02-2016 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

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

Java ist Programmiersprache des Jahres

Sicherheit
- Social Login
- Single Sign-on

Richtig testen
Statische Code-Analyse

Richtig entwickeln
Lösung mit JavaFX und JVx

4 191978 304903 02

iJUG Verbund

DOAG BS Business Solutions

Community-Konferenz *Logistik + IT*

Die aktuellen Trends der Logistik-Branche
und deren IT-Herausforderungen

DOAG 2016 Logistik + IT

@ CeMAT Hannover | 2. Juni 2016

DOAG
Deutsche ORACLE-Anwendergruppe e.V.

http://logistik.doag.org

Wolfgang Taschner
Chefredakteur Java aktuell

Java ist Programmiersprache des Jahres

Die niederländische Firma Tiobe Software BV untersucht anhand von Treffern in 25 verschiedenen Suchmaschinen die Popularität von Programmiersprachen. Der sogenannte „Tiobe-Index" gibt das Ranking an. Mit einem großen Sprung von fast sechs Prozent auf 21,5 Prozent hat Java im Jahr 2015 die Sprache C (16,1 Prozent) an der Spitze abgelöst. Die beiden Sprachen liefern sich seit dem Jahr 2002 ein Kopf-an-Kopf-Rennen in diesem Index. Nachdem Java im Jahr 2011 bereits die Nase vorn hatte, war der Anteil von Java zwischenzeitlich wieder rückläufig und C lag vorne.

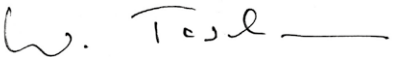
Deutlich dahinter liegen C++, C# und Python, keine dieser Sprachen kommt über sieben Prozent. Unter den Top 20 sind noch JavaScript (2,5 Prozent) mit einem Minus von 0,7 Prozent im Jahr 2015, während Ruby (2 Prozent) mit einem Plus von 0,9 Prozent und Groovy (1,2 Prozent, plus 1,1 Prozent) ebenfalls zu den Gewinnern zählen. Auf der Tiobe-Webseite *(siehe „http://www.tiobe.com")* ist genau beschrieben, wie der Index entsteht.

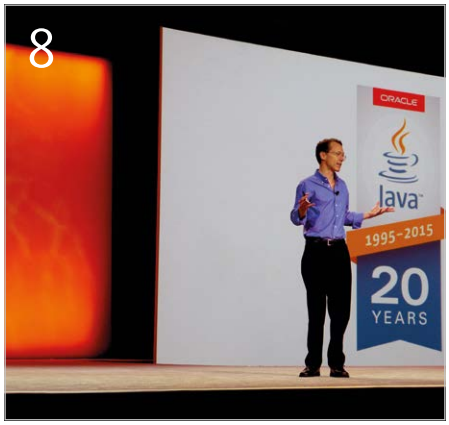
Man kann natürlich über den Erfolg von Java lange spekulieren, richtig ist jedoch, dass sowohl im Back-End-Bereich von Unternehmen als auch bei der mobilen Applikations-Entwicklung Java an erster Stelle steht. Sicher haben auch moderne Sprach-Features wie Lambda-Ausdrücke und Streams zum Erfolg beigetragen.

Auch wenn das Verfahren zur Findung des Tiobe Programming Community Index in der Entwicklergemeinde umstritten ist und der Einsatz einer Programmiersprache immer vom Projekt abhängt, kann man zumindest sagen, dass die Popularität von Java nach wie vor sehr hoch ist. Auch wenn man den Zahlenspielereien nicht unbedingt glaubt, ist es eine gewisse Selbstbestätigung für alle Programmiererinnen und Programmierer, dass sie mit Java im Trend liegen.

In diesem Sinne wünsche ich Ihnen weiterhin viele erfolgreiche Entwicklungsprojekte und hoffe, dass ich mit der Java aktuell auch einen kleinen Beitrag dazu liefern kann.

Ihr





Neues von der JavaOne

Die gängigsten Open-Source-Tools zur Code-Analyse im Praxis-Einsatz

Reaktive Anwendungen mit asynchronen, Event-getriebenen Architekturen gewinnen stark an Bedeutung

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 vierten Quartal 2015.

17. Oktober 2015

Akka, Scala, Docker, Holla!

Das Java-Ökosystem ist so groß und unübersichtlich geworden, dass es schon schwer fällt, mit den verbreiteteren Konzepten und Frameworks Schritt zu halten. Wenn man jedoch nicht abgehängt werden will, muss man immer wieder einen Blick über den Tellerrand werfen – wie Markus Eisele in seinem Blog-Eintrag zum Aktoren-basierten Framework Akka und Docker (Stichwort: „Microservices"). Den vorgeführten „Mini-HTTP-Server" in Scala beziehungsweise Java gibt es in verschiedenen Varianten und Code-Längen ja auch für andere Sprachen und Plattformen; aber es zeigt, dass Java in einem Bereich, in dem es immer mit Node.js und Co. verglichen und viel kritisiert wird, doch inzwischen einfache und elegante Lösungen zu bieten hat.

http://blog.eisele.net/2015/10/akka-http-in-the-cloud-maven-docker-for-java-developers.html

18. Oktober 201

Wahlen zum JCP Executive Committee

Die Nominierungen für das Kontrollgremium des Java Community Process sind abgeschlossen, die Wahlen finden allerdings erst vom 10. bis 23. November statt. Die Kandidaten für die „Ratified Seats" (Oracle schlägt vor): Credit Suisse, Ericsson, Fujitsu, HP, Intel, Red Hat, SouJava und IBM. Diese müssen also nur von der Community abgenickt werden – was jedoch auch schon mal schief gegangen ist. Die sechs Kandidaten, die um drei „Open Seats" konkurrieren, sind Eclipse, Jelastic, Jokl Technologies, London Java Community, Tomitribe und Twitter. JavaOne-Besucher können sich bei einem „Meet the Candidates"-Event ein eigenes Bild von den Kandidaten machen, es gibt aber noch eine Telekonferenz Anfang November, die auch aufgezeichnet wird.

https://blogs.oracle.com/jcp/entry/the_jcp_2015_executive_committee

21. Oktober 2015

Entwürfe für Java EE 8 und Servlet 4.0 zum Review freigegeben

Der erste Entwurf der Servlet-4.0-Spezifikation, also im Wesentlichen die Unterstützung von HTTP/2, ist auf jcp.org veröffentlicht. Wem das Spezifikations-Dokument etwas zu trocken ist, der kann sich auch in der deutlich bunteren Präsentation von Specification Lead Ed Burns die Neuerungen anschauen. Wem das auch noch nicht ausreicht, dem sei die Early Adopters' Area im JavaLand 2016 ans Herz gelegt, dort wird unter anderem auch Ed Burns aus dem Nähkästchen plaudern. Fast zeitgleich ist auch der erste Entwurf der „Umbrella"-Spezifikation, also EE 8 selber, veröffentlicht worden. Das öffentliche Review, an dem sich jede und jeder beteiligen kann, geht bis mindestens Ende Q1 2016.

http://www.slideshare.net/edburns/servlet-40-at-geekout-2015

23. Oktober 2015

Der Zustand von Java im Jahr 2015

Ein paar Details aus dem DZone „State of Java Survey 2015" kommen schon vor der offiziellen Veröffentlichung zur JavaOne ans Tageslicht: Java EE 7 ist inzwischen mit 39 Prozent die meistgenutzte Plattform, ganz knapp vor Spring 4 (Mehrfach-Nennungen waren möglich). Java EE 6 folgt nun ein Stück dahinter, wiederum knapp vor Spring 3. Schließt man ältere Versionen noch mit ein, hat Java EE insgesamt einen deutlichen Vorsprung, aber in den neuen Versionen bleibt es ein Kopf-an-Kopf-Rennen. Ich finde, das ist eher eine gute Sache: Konkurrenz sorgt für Innovation – und für Emotionen beim einen oder anderen JUG-Meeting. Beim Spezialthema „Persistence" sieht es anders aus. Der Standard JPA mit Hibernate liegt mit 64 Prozent ganz klar vorn. Dahinter folgt mit immerhin 38 Prozent das gute alte JDBC. Bei aller Nostalgie hoffe ich, dass das der großen Anzahl an Legacy-Systemen geschuldet ist, die fortlaufend Änderungen an der Persistenzschicht benötigen. Danach folgen Spring JDBC Template, Spring Data, (Nicht-Standard) Hibernate Native, MyBatis und ganz hinten JOOQ (mit immerhin noch 1,5 Prozent).

http://blog.cognitect.com/blog/2016/1/28/state-of-clojure-2015-survey-results

26. Oktober 2015

JavaOne 2015: Keynote

Nichts Wesentliches über Java SE, was nicht schon vorher bekannt gewesen wäre: Modularität ist das Haupt-Feature für SE 9. Chef-Architekt Reinhold will damit die „Jar Hell" loswerden, sieht aber für den oder die durchschnittliche Entwicklerin geringere Auswirkungen als durch die Lambda-Konstrukte in SE 8. Jenseits von SE 9 werden weiter die Pläne von „Project Valhalla" (unter anderem simple, aber hocheffiziente Value Types ohne Vererbung) und „Project Panama" (hochperformante Brücken zwischen Java- und Non-Java-APIs, etwa für C/C++ Function Calls) verfolgt. Ein paar vom Marketing getriebene Worte rund um IoT und die Cloud (insbesondere Oracles Java Cloud Services) durften natürlich nicht fehlen. Dass Intels IoT-Developer-Kit jetzt auch das OpenJDK unterstützt, war wohl noch eines der Highlights, ansonsten wurden hauptsächlich die Roadmaps von ME und EE bestätigt.

https://www.youtube.com/watch?v=LgkKc88nTwM

26. Oktober 2015

Oracle WebLogic Server 12.2.1 freigegeben

Die neueste Version von WebLogic ist auf Java EE 7 zertifiziert (Full Profile). Damit stehen nun sechs Application Server für Projekte auf der neuesten Enterprise Edition zur Wahl.

https://blogs.oracle.com/WebLogicServer/entry/announcing_oracle_weblogic_server_12

27. Oktober 2015

Adam Bien und Anatole Tresch Gewinner der „JCP 2015 Annual Awards"

Gleich zwei Mal ging in diesem Jahr ein Community-Process-Preis an deutschsprachige Vertreter: Adam Bien als „JCP Member/Participiant of the Year", und Anatole Tresch als „Outstanding Spec Lead". Letzterer ist übrigens auch in der Early Adopters' Area im JavaLand 2016 vertreten – unter anderem mit den Themen „CDI" und „Security", aber auch das geplante Configuration-API könnte ein wichtiges Thema sein. Als „Most Significant JSR" wurde „Units of Measurement API" ausgezeichnet, „Outstanding Adopt-a-JSR Participant" ist Raj Hegde von der JUG Chennai.

https://blogs.oracle.com/jcp/entry/jcp_2015_annual_award_winners

28. Oktober 2015

„Duke's Choice Award" – der Oscar für Java-Projekte

Wo wir gerade bei Thema „Preise" sind: Der „Duke's Choice Award" wird von einer Gruppe von Experten aus der Java-Community und von Oracle vergeben und geht an Projekte, die als besonders innovativ bewertet werden. Manche setzen Java in neuen, gesellschaftlich nutzbringenden Kontexten ein, wie die „Water Saver"-Applikation einer kalifornischen Schülerin. Andere sind im technischen Sinne innovativ. Daher ist es sinnvoll, sich die Liste einmal anzuschauen und auf das eine oder andere Projekt einen genaueren Blick zu werfen, darunter: AsciidocFX, ein JavaFX 8-basierter Editor mit Tools zur Erzeugung verschiedenster Medienformate aus AsciiDoc heraus (für die Älteren: „TeX on Steroids"); Byte Buddy, eine einfach zu nutzende Bibliothek für die Code-Generierung zur Laufzeit, ohne die Einschränkungen der in die Java Class Library integrierten Funktionalität, oder das Open-Source-KumuluzEE-Framework zur vereinfachten Konfiguration und Verteilung von Microservices auf Basis von Java-EE-APIs.

https://www.oracle.com/corporate/pressrelease/dukes-award-102815.html

24. November 2015

„Virtual Technology Summit"-Videos online

Virtual Technology Summits (VTS) sind im Grunde von Oracle organisierte Mini-Konferenzen zu jeweils einem bestimmten Thema. Zu Java gibt es eine eigene Serie von Summits. Gerade habe ich entdeckt, dass Videos von den Vorträgen verfügbar sind. Für diejenigen, die noch nicht genügend Java Content im Web finden … Scherz beiseite, einige interessante Sachen sind wirklich dabei.

https://community.oracle.com/groups/otn-vts-java-replay-library

25. November 2015

Wahlen zum JCP Executive Committee beendet

Die Wahlen zum JCP-Kontrollgremium sind zu Ende gegangen. Alle Kandidaten für „Ratified Seats" (siehe Eintrag vom 18. Oktober) wurden bestätigt, bei den frei gewählten Sitzen haben sich die Eclipse Foundation, die London Java Community und Twitter durchgesetzt. Herzlichen Glückwunsch – und wir hoffen auf vollen Einsatz für den Standard „Java".

https://jcp.org/en/whatsnew/elections

2. Dezember 2015

Java 9 soll weiter verschoben werden

Mark Reinhold, Chef-Architekt der Java Plattform, hat gestern in einer Mail an den OpenJDK-Verteiler vorgeschlagen, den Zeitplan für das JDK 9 um sechs Monate zu verlängern. „Feature Complete" würde dann Ende Mai 2016 erreicht, offizielles Release-Datum wäre Ende März 2017. Die Ankündigung kommt spät, da „Feature Complete" nach dem aktuellen Plan schon am 10. Dezember 2015 erreicht werden sollte. In der Begründung heißt es, dass nicht nur das zentrale Projekt „Jigsaw" (Laubsäge) bislang noch keinen Spezifikations-Entwurf geliefert hat, sondern dass in den vergangenen Monaten Menge und Qualität an Feedback zu den Plänen deutlich zugenommen hat. Ich vermute, das ist ein eher gutes Zeichen. Daher sollen laut Reinhold die bisherigen und potenziellen Feedback-Geber, insbesondere Hersteller von Build Tools und IDEs, noch mehr Zeit erhalten. Ob das jetzt vorgeschoben ist oder nicht, Reinholds Vorschlag dürfte kaum abgelehnt werden, also werden wir uns noch ein halbes Jahr länger gedulden müssen.

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

6. Dezember 2015

Änderung der Versionsnummerierung für Java 9

Für Java 9 soll die Versionsnummerierung angepasst werden. Klingt eigentlich harmlos, aber gerade diese kleinen Änderungen haben in der Vergangenheit oft für Aufregung gesorgt, weil entweder Verwirrung aufkam oder sich einzelne Tools oder Applikationen aus plausiblen bis obskuren Gründen auf eine bestimmte Versionsnummerierung verlassen haben. Diesmal wird es frühzeitig angekündigt, dann sollte nichts schiefgehen. Ein bisschen gewöhnungsbedürftig ist die neue Nummerierung schon, weil die zweite Stelle das Minor Release (funktionale Änderungen) und die dritte Stelle das Patch Level „Critical Patch Update" (CPU) angeben, und zwar jeweils fortlaufend bezogen auf das Gesamt-Release. Eine möglich Nummernfolge wäre 9, 9.0.1, 9.1.1, 9.1.2, 9.1.3, 9.2.3, … Das Patch Level wird nach einem Minor Release nicht neu gestartet.

https://blogs.oracle.com/java-platform-group/entry/a_new_jdk_9_version

8. Dezember 2015

Greenfoot und Greeps

Devoxx for Kids, JavaLand4Kids, neu entstehende Coding-Schulen … Das Thema „Programmieren für Kinder und Jugendliche" wird immer populärer. Eine gute Sache, schließlich lebt auch Deutschland zum guten Teil von den Fähigkeiten seiner Ingenieure und Naturwissenschaftler. Da macht es Sinn, auch den Kleinen schon Grundkonzepte, aber insbesondere ein bisschen Spaß an Informatik zu vermitteln. Ich habe gerade einen Blog-Eintrag von Michael Kölling von der Universi-

ty of Kent gelesen, in dem er das „Greeps"-Projekt für Greenfoot beschreibt. Greenfoot ist eine Art IDE zum Lehren und Lernen von Java und Michael Kölling ist eine der Personen dahinter. Die Greeps sind kleine knuffige Aliens, die gemeinsam in einer fremden Welt Tomaten sammeln müssen. Das Projekt ist dazu angelegt, für einen Programmier-Wettbewerb genutzt zu werden, und nicht nur Java-Kenntnisse, sondern auch generelle Lösungsstrategien zu vermitteln. Laut Kölling is es für Jugendliche ab etwa dreizehn Jahren geeignet. Wer also ein entsprechendes Programmier-Event plant, sollte ruhig mal einen Blick auf die Greeps werden.

http://blogs.kent.ac.uk/mik/

10. Dezember 2015

Siwpas Application Server für Java EE 6 „Web Profile" zertifiziert

Siwpas ist nicht ganz neu (obwohl ich zugegebenermaßen bis heute nichts davon gehört hatte), aber ganz frisch als kompatibel mit dem etwas älteren „Java EE 6 Web Profile" zertifiziert. Der Name steht für „Simple Web Profile Application Server". Die Zusammenarbeit mit dem OW2-Konsortium, das bereits seinen eigenen Server JOnAS durch die Zertifizierung gebracht hat und vor allem eine TCK-Lizenz besitzt, hat es möglich gemacht. Siwpas ist wie JOnAS Open Source und auf forge.ow2.org verfügbar.

https://blogs.oracle.com/theaquarium/entry/from_2_to_8_java

10. Dezember 2015

Planänderung für Java 9 angenommen

Jetzt ist es amtlich. Nachdem innerhalb der festgelegten Frist kein Einspruch eingelegt wurde, ist die von Mark Reinhold vorgeschlagene Verschiebung des JDK 9 um sechs Monate angenommen.

http://mail.openjdk.java.net/pipermail/jdk9-dev/2015-December/003149.html

7. Januar 2016

Optional statt „null"

Die Diskussion darüber, ob es eine gute Idee war, das Konzept „null" in den Java-Sprachstandard aufzunehmen, wird sicher nie enden. Aber mit Java 8 gibt es ja eine Alternative, die aber vermutlich noch nicht so bekannt ist: „java.util.Optional<T>". Es ist ein Wrapper um Objekte, der nie „null" enthält beziehungsweise zurückgibt, sondern entweder ein Objekt wrappt oder gar nichts, also leer bleibt. Mit seinen Methoden wie „ifPresent(Consumer<? super T> consumer)" kann Optional das Programmierer-Leben sehr erleichtern, etwa bei der Stream-Verarbeitung. Wer Beispiele sehen möchte, findet sie unter anderem in einem Blog-Eintrag von José Paumard.

https://community.oracle.com/docs/DOC-991686.

8. Januar 2016

SnopEE – eine Microservice Registry

Auf der JavaOne hat Ivar Grimstad (auch er wird im JavaLand 2016 in der Early Adopters' Area mitmachen) sein Open-Source-Projekt SnoopEE vorgestellt, eine Registry inklusive Lookup/Discovery für Microservices. Sein Talk ist jetzt auf Video verfügbar und präsentiert nicht nur das Projekt an sich, sondern auch die Motivation dahinter.

https://blogs.oracle.com/theaquarium/entry/meet_snoopee_a_discovery_service

13. Januar 2016

Hypermedia/HATEOAS Support in JAX-RS 2/Java EE 7

Für diejenigen, die sich mit HATEOAS beschäftigen wollen (also mit REST Level 3 APIs, bei denen anstelle vorgegebener Navigationspfade durch die Ressourcen der Client diese zur Laufzeit dynamisch entdecken kann), gibt es eine sehr gute Einleitung und daraus verlinkt eine detaillierte Beschreibung, wie gut JAX-RS diesen Stil unterstützt.

https://blogs.oracle.com/theaquarium/entry/hypermedia_hateoas_support_in_jax

20. Januar 2016

JSR Updates – insbesondere Java EE 8

Ende 2015 haben zahlreiche für Java EE 8 angepeilte JSRs die „Early Draft Review"-Phase erreicht. Ein paar wurden hier schon erwähnt, eine Übersicht enthält auch andere Spezifikationen, unter anderem sind Money & Currency und SIP Servlet inzwischen „final".

https://blogs.oracle.com/jcp/entry/jsr_updates_end_of_2015

27. Januar 2016

Java-Plug-in mit JDK 9 „deprecated"

Das Java-Browser-Plug-in soll mit JDK 9 „deprecated" werden. Das heißt also noch nicht, dass es ganz verschwindet – das soll mit einem späteren Release erfolgen, wobei sich Oracle dazu noch nicht konkret geäußert hat. Wer dann weiterhin Java aus dem Browser heraus anbieten möchte, muss auf Java Web Start zurückgreifen.

https://blogs.oracle.com/java-platform-group/entry/moving_to_a_plugin_free

29. Januar 2016

„@Lock"-Annotation für CDI-Beans

Wie man das CDI-Interceptor-Konzept nutzen kann, um parallele Zugriffe zu synchronisieren und mal schnell die Analogie zur EJB-3-„@Lock"-Annotation für CDI-Beans generell zu realisieren, zeigt Stephan Knitelius in seinem Blog-Eintrag „Concurrency Control for CDI". Mit einer selbst definierten „@Lock"-Annotation und einem Interceptor, der diese mithilfe eines „ReentrantReadWriteLocks" implementiert, ist das Ganze mit wenigen Zeilen Code erledigt. Anhänger des Reactive Programming werden jetzt vielleicht fragen „Wozu?", aber unabhängig vom Streit um das beste Architektur-Konzept (genau, das gibt es nicht) zeigt das Beispiel sehr schön die Flexibilität von CDI.

https://blogs.oracle.com/theaquarium/entry/ejb3_lock_annotation_for_cdi

Andreas Badelt
Leiter der DOAG SIG Java

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

JavaOne 2015: Java ist weiter auf einem guten Kurs

Wolfgang Taschner, Chefredakteur Java aktuell

Die Keynote der JavaOne in San Francisco steht unter dem Motto „20 Jahre Java" und greift technologische Meilensteine der Java-Zeitgeschichte auf. Ohne große Überraschungen werden die Roadmaps für Java SE, Java ME und Java EE aufgezeigt sowie planmäßig die gesteckten Technologie-Ziele eingehalten. Eines der größten Highlights kommt eher nur am Rande zur Sprache: Der WebLogic Server 12.2.1 steht als Java-EE-7-Implementierung ab sofort zum Download bereit.

Georges Saab, Oracle Vice President für die Java Platform Group, startet den Reigen der Keynote-Speaker mit einem Blick auf Java SE. Beim OpenJDK gibt es ein Wachstum von 136 Prozent und rund ein Drittel der neuen OpenJDK-Projekte sind extern orientiert. Dieses Wachstum belegt, dass das Commitment von Oracle nicht nachlässt, sondern Früchte trägt.

Im Anschluss daran stellt Shaun Smith, Oracle Product Manager, den neuen Java SE Cloud Service vor. Unter dem Motto „Build, Zip, Deploy" ist dieser die einfachste Art, um Java-Anwendungen beispielsweise mit Tomcat gebündelt in der Cloud zu nutzen. Bei der Implementierung des Cloud-Service wird zur Virtualisierung Docker benutzt und zum Tooling Flight Recorder for Production Use angeboten. Der Services ist skalierbar und es gibt die Funktion „One Click Java Version Update". Man muss sich also nicht um die Pflege der Plattform und die Security-Fixes kümmern.

Ein Blick in die Zukunft

Als Highlight stellt Mark Reinhold, Chief Architect Java Platform Group, die wesentlichen Dinge im kommenden JDK 9 vor (Projekt „Jigsaw"). Es sind die Eliminierung von Classpath-Hell und die Modularisierung der Java-Plattform. Ein Vorteil dabei sind Java-Profile (Subsets der Java-Plattform) für unterschiedliche Device-Typen. Er zeigt dazu am Beispiel einer Hadoop-Anwendung, wie sich der Classpath-Hell in der Realität manifestiert und wie die Einführung eines Modul-Systems diese Probleme löst. Durch die Modularisierung der Java-Plattform lassen sich auch passende Profile (Subsets davon) für unterschiedliche Device-Typen definieren.

Das nächste Highlight kommt von Brian Goetz, Chief Language Architect Java Platform Group, der andeutet, was nach Java 9 kommen soll: Das Valhalla Project bringt Specialized Generics sowie Value Types und im Panama Project wird es ein Foreign-Function-Interface, Data Layout Control sowie Array 2.0 geben.

Die Keynote von Anil Gaur, Oracle Vice President Software Development, ist nicht besonders technisch, sondern beleuchtet mehr die Management-Sicht. Im Vordergrund steht die Ausbalancierung von Specification Development, Developer Adoption und Vendor Adoption. Statistiken belegen, dass Oracle auf dem richtigen Weg ist: Java EE 7 ist heute die bevorzugte Java-EE-Platt-

Abbildung 1: Mark Reinhold, Chief Architect Java Platform Group, bei seiner Keynote

form auf dem Markt mit hohen Wachstumsraten.

Eher unspektakulär verkündet Anil Gaur die größte Neuigkeit: Der WebLogic Server 12.2.1 steht ab sofort als Java-EE-7-Implementierung zum Download bereit. Einige Mitbewerber wie IBM und Red Hat haben ebenfalls Java-EE-7-Implementierungen herausgebracht. Zu Java EE 8 nennt er den Status und eine Roadmap: GlassFish 5 wird die Java-EE-8-Referenz-Implementierung sein. Erste Builds sind bereits zum Download bereit.

Sun-Gründer Scott McNealy bringt die Zuhörer mit seinen „Top Ten Nightmares" sehr humorvoll und auflockernd in fast vergessene Zeiten zurück.

Michael Greene, Vice President and General Manager, System Technologies and Optimization, Software and Services Group bei Intel, demonstriert sein Hauptanliegen, eine gute Java-Performance auf der gesamten Intel-Plattform. Neu ist die Unterstützung für Java ME auf Intel Edge Devices sowie der „Intel IoT Developer Kit" für Java. Intel bereitet sich bereits heute auf die Optimierungen kommender Java-Technologien auf der kompletten Intel-Plattform vor.

Wolfgang Taschner
redaktion@ijug.eu

iJUG fordert von Oracle ein klares Bekenntnis zu JavaFX

Für den iJUG mehren sich die Anzeichen, dass JavaFX keine große Zukunft mehr bei Oracle hat. So wurde der Support für die Version 8u33 von JavaFX auf der ARM Embedded Platform eingestellt. Zudem hat Oracle den offiziellen Download des JavaFX Scene Builders beendet. Auch über die Roadmap von JavaFX ist nichts bekannt.

Die Webseite mit der Roadmap von JavaFX ist seit Monaten nicht mehr erreichbar. Das gibt Anlass zur Sorge im iJUG, dass JavaFX auf Dauer nicht mehr unterstützt wird. Viele Java-Entwickler sind verunsichert, ob sie bei ihren Projekten weiterhin auf JavaFX setzen sollen.

Die Java-Community bedauert die Einstellung des JavaFX-Supports auf der ARM Embedded Platform (zählt weltweit zu den meistverbreiteten Mikroprozessoren), weil damit sehr schön die Anwendungen für kleine Touchscreens an Embedded Devices bedient werden konnten. Auf Nachfrage nach dem Grund für diese Maßnahme teilte Oracle mit, dass es inzwischen zu viele unterschiedliche Devices gibt, die nicht alle unterstützt werden können.

Auch der JavaFX Scene Builder, ein visuelles Layout-Tool für JavaFX-Anwendungen, wird von Oracle nicht mehr zum Download angeboten. Als Alternative steht zwar von Gluon eine entsprechende Version bereit. Doch für viele Entwickler stellte gerade der Scene Builder eine Möglichkeit da, die Wartbarkeit ihrer Oberfläche zu gewährleisten. Der Scene Builder ist zwar nur ein Tool im Umfeld von JavaFX, doch es ist ärgerlich, dass es nicht mehr von Oracle weiterentwickelt wird.

„In den vergangenen Jahren hat Oracle eindeutig kommuniziert, dass man im Embedded-Umfeld mit Java punkten möchte", sagt Tobias Frech, Vorstand des iJUG. „Jetzt stellt sich natürlich die Frage, ob die Abkündigung von JavaFX auf der ARM Embedded Platform das Resultat anderer strategischer Änderungen bei Oracle bezüglich Java ist."

Der iJUG hat die Problematik in einem offenen Brief an Oracle kommuniziert und eine Antwort darauf erhalten:

Thanks for your letter. We answered the ARM specific questions in January 2015, details see here *„http://mail.openjdk.java.net/pipermail/openjfx-dev/2015-January/016570.html"*. We recommend those who are interested in JavaFX ports to participate in the OpenJFX project. A prime example of this working as intended can be found here *„http://gluonhq.com/gluon-javafx-embedded-arm-sdk-8-60-6-released"*. We are also aware of some groups interested in building their own variants, which are still in the early stages, but you can see one here *„http://www.freelists.org/archive/teamfx"*. We also have commercial partners who are capable and willing to support JavaFX ports as needed, we'd be happy to make introductions. Our strategy for SceneBuilder is for it to be integrated into tools and other development frameworks and thus it was released under a BSD license. An example of that working as intended can be found here *„http://gluonhq.com/open-source/scene-builder"*.

Regarding the roadmap, as of 7u6 JavaFX is included in the Oracle JDK. You can find out what items are targeted for JDK 9 via the targeted JEPs listed here *„http://openjdk.java.net/projects/jdk9"*.

We continue to recommend JavaFX for application developers building rich clients, see *„http://www.oracle.com/technetwork/java/javase/8u40-relnotes-2389089.html"*. As of 8u40, JavaFX added support for assistive technologies making it therefore the recommended UI framework for accessibility applications.

Die Java-Community beschäftigt sich im Rahmen der JavaLand 2016 in einem Workshop mit diesem Thema und wird in der nächsten Java aktuell die Ergebnisse präsentieren.

Development, Deployment und Management mit dem Oracle-Java-Cloud-Service

Marcus Schröder, ORACLE Deutschland B.V. & Co. KG

Die Überschrift sollte beim Lesen stutzig machen! Hier werden zwei Aufgaben vermischt, die man normalerweise getrennt voneinander betrachtet. Das Thema „Development" wird, wie der Name schon sagt, von den Entwicklern getragen. „Deployment" und „Management" sind normalerweise Aufgaben des IT-Betriebs. Doch wie können diese Bereiche zusammenkommen? Der Artikel geht dieser Frage nach und befasst sich mit dem Thema „DevOps" sowie dem Oracle-Java-Cloud-Service.

Um DevOps zu beschreiben, muss man an dieser Stelle weiter ausholen: In der Vergangenheit wurden Applikationen oft in einer Silo-Architektur betrieben, die Applikationen haben/hatten also oft eine eigene Infrastruktur und eigene Plattformen. Im Zuge der Kostenreduktion wurden/werden aus Gründen der Standardisierung und Konsolidierung Infrastrukturen und Plattformen der Applikationen zusammengefasst. Dies ermöglicht ein wesentlich effektiveres und kostengünstigeres Betreiben der Applikationsumgebungen.

Aus diesen Architekturen resultieren wiederum organisatorische Silos. Zum Beispiel sind in den meisten größeren Umgebungen die Organisationen nach Infrastruktur, Datenbank, Applikationsserver etc. aufgeteilt. Wenn die Applikationen ohne Probleme laufen, ist dies von (Kosten-)Vorteil, kommt es jedoch zu Störungen der Service-Level, ist die Silo-IT-Organisation bei der Ursachen-Analyse oft problematisch. Grund dafür sind vorher festgelegte Prozesse mit definierten Schnittstellen, die einem kreativen Lösungsansatz im Wege stehen.

Ein Workaround und das Wissen des Application-Owners sind oft die Hilfe, diese haben jedoch manchmal nicht die technische Tiefe, um eine effektive Vermittlung zwischen den IT-Silos zu ermöglichen. Eine mögliche organisatorische Lösung der IT-Silo-Problematik ist DevOps. Hier sei ausdrücklich darauf hingewiesen, dass es sich bei DevOps nicht um ein oder mehrere Produkte handelt, sondern um ein Paradigma beziehungsweise eine Kultur.

Der Haupttreiber der Entwicklung ist die Erhöhung der Funktionalität der jeweiligen

Applikationen. Der Betrieb benötigt Stabilität, Performance und Verfügbarkeit. DevOps soll die Anforderungen dieser beiden Gruppen vereinen. Zusammen ergibt dies bestimmte Charakteristiken von DevOps, zu diesen gehört die Automatisierung.

Sie betrifft in erster Line die Bereitstellung der Applikation auf den dafür vorgesehenen Plattformen beziehungsweise Infrastrukturen. In vielen Unternehmen wird dies durch „Continuous Delivery" und „Continuous Deployment" sichergestellt. Hierbei handelt es sich um automatische Bereitstellung und Testen in einer produktionsnahen Umgebung sowie anschließendes Bereitstellen in Produktion. Mittlerweile ist der Automationsgrad in vielen Unternehmen so weit fortgeschritten, dass auch die Bereitstellung der Plattform und teilweise sogar der Infrastruktur ohne manuelle Schritte erfolgt.

Ein weiterer Punkt ist die standardisierte Infrastruktur. Diese Anforderung stammt primär aus dem Betrieb. Standards unterstützen die Schaffung einer stabilen und sicheren Laufzeitumgebung für Applikationen. Die Entwicklung auf einheitlichen Konfigurationen und Versionen von Plattformenkomponenten minimieren potenzielle Fehlerquellen.

Die Verwendung von Open-Source-Produkten basiert auf den Anforderungen von Startup-Communities, ist aber nicht zwingend notwendig. Auf Entwicklungsseite existiert eine Reihe von Open Source-(ähnlichen)-Produkten, die als Standard-Tools (wie GIT, Jenkins, Bugzilla etc.) für den Entwicklungsprozess unverzichtbar sind. Die agile Software-Entwicklung ermöglicht durch agile Methoden, den gesamten Software-Entwicklungsprozess oder Teilbereiche durch Reduzierung des bürokratischen Aufwands, der Regeln und Iteration beweglicher und somit schneller zu gestalten.

Die offene Kultur zwischen Entwicklung und Betrieb ist einer der zentralen Pfeiler von DevOps. Respekt, Dialoge, Diskussion und faires Release-Management stehen bei der offenen Kultur an erster Stelle. Typisches Fehlverhalten und damit entgegen der DevOps-Kultur sind zum Beispiel gegenseitige Schuldzuweisungen, „über den Zaun schmeißen" von Deployments oder Information-Hiding.

Positive Merkmale von DevOps sind gegenseitiger Respekt, gemeinsame Teilnahme an Diskussionen, Systemzugriffe für das Development, Verantwortlichkeit trotz Produktion und gemeinsame Eskalationspläne. Zusätzlich zur DevOps-Kultur gibt es eine Reihe von Technologien, die die Zusammenarbeit von Entwicklung und Betrieb begünstigen. Mögliche Technologien sind: Infrastruktur-Konfiguration als Code, Versionskontrolle, One-Step Build/Deploy und keine Verwendung von Hotfixes.

In dem Punkt „Infrastruktur-Konfiguration als Code" geht es darum, applikationsspezifische Konfigurationen direkt in den Code zu implementieren. Der Vorteil dieser Vorgehensweise ist die Standardisierung der Infrastruktur/Plattform, die bei der Bereitstellung applikationsspezifisch konfiguriert wird.

Die Versionskontrolle ist ein System für Code-Build und Artefakte. Bei jeder Änderung sollte ein automatischer Build mit Verifikationstests durchlaufen werden. Eine weitere Möglichkeit in der Versionskontrolle ist das Einbauen von Schaltern in den Code, mit denen neue Funktionen deaktiviert werden können.

Mit „One-Step Build/Deploy" ist die höchstmögliche Automation des Build/Deploy-Prozesses gemeint. Der neueste Build sollte mit möglichst geringem Aufwand auf einer produktionsnahen Umgebung beziehungsweise in der Produktion bereitgestellt werden. Die Verwendung eines Scheduler ermöglicht es, einen gemeinsamen Code-Stand in eine Umgebung einzuspielen. Schlägt die Bereitstellung fehl, sollte eine Benachrichtigung versendet werden.

„Hot-Fixes" sind ein sehr sensibles Thema, da es in vielen (Produktions-)Umgebungen bei Problemen durchaus üblich ist, einen Hot-Fix aus der Entwicklung einzuspielen. Diese Fixes sind häufig unzureichend getestet und erhöhen die Instabilität der gesamten Umgebung. Daher ist es besser, bei Problemen ein Re-Deployment zu machen, anstatt einen Hot-Fix einzuspielen.

Die aufkommende Frage an dieser Stelle ist: „Wie kann Oracle bei der Umsetzung von DevOps in meinem Unternehmen helfen?" Die Antwort: Oracle bietet eine Reihe von Technologien, die bei der Umsetzung helfen können; die Verantwortung und Durchsetzung obliegt weiterhin dem Kunden. Dies

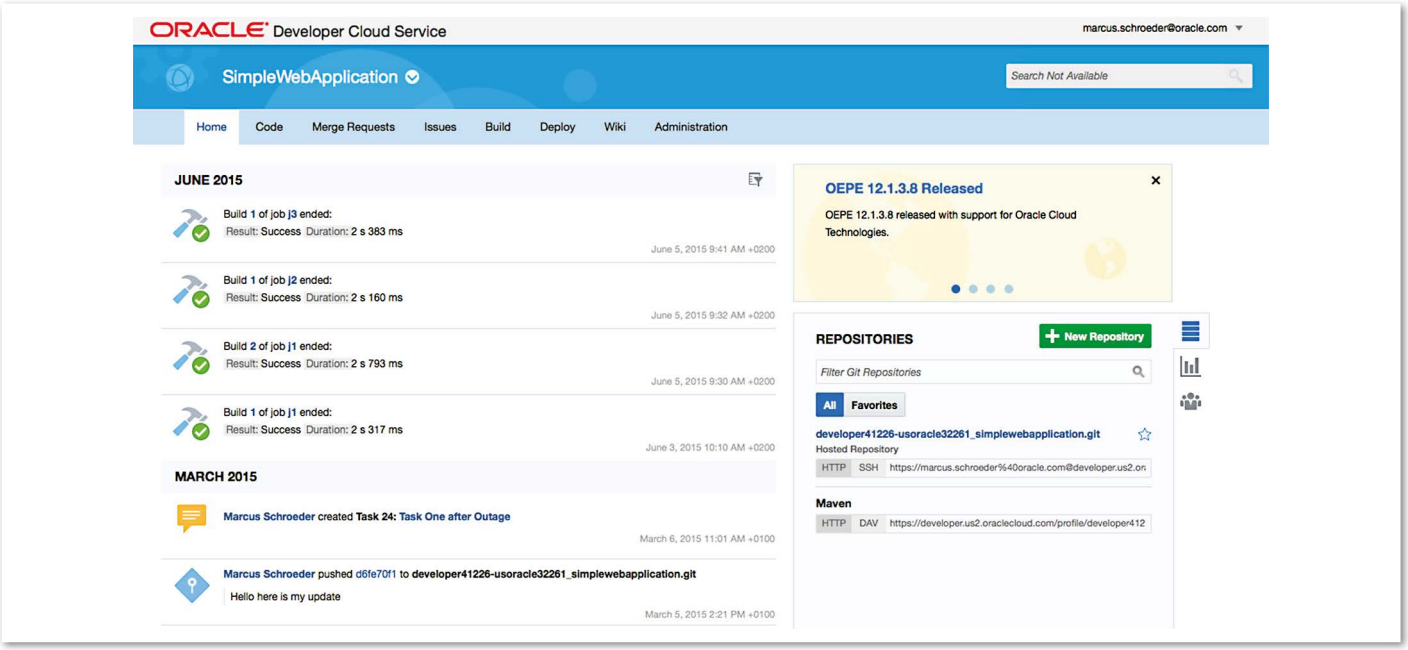


Abbildung 1: Projekt-Übersicht Java-Developer-Cloud-Service

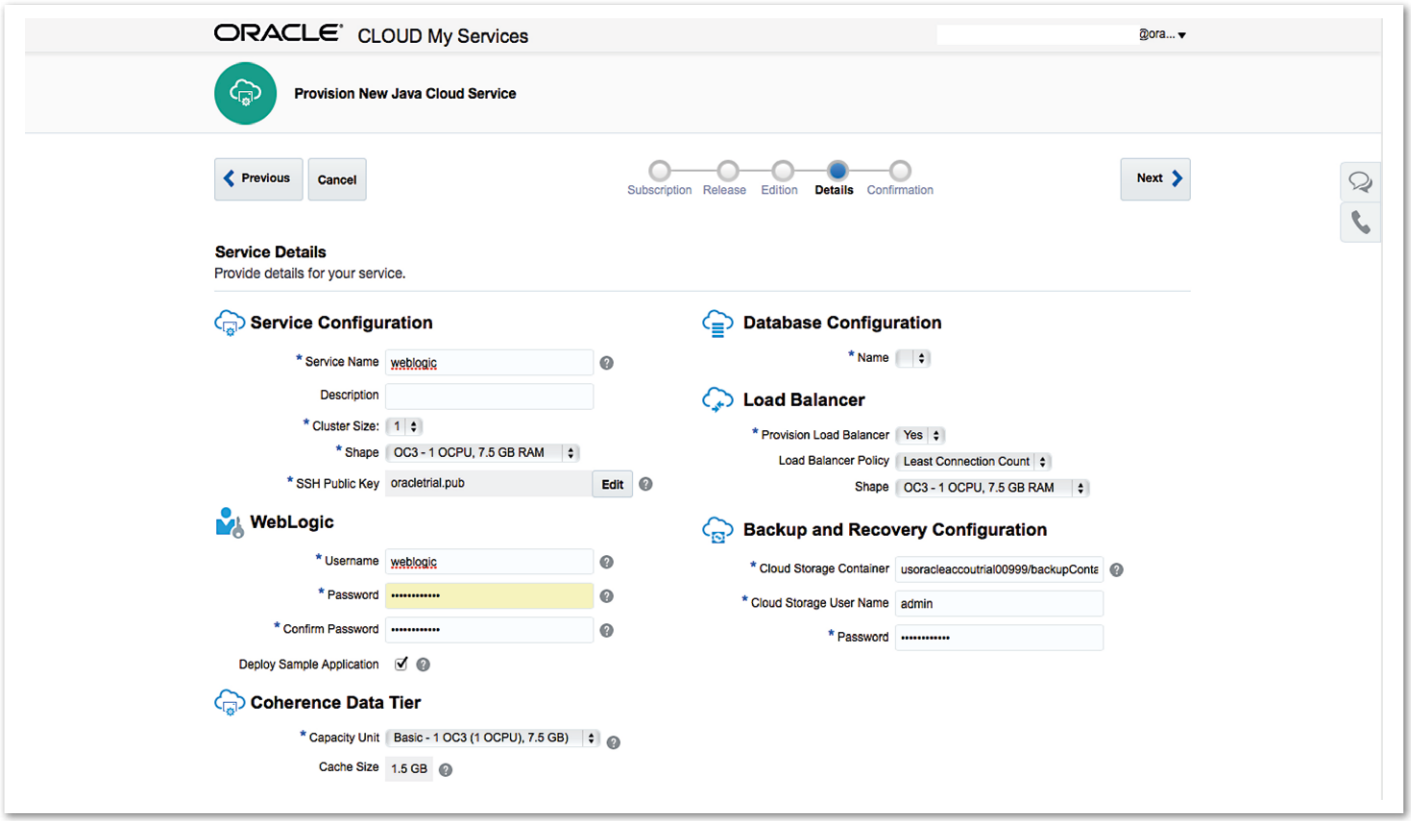


Abbildung 2: Eingabe der Java-Cloud-Service-Details

gilt besonders der Umsetzung der kulturellen Richtlinien von DevOps. In den technologischen Bereichen Infrastruktur als Code, Versionskontrolle, One-Step-Build und Vermeidung von Hot-Fixes kann Oracle mit Software und Services helfen.

Es gibt eine Reihe von On-Premise- und Oracle-Cloud-Komponenten, die in den Bereichen Lifecycle Management, Laufzeitoptimierung, Provisioning und Deployment unterstützten. Im Nachfolgenden wird auf die Oracle-Public-Cloud-Plattform-Services und das zentrale Management-System eingegangen.

Die Oracle Cloud

In der Public Cloud von Oracle gibt es eine Reihe von Plattform-Services für verteilte Entwicklung, Tests und Bereitstellung von Applikationen. Der Markenname für die Oracle Public Cloud ist „Oracle Cloud". Wer denkt, Oracle sei neu im Cloud-Geschäft, liegt an dieser Stelle falsch. Oracle hat seit dem Jahr 2012 ein umfangreiches „Software as a Service"-Angebot und unter anderem auch zwei Plattform-Services. Die drei Services „Java Cloud Service SaaS Extension", „Oracle Developer Cloud Service" und „Database Cloud Service SaaS Extension" bieten jedoch keine Möglichkeit, auf das Betriebssystem zuzugreifen, sie sind also als reine Plattform-Services konzipiert. Zugriff auf Betriebssystem-Ebene ist in modernen, komplexen Entwicklungsprojekten unumgänglich. Daher wurden im Jahr 2014 Oracle-Java-Cloud-Service und Database-Cloud-Service (ohne SaaS-Extension) vorgestellt.

Über die beiden genannten Services besitzt der Administrator volle Root-Rechte. Bei der Erstellung der Services wird ein Private-/Public-Key-Pair erzeugt, dadurch kann nur der Besitzer des Private-Key auf die Umgebung zugreifen. Der Oracle-Developer-Cloud-Service wurde seit dem Jahr 2012 permanent weiterentwickelt und steht ebenfalls mit den beiden neuen Services zur Verfügung.

Die vorgestellten Services werden im Oracle-Rechenzentrum in Detroit (Nord-Amerika) und Amsterdam (EMEA) gehostet. Die nachfolgenden Ausführungen beschäftigen sich primär mit der DevOps-Sicht auf die Cloud-Services. Java-Cloud-Service, Datenbank-Service und Developer-Service können nicht nur für Entwicklung und Tests verwendet werden, sondern auch für die Überführung in die Produktion.

Der Oracle-Developer-Cloud-Service ist im eigentlichen Sinne kein Plattform-Service. Er stellt eine automatische Bereitstellungsplattform dar, die den vollständigen Java-Entwicklungs-Lifecycle unterstützt, und ist als SaaS-Service implementiert. Das bedeutet, der Benutzer hat keinen Aufwand bezüglich Plattform-Maintenance. Der Developer-Cloud-Service ist im Java-Cloud-Service kostenfrei enthalten und bietet viele Tools für die Bereiche „Source Control Management", „Issue Tracking", „Hudson Continuous Integration" und „Collaboration (Wiki)".

Alle diese Komponenten sind ohne aufwändiges Setup sofort verfügbar. Es können eigene oder fremde GIT-Repositories eingebunden werden. Die Benutzerverwaltung erfolgt innerhalb der Cloud-Lösung, ideal für das gemeinsame Arbeiten von verteilten Teams. Das Deployment wird entweder manuell oder mittels Scheduler-Job direkt in den Java-Cloud-Service oder in ein Remote-Target durchgeführt. Aufgaben wie Merge-Request, Issue-Tracking oder Collaboration können zentral von einem Projekt-Mitarbeiter erstellt und an den jeweiligen Entwickler vergeben werden *(siehe Abbildung 1)*.

Beim Oracle-Java-Cloud-Service handelt es sich um ein auf Java basierendes PaaS-Angebot in der Oracle-Cloud. Es enthält das automatische Bereitstellen von Applikationsserver, Infrastruktur, Storage und Datenbanken. Damit wird ein manuelles Setup und Bereitstellen der Infrastruktur und

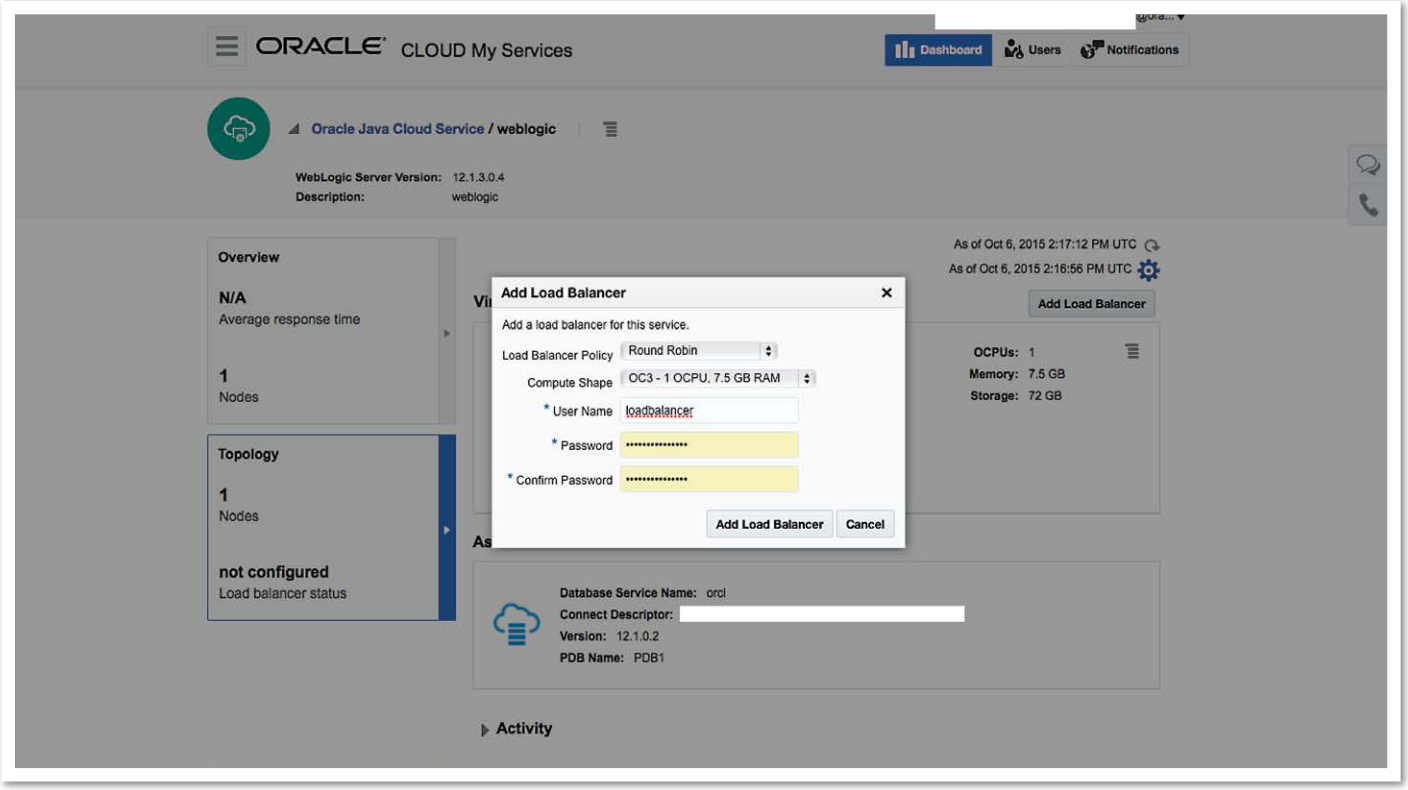


Abbildung 3: Einrichten eines Loadbalancer für den WebLogic-Server

Plattform überflüssig. Der gesamte Prozess – vom Beantragen einer Plattform bis zur Nutzung – dauert Minuten statt Wochen. Oracle unterstützt den gesamten Lifecycle-Support mit dem sogenannten „Cloud-Tooling". Backup und Recovery, Patching und Scale-out/in erfolgen auf Knopfdruck in der Cloud und müssen nicht zusätzlich aufgesetzt werden. „Oracle PaaS für Entwickler" besteht aus dem Java-Cloud-Service oder für Datenbank-Entwicklung aus dem Datenbank-Cloud-Service.

Die Anforderung der Umgebung erfolgt in einem Self-Service-Portal, das unter der vollständigen Kontrolle des Cloud-Administrators steht. Er gibt an, welche Version, Größe, Funktionen etc. in dem neuen Service enthalten sein sollen, und triggert die Erstellung. Der Plattform-Service wird inklusive Infrastruktur bereitgestellt; Netzwerk, Storage, Compute-Knoten und Betriebssystem werden also automatisch mit provisioniert. Die Bereitstellung erfolgt nach Oracle-eigenen Best-Practice-Standards.

Alle wichtigen Provisioning-Aktionen können auch durch das integrierte REST-API durchgeführt werden. Damit lassen sich die Oracle-Cloud-Operationen in ein bestehendes Unternehmensportal beziehungsweise eine Cloud-Broker-Umgebung integrieren *(siehe Abbildung 2)*.

Zentrales Management

Für den Betrieb bietet der Oracle Enterprise Manager Cloud Control (EM12c) ein zentrales Management der Cloud- und On-Premise-Umgebungen. EM12c wird on-Premise im eigenen RZ betrieben und verbindet sich über SSH mit den auf der Oracle-Cloud installierten Agenten. Diese liefern aktuelle und historische Metriken der überwachten Umgebungen, übernehmen remote Maintenance-Aufgaben und benachrichtigen im Falle einer Überschreitung eines festgelegten Schwellenwerts.

EM12c unterstützt nicht nur den Betrieb in seinen täglichen Aufgaben, sondern bietet umfangreiche Funktionen im Bereich „Diagnose". Dies ermöglicht das Auffinden von Performance-Engpässen, unabhängig davon, ob diese im Applikationsserver oder in der Datenbank ihre Ursache haben. Die Lifecycle-Unterstützung bietet von Patching über Deployment bis zum On-Premise-Provisioning alle benötigten Automatisierungen *(siehe Abbildung 3)*.

Fazit

DevOps ist in der modernen IT eine der wenigen Möglichkeiten, die Vorteile eines konsolidierten Betriebs und die daraus resultierenden Schnittstellen einer Silo-orientierten IT zu optimieren. In der heutigen modernen Cloud-Welt, in der die Umgebungen oft verteilt und sehr komplex werden können, ist ein Umdenken beider Welten (Entwicklung und Betrieb) nötig. Eine umsetzbare Lösung mit den Oracle-PaaS-Komponenten und dem Developer-Cloud-Service garantiert schnelle Ergebnisse und hält den Aufwand in Grenzen.

Marcus Schröder
marcus.schroeder@oracle.com

Marcus Schröder ist als Systemberater der ORACLE Deutschland B.V. & Co. KG zuständig für Cloud, Betriebsthemen und Engineered Systems. Er verfügt über langjährige Kunden-Erfahrung in den Bereichen „Cloud-Computing-Architekturen", „Middleware" und „allgemeine Betriebsthemen". Marcus Schröder studierte technische Informatik in Kiel mit Studienschwerpunkt „Systemtechnik" und arbeitet seit 16 Jahren für Oracle.

Leuchtfeuer in Innenräumen: Micro-location-based Services mit Beacons

Constantin Mathe und Bernd Müller, Ostfalia

Mithilfe sogenannter „Beacons" verschmilzt das Internet mit der direkten Umgebung des Benutzers. Die Beacon-Technologie wird in den nächsten Jahren grundlegend neue Möglichkeiten entstehen lassen, um mit dem Internet und der direkten Umwelt zu interagieren. Die Konzerne Apple und Google prognostizieren einen Zuwachs von Beacons in Millionenhöhe für die nahe Zukunft. Doch was verbirgt sich hinter Beacons und den angepriesenen Vorteilen für den Anwender?

Stellen Sie sich vor, Sie stehen in einem Museum und erhalten automatisch die passenden Informationen über das entsprechende Exponat oder bekommen beim Aufenthalt in einem Bahnhof Informationen über mögliche Reisemöglichkeiten direkt auf Ihr Smartphone. All dies und viel mehr ist bereits heute mit Beacons möglich. Dieser Artikel führt die zugrunde liegende Technologie ein und zeigt exemplarisch, wie sie verwendet werden kann.

Location Based Services/ Micro-Location

Welche Möglichkeiten ergeben sich, wenn wir in der Lage wären, mithilfe unseres Smartphones direkt mit unserer näheren Umgebung zu interagieren? Es ist längst kein Problem mehr, über GPS und andere Sensoren den ungefähren Standort des Benutzers zu ermitteln. Mit aktuellen Smartphones ist es etwa ohne Probleme möglich zu erkennen, ob man sich in einem Supermarkt, einem Kino oder einer Hochschule befindet. Doch wenn es darum geht zu ermitteln, ob man vor der Obsttheke oder der Fleischtheke, im Kinosaal A oder B oder im Hörsaal für Mathematik oder Informatik steht, hört es mit der Genauigkeit von GPS und Mobilfunkzellen auf. Hier beginnt „Micro-Location" interessant zu werden.

Um kontextbezogene Dienste in diesem Genauigkeitsbereich anzubieten, muss das Smartphone die Möglichkeit besitzen, den Benutzer auf wenige Zentimeter genau zu

lokalisieren. Location-based Services werden zu Micro-Location-based Services. Ziel ist es, dem Benutzer anhand seiner exakten Position die Interaktion mit seiner unmittelbaren Umgebung zu ermöglichen. Diese Contextual Services hängen damit mit der aktuellen Vision des Physical Web zusammen und sind ein zentraler Baustein zu dessen Verwirklichung.

Zur Veranschaulichung der Möglichkeiten vergleichen wir die aktuell gängigen und verwendeten Technologien. Mittels GPS sind wir in der Lage, bis auf etwa zehn Meter genau eine Position zu bestimmen. Diese Genauigkeit ermöglicht Dienste, die für die oben genannten Beispiele der Lokalisierung von Supermarkt, Kino oder Hochschule sinnvoll sind.

Eine GPS-Ortung funktioniert in vielen Fällen mehr schlecht als recht. Zum Beispiel sind in größeren Gebäuden mit guter Isolation kein GPS-Empfang und damit auch keine entsprechenden Dienste-Angebote möglich. Eine andere Möglichkeit wäre WLAN. Damit lässt sich zwar eine Lokalisierung in Gebäuden erreichen, allerdings ist diese Genauigkeit von rund fünfzig Metern für entsprechende Dienste nicht ausreichend.
Genau hier füllt Bluetooth als Basis-Technologie die Lücke zur Micro-Location. Mittels Bluetooth ist eine Ortsbestimmung bis auf wenige Zentimeter genau möglich. Wie das genau funktioniert, werden wir gleich sehen. Zur Verdeutlichung zeigt *Abbildung 1* noch einmal die Lokalisierungsgenauigkeiten der angesprochenen Technologien im Überblick.

Wer Beacons erfunden hat

Das englische Wort „Beacon" bedeutet auf Deutsch „Leuchtfeuer, Leuchtturm oder Blinklicht" und beschreibt einen von Apple im Jahr 2013 eingeführten Standard für die Navigation in geschlossenen Räumen. Beacons senden ein auf Bluetooth Low Energy (BLE) basierendes Signal aus, das von anderen BLE-fähigen Geräten empfangen und von entsprechenden Anwendungen verarbeitet werden kann. Das übermittelte Signal enthält die Identifikation des Beacon sowie eine Empfangsstärke (Received Signal Strength Indicator, RSSI). Ein Beacon selbst ist nicht in der Lage, Inhalte auszuliefern oder Benutzer zu lokalisieren und zu verfolgen. Es liegt also an uns Entwicklern, mithilfe von Anwendungen das gewünschte Verhalten zu realisieren.

Da es sich bei Bluetooth um ein grundlegend offenes Protokoll handelt, wird BLE bei iOS ab Version 7 und bei Android ab der Version 4.3 unterstützt. Alle aktuell verfügbaren Mobilgeräte mit Bluetooth unterstützen damit den Standard von Haus aus. Aktuelle Computer sind ebenfalls BLE-fähig. Die neueren Versionen von Windows, OS X und Linux haben die entsprechenden Treiber eingebaut.

Die Anwendungsszenarien von Beacons sind relativ breit gestreut: im Restaurant, Theater, Hotel, Schwimmbad, Arztpraxis, Krankenhaus, Universität, Supermarkt oder Kino. Beacons lässt sich nahezu überall für die Verbreitung unterschiedlichster Informationen einsetzen.

Da im Gegensatz zu normalen Bluetooth-Geräten bei Beacons keine Kopplung zwischen den an der Kommunikation beteiligten Geräten stattfindet, kommt bei Beacons ein anderer Bluetooth-Paket-Typ zum Einsatz. Es handelt sich um ein Advertising-Bluetooth-Package, also ein Broadcast-Paket. Um die Größen-Anforderungen eines einzelnen Pakets zu minimieren, wurde ein hochkomprimiertes Format definiert. *Abbildung 2* zeigt den schematischen Aufbau, den wir im Folgenden erläutern.

iBeacon (Apple Standard)

Der iBeacon-Standard *[2]* sieht im Wesentlichen drei unterschiedliche IDs für ein Beacon vor. Hierzu gehört eine Identifikation des Beacon oder auch Proximity ID in Form einer UUID. Dahinter verbirgt sich eine 16-Byte-Zahl, die hexadezimal notiert und in fünf Gruppen unterteilt wird. Eine solche UUID dient als Hersteller-Kennung und hilft somit, unterschiedliche Hersteller zu unterscheiden. Der Beacon-Hersteller Estimote verwendet

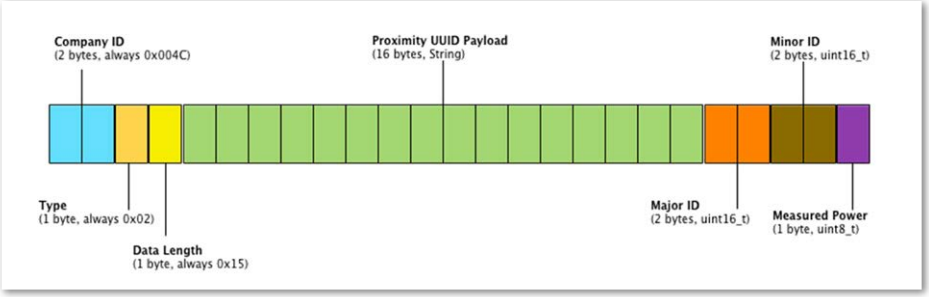


Abbildung 2: Der schematische Aufbau

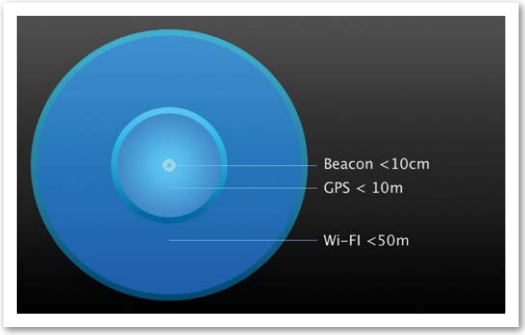


Abbildung 1: Lokalisierung entsprechend den Technologien

beispielsweise die UUID „B9407F30-F5F8-466E-AFF9-25556B57FE6D".

Zusätzlich besitzt jedes Beacon eine Major- und eine Minor-ID. Diese werden verwendet, um die Beacons bestimmten Zonen zuordnen zu können und somit einzelne Beacons zu identifizieren. Laut offizieller Darstellung von Apple sollte die Major-ID zum Beispiel dazu verwendet werden, Beacons einem bestimmten Gebäude zuzuordnen. Die Minor-ID kommt dann zum Einsatz, um den Raum oder die entsprechende Etage eindeutig zu definieren. Die eigentliche Verwendungsart steht aber jeder Anwendung frei.

Weiterhin wird in jedem Advertising-Paket die Signalstärke mit ausgeliefert, sodass auf dieser Basis die Entfernung zum Empfänger berechnet werden kann. Man darf aber nicht vergessen, dass Beacons auf einem 2,4-Ghz-Kanal senden und somit relativ stark durch andere Funksignale gestört werden können. Insbesondere räumliche Hindernisse wie Möbel oder Personen beeinflussen das Signal ebenfalls.

Im Netz lassen sich zahlreiche Tests finden, in denen die Genauigkeit von Beacons unterschiedlicher Hersteller untersucht wird. Eins lässt sich mit Sicherheit sagen: Die Genauigkeit von Beacons hängt stark vom Hersteller und der Bauweise sowie der Konfiguration des Beacon ab. Parameter, wie die Sendestärke (Transmit Power) selbst und die Wiederholungsfrequenz des Signals (Advertising Interval), haben eine große Auswirkung auf die Qualität des Signals. Beide Parameter haben auch großen Einfluss auf den Stromverbrauch und damit die Batterie-Lebensdauer. Das in iOS integrierte Framework von Apple bietet aufgrund der schlecht vorhersagbaren Genauigkeit nur drei Abstufungen von Entfernung an:

- Immediate (0-20 cm)
- Near (20 cm – 2 m)
- Far (2 – 7 m)

Anhand dessen ist schon ersichtlich, dass eine genaue Angabe der Distanz zwischen Beacon und Smartphone nicht immer angegeben werden kann. Wie kommen diese gravierenden Unterschiede zustande? Es gibt einige Unterschiede zwischen den SDKs der einzelnen Beacon-Hersteller:

- Drop-out-Rate
- Implementierung der Empfangsstärke (RSSI)
- Konsistenz von Beacons desselben Herstellers
- Ausrichtung der Antenne im Beacon
- Störsignale unter den Beacons

Die folgende Gleichung beschreibt, wie die Distanzmessung zwischen Beacon und Empfänger funktioniert „P(d)[dBM] = P(d0)[dBM] – 10n log(d/d0) – Z", wobei die einzelnen Parameter Folgendes bedeuten:

- D = erwartete Distanz
- d0 = ein Maß für eine Distanz für eine bekannte Signalstärke
- n = Parameter für den Signalstärkenverlust über steigende Distanz
- Z = Umwelt-Einflüsse

Es ist relativ schwierig, einen guten Algorithmus für die Distanz-Messung zu finden, der auch noch für einen Großteil der Geräte und verbauten Antennen gut funktioniert. Es hat sich aber gezeigt, dass die folgende Java-Methode eine sehr gute Annäherung an die wirkliche Distanz ermöglicht *(siehe Listing 1)*.

Der Code stammt von *[1]* und wurde durch Tests parametrisiert, die die Autoren durchführten. Die Konstanten „0.89976", „7.7095" und „0.111" wurden durch Ausgleichsrechnung, basierend auf den Testergebnissen, definiert. Sie sind also durch die Praxis validiert und keine Natur-Konstanten.

```
/* Calculate Distance of iBeacons */
protected static double calculateAccuracy(int txPower, double rssi) {
   if (rssi == 0) {
       return -1.0; // if we cannot determine accuracy, return -1.
   }

   double ratio = rssi*1.0/txPower;
   if (ratio < 1.0) {
       return Math.pow(ratio,10);
   }
   else {
       double accuracy = (0.89976)*Math.pow(ratio,7.7095) + 0.111;
       return accuracy;
   }
}
```

Listing 1

Eddystone

Im Juni dieses Jahres kam von Google relativ überraschend ein eigener Beacon-Standard namens „Eddystone" auf den Markt. Hierbei handelt es sich um eine direkte Alternative und damit Konkurrenz zum iBeacon-Standard von Apple. Im Gegensatz zu Apple versucht Google allerdings, einen Standard zu schaffen, der sowohl mit iOS als auch mit Android kompatibel ist. Ein weiterer Vorteil von Eddystone ist, dass er quelloffen auf GitHub unter einer Apache-2.0-Open-Source-Lizenz zur Verfügung steht *[3]*, während Apples Implementierung nicht öffentlich ist.

Während Beacons auf Basis des iBeacon-Standards eine zusätzliche App benötigen, geht Eddystone einen Schritt weiter und kann Informationen auch ohne zusätzliche App versenden. Dies ist möglich, da Eddystone im Gegensatz zu iBeacons nicht nur die üblichen UUIDs versenden, sondern im direkten Vergleich auch Internet-Adressen in Form von URLs ausstrahlen kann. Dies ermöglicht es, direkt Werbe-Inhalte an Anwender zu senden, ohne eine App dazwischenschalten zu müssen. Es entfällt also die Web-Komponente in Form eines zentralen Registers für UUIDs. Dieser Schritt ebnet den Weg zum Physical Web *[4]*. Es bleibt aber zu erwähnen, dass selbst URL-Frames nicht ohne die Chrome-App unter iOS lesbar sind. Es ist also nur unter Android möglich, ohne App Informationen zu erhalten, da hier Google bereits die nötigen Automatismen in das Betriebssystem eingebaut hat.

Eddystone Frames

Wie schon erwähnt, kann Eddystone auch ohne Umwege direkt Internet-Adressen ohne UUID ausstrahlen. Realisiert wird dies durch unterschiedliche Frame-Typen. Ein Eddystone-Beacon kann zwischen drei unterschiedlichen Frames wählen, die gesendet werden sollen:

- UID-Frame
- URL-Frame
- TLM-Frame

Der UID-Frame kommt dem des iBeacon am nächsten. Hierbei wird eine ID ausgesandt, die sich aus zwei weiteren IDs zusammensetzt. Die Namespace-ID entspricht hierbei einer Proximity-ID, während die Instance-ID dazu verwendet wird, zwischen einzelnen Beacons zu unterscheiden.

Der URL-Frame ermöglicht es, eine Internet-Adresse zu kodieren und diese anschließend direkt über das Bluetooth-Advertising-Paket zu verschicken. Android ist von Haus aus in der Lage, diese URLs aus den Paketen zu dekodieren und dann als Physical-Web-Objekte gruppiert darzustellen.

Beim TLM-Frame handelt es sich um einen speziellen Frame, der für die Überwachung der Beacons zum Einsatz kommt. Dieser Frame enthält verschiedene Hardware-nahe Informationen, etwa über Batteriestand, Temperatur, Advertising-PDU-Zähler und Uptime in Sekunden. Eddystone Beacons senden also nicht nur eine UID oder URL, sondern in konfigurierbaren Sekunden-Intervallen zusätzlich einen TLM-Frame mit Hardware-Informationen. Dieses Feature ermöglicht es erst, große Installationen von Beacons zu überwachen, um beispielsweise herauszufinden, wann die Batterien ausgewechselt werden müssen. Um schnell mit Eddystone Beacons starten zu können, sind auf der GitHub-Seite *[3]* mehrere Code-Beispiele für Android und iOS verfügbar.

Aktuell verfügbare Hardware (Beacons)

Mittlerweile gibt es eine große Anzahl von Beacon-Herstellern. Egal ob wetterfest, wasserdicht, mit Batteriebetrieb oder fester Stromversorgung mit Micro-USB – fast alles ist verfügbar. Doch was sollte man vor dem Kauf beachten? Viele Hersteller bringen ein eigenes Framework mit, um die Beacons direkt mit möglichst wenig Aufwand in bestehende Apps einzubinden. Doch nicht jedes Framework ist kompatibel mit iOS und Android. Außerdem muss man als Entwickler die Entscheidung treffen, ob man von den hauseigenen Cloud-Services der Hersteller abhängig sein möchte oder nicht.
Hinzu kommt, dass man darauf achten sollte, dass die Beacons über Firmware-Updates erweiterbar und somit Update-fähig sind. Da Googles Eddystone noch nicht finalisiert wurde, kann man davon ausgehen, dass es noch einige Änderungen und Er-

Abbildung 3: Der offene Estimote-Beacon, Quelle: http://estimote.com

weiterungen am Protokoll geben wird. Hier macht es sich bezahlt, wenn die Beacon-Hardware per Firmware- beziehungsweise Software-Update aktualisiert werden kann. Die Firma Estimote *[6]* hat Beacons und Nearables im Sortiment, die mit einem eigenen quelloffenen und auf GitHub gehosteten SDK leicht in vorhandene Projekte einbindbar sind. Außerdem ist das Framework Android- und iOS-kompatibel. Ein Developer-Kit, bestehend aus drei Beacons, ist bereits ab 99 Dollar im Angebot. Bei Testkäufen der Autoren wurden die Beacons innerhalb weniger Werktage geliefert. Estimote bietet auch eine eigene kostenlose App für iOS und Android an, um die Einstellungen der Beacons zu konfigurieren, also beispielsweise die Parameter. *Abbildung 3* zeigt ein Estimote-Beacon in geöffnetem Zustand.

Verwendung mit Java

Estimote stellt unter *[8]* ein SDK für Android zur Verfügung. Das API hinterlässt einen guten Eindruck und basiert hauptsächlich auf der Registrierung von Event-Listenern, die für Beacon-Sende- beziehungsweise Empfangsbereiche und Empfangsstärken parametrisiert werden können. Die Autoren können hier leider noch keine Beispiele präsentieren, hoffen aber, dies in einer der nächsten Ausgaben der Java aktuell nachholen zu können.

Was bringt die Zukunft?

Was die Zukunft angeht, rechnet man damit, dass bis zum Jahr 2018 allein in Deutschland die Anzahl an Beacons auf sechzig bis dreihundert Millionen ansteigen wird *[9]*. Nachdem Google in den Beacon-Markt eingestiegen ist, kann davon ausgegangen werden, dass das Unternehmen in den kommenden Monaten eventuell eigene Beacon-Hardware, basierend auf dem hauseigenen Framework, auf den Markt bringen wird.

Es bleibt jedoch das Problem, dass keines der beiden Protokolle, weder iBeacon noch Eddystone, plattformübergreifend ohne eine spezielle App funktioniert. Eddystone ist zwar iOS-kompatibel, benötigt aber aktuell noch die Chrome-App, die auf dem iOS Gerät installiert werden muss. Umgekehrt gilt dasselbe für Android mit iBeacons. Diese werden nur über Apps von Dritten gefunden. Es bleibt also die Hoffnung, dass sich beide Firmen auf einen Standard einigen und diesen dann jeweils in ihrem Betriebssystem fest verankern. Eine plattformübergreifende Nutzung von Beacons und deren Informationen würde das Physical Web dramatisch nach vorne bringen.

Fazit

Der durch Apple veröffentlichte iBeacon-Standard ermöglicht Micro-location-based Services beliebiger Couleur. Es zeichnet sich erst langsam ab, welch breites Spektrum an Anwendungen damit realisierbar ist. Nachdem Google auf den stark beschleunigenden Zug aufgesprungen ist, besteht kein Zweifel daran, dass Beacons zu einem großen Pfeiler in dem gerade gehypten Internet of Things werden. Mit Eddystone steht Java-Entwicklern eine Open-Source-Implementierung des Protokolls inklusive SDK zur Verfügung, um mit wenig Aufwand in diese interessante Welt neuartiger Anwendungen einsteigen zu können.

Referenzen

[1] *http://stackoverflow.com/questions/20416218/understanding-ibeacon-distancing/204340199-20434019*
[2] *https://developer.apple.com/ibeacon/*
[3] *https://github.com/google/eddystone/*
[4] https://google.github.io/physical-web/
[5] *https://developers.google.com/beacons/*
[6] *http://estimote.com*
[7] *http://beaconinside.com*
[8] *https://github.com/Estimote/Android-SDK*
[9] *https://iotcon.de/2015/sessions/future-beacons*

Constantin Mathe
con.mathe@ostfalia.de

Constantin Mathe studiert Informatik an der Ostfalia (Hochschule Braunschweig/Wolfenbüttel) und schreibt gerade seine Masterarbeit über das Thema „Micro-location mit Beacons". Während seines Studiums hat er mehrere Jahre als Java-Software-Entwickler bei der Symtavision GmbH gearbeitet.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia (Hochschule Braunschweig/Wolfenbüttel). Er ist Autor mehrere Java-EE-Bücher, Sprecher auf nationalen und internationalen Konferenzen und engagiert sich in der JUG Ostfalen sowie im iJUG.

Member Login

Username

Password

Remember me

Forgot password?

LOGIN

Social Login mit Facebook, Google und Co.

Georgi Kehaiov und Dr. Ing. Nadina Hintz, iC Consult GmbH

Facebook, Twitter oder Xing – jeder kennt sie und fast jeder nutzt sie auch täglich. Ob geschäftlich oder privat, ohne Social Networking ist mittlerweile eine Nutzung des Internets kaum vorstellbar. Dabei geht es allerdings nicht nur ums Liken (und seit Neustem auch ums Disliken) von Katzen-Fotos etc; für Web- und Mobile-Entwickler bieten die Social-Networking-Plattformen einiges mehr.

Social Networking ist zweifellos ein Teil unseres Online-Alltags geworden: Twitter, Facebook, Google+, Xing oder Instagram – jeder macht irgendwo mit. Allein Facebook genießt nach eigenen Angaben 1,49 Milliarden aktive Benutzer im Monat *[1]*, Tendenz steigend.

Die angelegten Benutzerprofile sind mit zahlreichen Daten des Benutzers angereichert: Profildaten wie Vor- und Nachname, Anschrift etc. sowie vom Benutzer selbst generierte und hochgeladene Inhalte wie Bilder, Videos und andere Dateien. In dem Zusammenhang wurden Lösungen gesucht, wie Daten einer Anwendung in einer anderen verwendet werden können. Das klassische Beispiel, das in der Literatur häufig angeführt wird, ist, dass eine Online-Anwendung, die kostenpflichtig Bilder ausdruckt und diese dann per Post verschickt, direkt auf die Bilder des Benutzers bei einer anderen Online-Anwendung, etwa einer Online-Galerie, zugreifen möchte.

An sich sollte diese Operation keine technischen Schwierigkeiten darstellen, es sei denn, die Online-Galerie setzt ein Login mit Benutzername und Passwort voraus, um die Bilder überhaupt anzuzeigen. Die einfachste Lösung wäre dann natürlich, dass die Online-Druckerei sich einfach die Login-Daten holt und damit den Zugriff gewährleistet. Lange Zeit war genau diese Praxis im Internet üblich. Mittlerweile redet man an der Stelle vom sogenannten „Password-Antipattern".

Gerade der Vorreiter des Social Networkings, Facebook, fragt bis heute beispielsweise nach dem Skype-Passwort des Benutzers, wenn dieser nach seinen Skype-Kontakten bei Facebook sucht. Bei einer solchen Abfrage reagieren allerdings viele User allergisch. Und das aus gutem Grund. Denn auch wenn ein Dienst verspricht, dass Login-Daten nicht anderweitig verwendet werden, ist es für Benutzer schwer nachvollziehbar, dass das Passwort nur das eine einzige Mal, von dem einen einzigen Dienst, zu dem einen einzigen Zweck verwendet wird.

OAuth 2.0: Das große Autorisieren

Anfang der 2000er Jahre haben sich einige schlaue Köpfe überlegt, wie das alles anders gehen könnte. Dabei, wie so oft in der Infor-

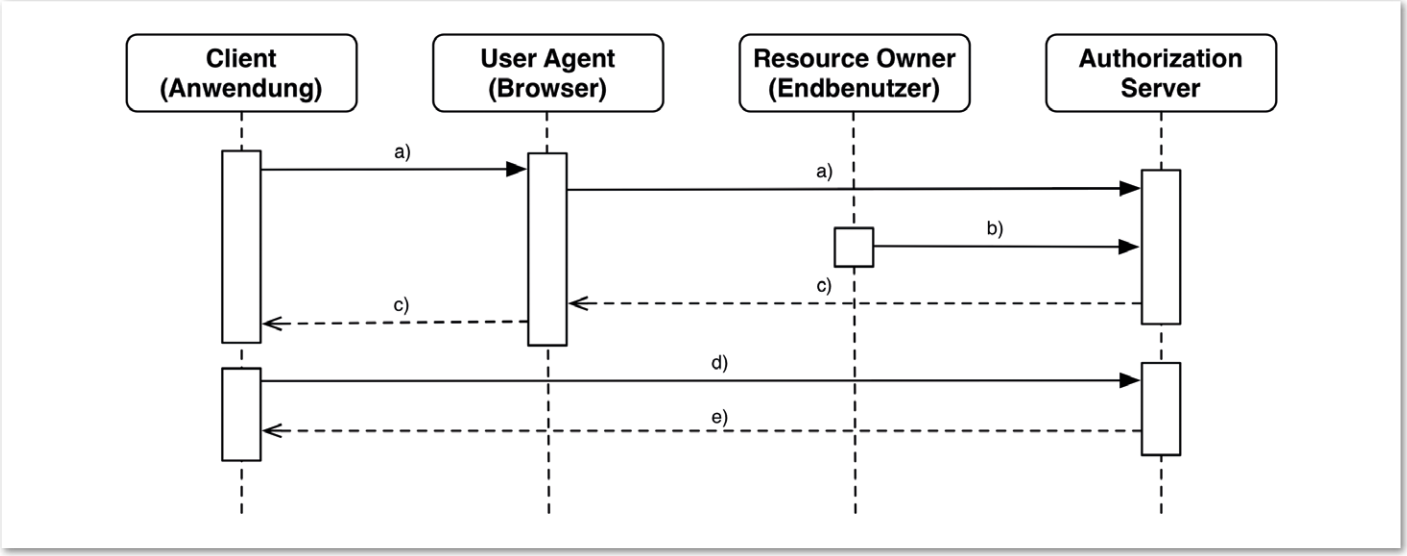


Abbildung 1: OAuth 2.0 Authorization Code Grant. a) Der Client generiert eine Authorization-Code-Abfrage und leitet den Benutzer über seinen Browser zum Authorization Endpoint am AzS. b) Der Benutzer authentisiert sich und autorisiert den Zugriff. c) Der AzS erstellt einen Authorization Code und gibt diesen über einen weiteren Redirect über den Browser an den Client. d) Der Client authentisiert sich und tauscht den Authorization Code gegen einen AT am Token Endpoint des AzSs. e) Der AzS erstellt einen AT (optional auch RT) und setzt diese in die Antwort zum Client.

matik, hat ein Beispiel aus der materiellen Welt den Weg zur Lösung gezeigt. In den USA wird an vielen Standorten, wie etwa an Restaurants, Kasinos, Hotels und Flughäfen, ein Service angeboten, bei dem Besucher ihr Auto von einem Angestellten einparken lassen können. Diesen Angestellten nennt man „Valet" und damit hat sich der Name des Dienstes als „Valet-Parking" etabliert. Bei etwas teureren Autos ist es üblich, dass sie gleich mit einem sogenannten „Valet-Key" ausgestattet sind. Damit kann das Auto nicht schneller als mit 20 bis 30 km/h gefahren werden. Diese maximale Geschwindigkeit reicht zwar vollkommen für den Valet aus, der das Auto von der Tür eines Kasinos bis zum Parkplatz und wieder zurück fahren darf, ist aber keineswegs ausreichend, um ihm etwa den Weg nach Mexiko zu ermöglichen.

Auf dieser Idee aufbauend, ist Ende 2007 das OAuth-Protokoll in der Version 1.0 *[2]* erschienen. Hier wird lediglich das Auto zu einer sogenannten „Ressource". Der Valet ist eine Dritt-Anwendung, der Valet-Key heißt „Access Token". Diese erste Version von OAuth hat mehrere Revisionen erlebt und am Ende ist daraus noch ein RFC geworden *[3]*. Entwickler hatten allerdings Probleme bei der Implementierung des Protokolls, unter anderem wegen der recht komplexen Signatur-Mechanismen, die dabei verwendet werden. Teilweise waren sich auch die Autoren nicht über alle Features einig und so kam es dazu, dass viele einzelne Implementierungen miteinander nicht kompatibel waren.

Erst im Jahr 2012 wurde in der Form eines weiteren RFC OAuth 2.0 vorgestellt *[4]*. Diesmal war es jedoch von Anfang an kein striktes Protokoll mehr, sondern nur ein Framework, das viele Besonderheiten absichtlich nicht bis ins Detail definiert und dem Implementierer viel Freiraum überlässt. Viele der großen Player – darunter auch Facebook, Google, Microsoft, Yahoo! – haben sich damit von Anfang an hinter den Standard gestellt.

Im Gegensatz zu den 1.0-Varianten werden in OAuth 2.0 Nachrichten erst einmal nicht signiert, dafür ist aber die Verwendung von TLS bei sämtlichen HTTP-Aufrufen vorausgesetzt. Im Allgemeinen ermöglicht OAuth den Zugriff auf geschützte Ressourcen. In diesem Zusammenhang redet man oft von einem bedingten Zugriff. In dem Framework sind vier Parteien definiert – Resource Owner (RO), Resource Server (RS), Client und Authorization Server (AzS).

Der Eigentümer einer Ressource ist am häufigsten ein Endbenutzer; die Ressourcen selbst sind dessen Daten. Ursprünglich hat lediglich der RO Zugriff auf die Ressourcen. Der RS kümmert sich um die Aufbewahrung der Ressourcen und reguliert den Zugriff darauf. Der Client ist die Dritt-Anwendung, die im Namen des ROs zugreifen will. Letztendlich ist der AzS dafür zuständig, den Zugriff des Clients zu gewährleisten. Häufig werden die Rollen von RS und AzS in der Praxis in einer Komponente abgebildet

Das OAuth-2.0-Framework beschreibt weiterhin vier grundlegende Typen von Grants – vier Möglichkeiten, wie ein Client an eine Ressource herankommen kann:

Der Authorization Code Grant *(siehe Abbildung 1)* eignet sich am besten für vertrauliche Clients. „Vertraulich" bedeutet in diesem Zusammenhang, dass der Client in der Lage sein soll, seine eigenen Credentials (etwa ein Client Secret, siehe weiter unten) vor der Außenwelt, inklusive dem Benutzer, zu schützen.

Ein Beispiel für einen vertraulichen Client wäre eine Web-Anwendung, die auf einem eigenen Server läuft und dem Benutzer nur Zugriff auf ein Frontend anbietet. Damit kann der Client sowohl Access Tokens (ATs) als auch Refresh Tokens (RTs) vom AzS anfordern. Die ATs werden später dazu verwendet, den tatsächlichen Ressourcenzugriff durchzuführen, und mit den RTs lassen sich neue ATs jederzeit (auch ohne die explizite Zustimmung des Benutzers bei jeder Iteration) generieren.

Der Flow wird vom Client mit einer Weiterleitung des Benutzers zu einem festgelegten Endpoint auf dem AzS initialisiert. Der Client muss sich an der Stelle anhand einer im Vorfeld vom AzS erstellten Client-Id identifizieren und darf noch einen gewünschten Scope angeben. Dieser bestimmt, welche Zugriffsrechte und -szenarien erwünscht sind. In einem weiteren Schritt wird der Benutzer zum RS weitergeleitet, wo er sich bei Bedarf anmelden und mit dem Scope dem Zugriff zustimmen muss.

Falls der Benutzer den Zugriff genehmigt, wird ein Authorization Code erstellt, der über eine letzte Weiterleitung zum Client weitergeschickt wird. Der Client darf dann mit dem Code direkt zum AzS gehen und sich dort ein AT (und, falls notwendig und zugestimmt, auch ein RT) holen. Beim letzten Schritt muss

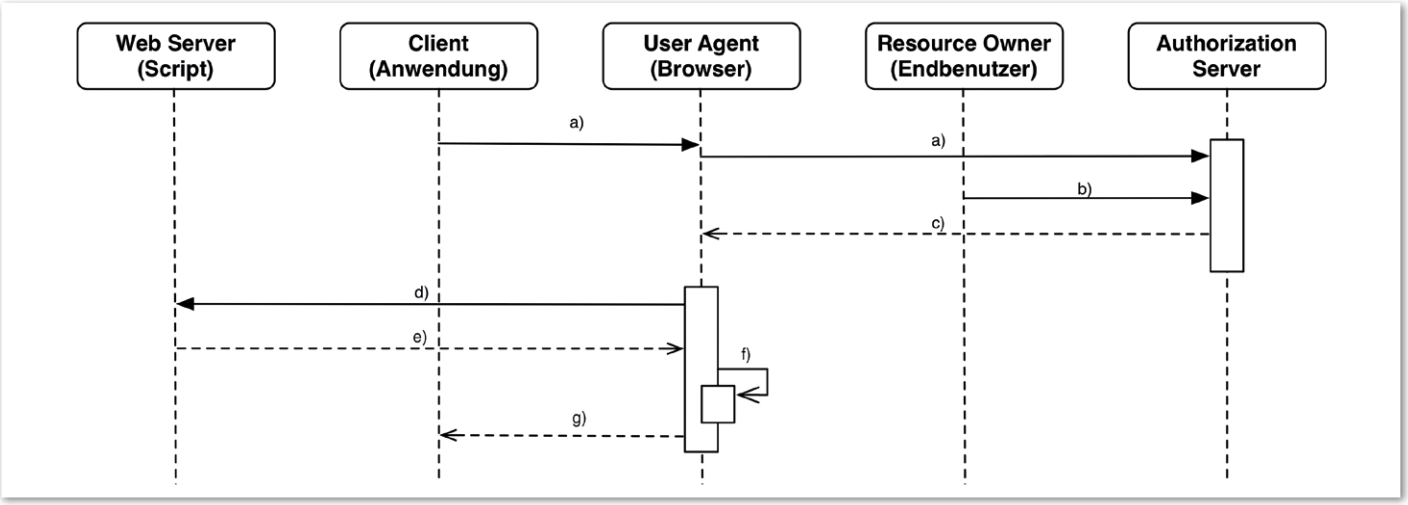


Abbildung 2: OAuth 2.0 Implicit Grant. a) Der Client generiert eine Authorization-Code-Abfrage und leitet den Benutzer über seinen Browser zum Authorization Endpoint am AzS. b) Der Benutzer authentisiert sich und autorisiert den Zugriff. c) Der AzS erstellt einen AT und schickt diesen als Fragment zurück zum Browser des Benutzers. d) Der Browser folgt einer Weiterleitung zu einer ins Internet gehosteten Ressource (ohne Angabe des Fragments). e) Dem Browser wird ein Script zur Verfügung gestellt, der zum Auspacken des ATs aus dem Fragment verwendet werden kann. f) Der Browser führt das Script aus und erhält den AT. g) Der AT wird zum Client weitergeleitet.

sich der Client über seine Client-Id und sein Client Secret beim AzS authentisieren.

Durch diesen Umweg stellt man sicher, dass der AT nicht in die virtuellen Hände des Endbenutzers geraten kann, wo er potenziell abgehört und auch missbraucht werden kann. Mit dem auf diese Art erstellten AT kann der Client endlich auf den RS zugreifen und sich die Ressourcen des Benutzers holen.

In der Praxis haben ATs eine kurze Lebensdauer und werden dann invalidiert. So stellt man sicher, dass, auch wenn Tokens in die falschen Hände geraten, kein Schaden in der Form von nicht erwünschten Zugriffen erfolgen kann. RTs dagegen sind langlebig, können jedoch zu jeder Zeit von dem Benutzer selbst zurückgezogen werden. Mit einem RT kann sich der Client auch im Hintergrund (ohne explizite Zustimmung des Benutzers für jede einzelne Transaktion) weitere ATs beschaffen und damit weitere Ressourcen-Anfragen beim RS stellen.

Beim Implicit Grant Flow *(siehe Abbildung 2)* kriegt der Client sein AT direkt als Resultat der Anmeldung des Benutzers beim AzS. Der Client muss sich also nicht explizit authentisieren; dadurch gilt dieser Flow als nicht so sicher wie der Authorization Code Grant. Deswegen sind hier auch keine RTs erlaubt. Das bedeutet wiederum, dass der RO bei jedem Request anwesend sein muss, um sich beim AzS autorisieren zu können und dem Request auch zuzustimmen. Außerdem wird der Token an der Stelle als URL-Fragment zum Client geschickt.

Sollte der Client nicht schon selber in der Lage sein, dieses Fragment auszulesen, muss er eine bestimmte Web-Ressource aufrufen, die ein Skript bereitstellt, das sich um das Entpacken des Tokens kümmert. Dieser Flow ist grundsätzlich für JavaScript-Clients gedacht, die komplett im Browser des Benutzers laufen und dadurch nicht in der Lage sind, eigene Client Secrets zu verwalten.

Das OAuth-2.0-Framework sieht zwei weitere Typen von Grants vor: Resource Owner Password Credentials Grant und Client Credentials Grant. Bei ersteren muss der Benutzer seinen Benutzernamen und sein Passwort preisgeben, dann führt der Client eine klassische Anmeldung durch. Dazu muss der Benutzer dem Client vollkommen vertrauen. Allerdings ist die Verwendung dieses Flows nicht empfehlenswert, wenn einer der schon besprochenen Flows umsetzbar ist.

Beim letzten Grant ist es ausreichend, wenn der Client ein Geheimnis (etwa ein Client Certificate) aufzeigen kann. Dann ist die Einmischung des ROs gar nicht notwendig. Dieser Flow wird generell nur dann empfohlen, wenn man eine sehr starke Vertrauensbeziehung zwischen allen beteiligten Parteien sicherstellen kann, wie bei der Server-to-Server-Kommunikation im eigenen Rechenzentrum oder im Unternehmensumfeld.

OpenID Connect: Identität leicht gemacht

Die grundlegenden Flows von OAuth 2.0 lassen sich sehr elegant in den meisten Web- oder Mobile-Applikationen anwenden, wenn der Benutzer Zugriff auf eine Ressource autorisieren soll. Nicht so günstig daran ist allerdings die Tatsache, dass die Client-Anwendung nichts von der externen Identität des Benutzers mitkriegt – diese muss weiterhin „lokal" gemanagt werden. Häufig ist die dadurch gewährleistete Anonymität auch tatsächlich ein Feature und kein Manko, meistens ist es jedoch von Vorteil, die Ressourcen des Benutzers in einen Kontext setzen zu können.

OpenID Connect (OIDC) *[5]* bietet laut der eigenen Dokumentation eine Identitätsschicht auf OAuth 2.0 und versucht damit die eben besprochene Lücke zu füllen. In der Realität zeigt OIDC, wie man die Identität des Benutzers als eine Ressource betrachten kann, und bietet einen formalisierten Weg, diese dann für die eigenen Zwecke zu verwenden. Damit tritt das Protokoll als direkter Nachfolger von SAML 2.0 und OpenID 2.0 auf die Bühne der Authentication-Protokolle und versucht diese zu ersetzen.

Für Entwickler heißt das wiederum, dass das Verwalten von Benutzerdaten komplett ausgelagert werden kann, inklusive Passwörter und Benutzernamen. Damit wird nicht nur der Aufwand bei der Implementierung reduziert, sondern insbesondere auch der Benutzer stark entlastet. Bisher war es meist üblich, für jede einzelne Anwendung einzelne Credentials zu verwenden – für jede Anwendung einen Benutzername und ein Passwort. Das führt aber dazu, dass Benutzer meistens viel zu schwache Passwörter wählen, ein einziges oder wenige Passwörter verwenden. damit sie sich diese merken können. Mit OIDC hingegen kann man das Identifizieren der Benutzer komplett auslagern. In diesem Zusammenhang redet man auch von „Social Login", da dieses Auslagern am häufigsten in Richtung der großen sozialen Netzwerke wie etwa Facebook und Google stattfindet.

Im konzeptuellen Teil baut OIDC sehr stark auf OAuth 2.0 auf und setzt dabei großen Wert auf das Format der Tokens, die verwendet werden – ATs, RTs und die neu hinzugekommenen ID-Tokens. Die ID-Tokens sind JSON Web Tokens (JWTs) *[6]*, die mit sogenannten „Claims" (auf Deutsch: Ansprüche) ausgefüllt werden. Diese Ansprüche und somit der ID-Token beschreiben die Authentisierung des Benutzers gegenüber dem AzS und dienen dazu, die reine Autorisierung, die mittels OAuth 2.0 erfolgt, durch eine Authentisierung anzureichern und eine Verbindung zwischen der Identität des Benutzers beim AzS und seinen Ressourcen herzustellen. Manche der im Standard vorgeführten Claims sind obligatorisch, andere nicht, und falls notwendig, darf man auch eigene Claims definieren. *Tabelle 1* zeigt eine kurze Übersicht über die obligatorischen Claims.

Claim	Beschreibung
iss	„Issuer" eines ID-Tokens ist diejenige Partei, die den Token erstellt hat
sub	„Subject" stellt eine eindeutige Kennzeichnung des Endbenutzers aus der Sicht des „Issuer"
aud	„Audience" beschreibt eine oder mehrere Parteien, für die dieser ID-Token gedacht ist
exp	Gibt einen Zeitpunkt in der Zukunft an; danach ist der Token nicht mehr gültig
iat	Zeitpunkt, an dem der Token erstellt wurde

Tabelle 1: OpenID Connect – obligatorische ID-Token-Claims

Beim OIDC-Standard werden auch weitere Konzepte feiner definiert, unter anderem wird bei der Beschreibung des AzS betont, dass dieser noch einen UserInfo-Endpoint unterstützen soll. Dort kann sich der Client Profil-Informationen des Benutzers besorgen. Weiterhin werden die schon in OAuth 2.0 definierten Endpoints – Authorization Endpoint und Token Endpoint – strikter formalisiert. Der erste dient dazu, eine Genehmigung durch den RO zu gestatten, der zweite dafür, einen Grant für einen Token auszutauschen. Außerdem beschreibt OIDC auch eine Steuerung des Anmeldevorgangs an dem AzS.

Der OIDC-Standard sieht grundsätzlich zwei Flows vor: Authorization Code und Implicit, die entsprechend auf den gleichnamigen OAuth 2.0 Flows basieren. Beim Authorization Code Flow erhält der Client wieder einen Code am Authorization Endpoint und darf dann später Tokens vom Token Endpoint be-

ziehen. Beim Implicit Flow werden gleich alle Tokens am Authorization Endpoint erfragt. Es wird auch ein hybrider Flow vorgesehen, bei dem ein Grant am Authorization Endpoint genehmigt wird und dann die Tokens von beiden Endpoints ausgestellt werden.

Der Authorization Code Flow bei OIDC wird vom Client initialisiert. Dabei wird zuerst der Benutzer zum Authorization Endpoint weitergeleitet. Am Beispiel von Google könnte so eine Weiterleitung in einem ganz rudimentären Spring-Controller aussehen *(siehe Listing 1)*. Bei den nachfolgenden Code-Beispielen hat man der Einfachheit halber und zur besseren Lesbarkeit auf die Sicherheits- und Integritäts-Aspekte der Implementierung verzichtet. Mehr Details dazu findet man in den offiziellen Implementer Guides unter *[7]* und *[8]*.

Dabei wird der Parameter „client_id" aus einer Konfigurationsklasse ausgelesen und dem Request hinzugefügt. Diese ID dient dazu, den Client beim AzS zu identifizieren, und muss im Voraus registriert werden. Beim „response_type" steht der Wert „code" für den Authorization Code und der „scope"-Wert gibt an, dass man auf die Profildaten des Benutzers zugreifen will. Ganz wichtig ist hier der letzte Parameter „redirect_uri". Dieser gibt an, unter welcher Adresse der Benutzer weitergeleitet werden soll, nachdem er sich bei Google angemeldet und den Zugriff zugelassen hat. In diesem Beispiel erfolgt die Weiterleitung auf einen weiteren Endpoint der lokalen Anwendung.

Angenommen, der Benutzer hat sich im AzS (hier Google) angemeldet und den Zugriff genehmigt, dann wird dort ein Authorization Code generiert, der in einer Weiterleitung als Parameter mit dem Namen „code" zu der „redirect_uri" von oben angehängt wird. Dementsprechend wird eine weitere Methode in dem Controller von oben mit der folgenden Signatur definiert *(siehe Listing 2)*.

Als nächster Schritt im Flow muss der Client mit dem gerade erzeugten Code den Token-Endpoint am AzS ansprechen, um dort AT- und ID-Token zu erhalten. Die „callback()"-Methode wird um die folgenden Zeilen ergänzt. Dabei wurde das REST-Assured-Framework von Jayway verwendet, um die REST-Aufrufe zu generieren *(siehe Listing 3)*.Hier wird der Authorization Code weitergereicht und man findet wieder die Parameter „client_id" und „redirect_uri" von eben. Der „client_secret"-Wert identifiziert den Client eindeutig. Es ist wichtig, an der Stelle zu beachten, dass dieser Request jetzt direkt zum AzS geht und nicht über eine Weiterleitung des Benutzers. Damit ist ausgeschlossen, dass böswillige Anwender an den Secret gelangen. Weiterhin muss man noch den Grant- (oder auch Flow-)Typ angeben, in diesem Fall „authorization_code". Die vorherigen Schritte entsprechen übrigens auch dem normalen OAuth-2.0-Authorization-Code-Flow. Der wesentliche Unterschied dazu besteht in der Antwort auf die letzte „POST"-Anfrage. Sie beinhaltet nämlich ein AT- und ein ID-Token, wobei der ID-Token als JWT dargestellt ist *(siehe Listing 4)*.

Im letzten Schritt des Flows muss der Client den ID-Token validieren und kann dann mit dem AT den Userinfo-Endpoint am AzS abfragen, um sich dort die Profildaten des Benutzers zu holen. Am Beispiel unten werden diese in ein generisches Bean „UserInfo" umgewandelt *(siehe Listing 5)*. Damit wären die „callback()"-Methode und auch der OIDC-Authorization-Code-Flow vollständig – ein Benutzer konnte sich bei Google anmelden und damit effektiv ein Social-Login in einer anderen beliebigen Anwendung durchführen.

```
@RequestMapping("/initflow")
public String initflow() {
  String redirectUrl =
      "https://accounts.google.com/o/oauth2/auth" +
      "?client_id=" + cfg.getProperty("client.id") +
      "&response_type=code" +
      "&scope=profile+email" +
      "&redirect_uri=http://localhost:8080/callback";
  return "redirect:" + redirectUrl;
}
```

Listing 1

```
@RequestMapping("/callback")
public String callback(
  @RequestParam(value = "code") String code,
  Model model) throws Exception { ... }
```

Listing 2

```
String authZResponse =
  with()
      .parameter("code", code)
      .parameter("client_id", cfg.getProperty("client.id"))
      .parameter("client_secret", cfg.getProperty("secret"))
      .parameter("redirect_uri", "http://localhost:8080/callback")
      .parameter("grant_type", "authorization_code")
      .post("https://accounts.google.com/o/oauth2/token")
      .asString();
```

Listing 3

```
{
  "access_token" : "ya29...Fw",
  "token_type" : "Bearer",
  "expires_in" : 3599,
  "id_token" : "eyJhb...ZmEifQ.eyJpc3...2NjQOfQ.Ah9x....S3MRQ"
}
```

Listing 4

```
String accessToken = from(authZResponse).get("access_token");
Response resourceResponse =
  with()
      .parameter("access_token", accessToken)
      .get("https://www.googleapis.com/oauth2/v3/userinfo");
UserInfo userInfo = resourceResponse.as(UserInfo.class);
model.addAttribute("userInfo", userInfo);
return "userinfo";
```

Listing 5

Fazit

Dieser Artikel dient lediglich als eine kurze Übersicht und Einleitung in die Themen „OAuth 2.0" und „OpenID Connect". Es wurde absichtlich auf zu viele Details verzichtet, um den wesentlichen Überblick zu erhalten. In den Code-Beispielen kamen Spring Boot und Jayway REST-assured zum Einsatz. Es existieren aber auch viele andere Open-Source-Bibliotheken, mit denen man die hier diskutierten Protokollflüsse implementieren kann, und es gibt auch viele Fertig-Lösungen (sowohl Open Source als auch proprietär), die OAuth 2.0 und OpenID Connect bis ins tiefste Detail unterstützen. Weitere Informationen kann man sich im Internet beschaffen. Dazu sind die am Ende des Artikels aufgeführten Quellenangaben ein guter Ausgangspunkt.

Sowohl OAuth 2.0 als auch OpenID Connect zählen heute zu wichtigen Bausteinen des Webs – sie erlauben die Integration und Interoperation von unzähligen Web- und mobilen Diensten. Beide Spezifikationen folgen dem Mantra „Keep simple things simple, make complex things possible" und bieten damit dem Entwickler nahezu unbegrenzte Möglichkeiten und viel Spielraum.

Weitere Informationen

[1] *http://newsroom.fb.com/company-info/ besucht am 20.09.2015*
[2] *http://oauth.net/core/1.0/*
[3] *http://tools.ietf.org/html/rfc5849*
[4] *http://tools.ietf.org/html/rfc6749*
[5] *http://openid.net/connect/*
[6] *http://tools.ietf.org/html/draft-ietf-oauth-json-web-token-25*
[7] *https://openid.net/specs/openid-connect-basic-1_0.html*
[8] *http://openid.net/specs/openid-connect-implicit-1_0.html*

Georgi Kehaiov
georgi.kehaiov@ic-consult.de

Georgi Kehaiov ist als junger Berater bei der iC Consult GmbH in Süddeutschland unterwegs. Schon im Studium war er von der IT-Sicherheit fasziniert. Im beruflichen Alltag versucht er, seine Kunden auf robuste und skalierbare Lösungen aufmerksam zu machen und unterstützt sie dann beim Entwurf und der Umsetzung.

Schließung des Oracle-Software-Produkt-Supports in Deutschland steht unmittelbar bevor

Marina Fischer, DOAG Online

Die bisherigen Gerüchte um den Stellenabbau im europäischen Support sollen sich schon bald bewahrheiten. Mario Kowalski, Customer Support Country Leader bei Oracle Deutschland, bestätigte im Gespräch mit der DOAG die bevorstehende Schließung des Oracle-Software-Produkt-Supports in Deutschland zum 31. März 2016. Er versprach: Sollten DOAG-Mitgliedern dadurch Probleme entstehen, stünde er als Ansprechpartner zur Verfügung.

In Deutschland sind rund 150 Mitarbeiter von der Stellenstreichung betroffen. Wie Oracle bestätigte, betrifft der Stellenabbau im Oracle-Support jedoch ganz Europa. Neben Deutschland trifft es auch die Supportzentren in Großbritannien, den Niederlanden und Frankreich. Insgesamt sollen 450 Stellen wegfallen. Ziel der Maßnahme ist die Verlegung des Supports nach Rumänien. Dort soll er laut Oracle quantitativ größer aufgestellt werden, genaue Mitarbeiterzahlen nannte das Unternehmen jedoch bisher nicht. Die Kunden stellt diese Entwicklung vor viele offene Fragen hinsichtlich der Gestaltung der Übergangszeit und der zu erwartenden Auswirkungen auf die Support-Qualität.

DOAG fürchtet Qualitätseinbußen

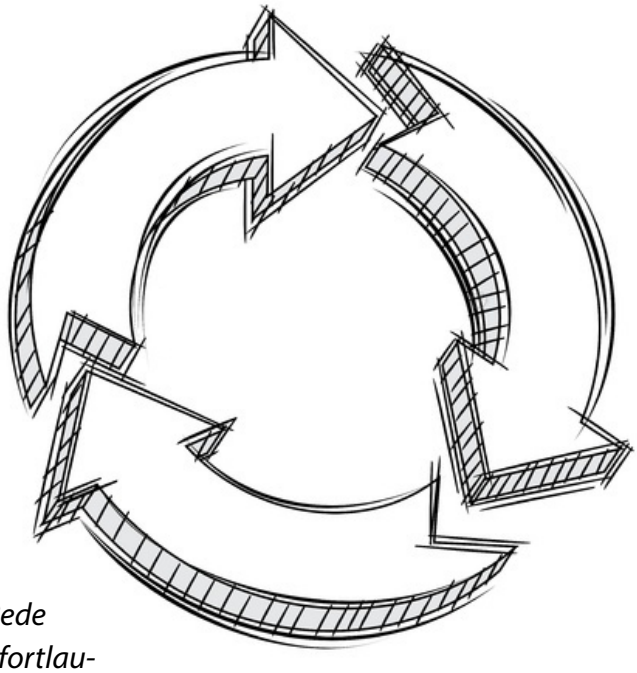
Die DOAG befürchtet hier einen zumindest kurzfristigen Qualitätsverlust in der Übergangsphase und initiierte daher ein Treffen mit Oracle, um über die Entscheidung und deren Auswirkungen zu sprechen. „Wichtig für die Kunden ist vor allem, ob Oracle in der Lage sein wird, die Qualität des Supports sowohl qualitativ als auch quantitativ aufrecht zu erhalten", so Dr. Dietmar Neugebauer, Vorstandsvorsitzender der DOAG. „Entscheidend wird dabei sein, wie der Übergang Ende März reibungslos durchgeführt werden kann. Nichtsdestotrotz bedauert die DOAG es sehr, dass Oracle Deutschland sich von vielen langjährigen Mitarbeitern verabschiedet und dadurch sehr viel Know-how in Deutschland verloren geht.

Kurzer Draht zu Oracle

Im Gespräch sicherte der Oracle-Supportverantwortliche Kowalski der DOAG zu, bei Support-Problemen den DOAG-Mitgliedern als Ansprechpartner zur Verfügung zu stehen. Christian Trieb, Leiter des DOAG Competence Center Support und der Datenbank Community, begrüßte im Namen der DOAG den angestrebten Dialog und hofft nun auf eine möglichst komplikationsfreie Übergangszeit. „Bezüglich der Support-Thematik wird die DOAG auch weiterhin eng mit Oracle in Kontakt stehen, denn erst langfristig wird sich herausstellen, wie sich die Servicequalität entwickelt und welche Rolle mögliche Schwierigkeiten in der sprachlichen Verständigung dabei spielen", so Trieb weiter. Wie die Kunden mit dem neuen Software-Produkt-Support in Rumänien zurechtkommen, wird auch die im Herbst dieses Jahres erneut geplante DOAG-Umfrage zur Qualität des Oracle-Supports zeigen.

Marina Fischer
marina.fischer@doag.org

Don't Repeat Yourself mit parametrisierten Tests



Bennet Schulz, codecentric AG

In der Literatur sind sie weit verbreitet und finden sich in nahezu jedem Buch über Testing wieder. In der Praxis dagegen sind sie nur bedingt angekommen oder werden bewusst ignoriert. Die Rede ist von parametrisierten Tests. Dieser Artikel zeigt anhand eines fortlaufenden Beispiels, wie und warum sie Bestandteil eines jeden Projekts sein sollten.

In vielen Projekten wird das Thema „Testing" noch immer stiefmütterlich behandelt. Dabei sollte Testcode nicht anders gepflegt werden als Produktionscode. Zumal er in Form von Akzeptanztests die Spezifikation der Software widerspiegelt und als Bindeglied zwischen den Stakeholdern und den Entwicklern dient. Als Unit-Test hingegen gibt er Entwicklern die nötige Rückendeckung für anstehende Refactorings und ein schnelles Feedback über die interne Struktur ihrer Software.

Ohne solche automatisierten Tests wird die Entwicklung neuer Features mit der Zeit immer zeitaufwändiger, teurer und Refactorings ähneln mehr und mehr einem Jenga-Spiel, bei dem der Turm mit jedem weiteren Baustein umzufallen droht. Aus diesem Grund ist die Investition in guten Testcode keine Zeitverschwendung, sondern vielmehr eine Notwendigkeit und eine bedeutende Investition in die Zukunft der Software.

Rekursive Berechnung der Fibonacci-Zahlen

Die Fibonacci-Folge eignet sich sehr gut, um zu zeigen, welchen Dienst parametrisierte Tests in Bezug auf Lesbarkeit und Wartbarkeit des Testens erweisen. Ihre Berechnung kann entweder iterativ oder rekursiv implementiert werden. Im Folgenden wird die rekursive Implementierung verwendet *(siehe Listing 1)*.

Die Fibonacci-Folge beginnt (optional) mit der Null als erster Fibonacci-Zahl. Die zweite und dritte Fibonacci-Zahl ist die Zahl eins. Um eine Fibonacci-Zahl größer eins zu berechnen, muss somit die Summe beider Vorgänger rekursiv ermittelt werden. Die „compute()"-Methode deckt diese Bedingungen ab.

Testen ohne Hilfsmittel

Um die korrekte Berechnung der Fibonacci-Zahlen ohne Hilfsmittel zu testen, sind mehrere (mindestens drei) Test-Methoden notwendig, die alle Äquivalenzklassen abdecken. Alle diese Test-Methoden berechnen zu einem übergebenen Parameter die zugehörige Fibonacci-Zahl *(siehe Listing 2)*.

Ein Blick auf die drei Tests genügt, um einen Verstoß gegen das aus dem Software-Craftmanship beziehungsweise aus dem „Clean Code" bekannten „Don't Repeat Yourself"- Prinzip (DRY) zu erkennen. Alle Testmethoden erwarten zu einem übergebenen Wert einen bestimmten berechneten Wert. Der einzige Unterschied besteht in den Übergabeparametern und in dem erwarteten Ergebnis. Interessant sind somit lediglich acht Zeichen – tatsächlich geschrieben wurden allerdings fast 350. Eine naheliegende Lösung wäre an dieser Stelle, alle Tests durch einen einzigen Test mit Streams oder auch einer „for"-Schleife zu ersetzen und die Assertions aus den einzelnen Tests innerhalb einer einzigen Schleife zu prüfen, beispielsweise wie in *Listing 3*.

Dieser Ansatz erfüllt zwar das DRY-Prinzip und reduziert die zuvor geschriebenen 350 Zeichen, scheitert allerdings im Falle eines Fehlers. Eine einzige fehlerhafte Iteration genügt, um alle Iterationen mit einem roten Balken zu versehen. Der Test würde nicht für eine einzige Iteration fehlschlagen, sondern für alle und somit wäre es nicht mehr ersichtlich, welche Teile der Berechnung noch korrekt funktionieren und welche nicht. Dieser Ansatz ist zwar gut gemeint, führt allerdings nicht zum Ziel.

Parametrisierte Tests mit JUnit

Abhilfe schaffen an dieser Stelle parametrisierte Tests. Seit JUnit 4 sind sie Teil der Standardbibliothek. Sie ermöglichen es, Tests zu parametrisieren und somit viele gleiche Tests mit unterschiedlichen Assertions zu einem einzigen parametrisierten Test zusammenzufassen.

```
public static int compute(int i) {
  if (i <= 0) {
     return 0;
  } else if (i == 1) {
     return 1;
  } else {
     return compute(i - 2) + compute(i - 1);
  }
}
```

Listing 1

In Bezug auf die Tests der Fibonacci-Zahlen bedeutet das: Alle Tests lassen sich auf einen einzigen parametrisierten zurückführen, ohne dadurch den Testumfang zu reduzieren. Aus den mindestens drei Tests wird somit ein einziger, unabhängig davon, wie viele Parameter beziehungsweise berechnete Fibonacci-Zahlen getestet werden *(siehe Listing 4)*.

Um mehrere Tests zu einem einzigen parametrisierten Test zu transformieren, müssen die JUnit-Testklassen mit einem speziellen JUnit-Runner versehen werden. Das geschieht durch die Annotation „@RunWith(Parameterized.class)" oberhalb der Testklasse.

Zusätzlich dazu wird eine Methode benötigt, die die zu parametrisierenden Daten zurückgibt. Im konkreten Fall sind es der übergebene Wert, zu dem die Fibonacci-Zahl berechnet werden soll, und die erwartete Fibonacci-Zahl als Ergebnis. Diese Tupel finden sich in der mit „@Parameters" annotierten Methode wieder.

Als Letztes bedarf es noch zweier Konstruktor-Parameter sowie der gleichen Anzahl an Feldern, um den parametrisierten Test ausführen zu können. Das ist notwendig, weil bei der Testausführung mehrere Instanzen der „FibonacciTest"-Klasse erzeugt werden. Genau genommen wird die Klasse „FibonacciTest" so oft instanziiert, wie es Tupel in der mit „@Parameters" versehenen Methode gibt. Im *Listing 4* sind es fünf Instanziierungen.

```
@Test
public void should_compute_fibonacci_number_of_zero() {
  assertEquals(0, Fibonacci.compute(0));
}

@Test
public void should_compute_fibonacci_number_of_one() {
  assertEquals(1, Fibonacci.compute(1));
}

@Test
public void should_compute_fibonacci_number_of_two() {
  assertEquals(2, Fibonacci.compute(3));
}
```

Listing 2

```
@Test
public void compute_fibonacci_numbers_without_parameters() {
  for (Integer[] tuple : tupleList) {
     assertEquals(tuple[1],Fibonacci.compute(tuple[0]));
  }
}
```

Listing 3

```
@RunWith(Parameterized.class)
public class FibonacciTest {

  @Parameters
  public static Collection<Object[]> data() {
     return Arrays.asList(new Object[][]{
        {0, 0}, {1, 1}, {2, 1}, {3, 2}, {4, 3}
     });
  }

  private int fInput;
  private int fExpected;

  public FibonacciTest(int input, int expected) {
     fInput = input;
     fExpected = expected;
  }

  @Test
  public void should_compute_fibonacci_numbers() {
     assertEquals(fExpected, Fibonacci.compute(fInput));
  }
}
```

Listing 4

Parametrisierte Tests mit JUnitParams

Einen etwas eleganteren Weg nimmt die Bibliothek „JUnitParams". Wie die Standard-Implementierung von JUnit verwendet auch JUnitParams einen spezialisierten JUnit-Runner. Durch die Verwendung des anderen Runner kann sowohl auf die mit „@Parameters" versehene Methode zur Bereitstellung der Parameter als auch auf die beiden Felder innerhalb der Klasse verzichtet werden. Im Gegenzug muss jede hier zu parametrisierende Methode mit „@Parameters" annotiert sein. *Listing 5* zeigt diesen Unterschied.

Durch den „JUnitParamsRunner" kann auf die Initialisierung in Form von Methoden und Feldern verzichtet werden. Das *Listing 5* testet dieselbe Funktionalität ab wie das vorherige Beispiel mit dem Standard JUnit Parameterized Runner. Wie zu erkennen ist, sorgen parametrisierte Tests mit „JUnitParams" im Vergleich zu der Standard-Implementierung dafür, dass Tests noch lesbarer werden, als sie es durch den Parameterized Runner ohnehin schon wären. Die Parameter befinden sich direkt an der parametrisierten Methode und müssen nicht zuvor durch Methoden in Feldern zwischengespeichert werden. Das bietet den Vorteil, dass größere Testklassen mit „JUnitParams" nicht unlesbar werden, da weder Bereitstellungsmethoden (mit „@Parameters" annotierte Methode) noch Felder zur Zwischenspeicherung erstellt werden müssen.

Besonders in größeren Testklassen mit verschiedenen parametrisierten Tests zeigt „JUnitParams" an der Stelle seine Stärken gegenüber dem Parameterized Runner von JUnit. Neben primitiven Datentypen bietet „JUnitParams" auch noch weitere Möglichkeiten: Es können ebenso objektwertige Datentypen wie Strings, aber auch Enumerations, Objekte, Methodenaufrufe und sogar „csv"-Dateien als Parameter vorkommen.

Fazit

Der Verzicht auf parametrisierte Tests bläht die Testcode-Basis unnötig auf und führt weder zu einer verbesserten Lesbarkeit noch zu einer besseren Wartbarkeit der Tests. Zusätzlich dazu wird auch noch gegen das DRY-Prinzip verstoßen. Parametrisierte Tests lösen dieses Problem. Durch sie können Tests

```
@RunWith(JUnitParamsRunner.class)
public class FibonacciTest {

  @Test
  @Parameters({"0, 0",
               "1, 1",
               "2, 1",
               "3, 2",
               "4, 3" })
  public void should_compute_fibonacci_numbers(int expected, int actual) {
     assertThat(expected, actual);
  }
}
```

Listing 5

mit der gleichen Sorgfalt wie der Produktionscode behandelt werden. Auf duplizierten Code wird verzichtet und das Augenmerk auf die eigentlich zu testende Funktionalität anstelle des Boilerplate-Codes gelegt.

Die Verwendung der Standard-Implementierung der parametrisierten Tests mit JUnit ist zwar besser als ein gänzlicher Verzicht auf parametrisierte Tests, dennoch stößt sie an die Grenzen der Lesbarkeit. Abhilfe schafft an dieser Stelle die Bibliothek „JUnitParams", die mit einer noch höheren Lesbarkeit auftrumpft.

Neben den Vorteilen der Verwendung gibt es aktuell auch noch Nachteile, die bedacht werden sollten. JUnit lässt leider nur einen einzigen Runner zu, sodass man sich gegebenenfalls zwischen verschiedenen Hilfsmitteln entscheiden muss. Bekannte Runner, wie zum Beispiel die verschiedenen Mockito JUnit Runner oder auch ein NestedRunner, um Tests innerhalb einer Klasse zu gruppieren, können dadurch nicht in Kombination mit parametrisierten Tests verwendet werden. Das könnte dazu führen, dass auf parametrisierte Tests verzichtet werden muss.

An dieser Stelle bleibt zu hoffen, dass mit dem zukünftigen JUnit 5 beziehungsweise JUnit Lambda eine Möglichkeit geschaffen wird, um Unit-Tests in einer ähnlichen Art und Weise zu parametrisieren, wie es bisher mit „JUnitParams" möglich ist. Aktuell scheitert dieser Ansatz daran, dass es der Standard JUnit Runner nicht zulässt, Test-Methoden mit Parametern zu versehen.

Bennet Schulz
mail@bennet-schulz.de

Bennet Schulz ist IT-Consultant und Entwickler mit Vorliebe für Java EE und JavaFX. In seiner Freizeit ist er an mehreren Java-User-Group-Aktivitäten und Konferenzen beteiligt. Er bloggt regelmäßig über seine Projekte und verschiedene Java-Themen (siehe „http://bennet-schulz.com").

Grundlagen des Batch Processing mit Java EE 7

Philipp Buchholz, esentri AG

In vielen Unternehmen gibt es immer wieder den Bedarf, große Datenmengen aus einem Quellsystem zu lesen, zu verarbeiten und abschließend in ein Zielsystem zu schreiben. Diese Vorgänge dauern typischerweise sehr lange, zumindest über mehrere Stunden, manchmal Tage. Aufgrund ihrer sehr langen Laufzeiten und sehr großen Datenmengen müssen diese Prozesse autark „unattended" ohne manuelle Eingriffe durchgeführt werden. Dieser Artikel ist der erste Teil einer zweiteiligen Serie und behandelt die Grundlagen von Batch-Verarbeitungen auf Basis von Java EE 7.

Typische Herausforderungen in diesen Verarbeitungen sind nicht nur technische Fehler, die innerhalb der Laufzeitumgebung auftreten können, sondern auch Fehler, die aus falschen oder nicht erwarteten Datenkombinationen resultieren. Mit der Einführung von Java EE 7 wurde der Java-Specification-Request 352 *(siehe „https://www.jcp.org/en/jsr/detail?id=352")* in den offiziellen Java-EE-Standard übernommen. Damit ist es möglich, solche lang laufenden, nicht interaktiven Batch-Prozesse zu implementieren. Bisher waren dafür ein außerhalb des Java-EE-Standards liegendes Framework oder eine Eigen-Implementierung notwendig.

Das Schaffen eines Standards innerhalb des Java-EE-Ökosystems bringt beim Projekteinsatz diverse Vorteile. Zum einen kann auf alle bekannten Technologien wie Context and Dependency Injection *(CDI, siehe „https://www.jcp.org/en/jsr/detail?id=299")*, Java Message Service *(JMS, siehe „https://docs.oracle.com/javaee/7/tutorial/jms-concepts.htm#BNCDQ")*, Java Management Extensions *(JMX, siehe „https://docs.oracle.com/javase/7/docs/technotes/guides/jmx/tutorial/tutorialTOC.html")* etc. zurückgegriffen werden. So sind unterschiedlichste Einsatz-Szenarien denkbar. Zum anderen kann auch das bereits im Java-EE-Bereich vorhandene Wissen von Mitarbeitern gut angewandt werden. Mit Java EE vertraute Mitarbeiter können dadurch schnell und produktiv Batch-Verarbeitungen entwickeln.

Zusätzlich sieht das Framework bereits eine Speicher-Möglichkeit vor, mit der Zustandsdaten von Batch-Verarbeitungen abgelegt werden. Lang laufende, datenintensive Verarbeitungen werden in Chunks aufgeteilt. Bei Fehlern dienen sie als Einstiegspunkt für die Wiederaufnahme der Verarbeitung. Die benötigten Informationen werden im oben erwähnten Zustandsspeicher, im Weiteren „JobRepository" genannt, hinterlegt.

Grundlagen

Alle benötigten Klassen für die Erstellung von Batch-Verarbeitungen befinden sich im Package „javax.batch". Beim Einsatz von Maven kann die benötigte Abhängigkeit auf das API mit der Angabe der groupID „javax.batch" und der artifactId „javax.batch-api" konfiguriert werden.

Während der Implementierung kann auf alle Möglichkeiten von Java EE zurückgegriffen werden. Assoziierte Objekte können beispielsweise per CDI injiziert sein. Dies ermöglicht einen einfachen Zugriff auf einen JPA-EntityManager, um Daten aus einer Datenbank zu lesen und in eine Datenbank zu schreiben. Zugriff auf den Kontext der Batch-Ausführung erhält man ebenfalls über die Injizierung per Dependency Injection. Dieser „JobContext" liefert Daten über den aktuellen Ausführungszustand des Jobs und der Property, mit der der Job gestartet wurde.

Verarbeitungsmodell

Die innerhalb von Java EE 7 zur Verfügung stehende Batch-Verarbeitung folgt einem bestimmten Aufbau *(siehe Abbildung 1)*. Das Modell zeigt den grundsätzlichen Aufbau einer Batch-Verarbeitung nach JSR 352. Ein Job stellt die gröbste Einheit einer Batch-Verarbeitung dar. Er besteht aus „N" Schritten („steps"), die innerhalb des Jobs ausgeführt werden müssen. Jeder Step liest Daten, die verarbeitet werden sollen, über einen „ItemReader" und übergibt sie an einen „ItemProcessor". Dieser übernimmt die Verarbeitung der Daten. Ein assoziierter „ItemWriter" ist für das Schreiben der Daten zuständig.

Jeder Job und die mit diesem assoziierten Schritte kommunizieren mit einem „JobRepository", um Metadaten, die die Ausführung derselben beschreiben, abzulegen und auszulesen. Das im Modell hervorgehobene „JobRepository" übernimmt innerhalb der Batch-Verarbeitung die zentrale Funktion der Zustandsspeicherung von aktuell laufenden Jobs beziehungsweise gelaufenen Jobs und deren Schritten. Das ermöglicht eine umfangreiche Kontrolle dieser Objekte. Damit lassen sich unter anderem Jobs neu starten.

Für das Realisieren von Neustarts und das Fortführen von Verarbeitungen kommt das Konzept der Checkpoints zum Einsatz. Ein „ItemReader" erhält beim Öffnen den Checkpoint des letzten erfolgreichen Lesevorgangs und kann damit die nächsten zu lesenden Daten bestimmen. Ein Checkpoint ist innerhalb des Modells als „CheckpointInfo" dargestellt. „CheckpointInfo"-Objekte müssen „Serializable" implementieren, da-

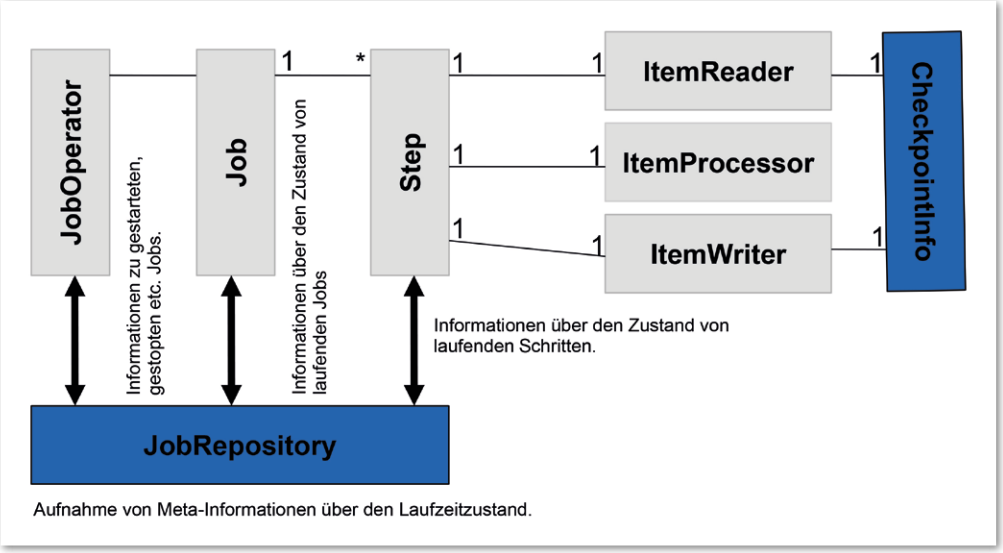


Abbildung 1: Grundsätzliche Batch-Verarbeitung nach JSR 352

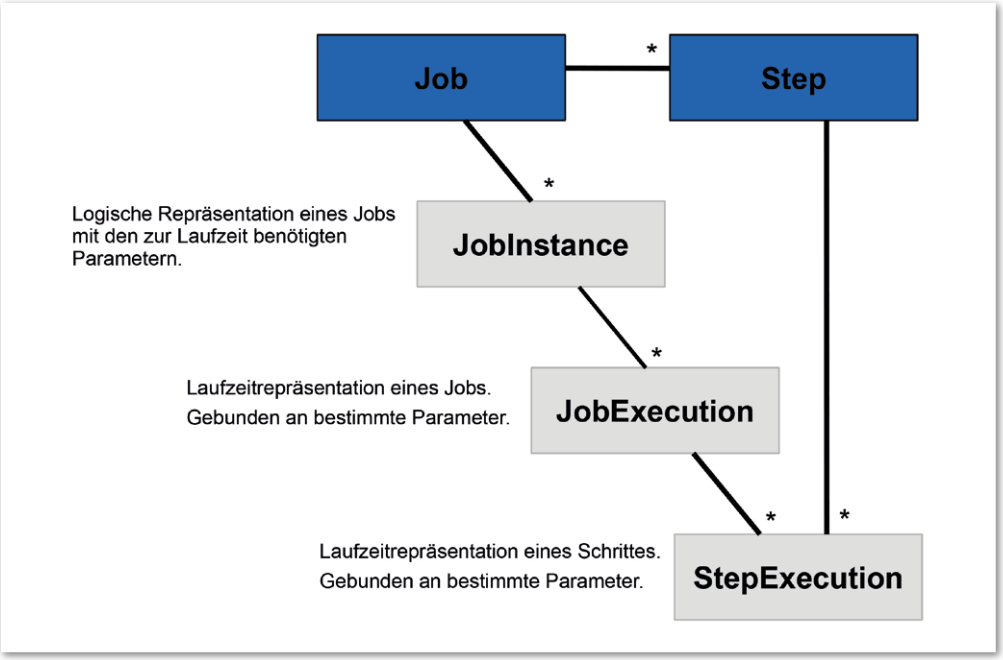


Abbildung 2: Laufzeit-Repräsentation von „Jobs" und „Steps"

mit sie innerhalb des „JobRepository" gespeichert werden können. Oftmals kann als „CheckpointInfo" die ID eines repräsentativen Datensatzes des zuletzt verarbeiteten Chunks verwendet werden, zum Beispiel die ID des letzten Datensatzes innerhalb des Chunks.

Der „JobOperator" dient dem Zugriff auf das Repository. Über diesen können Laufzeit-Informationen über ausgeführte oder aktuell in Ausführung befindliche Jobs und Schritte abgerufen werden. Außerdem lassen sich damit Jobs kontrollieren, zum Beispiel starten. Ein „JobOperator" bietet damit ein API für die Verwendung des „JobRepository".

Je nach verwendeter Implementierung gibt es unterschiedliche Möglichkeiten, die Informationen des „JobRepository" zu persistieren. Der JSR lässt die konkrete Implementierung offen.

Der Applikations-Server Oracle WebLogic bringt eine einfach zu konfigurierende Implementierung mit, die alle benötigten Informationen innerhalb einer relationalen Datenbank speichert. Auf die Möglichkeiten, die Oracle WebLogic im Rahmen der Batch-Verarbeitung bietet, geht der zweite Teil des Artikels genauer ein.

```
<?xml version="1.0" encoding="UTF-8"?>
<job id="OrderProcessingJob" xmlns=http://xmlns.jcp.org/xml/ns/javaee ver-
sion="1.0">
  <step id="loadIncomingOrders" next="createInvoice">
     <chunk item-count="2">
        <reader ref="incomingOrderReader"/>
        <processor ref="orderAcknowledgmentProcessor"/>
        <writer ref="orderAcknowledgmentWriter"/>
     </chunk>
  </step>

  <step id="createInvoice">
     <chunk item-count="2">
        <reader ref="orderAcknowledgmentReader"/>
        <processor ref="createInvoiceProcessor"/>
        <writer ref="sendInvoiceProcessor"/>
     </chunk>
  </step>
</job>
```

Listing 1

Laufzeit-Repräsentation

Es wurde bereits kurz beschrieben, dass innerhalb eines „JobRepository" Laufzeit-Informationen über gelaufene und aktuell laufende Jobs oder Schritte hinterlegt sind *(siehe Abbildung 2)*. Im Modell sind die Objekte „Job" und „Step" hervorgehoben, da sie Laufzeit-Repräsentationen bilden.

Als erster Schritt wird von einem Job zur Laufzeit eine „JobInstance" erstellt. Diese bildet eine logische Ausführung eines Jobs mit den entsprechenden Parametern ab. Damit können „N" „JobExecutions" assoziiert sein. Kann ein Job beispielsweise nicht erfolgreich ausgeführt werden, wird die „JobExecution" als fehlerhaft markiert. Ein Neustart erzeugt eine zusätzliche „JobExecution". Durch das Speichern dieser Informationen innerhalb des Repository kann die Ausführung von Jobs nachvollzogen werden, ein Neustarten wird möglich.

Eine „JobExecution" ist mit den Laufzeit-Repräsentationen der Schritte verknüpft, die ausgeführt wurden. Diese sind als „StepExecutions" abgebildet. Durch das Speichern dieser Informationen lässt sich bei Bedarf bestimmen, welcher Schritt innerhalb eines Jobs zu Fehlern geführt hat. Im Fehlerfall werden Informationen über aufgetretene Ausnahme-Situationen ebenfalls persistiert. Die beschriebenen Laufzeit-Repräsentationen sind im „JobRepository" gespeichert und können per „JobOperator" abgerufen und verwendet werden.

Beschreibung von Jobs mit einer „JobSpecification-Language"

Batch-Verarbeitungen werden mit einer sogenannten „JobSpecificationLanguage" (JSL) beschrieben und konfiguriert. Der Standard

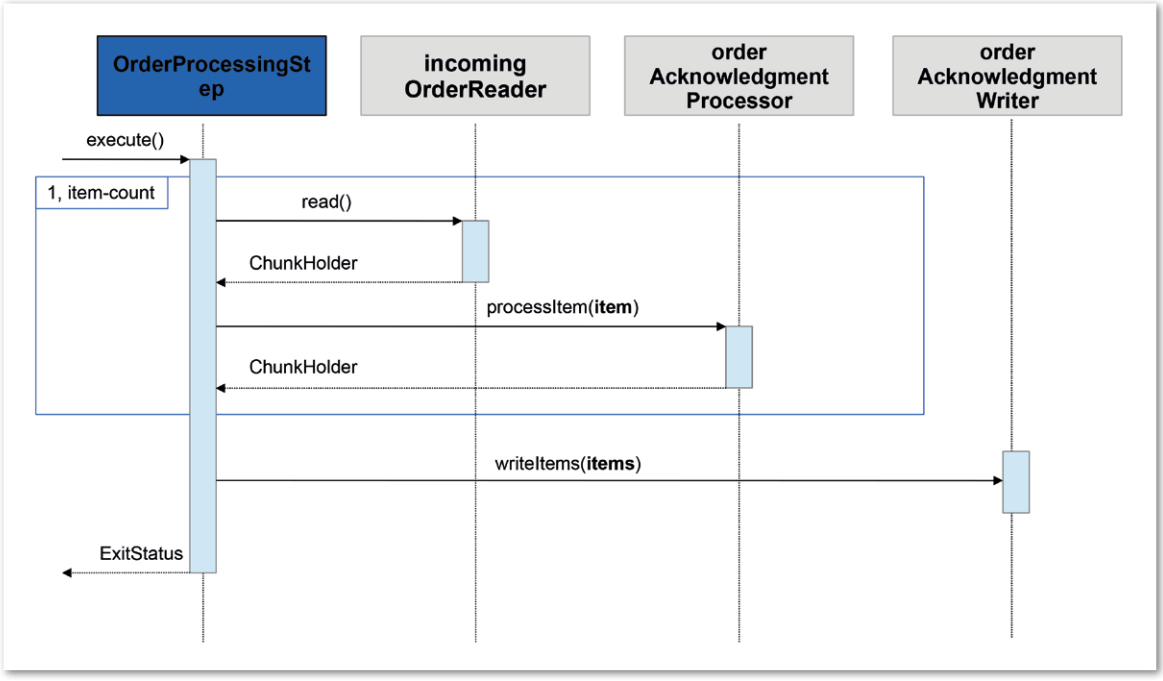


Abbildung 3: Ablauf Chunk-orientierte Datenverarbeitung

beschreibt eine XML-basierte JSL, kurz Job-XML *(siehe Listing 1)*.

Innerhalb einer Job-XML wird immer ein Batch-Job beschrieben. Das „job"-Element bildet das Root-Element der XML-Datei. Innerhalb einer Job-Beschreibung können „N" auszuführende Schritte definiert sein. Jedem Schritt muss über das „id"-Attribut eine eindeutige ID zugewiesen werden.

Die Reihenfolge der Schritte wird über das „next"-Attribut innerhalb des „step"-Elements festgelegt. Die Angabe einer „Step-ID" definiert den nächsten auszuführenden Schritt. Auf erweiterte Möglichkeiten der Flusssteuerung mit „Decision-" und „Flow-IDs" wird im zweiten Teil des Artikels eingegangen.

Innerhalb eines Schritts wird die Art der Verarbeitung festgelegt. Möglich ist eine Chunk-orientierte Datenverarbeitung (Chunk = zahlenmäßig beschränkte Anzahl an Datenblöcken, -sätzen oder einer anderen Dateneinheit) oder eine flexiblere Verarbeitung mit Batchlets. Die Verwendung von Batchlets wird ebenfalls im zweiten Teil des Artikels gezeigt. Die „item-count"-Attribute legen die Anzahl an Dateneinheiten fest, die in einem Chunk enthalten sind.

```
@Named
public class IncomingOrderReader extends AbstractItemReader {
  @PersistenceContext(unitName = "IncomingOrders")
  private EntityManager entityManager;

  @Inject
  private JobContext jobContext;

  private int maxResults;

  @PostConstruct
  public void setupIncomingOrderReader() {
     maxResults = Integer.parseInt(jobContext.getProperties().getProperty("maxResults"));
  }

  @Override
  public OrderChunkHolder readItem() throws Exception {
     CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
     CriteriaQuery<Order> criteriaQuery = criteriaBuilder.createQuery(Order.class);
     Root<Order> orderRoot = criteriaQuery.from(Order.class);
     criteriaQuery.where(criteriaBuilder.equal(orderRoot.get("orderStatus"), OrderStatus.NEW))
        .orderBy(criteriaBuilder.desc(orderRoot.get("createdAt")));
     TypedQuery<Order> typedQuery = entityManager.createQuery(criteriaQuery).setMaxResults(maxResults);
     OrderChunkHolder orderChunkHolder = new OrderChunkHolder();
     for (Order order : typedQuery.getResultList()) {
        orderChunkHolder.addOrder(order);
  }

     return orderChunkHolder;
  }
}
```

Listing 2

```
@Named
public class OrderAcknowledgmentProcessor implements ItemProcessor {

  @Inject
  private OrderPlacementService orderPlacementService;

  @Override
  public List<Acknowledgment<Order>> processItem(Object item) throws Exception {
     List<Acknowledgment<Order>> acknowledgments = new ArrayList<>();

     /* Check if we have correct instance. */
     if (!(item instanceof IncomingOrderChunkHolder))
        return null;

     /* Iterate and place incoming Orders. */
     IncomingOrderChunkHolder incomingOrderChunkHolder = (IncomingOrderChunkHolder) item;
     for (Order incomingOrder : incomingOrderChunkHolder.getIncomingOrders()) {
        acknowledgments.add(orderPlacementService.placeOrder(incomingOrder));
     }

     /* Return result of order placement. */
     return acknowledgments;
  }

}
```

Listing 3

Die Verarbeitung eines Chunks bildet immer die Demarkation einer Transaktion. Aus diesem Grund ist es sehr wichtig, eine passende Chunkgröße zu finden. Bei kleineren Chunkgrößen erhöht sich der Overhead durch die Transaktionssteuerung, bestehend aus dem Beginnen, dem Beenden und im Fehlerfall dem Zurückrollen einer Transaktion. Bei höheren Chunkgrößen ist die Verarbeitung insgesamt weniger granular steuerbar. Das liegt daran, dass beim Auftreten einer Fehlersituation dieser Chunk gesamt wiederholt werden muss. Bei einem großen Chunk dauert das entsprechend länger. Das Verhältnis zwischen fehlerfreien und fehlerhaften Daten in einem Chunk kann sich verschlechtern. Man denke an eine Situation, bei der nur ein Datensatz fehlerhaft ist und damit der gesamte Chunk nicht automatisiert verarbeitet werden kann.

Auflösen von Batch-Artifact-Referenzen

Innerhalb des beschriebenen Beispiels „Job XML" wird per „ref"-Attribut ein Batch-Artifact referenziert. Dieses kann im Grunde jede Klasse sein, die im Classpath liegt und vom Classloader des Containers für das Wiring (Injizieren von Referenzen mit Dependency Injection) zur Verfügung gestellt wird. In unserem Fall handelt es sich um einen Java EE Container. Aus diesem Grund werden die Referenzen auf Batch-Artifacts per CDI aufgelöst. Hierzu werden diese mit „@Named" annotiert und stehen damit dem Wiring per Dependency Injection (DI) mit Contexts and Dependency Injection (CDI) zur Verfügung. Die konkrete Verwendung von CDI in diesem Kontext sehen wir bei der Implementierung der benötigten Klassen.

Chunk-orientiert Datenverarbeitung

Das Sequenzdiagramm in *Abbildung 3* zeigt, dass die Implementierung eines „ItemReaders pro read()"-Aufrufs immer ein Item zurückliefert. Es wird innerhalb des „ItemProcessor" verarbeitet. Diese Schritte wiederholen sich so lange, bis die definierte „item-count"-Anzahl erreicht ist. Danach wird für den gesamten Chunk der „ItemWriter" einmal aufgerufen und das Ergebnis der Verarbeitung eines Chunks geschrieben.

Anzumerken ist, dass innerhalb der „ItemReader"-Implementierung benutzerdefinierte Typen polymorph zurückgegeben werden können. Innerhalb der „ItemProcessor"- und „ItemWriter"-Implementierung ist dann ein entsprechender Cast durchzuführen. Die Typisierung zur Designtime per Generics wird innerhalb des JSR 352 bisher nicht unterstützt.

Die XML-Datei kann einen benutzerdefinierten, sprechenden Namen tragen und muss innerhalb des Verzeichnisses „META-INF/batch-jobs" abgelegt sein. Wichtig ist hier, dass beim Starten eines Jobs über den „JobOperator" der Name der XML-Datei verwendet wird. Das wird im Beispielcode noch gezeigt.

Implementierung der benötigten Klassen

Um die Daten für die Verarbeitung zu lesen, implementiert man als Erstes einen „ItemReader". Dazu werden die Klasse „IncomingOrderReader" erstellt und das Interface „javax.batch.api.chunk.ItemReader" *(siehe „https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemReader.html")* implementiert.

Die Implementierung kann über das Ableiten der abstrakten Basisklasse „javax.batch.api.chunk.AbstractItemReader" *(siehe „http://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/AbstractItemReader.html")* erfolgen. Diese abstrakte Basis implementiert das genannte Interface mit leeren Methoden-Rümpfen. So müssen nur die wirklich benötigten Methoden überschrieben werden *(siehe Listing 2)*.

Dieser „ItemReader" erhält per Dependency Injection Zugriff auf einen „EntityManager", um Daten aus einer Datenbank zu laden, die eingehende Bestellungen enthält. Bestellungen werden in diesem Beispiel durch Objekte vom Typ „Order" repräsentiert. Für das Auslesen der Daten wird die Methode „readItem()" überschrieben. Der injizierte „JobContext" ermöglicht Zugriff auf den Kontext, innerhalb dessen der „ItemReader" ausgeführt wird. Über diesen kann beispielsweise ein Zugriff auf die Parameter erfolgen, mit denen die Ausführung vorgenommen wurde.

Um die geladene Bestellung verarbeiten zu können, wird das Interface „javax.batch.api.

```
@Named
public class OrderAcknowledgmentWriter extends AbstractItemWriter {

  @Inject
  private SendOrderAcknowledgmentService sendOrderAcknowledgmentService;

  @Override
  public void writeItems(List<Object> items) throws Exception {

      for (Object item : items) {

          if (!(item instanceof ChunkHolder)) {
              // throw appropriate exception.
          }

          /* Send Acknowledgment for each Order processed. */
          @SuppressWarnings(„unchecked")
          ChunkHolder<Acknowledgment<Order>> acknowledgmentChunkHolder = (ChunkHolder<Acknowledgment<Order>>) item;
          for (Acknowledgment<Order> orderAcknowledgment : acknowledgmentChunkHolder.getChunkedObjects()) {
  sendOrderAcknowledgmentService.sendOrderAcknowledgment(orderAcknowledgment);
          }
      }
  }
}
```

Listing 4

```
@Singleton
@Remote(IncomingOrderProcessingService.class)
public class TimerIncomingOrderProcessingService implements IncomingOrderProcessingService {

  private final static String ORDER_PROCESSING_JOB_NAME = "OrderProcessingJob";

  @Schedule(minute = "0,5,10,15,20,25,30,35,40,45,50,55")
  @Override
  public void startIncomingOrderProcessing() {

     JobOperator jobOperator = BatchRuntime.getJobOperator();

     /* Check if Job is already running. */
     if (this.jobAlreadyRunning(jobOperator)) {
        return;
     }

     Properties props = new Properties();
     props.setProperty("maxResults", "10");
     long jobExecutionId = jobOperator.start(ORDER_PROCESSING_JOB_NAME, props);
  }

  private boolean jobAlreadyRunning(JobOperator jobOperator) {
     List<Long> runningExecutions = jobOperator.getRunningExecutions(ORDER_PROCESSING_JOB_NAME);
     return runningExecutions != null && !runningExecutions.isEmpty();
  }
}
```

Listing 5

chunk.ItemProcessor" *(siehe „https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemProcessor.html")* implementiert. Diese Schnittstelle definiert die Methode „processItem", die immer ein Item abarbeitet und das Ergebnis der Verarbeitung zurückliefert *(siehe Listing 3)*.

Das gezeigte Beispiel simuliert das Aufrufen eines Systems, das eingehende Bestellungen entgegennimmt und als Ergebnis eine Bestätigung zurückgibt. Diese kann „positiv" oder „negativ" sein. Entsprechend wurde im Ergebnis die Bestellung angenommen oder abgelehnt.

Um den ersten Schritt zu vervollständigen, muss nun noch das Interface „javax.batch.api.chunk.ItemWriter" *(siehe „https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/ItemWriter.html")* implementiert werden. Hierfür kann, ähnlich wie bei der Implementierung von „javax.batch.api.chunk.ItemReader", die abstrakte Basisklasse „javax.batch.api.chunk.AbstractItemWriter" *(siehe „https://docs.oracle.com/javaee/7/api/javax/batch/api/chunk/AbstractItemWriter.html")* abgeleitet werden *(siehe Listing 4)*.

Im oben gezeigten Beispiel wird eine Fehlerbehandlung nur angedeutet. Im Praxiseinsatz muss auf eine korrekte Behandlung von Ausnahmesituationen geachtet werden. Definierte Exceptions können unter anderem für das Definieren von Ausnahmesituationen verwendet werden, die ignoriert werden können. Hierüber kann aber auch konfiguriert werden, wann eine Verarbeitung wiederholt werden muss. Details folgen im zweiten Teil des Artikels.

Nachdem diese Klassen implementiert wurden, ist der erste Schritt unserer Batch-Verarbeitung einsatzbereit.

Starten eines Batch-Jobs

Durch das Durchführen der Batch-Verarbeitung innerhalb eines Java-EE-Containers lässt sich ein Batch-Job auf unterschiedliche Arten starten; unter anderem ist die Nutzung von „TimerService" oder auch das Starten per HTTP-Request mithilfe eines Servlets möglich.

Im Beispiel ist eine SingletonBean *(siehe „http://docs.oracle.com/javaee/7/api/javax/ejb/Singleton.html")* implementiert. Sie enthält eine Schedule-Methode *(siehe „http://docs.oracle.com/javaee/7/api/javax/ejb/Schedule.html")*, die vom ausführenden Java-EE-Container in einem fest definierten zeitlichen Intervall aufgerufen wird. Innerhalb dieser Methode wird die Batch-Verarbeitung gestartet *(siehe Listing 5)*.

Innerhalb des Applikations-Servers Oracle WebLogic werden für das Starten von Batch-Jobs die Möglichkeiten der Executor-Services genutzt. Diese Verwendung zeigt der zweite Teil des Artikels in der nächsten Ausgabe.

Fazit

Der Artikel beschreibt das grundsätzliche Modell der Batch-Verarbeitung innerhalb von Java EE 7. Mit relativ wenig Aufwand ist das Erstellen einer solchen Verarbeitung möglich; vorhandenes Wissen aus dem Java-EE-Bereich kann eingebracht werden. Proprietäre Frameworks sind bei konsequenter Entwicklung gegen das angebotene API nicht mehr notwendig.

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.

Statische Code-Analyse – den Fehlern auf der Spur

Andreas Günzel, EXXETA AG

In der letzten Ausgabe wurden die Vorteile täglicher Code Reviews am Beispiel von Gerrit, Git und Jenkins vorgestellt. Darüber hinaus existiert eine Reihe weiterer Werkzeuge, um die Software-Qualität einer Anwendung zu messen und zu verbessern. Dieser Artikel stellt die gängigsten Open-Source-Tools zur Code-Analyse im Praxis-Einsatz vor.

Das Ziel der statischen Code-Analyse ist es, Fehler im Quelltext einer Anwendung zu finden. „Statisch" bedeutet in diesem Kontext, dass die Suche nach Fehlern durch Untersuchung des Bytecodes passiert. Somit muss – im Gegensatz zur dynamischen Analyse – das zu untersuchende Programm nicht ausgeführt werden. Diese Art der Code-Analyse erfolgt in der Regel mit Werkzeug-Unterstützung. Im Rahmen der statischen Code-Analyse wird der Programmcode einer Reihe formaler Prüfungen unterzogen. In diesem Zuge werden Metriken, zum Beispiel über die Lines of Code oder die zyklomatische Komplexität, berechnet und der Daten- und Kontrollfluss analysiert. Die Ergebnisse werden mit definierten Vorgaben und Grenzwerten verglichen und die Abweichungen protokolliert. Die gefundenen Probleme müssen letztendlich die Entwickler beseitigen.

Die Bandbreite der Prüfungen ist groß: Sie reicht von der Einhaltung von Coding-Standards bis zur Suche nach Speicherlecks oder Deadlocks. Einfache Analysen sind heute häufig bereits im Compiler integriert. Zusätzlich existieren allerhand Tools, auch „Code Analyzer" genannt, die sich auf verschiedene Fehlerarten spezialisiert haben. Die Vorstellung all dieser Werkzeuge würde den Rahmen dieses Artikels sprengen. Im Folgenden wird daher der Einsatz ausgewählter Open-Source-Lösungen gezeigt, die speziell für die Überprüfung von Java-Code ausgelegt sind.

Checkstyle

Checkstyle dient zur Prüfung der Einhaltung von Coding-Standards und ist somit kein Werkzeug zur direkten Erkennung von Programmierfehlern. Vielmehr werden unerwünschte Programmierstile aufgedeckt und bemängelt. Durch die Sicherstellung eines gut lesbaren Programmcodes unterstützt es die Entwickler jedoch indirekt dabei, Programmierfehler zu finden und zu vermeiden.

Die erste Version von Checkstyle wurde bereits im Jahr 2001 veröffentlicht. Das Tool hat sich seitdem stetig weiterentwickelt. Die aktuelle Version 6.10 umfasst rund 150 verschiedene Prüfregeln *[1]*. Diese sind nach Fehlerart in verschiedenen Modulen gruppiert. Sowohl Regeln als auch Module lassen sich nach Bedarf erweitern oder deaktivieren. Die wohl am häufigsten anzutreffenden Verstöße sind das Fehlen von geschweiften Klammern um Codeblöcke, die Verwendung von Konstanten oder die Sichtbarkeit von Variablen. Streiten lässt sich hingegen darüber, ob es sich beim Fehlen von Javadoc-Kommentaren oder einer Leerzeile am Ende der Class-Datei wirklich um einen Fehler handelt

Um den Programmcode mit Checkstyle zu prüfen, gibt es mehrere Möglichkeiten: manuelle Ausführung der Regeln auf der Kommandozeile, Integration mit einem Build-Management-Tool wie Apache Maven oder Ant oder die Einbindung als DIE-Plug-in für Eclipse, NetBeans oder IntelliJ. Für einen ersten Test kommt an dieser Stelle das Maven-Plug-in zum Einsatz. Dafür sind mehrere Arbeitsschritte erforderlich. Zunächst muss „maven-checkstyle-plugin" in der „pom.xml" aktiviert werden *(siehe Listing 1)*.

Um einen Checkstyle-Bericht für das Projekt zu erzeugen, genügt es nun, den Befehl „mvn site" in der Kommandozeile abzusetzen. Die Ergebnisse finden sich unter „target/site/checkstyle.html" beziehungsweise werden in der Übersicht der Projektberichte unter „target/site/project-reports.html" gelistet. Der erzeugte Bericht ist rein informativ und verpflichtet die Entwickler nicht dazu, sich an vorhandene Coding-Guidelines zu halten. Um dies zu erzwingen, kann das Plug-in so konfiguriert werden, dass der Build bei Verstößen als fehlerhaft markiert wird und abbricht *[2]*.

```
<reporting>
  <plugins>
     <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>2.16</version>
        <reportSets>
           <reportSet>
              <reports>
                 <report>checkstyle</report>
              </reports>
           </reportSet>
        </reportSets>
     </plugin>
  </plugins>
</reporting>
```

Listing 1

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-jxr-plugin</artifactId>
  <version>2.5</version>
</plugin>
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-pmd-plugin</artifactId>
  <version>3.5</version>
  <configuration>
     <minimumTokens>100</minimumTokens>
  </configuration>
</plugin>
```

Listing 2

PMD

Ähnlich wie Checkstyle dient PMD nicht dazu, Programmierfehler zu identifizieren. Der Schwerpunkt des Regelwerks liegt auf der Erkennung ineffizienter Stellen im Programmcode. Hierzu zählen zum Beispiel „Duplicate" oder „Dead Code" sowie leere Catch- oder Finally-Blöcke. Die gefundenen Verletzungen stellen in der Regel keinen Bug dar – haben jedoch hohes Potenzial, in Ausnahme-Situationen oder bei einem Refactoring zu einem zu werden.

Die aktuelle Version 5.3.4 unterstützt neben Java eine Reihe weiterer, teils exotischer Sprachen. So analysiert PMD auch JavaScript-, PL/SQL- oder Fortran-Code. Dabei beschränkt sich die Prüfung jedoch zumeist auf die Erkennung von Duplicate Code. Dies erledigt PMD durch das integrierte Modul „CPD". Somit liegt der Schwerpunkt erkennbar auf der Prüfung von Java-Code – hier existieren mehr als dreihundert Regeln.

Genau wie bei Checkstyle sind die Regeln logisch gruppiert. Einzelne Gruppen lassen sich bei Bedarf deaktivieren oder erweitern. Die Ergebnisse der PMD-Analyse können generell als sehr gut bezeichnet werden. Nur wenige der gefundenen Probleme lassen eine Diskussion über deren Sinnhaftigkeit zu. Typi-

```
<profile>
  <id>sonar</id>
  <properties>
      <sonar.host.url>
          http://localhost:9000
      </sonar.host.url>
  </properties>
  <build>
      <plugins>
          <plugin>
              <groupId>org.codehaus.sonar</groupId>
              <artifactId>sonar-maven-plugin</artifactId>
              <version>4.5.5</version>
          </plugin>
      </plugins>
  </build>
</profile>
```

Listing 4

sche Regelverstöße stellen das Fangen und Werfen generischer Exceptions, Klassen mit zu vielen Variablen oder die Instanziierung von Objekten innerhalb von Schleifen dar [3].

Die Möglichkeiten zur Einbindung von PMD in ein Entwicklungsprojekt entsprechen den bereits bei Checkstyle vorgestellten Ansätzen. Nur die Liste der unterstützten IDEs ist deutlich länger. Listing 2 erweitert die bereits vorgestellte Maven-Konfiguration innerhalb des „reporting"-Blocks um die Ergebnisse der PMD-Prüfung.

Nach der erneuten Ausführung des Befehls „mvn site" stehen in den Projektberichten nun drei neue Einträge. Der Report „PMD" listet alle Verstöße gegen das gleichnamige Regelwerk auf. Im „CPD"-Bericht werden alle Stellen mit Duplicate Code aufgeführt. Die Option „MinimumTokens" definiert dabei die Größe eines Codeblocks für die Erkennung einer Duplizierung. Der Default-Wert von „100" entspricht etwa fünf bis zehn Codezeilen und ist für die meisten Situationen gut gewählt.

Der Report „Source Xref" wird durch das optionale „maven jxr"-Plug-in erzeugt. Dieses enthält eine HTML-Repräsentation des Sourcecodes, mit deren Hilfe von einem PMD-Fehler direkt zur betroffenen Stelle im Programmcode verlinkt werden kann. Auch das PMD-Plug-in bietet die Möglichkeit, die verwendeten Regelwerke zu konfigurieren und bei Verstößen den Build abzubrechen [4].

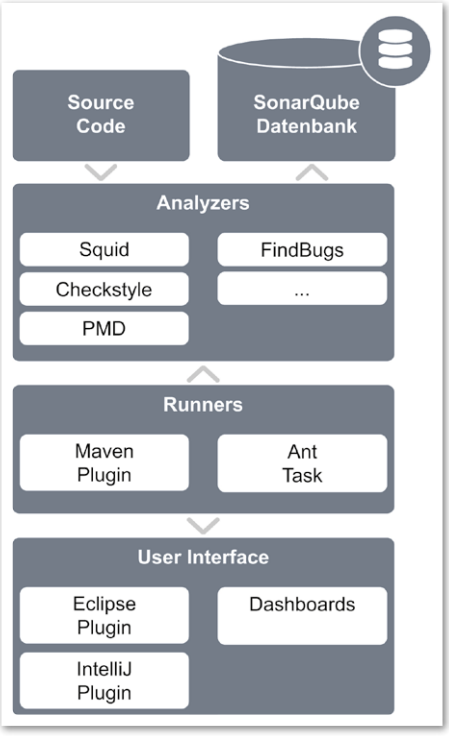


Abbildung 1: Die SonarQube-Architektur [7]

FindBugs

Das Tool „FindBugs" macht genau das, was sein Name impliziert: Es sucht nach potenziellen Fehlern im Programmcode. Im Rahmen der Analyse wird der kompilierte Bytecode dabei nach bekannten Fehlermustern durchsucht. Listing 3 zeigt einen klassischen Fehler, der mithilfe von FindBugs erkannt wird: Der Aufruf der Methode „trim()" soll unerwünschte Leerzeichen abschneiden. Dabei wurde vom Entwickler jedoch vergessen, der Variable „hello" den neuen Wert zuzuweisen.

Die aktuelle FindBugs-Version 3.0.1 erkennt mehr als vierhundert dieser Bug-Patterns. Häufig gefundene Probleme sind potenzielle Null-Pointer-Zugriffe, Endlosschleifen oder Deadlocks. Diese Art der Fehlerersuche hat jedoch eine Schwäche: Die Erkennung eines Fehlermusters deutet stets nur auf einen Fehler hin, es muss sich dabei nicht zwangsläufig um einen tatsächlichen Bug handeln. Auf der Homepage

```
String hello = " world ";
trim(hello);
if (hello.equals("world")) {
// ...
}

private String trim(String anyString)
{
  if (anyString == null) {
     return null;
  }
  return anyString.trim();
}
```

Listing 3

versprechen die FindBugs-Entwickler eine Fehlerrate von weniger als 50 Prozent [5].

Wie Checkstyle und PMD kann FindBugs über die Kommandozeile, als Eclipse- oder Maven-Plug-in angesprochen werden. Zusätzlich bietet es eine eigenständige GUI zur detaillierten Darstellung und Beschreibung der Bugs im Quellcode. Das Maven-Plug-in wird vom MojoHaus Project (früher Mojo@Codehaus) bereitgestellt, unterscheidet sich in der Handhabung jedoch nicht von den bereits vorgestellten Tools [6].

SonarQube: Ein Tool für alle Fälle

Auch wenn die drei bisher vorgestellten Tools unterschiedliche Ziele bei der Code-Analyse verfolgen, so ist die Konfiguration und Bedienung sehr ähnlich. Leider sind die über Maven erzeugten Berichte in der Praxis, insbesondere bei großen Projekten, nicht sonderlich gut auszuwerten. So sind alle gefundenen Verletzungen je Regelwerk auf nur einer HTML-Seite ohne Filterfunktion dargestellt. Eine übergreifende Bewertung der Kritikalität aller gefundenen Bugs fehlt ebenso. Zudem überschneiden sich bestimmte Regeln, so führen zum Beispiel sowohl Checkstyle als auch PMD eine Prüfung auf leere Catch-Blöcke durch.

Einen interessanten Ansatz zur Lösung der genannten Punkte bietet SonarQube, bis Mitte 2013 als „Sonar" bekannt. Hinter dem Namen verbirgt sich eine Plattform, die es ermöglicht, verschiedene Code-Analyse-Tools zu kombinieren, und die Ergebnisse in einer aufbereiteten Ansicht bereitstellt. Die Funktionsweise von SonarQube unterscheidet sich dabei deutlich von den bisher vorgestellten Tools. Abbildung 1 zeigt die beteiligten Komponenten bei der Durchführung des Analyse-Prozesses.

Die Unterstützung verschiedener Programmiersprachen, Code Analyzer oder

Visualisierungen ist in SonarQube mithilfe von Plug-ins möglich. Der Analyse-Prozess wird mithilfe eines sogenannten „Runners" gestartet. Dieser stellt eine Art Schnittstelle zwischen SonarQube und dem verwendeten Build-Management-Tool dar.

Der Sourcecode wird nun mithilfe verschiedener Code Analyzer geprüft und die Ergebnisse in einer eigenen Datenbank festgehalten. Neben den bereits vorgestellten Werkzeugen FindBugs, Checkstyle und PMD bringt SonarQube mit Squid ein eigenständiges Tool zur statischen Code-Analyse mit. Zudem wird unter anderem auch die Testabdeckung (per Default mit „JaCoCo") gemessen. Über ein Web-Frontend können die Ergebnisse von allen Entwicklern eingesehen werden.

Um einen ersten Eindruck von den Fähigkeiten der Plattform zu erlangen, stellt SonarQube eine Online-Demo bereit, in der verschiedene Open-Source-Projekte oder auch das JDK 7 analysiert werden *[8]*. Zur Überprüfung von eigenem Programmcode ist zunächst eine eigenständige Installation erforderlich. Unter Windows genügt es dabei, die aktuelle Version 5.1.2 von der Homepage herunterzuladen, zu entpacken und per Batch-Datei zu starten *[9]*.

Initial verwendet SonarQube eine „Embedded H2 Datenbank", der Webserver ist auf dem Port 9000 erreichbar. Dieses Setup ist jedoch nur für lokale Tests gedacht – für einen produktiven Einsatz sollte eine dedizierte Datenbank genutzt werden. SonarQube unterstützt hier unter anderem auch Oracle- und MS-SQL-Server. Um die Code-Analyse durchzuführen, wird wie in den vorherigen Beispielen Maven als Build-Management-Tool verwendet. In *Listing 4* ist dafür ein eigenes Maven-Profil definiert. Dies hat den Vorteil, dass die SonarQube-Konfiguration nicht über die „pom.xml" verstreut wird. Läuft der SonarQube-Server nicht lokal oder mit den Standardeinstellungen, muss dies ebenfalls in der Projekt-Konfiguration hinterlegt sein *[10]*.

Der Maven Runner wird über den Befehl „mvn sonar:sonar -Psonar" gestartet. Die Analyse-Ergebnisse stehen anschließend im Browser unter *„http://localhost:9000"* zur Verfügung. Per Default stellt das Projekt-Dashboard verschiedene Metriken wie die Lines of Code, Code-Duplizierungen und Probleme dar. Für alle Kennzahlen bietet SonarQube eine Drilldown-Funktion vom Package bis hin zur Methode. Zudem besteht die Möglichkeit, das aktuelle Analyse-Ergebnis mit vergangenen Prüfläufen zu vergleichen und einen Trend abzuleiten.

Das Regelwerk, das ein Projekt im Rahmen der Code-Analyse durchlaufen muss, ist in den sogenannten „Quality Profiles" definiert. Standardmäßig existiert ein vorkonfiguriertes Profil, in dem knapp zwei Drittel der dreihundert in SonarQube verfügbaren Regeln aktiviert sind. Die Regelwerke von PMD & Co. sind hier noch nicht beinhaltet und lassen sich mithilfe des Plug-in-Managers einfach nachinstallieren und aktivieren. Änderungen an den Regelwerken greifen dabei immer erst bei der nächsten Analyse.

Einsatz in der Praxis

Das initiale Setup der vorgestellten Werkzeuge ist teilweise in Minuten erledigt, die Einbindung in ein vorhandenes Continuous-Integration-System ebenfalls. Der erzielte Erfolg allerdings steht und fällt mit der Akzeptanz durch die Entwickler. Ein Team, das mit dem Gedanken an die Einführung eines Tools zur Code-Analyse spielt, sollte ein besonderes Augenmerk auf die folgenden typischen Fehler haben:

- Die vorgestellten Werkzeuge werden bevorzugt von Architekten eingeführt und zur Messung der Qualität genutzt. Dabei ist es wichtig, die Entwickler von Anfang an mit einzubeziehen. Die Tools müssen vorgestellt und der Mehrwert für das Team aufgezeigt werden. Das Ziel sollte sein, dass jeder Entwickler regelmäßig (oder gar bei jedem Commit) die Ergebnisse der Code-Analyse überprüft und bei Bedarf Code-Optimierungen vornimmt.
- Oftmals wird mit der Messung der Code-Qualität erst begonnen, wenn bereits ein Großteil der Entwicklung abgeschlossen ist. Die Behebung der dann gefundenen Probleme kann zu hohen Aufwänden führen, die das Team nicht neben der geplanten Weiterentwicklung der Anwendung leisten kann. Daher empfiehlt es sich, Tools zur Code-Analyse bereits zum Projektstart mit dem Team zu definieren und aufzusetzen.
- Insbesondere das Setup von SonarQube ist aufwändig und erfordert gegebenen-

falls separate Hardware. Für kleine Projekte mit einer vergleichsweisen kurzen Laufzeit und nur wenigen Entwicklern ist dieser Overhead in der Regel zu groß. Hier sollte sich im Team auf den Einsatz von Tools wie PMD, FindBugs oder Checkstyle verständigt werden.

- Die vollständige Aktivierung aller Regelwerke führt schnell zu einer hohen Anzahl an Verletzungen, von denen viele nur eine geringe Gewichtung haben. Dies macht es schwer, den Überblick zu behalten und die tatsächlichen Fehler zu identifizieren. Daher sollte das Regelwerk immer an das jeweilige Team und Projekt angepasst werden.
- Manche Projekte lassen den Build brechen, wenn die Code-Qualität unter einen bestimmten Schwellwert fällt. Dieser eigentlich ehrbare Ansatz ist jedoch mit Vorsicht zu betrachten. Bricht der Build unangekündigt durch eher zufällige Commits ab, führt dies schnell dazu, dass die Akzeptanz der Tools im Team sinkt. Ferner kann eine falsch gewählte Kennzahl, wie etwa die prozentuale Testabdeckung, zu Stilblüten wie dem wahllosen Testen von „equals()"-Methoden oder Interfaces führen.

Die Auswahl eines geeigneten Regelwerks hat sich in der Praxis als der vielleicht wichtigste Erfolgsfaktor beim Einsatz statischer Code-Analyse herausgestellt. Dabei haben sich zwei Strategien bewährt: Eine Möglichkeit besteht darin, zunächst ein möglichst großes Regelwerk zu aktivieren und die Ergebnisse der Prüfung zu bewerten. Regeln, die zu redundanten, unpassenden und unkritischen Verletzungen führen, sollte man schrittweise deaktivieren. Die Prüfung und Korrektur des Regelwerks ist dabei ein fortlaufender Prozess, der vor allem am Anfang einige Zeit in Anspruch nimmt. Auch darf sich das Team hier von den anfänglich vielleicht vielen Fehlern nicht abschrecken lassen.

Der zweite Ansatz geht den entgegengesetzten Weg. Zunächst wird ein minimales Regelwerk eingeführt, das zum Beispiel lediglich die Suche nach Dead Code oder leeren Codeblöcken beinhaltet. Nach der Beseitigung der ersten Probleme erweitert man das Regelwerk iterativ. Diese Vorgehensweise eignet sich vor allem bei Projekten mit vorhandenem Sourcecode. In jedem Fall ist dem Team bei der Planung Zeit zum Beseitigen der technischen Schulden einzuräumen.

Fazit

Am Markt existiert eine Reihe von Tools zur automatisierten Code-Analyse. Die Einbindung ist dank Maven-Plug-in oder Ant-Task in wenigen Schritten erledigt und die gefundenen potenziellen Probleme sind zumeist zahlreich. Für kurze Projekte sind kleine Tools wie PMD, FindBugs oder Checkstyle in der Regel ausreichend. Mit wachsender Komplexität im Programmcode führt dagegen kaum ein Weg an SonarQube oder einer vergleichbaren Plattform vorbei.

Der Einsatz dieser Werkzeuge hat sich bis heute nicht vollständig durchgesetzt. Dies liegt zum einen an der nicht unerheblichen Anzahl an falsch erkannten oder unwichtigen Regelverletzungen. Hier ist mit iterativem Feintuning das optimale Regelwerk zu ermitteln. Zudem müssen die gefundenen Probleme noch immer manuell gefixt werden. Folglich ist die Akzeptanz im Entwicklungsteam – der eigentlichen Zielgruppe – ein entscheidender Faktor. Richtig eingesetzt ist die statische Code-Analyse damit ein ausgezeichnetes Mittel, um zu einer durchgehend hohen Software-Qualität beizutragen.

Weitere Informationen

[1] *http://checkstyle.sourceforge.net/checks.html*
[2] *http://maven.apache.org/plugins/maven-checkstyle-plugin*
[3] *http://pmd.github.io/pmd-5.3.4*
[4] *http://maven.apache.org/plugins/maven-pmd-plugin/index.html*
[5] *http://findbugs.sourceforge.net*
[6] *http://gleclaire.github.io/findbugs-maven-plugin*
[7] Charalampos S. Arapidis, Sonar Code Quality Testing Essentials, Packt Publishing, 2012
[8] *http://nemo.sonarqube.org*
[9] *http://www.sonarqube.org*
[10] *http://docs.sonarqube.org/display/SONAR/Installing+and+Configuring+Maven*

Andreas Günzel
andreas.guenzel@exxeta.com

Andreas Günzel ist Principal Consultant bei der Exxeta AG in Frankfurt. Seine langjährigen Erfahrungen stammen aus verschiedenen Projekten bei Banken und Mittelstand. Er hat fachliche und technologische Schwerpunkte im Umfeld von Java Enterprise und agiler Software-Entwicklung

Oracle startet neues Cloud-Partner-Programm

DOAG Online

Oracle hat angekündigt, ein neues Cloud-Programm für seine Partner zu starten. Ziel des Programms sei die bessere Unterstützung beim Ausbau des Cloud-Geschäfts. Mitgliedern des Cloud-Partner-Programms verspricht Oracle, ihre Oracle-Cloud-Expertise besser vermarkten zu können sowie zusätzlichen Support und Unterstützung bei der Markteinführung. Es soll insgesamt vier neue Partnerstufen geben: Cloud Standard, Select, Premier und Elite. Die Qualifizierung für das Programm und die Einstufung des Partner-Levels werde dabei aufgrund von Cloud-Performance-Kennziffern vorgenommen und richte sich unter anderem nach dem Spezialisierungsgrad, der Anzahl der vertriebenen Applikationen, den erfolgreich implementierten Cloud-Lösungen sowie den umgesetzten Cloud-Projekten. Partner der höchsten Kategorie „Cloud Elite" sollen die höchste Priorität bei der Markteinführung und die breiteste Unterstützung des Herstellers erhalten.

Oracle wirbt mit dem aktuell umfassendsten Cloud-Portfolio im Markt und deckt sowohl die Bereiche SaaS, PaaS als auch IaaS ab. Die Partner-Community umfasse heute bereits zehntausende Implementierungs-Experten.

Modulare Web-Anwendungen mit Java – Theorie und Praxis

Jan Paul Buchwald, Freiberuflicher IT-Architekt und Software-Ingenieur

Modularisierung ist vor allem in großen Software-Projekten mit verteilten Teams essenziell, um die Wiederverwendbarkeit, Wartbarkeit und parallele Entwicklung zu verbessern. Im Bereich der Web-Anwendungen mit Java gibt es jedoch verschiedene und uneinheitliche Modularisierungsansätze, was in der Praxis oft zu ungenügender Modularisierung führt. Dieser Artikel gibt zuerst einen Überblick über die Historie und Gegenwart der Modularisierungsmöglichkeiten im Umfeld von Java-Web-Anwendungen und zeigt deren Schwächen auf. Mit pragmatischen Lösungen, Patterns und Best Practices kann man dem Ziel einer sauber modularisierten Architektur allerdings doch näherkommen.

Vermutlich ist jeder schon einmal in dem einen oder anderen Software-Projekt auf das Phänomen gestoßen, das mit der Metapher „Software-Monolith" beschrieben wird: Die Lösung wirkt wie ein in sich geschlossener großer Klotz, alles ist sehr eng verwoben und Änderungen an einer Stelle haben unvorhersehbare Auswirkungen an völlig anderer Stelle. Demgegenüber steht das Ideal der modularen Entwicklung, in der Lego-Bausteinen gleich die Gesamtlösung aus eigenständigen, standardisierten und wiederverwendbaren Komponenten zusammengesetzt wird.

Modularisierung verhilft zu besserer Wartbarkeit im Sinne von Verständlichkeit und Lokalität, Kombinierbarkeit, kürzeren Entwicklungszyklen sowie höherer Flexibilität durch parallele Entwicklung und Austauschbarkeit. Als Nachteil wird teilweise gesehen, dass die Komplexität auf System-Ebene steigt, da beispielsweise Abhängigkeiten explizit verwaltet werden müssen und ein gewisser Aufwand für die Entkopplung und spätere Integration der Komponenten erforderlich ist.

Modul-Definition und Module im Java-Kontext

Ein Software-Modul lässt sich als abgeschlossene funktionale Einheit einer Software definieren, die folgende Eigenschaften aufweist:

- Es existiert eine Schnittstelle nach außen
- Diese Schnittstelle kann versioniert werden
- Abhängigkeiten von anderen Modulen beziehen sich nur auf diese Schnittstelle
- Die Einheit kann eigenständig eingerichtet, verwaltet und getestet werden
- Die Einheit ist wiederverwendbar, mit anderen Modulen kombinierbar, und kann mehrfach instanziiert werden (zustandslos)

In Modulsystemen wird darüber hinaus typischerweise unterschieden zwischen dem Programmiermodell, also der Modul-Handhabung zum Entwicklungszeitpunkt, und dem Laufzeitmodell, also der Verwaltung

und dem Verhalten von Modulen in einer laufenden Software. Es stellt sich nun die Frage, wie sich eine solche Modul-Definition auf die Entwicklung von Web-Anwendungen mit Java abbilden lässt.

Von Seiten der Programmiersprache bieten im Wesentlichen Klassen und Packages einfache Strukturierungsmöglichkeiten, die jedoch zu feingranular und nicht abgeschlossen sind. Vom Nutzer aus betrachtet, werden komplette Anwendungen oder Dienste bereitgestellt, die in der Regel wiederum zu große Einheiten darstellen, als dass sie in Gänze wiederverwendet oder kombiniert werden können. Module sollten genau zwischen diesen beiden Ebenen angesiedelt sein, allerdings gibt es hier in Java keinen einheitlichen Standard. Im Folgenden werden deshalb verschiedene mögliche Ansätze für Module im Umfeld von Web-Anwendungen mit Java vorgestellt und bewertet.

Open Services Gateway Initiative (OSGi)

Auf der Suche nach Modularisierungs-Lösungen für Java stößt man als Erstes auf OSGi.

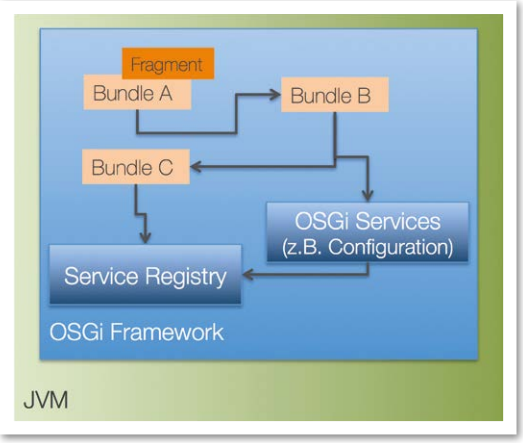


Abbildung 1: Schematische Darstellung OSGi

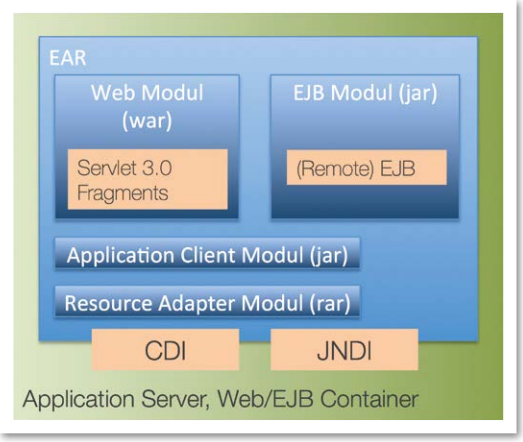


Abbildung 2: Module in Java EE

```
Bundle-Name: JavaAktuell
Bundle-SymbolicName: javaaktuell
Bundle-Description: A Java Aktuell bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0
Bundle-Activator: ja.Activator
Export-Package: ja.helloworld
Import-Package: ja.other;version="1.0"
```

Listing 1

Dieser Ansatz wurde bereits im Jahr 2000 primär für eingebettete Systeme eingeführt, seither regelmäßig weiterentwickelt und vor allem durch die Verwendung in der IDE und Rich Client Platform Eclipse bekannt.

Abbildung 1 zeigt eine schematische Darstellung der Bestandteile von OSGi: Das Framework bietet eine Service Registry und eine Reihe vordefinierter Services. Module werden als sogenannte „OSGi-Bundles" entwickelt, die im Wesentlichen angereicherte „jar"-Dateien mit zusätzlichen Einträgen zu den Bundle-Informationen und vor allem Schnittstellen und Abhängigkeiten in der Manifest-Datei darstellen *(siehe Listing 1)*.

Die Schnittstelle nach außen wird durch exportierte Packages deklariert, Abhängigkeiten auf andere Module durch Import-Deklarationen von Packages mit optionaler Versionsangabe. In neueren OSGi-Versionen kann dies teilweise auch über Annotationen (wie „@Component", „@Activate", „@Reference") realisiert werden. Damit können Bundles als Module mit deklarierten Schnittstellen, Version und Abhängigkeiten im OSGI-Framework eingerichtet werden und das Framework stellt die Abhängigkeit zur Laufzeit sicher. Als Erweiterung existiert das Konzept der Fragmente, die an bestimmte Bundles optional und austauschbar angehängt werden können. Beispiele für OSGi-Implementierungen sind ProSyst als kommerzielles Angebot oder Open-Source-Frameworks wie Equinox und Apache Felix.

Für Web-Anwendungen gibt es verschiedene Möglichkeiten, OSGi zu nutzen. Zum einen bieten OSGi-fähige Application-Server wie WebSphere, GlassFish, WebLogic oder WildFly die Möglichkeit, Web-Application-Bundles einzurichten. Dahinter steht eine Java-EE-Web-Anwendung („war") mit OSGi-Bundle-Deklarationen. Alternativ können reine OSGi-Bundles den OSGi-http-Service in einem http-fähigen OSGi-Container (wie Apache Karaf) nutzen, wobei diese sich meist im Hintergrund ebenfalls eines Application-Servers bedienen.

Insgesamt ist OSGi ein mächtiges Framework und erfüllt bei richtiger Anwendung alle anfänglich aufgezählten Merkmale von Modulen. Das Programmiermodell schränkt allerdings stark ein und es wird eine spezielle Laufzeitumgebung beziehungsweise ein Container innerhalb der JVM benötigt. Es gibt wenig vorhandene Bundles oder Beispiele und die Einstiegshürde ist entsprechend hoch. Vermutlich auch aus diesem Grund hat sich das Framework nicht so recht im Mainstream durchgesetzt und wird vor allem für Web-Anwendungen sehr selten eingesetzt. Die OSGi-Alliance hat dies auch selbst erkannt und versucht seit Kurzem, mit dem neuen Projekt „OSGi enRoute" *[1]* ein komplettes Anwendungs-Framework für OSGi zu etablieren.

Module in Java EE

Auch die Java-Plattform-Edition für Web-Anwendungen mit Java, Java EE, bietet das Konzept von Modulen: Wie *Abbildung 2* zeigt, ist in Java EE eine Enterprise Application („ear") per Definition in Module aufgeteilt, die gebräuchlichsten sind Web- und EJB-Module. Web-Module können seit der Version 3.0 der Servlet-Spezifikation auch aus Fragmenten (also quasi „Bausteinen") aggregiert werden, Remote EJBs lassen sich über Anwendungsgrenzen hinweg ansprechen und somit wiederverwenden.

Darüber hinaus gibt es noch Instrumente zur Entkopplung und Kapselung wie „CDI" oder auch dessen Vorgängerlösung „JNDI". Allerdings sind dies alles sehr spezielle und eher grob-granulare Modul-Definitionen, die eng an das Java-EE-Programmier-Modell angelehnt sind. Damit sind in der Praxis wiederverwendbare oder kombinierbare Bausteine oft nicht realistisch.

Als Mittel zur Realisierung von Wiederverwendbarkeit wird deshalb in Java-EE-Projekten oft auf Libraries in verschiedenen Ebenen zurückgegriffen. Normale „jar"-Dateien, etwa für fachliche Objekte oder Utility-Code, können in der Enterprise Application oder als „Installed

Libraries" auf Ebene des Application-Servers abgelegt und innerhalb der Anwendung oder global wiederverwendet werden. Hier besteht jedoch keine saubere Modul-Abgrenzung und es kommt häufig zu Versions- und Classloader-Konflikten, weshalb dies eher als pragmatische Notlösung in der Praxis einzustufen ist.

Modularisierung im Java Community Process

Auch im Java Community Process (JCP) gab es immer wieder Bestrebungen, ein Modulsystem in der Sprache selbst zu etablieren. Schon im Jahr 1999 wurde der JSR 8 für eine „Open Services Gateway Specification" formuliert, der allerdings dann zugunsten von OSGi als separate Initiative außerhalb der Kernsprache zurückgezogen wurde. Im Jahr 2005 wurde der JSR 277 „Java Module System" gestartet, in dem ein Format und ein Repository für eine Menge von Java-Code und Ressourcen definiert werden sollten. Hier wurde die Community sich allerdings nie einig, der JSR hat den Status „dormant".

Denselben Status hat auch der kurz darauf im Jahr 2006 gestartete JSR 294, in dem es um „Improved Modularity Support in the Java Programming Language" geht. Dort wurde im Wesentlichen eine Spracherweiterung um „superpackages" vorgeschlagen, was ebenfalls keine allgemeine Zustimmung fand. Im Jahr 2008 kam dann „Project Jigsaw" auf, in dem zunächst das JDK selbst besser modularisiert werden, als Nebenprodukt aber auch ein allgemeines Modulsystem für Java entstehen sollte. Die Modularisierung des JDK wurde jedoch weder in Java 7 noch in Java 8 realisiert und wird erst derzeit mit Java 9 angegangen.

Mittlerweile hat jedoch die Modularisierung des JDK Priorität und diese wurde zunächst ohne ein allgemeingültiges Modulsystem in mehreren Schritten durchgeführt: zuerst die Modularisierung des Sourcecodes mit JEP 201, dann die Restrukturierung der JDK- beziehungsweise JRE-Images sowie ein neues URI-Schema für Artefakte mit JEP 220 und schließlich die Auftrennung des JDK in Module mit JEP 200. Diese Bestandteile werden sehr wahrscheinlich in Java 9 umgesetzt sein, für die Modulauftrennung wurden vorerst aber nur minimale Annahmen zum Modulsystem getroffen in Form eines statischen „Modul-Graphs" als globale XML-Datei der JDK-Module und deren Abhängigkeiten *[2]*.

Für das allgemeine Modulsystem gibt es seit dem Jahr 2014 nun auch wieder einen offiziellen JSR 376 „Java Platform Module System", hier wurden allerdings erst vor Kurzem konkrete Vorschläge und Konzepte dokumentiert, die in der Community noch kontrovers diskutiert werden *[3]*. Sollte dies in Java 9 noch realisiert werden, stünde damit in der Programmiersprache selbst ein Modularisierungskonzept zur Verfügung. Mit den aktuellen Java-Versionen muss man sich jedoch anderer Ansätze bedienen.

Frameworks und Microservices

Eine weitere Möglichkeit, modulare Anwendungen zu entwickeln, ist der Einsatz von Frameworks, die selbst Modulkonzepte oder ähnliche Möglichkeiten der strukturierten Wiederverwendung von Bausteinen mitbringen. Beispiele hierfür sind Spring (etwa Spring Beans/Container, Spring MVC oder Web Flow, Spring AOP, Spring Dynamic Modules), Play, Sling, oder das Wisdom-Framework. Teilweise verwenden diese Frameworks auch einen OSGi-Unterbau. Nachteilig ist, dass man sich hier stark an eine konkrete Technologie bindet und diese Art von Modulen mit anderen Frameworks im Normalfall nicht verwendbar ist.

Schließlich gibt es noch den Microservices-Ansatz, bei dem die Modularisierung auf Anwendungs- beziehungsweise Service-Ebene stattfindet. Dabei werden die Anwendungen oder Services so feingranular geschnitten und als eigenständige, miteinander kombinierbare Komponenten modelliert, dass diese selbst als Module angesehen werden und die Notwendigkeit eines Modulkonzepts auf Sprachebene entfällt. Diese Art der Modularisierung hat jedoch starke Auswirkungen auf die Architektur des Gesamtsystems und es gibt aktuell kein übergreifendes, standardisiertes Modulsystem für Microservices im Sinne der Verwaltung von Abhängigkeiten und Schnittstellen-Versionierung.

Theorie und Praxis

Wie die bisherige Betrachtung gezeigt hat, sind die Modularisierungsansätze im Java-Umfeld und vor allem im Web-Bereich in der Theorie unvollständig oder haben klare Schwächen. Dennoch ist es von Vorteil, die Möglichkeiten zu kennen und sich je nach Kontext einzelner Elemente und Prinzipien der beschriebenen Ansätze zu bedienen. Beachtet man darüber hinaus noch gewisse Patterns und Best Practices, sind damit in der Praxis durchaus funktionierende pragmatische Lösungen möglich. Im Folgenden werden nun einige dieser Patterns und Best Practices beschrieben.

Modularity Maturity Model und Modularity Patterns

Zunächst ist es wichtig, in einem konkreten Projektkontext zu definieren, welcher Grad an Modularisierung sinnvoll ist. Graham Charters (IBM) hat dazu auf einem OSGi Community Event im Jahr 2011 ein sogenanntes „Modularity Maturity Model" mit sechs Stufen aufgestellt *(siehe Abbildung 3)*. Hier empfiehlt es sich zu bewerten, auf welcher Stufe sich das Projekt aktuell befindet, und als Ziel zu setzen, welche Stufe in einem bestimmten Zeitraum erreicht werden sollte. Auf dieser Basis können dann konkrete Maßnahmen getroffen werden. Für viele Projekte dürften die Stufen 3 oder 4 bereits große Vorteile bedeuten, Stufen 5 und 6 sind eher die „Kür" der Modularisierung, die man unter Kosten-Nutzen-Gesichtspunkten bewerten sollte.

Eher in den Bereich der Binsenweisheiten gehört die Erkenntnis, dass bei der Umsetzung und Durchsetzung von Modularisierungsmaßnahmen sehr entscheidend die Einstellung und das Wissen der Entwickler eine Rolle spielen. Diese lassen sich möglicherweise durch Patterns in die

Stufe	Bezeichnung	Beschreibung
1	Ad Hoc	Keine Modularisierung, globaler Classpath
2	Module	Formale Modul-Identitäten entkoppelt von Artefakt
3	Modularität	Formale Schnittstellen zwischen Modulen
4	Loose Coupling	Trennung Interface und Implementierung: Services, semantische Versionierung von Abhängigkeiten
5	Befugnisübergabe	Zentrale Modul-Repositories („Baukasten"), optional mit Collaboration- und Governance-Funktionen
6	Dynamik	Dynamischer Lebenszyklus von Modulen, Betriebsunterstützung für Hinzufügen, Entfernen, Ersetzen

Abbildung 3: Modularity Maturity Model nach Graham Charters (IBM)

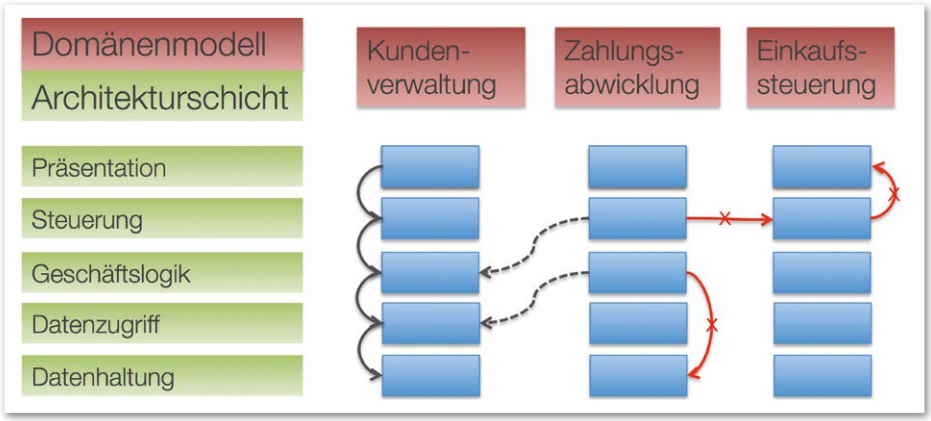


Abbildung 4: Modul-Zuschnitt nach fachlichen Domänen und Architektur-Schichten

gewünschte Richtung lenken. Eine gute Vorlage für solche Patterns liefern die „Modularity Patterns" von Kirk Knoernschild *[4]*. Diese beinhalten eine Reihe von nützlichen Vorlagen und Denkanstößen, etwa zu Abhängigkeiten, Erweiterbarkeit, Code „Usability", Exception Handling bis hin zu Build und Test. Beispiele für solche Patterns sind „Physical Layers" (Modul-Beziehungen sollten nicht die konzeptionellen Schichten verletzen), „External Configuration" (Module sollten von außen konfigurierbar sein) oder „Separate Abstractions" (Abstraktionen und implementierende Klassen sollten in separaten Modulen liegen).

In vielen Fällen tritt auch ein Lerneffekt durch Negativ-Beispiele in Form von Anti-Patterns ein. Hier lassen sich auch im Modularisierungsbereich Dinge finden, die immer wieder falsch gemacht werden. Beispiele sind etwa das Problem der Übergeneralisierung von Schnittstellen, falsche oder unnötige Abstraktionen und nicht benötigte Wiederverwendbarkeit („NeverReusedReusableModule"). Außerdem läuft man bei jeder Modul-Abgrenzung Gefahr, entweder einen zu feingranularen Modul-Schnitt vorzunehmen und dann der Anzahl an Modulen nicht mehr Herr zu werden oder zu große Module zu implementieren und damit wieder kleine Monolithen heranzuzüchten.

Modul-Zuschnitt und Abhängigkeiten

Als Hilfestellung für den Modul-Zuschnitt ist es sinnvoll, eine größere Lösung sowohl fachlich als auch basierend auf dem Schichtenmodell der Software-Architektur zu betrachten und entsprechend Module zuzuordnen. *Abbildung 4* zeigt ein Beispiel, in dem drei fachliche Domänen und fünf Architektur-Schichten definiert sind. Damit lässt sich nun eine Matrix aufspannen, in der jeder Eintrag ein Kandidat für ein Modul ist. Darüber hinaus können Vorgaben oder Regeln dafür definiert werden, welche Abhängigkeiten zwischen den Modulen erlaubt sind.

Im Beispiel ist festgelegt, dass Abhängigkeiten immer nur zwischen benachbarten Schichten und nur von der oberen auf die untere Schicht erlaubt sind. Dies gilt im Ausnahmefall auch Fachdomänen-übergreifend (gestrichelter Pfeil). Nicht erlaubt sind hingegen Abhängigkeiten innerhalb derselben Schicht, Abhängigkeiten von darüberliegenden Architekturschichten sowie Abhängigkeiten, die eine Schicht überspringen.

Als Erweiterung dieses Ansatzes kann auch noch die technische Bedeutung einer Komponente berücksichtigt werden, ob also diese zum Beispiel eher allgemeine Utility-Funktionen, Plattform-Funktionen oder fachliche Logik enthält. Damit sind auch Querschnittsthemen berücksichtigt, für die ebenfalls eigene Module oder Modul-Typen und entsprechende erlaubte und nicht erlaubte Abhängigkeiten definiert werden können.

Die Modul-Abhängigkeiten lassen sich dann idealerweise automatisiert über Tools mit geeigneten Metriken visualisieren oder mit Architektur-Constraints abprüfen. Beispiele aus der Vielzahl von Tools in diesem Bereich sind „jdeps", das Maven-Dependency-Plug-in, JarAnalyzer oder SonarQube. Falls im Projekt ohnehin Code-Reviews durchgeführt werden, sollte auch die Prüfung der Modul-Abgrenzung und der Abhängigkeiten zu einem solchen Review gehören. Mit Tools oder Reviews lassen sich dann Modul-Zuschnitt und Abhängigkeiten regelmäßig prüfen und bereinigen, um ein erneutes Abdriften unsauber abgegrenzter Module in Richtung eines Monolithen zu verhindern.

Fazit

Insgesamt bleibt festzuhalten, dass leider kein allgemein verwendbares Modulsystem für Java-Web-Anwendungen existiert. Es gibt mit OSGi, Java EE, Frameworks und Microservices jedoch wirksame Modularisierungs-Ansätze, deren Elemente man sich bei Bedarf auch einzeln bedienen kann. Bei der Umsetzung einer Modularisierungsstrategie ist es hilfreich, sich am Modularity Maturity Model zu orientieren, Modularisierungs-Patterns beziehungsweise Anti-Patterns zu beherzigen, und ein Regelwerk für den Modul-Zuschnitt und erlaubte Abhängigkeiten aufzustellen.

Weitere Informationen

[1] OSGi enRoute: *http://www.osgi.org/Main/OSGiEnRoute*
[2] JEP 200: The Modular JDK: *http://openjdk.java.net/jeps/200*
[3] JSR 376: *http://openjdk.java.net/projects/jigsaw/spec/*
[4] Kirk Knoernschild's Modularity Patterns: *http://www.kirkk.com/modularity/*

Jan Paul Buchwald
mail@j-pb.com

Jan Paul Buchwald ist nach Stationen in der Software-Entwicklung und Beratung bei IBM Deutschland Research&Development und BeOne Stuttgart seit Anfang 2014 als selbstständiger IT-Architekt und Software-Ingenieur tätig. In den mehr als zehn Jahren seiner bisherigen Berufstätigkeit konnte er in zahlreichen internationalen Projekten für große Unternehmenskunden vielfältige Erfahrungen aus allen Phasen der Software-Entwicklung mit Java beziehungsweise Java EE sammeln, überwiegend im Bereich portal- und webbasierte Lösungen, aber auch Rich-Client-Anwendungen.

DukeCon – das Innere der JavaLand-App

Gerd Aschemann, DukeCon.org

Bei der JavaLand 2015 dachten sich einige Teilnehmer aus der JUG Darmstadt, dass zu einer User-Konferenz auch eine eigene mobile Anwendung gehört. Hehre Visionen führten zu der vollmundigen Ankündigung, für das Jahr 2016 eine solche Applikation zu bauen. Wünsche wie Offline-Fähigkeit, Twitter-Integration oder Vortragsbewertungen sollten umgesetzt werden. Ein knappes Jahr später ist ein guter Zeitpunkt, das bisherige Ergebnis zu beschreiben und sich der Bewertung der Community zu stellen.

Ein kleines Team fand sich schnell zusammen; schon am Rande der JavaLand 2015 wurden Anforderungen zusammengestellt *[1]*, darunter:

- *Offline-Fähigkeit*
  Im Phantasialand ist die Netzanbindung oft suboptimal. Das Konferenz-Programm sollte auch ohne Zugriff auf die Web-Seiten der DOAG über unsere mobilen Helfer abrufbar sein, am besten noch mit den persönlichen Favoriten.
- *Feedback zu Veranstaltungen*
  Am Ende eines Vortrags sollte eine einfache Bewertung stehen, etwa durch Klick auf eine Ampelfarbe (rot, gelb, grün).
- *Twitter-Aktivitäten*
  Tweets zu „#javalandconf" sollten eingeblendet und verschickt werden können.

Der aufmerksame Leser stellt hier schon gewisse Widersprüche fest. Wir waren also auf dem besten Wege, ein typisches IT-Projekt ins Leben zu rufen. Dazu passten optimistische Annahmen wie: „Da bauen wir einen Rest-Service und einen HTML5-Client, mit Cordova ist das dann ganz flott eine echte App".

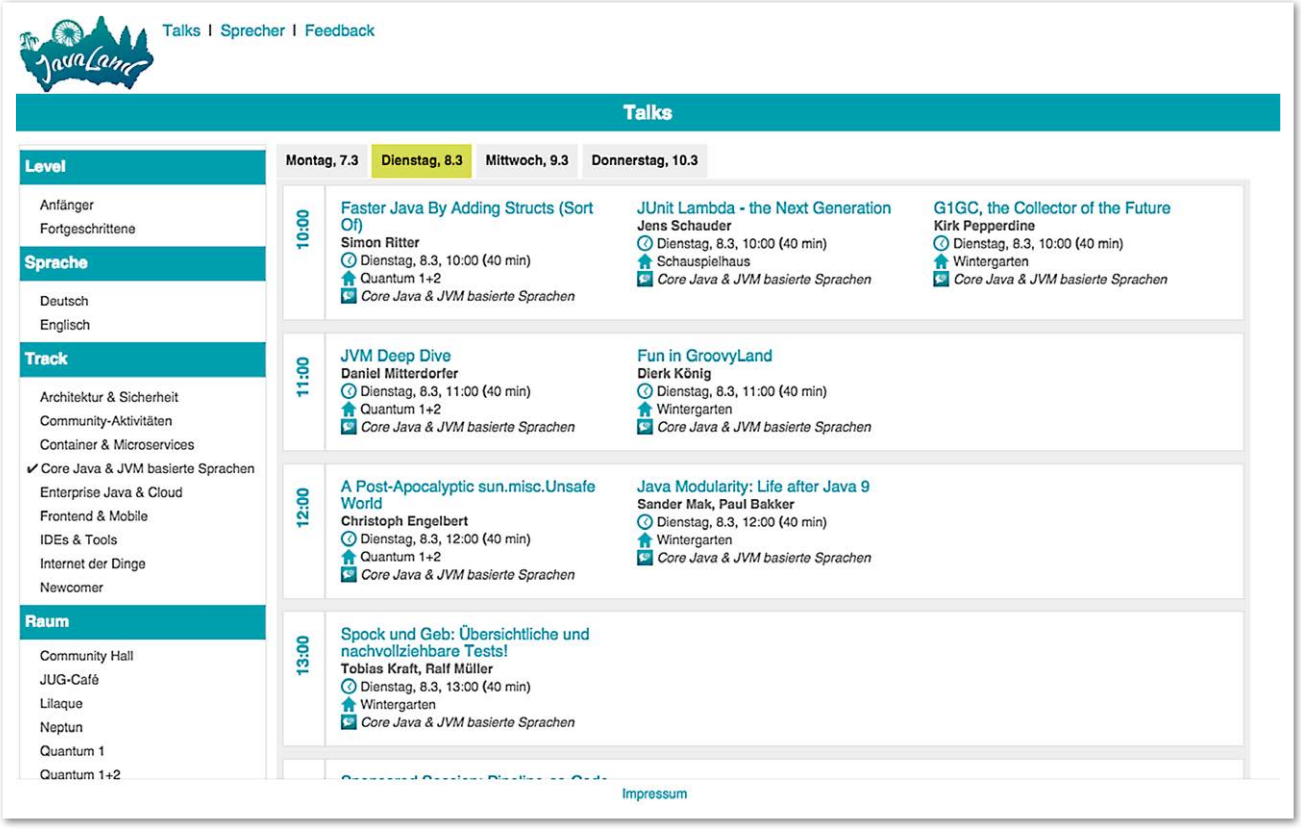


Abbildung 1: Release 1.0 zur Programmübersicht

Kurz nach der Konferenz fanden wir uns zusammen und begannen quasi am ersten Tag mit dem Kodieren. Wir hatten also nicht nur ein echtes IT-Projekt, sondern sogar ein agiles. Auch ein Name für das Projekt war schnell gefunden: „DukeCon". Glücklicherweise konnten wir bald einige Mitglieder der JUG Kaiserslautern und noch weitere einzelne Mitstreiter dazugewinnen. Aktuell gibt es ein Kern-Team von sechs EntwicklerInnen und eine Reihe weiterer Personen, die uns bei verschiedenen Aspekten mit Rat und Tat zur Seite stehen.

Go-Live im September: Pünktlich zum Programm 2016

In der Tat ließ sich eine erste Version als HTML5-Anwendung mit REST-Service für die Daten recht schnell zusammenstellen. Die DOAG als Organisator der JavaLand *[9]* fütterte uns zunächst mit Konferenzdaten von 2015, deren (JSON-)Struktur im Wesentlichen auch für 2016 erhalten blieb. Das UI-Design orientiert sich am bestehenden Programm-Planer, wurde aber bald in einigen Punkten angepasst, so zum Beispiel mit responsivem Design *[10]*.

Ende September 2015 stand das Programm der JavaLand 2016 fest; wir konnten eine erste Version bereitstellen: DukeCon ging als Release 1.0 an die Öffentlichkeit *[8] (siehe Abbildung 1)*. *Abbildung 2* zeigt bereits das Responsive Design, die Filter-Leiste kollabiert bei einer niedrigeren Bildschirmauflösung.

Release 1.1 *(siehe Abbildung 3)* wurde schon in einigen Punkte überarbeitet: Vor allem ist es damit möglich, Favoriten für die persönliche Programmplanung zu setzen sowie diese über mehrere Geräte zu synchronisieren. Außerdem ist das Format etwas kompakter, die Navigation wurde auf kleinen Geräten zu einem sogenannten „Hamburger Menü" und die Filter wurden durch Checkboxen klarer erkennbar.

Architektur und Organisation: Weil wir es können!

„Eine Architektur ist die Menge der signifikanten Entscheidungen über die Organisation eines Systems …" *[2]*. Jede Software hat eine Architektur, auch wenn diese nicht explizit geplant ist. Die Architektur von DukeCon folgt sowohl einem Anti-Pattern aus *[4]*: „Wir machen jetzt <Name-der-bevorzugten-Technologie>" als auch einem Standard-Muster: dem Erstellen der Architektur in Zyklen und Iterationen.

Im ersten Zyklus wollten wir uns bekannte und vor allem in diesem Kontext sinnvolle Technologien einsetzen, aber auch das eine oder andere ausprobieren. Für ein Green-Field-Projekt mit sehr überschaubarem Inhalt hatten und haben wir die Freiheit, die genutzten technischen Komponenten nach persönlichen Vorlieben, Kenntnissen und Möglichkeiten zu wählen und zu gestalten. Ein paar Rahmenbedingungen waren schnell abgesteckt:

- Die Daten standen uns als regelmäßig aktualisierter Download aus dem Back-Office der DOAG bereit, wir mussten die-

Abbildung 3: Das Release 1.1 sieht schon etwas moderner aus

Abbildung 2: Responsives Design des Release 1.0

se nur noch mit einem kleinen REST-Service veredeln.
- Die Implementierung sollte in Groovy [13] auf Basis von Spring Boot [11] erfolgen.
- Ein erstes Frontend wurde in HTML5 entwickelt, parallel dazu entstand als zweiter Client eine echte mobile App auf Basis von Flex.
- Wir nutzten Maven [14] als Build-Tool und begannen mit Travis-CI [11] für Continuous Integration. Bald brauchten wir aber eine eigene Build-Infrastruktur: Jenkins [15] plus Nexus [21].

Abbildung 4 zeigt DukeCon im Kontext von Benutzer und DOAG (als Datenlieferant). Daraus entwickelte sich schnell ein recht ansehnlicher Technologie-Stack (siehe unten). Parallel dazu mussten wir unsere Zusammenarbeit organisieren. Für beides bietet die Developer-Cloud genügend Möglichkeiten:

- Das (virtuelle) Team kommuniziert über eine Google Group/Mailingliste [5], Skype [24] als täglichen Chat und GoToMeeting [25] für regelmäßige Telefonkonferenzen (Dank an die DOAG).
- Mit Google Drive [23] koordinieren wir lebendige Dokumente und Tabellen, also Besprechungsprotokolle (Agenda sammeln, Ergebnisse darstellen etc.).
- Code und Dokumentation werden mit Git [37] verwaltet und auf GitHub gehostet [6], in der Regel folgen wir dem Gitflow-Workflow [40].
- Virtualisierte Linux-Server beherbergen mittlerweile sowohl die Continuous-Integration-Infrastruktur wie auch Test- und Produktions-Systeme [7].

Technologie-Stack

Der verwendete Technologie-Stack und die wohlbegründeten Architektur-Entscheidungen im Einzelnen: Der monolithische Backend-Server von DukeCon wurde mit Spring Boot [12] in Groovy [13] realisiert:

- Er holt die Vortragsdaten von der DOAG, speichert sie zwischen (Cache) und bereitet sie für die Clients auf.
- Mit Hystrix [29] stellen wir sicher, dass eine Nicht-Verfügbarkeit des DOAG-Servers abgefangen wird.
- Favoriten (persönliche Programmplanung der Teilnehmer) speichert der Server in einer PostgreSQL DB [22].
- Die Klassen des Domänenmodells bleiben dank Lombok [38] und Jackson Annotations [39] übersichtlich.
- Alle Services sind per REST/JSON API ansprechbar.

Aktuell gibt es zwei aktiv gepflegte Frontends und zwei Technologie-Studien:

- Ein Client ist in HTML5 [31] mit JavaScript auf Basis von Knockout [20] realisiert. Das ist unsere Referenz-Implementierung.
- Das andere Frontend ist mit Flex [16] umgesetzt und wird als native iOS- beziehungsweise Android-App ausgeliefert. Diese zweite Implementierung bot auch die Chance, das DukeCon-REST-API zu verifizieren.
- Für eine Einbettung der HTML5-Anwendung in Cordova [18] gibt es erste Prototypen.
- Ein Mitstreiter hat zudem eine Umsetzung mit Meteor [32] (Client + REST-Server) realisiert.

Das HTML5-Frontend wird als eigenständiges Projekt (Git/Maven) gepflegt und gebaut sowie als WAR-Overlay [33] über das Backend an den Browser ausgeliefert. Der Backend-Server wird als Docker-Image [19] erstellt, um ihn für Entwickler bereitzustellen und ohne Änderung über die verschiedenen Test-Stages bis in die Produktion ausrollen zu können.

Als Single-Sign-on-System mit direkter Benutzer-Registrierung bzw. Social-Login-Anbindung (GitHub, Google, Twitter und andere) verwenden wir KeyCloak [17], ebenfalls in einem Docker-Container. Die Speicherung der Daten in DukeCon und KeyCloak erfolgt über PostgreSQL [22], natürlich auch als Docker-Container.

Als Build-System kommt Maven [14] zum Einsatz, Jenkins [15] als Build-Server sowie Nexus [21] als Repository-Server für Artefakte. Tests werden mit Jasmine [34] (Unit/Frontend), Spock [36] (Unit/Backend) und JGiven [35] (Integration/End-To-End) automatisiert. Abbildung 5 zeigt die Hauptkomponenten von DukeCon im Zusammenspiel mit externen Social Media Clouds für die Benutzer-Authentifizierung über KeyCloak.

Eine große Herausforderung: die Offline-Fähigkeit

Die Verwaltung der Vortragsdaten und Favoriten funktioniert in jedem Client anders. Mit Flex wurde eine native App realisiert, die Daten lokal halten und bei Bedarf mit dem Backend synchronisieren kann. HTML5 bietet grundsätzlich ähnliche Möglichkeiten. Mit IndexedDB und Web Storage sollte sich ebenfalls eine Offline-Fähigkeit realisieren lassen, zumindest in der Theorie. In der Pra-

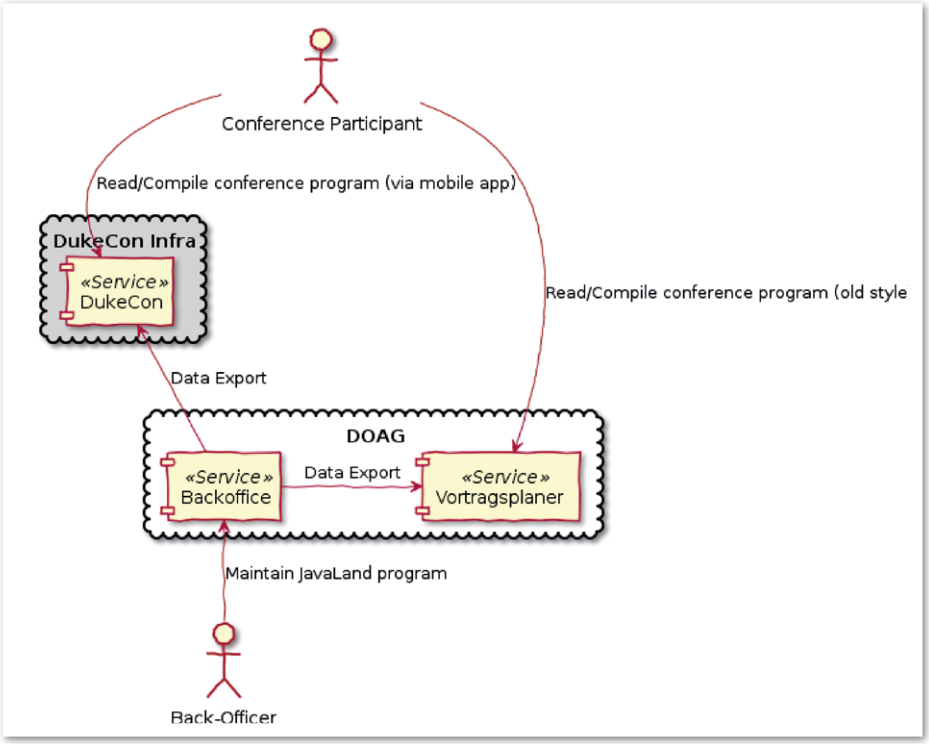


Abbildung 4: System-Kontext von DukeCon

xis musste eine ganze Reihe von Aspekten berücksichtigt werden.

HTML5 sieht eine Manifest-Datei vor, unter anderem für das Caching (Name frei wählbar, bei uns „cache.manifest"). Das Caching von Code/Ressourcen wie HTML-, JS- und Bild-Dateien ließ sich problemlos festlegen, aber wie sieht es mit dem Caching von Daten (Vortragsdaten) und vor allem dem Caching des Manifests selbst aus? Man könnte vielleicht annehmen, dass ein Browser prüft, ob sich die Cache-Datei geändert hat, um dann gegebenenfalls den Cache zu invalidieren. Aber wie prüft er das, und wann? Leider legt sich der W3C-Standard für HTML5 dazu nicht fest und entsprechend haben die Browser-Hersteller hier ihre Freiheitsgrade genutzt. Nach zahlreichen Try-and-Error-Zyklen über verschiedene Browser auf verschiedenen Plattformen haben wir Folgendes festgestellt:

- Ob sich das Manifest geändert hat, entscheiden Browser nicht etwa anhand von Inhalten oder eines Änderungs-Datums aus GET- oder HEAD-Requests. Ein ETag wäre auch hilfreich. Aber tatsächlich scheinen Kommentare in der Datei ausschlaggebend zu sein. Für manche Browser beziehungsweise Browser-Versionen braucht es dazu dann noch das Schlüsselwort „rev", gefolgt von einer geänderten Nummer beziehungsweise einem Zeitstempel.
- Die Browser prüfen anhand der gesetzten Header, ob sich das Manifest geändert hat beziehungsweise ob überhaupt auf eine neue Version geprüft werden muss. Leider verhalten sie sich alle ein bisschen anders, manche prüfen auf aktuelle (Cache-Control: no-cache, max-age=0), andere auf historische (Pragma: no-cache) Header.

Listing 1 zeigt unsere Cache-Manifest-Datei mit einem Platzhalter im Kommentar, der mithilfe des Maven-Build-Number-Plug-ins *[41]* gefüllt wird.

Kleine Fehler machen dabei das Leben zusätzlich schwer: Wenig hilfreich war beispielsweise bei der Fehlersuche, dass Chrome in den Developer Tools anzeigt, dass Daten aus dem Cache bedient wurden, auch wenn eine Netzwerk-Analyse bewies, dass diese bei einem geänderten Manifest zunächst korrekterweise vom Server geholt wurden. Merke: Traue niemals deinem Debugger.

Bezüglich der Vortragsdaten haben wir unser eigenes Caching im Browser imple-

```
CACHE MANIFEST
# rev ${buildNumber}

CACHE:
# Html
feedback.html
index.html
speakers.html
talk.html
impressum.html
# Css
css/dukecon.css
css/pure-min.css
css/side-menu.css
# Images
img/ajax-circle.gif
...
```

Listing 1

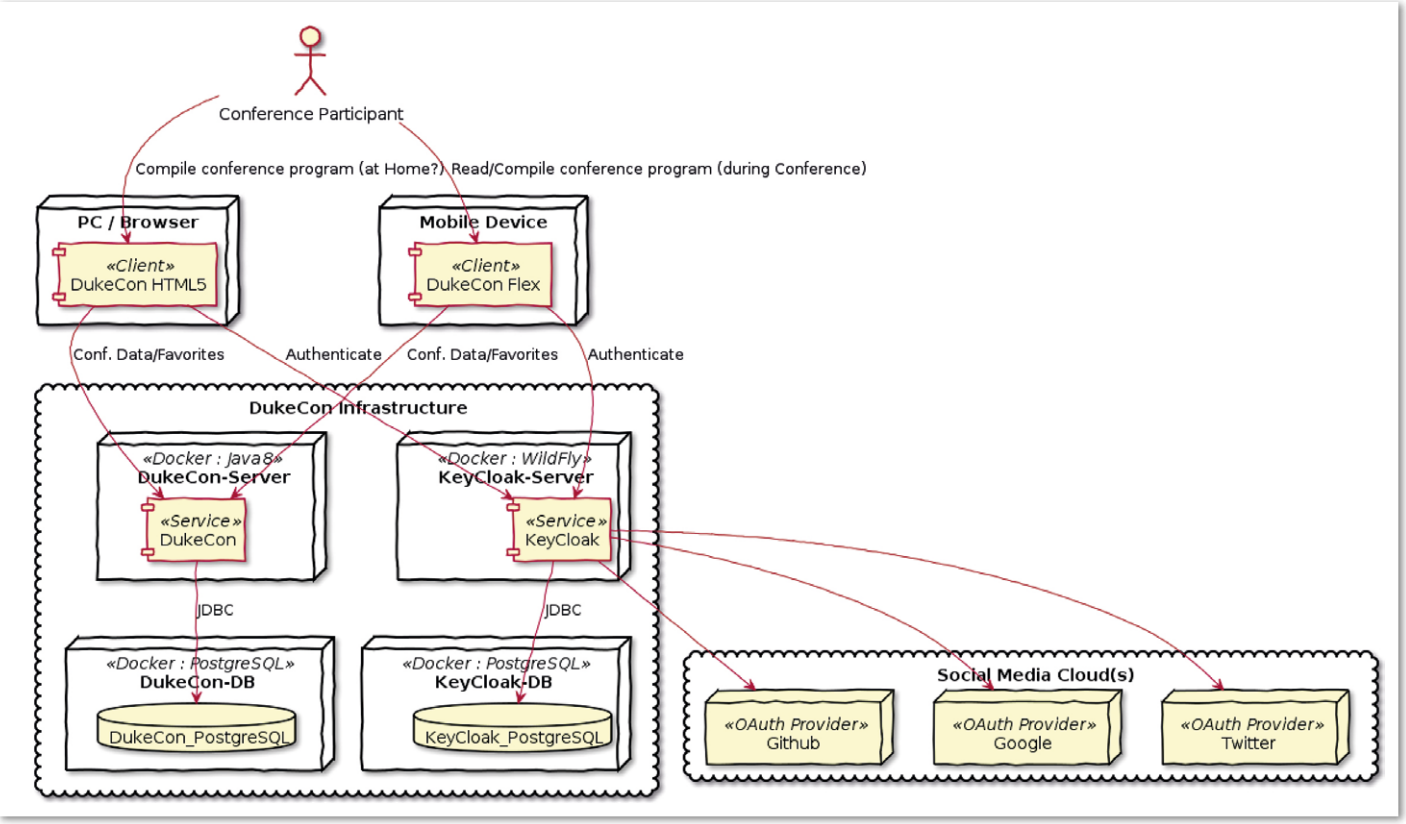


Abbildung 5: DukeCon-Architektur als White-Box-Komponenten-Diagramm

mentiert. Die Daten werden in einer lokalen Instanz der IndexedDB abgelegt. Mithilfe eines ETag wird auf Änderungen geprüft und entsprechend der interne Zustand (Knockout) neu berechnet. Erst danach wird die Anzeige aktualisiert.

Fachliche und technische Wunschliste

Das Projekt sieht auf den ersten Blick sehr übersichtlich aus. Der Aufwand, aus der ersten Idee und dem Prototyp ein stabiles Produkt zu machen, war allerdings nicht unerheblich. Dementsprechend sind durchaus noch ursprüngliche Anforderungen offen, wie etwa die Bewertung von Vorträgen. Dies ist nicht nur eine technische Fragestellung, sondern auch eine organisatorische. Wie stellt man beispielsweise sicher, dass die Vorträge nicht bereits vorher bewertet werden? Wie erreicht man gleichzeitig, dass sie erst zum Ende der offiziellen Vortragszeit für die Bewertung freigegeben werden? Wie verhindert man Mehrfachbewertungen durch eine Person und wahrt gleichzeitig die Anonymität? Ist Letzteres nur gegenüber dem Sprecher beziehungsweise anderen Auswertenden relevant oder generell? Muss man gar damit rechnen, dass Vorträge von Konkurrenten herab- oder von Freunden hochgesetzt werden? Kann man das überhaupt erkennen beziehungsweise vermeiden? Sollen Vortrags-Bewertungen auch offline abgegeben und synchronisiert werden können? Sollte es eine Aufforderung an die Teilnehmer geben, eine Bewertung abzugeben? Wird diese Aufforderung nur aufgrund der persönlichen Favoriten gesteuert? All diese Fragen sind sicher keine unlösbaren Probleme, müssen aber zunächst durchdacht und diskutiert werden.

In der heutigen vernetzten Zeit erwartet man nicht nur detaillierte Informationen über die Veranstalter und Sprecher, sondern auch die Möglichkeit, mit ihnen zu kommunizieren. Als Kanäle kommt neben Twitter und sozialen Netzwerken auch die mittlerweile schon altmodisch anmutende E-Mail in Betracht. Sprecher möchten sich über verschiedene Wege repräsentieren können oder ansprechbar sein. Alle diese Informationen könnten der Community später nützlich sein, wenn Usergroups Speaker zu lokalen Vortragsabenden einladen wollen. Die Realisierung eines entsprechenden Service steht noch aus.

Auch Teilnehmer haben möglicherweise weitere Wünsche. So sollten beispielsweise persönliche Filter oder Speicher-Optionen für Favoriten gepflegt werden können. Das Back-Office würde sich sicher über eine Auswertung der Favoriten freuen, um einen Trend der Vortragsbesucher erkennen und Raumbelegungen entsprechend einplanen zu können.

Was will ich werden, wenn ich einmal klein bin?

Wir haben bereits einen Monolithen gebaut. Was läge also näher, daraus nun Microservices zu extrahieren *[26]* oder eher „Right Sized Services" *[28]*, um dem neuesten Hype gerecht zu werden. Nicht nur technologisch bieten sich mehrere Teil-Komponenten an:

- Download/Synchronisation von Vortragsdaten mit dem Back-Office und deren Aufbereitung
- Ablage von Teilnehmer-Präferenzen und -Favoriten
- Die avisierte Pflege von Sprecher-Daten und Vortragsbewertungen

Ob es dazu kommt, hängt nicht zuletzt von der anhaltenden Begeisterung des Teams *[27]* und dem Gewinnen weiterer Helfer ab. Es ist ein Open-Source-Projekt: Liebe iJUG- und Java(Land)-Community, helft, es zu verkleinern. Ob es gelingt, mag das Publikum in ein paar Monaten entscheiden, spätestens aber auf der JavaLand 2017.

Disclaimer

Das Projekt entwickelt sich in Schüben, alles fließt („Panta Rhei" *[30]*). Zum Erscheinungsdatum des Artikels hat sich möglicherweise schon wieder einiges geändert. Der Autor übernimmt keine Gewähr dafür, dass der zu Papier gebrachte Status von Mitte Januar 2016 noch irgendetwas mit dem Zustand des Projekts im März oder April zu tun hat – willkommen in der Realität eines Software-Projekts.

Referenzen

1. Brainstorming JavaLandApp, Brühl 2015, interne Projektdokumentation *[23]*
2. Grady Booch, zitiert nach *[3]*
3. Matthias Bohlen; *http://mbohlen.de/was-genau-ist-softwarearchitektur*
4. Gernot Starke, Effektive Software-Architekturen, 5. Auflage, 2011
5. DukeCon Google-Group: *https://groups.google.com/forum/#!forum/dukecon*
6. DukeCon auf GitHub: *https://github.com/dukecon*
7. *http(s)://dev.dukecon.org*
8. *https://dukecon.org/javaland*
9. *http://javaland.eu*
10. *https://de.wikipedia.org/wiki/Responsive_Webdesign*
11. *http://travis-ci.org/dukecon/dukecon_server*
12. Spring Boot (Pivotal): *http://projects.spring.io/spring-boot*
13. Groovy (Apache): *http://groovy-lang.org*
14. Maven (Apache): *http://maven.apache.org*
15. Jenkins (CloudBees): *http://jenkins-ci.org*
16. Flex (Apache): *http://flex.apache.org*
17. KeyCloak (Redhat): *https://keycloak.jboss.org*
18. Cordova (Apache): *https://cordova.apache.org*
19. Docker: *https://www.docker.com*
20. Knockout.JS: *http://knockoutjs.com*
21. Nexus: *http://www.sonatype.com/nexus*
22. PostgreSQL: *http://www.postgresql.org*
23. Google Drive: *https://drive.google.com*
24. Skype: *http://www.skype.com/de*
25. GotoMeeting: *http://www.gotomeeting.com/*
26. Martin Fowler, Monolith *First: http://martinfowler.com/bliki/MonolithFirst.html*
27. DukeCon GitHub Team: *https://github.com/orgs/dukecon/people*
28. Weronika Łabaj, Goodbye microservices, hello right-sized services: *http://particular.net/blog/goodbye-microservices-hello-right-sized-services*
29. Hystrix: *https://github.com/Netflix/Hystrix*
30. *Panta Rhei - alles fließt, zitiert nach https://de.wikipedia.org/wiki/Panta_rhei*
31. HTML5: *https://www.w3.org/TR/html5*
32. Meteor: *https://www.meteor.com*
33. Maven WAR Overlay: *https://maven.apache.org/plugins/maven-war-plugin/overlays.html*
34. Jasmine: *http://jasmine.github.io*
35. JGiven: *http://jgiven.org*
36. Spock: *https://github.com/spockframework*
37. Git: *https://git-scm.com*
38. Lombok: *https://projectlombok.org*
39. Jackson Annotations: *https://github.com/FasterXML/jackson-annotations*
40. Gitflow Workflow: *https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow*
41. Maven Build Number Plug-in: *http://www.mojohaus.org/buildnumber-maven-plugin/usage.html*

Gerd Aschemann
gerd@dukecon.org

Gerd Aschemann ist freiberuflicher Software-Entwickler und -Architekt mit einem Faible für Continuous Delivery. An dieser Stelle möchte er dem DukeCon-Team für die Arbeit des vergangenen Jahres danken, es hat viel Spaß gemacht mit Euch: Anna Maier, Stephanie Peters, Falk Sippach, Christofer Dutz, Alexander Schwartz, Niko Köbler und einigen Weiteren, die uns gelegentlich mit Anregungen und Rat und Tat zur Seite gestanden haben.

Groovy und Grails – quo vadis?

Falk Sippach, OIO Orientation in Objects GmbH

Das Jahr 2015 begann turbulent für die beiden bekanntesten Projekte des Groovy-Universums. Die daraus entstandenen Unsicherheiten haben die neuen Funktionen der Releases 2.4 (Groovy) beziehungsweise 3.0 (Grails) ziemlich in den Hintergrund gedrängt. Dabei sind die Projekte lebendiger denn je und vor allem schon längst reif für den produktiven Einsatz. Der Artikel analysiert den aktuellen Stand und wagt einen Ausblick in die Zukunft. In der nächsten Ausgabe folgt in einem zweiten Teil ein Blick auf die wichtigsten und interessantesten Neuerungen der vergangenen Releases.

Der bisherige Herausgeber Pivotal hatte im Januar 2015 angekündigt, sowohl Groovy als auch Grails nicht mehr zu unterstützen. Viele haben sich damals gefragt, wie und ob es überhaupt weitergeht. Dem Team rund um Groovy wurden durch die Schließung der Codehaus-Plattform zudem weitere Steine in den Weg gelegt. Dank einer starken Community und einer gewissen Reife haben sich beide Projekte durch diese Probleme aber nicht entmutigen lassen und konnten sogar gestärkt aus der Situation hervorgehen.

Ruhig geworden

In den vergangenen Jahren war es bereits deutlich ruhiger um Groovy und Grails geworden. Die Gründe dafür sind vielfältig. Einerseits sind beide Projekte mittlerweile mehr als zehn Jahre alt. Die Welt ist jedoch nicht stehen geblieben und andere interessante Sprachen und Frameworks haben die Aufmerksamkeit in Fachmagazinen oder auf Konferenzen auf sich gezogen. Hinzu kam eine scheinbar fehlende Mainstream-Tauglichkeit im Falle von Groovy (Stichwort: „dynamische Typisierung"), wodurch ein Großteil der Java-Entwickler unnötigerweise abgeschreckt wurde. Die vielen Vorteile von Groovy und Grails gingen da zuletzt leider unter.

Am Beispiel der Gardner-Hype-Kurve *[1]* kann man erkennen, dass es vielen Technologien so ergeht. Nach dem Gipfel der Erwartungen in den Anfangsjahren folgt meist das Tal der Ernüchterung, wenn die breite Masse merkt, dass man nicht alle Vorteile haben kann, ohne auch die einen oder anderen Konsequenzen in Kauf zu nehmen. Typischerweise erholt sich die Kurve schließlich, wenn die ernsthaft Interessierten den Pfad der Erleuchtung beschreiten und sich die Technologie auf einem gewissen Niveau (der Produktivität) etabliert. Bei Groovy ist es schwer, diesen Verlauf mit Jahreszahlen zu versehen. Aber darauf kommt es auch nicht wirklich an. Die Tendenz lässt sich auf jeden Fall nachvollziehen.

Aktuell bekommen wir mit dem Thema „Microservices" wieder ein schönes Beispiel für einen solchen Verlauf geliefert. Obwohl der Ansatz bereits einige Jahre alt war, begann mit dem Artikel von Martin Fowler und James Lewis Anfang 2014 ein regelrechter Hype (technologischer Auslöser). Aber bereits ein Jahr später machte sich Ernüchterung breit. Auch bei Microservices gibt es keinen „Free Lunch" und die Umsetzung passt nicht zu jeder Unternehmensform („Conway's Law"). Zudem sind die Nachteile nicht trivial und müssen irgendwie behandelt werden, insbesondere die verteilte Natur der Services und die daraus resultierenden Folgen. Trotzdem werden sich Microservices für ausgewählte Szenarien als adäquater Architektur-Stil durchsetzen. Überprüfen können wir diese These allerdings frühestens in ein paar Monaten, wenn nicht sogar Jahren. Übrigens, wer jetzt schon auf den Microservices-Zug aufspringen möchte, findet mit Grails ein bereits vollkommenes Framework für die Umsetzung vor.

Dass es um Groovy ruhiger geworden ist, kann man auch gut an ReGina sehen. Es handelt sich hier nicht um die Schwiegermutter eines Groovy-Core-Entwicklers. Vielmehr kam im Januar 2007 (kurz nach Groovy 1.0) ein Buch heraus, das bis heute die Bibel der Groovy-Enthusiasten ist: „Groovy in Action". Im Zeitalter der maximal 140 Zeichen bei Twitter musste eine Abkürzung gefunden werden, was zu „GinA" führte. Zwei Jahre später überlegten sich die Autoren, die Neuerungen aus zwei Minor-Releases in eine zweite Auflage einfließen zu lassen. Anfangs war man optimistisch, aber erst sechs Jahre und viele Twitter-Nachrichten später (Suche nach „#regina", „#groovy" beziehungsweise „#groovylang") konnte man im Juli 2015 diese zweite Auflage (Wortspiel: „ReGinA") in den Händen halten.

Oberflächlich betrachtet, schienen die Autoren nicht mehr genug Enthusiasmus an den Tag zu legen. Aber weit gefehlt, vielmehr war Groovy als moderne, sich stetig weiterentwickelnde Sprache der eigentliche Grund für die lange Verzögerung. Durch die hohe Häufigkeit der Releases ergab sich ständig neuer Input, den man auch noch im Buch verarbeiten wollte. Man kämpfte also tapfer dagegen an, dass das Buch bei der Veröffentlichung nicht schon wieder veraltet war. Das Pivotal-Debakel Anfang 2015 kam den Autoren um Dierk König somit gerade recht, ergab sich dadurch doch endlich mal eine Zwangspause. Dank des Early-Access-Programms des Verlags konnte man zudem bereits lange eine vorläufige Fassung kaufen und bekam außerdem die Aktualisierungen regelmäßig frei Haus geliefert.

Historie

Ein Blick in die Geschichte zeigt, dass beide Projekte immer eng verbunden waren. In den Jahren 2003 respektive 2005 erblickten sie das Licht der Welt. Während sich bei Groovy die beiden maßgeblichen Initiatoren James Strachan und Bob McWhirter relativ schnell wieder zurückzogen, sind bei Grails die Gründer bis heute aktiv. Und auch bei Groovy fanden sich rasch neue Mitstreiter, die auch heute noch die Geschicke des Projekts mitbestimmen.

Es wurde früh versucht, die Open-Source-Entwicklung über eine Firma (G2One) abzusichern, die sich wiederum mit Schulungen und Beratungsdienstleistungen finanzieren sollte. SpringSource, die Firma hinter dem Spring-Framework, übernahm aber schon kurz darauf G2One und damit Groovy und Grails in sein Open-Source-Portfolio. Wenig später wurde SpringSource dann Teil der Firma VMware, die vor vier Jahren seine Open-Source-Entwicklungsabteilung in die Firma Pivotal ausgründete. Groovy und Grails haben also schon eine bewegte Geschichte hinter sich. Trotz aller Probleme und Widerstände konnten über all die Jahre immerhin mehrere Entwickler in Vollzeit an den Projekten arbeiten. Jochen Theodorou hat die Geschehnisse aus seiner Sicht in einem nicht unkritischen Blog-Beitrag zusammengefasst [2].

Cedric Champeau hat in seinem Blog-Post „Who is Groovy?" [3] die Geschichte aus einem anderen Blickwinkel betrachtet und anhand der Commits im Sourcecode-Repository nachvollzogen. Interessanterweise ist der fleißigste Entwickler, Paul King, gar nicht von Pivotal bezahlt worden – wiederum ein Zeichen für eine starke Community.

Beide Projekte entwickelten sich ständig weiter und insbesondere Grails wurde für Groovy der Innovationstreiber. Auch nach den Versionen 1.0 Ende 2006 beziehungsweise 2008 kam es jährlich mindestens zu Minor-Releases und damit zu stetigen Verbesserungen und natürlich auch jeder Menge neuer Funktionen.

Jähes Ende

Anfang des Jahres 2015 kam nun der gravierende Einschnitt. Das Management von Pivotal suchte Kosten-Einsparpotenzial, man wollte sich auf die Kern-Kompetenzen (Cloud, Virtualisierung) konzentrieren. Dabei gerieten Groovy und Grails auf die Streichliste, möglicherweise als Alibi aus Ermangelung an anderen Alternativen. Vermutlich dürfte sich die Einsparung dieser sechs Gehälter im Gesamtbudget kaum bemerkbar gemacht haben. Stattdessen hat man nun den Einfluss über Groovy und Grails verloren, obwohl beide Projekte mittlerweile stark in der Spring(-Boot)-Welt verankert sind.

Groovy 2.4 und Grails 3.0 sind die letzten Major-Releases unter dem Sponsorship von Pivotal: „The decision … is part of Pivotal's larger strategy to concentrate resources on … its growing traction in Platform-as-a-Service, Data, and Agile development … Pivotal has determined that the time is right to let further development … be led by other interested parties … who can best serve the goals …" [4]

Im Nachhinein betrachtet war der Zeitpunkt aber gar nicht schlecht gewählt. Immerhin waren beide Projekte mittlerweile sehr ausgereift und haben seit Langem einen festen Platz in der Open-Source-Welt eingenommen. Pivotal hatte außerdem eine Gnadenfrist von etwa drei Monaten zugestanden. Die bereits angekündigten Releases konnten so noch fertiggestellt werden. Für die bereits gekündigten Entwickler trotzdem keine leichte Zeit, da man nebenher nach einem anderen Sponsor oder einer neuen Arbeitsstelle suchen musste.

Wechsel zu Apache

Im Fall von Groovy verschärfte sich die Situation letztlich noch. Der bisherige Infrastruktur-Anbieter Codehaus (Mailinglisten, Jira, Homepage etc.) musste sich der erdrückenden Marktmacht von GitHub beugen und schaltete seine Systeme im Frühjahr 2015 ab. Auch wenn sich das schon längere Zeit angekündigt und man bereits Vorkehrungen getroffen hatte, diese Ereignisse fielen einfach unglücklich zusammen. Als Alternative wurde sehr ausführlich über den Anschluss an eine Open Source Foundation debattiert. Apache, Eclipse und Software Freedom Conservancy standen zur Auswahl [5]. Nach Abwägen der Vor- und Nachteile entschied man sich für die Apache Software Foundation. Mit der Aufnahme in den dortigen Inkubator mussten allerdings bestimmte Anforderungen und Konventionen erfüllt werden. Das bedeutete neben dem Umzug der Infrastruktur (Issue Tracker, Code-Repo, Mailinglisten) auch Umstrukturierungen und Anpassungen an den Prozessen und am Projekt selbst.

Im Sommer konnten mit 2.4.4 und 2.4.5 zwar schon erste kleinere Bugfix-Releases unter der neuen Schirmherrschaft herausgebracht werden, wirklich neue Funktionalität gab es aber seit 2.4 verständlicherweise nicht mehr. Immerhin ist mittlerweile wieder ein Großteil der ursprünglichen Committer an Bord. Auch da gibt es bei Apache gewisse Prozesse und Formalitäten, die es einzuhalten galt. Ende 2015 standen die Aussichten bereits gut, den Inkubator bald verlassen zu können. Als vollwertiges Apache-Projekt wird Groovy vermutlich nochmal mehr Aufmerksamkeit bekommen. Die steigenden Download-Zahlen im Jahr 2015 deuten schon darauf hin. Der Wechsel zu Apache hat dem Projekt also definitiv gutgetan.

Grails zu OCI

Das Grails-Team, das bis kurz vor dem Ende der Zeit bei Pivotal hart an dem großen 3.0er-Release-Sprung arbeitete, gönnte sich zunächst einige Wochen der Ruhe. Aber bereits im Mai konnte mit OCI Inc. ein neuer Hauptsponsor präsentiert werden, der direkt zwei der Haupt-Entwickler auf seine Lohnliste gesetzt hatte. Zudem verstärkte OCI seine Bemühungen im zweiten Halbjahr und stellte weitere Grails-Committer an. Man ist also sehr stark an einer Weiterentwicklung interessiert und versucht, hauptsächlich über Beratungsdienstleistungen und Projektgeschäft mit Grails die Ausgaben zu finanzieren.

Auch wenn sich für die Projekte letztlich alles zum Guten gewendet hat, so haben die Umstrukturierungen auch menschliche Schicksale betroffen. Natürlich sind mittlerweile alle untergekommen. Aber ein Teil musste sich umorientieren. Von den ursprünglich sechs Pivotal-Angestellten können nur Graeme Rocher und Jeff Scott Brown weiter Vollzeit an ihrem Projekt (Grails) entwickeln. Cedric Champeau (Groovy) und Lari Hotari (Grails) sind zu Gradle gewechselt und bleiben so immerhin dem Groovy-Universum treu. Ihre Zeit für Groovy beziehungsweise Grails wird sich jedoch aufgrund der großen Aufgaben bei Gradle in Grenzen halten. Der ehemalige Groovy-Projektleiter Guillaume Laforge wird auch weiterhin als Botschafter und nun Apache PMC Chair zur Verfügung stehen, er hat sich beruflich aber mittlerweile der Firma Restlet zugewandt. Die längste Aus- und Bedenkzeit hatte sich Jochen Theodorou genommen. Seit Kurzem arbeitet er nun bei der Schweizer Firma Canoo, die sich in den vergangenen Jahren im Umfeld von Groovy und Grails bereits einen Namen gemacht hat.

Im zweiten Teil dieses Artikels werden wir einen genaueren Blick auf die Eigenschaften und Neuerungen der letzten Groovy- und Grails-Versionen werfen. Für die Eiligen und die Entscheider wollen wir an dieser Stelle aber schon mal ein Fazit ziehen und den Ausblick in die Zukunft wagen.

Ausblick

Nach den politischen Querelen waren viele zunächst verunsichert. Aber beide Projekte sind definitiv noch nicht reif für den Framework-Friedhof. Vielmehr sind sie gestärkt aus den Geschehnissen hervorgegangen, dank der bereits stabilen Codebasis und der großen Community. Dazu passt auch die Aussage von Guillaume Laforge: „Here to stay. Independence. Community above all."

Dem Einsatz von Groovy und Grails steht also auch in Zukunft nichts im Wege. Aber wie bei allem ist es sinnvoll, die Stärken und Schwächen abschätzen zu können und entsprechend für die Einsatz-Szenarien abzuwägen. Man hat immer eine Wahl und sollte ganz bewusst entscheiden. Groovy eignet sich beispielsweise für Skripte, das Schreiben von Tests, die Verwendung als DSLs, Zugriffe über eine Admin-Konsole in Enterprise-Systemen und natürlich auch im Zusammenhang mit Grails, stellvertretend für viele andere Bibliotheken/Frameworks aus dem Groovy-Ökosystem.

Grails wurde immer als reines Web-Framework abgestempelt. Allerdings sollte man es nicht unterschätzen, es deckt schließlich den ganzen Bereich von der Persistenz bis zum Frontend ab und bietet durch seine Modularität viel Flexibilität. Man hat immer die Möglichkeit, auf bestehende Plug-ins zurückzugreifen oder eigene zu entwickeln. Grails eignet sich von jeher auch gut für das Prototyping, weil man in kurzer Zeit sehr viel erreicht. Firmen sollen schon Ausschreibungen gewonnen haben, weil sich der Kunde die Umsetzung aufgrund des Prototyps bereits sehr gut vorstellen konnte.

Als klassische Web-Anwendung empfehlen sich aus eigenen, etwas leidvollen Erfahrungen aber eher kleine Intranet-Anwendungen. Es gibt jedoch auch diverse Erfolgsgeschichten zu Umsetzungen von größeren Projekten mit Grails. Eine wichtige Voraussetzung dafür ist, dass man ein Team zusammenhat, das die Stärken und Schwächen von Groovy und Grails kennt und damit umgehen kann. Alle müssen am gleichen Strang ziehen, was bei einem Mischmasch aus Groovy-affinen und klassischen Java-Entwicklern meist nicht der Fall ist.

Mit Grails kann man beispielsweise genauso gut auch ein schlankes REST-Backend implementieren, auf dem dann SPA/RIA-JavaScript-Clients aufsetzen. Und da Grails seit Version 3.0 auf Spring Boot aufsetzt, eignet es sich auch wunderbar für die Entwicklung der verschiedenen Ausprägungen von Microservices.

Roadmaps

Ein Blick auf die aktuellen Roadmaps zeigt, dass sich Groovy und Grails erstmals in ihrer langen Geschichte auseinanderbewegen. Während man bei Grails mittlerweile wieder sehr positiv und mit vielen Ideen in die Zukunft blickt, versucht man sich beim Groovy-Projekt nach der Aufnahme bei Apache zunächst mal zu sammeln. Hier macht sich bemerkbar, dass das Grails-Team zum größten Teil zusammenbleiben konnte und weiterhin von einer Firma finanziert wird, die aus wirtschaftlichen Gründen auch an einer raschen Weiterentwicklung interessiert ist. Am Groovy-Projekt hingegen arbeitet derzeit niemand Vollzeit, die drei bisherigen Core-Entwickler sind bei unterschiedlichen neuen Arbeitgebern untergekommen.

Zum Glück ist der Bedarf bei Groovy aber auch nicht ganz so hoch. In den letzten Jahren ist bereits eine mächtige JVM-Sprache entstanden und es fehlen keine wichtigen Show Stopper. Die noch im Jahr 2014 ausgegebenen größeren Ziele, die Neukonzeption des Meta Object Protocol (MOP) basierend auf Invoke Dynamic und die Re-Implementierung der Sprachgrammatik in Antlr v4, wurden nach dem Abschied von Pivotal zunächst auf Eis gelegt. Ganz typisch Open Source wird natürlich trotzdem weiterentwickelt. Auf der Developer-Mailingliste herrscht reger Betrieb und für die kommende Version 2.4.6 wurden auch schon wieder etwa siebzig Tickets abgeschlossen.

Während man bei Groovy also zunächst kleinere Brötchen bäckt, gibt es bei Grails aufbauend auf dem letzten großen Release 3.0 noch einiges zu tun. Schon im Januar 2016 folgte Version 3.1 mit ausgebauten Profile-Support und zum Beispiel Profilen für REST und AngularJS. Außerdem wurde die Persistenz-Schicht GORM komplett überarbeitet und es werden jetzt neue Versionen von Hibernate, MongoDB, Neo4j und Cassandra unterstützt. Bereits im zweiten Quartal soll dann schon Version 3.2 mit weiteren Verbesserungen am REST bzw. AngularJS Support und am Grails Object Relational Mapping (GORM) folgen, unter anderem für nicht-blockierende Aufrufe. Anschließend will man wieder in den jährlichen Release-Prozess wechseln. Grails 3.3 ist für 2017 angekündigt, unter anderem mit Non Blocking IO, basierend auf Netty. Dass sich OCI viel vorgenommen hat, sieht man auch daran, dass weiterhin bezahlte Mitstreiter für das Grails-Projekt gesucht werden.

Fazit

Alles in allem scheinen die erzwungenen Änderungen beiden Projekten gutgetan zu haben, wenn auch auf unterschiedliche Art und Weise. Bei Grails ist es anscheinend optimal verlaufen – es gibt weiterhin eine Firma, die vor allem ein wirtschaftliches Interesse am Fortbestehen des Frameworks hat. Hoffen wir, dass sich OCI nicht verhebt und Grails lange begleiten wird.

Für Groovy sah es zunächst düsterer aus. Aber gerade der Wechsel zur Apache Foundation wird dem Projekt in Zukunft viele Türen öffnen und neue Nutzer bescheren. War man bisher zu stark von der Politik einer einzelnen Open-Source-Firma abhängig, hat man nun eine starke unabhängige Gesellschaft mit sehr gutem Ruf im Rücken. Die Popularität scheint damit jetzt schon gewachsen zu sein, wenn man Statistiken wie dem Tiobe-Index (von Platz 80+ mittlerweile in die Top 20 *[6]*) oder den offiziellen Downloadzahlen (2015 mehr als verdoppelt im Vergleich zum Vorjahr) Glauben schenken mag.

Positiv ist natürlich, dass neben Grails noch einige weitere, vor allem schon etablierte Bibliotheken und Frameworks im Groovy-Umfeld existieren (wie Spock, Gradle und Ratpack). Hinzu kommt, dass auch viele Java-Bibliotheken auf Groovy als DSL oder Skriptsprache setzen. Groovys besondere Features (etwa Closures oder Builder) und die ausdrucksstarke Syntax scheinen zudem Treiber für andere etablierte (Java 8) oder neue Sprachen (Swift) gewesen zu sein.

Obwohl das Interesse bei Konferenzen und in Magazinen im Vergleich zur Anfangszeit natürlich spürbar zurückgegangen ist, existieren noch einige kleinere Veranstaltungen (GR8Conf, Greach, Spring 2GX), die sich nur dem Groovy-Ökosystem widmen. Und bei dem einzigen, aber hochgelobten Groovy-Vortrag auf der letztjährigen WJAX in München gab es über Twitter Anfragen/

Empfehlungen an die Organisatoren, das Thema statt im Keller mal wieder im Ballsaal anzubieten.

Die Webseite listet zudem einige namhafte Firmen auf, die Groovy verwenden. Natürlich weiß man nicht, in welchem Umfang. Aber genau das ist ja eine der Stärken: Groovys Universalität von Skripten bis hin zu ausgewachsenen Enterprise-Anwendungen. Von daher eine klare Empfehlung, Groovy und gegebenenfalls auch Grails mal (wieder) eine Chance zu geben.

Referenzen

[1] Gardners Hype-Kurve: *https://de.wikipedia.org/wiki/Hype-Zyklus*

[2] About being a paid OSS Developer for Groovy: *http://blackdragsview.blogspot.de/2015/04/about-being-paid-oss-developer-for.html*

[3] Who is Groovy: *http://melix.github.io/blog/2015/02/who-is-groovy.html*

[4] Groovy 2.4 And Grails 3.0 To Be Last Major Releases Under Pivotal Sponsorship: *http://blog.pivotal.io/pivotal/news-2/groovy-2-4-and-grails-3-0-to-be-last-major-releases-under-pivotal-sponsorship*

[5] Moving Groovy to a Foundation: *http://www.groovy-lang.org/mailing-lists.html#nabble-td5722483*

[6] Tiobe-Index: *http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html*

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 @sippsack.

Frontend-Entwicklung mit ClojureScript und React/Reacl

Michael Sperber, Active Group GmbH

Welches ist das beste Framework, um ein GUI-Frontend in Java zu entwickeln? Swing, SWT oder gleich JavaFX? Es geht auch ganz anders: Seit Java 8 wird ein kompletter Web-Browser mit HTML5 und JavaScript mitgeliefert. So können JavaScript-Anwendungen auch innerhalb von Java-Anwendungen laufen.

HTML5 ist eine echte Alternative zu den „Native"-Toolkits. Diese Option wird besonders attraktiv durch React, das GUI-Framework für HTML5 von Facebook. Es erschließt eine völlig neue Art der GUI-Programmierung. Noch besser lässt sich React mithilfe von ClojureScript und dem React-Wrapper Reacl programmieren.

Zunächst einige grundsätzliche Gedanken zur GUI-Programmierung: Eine vernünftige GUI-Architektur setzt auf die Trennung von „Modell" (oder „Business-Logik") und „View". Unterschiede zwischen Patterns wie MVC oder MVVM sind für die Zwecke dieses Artikels nebensächlich. Damit die GUI stets den korrekten Zustand des Modells anzeigt, muss das Modell sie über Änderungen informieren. Bei „Native"-Toolkits für Java (Swing, SWT und JavaFX) geschieht das in der Regel durch „Callbacks" im Modell, um die GUI zu verändern, in Java gern über das Observer-Pattern *(siehe Abbildung 1)*.

Eine solche Applikation steuert die GUI auf zwei verschiedene Arten:

- Eine Funktion generiert beim Start der Applikation die GUI aus dem Anfangs-Modell
- Callbacks am Modell machen aus einer Veränderung des Modells eine Veränderung der GUI

Der zweite Punkt hat es in sich, weil es oft gar nicht so einfach ist, eine Veränderung der GUI zu finden, die genau der Veränderung des Modells entspricht. Außerdem gibt es Callbacks nicht nur vom Modell in die GUI,

sondern auch in der anderen Richtung. Das kann dazu führen, dass sich Callbacks endlos im Kreis aufrufen. Diese beiden Probleme zusammen machen GUI-Programmierung bei komplexen Anwendungen oft zu einer unbefriedigenden Erfahrung. Daran ändern auch JavaFX und Data Binding nichts Wesentliches.

Das recht neue HTML5/JavaScript-Toolkit React ist von diesem klassischen Modell für die GUI-Programmierung grundlegend abgekehrt. Nach dessen Einführung gab es erst einmal regelrechte Proteste deswegen. Es verzichtet einfach auf Punkt 2 der GUI-Steuerung und auf Callbacks aus dem Modell in die GUI *(siehe Abbildung 2)*.

Das Modell muss natürlich auch hier die GUI über Änderungen informieren, gibt aber bei React nur bekannt, dass sich etwas geändert hat, jedoch nicht, was. React kümmert sich dann automatisch darum, die Änderungen an der GUI durchzuführen, die sich aus den Änderungen des Modells ergeben. Diese Vorgehensweise erschlägt beide Probleme klassischer GUI-Programmierung auf einmal: Die GUI ist immer konsistent mit dem Modell und es gibt keine Callback-Zyklen mehr, da es nur noch in einer Richtung Callbacks gibt. React generiert also konzeptuell bei jeder Interaktion die komplette GUI neu. Dass React es trotzdem schafft, extrem schnell zu arbeiten, grenzt an Zauberei und ist eine substanzielle Leistung der Facebook-Ingenieure.

React ist inzwischen immens populär und es gibt entsprechend viele Anleitungen und Einführungen im Internet. In diesem Artikel geht es deswegen um Reacl, einen Wrapper um React in ClojureScript, der die Programmierung mit React zusätzlich vereinfacht.

ClojureScript ist ein Dialekt von Clojure, der statt JVM-Code JavaScript produziert, der in einem Browser oder eben innerhalb einer Java-8-Anwendung laufen kann. Wer also Clojure kennt, kann fast direkt loslegen, und wer Clojure noch nicht kennt, sollte es schnellstmöglich kennenlernen. Dieser Artikel hat nicht genug Platz, um in Clojure/ClojureScript einzuführen, erklärt aber nahezu jede Programmzeile. Am Ende dieses Artikels gibt es ein paar Links mit Material zu Clojure und ClojureScript. Als Einführung in Reacl dient ein laufendes Beispiel, und zwar – wie inzwischen fast überall – eine einfache To-do-Applikation *(siehe Abbildung 3)*.

Benutzer können To-dos anlegen, abhaken und wieder löschen. Die Grundlage für die Applikation ist das Modell; zunächst einmal einzelne To-dos *(siehe Listing 1)*.

Dies ist eine Record-Definition („Pojo" in Java) mit drei Feldern: „id" ist eine eindeutige Nummer für das To-do (warum die benötigt wird, wird später erläutert), „text" der Text des To-dos und „done?" ein Boolean, der besagt, ob das To-do schon abgehakt ist oder nicht. *Listing 2* zeigt zwei Beispiele für To-dos.

Die Funktion „->Todo" ist der Konstruktor für To-dos; „def" gibt den beiden Beispielen die Namen „t1" und „t2". *Listing 3* zeigt eine Record-Definition für den Zustand der gesamten Applikation, also die Liste aller To-dos.

Außer einer Liste aus To-dos („todos") enthält der Applikationszustand außerdem noch die nächste zu vergebende Nummer „next-id" für IDs. *Listing 4* ist ein Beispiel für eine To-do-Liste mit den zwei To-dos von oben.

Um nun die GUI zu schreiben, muss Reacl erst einmal in den Namespace (das ClojureScript-Pendant zum Java-Package) importiert werden *(siehe Listing 5)*. Das heißt, dass ab hier alle Namen, die mit „reacl/" anfangen, aus dem Reacl-Kern kommen. Alle Namen mit „dom/" kommen aus der Reacl-DOM-Library. Sie werden verwendet, um die GUI-Darstellung als HTML-DOM zu erzeugen.

Da der Applikationszustand in zwei Teile aufgeteilt ist, ist es sinnvoll, die GUI-Programmierung entsprechend zu strukturieren. Erste Aufgabe ist also, eine GUI-Komponente für ein einzelnes To-do zu schreiben. Mit Reacl ist dafür eine „Klasse" zuständig. Sie dient wie in Java auch als Schablone für jede einzelne To-do-Komponente in der GUI, hat aber ansonsten nichts mit dem Java-Konzept „Klasse" zu tun. *Listing 6* zeigt die minimale Klasse, um ein To-do darzustellen.

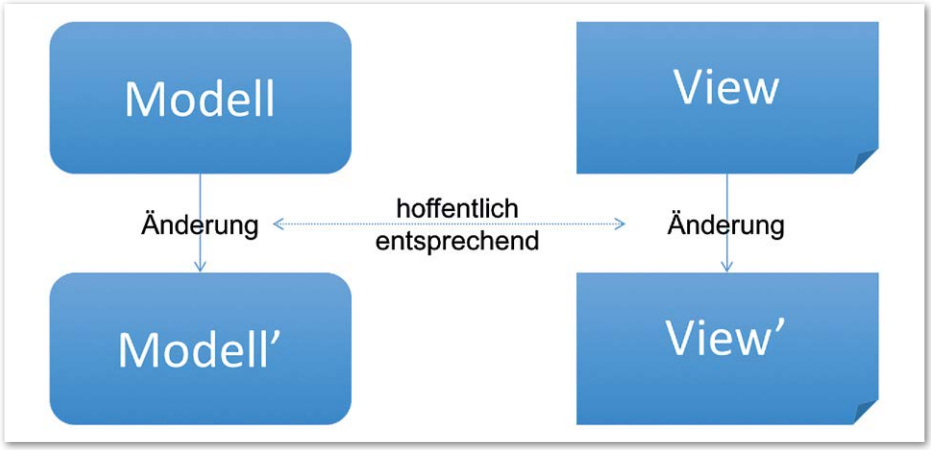


Abbildung 1: Klassische GUI-Programmierung

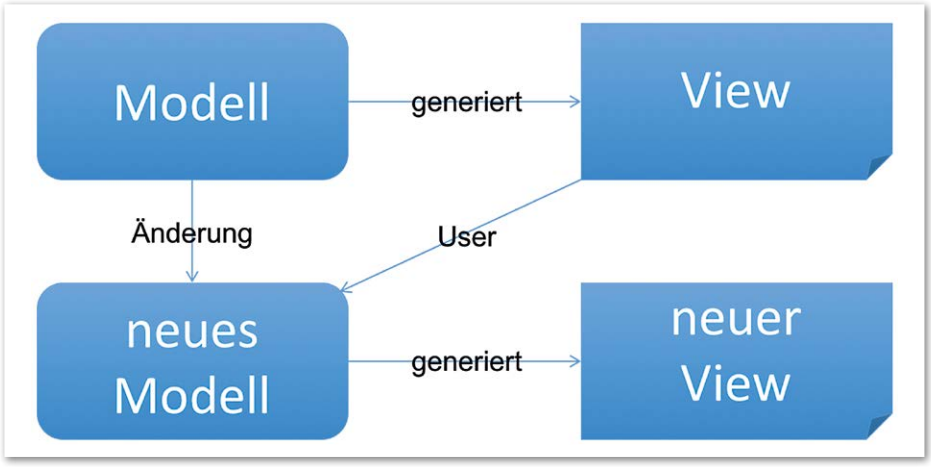


Abbildung 2: GUI-Programmierung mit React

Abbildung 3: Einfache To-do-Applikation

Die Klasse heißt „to-do-item" und wird bei der Konstruktion zwei Argumente für die Parameter „todo" und „parent" akzeptieren. Ersteres ist das To-do-Objekt, das dargestellt werden soll; „parent" wird die darum herumliegende GUI-Komponente sein – dazu später.

Eine Reacl-Klassen-Definition hat mehrere Abschnitte, die jeweils Namen haben. Der wichtigste Abschnitt heißt „render" und ist dafür zuständig, eine View für das To-do-Objekt zu generieren. Die View wird als HTML5 generiert, aber nicht als HTML-Text mit spitzen Klammern, sondern über Funktionen wie „dom/div" und „dom/input", die HTML-Elemente „div" und „input" generieren. Attribute werden dabei als Maps angegeben, die Namen der Attribute als Clojure-Keywords.

Der Aufruf von „dom/input" generiert also beispielsweise folgendes HTML: „<input type="checkbox" value=.../>". Als Anfangswert für die Checkbox dient das „done?"-Feld des To-do-Records, das mithilfe des Record-Getters „:done? <"extrahiert wird. Als Inhalt dient der Text des To-dos „:text todo)".

„todo" bezieht sich also auf den Parameter weiter oben im „defclass". Der Parameter „parent" sowie „this", mit dem der Code in der Klasse auf die Komponente zugreifen kann, werden nicht benutzt. Noch nicht, denn die Klasse ist noch nicht fertig: Sie kann ein To-do darstellen, aber es passiert noch nichts, wenn auf die Checkbox oder den Zap-Knopf gedrückt wird. Dazu müssen an die jeweiligen DOM-Elemente noch Callbacks geklebt werden, zunächst an die Checkbox *(siehe Listing 7)*.

Zur Erinnerung: React verzichtet auf die Callbacks in der anderen Richtung – vom Modell in die GUI. Dies hier ist ein Callback von der GUI ins Modell. Der Callback ist eine Clojure-Funktion, die ein „Event"-Objekt namens „e" übergeben bekommen wird. Der Ausdruck „(.. e -target -checked)" extrahiert aus dem Event-Objekt den Zustand des Häkchens: „true" wenn abgehakt, sonst „false".

Bei normaler GUI-Programmierung würde der Callback jetzt das Modell verändern. Reacl allerdings zwingt das Programm, eine Nachricht an eine GUI-Komponente zu schicken, die den Wunsch des Benutzers beschreibt – mit einem Aufruf von „reacl/send-message!". Dies verbessert die Modularität des Codes und macht ihn auch einfacher testbar. In diesem Fall schickt die GUI-Komponente die Nachricht an „this", also an sich selbst. Die Veränderung des Modells findet in einem anderen Abschnitt der Klassendefinition statt, der „handle-message" heißt *(siehe Listing 8)*.

Bei „handle-message" muss eine Funktion stehen, und die bekommt die Message aus dem Aufruf an „send-message!" übergeben. In diesem Fall ist das gerade das Häkchen. Die „handle-message"-Funktion muss jetzt einen neuen Zustand für das To-do-Item erzeugen. In Clojure wird dazu nicht ein Setter aufgerufen, sondern ein neues To-do erzeugt. Der Ausdruck „(assoc todo :done? checked?)" liefert ein neues To-do-Objekt, das dem ursprünglichen „todo" bis auf das „done?"-Feld entspricht, das mit dem Wert von „checked?" befüllt ist.

Die „handle-message"-Funktion gibt nun bekannt, dass es einen neuen Applikationszustand gibt, indem sie „reacl/return" aufruft. Anmerkung: „handle-message"-Funktionen rufen immer „reacl/return" auf. Der Keyword-Parameter „:app-state" besagt, dass ein neuer Komponenten-Zustand geliefert wird. Es gibt noch eine andere Option bei „reacl/return" – dazu später.

Es ist also erklärt, wie das Häkchen am To-do funktioniert. Was ist aber mit dem „Zap"-Knopf? Er soll ja das To-do aus der Liste aller To-dos entfernen. Von dieser Liste weiß aber die Klasse „to-do-item" gar nichts, sie muss also den Wunsch, ein To-do zu entfernen, nach oben delegieren – dafür ist der Parameter „parent" da. Damit dieser auch weiß, dass etwas gelöscht werden soll, wird ein extra dafür gemachter Record verschickt, in dem das zu löschende To-do steht: „(defrecord Delete [todo])".

Der entsprechende Callback kommt an das „onclick"-Attribut des Buttons. *Listing 9* zeigt jetzt den vollständigen Code für „to-do-item". Das „_" beim Button-Callback ist ein Pseudo-Parameter, der „interessiert nicht" aussagt.

Damit ist die erste vollständige Reacl-Klasse fertig. Als Nächstes kommt die Klas-

```
(defrecord Todo
  [id text done?])
```

Listing 1

```
(def t1 (->Todo 0 "ClojureScript lernen" false))
(def t2 (->Todo 1 "Reacl lernen" false))
```

Listing 2

```
(defrecord TodosApp
  [next-id todos])
```

Listing 3

```
(def ts (->TodosApp 2 [t1 t2]))
```

Listing 4

```
(ns examples.todo.core
  (:require [reacl.core :as reacl :include-macros true]
            [reacl.dom :as dom :include-macros true]))
```

Listing 5

```
        {:type "checkbox"
        :value (:done? todo)
        :onchange (fn [e]
           (reacl/send-message!
           this
           (.. e -target -checked)))})
```

Listing 7

```
(reacl/defclass to-do-item
  this todo [parent]
  render
  (dom/div (dom/input
            {:type "checkbox"
            :value (:done? todo)})
            (dom/button "Zap")
            (:text todo)))
```

Listing 6

se, die ein „TodoApp"-Objekt mit der ganzen To-do-Liste und dem Eingabefeld für ein neues To-do anzeigen kann. Hier lohnt es sich, vorher darüber nachzudenken, was für Benutzerwünsche behandelt werden sollen:

- Der Text im Eingabefeld ändert sich
- Aus dem Text im Eingabefeld soll ein neues To-do gemacht werden
- Ein To-do soll gelöscht werden
- Ein To-do soll geändert werden

Aus jedem dieser Punkte wird jeweils ein separater Message-Typ werden, der den Wunsch beschreibt. Der letzte Punkt verdient besondere Aufmerksamkeit: Wenn das Häkchen an einem To-do geändert wird, dann erzeugt die „to-do-item"-Klasse ein neues To-do-Objekt. In der Liste der To-dos steht allerdings noch das alte Objekt – es muss noch durch das neue ausgetauscht werden. Dafür ist also ein separater Message-Typ erforderlich. *Listing 10* zeigt die Record-Definitionen. Für das Löschen gibt es schon den „Delete"-Typ.

Eine weitere Vorüberlegung ist notwendig: Die GUI hat ein Eingabefeld für das nächste To-do. Solange aber nicht auf den „Add"-Knopf oder „Return" gedrückt wird, gibt es kein To-do. Der dort eingetippte Text gehört also zum Zustand der GUI, nicht zum Zustand der Applikation. Der Typ „NewText" hat gar kein Feld dafür. Irgendwo muss dieser Zustand aber hin. Reacl unterstützt dafür den sogenannten „local state", und der kann bei „defclass" als zusätzlicher Parameter angemeldet werden *(siehe Listing 11)*.

Hier heißt der Parameter „local-state", er kann wie „this" und „app-state" beliebig gewählt werden. Der „app-state"-Parameter wird für das „TodoApp"-Objekt verwendet. Für den „local state" gibt es einen zusätzlichen Abschnitt in der Klassen-Definition, der den initialen Wert festlegt – beim Start ist noch nichts eingetippt: „initial-state „"". *Listing 12* zeigt den Anfang des „render"-Abschnitts, der die Überschrift und die To-dos darstellt.

Der Ausdruck „(:todos app-state)" extrahiert die Liste der To-dos aus dem „app-state". Die Funktion „map" wendet nun auf jedes To-do eine Funktion an, die daraus HTML macht. Dabei kommt die vorher definierte Klasse „to-do-item" zum Einsatz. Die Erläuterung zu „dom/keyed" kommt später. Sie wird als Funktion mit den drei folgenden Argumenten aufgerufen:

```
(reacl/defclass to-do-item
  this todo [parent]
  render
  (dom/div ...)
  handle-message
  (fn [checked?]
     (reacl/return :app-state
        (assoc todo :done? checked?))))
```

Listing 8

```
(reacl/defclass to-do-item
  this todo [parent]
  render
  (dom/div (dom/input
         {:type "checkbox"
         :value (:done? todo)
         :onchange (fn [e]
            (reacl/send-message!
            this
            (.. e -target -checked)))})
         (dom/button
         {:onclick
         (fn [_]
            (reacl/send-message! parent (->Delete todo)))}
         "Zap")
         (:text todo))
  handle-message
  (fn [checked?]
     (reacl/return :app-state
           (assoc todo :done? checked?))))
```

Listing 9

```
(defrecord NewText [text])
(defrecord Submit [])
(defrecord Change [todo])
```

Listing 10

```
(reacl/defclass to-do-app
  this app-state local-state []
  ...)
```

Listing 11

```
  render
  (dom/div
     (dom/h3 "TODO")
     (dom/div
        (map (fn [todo]
           (dom/keyed (str (:id todo))
              (to-do-item
              todo
              (reacl/reaction this ->Change)
              this)))
           (:todos app-state)))
```

Listing 12

```
  (dom/form
     {:onsubmit (fn [e]
        (.preventDefault e)
        (reacl/send-message! this (->Submit)))}
  (dom/input {:onchange
     (fn [e]
        (reacl/send-message!
        this
        (->NewText (.. e -target -value))))
     :value local-state})
  (dom/button
     (str "Add #" (:next-id app-state)))))
```

Listing 13

- „todo" ist das To-do, das den Zustand der Komponente bildet. Es wird an den gleichnamigen Parameter von „to-do-item" gebunden.
- Dann kommt eine sogenannte „Reaktion", die beschreibt, was passiert, wenn der Zustand der Komponente sich ändert. Zur Erinnerung: Davon muss „to-do-app" erfahren, damit es das To-do in der Liste auswechseln kann. Sie erfährt dies mit einer „Change"-Nachricht an „this". Die Reaktion „(reacl/reaction this ->Change)" besagt, dass, wenn ein neuer Zustand des To-dos vorliegt, dieser mithilfe des Konstruktors „->Change" zu einer „Change"-Nachricht gemacht und diese an „this" geschickt wird. Anmerkung: Dass hier nicht einfach wieder eine Funktion steht, die „sendmessage!" aufruft, hat technische Gründe – es erlaubt Reacl, noch bessere Performance herauszuholen.
- Des letzte Argument „this" wird an den „parent"-Parameter in „to-do-item" gebunden.

```
handle-message
(fn [msg]
   (cond
      (instance? NewText msg)
      (reacl/return ...)
      (instance? Submit msg)
      (reacl/return ...)
      (instance? Delete msg)
      (reacl/return ...)
      (instance? Change msg)
      (reacl/return ...)))
```

Listing 14

Jedes To-do wird noch mit „dom/keyed" eingewickelt. Der Grund dafür ist etwas subtil: Alle To-dos stehen in einer Liste untereinander. Nehmen wir an, die Listenelemente werden in eine andere Reihenfolge gebracht. Ohne weitere Informationen sieht das für React aus, als ob sich (potenziell) jedes einzelne To-do geändert hat, da an seiner Stelle jetzt ein anderes steht. Es müsste also jedes To-do neu rendern. Mithilfe von „dom/keyed" sind die Listenelemente eindeutig mit einem Schlüssel markiert, sodass React gegebenenfalls die DOM-Elemente einfach nur umsortiert. Es geht auch ohne, aber React zeigt dann eine Warnung an. Im Beispiel hat schon jedes To-do eine eindeutige Nummer im „id"-Feld, die durch „str" in einen String umgewandelt als Schlüssel dienen kann.

Der „render"-Abschnitt ist noch nicht ganz vollständig, es fehlen noch das Textfeld für das nächste To-do und der Add-Knopf. Textfeld und Button werden in einem HTML „form" eingepackt, dadurch funktioniert auch die Return-Taste als „Submit"-Aktion des Formulars *(siehe Listing 13)*.

Zunächst zum „onsubmit"-Callback: Der Aufruf von „.preventDefault" ist ein HTML5-Artefakt, das dafür sorgt, dass die Applikation nach dem Drücken von Return nicht neu geladen wird, sonst wäre das To-do ja gleich wieder weg. Dann wird eine „->Submit"-Nachricht verschickt. Der Callback beim „input"-Textfeld wird bei jeder Änderung des Textes aufgerufen. Der Ausdruck „(.. e -target -value)" extrahiert den Text. Dieser wird dann in eine „NewText"-Nachricht eingepackt und verschickt. Das Rendering ist damit fertig.

Es fehlt noch der „handle-message "-Abschnitt, in dem die ganzen Nachrichten verarbeitet werden. Anders als bei „to-do-item " können hier vier verschiedene Sorten von Nachrichten auflaufen, die Funktion muss also eine Verzweigung nach dem Record-Typ machen. In Clojure geht das mit „cond " und „instance? " *(siehe Listing 14)*.

Der erste Fall soll einfach nur den „local state" aktualisieren, und zwar auf den Wert des „text"-Felds des Reocrds. Das geht auch mit „reacl/return", allerdings wird dabei nicht das Keyword „:app-state", sondern stattdessen „:local-state" verwendet *(siehe Listing 15)*.

Bei „Submit" müssen der „local state" und auch der Applikationszustand ausgewechselt werden: Das neue To-do muss angelegt werden und danach soll das Textfeld leer sein. Dafür kann „reacl/return" sowohl ein „:local-state" als auch ein „:app-state" bekommen *(siehe Listing 16)*.

In „app-state" müssen beide Felder ausgewechselt werden – das erfolgt wieder mit „assoc:". Bei „todos" muss ein neues To-do erzeugt und angehängt werden. Das neue To-do bekommt die nächste ID, die im „app-state" steckt. Daraufhin muss natürlich die

```
   (instance? NewText msg)
   (reacl/return :local-state (:text msg))
```

Listing 15

```
(instance? Submit msg)
(let [next-id (:next-id app-state)]
   (reacl/return
      :local-state ""
      :app-state
      (assoc app-state
         :todos
         (concat (:todos app-state)
            [(->Todo next-id local-state false)])
         :next-id (+ 1 next-id))))
```

Listing 16

```
(instance? Delete msg)
(let [id (:id (:todo msg))]
   (reacl/return :app-state
      (assoc app-state
      :todos
      (remove (fn [todo] (= id (:id todo)))
         (:todos app-state)))))
```

Listing 17

```
(instance? Change msg)
(let [changed-todo (:todo msg)
   changed-id (:id changed-todo)]
(reacl/return :app-state
      (assoc app-state
         :todos (map (fn [todo]
            (if (= changed-id (:id todo))
               changed-todo
               todo))
            (:todos app-state))))))))
```

Listing 18

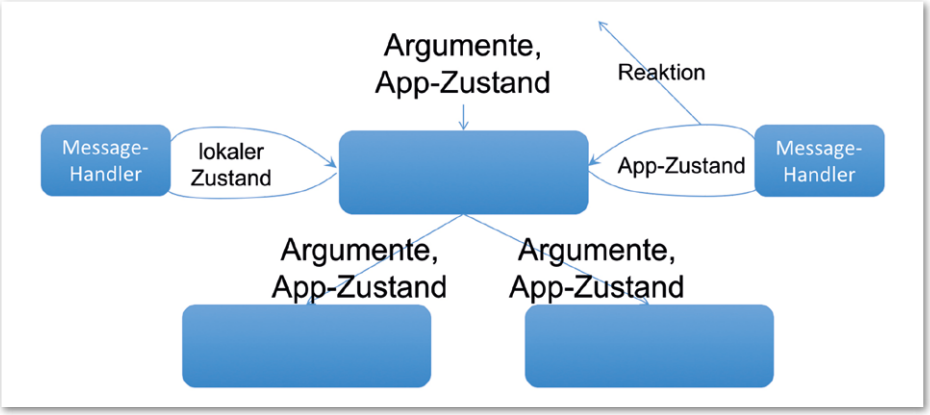


Abbildung 4: Arbeitsweise von Reacl-Komponenten

ID hochgezählt werden. Als Nächstes ist „Delete" dran *(siehe Listing 17)*.

Der Code benutzt die Clojure-Funktion „remove", um das To-do zu entfernen, dessen ID mit der ID des zu löschenden übereinstimmt. Es fehlt noch „Change", bei dem der Code über die Liste der To-dos mappt, dort das auszutauschende To-do mithilfe der ID identifiziert und austauscht *(siehe Listing 18)*.

Natürlich ist noch etwas Drumherum notwendig, um aus dem Code eine lauffähige Applikation zu machen (Build-Skript, HTML-Skelettseite etc.), aber das wäre in einem Artikel langweilig: Das komplette To-do-Beispiel wird bei Reacl mitgeliefert. Auf der Reacl-Homepage gibt es weitere Dokumentation.

In *Abbildung 4* ist noch einmal zusammenfassend dargestellt, wie Reacl arbeitet: Die Komponenten einer GUI bilden einen Baum. Jede Komponente verwaltet einen Applikationszustand und einen lokalen Zustand. Die „render"-Funktion macht aus beidem HTML für die Darstellung. Der Applikationszustand fließt zusammen mit weiteren Argumenten im Baum nach unten. Der lokale Zustand („local state") wird von jeder Komponente komplett intern verwaltet.

Die Message-Handler nehmen Nachrichten über Benutzerwünsche entgegen und setzen sie um, indem sie jeweils einen neuen lokalen Zustand oder einen neuen Applikationszustand (oder beides) erzeugen. Außerdem führt ein neuer Applikationszustand in einer Komponente zu einer „Reaktion" weiter oben. Damit werden die Probleme traditioneller GUI-Programmierung vermieden:

- Es werden keine Callbacks am Modell benötigt
- Es muss nur einmal programmiert werden, wie aus dem Modell eine View wird

Wie am Anfang angekündigt, lässt sich so eine React-Anwendung in ganz unterschiedlichen Kontexten zum Laufen bringen: Zu React gibt es ein richtiges Framework, um mobile Apps zu entwickeln *(siehe „https://facebook.github.io/react-native/")*. Außerdem kann die JavaFX-Klasse „javafx.scene.web.WebView" eine HTML5-Anwendung kapseln: Das lässt sich so gestalten, dass Benutzer gar nicht merken, dass sie es nicht mit einer „Native"-GUI zu tun haben.

Links

1. Clojure-Homepage: *http://clojure.org/*
2. ClojureScript-Homepage: *https://github.com/clojure/clojurescript*
3. Mehrteilige Einführung in Clojure im Blog „Funktionale Programmierung": *http://funktionale-programmierung.de/2014/11/27/clojure-first-steps.html*
4. React-Homepage: *https://facebook.github.io/react/*
5. React Native: *https://facebook.github.io/react-native/*
6. Reacl-Homepage: *https://github.com/active-group/reacl*
7. Eine Einführung in Clojure gab es außerdem im Artikel „Eingebettete DSLs mit Clojure" in Java aktuell 02/2014.

Michael Sperber
michael.sperber@active-group.de

Dr. Michael Sperber ist CTO der Active Group GmbH. Er ist seit mehr als zwanzig Jahren in der Forschung über funktionale Programmierung und deren industrielle Anwendung tätig. Er hat zahlreiche Fachartikel zum Thema verfasst, Anfänger-Ausbildungen in Programmierung an den Universitäten Tübingen, Freiburg und Kiel konzipiert und (in Tübingen) mehrfach durchgeführt. Michael Sperber gehört zu den Mit-Initiatoren und Autoren des Blogs „funktionale-programmierung.de" sowie der Entwicklerkonferenz BOB.

Oracle Critical Patch Update Januar 2016: Der Konzern schließt 248 Schwachstellen in Oracle-Produkten

Mit der Veröffentlichung des vierteljährlichen Critical Patch Updates *(siehe „http://www.oracle.com/technetwork/topics/security/alerts-086861.html")* stellt Oracle insgesamt 248 Sicherheitsupdates für hunderte von Oracle-Produkten zum Download bereit. Der Patch umfasst zahlreiche Oracle-Produktfamilien: Datenbank, Middleware, Applikationen, Enterprise Management, Java und Virtualisierung.

Die kritischsten Schwachstellen mit der höchsten Gefahrenstufe 10.0 nach dem Common Vulnerability Scoring System (CVSS) schließt der Konzern bei Java SE und GoldenGate. Der Konzern empfiehlt ein schnelles Update aller betroffenen Produkte, da viele der Lücken ohne Authentifizierung ausgenutzt werden können.

Für den Datenbank-Server werden sieben Sicherheitslücken geschlossen – jedoch kann keine dieser Schwachstellen von Angreifern über das Netzwerk ohne Authentifizierung missbraucht werden. Der höchste CVSS-Wert beträgt dennoch 9.0. Betroffen sind Database Vault, Java VM, Security, Workspace Manager, XDB – XML Database und XML Developer's Kit for C.

Für die Fusion Middleware stellt Oracle 27 neue Fixes bereit. Hier wird die höchste Sicherheitseinstufung von 7.5 erreicht – 17 Sicherheitslücken erlauben einen unautorisierten Zugriff ohne Benutzernamen und Passwort. Java SE wird sogar mit einer Sicherheitseinstufung von 10.0 belegt.

Grundlagen und Patterns von reaktiven Anwendungen am Beispiel von Vert.x und Reactor

Martin Lehmann, Accso – Accelerated Solutions GmbH

Reaktive Anwendungen mit asynchronen, Event-getriebenen Architekturen haben stark an Bedeutung gewonnen. Ein Treiber für diese Entwicklung sind Web, Mobile und Internet of Things mit ständig steigenden Anforderungen. In den letzten Jahren sind verschiedene Frameworks entstanden, die solche Architekturen unterstützen, indem sie Requests asynchron auf nur wenigen Threads verarbeiten.

Moderne Anwendungen und Anwendungslandschaften sind verteilt, ihre Systemkomponenten müssen folglich miteinander kommunizieren. Dabei muss besonderes Augenmerk auf nichtfunktionale Anforderungen hinsichtlich Performance (Antwortzeiten, Durchsatz, Latenzen), Skalierbarkeit und Verfügbarkeit gelegt werden. Fachlichkeiten, die stark auf Event-basierten und Datenfluss-orientierten Verarbeitungsketten basieren (etwa Aktienkurse, Infoticker, Wetterdaten *[1]*), können gut mit reaktiven Anwendungen umgesetzt werden. Die Philosophie solcher Anwendungen beschreibt das Reactive Manifesto *[2]* mit vier Anforderungen *(siehe Tabelle 1)*.

Jonas Bonér fasst diese Punkte in *[3]* wie folgt zusammen: „We need to build systems that react to events („event-driven"), react to load („scalable"), react to failure („resilient"), react to users („responsive")."

Grundlegende Konzepte

In den letzten Jahren erfreuen sich Frameworks großer Beliebtheit, auf deren Basis sich reaktive Systeme erstellen lassen. Prominenter Vorreiter ist Node.js, aber auch Akka, Ruby's EventMachine, Vert.x sowie

Anforderung	Bedeutung
Responsive (deutsch: antwortbereit)	Das System antwortet möglichst zeitgerecht und schnell. Außerdem: In verteilten Systemen können Fehlersituationen letztlich nur durch das Ausbleiben einer Antwort (nach Timeout) festgestellt werden.
Resilient (deutsch: widerstandsfähig)	Das Gesamtsystem bleibt auch bei Ausfällen der lose gekoppelten Einzelkomponenten antwortbereit. Fehler in Einzelbereichen führen nicht zum Ausfall des Gesamtsystems oder zu signifikanten Verschlechterungen der Antwortzeiten.
Elastic (deutsch: elastisch)	Das System bleibt auch unter sich ändernden Lastbedingungen antwortbereit, kann also seinen Ressourcen-Bedarf flexibel an Anfragezahlen anpassen: Mehr Anfragen werden durch die Hinzunahme von mehr Ressourcen abgedeckt. Auch sinkende Anfragezahlen werden erkannt und Ressourcen wieder freigegeben.
Message-driven (deutsch: nachrichtenorientiert)	Anders als die ersten drei Punkte ist „message-driven" weniger Anforderung als vielmehr Lösungsbeschreibung: Performance, Stabilität und dynamischer Ressourcen-Bedarf waren schon immer in nichttrivialen Systemen abzubilden. Das Reactive Manifesto nimmt daher Nachrichten-orientierung explizit mit auf, da dies asynchrone, nicht-blockierende Kommunikation zwischen den Komponenten bedingt.

Tabelle 1: Reactive Manifesto

die Basis-Frameworks Netty und Reactor, RxJava und Reactive Extensions sind hier zu nennen.

Bevor auf Vert.x und Reactor eingegangen wird, werden mit Reactor Pattern und Aktorenmodell zwei Basiskonzepte vorgestellt, die altbekannt sind und in den letzten Jahren wiederentdeckt wurden. Zwar erfreut sich die Nutzung sogenannter „nackter Threads" (also APIs auf Basis von „java.lang.Thread") noch immer erstaunlich großer Beliebtheit (siehe Umfrage *[4]*). Doch die Implementierung verteilter und nebenläufiger Systeme ist alles andere als trivial und birgt das Risiko von Fehlersituationen wie Deadlocks und Race Conditions. Durch die Einführung von Abstraktionen wie den nachstehenden Konzepten muss sich der Anwendungsprogrammierer nicht mehr mit komplizierten Low-Level-Konzepten wie Threads und Locks beschäftigen.

Reactor Pattern

Bereits im Jahr 1996 hat POSA *[5]* ein Design-Pattern für „Event-Handling" beschrieben. Der „Reactor" nimmt als zentraler Dispatcher Events entgegen und reicht diese an Callback-Handler weiter. Das entkoppelt Applikations- von Ausführungslogik *(siehe Abbildung 1)*.

Das „POSA"-Pattern beschreibt einen Event-Loop, in dem Events in einem Event-Loop-Thread an Callback-Handler delegiert werden. Dieses Prinzip kann man als Multi-Reactor-Pattern auf mehrere Event-Loop-Threads verallgemeinern. Damit ändert sich auch das Programmiermodell: Der Event-Loop verwaltet den zentralen Kontrollfluss und ruft bei einem Event einen registrierten Callback-Handler auf *(siehe rechts in Listing 1)*:

Aktorenmodell

Aus der IT-Steinzeit stammt das zweite hier vorgestellte Konzept: Bereits im Jahr 1973 stellte Carl Hewitt das Aktorenmodell als Modell für nebenläufige Programmierung vor *[6]*. Darin sind „Aktoren" die ausführbaren Einheiten im System. Ein Anwendungsprogrammierer kann (theoretisch) beliebig viele definieren und instanziieren. Wie Aktoren zur Laufzeit auf Ressourcen abgebildet werden (letztendlich in der Regel auf Threads), muss ihn nicht interessieren. Die Laufzeitumgebung fungiert als Scheduler für die Aktoren (ähnlich wie Betriebssystem-Prozesse durch einen Scheduler verwaltet werden).

Abbildung 2 zeigt, wie Aktoren über asynchronen Nachrichtenaustausch miteinander kommunizieren (typischerweise mit Nachrichten-Pufferung). Jeder Aktor verwaltet seinen lokalen Zustand. Nach dem „Share-Nothing-Prinzip" gibt es keinen „Shared Mutable State".

Es gilt, „Shared Mutable State" aus folgenden Gründen zu vermeiden:

- Code mit global veränderlichen Variablen ist schwierig zu pflegen, da die Auswirkungen von Änderungen schwer vorhersehbar sind.
- Zudem sind Tests in der Regel aufwändiger zu konfigurieren, wenn ein globaler Zustand erst Testfall-gerecht eingestellt werden muss. Testen wird also aufwändiger, was guter Test-Abdeckung kaum förderlich ist.
- Code ohne globalen Zustand ist besser komponier- und testbar. Bei Code in funktionalem Stil hängt ein Ergebnis ausschließlich von den Eingabe-Parametern ab und ist Seiteneffekt-frei.
- Nebenläufigkeitsprobleme wie „Race Conditions" sind schwierig nachzustellen und zu beseitigen.
- Schlussendlich können Synchronisierungen und Locks am globalen Zustand zu Performance-Engpässen führen.

Es gibt verschiedene Frameworks für die JVM, die das Aktorenmodell implementieren. Bekanntester Vertreter ist das Akka-Framework, aber auch in Scala und Groovy (über die „Gpars"-Bibliothek) gibt es Aktoren.

Das Vert.x-Framework für die JVM

Vert.x ist ein asynchrones, Event-getriebenes und nichtblockierendes Framework auf der JVM *[7]*, veröffentlicht unter der Apache License 2. Im Juni 2015 ist Version 3.0 erschienen, das neue Funktionalität, aber auch Struktur-Änderungen gegenüber Vert.x 2.x mit sich bringt. Reactor Pattern und Aktorenmodell werden von Vert.x implementiert, um Nebenläufigkeit, Performance und Skalierbarkeit zu optimieren. Das Programmiermodell beruht auf Event-basierten Callbacks, das Laufzeitmodell lehnt sich an das Aktorenmodell an *(siehe Tabelle 2)*.

Sogenannte „Verticles" bilden die fachlichen Komponenten einer Vert.x-Anwendung und sind damit Strukturvorgabe für den technischen Komponentenschnitt sowie ausführbare Laufzeit-Instanzen. Vert.x bedient die Kommunikationsprotokolle TCP/SSL, HTTP/HTTPS, DatagramSockets, SockJS/WebSockets und DnsClient in Form der oben beschriebenen Event-Driven-Programmierung: Geht eine Anfrage beispielsweise als HTTP-Request ein, so wird der dazu registrierte Callback-Handler aufgerufen. Dies zeigt ein einfaches Beispiel *(siehe Listing 2)*. Der „HelloWorldHttpServer" leitet

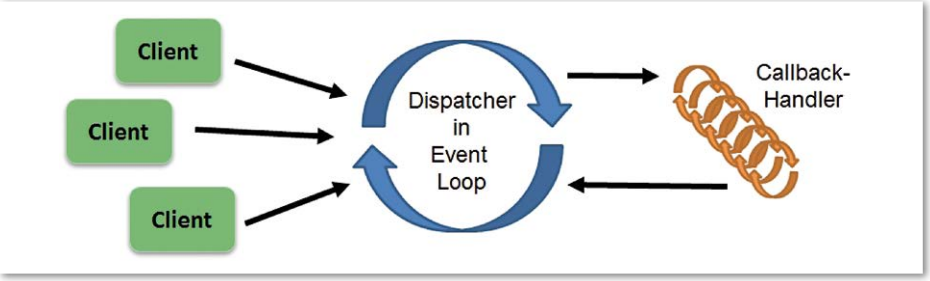


Abbildung 1: „Reactor"-Pattern

Vert.x ist ...	
Leichtgewichtig	Der Kern des Frameworks ist weniger als 1 MB groß und hat ein überschaubares API
Laufzeitumgebung	Vert.x bringt eine eigene Laufzeitumgebung mit, Anwendungen müssen also nicht in einem Applikationsserver eingerichtet werden.
Erweiterbar	Weitere Funktionalität ist über Module einbindbar (aus Repositories)
Performant	In TechEmpower-Benchmarks *[8]* ging Vert.x in der Testkategorie „Web-Frameworks" (HTTP/Plain-Text) bereits als Testsieger hervor
Polyglott	Vert.x erlaubt den kombinierten Einsatz verschiedener JVM-Sprachen, neben Java unter anderem JavaScript, Groovy, JRuby, Clojure und Scala. In Vert.x 3.0 fehlen leider aktuell einige der idiomatischen Sprach-Anbindungen

Tabelle 2: Vert.x-Steckbrief

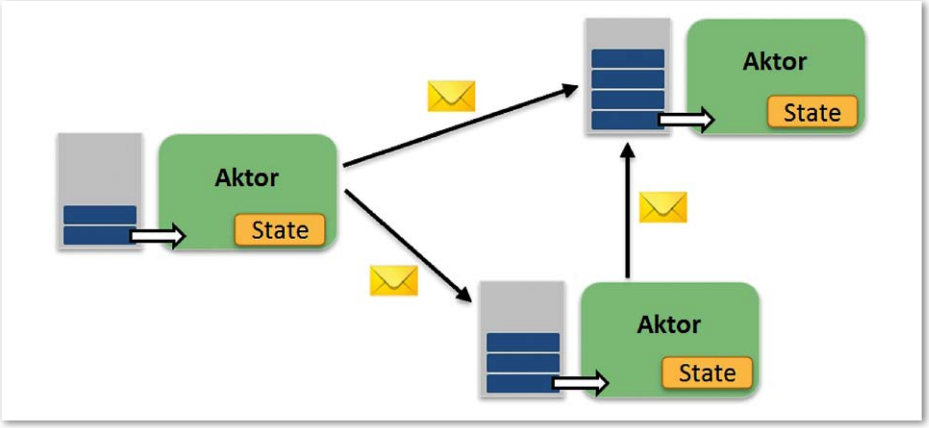


Abbildung 2: Das Aktorenmodell

von der Vert.x-Framework-Klasse „AbstractVerticle" ab (Zeile 5).

Die Vert.x-Laufzeitumgebung ruft die Lifecycle-Methode „start" (Zeile 7) auf. Über „vertx()" hat der „HelloWorldHttpServer" Zugriff auf das API. Das Verticle beantwortet für Port 8123 im Callback-Handler (ab Zeile 8) HTTP-Requests mit „Hello World!". Für solche Callbacks bietet sich ab Java 8 die Verwendung von Lambda-Ausdrücken an, da andernfalls Code durch geschachtelte Callback-Schnittstellen schnell unübersichtlich wird. Frameworks wie Reactor bieten dazu ein lesbares Stream-API an, das geschachtelte Callbacks vermeidet.

Verticles als Basis-Komponenten: Statische und dynamische Sicht

Abbildung 3 zeigt die statische Komponenten-Sicht von Vert.x. Verticles sind die Laufzeit-Komponenten einer Vert.x-Anwendung. In Version 2.x können sie zu Modulen gruppiert werden. Das Modul-Konzept wurde in Version 3.0 in weiten Teilen aufgegeben. Mehrere Vert.x-Laufzeitumgebungen (also mehrere VMs) können einen Cluster über Prozess- und Maschinen-Grenzen hinweg bilden.

Eine Vert.x-Laufzeitumgebung startet automatisch einen Pool von Event-Loop-Threads. Poolgröße ist die Anzahl der CPUs/Cores der Maschine. Die Event-Loop-Threads führen die Verticles aus.

Die Laufzeitumgebung garantiert zwei Eigenschaften:

- Jede Verticle-Instanz wird immer auf demselben Event-Loop-Thread ausgeführt.
- Verticles teilen sich keine Daten. Ausnahme ist ein „Shared Data"-Bereich innerhalb einer Vert.x-Laufzeitumgebung, über den Daten austauschbar sind – erlaubt sind nur Immutable-Datentypen. In Vert.x 2.x erhält jedes Verticle seinen eigenen Classloader, sodass kein globaler Zustand (selbst in „static"-Variablen nicht) zwischen Verticles geteilt werden kann. Diese strikte Classloader-Trennung wurde in Vert.x 3.0 wieder aufgegeben.

Ein Verticle kann also implementiert werden, als ob es „single threaded" ausgeführt würde. Das Programmiermodell verbirgt die mit Nebenläufigkeit verbundene Komplexität vor dem Anwendungsentwickler. Verticles sind also Aktoren aus dem oben beschriebenen Aktorenmodell (auch wenn sie nicht so heißen).

Ein Event-Loop-Thread reagiert auf eingehende Events und ruft das entsprechende Verticle (beziehungsweise einen seiner Event-Handler) auf. *Abbildung 4* zeigt dieses Multi-Reactor-Pattern am Beispiel von vier Event-Loop-Threads einer Quad-Core-Maschine.

Verarbeitungen müssen in Verticles möglichst kurz ablaufen, um den Event-Loop schnell wieder freizugeben. Anders als Preemptive Scheduling von Betriebssystemen unterbricht die Vert.x-Laufzeitumgebung die Verticles nicht. Ein Verticle muss insbesondere blockierende Aktionen unbedingt

```
// Normales
// Programmiermodell

loop {
  // Anwendung hat
  // Kontrollfluss
  doSomething();

}
```

```
// Event-Driven-Programmiermodell

registerHandler(doSomething_Handler);
loop {
  // Event Loop hat Kontrollfluss,
  //   ruft Callback-Handler auf
  event = getNextEventFromQueue();
  dispatchToHandler(event);
}
```

Listing 1: Normales Programmiermodell (links) vs. Event-Driven (rechts)

```
 1: import io.vertx.core.AbstractVerticle;
 2: import io.vertx.core.Vertx;
 3: import io.vertx.core.http.HttpServerRequest;
 4:
 5: public class HelloWorldHttpServer extends AbstractVerticle {
 6:  @Override
 7:  public void start() throws Exception {
 8:     Vertx.vertx().createHttpServer().requestHandler((HttpServerRequest req)->{
 9:        log("received request " + req.remoteAddress());
10:        req.response().putHeader("content-type", "text/plain")
11:              .end("Hello World!");
12:     })
13:     .listen(8123);
14:
15:     log("registered on port 8123");
16:  }
16:}
```

Listing 2: Vert.x-Beispiel

vermeiden, da sonst der Durchsatz des Gesamtsystems beeinträchtigt würde.

Event-Loops stellen die teure Ressource „Thread" mehreren Verarbeitungen zur Verfügung und nutzen sie so besser aus als beim „Thread per Request"-Ansatz der Servlet-APIs (zu weiteren Konsequenzen wie dem notwendigen Verzicht auf „ThreadLocal" siehe auch *[9]*). So sind „Thread.sleep" – lange, rechenintensive Berechnungen oder langläufige blockierende Zugriffe durch Dritt-Bibliotheken (wie JDBC) – in Verticle-Code zu vermeiden. Kommt man um blockierende Aktionen nicht herum, so sollte man diese in spezielle Worker-Verticles auslagern, die in einem Worker-Thread-Pool – von den Event-Loop-Threads getrennt – abgearbeitet werden.

Verticles im Zusammenspiel: Kommunikation über den Bus

Die Kommunikation zwischen Verticles erfolgt als asynchroner Nachrichtenversand über den Event-Bus. Die Zustellung einer Nachricht ist nicht garantiert. Ein Verticle kann Nachrichten unter Angabe einer Adresse auf den Bus schreiben beziehungsweise Nachrichten für eine Adresse empfangen. Solche Adressen sind beliebige Strings – es gibt also keine Vorgaben an Struktur oder Hierarchien solcher Adressen. Bus-Nachrichten sind in Vert.x 2.x entweder primitive Typen (String, Integer etc.) oder JSON-Strukturen. Ab Vert.x 3.0 können über austauschbare Bus-Serialisierungen auch andere Datenstrukturen übermittelt werden. Bus-Nachrichten können sogar im Browser in JavaScript (über WebSockets, mit SockJS) verarbeitet werden.

Eine Bus-Nachricht wird an alle Empfänger-Verticles verschickt – unabhängig davon, ob der Empfänger in der gleichen Laufzeitumgebung oder im Cluster läuft. Vert.x bietet drei Kommunikationsformen:

- *Publish-Subscribe*
  Eine mit „publish" verschickte Nachricht wird allen Subscribern zugestellt (also allen Verticles, die sich auf die Ziel-Adresse registriert haben).
- *Punkt-zu-Punkt*
  Eine mit „send" verschickte Nachricht wird nur einem Empfänger aus der Subscriber-Liste zugestellt. Die Auswahl erfolgt per „Round Robin". Eine gezielte Adressierung einer bestimmten Verticle-Instanz ist nicht möglich.
- *Request-Reply*
  Auf eine Punkt-zu-Punkt-Nachricht kann der Empfänger dem Sender mit „reply" antworten.

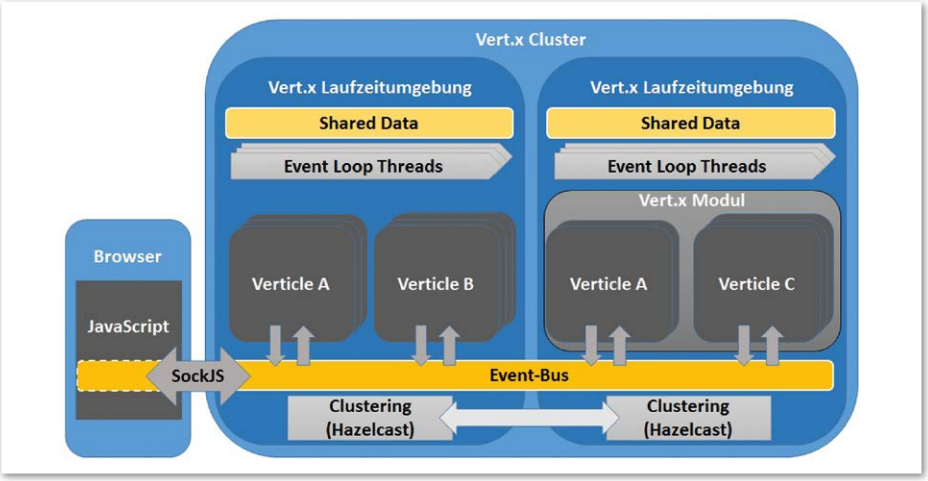


Abbildung 3: Statische Komponenten-Sicht

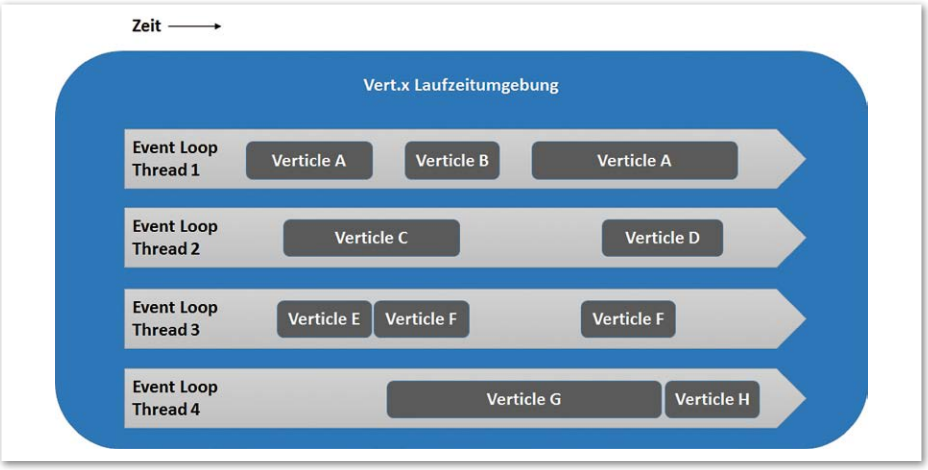


Abbildung 4: Dynamische Abarbeitung von Verticles auf den Event-Loop-Threads

Reactor-Framework für die JVM

Die beschriebenen Basiskonzepte sind auch in anderen Frameworks verbaut, so auch im Reactor-Framework (von SpringSource, veröffentlicht unter Apache Software License 2.0), für das das Reactor Pattern namensgebend war. Reactor wird als Bibliothek innerhalb einer Anwendung genutzt, bringt also im Gegensatz zu Vert.x oder Node.js keine Laufzeitumgebung mit. Reactor ist auf hohe Performanz ausgelegt: Eine Verarbeitung von 15 bis 25 Millionen Events pro Sekunde soll auf einem handelsüblichen Rechner möglich sein.

Ein asynchroner Event-Bus ist der Kern von Reactor, der für Routing und Event-Zustellung zuständig ist. Sogenannte „Consumer" können sich am Bus registrieren und Bus-Nachrichten empfangen, die als Events eintreffen. Das Beispiel in *Listing 3* zeigt Event-Produktion (in Zeile 13 ff) und Event-Verarbeitung im Consumer (in Zeile 6 ff). Sie tauschen asynchron Daten aus.

Ein Dispatcher erledigt die Arbeit

Für Event-Zustellung und Ausführung des Consumer ist ein Dispatcher zuständig, indem er einen Thread zur Verfügung stellt. Wie viele Threads es gibt, wie sie verwaltet werden und wie die Verteilung auf Threads passiert, hängt von der gewählten Dispatcher-Implementierung ab. Default ist der Ringbuffer-Dispatcher, der Events in einem Ringbuffer speichert und mit nur einem Thread abarbeitet. Dies nutzt Konzepte von Disruptor *(siehe [10] und [11])* – ideal für hohen Durchsatz bei kurz laufenden Consumern. *Tabelle 3* zeigt einen Vergleich zwischen Reactor und Vert.x.

```
 1: // Init
 2: CountDownLatch latch = new CountDownLatch(10);
 3: Environment.initialize();
 4: Broadcaster<String> stream = Broadcaster.create(Environment.get());
 4:
 5: // Consumer
 6: stream.map(String::toUpperCase)
 7:  .consume(msg -> {
 8:     System.out.printf("Received %s (thread %s)%n", msg, Thread.currentThread());
 9:     latch.countDown();
10: });
10:
11: //--------------------------------------------------------------------------
12: // Publisher
13: for (int i=0; i<10; i++) {
14:  String msg = "hello world " + i;
15:  System.out.printf("Sending %s (thread %s)%n", msg, Thread.currentThread());
16:  stream.onNext(msg);
17: }
18: stream.onComplete();
19:
20: // shutdown
21: latch.await();
22: Environment.terminate();
```

Listing 3: Reactor-Beispiel (mit Stream-API)

Reactor	Vert.x
Bibliothek und Basis-Framework innerhalb einer Anwendung (ohne eigene Laufzeitumgebung)	Framework mitsamt Laufzeitumgebung sowie Lösungen für Clustering und Hochverfügbarkeit
Die Adressierung ist durch Selektoren sehr mächtig.	Nur simple Bus-Adressierung mit Strings
Nachrichten können beliebige Daten enthalten.	Nachrichten sind (in Version 2.x) primitive Datentypen oder JSON-Objekte.
Erlaubt die Auswahl des Dispatchers für ein passendes Laufzeitmodell.	Bietet mit Event-Loops eine klar strukturierte Laufzeitumgebung. Das Programmiermodell fühlt sich wie „single threaded" an.
Nachrichten werden nur lokal (in einer JVM) zugestellt. Eine Verteilung an andere Prozesse ist derzeit nicht möglich.	Bietet einen Event-Bus für Nachrichten, auch im Cluster über Prozess- und Maschinengrenzen hinweg.

Tabelle 3: Vergleich Reactor mit Vert.x

Fazit

Reaktive Anwendungen und deren grundlegende Konzepte gewinnen immer mehr an Bedeutung: Im Zeitalter stetig wachsender Anforderungen bei steigenden Request-Zahlen sind die Frage nach Skalierbarkeit und die Ausnutzung vorhandener Ressourcen (Stichwort: „Multi Core") essenziell und erfordern eine inhärent verteilte und nebenläufige Abarbeitung der Anfragen. Altbekannte Konzepte wie Reactor Pattern und Aktorenmodell helfen dabei und kommen in reaktiven Frameworks wie Vert.x und Reactor zum Einsatz.

Literatur

[1] Martin Lehmann: „Schnell reagiert! Reaktive Systeme auf der Java-Plattform mit Vert.x und Reactor". Java Forum Stuttgart 2015: *http://www.java-forum-stuttgart.de/de/Abstracts+Slot+2.html#art484*

[2] Reactive Manifesto, Version 2.0: *http://www.reactivemanifesto.org*

[3] Jonas Bonér: „Go Reactive: Event-Driven, Scalable, Resilient & Responsive Systems": *http://www.slideshare.net/jboner/going-reactive-event-driven-scalable-resilient-systems*

[4] Oleg Shelajev, „Flavors of Concurrency in Java": *http://zeroturnaround.com/rebellabs/flavors-of-concurrency-by-oleg-shelajev*

[5] Douglas C. Schmidt, Michael Stal, Hans Rohnert, Frank Buschmann, „POSA2. Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects", ISBN 0-471-60695-2. Kurzform: *http://www.dre.vanderbilt.edu/~schmidt/POSA/POSA2/event-patterns.html*

[6] Letitcrash-Video „Carl Hewitt explains the essence of the Actor Model of computation": *http://letitcrash.com/post/20964174345/carl-hewitt-explains-the-essence-of-the-actor*

[7] Vert.x Projekt-Homepage: *http://vertx.io*

[8] Techempower Benchmarks: *https://www.techempower.com/benchmarks*

[9] Lutz Huehnken, Typesafe, JAX 2015, „Von Enterprise zu Reactive": *http://de.slideshare.net/lutzh/von-enterprise-zu-reactive*

[10] Martin Thompson u.a., „Disruptor: High performance alternative to bounded queues for exchanging data between concurrent threads", Mai 2011: *https://lmax-exchange.github.io/disruptor*

[11] Jochen Mader, „Lockperformance - Project to reproduce the numbers presented in Disruptor": *https://github.com/codepitbull/lockperformance*

Martin Lehmann
martin.lehmann@accso.de

Martin Lehmann ist Diplom-Informatiker und als Chef-Technologe und Software-Architekt bei der Accso – Accelerated Solutions GmbH tätig. Seit Ende der 1990er Jahre arbeitet er in der Software-Entwicklung in diversen Projekten der Individual-Entwicklung für Kunden verschiedener Branchen. Vornehmlich beschäftigt er sich dabei mit der Konzeption technischer Architekturen und dem Software-Engineering im Zuge der Umsetzung. Sein Fokus liegt auf Neuerungen der Java-Plattform, sowohl der Programmiersprache als auch des Ökosystems, aktuell besonders bei reaktiven Frameworks.

Effiziente Software-Entwicklung mit JavaFX und JVx

Roland Hörmann, SIB Visions GmbH

Vor Kurzem wurde der Autor von einem großen deutschen IT-Dienstleister mit der Aussage konfrontiert, dass die Entwicklung von Backoffice-Lösungen nicht kostengünstig und zeitnah durchführbar sei. Das Problem wurde sowohl in aktuellen Technologie-Entwicklungen als auch in organisatorischen Abläufen vermutet.

Die Wunschvorstellung wäre gewesen, dass die Umsetzung von Anforderungen im Backoffice-Bereich nicht länger als die Erstellung von Excel-Sheets in Anspruch nehmen sollte. Wer an dieser Stelle denkt, dass Scrum und andere agile Methoden die Effizienz gravierend verbessern, der sollte das stark anzweifeln. Die Effizienz wird schlussendlich an der Umsetzungszeit der Kunden-Anforderungen gemessen. Mit den richtigen Mitteln ist das alles kein Problem.

Doch welche Mittel sind das genau? Seit Längerem geht der Trend in Richtung Web-Entwicklung mit geeigneten Full-Stack-JavaScript-Frameworks oder leichtgewichtigen Bibliotheken. Es besteht natürlich auch Bedarf an Desktop- und nativen mobilen Anwendungen. Für alle Plattformen gibt es ausreichend Frameworks und Bibliotheken, nachdem das Angebot so riesig ist, fällt die Auswahl der richtigen Mittel extrem schwer.

Selbst wenn die richtige Technologie gefunden ist, bietet das noch keine Garantie für Effizienz, denn es braucht mehr als nur Frameworks und Bibliotheken, um effizient zu sein. Wenn unstrukturiert, ohne standardisiertes Vorgehen und ohne Rücksicht auf die Qualität entwickelt wird, kann nicht von Effizienz gesprochen werden.

Das richtige Mittel muss also ein Mix aus Technologie(n) und einer standardisierten Vorgehensweise bei höchsten Qualitätsansprüchen sein. Am besten ist alles in einem leichtgewichtigen Framework vereint. Wer den heiligen Gral in der Software-Entwicklung vermutet, liegt damit gar nicht so falsch. Doch alles schön der Reihe nach.

In den Gesprächen mit dem IT-Dienstleister stellte sich heraus, dass vom Wunsch

```
CRUDTable table = new CRUDTable();

Grid grid = new Grid();
grid.setTable(table);

Screen screen = new Screen();
screen.add(grid);
screen.show();
```

Listing 1: Pseudo-Code „Bearbeitungsmaske"

nach effizienter Entwicklung von Desktop-Anwendungen die Rede war. In einigen wenigen Fällen waren Desktop-Anwendungen nicht ausreichend und daher Web-Anwendungen gefragt. Die Erstellung von mobilen Clients war nicht weiter wichtig, es war aber klar, dass sich das in den nächsten Monaten ändern könnte.

Als Beispiel nannte der Dienstleister eine Backoffice-Lösung die rund hundert Bearbeitungsmasken hatte, mit denen die Verwaltungsaufgaben eines Unternehmens durchgeführt wurden. Die Masken selbst hatten Funktionen, die als De-facto-Standard bezeichnet werden können, wie CRUD, Volltextsuche, Sortiermöglichkeiten und CSV-Export. Die Erstellung von Berichten fehlte in der Anwendung natürlich nicht. Es war auch schon selbstverständlich, dass eine Backoffice-Anwendung über eine geeignete Benutzerführung verfügt beziehungsweise benutzerfreundlich ist, dass ein einheitliches Erscheinungsbild sein muss und dass ein Hilfesystem nicht fehlen darf – natürlich sollte alles mehrbenutzerfähig und mehrsprachig sein.

Aus Sicht des IT-Dienstleisters war die Anwendung nichts Besonderes und üblich in der Branche. Aus der Sicht eines Entwicklers sieht die Sache natürlich ganz anders aus, denn mehr Funktionalität bedeutet auch eine längere Umsetzungszeit. Und genau das war dem IT-Dienstleister ein Dorn im Auge. Die Erstellung von Anwendungen mit den genannten Funktionen sollte in wenigen Tagen anstatt mehreren Monaten möglich sein. Dieser Wunsch wird auch in Zukunft immer häufiger zu hören sein. Denn die Anwendungsentwicklung ist ein enormer Kostenfaktor. Doch wie können solche Wünsche erfüllt werden?

Der Faktor „Mensch" darf beim Thema Effizienz nicht vernachlässigt werden, denn die Grundvoraussetzung effizienter Arbeit ist ein geeignetes Arbeitsumfeld, um beim Entwickeln in den richtigen „Coding Flow" zu kommen, der im Idealfall nicht unterbrochen wird. Soweit die Theorie, doch leider ist die Praxis nicht immer ideal. Aus diesem Grund müssen andere Faktoren helfen, um die Effizienz zu steigern. Eine geeignete Maßnahme wäre, ganz einfach weniger Code zu schreiben. Denn weniger Code reduziert den Aufwand für Test, Wartung und Weiterentwicklung.

Statistisch gesehen passieren auch weniger Fehler, wenn weniger Code erzeugt wird. Es bedeutet zwar nicht, dass der Code weniger komplex ist, aber auf jeden Fall ist er einfacher und schneller zu lesen, es gibt also eine raschere Fehleranalyse. Wenn von weniger Code gesprochen wird, dann ist damit nicht nur gemeint, Leerzeilen zu löschen, Variable kürzer zu benennen und Zeilenumbrüche zu entfernen. Das Ergebnis wäre zwar kürzer, aber auch unbrauchbar. Eine Code-Reduktion lässt sich durch Zentralisierung von Funktionalitäten oder den Einsatz von hilfreichen Paradigmen wie beispielsweise Don't Repeat Yourself (DRY) *[1]* oder dem Ableger Convention over Configuration (CoC) *[2]* erreichen.

Mit Zentralisierung ist nicht einfach nur gemeint, dass ähnliche Funktionen in eine zentrale Methode ausgelagert werden, um Code zu vermeiden. Dies sollte selbstverständlich sein. Überwiegend ist gemeint, dieselben Informationen nicht an mehreren Stellen in einem Projekt zu definieren. Es muss eine zentrale Stelle geben, an der diese zu finden sind, um bei Änderungen alle logisch verwandten Informationen implizit anzupassen. Mit dem DRY-Prinzip kann dies erreicht werden. Die Anwendung des Prinzips ist vom Entwickler abhängig und erfordert Konsequenz. Um den Entwickler zu entlasten, sollte auch CoC Berücksichtigung finden, denn dadurch kann auch ohne strikte Konsequenz der Sourcecode reduziert werden.

Eine Konvention ist dabei nichts anderes als eine übliche Funktionalität, die ohne Code oder Konfiguration einfach implizit vorhanden ist. Damit ist für eine Konvention einfach kein Code notwendig. Aber natürlich passen nicht immer alle Konventionen hundertprozentig. Deswegen muss es möglich sein, die Abweichung zu codieren. Das darf allerdings nicht die Regel sein, denn sonst muss die Konvention überarbeitet werden.

Durch die Anwendung von DRY und CoC kann beispielsweise eine einzelne Bearbeitungsmaske in wenigen Code-Zeilen umgesetzt werden, mit den zuvor beschriebenen Funktionalitäten (CRUD, CSV Export, Volltextsuche …). *Listing 1* zeigt mit Pseudo-Code, wie eine Maske codiert werden könnte.

Eine Konvention besagt, dass mit der „CRUDTable" ohne weitere Codierung Datensätze gelesen, erstellt, geändert und gelöscht werden können. Wenn bestimmte Operationen nicht möglich sein sollen, müssen sie deaktiviert werden, etwa mit „setInsertEnabled(false);". Eine weitere Konvention besagt, dass ein Grid automatisch alle Daten laut „CRUDTable" anzeigen, sortieren, durchsuchen und als CSV-Datei speichern kann.

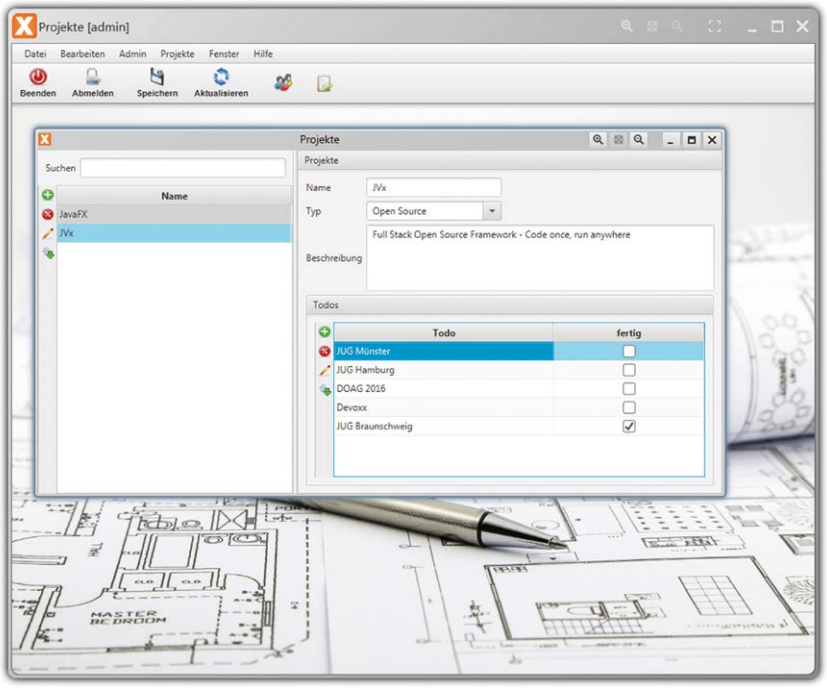


Abbildung 1: Projekt-Verwaltung als JavaFX-Applikation

Das Beispiel zeigt, dass nicht sehr viel Sourcecode nötig ist, um eine Maske zu erstellen. Es können auch nicht viele Fehler im eigenen Code sein, da durch die Anwendung der Konventionen die Komplexität vom Entwickler ferngehalten wird. Erst mit Sonderwünschen, also den Abweichungen von den Konventionen, und durch die Business-Logik würde zusätzlicher Sourcecode entstehen. Ein Entwickler muss sich jedoch ausschließlich darum kümmern. Das erleichtert einerseits die Arbeit und erhöht andererseits die Effizienz.

Theorie und Praxis

Die Anwendung von Software-Paradigmen ist natürlich mit jeglicher Technologie möglich und effiziente Software-Entwicklung sollte üblich sein. Leider ist das nur die Theorie und in der Praxis aufgrund diverser Einflüsse nicht die Regel, wie auch die Gespräche mit dem IT-Dienstleister verdeutlichten.

Nur einige wenige Frameworks wenden die zuvor genannten Paradigmen an und haben diese verinnerlicht. Erst durch den Einsatz solcher Frameworks werden Entwickler entlastet und können maximal effizient arbeiten. Eines dieser Frameworks ist das Open-Source-Java-Framework JVx *[3]*. Es wurde auf den Grundprinzipien von Best-Practice-Paradigmen erstellt und hat zum Ziel, den Sourcecode einer Anwendung auf ein Minimum zu reduzieren.

Doch JVx selbst ist kein GUI-Framework und benötigt etwas Unterstützung. An dieser Stelle kommt JavaFX ins Spiel. Als offizieller Nachfolger von Swing ist JavaFX bestens für die Erstellung von Backoffice-Anwendungen geeignet und enthält alle notwendigen GUI-Komponenten wie „TableView". Auch wenn JavaFX ganz gut ohne JVx zurechtkommt, müssten alle Paradigmen vom Entwickler selbst angewandt werden. Doch das widerspricht der gewünschten Effizienz.

Effizienz in der Praxis

Ein konkretes Beispiel mit JavaFX und JVx zeigt, wie effizient eine Bearbeitungsmaske *(siehe Abbildung 1)* erstellt werden kann, die noch dazu alle Anforderungen abdeckt. Im linken Bereich der Maske ist nun das Grid mit der Liste aller Projekte ersichtlich. Passend zum Pseudo-Code aus *Listing 1* ist es möglich, mittels Suchfeld (oberhalb des Grid) nach Projekten zu filtern. Über die Buttonleiste links vom Grid können neue Projekte erstellt, bestehende geändert, gelöscht oder als CSV-Datei exportiert werden. Im rechten Bereich befindet sich die Detail-Ansicht mit dem Namen, dem Typ und der Beschreibung zum aktuell gewählten Projekt. Darunter wird eine weitere Liste mit To-dos dargestellt. Der Funktionsumfang der Maske ist in ERP-Systemen durchaus üblich.

Ausgehend von *Listing 1* werden der Sourcecode vervollständigt und die Detail-Ansicht ergänzt. Dazu muss die „CRUDTable" mit Daten in Verbindung gebracht werden. Dies wird mit „table.setTable("PROJEKTE");" gelöst. Dadurch wird die Datenbanktabelle PROJEKTE mit dem Grid verknüpft. Mehr Code ist nicht notwendig, um vollständiges „CRUD" zu ermöglichen.

```
EditorLabel name = new EditorLabel(table, "NAME");
EditorLabel typ = new EditorLabel(table, "TYP");
EditorLabel beschreibung = new EditorLabel(table, "BESCHREIBUNG");
screen.add(name);
screen.add(typ);
screen.add(beschreibung);
```

Listing 2: Pseudo-Code der Detail-Ansicht

```
CRUDTable todos = new CRUDTable("TODOS");
Grid gridTodos = new Grid(todos);
gridTodos.setFilterable(false);
screen.add(gridTodos);
```

Listing 3: Pseudo-Code für To-dos in abgekürzter Schreibweise

```
todos.setMasterReference(table,
     new String[] {"PROJEKTE_ID", "ID"}
```

Listing 4: Master/Detail-Beziehung herstellen

Im nächsten Schritt wird die Detail-Ansicht eingefügt. Dabei handelt es sich um drei Editoren mit je einem Label und einem Grid, das jedoch keine Filterung ermöglicht. *Listing 2* enthält den Sourcecode für die Editoren. Das Grid mit den To-dos wird in *Listing 3* hinzugefügt.

Die Bindung der Editoren an die Projekte erfolgt mit der „CRUDTable" und der Angabe des Spaltennamens, aus dem die Daten gelesen und gespeichert werden. Durch CoC wird festgelegt, dass das Label aus dem Spaltennamen gebildet wird. Somit wird aus „NAME" das Label „Name". Natürlich muss auch das Label durch den Entwickler veränderbar sein (siehe „name.setLabel("Projektname");"), aber wenn die Spalten-Bezeichnung gut gewählt wurde, sollte die Konvention sehr gut funktionieren.

Die Verknüpfung der Editoren mit der Tabelle wurde bereits durchgeführt. Es fehlt nun noch die Verknüpfung der To-dos mit den Projekten, da nur die To-dos des gewählten Projekts angezeigt werden sollen. Im Fachjargon wird von Master/Detail-Beziehung gesprochen. Um die Verknüpfung herzustellen, ist der Code aus *Listing 4* vollkommen ausreichend.

Die Sourcecode-Zeile besagt, dass alle To-dos, bei denen der Wert der Datenbankspalte „PROJEKTE_ID" mit der Spalte „ID" aus der Projekte-Tabelle übereinstimmt, mit einem Projekt verknüpft werden. In knapp zwanzig Zeilen Sourcecode wurde die Maske umgesetzt; der Zeitaufwand von knapp fünf Minuten unterstreicht die Effizienz. Im konkreten Beispiel bedeutet wenig Sourcecode auch weniger Fehler und minimalen Testaufwand.

Da geht doch noch mehr

Die Maske enthält einige Details, die im Sourcecode und im Screenshot nicht ersichtlich sind. Der Editor für den Projekttyp ist eine Auswahlliste, obwohl wir keine definiert haben. Im Grid mit den To-dos wird eine Checkbox für die Spalte „FERTIG" angezeigt, obwohl auch diese nicht explizit festgelegt wurde.

Der Projektname ist in der Datenbank auf hundert Zeichen begrenzt, der Editor erlaubt ebenfalls nicht mehr. Ohne CoC müssten all diese Features manuell codiert werden. Das würde zu mehr Sourcecode führen und ein Entwickler wäre damit einige Stunden beschäftigt. Diese Zeit kann sicherlich besser eingesetzt werden, um gegebenenfalls die Business-Logik zu codieren.

Damit aus dem Projekttyp eine Auswahlliste wird, können die Metadaten der Datenbanktabelle ausgelesen werden. Durch eine Fremdschlüssel-Beziehung ist eigentlich ganz klar definiert, woher die Daten für ein bestimmtes Feld bezogen werden können. Auch die Längenbeschränkung des Projektnamens ist bereits in den Metadaten der Datenbanktabelle verankert. Die Checkbox für To-dos ergibt sich ebenfalls aus den Metadaten, wenn die möglichen Werte in der Datenbank hinterlegt wurden (Check-Constraints oder Enum-Datentyp). Wenn die Projekte noch über ein Fertigstellungsdatum verfügen würden, wäre auch damit klar definiert, dass ein Datum im Editor angezeigt werden muss.

Fazit

Gerade bei Backoffice-Lösungen oder generell großen Datenbank-Applikationen sollten die hier beschriebenen Methoden in Betracht gezogen werden. Damit können der Code und die Komplexität von Applikationen soweit reduziert werden, dass Applikationen mit beispielsweise zweihundert Masken von einem Team mit zwei Entwicklern in wenigen Monaten umgesetzt werden können – anstatt in einem Jahr. Die Praxis hat gezeigt, dass die Wartungsfälle mit beschriebener Vorgehensweise kaum der Rede wert sind und die überschüssige Zeit in neue Projekte investiert werden kann.

Links

[1] DRY: *https://de.wikipedia.org/wiki/Don%E2%80%99t_repeat_yourself*

[2] CoC: *https://en.wikipedia.org/wiki/Convention_over_configuration*

[3] JVx: *http://sourceforge.net/projects/jvx*

Roland Hörmann

roland.hoermann@sibvisions.com

Roland Hörmann ist Gründer und CEO der SIB Visions GmbH sowie Lektor an der FH Technikum Wien. Er verfügt über langjährige Erfahrung in der Entwicklung von Enterprise-Lösungen. Sein Interessenschwerpunkt liegt auf effizienter Software- und Framework-Entwicklung im Umfeld von klassischen Business-Applikationen. Er ist regelmäßig Speaker auf Oracle-User-Group-Veranstaltungen (DOAG, AOUG) sowie auch bei Java-User-Group-Events.

Elasticsearch – ein praktischer Einstieg

gelesen von Daniel Grycman

Auf dem Buchrücken versprechen der Autor Florian Hopf und der dpunkt Verlag, dass der Leser nach der Lektüre das nötige Rüstzeug besitzt, um eigene Anwendungen auf Basis von Elasticsearch zu entwickeln. Diese Rezension beleuchtet, ob das auch wirklich gelungen ist.

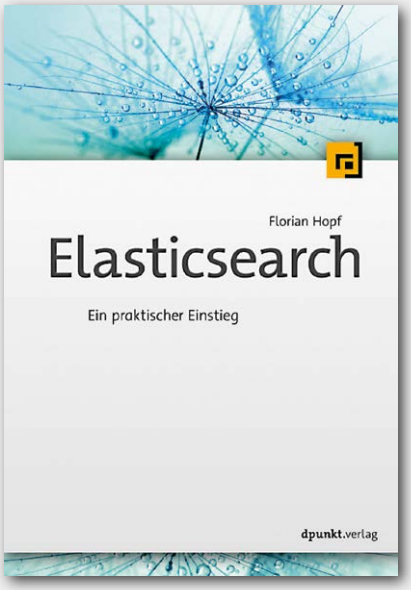
Das Buch ist in zwölf Kapitel aufgeteilt. Den Abschluss bilden zwei Anhänge, das Literaturverzeichnis und der Index. Gleich zu Beginn geht Florian Hopf auf die Geschichte von Elasticsearch ein und schafft dabei einen Bezug zu Lucene und Solr, wobei er gleichzeitig eine Abgrenzung zu Solr vornimmt.

Die ersten vier Kapitel führen in die grundlegenden Konzepte und Basics von Elasticsearch und Lucene ein, ebenso vertieft der Autor den Umgang mit Texten und Relevanz-Berechnung. Es folgt ein Kapitel zur Indizierung von verschiedenen Datenquellen. Das sechste Kapitel beschäftigt sich mit allen Aspekten der Verteilung von Daten. Die zwei folgenden Kapitel gehen auf die Modellierung und Aggregation von Daten ein. Im neunten Kapitel wird der Zugriff auf Elasticsearch mittels Java und JavaScript beschrieben.

„Elasticsearch in Produktion" ist das Thema des zehnten Kapitels, wobei Florian Hopf nicht nur die Inbetriebnahme erläutert, sondern auch Themen wie „Monitoring" und „Datensicherung" Platz gibt. Das vorletzte Kapitel widmet sich der Speicherung von Logging-Dateien, wozu Elasticsearch als Teil des ELK-Stacks benutzt werden kann. Zusätzlich wird Graylog als Alternative präsentiert. Das zwölfte und zugleich letzte Kapitel gibt einen Ausblick in die Zukunft von Elasticsearch und stellt in aller Kürze einige Themen vor, auf die im Buch aufgrund ihrer Komplexität nicht eingegangen werden kann.

Bemerkenswert ist das Expertenwissen von Florian Hopf zu Apache Solr, Lucene und Elasticsearch. Es ist ein Buch für Praktiker aus Sicht eines Praktikers. Es finden sich viele Beispiele, die zum direkten Ausprobieren einladen. Das Buch basiert auf der Version 1.6 von Elasticsearch, was für dieses Einsteigerwerk kein Problem darstellt, da in den neueren Versionen Funktionen aus der Version 1.6 erhalten geblieben sind.

Fazit: Dieses Werk eignet sich für den Leser, der auf der Suche nach einem deutschsprachigen Einsteigerbuch für Elasticsearch ist. Mithilfe der Lektüre sollten sich erste Projekte in Elasticsearch zeitnah umsetzen lassen.



Titel:	Elasticsearch
Autor:	Florian Hopf
Verlag:	dpunkt Verlag
Umfang:	262 Seiten
Preis:	32,90 Euro, auch als eBook erhältlich
ISBN:	978-3-86490-289-5

Single Sign-on mit Keycloak

Sebastian Rose, AOE GmbH

Single Sign-on (SSO) ist ein Ansatz, bei dem sich ein Benutzer einmalig bei einem SSO-Authentifizierungsdienst anmeldet und daraufhin auf alle für ihn autorisierten Daten, Dienste und Rechner zugreifen kann, ohne sich erneut bei jedem einzelnen Login anmelden zu müssen.

Der Benutzer erhält über SSO wie in der realen Welt eine einzige virtuelle Identität. Die Autorisierung bei den einzelnen Diensten wird dabei nicht umgangen, vielmehr übernimmt der SSO-Dienst die Bestätigung der Identität und damit die Authentifizierung für die einzelnen Systeme. Software-Systeme mit mehr als einem gleichzeitigen Benutzer enthalten häufig ein oder mehrere der folgenden Anforderungen:

- Autorisierung/Authentifizierung: Ein Benutzer soll sich am System mit einem Konto anmelden können
- Benutzer sollen die Daten ihres eigenen Kontos verwalten können
- Administratoren sollen benutzerübergreifende Funktionalität nutzen können
- Es gibt auf Rollen abgestimmte Inhalte und Funktionalität
- Es gibt personalisierte Inhalte und Funktionalität
- In einem verteilten System mit verschiedenen Teilen einer Anwendung soll es ausreichen, sich ein einziges Mal mit Single Sign-on (SSO) zu authentifizieren
- In einem verteilten System mit verschiedenen Anwendungsteilen soll es möglich sein, sich zentral auszuloggen (Single Log-out)

Single-Sign-on-Authentifizierungsdienste sind ein Lösungsansatz für diese Anforderungen. Keycloak ist dabei eine integrierte Open-Source-Lösung für Single Sign-on und Identitätsmanagement für Browser Apps und Web Services.

Systemsicherheit hat oberste Priorität

Der Umgang mit Benutzerdaten in einem Software-System ist heikel. Neben Datenschutzbestimmungen und anderen rechtlichen Aspekten ist im Falle von Problemen ein direkter Schaden für den Benutzer möglich. Sicherheit hat verschiedene Dimensionen, die gesamtheitlich für die System-Sicherheit auschlaggebend sind.

Die Anforderungen sind, bezogen auf die Authentifizierung, beispielsweise als Login-Fenster in der eigenen Anwendung realisiert. Dieses umfasst häufig die Funktion, sich zu registrieren, oder bietet Unterstützung im Falle eines vergessenen Passworts. Aus zeitlichen oder finanziellen Gründen kann es sinnvoll sein, für das eigene System eine Komponente zu wählen, die auf das Thema „Benutzerverwaltung und Authentifizierung" im Sinne von SSO spezialisiert ist. Ein weiterer Grund kann das fehlende Know-how sein, um selbst eine entsprechende Lösung zur Verfügung stellen zu können.

Die Open-Source-Lösung Keycloak

Eine mögliche Lösung stellt das Projekt „Keycloak" dar, eine Open-Source-Lösung unter der Regie von Bill Burk und Stian Storgersen. Es erfüllt die genannten Anforderungen; für die Integration in die eigenen Anwendungen gibt es verschiedene Möglichkeiten. Der Contribution Graph auf GitHub zeigt, wie regelmäßig und mit wie viel Unterstützung am Projekt gearbeitet wird. Die Entwicklungs- und Benutzer-Mailinglisten sind mit vielen Benutzern und Nachrichten hilfreich bei Problemen oder ermöglichen die Teilnahme an Diskussionen rund um Features und Fehler. Ende 2014 erschien die Version 1.0.0, die einen ersten Meilenstein für die Verwendbarkeit von Keycloak darstellt. Seitdem werden regelmäßig neue Releases veröffentlicht. Die aktuellste Version 1.6 ist unter „keycloak.jboss.org" zu finden.

Keycloak stellt eine Software-as-a-Service-Lösung (SAAS) für Single Sign-on dar. Grundlage für die SSO-Funktionalität sind die Standards Open ID Connect 1.0 (OIDC) und damit auch OAuth 2.0. Als Austauschformat für die sicherheitsrelevanten Informationen werden signierte JSON Web Tokens (JWT) verwendet. OIDC benutzt Prinzipien und Methoden, die in OAuth 2 definiert sind. OIDC ermöglicht es, die Identität eines Benutzers zu verifizieren, der sich zuvor gegenüber einer zentralen Instanz autorisiert hat. Jedes andere System kommt nicht mit den Credentials eines Benutzers in Kontakt, sondern kann auf Eigenschaften und Rollen eines Benutzers nur über einen Token zugreifen. Die Inhalte des Tokens können für jedes System konfiguriert werden.

Die Download-Seite des Keycloak-Projekts bietet Keycloak als Modul sowie als Stand-alone-Applikation an. Die Stand-alone-Variante ist ein Bundle, das alles Nötige enthält, um eine laufende Keycloak-Instanz zu erhalten. Ohne weitere Anpassungen und mit vorhandener Java Runtime (JRE) lässt sich Keycloak nach dem Entpacken über bin/standalone.sh starten. Danach steht die eigene Keycloak-Instanz unter *http://localhost:8080/auth* über den Browser zur Verfügung. Als Modul kann Keycloak in einem eigenen Applikationsserver deployt werden. Ebenfalls verfügbar ist Keycloak als Docker Image.

Im Browser ist die Administrationsoberfläche über *http://localhost:8080/auth/admin* erreichbar *(siehe Abbildung 1)*. Diese Administrationsanwendung ist bereits per Keycloak abgesichert. Direkt nach der Installation muss hier per admin:admin ein neues Passwort vergeben werden. Anschließend zeigt sich die Administrationsoberfläche und die Konfiguration kann beginnen.

Oberstes Element innerhalb einer Keycloak-Instanz ist ein Realm (frei übersetzt Bereich). Ein Realm bündelt alle Sicherheitsinformationen für eine Anzahl von Benutzern und Anwendungen. Bereits vorhanden

ist der Master-Realm, der eine zentrale Rolle für die Administration spielt. Dieser Artikel bezieht sich auf den Realm „myAccount", der über die Aktion „create Realm" angelegt wird. Für jede Anwendung, die innerhalb eines Systems über den Realm abgesichert werden soll, wird ein Client konfiguriert. Ebenfalls konfiguriert werden grundlegende Einstellungen wie E-Mail-Server, Passwort-Eigenschaften etc.

Als Teil des Realm gibt es die Möglichkeit, Rollen zu definieren. In den nachfolgenden Beispielen werden die Rollen „customer" und „customercare" verwendet. Im Menüeintrag „Users" kann eine Person mit dem entsprechenden Zugang und Rechten auch Benutzer anlegen. Im Menüeintrag „Clients" sind bereits Clients vorkonfiguriert. Über die dort konfigurierte Account-Anwendung kann ein vorhandener Benutzer seine eigenen Daten verwalten.

Die Architektur von Keycloak

Keycloak steht nun für die Benutzung bereit. In der schematischen Architektur ist zu sehen, welche Schnittstellen Keycloak bereitstellt und auf welche es zugreifen kann. Im Fokus von Keycloak stehen die bereits erwähnten Konzepte Realm, Client, Benutzer und Rollen *(siehe Abbildung 2)*. Auf der linken Seite beginnend, ist für die Interaktion per Mail ein Mailbackend zu konfigurieren. Per

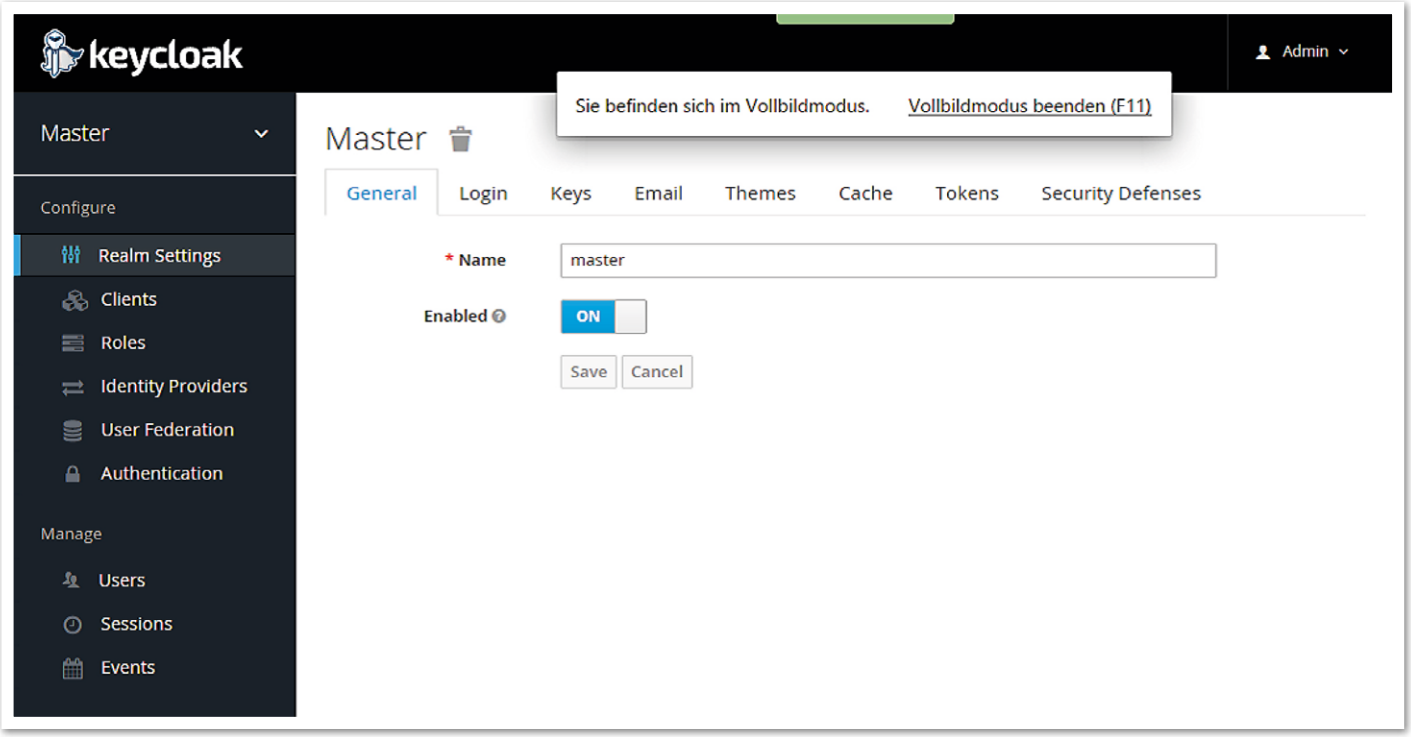


Abbildung 1: Administrationsoberfläche

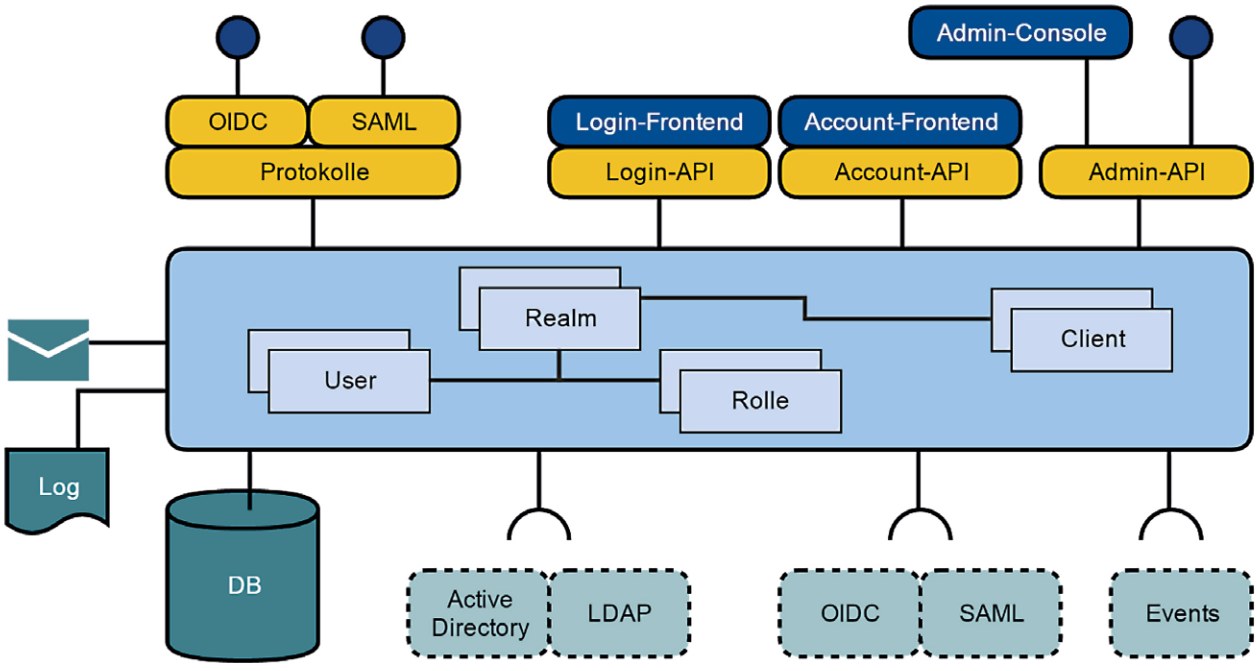


Abbildung 2: Architektur von Keycloak

Log-Konfiguration lässt sich einstellen, welche Informationen von welcher Teilkomponente im Logfile landen.

Im unteren Teil der Abbildung ist dargestellt, welche Schnittstellen das Keycloak-System verwendet oder optional verwenden kann. Keycloak benötigt eine Datenbank, um Konfiguration und Benutzer zu speichern. Ohne Änderung benutzt Keycloak eine H2-Datenbank. Über Konnektoren ist es möglich, andere Datenbank-Produkte (wie MySQL, MongoDB) zu konfigurieren *(siehe „http://keycloak.jboss.org/docs")*. Weitere Schnittstellen, die optional von Keycloak verwendet werden können, sind:

- *LDAP/Active-Directory-Benutzerverzeichnis*
  Keycloak speichert verwendete Benutzer immer lokal, sodass im Falle einer LDAP-Anbindung konfiguriert werden muss, wann und in welche Richtung die Benutzer synchronisiert werden sollen.

- *OIDC/SAML Provider*
  In der Auswahl für die Konfiguration solcher Provider findet sich neben den prominenten Social Logins (Twitter, GitHub etc.) auch die Auswahl OIDC oder OIDC Keycloak.

- *Events*
  Keycloak erlaubt es, eigene Eventhandler zu registrieren, die dann für bestimmte Events weitere Aktionen ausführen. Beispiel für ein Event ist eine vollständige Benutzer-Registrierung oder ein erfolgreicher Login. Mit Version 1.4. stehen auch die Events des Admin-API zur Verfügung.

Im oberen Teil der Abbildung sind alle Schnittstellen dargestellt, die Keycloak zur Verfügung stellt. Auf der linken Seite befinden sich die Protokoll-Schnittstellen für SAML und OIDC. Teil der OIDC-Dokumentation ist ein Meta-Konfigurations-Endpunkt, der für den Beispiel-Realm „myAccount" unter *„localhost:8080/auth/realms/myAccount/.well-known/openid-configuration"* erreichbar ist.

Listing 1 zeigt die Ausgabe des Aufrufs. Zu sehen sind die Endpunkte für die Authentifizierung und die Beschaffung von Tokens, die Teil des OAuth-2-Standards sind. Sie ermöglichen es in Kombination mit einer Client-Konfiguration, eine Anwendung mit Keycloak zu integrieren. In manchen Bibliotheken für OAuth 2 kann dieser Endpunkt konfiguriert werden. Andere Bibliotheken verwenden unter anderem die aufgeführten „auth-" und „token"-Endpunkte. Andere hier aufgeführte Endpunkte, sowie der Aufruf selbst, sind Teil von OIDC.

Die beiden Schnittstellen „Login" und „Account" bietet Keycloak als integrierte Lösung an. Im Bereich „Login" sind Web-Formulare und zugrunde liegende Endpunkte rund um das Thema „Authentifizierung und Registrierung eines Benutzers" gespeichert. Für die Verwaltung eines Benutzerkontos steht die Account-Anwendung zur Verfügung. Beides ist durch die Konfiguration steuerbar und kann in Form einer eigenen Implementierung auch vollständig ersetzt werden. Das Admin-API (inklusive zugehöriges Frontend) ist unter *„http://keycloak.github.io/docs/rest-api/index.html"* dokumentiert und kann auch direkt in eigenen Anwendungen verwendet werden, um administrative Aufgaben zu erledigen.

Keycloak aus der Benutzer-Perspektive

Keycloak besitzt neben der Administrations-Oberfläche auch alles für Web-Form-basierte Authentifizierung sowie die Verwaltung des eigenen Kontos. Ein Benutzer mit einem Konto (etwa durch den Administrator vorkonfiguriert) wird durch Aufruf von *„http://localhost:8080/auth/realms/myAccount/account"* per Login-Fenster zu Authentifizierung aufgefordert. Nach erfolgreicher Authentifizierung befindet sich der Benutzer in der Account-Verwaltung. Der Benutzer kann hier seine persönlichen Daten pflegen sowie einige administrative Tasks durchführen. Die Features bezogen auf den Login sind in der Administrations-Oberfläche zu finden *(siehe Abbildung 3)*.

Mit einigen Schaltern lassen sich hier Features wie „Passwort vergessen" und „Selbstregistrierung" aktivieren. Nach deren Aktivierung und Abmelden des bisher eingeloggten Benutzers sind die neuen Features auf dem Login-Fenster sichtbar. Weitere Einstellungen bezüglich des Logins und der Registrierung wie „Änderbarkeit des Benutzernamens" oder „Benutzername gleich E-Mail-Adresse" können ebenfalls aktiviert sein.

Inzwischen ist die Möglichkeit immer weiter verbreitet, sich über bereits existierende Accounts bei einem Dienst anzumelden. Unter dem Stichwort „Social Login" ermöglicht es auch Keycloak, den Login beispielsweise über Twitter durchzuführen, statt einen neuen Account anzulegen. Dazu muss in der Administrations-Oberfläche unter „Identity Providers" der entsprechende Anbieter konfiguriert sein.

```
{
"issuer": "http://localhost: 8080/auth/realms/myAccount",

"authorization_endpoint":
"http://localhost:8080/auth/realms/myAccount/protocol/openidconnect/auth",

"token_endpoint":
"http://localhost:8080/auth/realms/myAccount/protocol/openidconnect/token",

"userinfo_endpoint":
"http://localhost:8080/auth/realms/myAccount/protocol/openidconnect/userinfo",

"end_session_endpoint":
"http://localhost:8080/auth/realms/myAccount/protocol/openidconnect/logout",

"jwks uri": "http://localhost:8080/auth/realms/myAccount/protocol/openidconnect/certs",

"grant_types_supported": [
"authorization_code",
"refresh token",
"password"
],
```

Listing 1: Ausgabe des OIDC-Konfigurations-Endpunkts

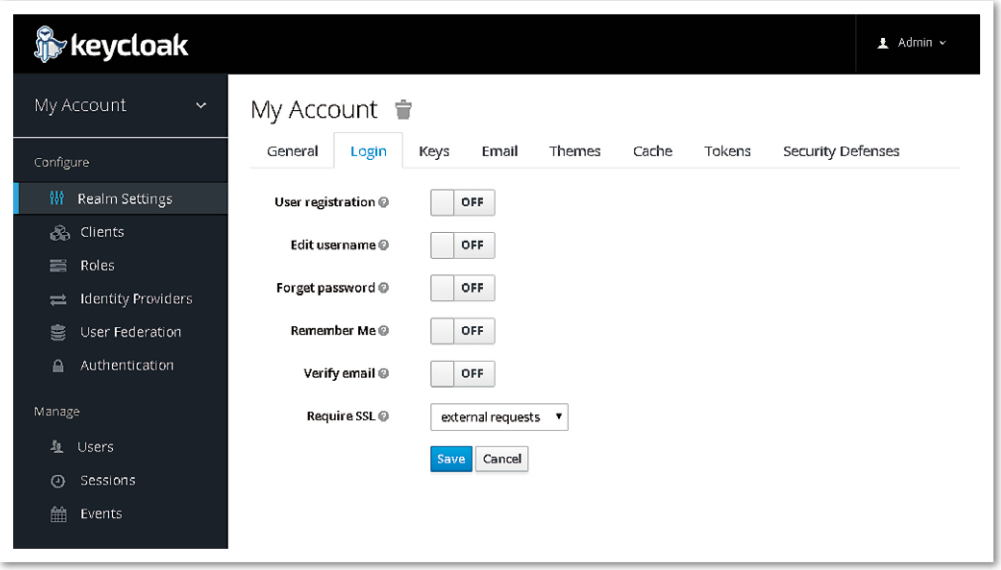


Abbildung 3: Einstellungen bezogen auf das Login-Verhalten

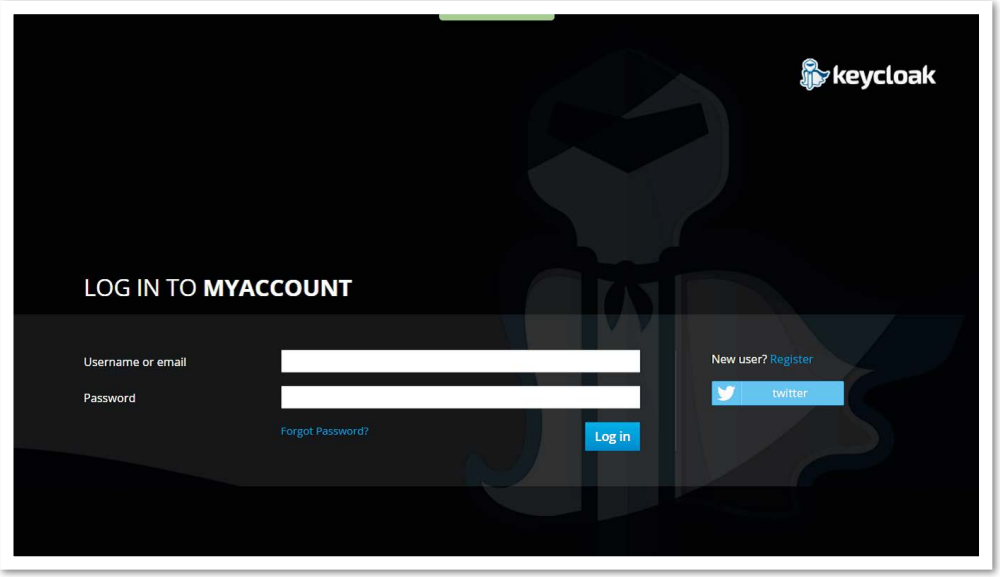


Abbildung 4: Konfigurierter Login

Die jeweils nötigen Schritte sind in der Keycloak-Dokumentation beschrieben.

Ebenfalls konfigurierbar sind Aktionen und Workflows, die seit Version 1.3 kontinuierlich ausgebaut werden. Ein Beispiel für Aktionen und Workflows ist „E-Mail verifizieren". Ist dieses Feature aktiviert, fordert Keycloak nach der Registrierung in Form eines Links per E-Mail zur Bestätigung auf. Der Account des Benutzers bleibt, solange diese Bestätigung noch nicht erfolgt ist, gekennzeichnet als unbestätigter Account. In der Administrations-Oberfläche kann „Verify E-Mail" auch als Default hinterlegt sein. Ein Login mit dem Standard-Theme von Keycloak kann nach ein paar wenigen Handgriffen wie in *Abbildung 4* aussehen.

Das Feature „Konfiguration von Themes" und die Möglichkeit, Inhalte in verschiedenen Sprachen darzustellen („Internationalisierung") ist ebenfalls in der Administrations-Oberfläche. Per Default ist die Internationalisierung ausgeschaltet. Nach deren Aktivierung ist es möglich, eine Liste von zu unterstützenden Sprachen zu definieren sowie eine Default-Sprache als Fallback anzugeben.

Mit eigenen Themes lassen sich die Oberfläche von Login und Account-Anwendungen auf eigene Layout- und Style-Wünsche anpassen. Die Themes basieren in der mitgelieferten Implementierung auf FreeMarker-Templates. Ein Beispiel wird im Keycloak-Projekt auf GitHub unter *„https://github.com/keycloak/keycloak/tree/master/examples"* mitgeliefert. Darüber hinaus können Themes dazu verwendet werden, um eigene Attribute im Datenmodell des Benutzers hinzuzufügen. Hierzu genügt es, den entsprechenden Block html/Freemarker-Template für ein Attribut im entsprechenden Theme-File zu ergänzen. Je nach Anforderung kann ein eigenes Attribut nur bei der Pflege der eigenen Daten („account.ftl") oder auch bei der Registrierung („register.ftl") zur Verfügung gestellt werden *(siehe Listing 2)*.

Dort wird der Benutzer durch ein eigenes Theme mit dem Attribut „phone" ausgestattet. Die gezeigte Änderung bewirkt, dass die Account-Seite zur Pflege der Benutzerattribute auch ein Feld für „phone" besitzt. Nicht unerwähnt bleiben soll die Möglichkeit, eigene Komponenten für die Authentifizierung („Authenticator"), Actions und Workflows zu definieren. Dazu stellt Keycloak an vielen Stellen Erweiterungsmöglichkeiten über Service Provider Interfaces *(SPI, siehe „http://keycloak.github.io/docs/userguide/keycloak-server/html/providers.html")* zur Verfügung, sodass eigene Anpassungen ohne Einschränkung möglich sind.

Keycloak für eigene Anwendungen

Um eigene Anwendungen per Keycloak abzusichern und Teil des SSO zu werden, gibt es je nach Bedarf und Art der Anwendung verschiedene Möglichkeiten. Keycloak selbst liefert Adapter für JavaScript, gängige JEE Container und Spring beziehungsweise Spring Boot. Für den Fall, dass die eigene Anwendung ohne einen solchen Adapter auskommen muss, gibt es von Keycloak selbst eine Proxy-Lösung, die die Sicherheitsinformationen per HTTP-Header an den per Proxy geschützten Server liefert. Darüber hinaus können die Standards „OpenID Connect", „OAuth 2", „Token-Validierung" und „SAML" als Integrationsgrundlage dienen. Dazu steht eine Reihe von Frameworks und Bibliotheken zur Unterstützung bereit. Es ist außerdem möglich, gegen die Standards selbst zu implementieren und so die eigene Lösung mit Keycloak zu integrieren.

Für die Absicherung von Anwendungen gibt es auf GitHub *(siehe „https://github.com/keycloak/keycloak/tree/master/examples")* einige Beispiele von Keycloak selbst. Dort ist auch das nachfolgend vorgestellte Beispiel *(„http://github/srose/keycloak-samples")* zu finden. Es besteht aus folgenden Elementen:

- Ein Backend-Service, der per HTTP eine Liste von Kunden und eine Liste von Produkten zurückliefert

- Eine JEE-Anwendung, die in einem Apache Tomcat betrieben wird und neben den persönlichen Daten des Benutzers die Liste der Produkte und die Liste der Kunden anzeigt
- Eine JavaScript-Anwendung, die die gleiche Funktionalität besitzt wie die JEE-Anwendung

Der Master-Branch („git") besteht aus den ungesicherten Anwendungen. Im Branch „secured" ist die abgesicherte Variante enthalten. Für die Absicherung wird das Backend so erweitert, dass ein Request im HTTP-Header „Authorization" einen gültigen Token enthalten muss. Die Anwendung prüft, ob dieser vorhanden ist, und führt anschließend eine Validierung des Tokens aus. Für die Validierung kann im Projekt des Backend-Service eine Abhängigkeit zu „keycloak-core" hinzugefügt werden, die die nötige Java-Klasse enthält.

Alternativ gibt es die Bibliothek „nimbus" *(siehe „http://connect2id.com/products/nimbus-jose-jwt")*, die auf die Validierung von Tokens spezialisiert ist. Mit der Keycloak-Klasse „RSATokenVerifier" wird die Methode „verifyToken" mit den Parametern Token-String, der URL des Keycloak-Servers, dem Public Key des Realm und einem Flag für die Prüfung auf einen aktiven Token aufgerufen. Ist der Token vorhanden und gültig, liefert der Service die jeweils bisherige Antwort. Ist dies nicht der Fall, liefert der Service den HTTP-Code für Not-Authorized („401").

Statt den Token in der Anwendung offline zu validieren, kann auch gegen einen Keycloak-Endpoint online validiert werden. Der Vorteil der Offline-Validierung ist die geringere Last auf dem Keycloak-Server. Nachteil ist, dass ein verwendeter Token eine gewisse Lebenszeit besitzt und, falls der Benutzer ausgeloggt wurde, der Token für eine gewisse Zeit immer noch für den Zugriff benutzt werden kann.

In diesem Zustand bekommen die beiden Anwendungen „2" und „3" wie oben beschrieben nun keinen Zugriff mehr auf die Daten des Backends, sondern werden mit „Unauthorized" abgewiesen. In beiden Anwendungen muss der Aufruf des Backends mit einem gültigen Token erfolgen. Einen gültigen Token zu erhalten, ist Teil der Absicherung, die nun online mit dem Keycloak-Server erfolgen muss. Hier beschrieben ist eine Absicherung über das OIDC-Protokoll und die dazu von Keycloak zur Verfügung gestellten Adapter.

Die Absicherung der JEE-Anwendung erfolgt über den Keycloak-Adapter *(siehe „http://keycloak.jboss.org/keycloak/downloads.html?dir=0%3Dadapters/keycloak-oidc%3B")* für den Apache Tomcat, da die Anwendung dort zum Einsatz kommt. Dazu ist es notwendig, den Keycloak Tomcat Adapter im „lib"-Verzeichnis des Apache Tomcat zu entpacken. Das Komponenten-Projekt selbst benötigt eine Abhängigkeit auf Keycloak-Adapter mit dem Scope „provided" in der Datei „pom.xml". Dem Projekt hinzugefügt werden muss eine „context.xml"-Datei im Verzeichnis „META-INF" mit Valve und dem Verweis auf die Klasse „KeycloakAuthenticatorValve". In der bereits vor der Absicherung vorhandenen „web.xml" müssen „login-config"-, „security-role"- und „security-constraint"-Einträge vorgenommen werden, die zur Keycloak-Konfiguration (bereits konfigurierte Rollen) passen müssen und die Anforderungen berücksichtigen. Im vorliegenden Beispiel wird für den Teil der Anwendung, die über das URL-Pattern „/products/*" erreicht wird, die Rolle „customer" gefordert.

In Keycloak muss über die Administrations-Konsole eine Client-Konfiguration hinzugefügt werden. Die anzulegende Konfiguration wird mit dem Client-Protokoll „openid-connect" und dem Access Type „confidential" konfiguriert (siehe „OAuth 2 authorization code grant"). Mindestens eine gültige Redirect-URL muss ebenfalls konfiguriert seine. Sie enthält im vorliegenden Beispiel die URL der abzusichernden Anwendung mit einer Wildcard („*") am Ende. Die so erzeugte Client-Konfiguration kann im Bereich „Installation" im Format „Keycloak Json" exportiert werden. Dieses Teil muss in der Anwendung im „WEB-INF"-Verzeichnis als „keycloak.json" gespeichert sein.

Im Quellcode kann durch die so konfigurierte Anwendung beim Zugriff auf das Backend ein Token aus dem Kontext mitgeschickt werden. Die Methode „getProducts" im „ProductsController" ist so modifiziert, dass zunächst der „KeycloakSecurityContext" extrahiert wird. Teil dieses Kontexts ist der Token, der über die Methode „getTokenString" geholt werden kann. Anschließend wird dieser String als Teil des HTTP-Headers („Authorization") an das Backend übergeben. Die Gewährleistung, dass der Token vorliegt und aktuell ist, erfolgt durch den konfigurierten Keycloak-Adapter.

Die Absicherung der JS-Anwendung erfolgt über den Keycloak-JS-Adapter. Dazu muss zunächst der Adapter in die eigene Seite eingebunden sein. Der Adapter wird in Form einer JS-Bibliothek vom Keycloak-Server bereitgestellt und ist in der „index.html"-Datei des Beispiels über *„http://<Keycloak URL>/auth/js/keycloak.js"* eingebunden. Der eigene JS-Code in der Datei „app.js" enthält zwei zusätzliche Zeilen für die Initialisierung.

```
<div class="form-group ${messagesPerField.printIfExists('lastName','has-error')}">
  <div class="col-sm-2 col-md-2">
     <label for="lastName" class="control-label">${msg("lastName")}</label> <span class="required">*</span>
  </div>

  <div class="col-sm-10 col-md-10">
     <input type="text" class="form-control" id="lastName" name="lastName" value="${(account.lastName!'')?html}"/>
  </div>
</div>
<div class="form-group">
  <div class="col-sm-2 col-md-2">
     <label for="user.attributes.phone" class="control-label">${msg("phone")}</label>
  </div>

  <div class="col-sm-10 col-md-10">
     <input type="text" class="form-control" id="user.attributes.phone" name="user.attributes.phone" value="${(account.
attributes.phone!'')?html} "/>
  </div>
</div>
```

Listing 2: Eigenes Benutzerattribut „phone" in account.ftl

Die im Keycloak nötige Client-Konfiguration liegt als exportierte „keycloak.json"-Datei innerhalb der Anwendung. Die Konfiguration ähnelt der in Anwendung „2" (siehe oben). Da es sich um eine JS-Anwendung handelt, wird der Access-Type „Public" verwendet (siehe „OAuth2 implicit grant"); dieser wird bei der Konstruktion eines Adapter-Objekts übergeben. Anschließend wird per „init"-Methode auf dem Adapter angegeben, dass direkt beim Laden eine Authentifizierung erforderlich ist („onLoad: 'login-required'") und welche Methode im Erfolgsfall benutzt werden soll. Jede Funktion, die Authentifizierungs-Informationen benötigt, wird durch die Methode „withSecureContext" mit dem nötigen Kontext versorgt. Beim Aufruf der Backend-Endpunkte wird ähnlich wie zuvor der aktuelle Token aus dem Keycloak-Adapter im Authorization-HTTP-Header mitgeschickt.

Mit den nun abgesicherten und im Realm „myAccount" konfigurierten Clients erfolgt beim Zugriff auf eine geschützte Ressource zunächst ein Redirect auf die Login-Seite des Keycloak-Servers. Sobald sich der Benutzer dort erfolgreich authentifiziert hat, erfolgt wiederum ein Redirect auf die Anwendung. Der Anwendung steht dann ein Token zur Verfügung, um geschützte und personalisierte Inhalte anzuzeigen. Für das vorliegende Beispiel benötigt der Benutzer die Rolle „customer". Beim Zugriff mit einer anderen Anwendung auf eine ebenfalls geschützte Ressource erfolgt wiederum ein Redirect auf den Keycloak-Server. Im Falle eines bereits authentifizierten Benutzers erfolgt sofort der Redirect auf die Anwendung. Auch hier steht der Anwendung ein Token zur Verfügung, um geschützte und personalisierte Inhalte anzuzeigen.

Fazit

Keycloak bietet zahlreiche Features für die einfache Integration von Anwendungen in einen SSO-Verbund. Die zur Verfügung gestellte Funktionalität ermöglicht die Abdeckung einer Vielzahl von Anforderungen, ohne dass viel eigener Code geschrieben werden muss. Die Implementierung ist konform zu den Standards „OIDC 1.0" und „OAuth 2.0". Auch für ausgefallene Anwendungen bietet Keycloak Möglichkeiten, per Proxy zu integrieren. Das Projekt ist sehr aktiv und die regelmäßigen Releases sowie die umfangreiche Roadmap lassen auf eine zuverlässige Entwicklung in der Zukunft schließen.

Sebastian Rose
sebastian.rose@aoe.com

Sebastian Rose ist Software-Entwickler bei der AOE GmbH. Er interessiert sich für Themen rund um Integration in Unternehmensanwendungen und Handels-Plattformen. Dabei ist ein Schwerpunkt die Nutzung von Keycloak als SSO-Komponente. Außerdem interessiert er sich für Scala und aktorbasierte Systeme.

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: © benidict83/123RF
Foto S. 14 © lightwise/123RF
Foto S. 18 © jozefmici/123RF
Foto S. 37 © Sashkin/fotolia
Foto S. 55 © krasimiranevenova/123RF
Foto S. 60 © Shutter999/123RF
Foto S. 64 © Artsen Martysiuk/ fotolia

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

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

Inserentenverzeichnis

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

X

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 Rechung. 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.

March 2017,
28th-30th

JavaLand 2017

in Phantasialand | Bruehl near Cologne

Save | the | Date

JavaLand 2017