KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

05/2016

Jahrgang 9

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

Sie können die elektronische Form des KaffeeKlatschs
monatlich, kostenlos und unverbindlich
durch eine E-Mail an

abo@bookware.de

abonnieren.

Ihre E-Mail-Adresse wird ausschließlich für den Versand
des KaffeeKlatschs verwendet.

Editorial

Hundert

Wenn mich einer vor mehr als hundert Monaten gefragt hätte, ob ich mir hundert Ausgaben KaffeeKlatsch vorstellen kann, dann hätte ich es bestimmt verneint.

Rückblickend ist immer alles so selbstverständlich. Unzählige Autoren haben in liebevoller Arbeit Unglaubliches geleistet und wohlwollende Leser haben Leserbriefe und insbesondere das Allerletzte beigesteuert. Und all das zusammen hat schon hundert Mal für ein unterhaltsames und lehrreiches Lesevergnügen gesorgt.

Seit dem hat sich aber vieles verändert. Der Ruf nach einer E-Book-Version des KaffeeKlatsch wird immer lauter. Denn während 2008 noch weniger als 6 % in Deutschland ein Smartphone besaßen, sind es heute schon mehr als 46 %.

Dann will die New York Times das Wort „Internet“ verkleinern. Offensichtlich genügen ein paar Jahre und das Internet verdient es im Englischen nicht mehr großgeschrieben zu werden. Man hat sich dazu entschlossen ab 1. Juni dem Rat der American Copy Editors Society zu folgen, die durch die Kleinschreibung das Internet zu einem „normalen Ding des Alltags“ machen wollen.

Wie dem auch sei, ich möchte mich an dieser Stelle ganz besonders bei allen Autorinnen und Autoren und nicht weniger bei allen treuen Leserinnen und Lesern bedanken, dass sie die Existenz des KaffeeKlatsch möglich gemacht haben.

Ihr Michael Wiedeking
Herausgeber

Beitragsinformation

Der KaffeeKlatsch dient Entwicklern, Architekten, Projektleitern und Entscheidern als Kommunikationsplattform. Er soll neben dem Know-how-Transfer von Technologien (insbesondere Java und .NET) auch auf einfache Weise die Publikation von Projekt- und Erfahrungsberichten ermöglichen.

Beiträge

Um einen Beitrag im KaffeeKlatsch veröffentlichen zu können, müssen Sie prüfen, ob Ihr Beitrag den folgenden Mindestanforderungen genügt:

- Ist das Thema von Interesse für Entwickler, Architekten, Projektleiter oder Entscheider, speziell wenn sich diese mit der Java- oder .NET-Technologie beschäftigen?
- Ist der Artikel für diese Zielgruppe bei der Arbeit mit Java oder .NET relevant oder hilfreich?
- Genügt die Arbeit den üblichen professionellen Standards für Artikel in Bezug auf Sprache und Erscheinungsbild?

Wenn Sie uns einen solchen Artikel, um ihn in diesem Medium zu veröffentlichen, zukommen lassen, dann übertragen Sie Bookware unwiderruflich das nicht exklusive, weltweit geltende Recht

- diesen Artikel bei Annahme durch die Redaktion im KaffeeKlatsch zu veröffentlichen
- diesen Artikel nach Belieben in elektronischer oder gedruckter Form zu verbreiten
- diesen Artikel in der Bookware-Bibliothek zu veröffentlichen
- den Nutzern zu erlauben diesen Artikel für nicht-kommerzielle Zwecke, insbesondere für Weiterbildung und Forschung, zu kopieren und zu verteilen.

Wir möchten deshalb keine Artikel veröffentlichen, die bereits in anderen Print- oder Online-Medien veröffentlicht worden sind.

Selbstverständlich bleibt das Copyright auch bei Ihnen und Bookware wird jede Anfrage für eine kommerzielle Nutzung direkt an Sie weiterleiten.

Die Beiträge sollten in elektronischer Form via E-Mail an redaktion@bookware.de geschickt werden.

Auf Wunsch stellen wir dem Autor seinen Artikel als unveränderlichen PDF-Nachdruck in der kanonischen KaffeeKlatsch-Form zur Verfügung, für den er ein unwiderrufliches, nicht-exklusives Nutzungsrecht erhält.

Leserbriefe

Leserbriefe werden nur dann akzeptiert, wenn sie mit vollständigem Namen, Anschrift und E-Mail-Adresse versehen sind. Die Redaktion behält sich vor, Leserbriefe – auch gekürzt – zu veröffentlichen, wenn dem nicht explizit widersprochen wurde.

Sobald ein Leserbrief (oder auch Artikel) als direkte Kritik zu einem bereits veröffentlichten Beitrag aufgefasst werden kann, behält sich die Redaktion vor, die Veröffentlichung jener Beiträge zu verzögern, so dass der Kritisierte die Möglichkeit hat, auf die Kritik in der selben Ausgabe zu reagieren.

Leserbriefe schicken Sie bitte an leserbrief@bookware.de. Für Fragen und Wünsche zu Nachdrucken, Kopien von Berichten oder Referenzen wenden Sie sich bitte direkt an die Autoren.

Werbung ist Information

Firmen haben die Möglichkeit Werbung im KaffeeKlatsch unterzubringen. Der Werbeteil ist in drei Teile gegliedert:

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

Die Werbeflächen werden als Vielfaches von Sechsteln und Vierteln einer DIN-A4-Seite zur Verfügung gestellt.

Der Werbeplatz kann bei Frau Natalia Wilhelm via E-Mail an anzeigen@bookware.de oder telefonisch unter 09131/8903-90 gebucht werden.

Abonnement

Der KaffeeKlatsch erscheint zur Zeit monatlich. Die jeweils aktuelle Version wird nur via E-Mail als PDF-Dokument versandt. Sie können den KaffeeKlatsch via E-Mail an abo@bookware.de oder über das Internet unter www.bookware.de/abo bestellen. Selbstverständlich können Sie das Abo jederzeit und ohne Angabe von Gründen sowohl via E-Mail als auch übers Internet kündigen.

Ältere Versionen können einfach über das Internet als Download unter www.bookware.de/archiv bezogen werden.

Auf Wunsch schicken wir Ihnen auch ein gedrucktes Exemplar. Da es sich dabei um einzelne Exemplare handelt, erkundigen Sie sich bitte wegen der Preise und Versandkosten bei Natalia Wilhelm via E-Mail unter natalia.wilhelm@bookware.de oder telefonisch unter 09131/8903-90.

Copyright

Impressum

KaffeeKlatsch Jahrgang 9, Nummer 5, Mai 2016
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-90
Telefax: 0 91 31 / 89 03-99
E-Mail: redaktion@bookware.de
Internet: www.bookware.de

Herausgeber/Redakteur: Michael Wiedeking
Anzeigen: Natalia Wilhelm
Grafik: Nicole Delong-Buchanan

Inhalt

Artikel

Kolumnen

Entwicklung mit Raketengeschwindigkeit

von Hristiyan Pehlivanov

Der Alltag von vielen Java-Entwicklern ist von einem Zyklus geprägt – Code, Build, Redeploy, Test. Die Zeit, die man in großen Projekten durch Warten auf Build und Redeployment verliert, ist nicht zu unterschätzen. Wäre es nicht schön, eine Code-Änderung sofort auf dem laufenden Testsystem zu sehen? Genau das ermöglicht JRebel! Dabei kann es nicht nur Code-Änderungen, sondern auch die Konfiguration von Frameworks neu laden. Dieser Artikel stellt die Funktionalität von JRebel vor und erläutert, wie dieser „Zaubertrick" möglich ist.

Heute Abend gibt es REST im Java-Haus

von Ángel Salces Sillero

RESTful Web-Services werden infolge einer steigenden Tendenz zur Vereinfachung der Kommunikation zwischen Services immer mehr benutzt. Dabei wird SOAP durch REST ersetzt und JSON zunehmend durch XML. Dieser Artikel zeigt, wie man mittels Java EE 7 einen JSON-konsumierenden REST-Client implementieren kann.

Königsdisziplin

von Michael Wiedeking

Seit fast einhundert vergnüglichen Kolumnen wurde die schwierigste aller Disziplinen geschickt umschifft: das Dividieren. Hier nun endlich – allerdings der Division gebührend in mehreren Teilen – die Lösung, wie man beliebig große Zahlen teilt.

Entwicklung mit Raketengeschwindigkeit

JRebel – kein Warten auf Redeployments mehr

von Hristiyan Pehlivanov

Der Alltag von vielen Java-Entwicklern ist von einem Zyklus geprägt – Code, Build, Redeploy, Test. Die Zeit, die man in großen Projekten durch Warten auf Build und Redeployment verliert, ist nicht zu unterschätzen. Wäre es nicht schön, eine Code-Änderung sofort auf dem laufenden Testsystem zu sehen? Genau das ermöglicht JRebel! Dabei kann es nicht nur Code-Änderungen sondern auch die Konfiguration von Frameworks neu laden. Dieser Artikel stellt die Funktionalität von JRebel vor und erläutert, wie dieser „Zaubertrick" möglich ist.

Stellen Sie sich vor, dass eine Uhr neben Ihrer Tastatur stehen würde. Sie soll die Zeit messen, die der Entwickler blockiert ist, weil er auf Build oder Redeployment wartet. In vielen Projekten wird diese Uhr tagtäglich eine beträchtliche Menge an verlorenen Minuten anzeigen. Dabei enthält diese Summe nicht mal die Zeit, die man wegen der Unterbrechung der Arbeit abgelenkt wird und sich nun erneut konzentrieren muss. Etwas anderes auf die Schnelle zu erledigen ist zwar praktisch, überschreibt aber den „Arbeitsspeicher" des Gehirns. Gibt es in der Tat keine Alternative?

Problemstellung

Wenn ein Entwickler so lange auf Build und Redeployment warten muss, liegt das grundlegende Problem meistens am Aufbau des Systems. Der Entwickler muss das komplette System deployen, selbst wenn er nur einzelne kleine Komponenten testen will. Den Build-Prozess könnte man theoretisch nur auf diese Komponenten einschränken. Man kann es aber nicht vermeiden, die ganze Applikation neu zu starten, damit die Änderungen geladen werden.

In einer idealen Welt würde man seine Komponente auch isoliert testen können. Leider darf ein Software-Entwickler nicht immer die Architekturentscheidungen treffen oder das bestehende System umbauen. Große Software-Projekte enthalten viele Kompromisse und Altlasten. Deshalb nimmt dieser Artikel an, dass der Entwickler aus diversen Gründen das komplette System deployen muss, damit die Tests sinnvoll funktionieren. In diesem Fall wäre eine Alternative, dass man nicht die ganze Applikation immer wieder neu startet, sondern nur einzelne Komponenten zur Laufzeit austauschen lässt.

Java HotSwap

Die Idee, Java-Code zur Laufzeit neu zu laden, existiert schon lange. Bereits mit Java 1.4 ist das HotSwap-Feature als Teil der Java Platform Debugger Architecture (JPDA) [1] vorgestellt worden. HotSwap erlaubt es, sich mit der Java Virtual Machine (JVM) zu verbinden und den Byte-Code einer Klasse neu zu laden. Die initiale Begeisterung des Entwicklers verschwindet aber, wenn man den folgenden kleinen Haken bemerkt. HotSwap erlaubt keine Änderungen am Klassenschema. Das bedeutet, dass man keine Felder und Methoden hinzufügen oder löschen darf. Als Ergebnis kann man nur die Implementierung einer Methode ändern, ohne die Signatur der Methode anzupassen.

Die Gründe für diese Einschränkung scheinen auf den ersten Blick verwirrend zu sein. Sollte es nicht ausreichen, die neue Version der Klasse einfach zu laden? Danach können die bestehenden Objekte auf die neue Klasse zugreifen. Fertig! Leider ist das nicht möglich. Die JVM überprüft, ob eine Klasse bereits geladen worden ist und erlaubt das Überschreiben geladener Klassen nicht, falls das Klassenschema sich geändert hat. Die technische Erklärung würde den Rahmen dieses Kurzartikels überschreiten.

Grundlegende Komponenten der JVM, wie der Just-In-Time-Compiler und der Garbage Collector, verlassen sich aber darauf, dass eine geladene Klasse unverändert bleibt. Das deutet die Komplexität der Aufgabe an, falls

man Java HotSwap erweitern will, um auch Änderungen am Klassenschema zu ermöglichen. Als vertiefende Informationsquelle zu diesem Thema empfiehlt sich die wissenschaftliche Arbeit von Mikhail Dmitriev [2][3].

JRebel

Im Gegensatz zu HotSwap erlaubt JRebel Änderungen im Klassenschema. Außerdem bietet es Integration mit den populärsten Frameworks, Application Servern und IDEs an. Der Entwickler kann den Java-Code oder die Konfigurationsdateien des Frameworks bearbeiten und JRebel wird die aktualisierte Version in der laufenden JVM neu laden. Dafür sind keine Änderungen im Quellcode oder in der Architektur der Applikation notwendig. Da JRebel die Möglichkeit ausnutzt, seinen Byte-Code über das Instrumentation API [4] hinzuzufügen, funktioniert es mit allen Technologien, die auf Java-Byte-Code basieren, z. B. Groovy oder Scala.

Aus Entwicklersicht unterstützt JRebel fast alle Arten von Code-Änderungen. Man kann Methoden, Felder, Konstruktoren und Annotationen hinzufügen sowie löschen. Außerdem ist es möglich, Interfaces und statische Felder zu ändern. Man kann sogar Klassenhierarchien umbauen, ohne dass die Applikation neu gestartet werden muss. Ein weiterer Vorteil besteht darin, dass der Zustand der Instanzen einer Klasse beibehalten wird. Somit muss man komplexe Zustände nicht immer wieder nachbauen, um eine Änderung zu testen.

Was aber JRebel von anderen ähnlichen Tools unterscheidet, ist der Framework-Support. Die oben genannten Features werden teilweise auch von Open-Source-Tools unterstützt, wie z. B. HotSwap Agent [5] oder Spring Loaded [6]. Heutzutage werden Frameworks in den meisten Projekten eingesetzt. In der Tat haben sie einen wichtigen Einfluss auf das Verhalten der Applikation. Das ist nicht nur auf klassischen Java-Code sondern auch auf Konfigurationsdateien und Annotationen zurückzuführen. Aus diesem Grund ist es so wichtig, dass man auch Änderungen der Framework-Konfiguration neu laden kann. Kein anderes Tool unterstützt annähernd so viele Technologien wie JRebel: JSF, Hibernate, Spring, Wicket, DeltaSpike u. v. m.

Konfiguration von JRebel

Falls man JRebel ausprobieren will, empfiehlt es sich das passende IDE-Plugin zu verwenden. Solche Erweiterungen existieren für Eclipse, IntelliJ, NetBeans usw. Das ist allerdings kein Muss. Es funktioniert auch, wenn man das JRebel-Jar herunterlädt und beim Start der JVM den Parameter *-javaagent:jarpath[=Options]* mit den passenden Optionen angibt. Außerdem ist nur noch eine Konfigurationsdatei notwendig (*rebel.xml*), die den Pfad zu den Source-Dateien enthält. Damit kann JRebel den Zeitstempel der kompilierten Dateien im Dateisystem beobachten und eine aktualisierte Datei automatisch neu laden.

Falls die Konfiguration erfolgreich ist, kommt beim Start der JVM als erstes eine Meldung auf der Konsole von JRebel. Wenn man später den Source-Code bearbeitet, kann man dank der folgenden Konsolenausgabe erkennen, welche Klassen neu geladen worden sind:

```
Redefined all recompiled classes that are loaded in the
debugee process.
JRebel: Reloading class 'de.test.MyBeanClass$1'
```

Wie funktioniert es?

Aus Entwicklersicht ist es sehr spannend, wie genau diese Technologie funktioniert. Über JRebel ist aber soweit relativ wenig Information veröffentlich worden, weil es sich um ein kommerzielles Tool handelt. Die Firma hinter JRebel, Zeroturnaround, schützt die Details ihres technischen Wissens, führt jedoch einen Blog auf ihrer Webseite, der kleine Einblicke hinter die Kulissen gibt.

JRebel schließt sich als Java-Agent [7] an der JVM an und fängt alle Klassen beim Laden ab. Dann fügt es den eigenen Byte-Code über das Instrumentation API [8] hinzu und ändert die Interaktion zwischen den Klassen. Ab diesem Moment wird eine Umleitung eingeführt und keine Klasse ruft direkt eine andere auf. Stattdessen geht alles über einen zentralen Dienst, der die passende Zielklasse findet.

Man kann in Java eine bereits geladene Klasse nicht erneut laden, falls das Klassenschema sich geändert hat. Man kann aber beliebig viele neue Klassen laden. Genau das passiert jedes Mal, wenn JRebel eine Änderung entdeckt – die aktualisierte Klasse wird mit neuer Identität in die JVM geladen. Danach verweist der zentrale Dienst auf die neue Version. Nehmen wir die folgende Klasse *C* als Beispiel.

```
public class C {
  D d;

  int method1(int x) {
    return d.get(x);
  }
}
```

Nachdem JRebel seine Byte-Code-Magie [9] anwendet, sollte die Methode *methode1* so aussehen:

```
int method1(int x) {
   Object[] o = new Object[1];
   o[0] = x;
   return Runtime.redirect(d, o, "D", "get", "I" );
}
```

Runtime.redirect() nimmt das aufgerufene Objekt und die in einem Array gespeicherten Eingabeparameter an. Die *String*-Parameter beschreiben die Zielklasse – Klassenname, Methodenname und Rückgabetyp der Methode. Damit verwendet der Dienst die ursprüngliche Instanz *d*, verweist aber auf die neuste Version der Klasse *D*.

Framework-Integration

Wie früher erwähnt, ermöglicht JRebel die Integration mit vielen populären Frameworks. Ein Beispiel wäre, dass man neue Bean-Klassen im Spring-Framework registriert und sie als Abhängigkeiten in bestehenden Klassen hinzufügt. JRebel wird die Konfiguration aktualisieren und man kann sofort die neuen Beans in der laufenden Applikation sehen.

Die technische Erklärung im Zeroturnaround-Blog, wie genau das funktioniert, geht nicht ins Detail. Es wird aber darauf hingewiesen, dass die Entwickler von JRebel das Verhalten der Frameworks analysiert und eine Vielzahl von Workarounds implementiert haben. Dabei spricht man von Workarounds, weil solche Eingriffe ins Framework generell nicht vorgesehen sind und ein offizielles API für solche Zwecke fehlt.

Laut eines Artikels von Simon Maple (Entwickler bei Zeroturnaround) [10] kann JRebel bestimmte Teile des Framework-Codes ausführen, die die Konfiguration aktualisieren oder Komponenten neu laden. Dabei entstehen andere Probleme, weil man manche Methoden nur beim Start ausführen darf, ohne das funktionierende System zu beeinträchtigen. Das JRebel-Team setzt auch hier Byte-Code-Instrumentation ein, um ins Framework-Verhalten einzugreifen. Wie Simon Maple erklärt, ist das Ergebnis alles andere als perfekt und muss mit jeder neuen Version des Frameworks angepasst werden. Diese Problematik bleibt aber dem Nutzer von JRebel erspart.

Wo ist der Haken?

Alle oben beschriebene Features bekommt man nicht umsonst. JRebel ist ein kommerzielles Produkt und die Lizenzen müssen jährlich verlängert werden. Hier muss jeder selbst die Entscheidung treffen, ob das Preis-Leistungs-Verhältnis stimmt. Dabei ist zu berücksichtigen, dass die Arbeitszeit der Entwickler oft die teuerste Investition in einem Software-Projekt ist. Dann würde die Frage aus der Praxis kommen, ob JRebel genug Zeit spart. Falls die Antwort „nein“ wäre, könnten die Open-Source-Alternativen (*HotSwap Agent*, *Spring Loaded*) von Interesse sein.

Bei Interesse an JRebel empfiehlt es sich, Zeroturnaround zu kontaktieren, falls das Tool mit dem eigenen Technologie-Stack nicht funktioniert. Die verschiedenen Versionen vieler Frameworks zu unterstützen, ist eine anspruchsvolle Aufgabe. Es ist unwahrscheinlich, dass die Integration mit JRebel immer reibungslos funktionieren wird. Kunde zu sein, hat aber auch seine Vorteile. Man bekommt nämlich Unterstützung.

Fazit

Der Zeitverlust durch Warten auf Redeployments ist nicht zu unterschätzen. JRebel unterstützt die wichtigsten Code-Änderungen und kann die aktualisierten Java-Klassen zur Laufzeit neu laden, ohne die Applikation neu zu starten. Außerdem wird der Zustand beibehalten. Darüber hinaus steht Integration mit einer Vielzahl von Frameworks, IDEs und Application Servern zur Verfügung. Somit kann der Entwickler effizienter arbeiten und gleichzeitig mehr Spaß daran haben.

Referenzen

[1] Oracle Java *Documentation*, https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/enhancements1.4.html, 2016

[2] Dmitriev, Mikhail *Towards Flexible and Safe Technology for Runtime Evolution of Java Language Applications*, In Proceedings of the Workshop on Engineering Complext Object-Oriented Systems for Evolution, in association with OOPSLA 2001 International Conference, 2001

[3] Dmitriev, Mikhail Safe Class and Data Evolution in Large and Long-Lived Java Applications, PhD Thesis, University of Glasgow, 2001

[4] Oracle Java *Documentation*, https://docs.oracle.com/javase/8/docs/api/java/lang/instrument/Instrumentation.html, 2016

[5] HotSwap Agent *Open-Source-Erweiterung von Java HotSwap*, http://www.hotswapagent.org

[6] Spring Loaded *Open-Source-Javaagent,* https://github.com/spring-projects/spring-loaded

[7] Oracle Java *Documentation*, http://docs.oracle.com/javase/8/docs/api/java/lang/instrument/package-summary.html, 2016

[8] Shelajev, Oleg *How-to guide to writing a javaagent*, http://zeroturnaround.com/rebellabs/how-to-inspect-classes-in-your-jvm, Mai 2015

[9] Maple, Simon *Why HotSwap wasn't good enough in 2011...and still isn't today*, http://zeroturnaround.com/rebellabs/why-hotswap-wasnt-good-enough-in-2001-and-still-isnt-today, Mai 2014

[10] Maple, Simon *If you think Java development is already slow, just wait till you add a Java Web Framework*, http://zeroturnaround.com/rebellabs/if-you-think-java-development-is-already-slow-just-wait-till-you-add-a-java-web-framework, Juni 2014

Kurzbiografie

Hristiyan Pehlivanov ist als Entwickler und Consultant bei MATHEMA Software GmbH im Java Umfeld tätig. Er ist leidenschaftlicher Agilist und Clean Code Anhänger, der immer auf der Suche nach neuen Herausforderungen ist. Außerdem glaubt er an die kontinuierliche Evolution von Systemen, die über leichtgewichtige Lösungen und Testautomatisierung ermöglicht wird.

Heute Abend gibt es REST im Java-Haus

JAX-RS und JSON API im Einsatz

von Ángel Salces Sillero

RESTful Web-Services werden infolge einer steigenden Tendenz zur Vereinfachung der Kommunikation zwischen Services immer mehr benutzt. Dabei wird SOAP durch REST ersetzt und JSON zunehmend durch XML. Dieser Artikel zeigt, wie man mittels Java EE 7 einen JSON-konsumierenden REST-Client implementieren kann.

Eine Datenquelle auswählen

Es gibt zahlreiche Anwendungen im Internet, die eine REST-API zur Verfügung stellen. Um effizienter nach einer geeigneten API zu suchen, bieten einige WebSites einen Repository-Browser an. Ein Beispiel dafür ist ProgrammableWeb [1].

Der WebClient im kommenden Beispiel fragt Daten von OpenWeatherMap ab, das verschiedene Sorten von Wetterinformationen anbietet. Auf der Website werden die dazu entsprechenden REST-APIs ausführlich erläutert [2]. Um Daten abrufen zu können, wird eine Registrierung vorausgesetzt. Danach erhält man einen Applikationsschlüssel, den man als Parameter zu jedem REST-Aufruf hinzufügen muss. Der Applikationsschlüssel wird in den folgenden Absätzen als APIKEY bezeichnet. Um sich über den aktuellen Wetterbericht eines Ortes zu erkundigen, lassen sich folgende REST-Aufrufe durchführen:

```
http://api.openweathermap.org/data/2.5/weather?
q=Madrid,ES&appId={APIKEY}

http://api.openweathermap.org/data/2.5/weather?
lon=-3.7&lat=40.42&appId={APIKEY}

http://api.openweathermap.org/data/2.5/weather?
id=3117735&appId={APIKEY}
```

Die Aufrufe fragen den Wetterbericht der Stadt Madrid ab. Der erste Aufruf wird dem Stadt- und Landes-Code gemäß ISO 3166 gegeben [3]. Der zweite Aufruf leitet den Längen- und Breitengrad der Hauptstadt Spaniens als Parameter weiter, wohingegen der dritte auf die gewünschte Stadt mit der eigenen OpenWeatherMap-Identifikationsnummer hinweist. Die Liste verfügbarer Städte der OpenWeatherMap-API kann einer JSON-Datei entnommen werden [4].

Das folgende Beispiel zeigt eine JSON-Antwort für die genannten Aufrufe:

```
{
  "coord":{"lon":-3.7,"lat":40.42},
  "weather":
    [{"id":800,"main":"Clear",
      "description":“clear sky",
      "icon":"01d"}],
  "base":"cmc stations",
  "main":
    {"temp":12.47,"pressure":1017,
     "humidity":50,
     "temp_min":12,"temp_max":13},
  "wind":{"speed":1.5,"deg":170},
  "clouds":{"all":0},
  "dt":1458214503,
  "sys":
    {"type":1,"id":5488,"message":0.0039,
     "country":"ES","sunrise":1458195709,
     "sunset":1458239072},
  "id":6544314,
  "name":"Madrid",
  "cod":200
}
```

Die Antwort enthält verschiedene Wetterinformationen bezüglich Temperaturen, Feuchtigkeit, Druck, Wind und Wolken. Eine besondere Rolle spielt die JSON-Eigenschaft „cod“, deren Ausprägung 200 einen erfolgreichen Aufruf bedeutet. Ausprägungen größer als 200 weisen auf einen Fehler hin, z. B. 204 für einen unbekannten Ort. Zudem erhält man eine textuelle Beschreibung des Wetterzustands. Um die Sprache und das Einheitensystem zu bestimmen, kann man zusätzliche Parameter der REST-Abfrage hinzufügen, z. B. für Deutsch und das metrische System:

```
http://api.openweathermap.org/data/2.5/weather?
lon=-3.7&lat=40.42&
   units=metric&lang=de&appId={APIKEY}
```

Obwohl das vorausgesetzte Antwortformat JSON ist, lassen sich aber auch XML-Daten anfordern, indem man den zusätzlichen Parameter *mode=xml* mit der Anfrage verkettet. Um die OpenWeatherMap-API zu testen, kann man die Aufrufe direkt in einem Web-Browser durchführen. Wenn jedoch mehr Einstellungsmöglichkeiten für die Anfrage benötigt werden, dann wird man auf die Verwendung von curl [5] hingewiesen. Tritt dieser Fall ein, muss die gefragte URI in Anführungszeichen gesetzt werden, damit der parametertrennende Charakter „&“ richtig interpretiert wird.

In die Java-REST API eintauchen

Möchte man nun eine REST-Client Java-Klasse schreiben, die den Wetterbericht einer Stadt von OpenWeatherMap anfordert, kann der Code folgendermaßen lauten:

```
public class WetterberichtInfoProvider {
  private static final String  ENDPOINT_URL =
    "http://api.openweathermap.org/data/2.5/weather?";

  public String getWetterberichtFuerStadtId(
    String cityId, String units, String language
  ) {
    ClientBuilder builder=ClientBuilder.newBuilder();
    Client  restClient=builder.build();
    String  resourceURI = buildURI(
      ENDPOINT_URL, cityId, units, language
    );
    WebTarget webTarget = restClient.target(resourceURI);
    String response = webTarget
      .request(MediaType.APPLICATION_JSON)
      .get(String.class);
    return response;
  }
}
```

Der REST-Client basiert auf der JAX-RS-API, wobei die Klassen *Client*, *ClientBuilder* und *WebTarget* des Packages *javax.ws.rs.client* verwendet werden. Der Einstiegspunkt stellt die *ClientBuilder*-Klasse dar, die einen Client instanziiert:

```
ClientBuilder builder = ClientBuilder.newBuilder();
Client restClient = builder.build();
```

Bei dem *restClient* handelt es sich um ein schweres Objekt, das die Kommunikationsinfrastruktur verwaltet und den Zugriff auf eine WebRessource verwirklicht. WebRessourcen werden in dem JAX-RS Client API als „targets“ bezeichnet. Jede Instanz eines *WebTarget* ist einer bestimmten URI zugeordnet:

```
WebTarget webTarget = restClient.target(resourceURI);
```

Ein instanziiertes *WebTarget* ermöglicht die Verkettung von Methoden, um einen Aufruf mit bestimmten Eigenschaften aufzubauen und durchzuführen. In der folgenden Anweisung wird also mit der Methode get(*String.class*) der Aufruf als HTTP-GET ausgeführt und die Antwort des Java-Typs als *String* gesetzt. Die Methode *request(MediaType.APPLICATION_JSON)* stellt JSON als gewünschtes Datenformat ein:

```
String response = webTarget
  .request(MediaType.APPLICATION_JSON)
  .get(String.class);
```

Die Klasse lässt sich in andere Klassen als Stateless-EJB oder als *CDI*-Bean mit der zum Paket *javax.enterprise.context* gehörenden Annotation *@Dependent* injizieren. Da der *restClient* eine teure Ressource ist, lohnt es sich aber auch, ihn als Instanz-Variable zu setzen und dessen Initialisierung auf die *Constructor*- bzw. *Postconstruct*-Methode auszulagern. Folglich sollte die EJB zu *Stateful* oder *Singleton* werden. Wenn CDI benutzt wird, dann ist der Scope der Bean zu *ApplicationScoped* zu erweitern.

Das Paket *javax.ws.rs.client* enthält auch die Klasse Invocation, die eine vorbereitete und ausführbare Anfrage enthält. Sie bietet eine generische Schnittstelle, die die „Separation of concerns“ zwischen Aufbau und Ausführung der Anfrage fördert. Der Code vom letzten Absatz kann wie folgt zu einer Invocation umgeschrieben werden:

```
Invocation invocation = webTarget
  .request(MediaType.APPLICATION_JSON)
  .buildGet();
```

Eine Aufrufer-Klasse lässt die Anfrage mit folgender Anweisung synchron ausführen:

```
String response = invocation.invoke(String.class);
```

Um eine asynchrone Anfrage zu schicken, nutzt man die Methode submit:

```
Future<String> response =
    invocation.submit(String.class);
```

Die Antwort gehört zur Schnittstelle Future (Paket *java.util.concurrent*), die die Methoden *isDone()* und *get()* bereitstellt, um zu prüfen, ob die Antwort schon verfügbar ist bzw. um die Antwort zu lesen.

Moment, Java ist eine typisierte Sprache!

Der erste REST-Client liefert einen *String* zurück. Eine Verarbeitung des Strings, um den Inhalt zu parsen, ist möglich, aber umständlich und nicht besonders elegant. Da Java eine typisierte Sprache ist, ist es vorteilhafter, eine POJO-Klasse für einen Wetterbericht zu definieren, sodass für einen REST-Aufruf eine Instanz von ihr als Antwort erzeugt wird. Für das folgende Beispiel nimmt man nun an, dass eine POJO-Klasse Wetterbericht durch die Instanz-Variablen Temperatur, Druck, Feuchtigkeit, Windgeschwindigkeit, Regenmenge und Beschreibung vom Java-Typ *String* bestimmt ist.

Um eine JSON-Antwort in einen Java-Typ umzuwandeln, bietet JAX-RS die zum Paket *javax.ws.rs.ext* gehörende Schnittstelle *MessageBodyReader<T>*, die folgende Methoden definiert:

```
public boolean isReadable(
    Class<?> type,
    Type genericType,
    Annotation[ ] annotations,
    MediaType mediaType
)

public T readFrom(
    Class<T> type,
    Type genericType,
    Annotation[ ] annotations,
    MediaType mediaType,
    MultivaluedMap<String, String> httpHeaders,
    InputStream is
) throws IOException, WebApplicationException
```

Die erste Methode sagt mittels einer Boole'schen Variable aus, ob eine Instanz vom Typ *T* erzeugt werden kann oder nicht. Die Methode *readFrom()* enthält die tatsächliche JSON-Parser-Logik.

Java kommt mit JSON gut klar

Die Java-API für die JSON-Verarbeitung (Paket *javax.json*) bietet zwei Vorgehensweisen: Die erste Variante liest JSON aus einem Strom und erzeugt ein Modell im Speicherplatz. Die zweite Variante stellt einen eventbasierten Parser dar. Im genannten Beispiel wird die erste Variante für den Client mit den Klassen *JsonReader*, *JsonObjekt* und *JsonArray* genutzt.

Ein *JsonReader* wird aus einem *InputStream* is durch die *Factory*-Klasse *Json* mit folgender Anweisung erzeugt:

```
JsonReader reader = Json.createReader(is);
```

Aus einem instanziierten *JsonReader* lässt sich mithilfe der Methode *readObject()* eine Instanz eines *JsonObjekt* erzeugen. Ähnliches gilt für *JsonArray* und die Methode *readArray()*. Um zu prüfen, ob ein *JsonObject* einen bestimmten Schlüssel enthält, verfügt *JsonObject* über die Methode *containsKey(Object o)*. Mit dem Aufrufen von *getString(String str)* erfährt man dessen Wert. Ein *String* kann aus einer bestimmten Position eines *JsonArray* mit der Methode *getString(int index)* gelesen werden. Die Methode *getJsonObject(int index)* liefert das *JsonObjekt*, das sich an der Position *index* des *JsonArray* befindet, zurück. Mit diesen Methoden sind wir schon in der Lage, eine Implementierung für *MessageBodyReader* zu schreiben:

```
@Provider
@Consumes(MediaType.APPLICATION_JSON)
public class WetterberichtMessageBodyReader
  implements MessageBodyReader<Wetterbericht>
{

  @Override
  public boolean isReadable(Class<?> type,
    Type genericType, Annotation[ ] annotations,
    MediaType mediaType
  ) {
    return true;
  }

  @Override
  public Wetterbericht readFrom(
    Class<T> type,
    Type genericType, Annotation[ ] annotations,
    MediaType mediaType,
    MultivaluedMap<String, String> httpHeaders,
    InputStream is
  ) throws IOException, WebApplicationException {
    Wetterbericht wetterbericht = new Wetterbericht();
    JsonReader reader = Json.createReader(is);
    JsonObject jsonObject = reader.readObject();

    JsonArray weather = jsonObject.getJsonArray("weather");
    JsonObject weatherSub = weather.getJsonObject(0);
    wetterbericht.setBeschreibung(
      weatherSub.getString("description")
    );
```

```
        JsonObject main = jsonObject.getJsonObject("main");
        wetterbericht.setTemperatur(main.getString("temp"));
        wetterbericht.setDruck(main.getString("pressure"));
        wetterbericht.setFeuchtigkeit(main.getString("humidity"));

        JsonObject wind=jsonObject.getJsonObject("wind");
        wetterbericht.setWindgeschwindigkeit(
          wind.getString("speed")
        );

          if (jsonObject.containsKey("rain")) {
          JsonObject rain = jsonObject.getJsonObject("rain");
          wetterbericht.setRegenmenge(rain.getString("3h"));
        }

        reader.close();
        return wetterbericht;
    }
}
```

Ein wichtiger Punkt ist die Annotation *@Provider* des Pakets *javax.ws.rs.ext*. Die JAX-RS Runtime-Umgebung sucht nach dieser Annotation, um Provider-Klassen zu entdecken. Die Annotation *@Consumes* verdeutlicht, in welches Format konvertiert wird. Die Annotation *@Produces* steht auch mit umgekehrter Bedeutung zur Verfügung.

Die beiden Teile verbinden

Um von der *WetterberichtMessageBodyReader*-Klasse profitieren zu können, sind noch einige Anpassungen an der Klasse *WetterberichtInfoProvider* vorzunehmen. Zuerst muss die *WetterberichtMessageBodyReader*-Klasse beim Erzeugen des *ClientBuilder* registriert werden:

```
ClientBuilder builder = ClientBuilder.newBuilder();
builder.register(WetterberichtMessageBodyReader .class);
restClient = builder.build();
```

Als nächstes wird die Methode *getWetterberichtFuerStadtId* umgeschrieben:

```
WebTarget webTarget = restClient.target(resourceURI);
GenericType<Wetterbericht> response =
  new GenericType<Wetterbericht>(){};
Wetterbericht wetterbericht = webTarget.
  .request(MediaType.APPLICATION_JSON)
  . get(response);
return wetterbericht;
```

Eine Instanz von *GenericType<Wetterbericht>* wird der Methode get von *webTarget* gegeben. Dadurch merkt sich die JAX-RS Runtime-Umgebung, dass eine entsprechende Implementierung der *MessageBody-Reader* Schnittstelle einzusetzen ist. Als Ergebnis der JSON-Anfrage gibt die Methode *getWetterbericht FuerStadtId* eine Instanz von *Wetterbericht* zurück.

Fazit

Die Architektur moderner Anwendungen beschränkt sich nicht nur auf die Mehrschicht- bzw. SOAP-Web-Services-Paradigmen. Andere wie z. B. eventbasierte Architekturen oder vor allem Microservices sind derzeit im Trend und gewinnen zunehmend an Aufmerksamkeit. Einer der grundlegenden Bausteine dafür sind REST-Services, die sich mit Java EE 7 und deren APIs für JSON-Verarbeitung und JAX-RS gut realisieren lassen. Die neue Version von Java EE kündigt für diese APIs Erweiterungen an [6], wie z. B. eine JSON-Binding-API, die ähnlich wie JAXB [7] funktionieren sollte. Damit wäre Java EE 8 für die neuen, architekturalen Herausforderungen bestens vorbereitet.

Referenzen

[1] Programmableweb *Browse the world's largest API repository*, http://www.programmableweb.com/category/all/apis

[2] OpenWeatherMap *Weather API*, http://openweathermap.org/api

[3] ISO *Country Codes - ISO 3166*, http://www.iso.org/iso/country_codes

[4] OpenWeatherMap *Liste Von Städten Der OpenWeatherMap Api*, http://bulk.openweathermap.org/sample/

[5] Curl *curl and libcurl*, https://curl.haxx.se

[6] Gupta, Arun *Java EE 8 Status*, http://blog.arungupta.me/javaee8

[7] Salces Sillero, Ángel *Der Umzug von XML in die Java-Welt – JAXB im Einsatz*, KaffeeKlatsch, Jahrgang 8, Nummer 4, Seiten 15 – 18, Bookware, April 2015, http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2015-04.pdf

Weiterführende Literatur

- Oracle Technology Network *Package javax.ws.rs.client for Java Platform EE 7*, https://docs.oracle.com/javaee/7/api/javax/ws/rs/client/package-summary.html
- Oracle Technology Network *Package javax.ws.rs.core for Java Platform EE 7*, https://docs.oracle.com/javaee/7/api/javax/ws/rs/core/package-summary.html
- Java.Net The Source For Java Technology Collaboration *Java API for RESTful Services (JAX-RS)*, https://jax-rs-spec.java.net
- Oracle Technology Network *Package javax.json for Java Platform EE 7*, http://docs.oracle.com/javaee/7/api/javax/json/package-summary.html

Kurzbiografie

Ángel Salces ist als Entwickler, Berater und Trainer für MATHEMA Software GmbH tätig. Sein Spezialgebiet ist die Programmierung von Anwendungen mit Java/JEE und die Design und Implementierung von Datenbanken. Daneben beschäftigt er sich mit den Neuerungen der Java-Welt und den neuen Trends der Datenbanken, insbesondere Big Data. Er setzt gerne die Philosophie des Software-Craftmanships und des Clean-Codes ein.

Des Programmierers kleine Vergnügen

Königsdisziplin

von Michael Wiedeking

Seit fast einhundert vergnüglichen Kolumnen wurde die schwierigste aller Disziplinen geschickt umschifft: das Dividieren. Hier nun endlich – allerdings der Division gebührend in mehreren Teilen – die Lösung, wie man beliebig große Zahlen teilt.

Wie schon in der Vergangenheit soll eine vorzeichenlose, nicht negative Zahl n mit m Ziffern

$$u = u[m-1] \quad u[m-2] \quad \ldots \quad u[1] \quad u[0]$$

durch die vorzeichenlose, positive Zahl v mit n Ziffern

$$v = v[n-1] \quad v[n-2] \quad \ldots \quad v[1] \quad v[0]$$

derart geteilt werden, dass sowohl der Quotient u/v als auch auf Wunsch der ganzzahlige Rest dieser Division geliefert wird. Dabei wird festgelegt, dass $0 < n \leq m$ und

$v[n-1] \neq 0$ ist.

Die einzelnen Ziffern einer Zahl z, die hier *little-endian* angeordnet sind – womit $z[0]$ die niederwertigste Ziffer enthält –, sind allgemein Ziffern zu einer beliebigen Basis B. In unser Implementierung wird es aber eine Zweierpotenz sein, die so gewählt wird, dass eine Division 2^{2k} Bit / 2^k Bit zur Verfügung steht. In Java oder C# kann man sich also eines 32-Bit *ints* bedienen, womit $B = 2^{32}$ gilt und die Division eines 64-Bit *longs* zur Verfügung steht.

Um nun ganz harmlos anzufangen, soll die Zahl u zunächst nur durch eine einzige Ziffer geteilt werden. Das können wir genau so machen, wie wir es in der Schule gelernt haben, um beispielsweise $u = 793$ durch $v = 6$ zu teilen.

763 ÷ 6 =

Wie ging das noch gleich? Zunächst versucht man die 6 durch die erste Ziffer zu teilen. Das klappt erfreulicherweise.

763 ÷ 6 = **1**

Nun multipliziert man dieses Ergebnis mit der Zahl, mit der man geteilt hat und zieht dies von der ersten Ziffer ab.

```
  763 ÷ 6 = 1
– 6     (= 1 × 6)
-------
  1
```

Da das nicht ganz passt, haben wir nun einen Rest, der bei der nächsten Iteration berücksichtigt werden muss. Dazu holt man nun die nächste Ziffer „herunter“ und wiederholt den ersten Schritt. Dieses Herunterholen ist eigentlich eine Addition, wobei man den Rest der letzten Multiplikation mit B (hier ist $B = 10$) multipliziert und die nächste Ziffer dazu addiert.

Nun gilt es also 16 durch 6 zu teilen, was 2 ergibt. $2 \times 6 = 12$ und damit ergibt sich ein Rest von $16 - 12 = 4$.

```
  763 ÷ 6 = 12
– 6
-------
  16 = 1 × 10 + 6
– 12      (= 2 × 6)
-------
   4
```

Schließlich muss noch $4 \times 10 + 3$ durch 6 geteilt werden, was 7 mal geht und ein Rest von 1 verbleibt.

```
  763 ÷ 6 = 127 mit Rest 1
– 6
-------
  16
– 12
-------
   43 = 1 × 10 + 3
 – 42    (= 7 × 6 )
-------
   1
```

Wandelt man das nun in einen passenden Algorithmus um, so müssen wir beginnend mit der höchstwertigsten Ziffer durch v teilen. Dann multiplizieren wir das Ergebnis mit v, um den Rest r zu bestimmen. Dieser Rest besteht aus genau einer Ziffer (wie man sich leicht klar macht, da ja ein Rest mit zwei Ziffern auf jeden Fall noch durch eine Ziffer teilbar wäre). Durch Multiplikation mit B wird dieser Rest um eine Stelle verschoben, wodurch sich die nächst niederwertigere Ziffer (überschneidungsfrei) ergänzen lässt.

Die folgende Funktion soll also die Zahl u mit den m Ziffern mit den Indizes $m - 1$ bis 0 durch die Zahl v mit genau einer Ziffer teilen und den Quotienten in q und den Rest dieser Division in r ablegen. q und r müssen natürlich ausreichend Platz bieten: q muss also mindestens Platz für m Ziffern bieten und r Platz für mindestens eine Ziffer. Der Code in einem *C*-artigen Dialekt könnte in etwa so aussehen:

```
void divide(int[] u, int m, int v, int[] q, int[] r) {
  long k = 0;
  for (int i = m - 1; m ≥ 0; m -= 1) {
    long p = k × B + u[i];          // (1)
    long d = p / v;                 // (2)
    q[i] = d;                       // (3)
    k = p - d × v;                  // (4)
  }
  r[0] = k;
}
```

Wie bereits erwähnt funktioniert (1) problemlos, da man es hier mit maximal zwei Ziffern zu tun hat und die Multiplikation mit B sicher stellt, dass bei der Addition keine Überträge entstehen können.

Das Ergebnis der Division in (2) ist auf jeden Fall eine einzige Ziffer (auch wenn hier mit einer Variablen mit doppelter Breite gearbeitet wird). Das macht man sich auch leicht im Fall $B = 10$ klar. Der Übertrag kann da nämlich höchstens 8 (also $B - 2$) sein, wenn man durch die größtmögliche Ziffer 9 (entspricht $B - 1$) teilt. Die größtmögliche nächste Ziffer ist ebenfalls 9 (wieder $B - 1$), weswegen man maximal durch 89 teilen muss, was aber als Ergebnis maximal 9 (also $B - 1$) ist.

Teilt man durch eine kleinere Zahl, so ist der Fall mit der Division durch 1 trivial, weil der Rest immer 0 ist. Nimmt man also als kleinste Zahl die 2, so liefert die höchstens einen Rest von 1, da der maximale Rest immer um eins kleiner ist als der Divisor. Damit ist die größtmögliche Zahl in der nächsten Iteration 19, was aber geteilt durch 2 auch ein Ergebnis liefert, dass nur eine Ziffer hat.

Damit passt der Quotient d in (3) auch wirklich an die angegebene Stelle und der verbleibende Rest in (4) ist auf jeden Fall kleiner als $B - 1$.

Hierbei darf man allerdings nicht vergessen, dass alle Operationen auf Ziffern basieren, die vorzeichenlos sind. Unter C# lässt sich das Problem lösen, indem alle Daten als *unsigned* markiert werden. In Java kann das dementsprechend nicht ganz so einfach funktionieren. Hier müsste der Code wie folgt aussehen:

```
long k = 0;
for (int i = m - 1; m ≥ 0; m -= 1) {
  long p = (k << 32) + (u[i] & 0xFFFF_FFFFL);   // (1)
  long d = Long.divideUnsigned(p, v);           // (2)
  q[i] = (int) d;                               // (3)
  k = p - d * v;                                // (4)
}
r[0] = (int) k;
```

Zunächst muss in (1) damit gerechnet werden, dass bei den großen Ziffern das höchstwertigste Bit gesetzt sein kann. Damit diese Ziffern nicht als negativ interpretiert werden, muss explizit sicher gestellt werden, dass beim Übertragen der Ziffer in die Variable, die Platz für zwei Ziffern hat, nicht das höchstwertigste Bit durch *Sign-Extension* auf die zweite Ziffer ausgebreitet wird (weswegen peinlich darauf zu achten ist, dass das L bei der Maske nicht vergessen wird).

Während nämlich die Addition und Subtraktion immer das auch bei vorzeichenloser Interpretation korrekte Bit-Muster liefern, ist das bei der Multiplikation und Division meistens nicht der Fall. Die vorzeichenbehaftete Multiplikation in (4) ist nur deshalb korrekt, weil hier zwei einzelne (nicht negative) Ziffern multipliziert werden. Das Ergebnis ist dann eine Zahl mit zwei Ziffern, dessen Bit-Muster (vorzeichenlos) korrekt ist, obwohl sie bei vorzeichenbehafteter Interpretation, wenn das höchstwertigste Bit gesetzt ist, wegen dieses Überlaufs ein falsches Ergebnis liefern würde.

Bei der Division in (2) kann leider nicht garantiert werden, dass die betroffenen Ziffern nie das höchstwertigste Bit gesetzt haben, also nicht doch negativ sind. Deshalb muss hier auf eine vorzeichenlose Division zugegriffen werden. Ein „Geteiltes Vergnügen“ [1] wurde aber schon vorgestellt, in dem die vorzeichenlose Division effizient auf Basis der vorzeichenbehafteten implementiert wurde. Doch wie immer gilt es hier, die Standardfunktion ungeachtet ihrer Implementierung zu verwenden, damit der Just-in-Time-Compiler diese eventuell in einen einzigen, unschlagbaren Maschinenbefehl umwandeln kann.

Damit wäre geklärt, wie die Division grundsätzlich ablaufen könnte. Leider ist dies, wenn v aus mehreren Wörtern besteht, nicht mehr ganz so einfach. Dann muss man nämlich tatsächlich „raten“, wie man am besten teilt. Das ist jedoch eine Geschichte, die dann wohl bis zum nächsten Vergnügen warten muss.

Referenten

[1] Wiedeking, M. *Des Programmierers kleine Vergnügen – Geteiltes Vergnügen,* KaffeeKlatsch, Jahrgang 7, Nr. 2, S. 12f, Bookware, Februar 2014 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2014-02.pdf

Kurzbiographie

Michael Wiedeking (michael.wiedeking@mathema.de) ist Gründer und Geschäftsführer der MATHEMA Software GmbH, die sich von Anfang an mit Objekttechnologien und dem professionellen Einsatz von Java einen Namen gemacht hat. Er ist Java-Programmierer der ersten Stunde, „sammelt“ Programmiersprachen und beschäftigt sich mit deren Design und Implementierung.

User Groups

Fehlt eine User Group? Sind Kontaktdaten falsch? Dann geben Sie uns doch bitte Bescheid.

Bookware, Henkestraße 91, 91052 Erlangen
Telefon: 0 91 31 / 89 03-0, Telefax: 0 91 31 / 89 03-55
E-Mail: redaktion@bookware.de

Java User Groups

Deutschland

JUG Berlin Brandenburg
http://www.jug-bb.de
Kontakt: Herr Ralph Bergmann (orga@jug-bb.de)

Java UserGroup Bremen
http://www.jugbremen.de
Kontakt: Rabea Gransberger (rgransberger@gmx.de)

JUG DA
Java User Group Darmstadt
http://www.jug-da.de
Kontakt: jug-da-orga@googlegroups.com

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxony.de
Kontakt: Herr Falk Hartmann
(falk.hartmann@jugsaxony.org)

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
http://www.ruhrjug.de
Kontakt: Herr Heiko Sippel (heiko.sippel@ruhrjug.de)

JUGF
Java User Group Frankfurt
http://www.jugf.de
Kontakt: Herr Alexander Culum
(alexander.culum@web.de)

JUG Deutschland e.V.
Java User Group Deutschland e.V.
c/o Stefan Koospal
http://www.java.de (office@java.de)

JUG Hamburg
Java User Group Hamburg
http://www.jughh.org

JUG Karlsruhe
Java User Group Karlsruhe
http://jug-karlsruhe.de
(jugkarlsruhe@gmail.com)

JUGC
Java User Group Köln
http://www.jugcologne.org
Kontakt: Herr Michael Hüttermann
(michael@huettermann.net)

jugm
Java User Group München
http://www.jugm.de
Kontakt: Herr Andreas Haug (ah@jugm.de)

JUG Münster
Java User Group für Münster und das Münsterland
http://www.jug-muenster.de
Kontakt: Herr Thomas Kruse (tkjugi@sforce.org)

JUG MeNue
Java User Group der Metropolregion Nürnberg
c/o MATHEMA Software GmbH
Henkestraße 91, 91052 Erlangen
http://www.jug-n.de
Kontakt: Frau Natalia Wilhelm
(info@jug-n.de)

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
http://www.jug-ostfalen.de
Kontakt: Uwe Sauerbrei (info@jug-ostfalen.de)

JUGS e.V.
Java User Group Stuttgart e.V.
c/o Dr. Michael Paus
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)
Rainer Anglett (ra@jugs.org)

Schweiz

JUGS
Java User Group Switzerland
http://www.jugs.ch (info@jugs.ch)

.NET User Groups

Deutschland

.NET User Group Bonn
.NET User Group "Bonn-to-Code.Net"
http://www.bonn-to-code.net (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

.NET User Group Dortmund (Do.NET)
c/o BROCKHAUS AG
http://do-dotnet.de
Kontakt: Paul Mizel (pmizel@do-dotnet.de)

Die Dodnedder
.NET User Group Franken
http://www.dodnedder.de
Kontakt: Herr Udo Neßhöver, Frau Ulrike Stirnweiß
(info@dodnedder.de)

.NET UserGroup Frankfurt
http://www.dotnet-usergroup.de

.NET User Group Friedrichshafen
http://www.dotnet-fn.de
Kontakt: Tobias Allweier
(info@dotnet-fn.de)

.NET User Group Hannover
http://www.dnug-hannover.de
Kontakt: (dnug@indisoftware.de)

INdotNET
Ingolstädter .NET Developers Group
http://www.indot.net
Kontakt: Herr Gregor Biswanger
(gregor.biswanger@web-enliven.de)

DNUG-Köln
DotNetUserGroup Köln
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.NET User Group Leipzig
http://www.dotnet-leipzig.de
Kontakt: Herr Alexander Groß (agross@dotnet-leipzig.de)
Herr Torsten Weber (tweber@dotnet-leipzig.de)

.NET Developers Group München
http://www.munichdot.net
Kontakt: Hardy Erlinger (hardy_erlinger@hotmail.com)

.NET User Group Oldenburg
c/o Hilmar Bunjes und Yvette Teiken
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Frau Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET Developers Group Stuttgart
http://www.devgroup-stuttgart.net
(GroupLeader@devgroup-stuttgart.net)
Kontakt: Herr Michael Niethammer

.NET Developer-Group Ulm
c/o artiso solutions GmbH
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

Österreich

.NET User Group Austria
c/o Global Knowledge Network GmbH,
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

Software Craftsmanship Communities

Deutschland, Schweiz, Österreich

Softwerkskammer – Mehrere regionale Gruppen und Themengruppen unter einem Dach
http://www.softwerkskammer.org
Kontakt: Nicole Rauch (nicole.m@gmx.de)

MATHEMA

meet the experts of enterprise infrastructure

- **Generation 8 – Java Update**
  6. – 7. Juni 2016, 21. – 22. November 2016
  950,– € (zzgl. 19 % MwSt.)
- **AngularJS**
  13. – 14. Juni 2016, 28. – 29. November 2016
  950,– € (zzgl. 19 % MwSt.)
- **HTML5, CSS3 und JavaScript**
  27. – 30. Juni 2016, 1.650,– € (zzgl. 19 % MwSt.)
- **Einführung in die objektorientierte Programmiersprache Java**
  11. – 15. Juli 2016, 2.150,– € (zzgl. 19 % MwSt.)
- **Entwicklung mobiler Anwendungen mit iOS**
  18. – 20. Juli 2016, 1.250,– € (zzgl. 19 % MwSt.)
- **Einführung in C# und .NET**
  26. – 30. Sept. 2016, 2.150,– € (zzgl. 19 % MwSt.)
- **Java FX8**
  14. – 16. November 2016, 1.250,– € (zzgl. 19 % MwSt.)

Lesen bildet. Training macht fit.

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de

MATHEMA

„Ich bin bei MATHEMA, weil hier gelebt wird, dass Software auch etwas mit Softskills zu tun hat."

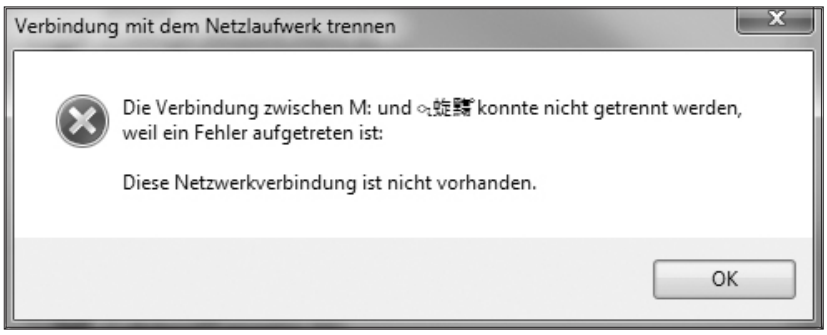
Thomas Bertz, Senior Consultant

Wir sind ein Consulting-Unternehmen mit Schwerpunkt in der Entwicklung unternehmenskritischer, verteilter Systeme und Umsetzung von Service-orientierten Architekturen und Applikationen von Frontend bis Backend. Darüber hinaus ist uns der Wissenstransfer ein großes Anliegen: Wir verfügen über einen eigenen Trainingsbereich und unsere Consultants sind regelmäßig als Autoren in der Fachpresse sowie als Speaker auf zahlreichen Fachkonferenzen präsent.

join the experts of enterprise infrastructure

MATHEMA Software GmbH | Telefon: 09131 / 89 03-0 | Internet: www.mathema.de | Ansprechpartnerin:
Henkestraße 91, 91052 Erlangen | Telefax: 09131 / 89 03-55 | E-Mail: info@mathema.de | Britta Kolb

Das Allerletzte



Dies ist kein Scherz!
Diese Meldung wurde tatsächlich in der freien Wildbahn angetroffen.

Ist Ihnen auch schon einmal ein Exemplar dieser Gattung über den Weg gelaufen?
Dann scheuen Sie sich bitte nicht, uns das mitzuteilen.

Der nächste KaffeeKlatsch erscheint im Juni.

Herbstcampus

Wissenstransfer par excellence

30. August – 1. September 2016
in Nürnberg