokumentation verwendet. Es unterstreicht, dass es hier nur um eine initiale Ersterzeugung von Code geht.

Der Generator fungiert folgerichtig nur als Hilfsmittel und einmal generierter Quellcode wird nicht neu generiert. Dieser pragmatische Ansatz ermöglicht das einfache Anpassen von Funktionalität und Oberfläche nach individuellen Wünschen. Zudem sind durch

die Code-Templates Namens-Konventionen und Standardvorgehen festgelegt, was in einem Team von Software-Entwicklern einen einheitlichen und damit weniger fehleranfälligen Quellcode verspricht.

Fazit

Dieser Artikel beschreibt einen Architektur-Ansatz, der als Startpunkt für neue JEE-Web-Projekte dienen soll. Dazu wurden die in *Tabelle 1* aufgelisteten Technologien verwendet. Nach dem Auschecken des Projekts werden nur drei Schritte für das Starten benötigt (Bauen des

```
@XmlElement
@Entity
@Table(name = "CATEGORY")
public class CategoryEntity extends AbstractEntity {
    ...
    @NotNull
    @Size(max = 30)
    private String name;
    ...
    @Column(name="NAME", nullable = false, length=100)
    public String getName() {
        return name;
    }
    ...
    @XmlTransient
    @OneToMany(mappedBy="category", cascade={CascadeType.ALL})
    public List<NewsEntity> getNews() {
        return news;
    }
    ...
}
```

Listing 2: Definition der Entitäten/Geschäftsobjekte

```
public abstract class AbstractServiceTest<E extends AbstractService<T>, T
extends AbstractEntity> {
    ...
    protected E service;
    protected T entity;
    ...
    @Inject
    public abstract void setService(E service);
    ...
}
```

Listing 3

```
@RunWith(Arquillian.class)
public class CatalogServiceTest extends AbstractServiceTest<CatalogService,
CatalogEntity> {
    @Override
    public void setService(CatalogService service) {
        this.service = service;
    }
    @Override
    public void testCreate() {
        ...
    }
    @Override
    public void testUpdate() {
        ...
    }
}
```

Listing 4

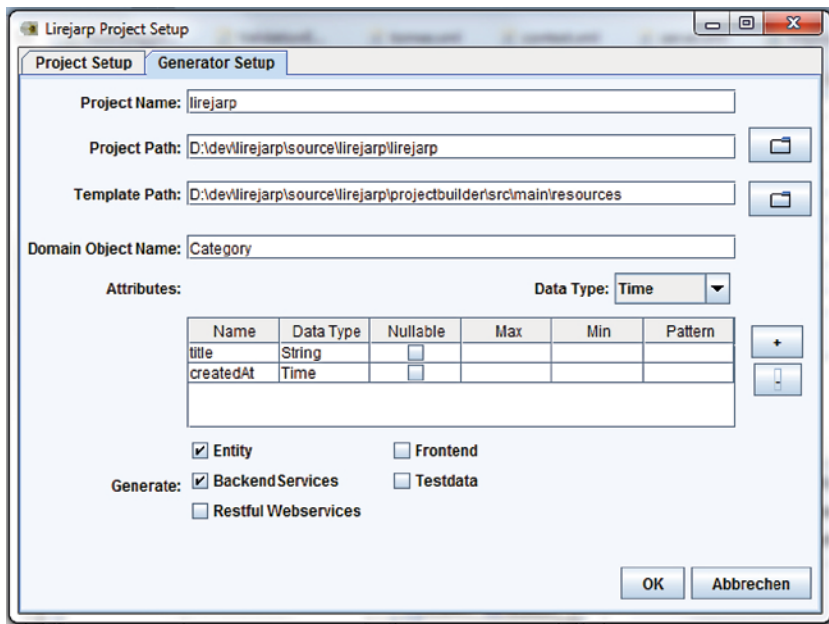


Abbildung 5: Programm für die Generierung der Geschäftsobjekte

„LireJarp“-Projekts, Bauen des „tomee“-Projekts sowie Starten des „tomee“-Servers) und das Projekt ist mit geringem Zeitaufwand eingerichtet. Eine schnelle Realisierung von Standard-Funktionalitäten (Anzeigen, Erstellen, Bearbeiten und Löschen von Domain-Objekten) ist durch einen generischen Architektur-Ansatz sowie Code-Templates gegeben.

Der Einsatz von AngularJS und HTML als Oberflächen-Technologie und RESTful-Webservices als Schnittstelle zum Backend stellen eine strikte Trennung von Oberfläche und Programmlogik dar und ermöglichen das Anpassen des Quellcodes durch einen Web-Designer. Code-Templates können die Software-Entwicklung weiterhin beschleunigen und sorgen durch Namens-Konventionen

sowie Standardvorgehen für eine geringere Fehleranfälligkeit (siehe Tabelle 1).

Ausblick

Der hier vorgestellte Architektur-Ansatz unterliegt der ständigen Weiterentwicklung und stellt somit eine Momentaufnahme dar. Als ein nächster Schritt ist geplant, die Testabdeckung im gesamten Projekt zu erhöhen. Es existieren zwar JUnit-Tests für die EJB-Services, doch fehlen automatische Tests des AngularJS-Codes, Oberflächentests mit Selenium und JUnit-Tests für die RESTful-Webservices.

Ein weiterer offener Punkt ist der Umgang mit Datenbank-Skripten. Die Erzeugung der Datenbank über Hibernate sollte nur für Test- und Entwicklungszwecke verwendet werden.

Im UI-Bereich wird der Schwerpunkt der Weiterentwicklung auf der Erstellung besserer Oberflächentemplates und einer klareren Ordnerstruktur im Frontend-Teilprojekt liegen.

Die Architektur soll schließlich um einen Standardweg zur Einbindung von Nutzeranmeldungen durch externe Verzeichnisse (LDAP = Lightweight Directory Access Protocol) erweitert werden. Dazu wird derzeit eine Komponente zum Auslesen von Nutzerdaten [11] entwickelt. Die Maven-basierte Testumgebung (TomEE/H2) soll auf Basis von ApacheDS [14] um einen LDAP-Server mit Beispieldaten erweitert werden.

Literaturverzeichnis

- [1] J.Lewis, M.Fowler, Microservices: <http://martinfowler.com/articles/microservices.html>
- [2] Sam Newman: Microservices - Konzeption und Design, mitp Verlags GmbH & Co. KG, 2015, ISBN: 978-3-95845-081-3
- [3] LireJarp-Projekt, Quellcode: <https://github.com/witchpou/lirejarp>
- [4] LireJarp-Blogartikel: <http://softengart.blogspot.de/2014/03/generic-jee-architecture.html>
- [5] Domain Driven Design: https://de.wikipedia.org/wiki/Domain-driven_Design
- [6] Generic DAO Pattern: <http://www.codeproject.com/Articles/251166/The-Generic-DAO-pattern-in-Java-with-Spring-3-and>
- [7] Arquillian Test Framework: <http://arquillian.org>
- [8] AngularJS Framework: <https://angularjs.org>
- [9] Freemarker Template-Engine: <http://freemarker.org>
- [10] HTML-Layouttemplates: <http://www.templateemo.com>
- [11] UserDataFilter: <https://github.com/ztarbug/User-DataFilter>
- [12] RestClient für Firefox: <https://addons.mozilla.org/de/firefox/addon/restclient>
- [13] Bean Validation: https://en.wikipedia.org/wiki/Bean_Validation
- [14] ApacheDS: <http://directory.apache.org>

Anett Hübner

anett.huebner@hud.de



Anett Hübner entwickelt und entwirft seit acht Jahren Software-Systeme. Dabei hat sie in vielen Branchen (Telekommunikation, eCommerce, Automobilbau) individuelle Software-Lösungen realisiert. Sie kennt die immer wiederkehrenden Aufgabenstellungen und Probleme in Software-Entwicklungsprojekten. Daher hat sie sich die Optimierung des Entwurfsprozesses und auf Architektur-Templates für Java-Enterprise-Anwendungen zum Thema gemacht.

Bereich	Verwendete Technologien
Oberfläche	HTML AngularJS 1.2.5
Validierung	BeanValidation
Rest-API	JAX-RS-2.0-Implementierung: Apache CXT JSON-Parser: Jackson 2.2.3
Business Services	EJB 3.2 (Implementierung Open EJB)
Automatische Tests	Arquillian 1.1.2 Arquillian Suite Extension 1.0.6 Arquillian Tomee Embedded 1.0.0
Datenbank (OR-Mapping)	JPA 2.1 Hibernate 4.2 2H / MySQL
Buildtool und Deployment	Maven 3.3.3 Tomee 7

Tabelle 1: Verwendete Technologien

Die iJUG-Mitglieder auf einen Blick

Java User Group Deutschland e.V.
www.java.de

DOAG Deutsche ORACLE-Anwendergruppe e.V.
www.doag.org

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

Java User Group Köln
www.jugcologne.eu

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

Java User Group München (JUGM)
www.jugm.de

Java User Group Metropolregion Nürnberg
www.source-knights.com

Java User Group Ostfalen
www.jug-ostfalen.de

Java User Group Saxony
www.jugsaxony.org

Sun User Group Deutschland e.V.
www.sugd.de

Swiss Oracle User Group (SOUG)
www.soug.ch

Berlin Expert Days e.V.
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
www.jug-h.de

Java User Group Augsburg
www.jug-augsburg.de

Java User Group Bremen
www.jugbremen.de

Java User Group Münster
www.jug-muenster.de

Java User Group Hessen
www.jugh.de

Java User Group Dortmund
www.jugdo.de

Java User Group Hamburg
www.jughh.de

Java User Group Berlin-Brandenburg
www.jug-berlin-brandenburg.de

Java User Group Kaiserslautern
www.jug-kl.de

Java User Group Switzerland
www.jug.ch

Java User Group Euregio Maas-Rhine
www.euregjug.eu

Java User Group Görlitz
www.jug-gr.de

Java User Group Mannheim
www.majug.de

Lightweight Java User Group München
www.meetup.com/de/lightweight-java-user-group-munchen

Java User Group Düsseldorf rheinjug
www.rheinjug.de

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

Java aktuell wird vom Interessenverbund der Java User Groups e.V. (iJUG) (Tempelhofer Weg 64, 12347 Berlin, www.ijug.eu) herausgegeben. Es ist das User-Magazin rund um die Programmiersprache Java im Raum Deutschland, Österreich und Schweiz. Es ist unabhängig von Oracle und vertritt weder direkt noch indirekt deren wirtschaftliche Interessen. Vielmehr vertritt es die Interessen der Anwender an den Themen rund um die Java-Produkte, fördert den Wissensaustausch zwischen den Lesern und informiert über neue Produkte und Technologien.

Java aktuell wird verlegt von der DOAG Dienstleistungen GmbH, Tempelhofer Weg 64, 12347 Berlin, Deutschland, gesetzlich vertreten durch den Geschäftsführer Fried Saacke, deren Unternehmensgegenstand Vereinsmanagement, Veranstaltungsorganisation und Publishing ist.

Die DOAG Deutsche ORACLE-Anwendergruppe e.V. hält 100 Prozent der Stammeinlage der DOAG Dienstleistungen GmbH. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. wird gesetzlich durch den Vorstand vertreten; Vorsitzender: Dr. Dietmar Neugebauer. Die DOAG Deutsche ORACLE-Anwendergruppe e.V. informiert kompetent über alle Oracle-Themen, setzt sich für die Interessen der Mitglieder ein und führen einen konstruktiv-kritischen Dialog mit Oracle.

Redaktion:

Sitz: DOAG Dienstleistungen GmbH, (Anschrift siehe oben)
Chefredakteur (ViSdP): Wolfgang Taschner
Kontakt: redaktion@doag.org

Redaktionsbeirat:

Ronny Kröhne, IBM-Architekt; Daniel van Ross, FIZ Karlsruhe; André Sept, InterFace AG; Jan Diller, Triestram und Partner

Titel, Gestaltung und Satz:

Alexander Kermas,
DOAG Dienstleistungen GmbH

Fotonachweis:

Titel: © Victor Correia/123RF
Foto S. 8 © chuyu/123RF
Foto S. 25 © studiostoks/123RF
Foto S. 34 © Tommaso Altamura/123RF
Foto S. 38 © Udo Schotten/123RF
Foto S. 44 © Vereshchagin Dmitry/123RF
Foto S. 49 © Štěpán Kápl/ fotolia
Foto S. 57 © Artisticco LLC/123RF
Foto S. 61 © Buchachon Petthanya/123RF

Anzeigen:

Simone Fischer, DOAG Dienstleistungen GmbH
Kontakt: anzeigen@doag.org

Druck:

adame Advertising and Media GmbH,
www.adame.de

Alle Rechte vorbehalten. Jegliche Vervielfältigung oder Weiterverbreitung in jedem Medium als Ganzes oder in Teilen bedarf der schriftlichen Zustimmung des Verlags. Die Informationen und Angaben in dieser Publikation wurden nach bestem Wissen und Gewissen recherchiert. Die Nutzung dieser Informationen und Angaben geschieht allein auf eigene Verantwortung. Eine Haftung für die Richtigkeit der Informationen und Angaben, insbesondere für die Anwendbarkeit im Einzelfall, wird nicht übernommen. Meinungen stellen die Ansichten der jeweiligen Autoren dar und geben nicht notwendigerweise die Ansicht der Herausgeber wieder.

Inserentenverzeichnis

cellent AG www.cellent.de	S. 29
DOAG e.V. www.doag.org	U 4
Java Forum Nord www.javaforumnord.de	S. 41
JUG Saxony Day www.jug-saxony-day.org	U 2



www.ijug.eu

**JETZT
ABO
BESTELLEN**

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG News und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo



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

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

Ja, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

Ja, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer



Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

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



Bis 30.09. Early Bird

2016 DOAG

Konferenz + Ausstellung
15. - 18. November in Nürnberg



Eventpartner:

2016.doag.org

