KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

01/2014

Jahrgang 7

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

Verantwortungsbewusst

Die Snowden-„Affäre“ lässt mich nicht los. Dabei geht es mir nicht um Snowden selbst und die damit verbundenen Dinge – wenngleich ich seine Entscheidung, die geheimen Dokumente zu veröffentlichen, sehr mutig und bewundernswert finde. Bei der Gelegenheit sollten auch die Redakteure der Washington Post und des Guardian nicht unerwähnt bleiben, die ebenfalls ein nicht zu unterschätzendes Risiko eingegangen sind, auch wenn diese, im Gegensatz zu Snowden, unter dem besonderen Schutz der Presse stehen. Mir geht es darum, welche Bedeutung die offengelegte Überwachung für uns Menschen hat.

Unser Grundgesetz ist wenige Jahre nach dem zweiten Weltkrieg im Mai 1949 „im Bewußtsein seiner Verantwortung vor Gott und den Menschen, von dem Willen beseelt, als gleichberechtigtes Glied in einem vereinten Europa dem Frieden der Welt zu dienen“ verabschiedet worden und beginnt mit dem Satz „Die Würde des Menschen ist unantastbar. Sie zu achten und zu schützen ist Verpflichtung aller staatlichen Gewalt.“ Auch wenn hier der Begriff Würde nicht genauer definiert ist, wird hier doch viel darüber gesagt, wie mit dem Menschen umzugehen ist. Und darauf aufbauend ergeben sich eine Reihe von Rechten, die allen Menschen eingeräumt werden, wie etwa das Recht auf freie Entfaltung der Persönlichkeit, körperliche Unversehrtheit oder Religionsfreiheit. Insbesondere sagt Artikel 5: „Jeder hat das Recht, seine Meinung in Wort, Schrift und Bild frei zu äußern und zu verbreiten und sich aus allgemein zugänglichen Quellen ungehindert zu unterrichten.“

Die Charta der Vereinten Nationen spricht davon, dass die Völker der Vereinten Nationen den „Glauben an die Grundrechte des Menschen, an Würde und Wert der menschlichen Persönlichkeit, an die Gleichberechtigung von Mann und Frau sowie von allen Nationen, ob groß oder klein, erneut“ bekräftigen und „den sozialen Fortschritt und einen besseren Lebensstandard in größerer Freiheit“ fördern. Und in Artikel 1 versprechen die Vereinten Nationen „die Achtung vor den Menschenrechten und Grundfreiheiten für alle ohne Unterschied der Rasse, des Geschlechts, der Sprache oder der Religion zu fördern und zu festigen“.

In vielen Ländern wird darüber hinaus noch wie bei uns geregelt, dass eine Überwachung nur bei Verdacht – und das auch nur mit richterlicher Zustimmung – zulässig ist.

Jetzt stammen diese Gesetze alle aus Zeiten, wo der Großteil der Kommunikation nur auf dem Postwege oder via Telefon möglich war. Damals war es also auch aus technischer Sicht nicht so einfach möglich, beliebig viele Menschen zu überwachen. Die Zeiten haben sich geändert, was man ja schon allein daran merkt, dass es praktisch nichts kostet, eine E-Mail an Millionen von Menschen zu verschicken. Aber rechtfertigt dies, dass die Grundrechte außer Kraft gesetzt werden?

So argumentiert der US-amerikanische Geheimdienst etwa bei der Erfassung von den sogenannten „Metadaten“, dass diese nicht durch die Verfassung geschützt wären. Ein Gespräch darf also laut Verfassung nicht abgehört werden, aber es spricht nichts gegen eine ewige Speicherung der Verbindungsdaten, wer mit wem, wie lange und von wo aus telefoniert hat? Kann ich mich tatsächlich noch frei informieren, wenn jede meiner Suchabfragen für immer und ewig gespeichert wird?

Politiker und Patrioten mögen darüber entscheiden, was für ein Land gut ist, und sich dabei vielleicht das eine oder andere Gesetz zurecht biegen oder eine offensichtliche Gesetzeslücke ausnutzen, aber es sind die Techniker und Programmierer, die diese Vorhaben umsetzen. Aber dabei dürfen wir nicht vergessen, dass nicht alles was machbar ist, auch gut ist.

Wir haben übrigens aus unserer Geschichte gelernt. Deshalb können sowohl Setzer als auch Drucker – beides im ursprünglichen Sinne leider aussterbende Berufe – für das von ihnen erstellte Druckwerk mit zur Verantwortung gezogen werden. Man darf sich also nicht wundern, wenn dies auch mal für Programme der Fall sein könnte.

Vergessen wir also nicht unsere moralische Verpflichtung. Wobei wir uns vor Augen halten sollten, dass es sich bei Gesetzen in der Regel nur um das moralische Minimum handelt.

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 09131/8903-16 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 09131/8903-16.

Copyright

Impressum

KaffeeKlatsch Jahrgang 7, Nummer 1, Januar 2014
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

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

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

Inhalt

Artikel

Kolumnen

Das Türchen ins Web

von Róbert Bräutigam

Es gibt zahlreiche *Web-Frameworks* für Java und oft genug entscheiden nicht-funktionale Anforderungen nicht eindeutig welches man nehmen sollte. So kann man sich den Luxus erlauben selbst, anhand von persönlichen (oder Team-) Präferenzen, zu wählen. *Wicket* sollte in diesen Fällen auch ein Mitbewerber sein.

Abra Kadabra?

von Frank Ganske

Immer wieder gibt es diese Aufgaben – wiederkehrende, gleichförmige Software-Komponenten, die langweilig und fehlergefährdet sind, wenn sie manuell erstellt werden. Das soll automatisiert werden, im kleinen Rahmen, ohne den regulären Projekt- und Bauprozess zu stören. Im KaffeeKlatsch vom Juni 2013 wurde unter dem Titel „Simsalabim!" beschrieben, wie das mit *Velocity* und *Ant* gelöst wird. Prompt fand sich der Autor danach in einer Umgebung wieder, in der die eingesetzten Hilfsmittel klar und restriktiv definiert sind. Wie macht man es also mit den Bordmitteln eines Java-EE-Entwicklers? Zusätzlich wird gezeigt, wie aus einem großen XML-Modell Teilbereiche selektiert und so kleine Teilmodelle erstellt werden. Dieser häufige Einsatzfall führt den ersten Artikel weiter.

You've been PhoneGapped

von Tim Bourguignon

Man hat manchmal komische Ideen, wie zum Beispiel eine mobile Applikation in einer neuen Technologie zu schreiben, nur um danach festzustellen: „Tja, ging doch"! In diesem Fall handelt es sich um eine mobile Applikation mit *PhoneGap*, und sie funktioniert noch nicht. Dies ist mein Erfahrungsbericht.

Das Türchen ins Web

Web-Entwicklung mit Wicket

von Róbert Bräutigam

Es gibt zahlreiche *Web-Frameworks* für Java und oft genug entscheiden nicht-funktionale Anforderungen nicht eindeutig welches man nehmen sollte. So kann man sich den Luxus erlauben selbst, anhand von persönlichen (oder Team-) Präferenzen, zu wählen. *Wicket* sollte in diesen Fällen auch ein Mitbewerber sein.

Dieser Artikel ist keine Anleitung um Wicket-Applikationen zu schreiben, dafür gibt es Projekte und Dokumentationen im Internet [1] [2] [3]. Vielmehr helfen die folgenden Kapitel festzustellen, ob Wicket für Sie das richtige Framework ist, indem Unterschiede zu anderen Frameworks aufgezeigt werden.

Wie es aussieht

Für eine einfache Seite braucht man (nach Erzeugen eines leeren Projekts mit Wicket konfiguriert) nur 3 Dateien anzulegen. Das *Application*-Objekt konfiguriert die Applikation und legt z. B. die Startseite fest:

```
public class Application extends WebApplication {
  @Override
  public Class<? extends Page> getHomePage() {
    return HomePage.class;
  }
}
```

Dann muss man die *HomePage*-Seite implementieren:

```
public class HomePage extends BasePage {
  public HomePage() {
    add(new Label(„greeting“, „Hello, World!“));
  }
}
```

Um die programmatisch erzeugte Seitenstruktur anzeigen zu können, schreibt man den *View* als *HTML*-Datei *HomePage.html* und legt diese gleich neben die Java-Datei:

```
<html>
  <head><title>Wicket Test</title></head>
  <body>
    <h1 wicket:id=“greeting“>Das wird weggeworfen!</h1>
  </body>
</html>
```

Die Verbindung zwischen den Strukturen im Code und HTML beschreibt man mit Hilfe von *wicket:id-Attributen*.

Reife

Das Projekt selbst ist fast zehn Jahre alt, und seit fast neun Jahren veröffentlicht. Mitte 2007 ist es ein Apache-Projekt geworden und wird seitdem auch aktiv entwickelt.

Die Mailing-Liste ist auch aktiv [4] und reagiert schnell auf Anfragen.

Es gibt nicht nur kleine sondern auch große Web-Projekte mit Wicket, wie z. B. die Homepage des Springer Verlags [5].

Leichtgewichtigkeit

Wenn man in einer IDE mit einem Klick fast alles machen kann, inklusive Web-Projekte zu erstellen, vergisst man oft wie viel überflüssiger Code (*Cruft* und *Bloat*) sich leicht in einem Projekt einschleichen kann.

Man sollte sich immer bemühen nur Abhängigkeiten zu definieren, die überschaubar sind und die man wirklich braucht. „Leichtgewichtig" heißt nicht unbedingt das etwas „klein" ist, sondern dass es einfach festzustellen ist, was drin ist und was man davon braucht oder nicht braucht. Es heißt auch, dass es keine oder wenige zusätzliche Anforderungen für die Applikation stellt. Beispielsweise fordert es nicht andere Frameworks mitzunutzen, die Auswahl von Servern einzugrenzen usw. Kurz: Es ist ein Baustein, den man mit anderen Bausteinen kombinieren kann und nicht ein komplettes Haus.

Wicket ist leichtgewichtig. Es ist nicht nur klein (ein paar Megabyte in mehreren JARs verteilt), sondern definiert fast keine weiteren Abhängigkeiten, und lässt sich mit fast allem kombinieren, von *EJBs* über *Injection* Frameworks zu *AJAX* und *Bootstrap*.

Werkzeugkasten

Mit Wicket programmiert man in Java und baut HTML für die Web-Ansicht. Da (mutmaßlich) die Business Logik auch in Java ist, macht es Sinn Java für die Web-Schicht zu verwenden. Da diese Web-Schicht am Ende HTML erzeugen muss, scheint es auch sinnvoll dazu HTML zu kennen und es zu benutzen.

Man könnte argumentieren, dass man als Java-Entwickler eigentlich kein HTML schreiben will und alles purer Java-Code sein sollte. Wicket lässt diese Entscheidung offen und man kann auch HTML (typsicher) von Java erzeugen, aber in der Praxis scheint es zu wenig Vorteile zu bringen, für den Mehraufwand HTML in Java zu schreiben.

In Gegensatz dazu fordern andere Web-Frameworks oft, dass man zusätzlich zu Java und HTML andere Sprachen auch benutzen soll, wie *JSTL, XML, XSLT, Velocity Markup, JSP, JSF* usw.

Trennung von Anliegen

Die meisten Web-Frameworks versuchen *Model*-, *View*- und *Controller*-Code von einander abzugrenzen, aber nicht alle schaffen es so wie man es sich wünschen würde.

In Wicket werden (wie oben) View und Controller tatsächlich getrennt. Sie sind nicht nur in einer anderen Datei, sondern der View (also das *HTML File*) beinhaltet auch keine *Bindings* (also Zugriffe) auf Model-Objekte, keine Schleifen oder Bedingungen. Die einzige Verbindung zwischen View und Controller besteht darin, dass die Seitenstruktur in HTML und im Code zusammengebunden wird durch die Wicket-Ids. Dies muss eine eindeutige eins-zu-eins-Verbindung sein, sonst kriegt man (leider zur Laufzeit, also Testzeit) was auf die Finger.

Model-Objekte sind nicht ad-hoc *POJOs*, die einfach herumgereicht werden, oder in *Session* oder *Request* gelegt werden wie bei vielen anderen Web-Frameworks. Jede Komponente hat explizit genau ein typisiertes Model-Objekt, und alles ist eine Komponente inklusive *Input*-Felder, Formulare und auch Seiten. Die Komponenten arbeiten dann immer mit ihren Models weiter, darum ist es wichtig Models anzugeben, wenn man eine Komponente erzeugt:

```
IModel<String> nameModel = new Model<String>();
add(new TextField<String>(„name“, nameModel));
```

In diesem Fall wird das *TextField*, wenn es ausgefüllt wird, den Text ins *nameModel* legen. *IModel* ist eine einfache *setter/getter*-Hülle um ein Objekt. Wenn man den Text in ein Feld eines POJOs legen will, kann man kompliziertere Model-Implementierungen benutzen, z. B. das *PropertyModel*:

```
add(
  new TextField<String>(
    „name“, new PropertyModel<String>(person, „name“)
  )
);
public class Person {
  private String name;

  // setter, getter
}
```

Objektorientierung

Wicket-Komponenten können genauso objektorientiert verwendet werden wie andere Java-Komponenten.

Komponenten können andere vererben und weitere Funktionalität hinzufügen. Wicket lässt das nicht nur zu sondern fördert damit, dass Projekte eigene Komponenten herstellen und diese wiederverwenden. Folgendes ist z. B. eine mögliche Basisklasse für Seiten, die für jede Seite die Navigationskomponente schon anzeigt:

```
public abstract class BasePage<T>
  extends GenericWebPage<T> {
  public BasePage() {
    Navbar navbar = new Navbar(„navbar“);
    navbar.setPosition(Navbar.Position.TOP);
```

```
        navbar.addComponents( ... ); // Menuitems
        add(navbar);
    }
}
<html>
    <head><title>Test</title></head>
    <body>
        <div wicket:id="navbar"/>
        <wicket:child />
    </body>
</html>
```

Man muss dann diese *BasePage* nur vererben um das Menü zu haben. <*wicket:child /*> spezifiziert hier, dass alle Komponenten die zu der Seite von Subklassen hinzugefügt werden an diese Position kommen sollen.

Komponenten sind immer typsicher. Das bezieht sich auf Links zwischen Seiten (wie auch die Startseite), bei denen man die Seitenklasse oder das Seitenobjekt mit Zustand angeben kann, und es bezieht sich auch auf Komponenten und Model-Objekte. Man muss niemals casten oder Objekte in generische *Maps* ablegen, nicht einmal bei *Session*-Objekten.

Standards und Support

Der unbestrittene Standard für Java-Web-Entwicklung ist JSF. Allerdings muss man beachten, was ein solcher Standard für Vorteile bringt für ein Projekt. Besonders im IT-Umfeld, wo sich die Anforderungen sehr schnell ändern können, sowie das Wissen und die Technologie. Es passiert oft im Java-Umfeld, dass sich Standards sehr langsam und mit Mühe entwickeln, wo Drittanbieter (*Free Software*-Projekte oder sogar kommerzielle Projekte) Lichtjahre weiter sind. Beispiele dazu sind: *CDI* versus *Spring*, *Java Logging* versus *Log4j*, *JPA* versus *Hibernate*.

Wichtiger ist, dass man eine Gemeinschaft, Dokumentation und Unterstützung erhält. Das alles ist mit Wicket auch gegeben.

Testing

Ein sehr wichtiger Punkt bei Wicket ist, dass auch *Unit Testing* eingebaut ist. Da alle Komponenten eigentlich POJOs sind und keine statischen Verbindungen mit versteckten Systemen oder Lebenszyklen haben, sind sie leicht zu testen. Aber auch darüber hinaus kann man mit einer speziellen Wicket-Komponente andere Komponenten oder sogar Seiten komplett durchtesten inklusive wie sie auf *Ajax-Events* reagieren, blitzschnell und ganz ohne Server.

Folgender Test schaut nach ob die Startseite den richtigen Text anzeigt:

```
@Test
public void testStartPageHasCorrectGreeting() {
    WicketTester tester =
        new WicketTester(new Application());
    tester.startPage(HomePage.class);

    tester.assertLabel("greeting", "Hello, World!");
}
```

Fazit

Diese Zusammenfassung dient dazu ein Gefühl für Wicket zu bekommen, so dass es nicht unnötig aus Evaluierungen ausfällt.

Viele interessante und wichtige Themen hatten hier keine Erwähnung, bei denen Wicket auch ganz viele gute Ideen zu bieten hat:

- Zustandslosigkeit und Zustandshandling
- *Redirect-after-POST* Fähigkeit
- Internationalisierung
- Formulare
- Navigation
- JavaScript und AJAX
- Integration mit EJBs, CDI, Spring, Bootstrap usw.

Ein guter Ansatzpunkt für diese Themen ist die Wicket-Dokumentation [1].

Referenzen

[1] The Apache Software Foundation *Apache Wicket – User Guide,* http://wicket.apache.org/guide/guide/index.html

[2] Wicket *Beispielprojekte*, https://github.com/bitstorm/Wicket-tutorial-examples

[3] Jaxenter *Wicket Tutorial*, http://jaxenter.com/tutorial-apache-wicket-the-fun-web-framework-42770.html

[4] Wicket *Mailingliste Archiv,* http://mail-archives.apache.org/mod_mbox/wicket-users

[5] Springer Verlag *Home* http://springer.de

Kurzbiographie

Robert Bräutigam ist als Senior-Consultant für die MATHEMA Software GmbH tätig. Er ist seit 1999 als Entwickler und Architekt im Java Enterprise-Umfeld beschäftigt und interessiert sich für leichtgewichtige Lösungen in Technologien sowie im Projektmanagement.

Abra Kadabra?

Generierung im Werkzeugkasten des JEE-Entwicklers

von Frank Ganske

Immer wieder gibt es diese Aufgaben – wiederkehrende, gleichförmige Software-Komponenten, die langweilig und fehlergefährdet sind, wenn sie manuell erstellt werden. Das soll automatisiert werden, im kleinen Rahmen, ohne den regulären Projekt- und Bauprozess zu stören. Im KaffeeKlatsch vom Juni 2013 wurde unter dem Titel „Simsalabim!“ beschrieben, wie das mit *Velocity* und *Ant* gelöst wird. Prompt fand sich der Autor danach in einer Umgebung wieder, in der die eingesetzten Hilfsmittel klar und restriktiv definiert sind. Wie macht man es also mit den Bordmitteln eines Java-EE-Entwicklers? Zusätzlich wird gezeigt, wie aus einem großen XML-Modell Teilbereiche selektiert und so kleine Teilmodelle erstellt werden. Dieser häufige Einsatzfall führt den ersten Artikel weiter.

Die Umgebung

Der einsetzbare Werkzeugkasten ist bestimmt durch einen JEE-Server und das *Oracle Enterprise Pack for Eclipse* (OEPE) – in einer Juno (3.8) Version. Darin ist Velocity übrigens enthalten. Das Eclipse Plug-in *org.apache.velocity_1.5.0.v200905192330.jar* ist ein *OSGI-Bundle* und beschreibt in der *META-INF/MANIFEST.MF* seine Abhängigkeiten. Gleichzeitig ist es ein JAR, das man auch allein verwenden könnte. Es wurde dagegen entschieden, diese JARs zusammenzuklauben. Gegen Velocity spricht auch, dass das Modell mit *JDOM* statt mit dem inzwischen üblichen *Document Object Model* (DOM) des W3C angesprochen wird und die abweichende API schnell für Verwirrungen sorgt.

Das Vorgehen entspricht dem mit Velocity. Es gibt ein oder mehrere *Templates* und für jede Ausgabedatei ein Modell. Auch das beschriebene Ant-Target *master* zur Vervielfältigung der Modelle, bei mehreren Java-Klassen je Datenelement, kann aus dem vorherigen Artikel übernommen werden. Es ist sinnvoll, diesen Schritt von der eigentlichen Code-Generierung zu trennen. (Für den grundsätzlichen Aufbau der Ant-Scripte siehe [1]). Dies ist der Beginn des Ant-Skripts bis zum Aufruf des ersten Beispiels:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="generator2" default="clean">

  <property name="build.dir" location="${basedir}/build" />
  <property name="dist.dir" location="${basedir}/dist" />
  <property name="templates.dir" location=
    "${basedir}/templates" />
  <property name="lib.dir" location="${basedir}/libs" />

  <target name="prepare">
    <mkdir dir="${build.dir}" />
    <mkdir dir="${dist.dir}" />
  </target>

  <target name="clean" depends="prepare"
    description="Delete artifacts">
    <delete dir="${build.dir}"
      includes="**/*" includeemptydirs="yes" />
    <delete dir="${dist.dir}"
      includes="**/*" includeemptydirs="yes" />
  </target>

  <target name="version1" depends="clean"
    description="Generate Android example">
    <xslt includes="ItemDetailFragment.xml"
      destdir="${dist.dir}"
      extension=".java"
      style="${templates.dir}/DetailFragment.xslt" />
  </target>
```

XSLT statt Velocity

Ant bietet den *XSLT-Task*. Dieser *MatchingTask* kann Modelle in Verzeichnissen mit *in-/excludes* abarbeiten, aber aus einem Modell wird immer eine Ausgabedatei. Die Code-Generierung erfolgt mit XSLT, das eine XML-Syntax für Schleifen, Bedingungen und Ausdrücke zur Steuerung der Ausgabe innerhalb des Templates bietet. Mit *XPath* werden dabei Modellelemente adressiert, also die Navigation innerhalb der XML-Daten umgesetzt. XPath ist eine Sprache, die im einfachsten

Fall wie ein Dateisystempfad erscheint. Um Attribute und Entitäten oder Namensräume in den XML-Strukturen anzusprechen, wird es schnell komplexer. Es gibt Funktionen zur Fehlerbehandlung, für *Strings*, Datum und Zeit, Zahlen und vieles mehr. Eine übersichtliche Referenz und Beispiele für XSLT und XPath findet man auf [2].

XSL-Templates sind XML-Dateien. Zeichen, die in XML besondere Bedeutung haben (&, ", < und >), müssen maskiert werden. Man muss damit rechnen, dass Tools die Templates verändern, beispielsweise automatisch formatieren. So etwas kann zu syntaktisch richtigem, aber nicht mehr lesbaren generiertem Code führen. Im Eclipse XML-Editor wird das Element <*xsl:text*> richtig erkannt und die Java-Formatierung im folgenden Beispiel *DetailFragment.xslt* bleibt bestehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text" />
  <xsl:template match="/">
  <xsl:text>package </xsl:text>
```

Hier wird der Wert eines XPath-Ausdrucks ausgegeben und dann mit <*xsl:text*> nahtlos das Semikolon:

```
    <xsl:value-of select="/*/@package" />
    <xsl:text>;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import </xsl:text>
    <xsl:value-of select="/*/@package" />
    <xsl:text>.dummy.DummyContent;

public class </xsl:text>
    <xsl:value-of select="/*/@type" />
    <xsl:text>DetailFragment extends Fragment {

    public static final String ARG_ITEM_ID = "item_id";

private DummyContent.Dummy</xsl:text>
    <xsl:value-of select="/*/@type" />
    <xsl:text> mItem;

public </xsl:text>
    <xsl:value-of select="/*/@type" />
    <xsl:text>DetailFragment() {
}

@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  if (getArguments().containsKey(ARG_ITEM_ID)) {
    mItem = DummyContent.ITEM_MAP.
      get(getArguments().getString(ARG_ITEM_ID));
  }
}

@Override
public View onCreateView(LayoutInflater inflater,
  ViewGroup container, Bundle savedInstanceState) {
  View rootView = inflater.inflate(
    R.layout.fragment_${table}_detail, container, false
  );
  if (item != null) {
    ((TextView) rootView.findViewById(
      R.id.idTextView
    )).
    setText(String.valueOf(item.getId()));</xsl:text>
```

Und so wird die Ausgabesteuerung mit *for-each* programmiert:

```
    <xsl:for-each select="/*/field[@ui]">
      <xsl:text>
      ((TextView) rootView.findViewById(R.id.</xsl:text>
      <xsl:value-of select="@field" />
      <xsl:text>TextView)).setText(item.get</xsl:text>
      <xsl:value-of select="@as" />
      <xsl:text>());</xsl:text>
    </xsl:for-each>
    <xsl:text>
    }
    return rootView;
  }

}
</xsl:text>
  </xsl:template>
</xsl:stylesheet>
```

XSLT benötigt mit 66 Zeilen deutlich mehr Platz als das Velocity-Template mit 46 Zeilen für das gleiche Ergebnis. Trotzdem sind die Templates klar gegliedert und lesbar.

Die Modelldatei dieses Beispiels *ItemDetailFragment.xml* ist:

```
<DetailFragment table="item" type="Item"
  package="com.example.listdetailexample">
  <field type="String" field="description"
    as="Description" ui="Bezeichnung" />
  <field type="int" field="amount"
    as="Amount" ui="Anzahl" />
  <field type="Date" field="appointment"
    as="Appointment" ui="Termin" />
</DetailFragment>
```

Verwendete XPath-Ausdrücke:

- */*/@package* selektiert *com.example.listdetailexample* den Inhalt des Attributs mit dem Namen *package* des *Root*-Elements, dessen Name durch das Wildcard * beliebig ist, */*/@type* selektiert *Item*
- */*/field[@ui]* selektiert drei Elemente mit dem Namen *field*, die ein Attribut mit dem Namen *ui* besitzen
- *@field* selektiert *description*, *amount* und dann *appointment* die Inhalte des Attributs mit Namen *field* aus dem selektierten Element in der Schleife, *@as* selektiert *Description*, *Amount* und *Appointment*

XSLT ist seit JDK 1.5 durch die *Xalan*-Implementierung der Apache Foundation enthalten. Allerdings wurde hier eine Kopie der Software unter dem Paket *com.sun* integriert, also modifiziert und der regulären Pflege entzogen. Das ist nie gut. Beispiel dafür ist ein unangenehmer Fehler, der zwischen den Eclipse-, Ant- und JDK-Entwicklern hin und hergeschoben wurde [3]. Das Ant-Skript bricht beim *JUNITREPORT-Task* (oder auch XSLT-Task) in Eclipse mit einem Fehler ab. Wenn man JDK, Ant und Eclipse nicht ändern kann, ist man blockiert. Workaround war *org.apache.xalan_2.7.1.v201005080400.jar* und *org.apache.xml.serializer_2.7.1.v201005080400.jar* aus *OEPE* zum *Classpath* der Eclipse-Startkonfiguration des Ant-Skripts hinzuzufügen. Wer kann, sollte die Apache-Originale von *Xalan* und *Xerces* denen des JDK vorziehen. Dr. Michael Kay, der Entwickler von *Saxon* und Editor der W3C XSLT Working Group, schreibt dazu [4]:

> *The parser in the JDK was a fork of Xerces, but it is very buggy. I would recommend production applications always to use the Apache version of the parser in preference. The bugs are rare, but they are unpredictable, and they don't only affect corner cases that aren't seen in real life; I've seen many cases where quite boring XML documents are being parsed, and corrupt data is passed to the application for attribute values. Sun/Oracle have shown no interest in fixing the problem. Use Apache Xerces every time.*

Vorbereitung der Eingabedaten

Die Beispieldatei für das Datenmodell ist das öffentlich zugängliche Verzeichnis aller Gemeinden von Österreich als XML [5] von Statistik Austria [6]. Als Beispiel einer Programmkomponente gelten alle Gemeinden deren Name mit A beginnt. Die generierten Klassen könnten jeweils die Postleitzahl und den Namen einer Gemeinde ausgeben. Man könnte sich stattdessen die Ausgabe von Fehlermeldungen aus Modellen des Fachkonzepts vorstellen.

Als Eingabedatei dient also eine recht große XML-Datei, aus der nur bestimmte Bereiche für das angenommene Modul relevant sind. Daraus sollen einzelne Java-Klassen für jedes selektierte Element generiert werden. Unabhängig von der Umsetzung ist die Aufteilung großer Modelle wesentlich für die Performanz. Man kann mit XPath-Ausdrücken syntaktisch leicht und präzise auf beliebige Knoten zugreifen, aber dabei wird immer das gesamte Modell gescannt. Deshalb muss man semantisch zusammenhängende Bereiche unbedingt voneinander trennen. Programmatisch erstellt man dazu neue *org.w3c.dom.Document*-Objekte und übernimmt die selektierten Elemente mit *adoptNode(Node)*.

Es gibt ein XSLT-Element <*xsl:result-document*>, mit dem sich mehrere Ausgabedateien erzeugen lassen. Das ist XSLT Version 2.0 und das von Xalan bereitgestellte XSLT ist Version 1.0. Dazu ist Saxon erforderlich [7]. XSLT Version 2.0 enthält weitere sehr nützliche Erweiterungen für Code-Generierung, wie das angeben von Namespaces mit dem Platzhalter *:. Liegt das JAR im Classpath von Ant, stehen die neuen Funktionen zur Verfügung. Das folgende Template *spliter.xslt* verwendet Version 2.0:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Neben der Standardausgabe des Templates wird für die generierten Teilmodelle ein Format definiert, das über seinen Namen *xml* angesprochen wird und die Ausgabe als XML behandelt:

```
<xsl:output method="text" />
<xsl:output method="xml"
  indent="no" omit-xml-declaration="yes"
  name="xml" />
<xsl:template match="/">
```

Namespaces richtig zu behandeln ist mit XPath 1.0 nicht einfach. Der Version 2.0-Ausdruck //*:*datensatz* selektiert mit dem Wildcard *: beliebige Namespaces inklusive keinen:

```
<xsl:for-each
  select="//*:datensatz[starts-with(element[2],'A')]">
```

Ein Template hat immer eine Ausgabe, die hier für *Log*-Ausgaben verwendet wird:

```
<xsl:value-of
  select="concat(element[5],' ',element[2],'&#13;&#10;')" />
```

Die zusätzlichen Ausgabedateien werden erstellt und mit den Elementen der Schleife im XML-Format gefüllt:

```
    <xsl:result-document href="{concat(element[1],'.xml')}"
      format="xml">
      <xsl:copy-of select="." />

    </xsl:result-document>

    </xsl:for-each>
    </xsl:template>

</xsl:stylesheet>
```

Es folgt der zweite Teil des Ant-Skripts. Durch das *Path*-Element müssen JARs nicht mit genauer Versionsnummer wie *lib/saxon9he.jar* angegeben werden, sondern im *Fileset* mit Wildcards:

```
<target name="version2" depends="clean"
  description="Create input models">
  <path id="xslt2.path">
    <fileset dir="${lib.dir}"
      includes="saxon*.jar" />
  </path>
  <xslt in="gemliste_knz.xml" out="${dist.dir}/split.log"
    style="${templates.dir}/splitter.xslt"
    classpathref="xslt2.path" />
</target>
```

Mit XSLT Version 2.0, bereitgestellt durch die Saxon-Implementierung, ist es möglich aus einem großen Modell genau die Daten zu extrahieren, die für die Code-Generierung erforderlich sind.

Alternativ ein Ant-Task

Solche weit verbreiteten Techniken, jeder Browser kann XSLT, müssen viele Aspekte berücksichtigen und deshalb wird so etwas kaum verändert. In der vorhandenen restriktiven Umgebung gab es jedenfalls kein Saxon. Ist XSLT Version 2.0 nicht verfügbar, bleibt ein selbst programmierter Ant-Task. Das ist kein Hexenwerk. Schließlich sind in den Ant-Sourcen jede Menge Beispiele für Tasks enthalten. Eine Datei soll verarbeitet werden, deshalb wird vom *org.apache.tools.ant.Task* abgeleitet. Bei mehreren Eingaben wäre *MatchingTask* die richtige Wahl. In der Methode *split()* wird das gesamte *DOM-Document* in den Speicher geladen. Bei sehr großen Eingabemodellen müsste man eine sequenzielle Lösung mit *SAX* oder *StAX* implementieren. Die mit dem übergebenen XPath-Ausdruck ausgewählten Elemente werden in einzelne *Document*-Instanzen verschoben. Die Methode *write()* schreibt sie in Dateien entsprechend dem zweiten übergebenen XPath-Ausdruck.

```
package xmlconverter;

import java.io.BufferedInputStream;
import java.io.BufferedOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.xpath.XPath;
import javax.xml.xpath.XPathConstants;
import javax.xml.xpath.XPathExpressionException;
import javax.xml.xpath.XPathFactory;

import org.apache.tools.ant.BuildException;
import org.apache.tools.ant.Task;
import org.apache.tools.ant.util.FileUtils;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

public class XmlSplitTask extends Task {
  private File inFile;
  private File destDir;
  private String selectionExpression;
  private String filenameExpression;
  private void setInFile(File inFile) {
    this.inFile = inFile;
  }
  public void setDestdir(File destDir) {
    this.destDir = destDir;
  }
  public void setSelectionExpression(String xpath) {
    this.selectionExpression = xpath;
  }
  public String getSelectionExpression() {
    return selectionExpression;
  }
  public void setFilenameExpression(String xpath) {
    this.filenameExpression = xpath;
  }

  public String getFilenameExpression() {
    return filenameExpression;
  }

  public void execute() throws BuildException {
    if (destDir == null) {
      throw new BuildException(
        "dstdir attribute must be set!"
      );
    }
```

```
    if (inFile != null && !inFile.exists()) {
      throw new BuildException(
        "input file " + inFile + " does not exist"
      );
    }
    split(inFile);
  }

  private void split(File file) {
    InputStream is = null;
    try {
      is = new BufferedInputStream(
        new FileInputStream(file)
      );
      DocumentBuilder db = getDocumentBuilder();
      Document document = db.parse(is);
      NodeList nodes = (NodeList) getXPath().evaluate(
        selectionExpression,
        document.getDocumentElement(),
        XPathConstants.NODESET
      );
      for (int i = 0, count = nodes.getLength(); i < count; i++) {
        Document doc = db.newDocument();
        doc.appendChild(doc.adoptNode(nodes.item(i)));
        write(doc);
      }
    } catch (XPathExpressionException e) {
      throw new BuildException(
        "Error evaluating element by " +
        selectionExpression + " in file " + inFile, e
      );
    } catch (IOException e) {
      throw new BuildException(
        "Error reading input file " + inFile, e
      );
    } catch (SAXException e) {
      throw new BuildException(e);
    } catch (ParserConfigurationException e) {
      throw new BuildException(e);
    } finally {
      FileUtils.close(is);
    }
  }

  private void write(Document doc) {
    OutputStream os = null;
    try {
      String fileName =
        getXPath().evaluate(
          filenameExpression,
          doc.getDocumentElement()
        );
      File file = new File(destDir, fileName);
      if (!file.exists()) {
        FileUtils.getFileUtils().createNewFile(file, true);
      }
      os = new BufferedOutputStream(
        new FileOutputStream(file)
      );
      XmlWriter w = new XmlWriter(os);
      w.write(doc);
      w.flush();

    } catch (XPathExpressionException e) {
      throw new BuildException(
        "Error evaluating filename by " +
        filenameExpression + " in " + doc, e
      );
    } catch (IOException e) {
      throw new BuildException(
        "Error writing output file", e
      );
    } finally {
      FileUtils.close(os);
    }
  }

  private DocumentBuilder getDocumentBuilder()
    throws ParserConfigurationException {
    DocumentBuilderFactory df =
      DocumentBuilderFactory.newInstance();
        // to strip namespaces
    df.setNamespaceAware(false);
    df.setIgnoringComments(true);
    df.setIgnoringElementContentWhitespace(true);
    DocumentBuilder db = df.newDocumentBuilder();
    return db;
  }

  private XPath getXPath() {
    XPathFactory xf = XPathFactory.newInstance();
    XPath xp = xf.newXPath();
    return xp;
  }
}
```

Interessant ist hier noch die Methode *getDocument-Builder()*, die mit *setNamespaceAware(false)* dafür sorgt, dass man keine Namespace-Präfixe angeben muss. Die *write()*-Methode verwendet eine zweite Klasse *XmlWriter* um das *org.w3c.dom.Document* zu serialisieren. Besonders an der *XmlWriter*-Klasse ist die Methode *masq(),* die neben den in XML zu maskierenden Zeichen (ja, das gab es) auch solche berücksichtigt, die in den *Strings*, die im generierten Java-Code ausgegeben werden sollen, zu Problemen führen. Sonst müsste man sich im Template darum kümmern:

```
package xmlconverter;

import java.io.IOException;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.util.regex.Pattern;
```

```
import org.w3c.dom.Attr;
import org.w3c.dom.CDATASection;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Text;

public class XmlWriter {

   private static final Pattern JAVA_PATTERN =
      Pattern.compile("[\\\"]");

   private static final Pattern AMP_PATTERN =
      Pattern.compile("&"), //
      QUOT_PATTERN = Pattern.compile("\""), //
      LT_PATTERN = Pattern.compile("<"), //
      GT_PATTERN = Pattern.compile(">");

   private static String endl =
      System.getProperty("line.separator");

   private Writer writer;

   private boolean isNewLine;

   public XmlWriter(OutputStream os)
      throws IOException
   {
      this.writer = new OutputStreamWriter(os, "UTF-8");
      this.isNewLine = true;
   }


   public void write(Element element, int tabs)
      throws IOException
   {
      if (!isNewLine)
      write(endl);
      if (!element.hasChildNodes()) {
         writeTabs(tabs);
         write(
            "<" + element.getTagName() +
               writeAttributes(element) + " />"
         );
      } else {
         writeTabs(tabs);
         write(
            "<" + element.getTagName() +
            writeAttributes(element) + ">"
         );
         write(element.getChildNodes(), tabs + 1);
         if (isNewLine)
            writeTabs(tabs);
         write("</" + element.getTagName() + ">");
      }
      write(endl);
   }

   public void write(Document doc)
      throws IOException
   {
      write(doc.getDocumentElement(), 0);
   }

   public void write(NodeList nodes)
      throws Exception
   {
      write(nodes, 0);
   }


private String writeAttributes(Element element) {
      StringBuffer attributeString = new StringBuffer();
      NamedNodeMap attributeMap =
         element.getAttributes();
      int length = attributeMap.getLength();
      for (int i = 0; i < length; i++) {
         Attr attributeNode = (Attr) attributeMap.item(i);
         String name = attributeNode.getName();
         String value = masq(attributeNode.getValue());
         attributeString.append(" ").append(name)
            .append("=\"").append(value).append("\"");
      }
      return attributeString.toString();
   }

   private void write(NodeList nodes, int tabs)
      throws IOException
   {
      for (int i = 0; i < nodes.getLength(); i++) {
         Node node = nodes.item(i);
         write(node, tabs);
      }
   }

   private void writeText(Text text)
      throws IOException
   {
      String nodeValue = masq(text.getNodeValue());
      write(nodeValue.trim());
   }

   private void writeCdata(CDATASection cData)
      throws IOException
   {
      String cDataText = "<![CDATA[" + masq(
         cData.getNodeValue()
      ) + "]]>";
      write(cDataText);
   }

   private String masq(String text) {
      String result;
      result = JAVA_PATTERN.matcher(text)
         .replaceAll("\\\\$0");
      result = AMP_PATTERN.matcher(text)
         .replaceAll("&amp;");
      result = QUOT_PATTERN.matcher(text)
         .replaceAll("&quot;");
      result = LT_PATTERN.matcher(text)
         .replaceAll("&lt;");
      result = GT_PATTERN.matcher(text)
         .replaceAll("&gt;");
      return result;
   }
```

```
    private void write(Node node, int tabs)
        throws IOException
    {
        if (node instanceof Element) {
            write((Element) node, tabs);
        } else if (node instanceof CDATASection) {
            writeCdata((CDATASection) node);
        } else if (node instanceof Text) {
            writeText((Text) node);
        } else {
            throw new IOException(
                "XmlWriter: unsupported node type: " +
                    node.getClass()
            );
        }
    }

    private void writeTabs(int tabs) throws IOException {
        for (int i = 0; i < tabs; i++) {
            write("  ");
        }
    }

    private void write(String value) throws IOException {
        if (value == null || "".equals(value)) {
            return;
        }
        isNewLine = endl.equals(value);
        writer.write(value);
    }

    public void flush() throws IOException {
        writer.flush();
    }

    public void close() throws IOException {
        writer.close();
    }
}
```

Der Schluss des Ant-Skripts compiliert den Task und verwendet den eigenen Classpath für die Ant-Abhängigkeiten. Anschließend wird der Task definiert und eingesetzt. Die angegebenen XPath-Ausdrücke entsprechen dem XSLT Version 2.0-Template *splitter.xslt* ohne Namespace:

```
    <target name="compile" depends="prepare">
        <javac srcdir="src" destdir="${build.dir}"
            includeantruntime="true" />
    </target>

    <target name="standalone" depends="clean, compile"
        description="Create input models with an Ant task">
        <taskdef name="split"
            classname="xmlconverter.XmlSplitTask"
            classpath="${build.dir}" />
        <split infile="gemliste_knz.xml" destdir="${dist.dir}"
            selectionExpression=
                "//datensatz[starts-with(element[2],'A')]"
            filenameExpression="concat(element[1],'.xml')" />
    </target>

</project>
```

Der Ant-Task erstellt die gleichen Modelldateien, wie die Lösung mit XSLT Version 2.0, ohne Saxon zu benötigen.

Zusammenfassung

Gezeigt wurde Code-Generierung als Werkzeug für den Java-Entwickler mit möglichst wenig Abhängigkeiten. Statt Velocity kann XSLT zur Generierung verwendet werden, das in Java enthalten ist. Werden statt XML Sourcen generiert, sollte man <*xsl:text*> verwenden, um die Formatierung zu erhalten. Die Versionen der XML-Bibliotheken können meist einfach über den Classpath gesteuert werden. Die Vorbereitung der Daten, die ja aus verschiedensten Quellen stammen können, ist von der eigentlichen Generierung getrennt. Es wird so einfacher und flexibler. Das Splitten eines Modells in mehrere wurde mit XSLT Version 2.0 und mit einem eigenen Ant-Task realisiert.

Referenzen

[1] Ganske, Frank *Simsalabim!,* KaffeeKlatsch Juni 2013

[2] w3schools *XSLT-Tutorial inclusive XPath* http://www.w3schools.com/xsl/default.asp

[3] Eclipse Bugs *Fehlermeldung: Ant build fails in junitreport using Java 6u32 or later,* https://bugs.eclipse.org/bugs/show_bug.cgi?id=384757

[4] Stackoverflow *Beitrag zu JDK 1.6 and Xerces? mit Kommentar des Saxon Entwicklers,* http://stackoverflow.com/questions/7794281/jdk-1-6-and-xerces

[5] Download Beispieldatei http://www.statistik.at/web_de/static/gemeinden_sortiert_nach_gemeindekennziffer_mit_status_und_postleitzahlen_x_025150.zip

[6] Statistik Austria *Klassifikationen* http://www.statistik.at/web_de/klassifikationen/regionale_gliederungen/gemeinden

[7] SAXON *The XSLT and XQuery Processor* http://saxon.sourceforge.net

Kurzbiografie

Frank Ganske (frank.ganske@mathema.de) ist Senior Developer bei der MATHEMA Software GmbH in Erlangen. Er entwickelt seit 1991 betriebswirtschaftliche Individual- und Standardsoftware. Besonderes Interesse hat er an Veränderungsprozessen und deren Absicherung. Das umfasst sowohl Bauprozesse, Testautomatisierung, statische Code-Analyse und die Feststellung von Systemeigenschaften und -abweichungen.

You've been PhoneGapped

Framework-Entscheidungen, JQueryMobile und graue Haare – ein bitterer Erfahrungsbericht

von Tim Bourguignon

Man hat manchmal komische Ideen, wie zum Beispiel eine mobile Applikation in einer neuen Technologie zu schreiben, nur um danach festzustellen: „Tja, ging doch"! In diesem Fall handelt es sich um eine mobile Applikation mit *PhoneGap*, und sie funktioniert noch nicht. Dies ist mein Erfahrungsbericht.

Es hat alles relativ harmlos angefangen: ein Kollege hat ein neues Smartphone gekauft und ist auf die Idee gekommen eine App für unsere jährliche Konferenz zu basteln [1]. Sein Smartphone hat ein *Android*-Betriebssystem und in meiner Hosentasche regiert das iPhone: Problem! Wie jeder gute Techie habe ich mir gedacht: „so eine App hätte ich auch gerne", und wie bei jedem guten Entwickler haben meine grauen Zellen sofort geantwortet: „vielleicht kann ich es selber schreiben?" So etwas ist jedem von uns schon mal passiert oder?

Außer iPhone, iPad und iPod (hum) bin ich leider kein Apple-Nutzer und habe noch nie die Finger in *Objective-C* reingesteckt. Dies ist für *iOS*-Entwicklung aber ein Muss. Was ich aber halbwegs gut kann ist Web-Entwicklung und wie im KaffeeKlatsch schon beschrieben wurde, ist PhoneGap ein Kinderspiel [2]. Ich habe mich also entschieden PhoneGap auszuprobieren und so eine App zu schreiben!

Was soll sie tun?

Diese App sollte erst mal die Konferenz unterstützen, Vorträge, Referenten und den Zeitplan darstellen. Also ein reines Datendarstellungsprogramm. Sie sollte *out-of-the-box* offline funktionieren, aber auch neue Daten holen können. Ein paar andere Features wie *Push-Notifications* und Twitter-Unterstützung wären noch „nice to have". Grundsätzlich kein großes Ding.

PhoneGap?

PhoneGap wurde schon im Detail im KaffeeKlatsch vorgestellt, ich werde also nicht zu viel darüber erzählen. Was man wenigstens wissen sollte:

1. Es ist ein *HTML5*-basiertes (*HTML*, *JavaScript*, *CSS*) Framework um mobile Apps zu implementieren.
2. Pro Plattform (iPhone, Android usw.) bastelt man eine mini-native Applikation. Diese native Applikation startet einen *WebView*, der als *Host* für eine Web-Applikation dient.
3. Diese Web-Applikation entwickelt man in HTML5 anhand von Standard-Web-Entwicklungstools. Bis auf feine Unterschiede sollte diese Web-Applikation nur einmal geschrieben werden.
4. Am Ende der Pipeline bekommt man eine native App, die in jedem *AppStore* akzeptiert wird.

Begriff-Salat

Eines müssen wir aber trotzdem klären: Was repräsentiert PhoneGap wirklich?

PhoneGap wurde 2009 von der Firma Nitobi entwickelt. Kurz danach wurde es von Adobe Systems [3] gekauft und an die Apache Foundation übergeben. Es wurde dann in *Apache Callback* umbenannt, aus dem danach *Apache Cordova* [4] geworden ist. Adobe hat aber den Namen PhoneGap für sich behalten und produziert unter diesem Namen ein Fork von *Cordova* und verkauft ein *BuildSystem* unter dem Namen *PhoneGap Build* [5].

PhoneGap ist also ein Oberbegriff für Cordova und PhoneGap, was die Suche im Internet natürlich nicht einfacher macht.

Um das Problem noch weiter zu vertiefen, kann man mit Telerik's *Icenium* [6] in .NET arbeiten, was auch Cordova-basiert ist. Dazu müsste man ggf. nach PhoneGap, Cordova und Icenium im Internet suchen. Wie praktisch!

Naja, für diesen Artikel – und auch für viele PhoneGap-Entwickler – fällt das alles unter „PhoneGap".

Unser Beispiel

In unserer App geht es um Datendarstellung. Wir haben circa 100 Vorträge und 60 Referenten. Die App sollte

den zeitlichen Ablauf darstellen und eine Schlüsselwortsuche auf Vortragsinhalte und Viten der Referenten ermöglichen.

Die App funktioniert offline, die Daten sind relativ fest. Wir verwenden also keine Kamera, Gyroscope, GPS oder Speichermechanismen der PhoneGap-API. Von PhoneGap werden wir nur die native Schale verwenden. Eins bleibt uns aber: der UI-Kampf.

„Nur" einmal schreiben?

Nicht so lange her hieß es: Design-Style der Plattform! iOS muss nach iOS aussehen, Android nach Android. Dies hatte als Auswirkung, dass man eine UI pro Plattform produzieren musste; was PhoneGap deutlich weniger interessant gemacht hätte.

Die Trends gehen heutzutage mehr Richtung *Branding*-Design, nämlich ein einheitliches Design, das Ihre Organisation oder das Produkt spiegelt und weniger nach iOS oder Android aussieht.

Natürlich muss man sich an manche Navigations- oder Darstellungsregeln halten, aber mit ein bisschen Mühe kann man eine relativ einheitliche UI produzieren und diese nur einmal schreiben.

Willkommen im wilden Westen

Wer PhoneGap verwenden und mehr als eine Plattform angreifen will, muss sich aber darüber klar sein, dass es sich um Web-Entwicklung mit einer unendlichen Palette an Zielgeräten handelt. Da wo iOS-Entwickler sich auf zwei Bildschirmauflösungen und eine handvoll Geräte konzentrieren können, landet man mit PhoneGap im mobilen Dschungel, was massive Design-Auswirkungen hat. Ohne *Responsive* Framework empfehle ich zum Beispiel gar nicht erst zu starten. Ohne UI-Framework müsste man die Browser-Unterschiede selbst glatt bügeln, was weder lustig noch einfach ist.

Neben der Bildschirmauflösung hat man auch mit unterschiedlichen Prozesskapazitäten zu tun, was auf Performanz und *Usability* starke Auswirkungen hat.

Natürlich sind die nativen Container-Applikationen auch zu tweaken um ein einheitliches Design zu erreichen. Und noch weiter, die Browser (die den *WebView* hosten) der verschiedenen Plattformen haben kleine aber feine Unterschiede, die uns trotz PhoneGap-API das Leben schwer machen wollen.

Wie gesagt, der wilde Westen.

UI-Framework

Weil es so viele unbekannte Variablen im Projekt gab (z. B. die Anzahl an Abenden, die ich investieren kann), habe ich mir von Anfang an nur mobile Geräte als Zielplattform gesetzt – und zwar nur Smartphones, nicht mal Tablets – und als Standard die iPhone-Größe. Ich habe mich also gegen eine voll Responsive-Lösung *a-la-Bootstrap* [7] entschieden und für ein UI-Framework, das die Browser-Unterschiede möglichst vollständig abdecken kann.

JQueryMobile (JQM) [8] wird als sehr einfache Möglichkeit gesehen, um mobile Web-Seiten zu bauen (ein Artikel darüber gab es auch im KaffeeKlatsch [9]).

Ein bisschen Erfahrung mit JQM hatte ich schon gesammelt. Ich habe mich also entschieden damit anzufangen und relativ schnell meine erste Version gebaut.

Diese erste Version habe ich zu 100% auf einem *Desktop*-Rechner gebaut, in *Chrome*, mit Hilfe von Plugins wie *Ripple Emulator* [10] oder *Mobile/RDW* Tester [11], welche automatisch die Größe des Browser-Fensters an das Gerät anpassen. Alles ging gut, es war schnell und schön. Obwohl ich eine schöne *SinglePage*-Applikation gebaut hatte, waren die architekturellen Teile schön getrennt, die JavaScript-Seite klar und übersichtlich. Die Transitionen waren chic und das Design echt gut für einen ersten Entwurf. Dafür habe ich auch Nächte lang am Design geschraubt. Der Realitätstest hat sich daher etwas verzögert.

Irgendwann ist aber trotzdem die Realität aufgetaucht: ich habe versucht die App auf die Zielgeräte auszurollen. Das erste Smartphone war alt, lief mit Android 2.3 und meine App war unbedienbar. Ganz verzweifelt habe ich es auch auf einem *Entry-Level* Windows Phone 8 Smartphone ausgerollt, mit etwas besserem aber trotzdem ähnlichen Ergebnis.

Ich hatte PhoneGap als nicht blitzschnell erwartet, aber so langsam auch nicht. Da hatte ich sicherlich was falsch gemacht.

Zurück zum DesignBoard

Ein paar Abende lang habe ich im Internet nach Erfahrungen anderer Entwickler gesucht. Es gibt ungefähr so viele Ahnungen wie Autoren und es geht von „PhoneGap mit JQM? Nie und Niemals", bis zu „PhoneGap mit JQM, geil wenn man weiß wie" (wie verraten sie allerdings nicht so richtig).

Ich hatte mehrere Optionen zur Auswahl:

1. Mit JQM und diesem Design auf Performanz-Jagd gehen und ggf. verschiedene UI-Teile der Anwendung selbst implementieren.
2. Bei JQM bleiben, aber ein performanteres Design entwickeln.

3. Ein neues UI-Framework suchen oder ggf. ohne UI-Framework probieren.
4. Framework und Design wechseln.
5. PhoneGap aufgeben.

Ich habe mich also auf Performanz-Jagd gemacht.

Schlüsselwort „Beschleunigen"

Wenn man nach „JQueryMobile Performance" bei Google sucht, sind mehr als 50% der Treffer mit PhoneGap verbunden; und dies ist kein Wunder. Viele Entwickler haben dieselbe Erfahrung wie ich gemacht: HTML5 „wie immer" gebastelt, weit entfernt von mobilen Geräten gearbeitet und sich danach wundern, warum das Ganze eine schlechte Performanz hat.

Hier sind alle Tricks, die ich bisher im Einsatz gefunden habe, um meine App zu beschleunigen; und der „nullte" Tipp wäre: von Tag eins an auf Zielgeräten testen!

1. DOM-Größe

Das größte Problem mit JQuery und JQueryMobile wird die DOM-Größe sein. Wenn man mit JQM z. B. eine *ListView* baut, produziert JQM für jeden *li*-Tag eine vierstufige *li/div/div/a*-Struktur. Mit etwas mehr Inhalt sind meine *List*-Elemente schnell 6- oder 7-stufig geworden mit insgesamt 15 bis 20 Elementen. Das für mehr als 100 Vorträge fängt an eine schöne Menge zu werden.

Es ist kritisch diese Menge im Kopf zu behalten und zu versuchen sie unter Kontrolle zu halten. „Ist es wirklich nötig diese Information im DOM zu haben oder kann ich sie bei Bedarf nachladen?", ist z. B. eine Frage, die ich mir dann häufig stellen musste.

Obwohl JQM erlaubt mehrere Seiten innerhalb einer HTML-Seite zu definieren, ist es nicht empfohlen. Der DOM wird dennoch größer und umso komplexer zu navigieren (für *Caching* und *Prefetching* Tipps siehe unten Punkt vier).

Die Größe des DOMs ist natürlich nicht allein das Problem. Wird aber im Kombination mit Punkt zwei etwas kritischer.

2. JQuery-Selektoren

Dies ist wieder kein PhoneGap/JQM-Problem per se. Es liege eher unter den JavaScript „Best Practice".

Es wird für viele doof klingen, aber wie oft habe ich JS-Funktionen geschrieben ohne den ganzen *Scope* nachzuschauen und am Ende gemerkt, das JQuery-Selektoren immer wieder aufgerufen wurden. JQuery-Selektoren sind teuer. Man sollte sie besser speichern und wiederverwenden. Immer.

JQuery-Selektoren können auf verschiedene Arten mit unterschiedlichen Performanz-Effekten geschrieben werden:

- JQuery-Id-Selektoren verwenden direkt die native Browser-Methode *getElementByID()*, was natürlich am effizientesten ist.
- Um die Suche nach einem Element zu beschränken, können wir einen Context einsetzen. Es lohnt sich nicht über den gesamten DOM nach einem Element zu suchen, wenn wir schon ungefähr wissen wo es liegen soll. Im Falle von z. B. mehreren *spans* in einem *div* können wir entweder *$('#divId span')* oder *$('span', '#divId')* verwenden.
- Selektoren mit Context werden bei JQM von rechts nach links abgearbeitet. Im oberen Beispiel ist es also besser die *$('span', '#divId')*-Variante zu verwenden. Im Hintergrund wandelt JQM diese Variante in einen *$('#divId').find('span')* um, wir können also direkt diese Variante verwenden und uns noch ein bisschen Zeit sparen.

3. onClick-Verzögerung

Um Gesten zu erlauben wird künstlich eine Verzögerung von ca. 300 ms auf das *onClick()*-Event gebaut. In den meisten Fällen braucht man es nicht und könnte so viel *Responsiveness* gewinnen.

Eine Lösung ist das *Tap-Event* oder eine JavaScript-Bibliothek wie *FastClick* [12].

4. Page Prefetching / Caching

JQM erlaubt uns Seiten zu cachen, was in unserem Fall, bei dem sich die Daten selten ändern, wirklich Sinn macht. Dies kann am *div*-Element Level *viadata-dom-cache=true* direkt als globale Einstellung via *$.mobile.page .prototype.options.domCache = true* oder via JavaScript *pageContainerElement.page({ domCache: true });* eingestellt werden.

Im Fall von mehreren HTML-Seiten kann man via des Tags *data-prefetch* Seiten im Hintergrund laden. Bei mobilen Web-Seiten würde es *Bandwidth* brauchen, aber in unserer Offline-App ist es natürlich ganz praktisch.

5. Asynchron DOM Refresh

Eins, habe ich festgestellt, ist sehr teuer: den DOM dynamisch befüllen/manipulieren und auffrischen. Wenn man z. B. Elemente in einer *CollapsibleSet* (eine Liste, in der jedes Element einen klappbaren Inhalt besitzt) hinzufügt, sollte man dann etwas wie *mySet.trigger('create')*

aufrufen, so dass das *Set* neu gemalt wird. Mein 100 Elemente-*Set* vollständig dargestellt zu kriegen, hat auf den Geräten bis zu 15 Sekunden gedauert.

Die beste Lösung, die ich dafür gefunden habe, ist eine asynchrone Ladung von einzelnen Elementen (oder bis zu zehn auf einmal). Das lässt sich in JavaScript gut anhand einer *self.setInterval()*-Funktion machen. Das *Set* ist nicht sofort vollständig befüllt, der Bildschirm aber schon. In meinem Fall ging es trotzdem schnell genug, so dass es der Anwender mit einem schnellen Scrollen nicht überholen konnte. Somit ist es für den Anwender genug.

Bitteres Ende

Trotz dieser Tipps und Tricks ist meine App nicht schnell genug geworden. Sie läuft zwar sehr gut auf neueren/starken Geräten, aber auf älteren/schwächeren Kisten leiden die Antwort-Zeiten und damit die gesamte Usability.

Schlaues *Rendering* von Seiten via *asynchrone Patterns* und caching helfen um schnelle Seiten-Transitionen zu bekommen, lösen aber nicht die Schnelligkeit der Navigation innerhalb einer Seite. Die *Set*-Elemente z. B. brauchen immer noch zu viel Zeit um sich aufzuklappen. Tap-Kommandos fühlen sich wohl noch sehr fremd an; man überlegt sich immer ob das Smartphone das Event wirklich registriert hat. Das darf leider nicht sein und entspricht nicht der Qualität, die ich erwarte.

Zusätzlich sollte diese App als Tool für Entwickler-Konferenzen dienen. Wie soll ich dann meinem Techniker-Publikum erklären, dass ein Smartphone mit einem Quad-Core Processor gebraucht wird, um ein Datendarstellungsprogramm sauber laufen lassen zu können?

So geht es mit JQM nicht. Ich arbeite die Liste also weiter ab.

Wie geht's weiter?

Zuerst werde ich versuchen ein sehr einfaches Design zu schaffen, mit weniger Informationen auf jeder Seite, aber dafür mehr Seiten und mehr Navigation. Da es sich um eine App und nicht eine mobile Web-Seite handelt, sollte nachladen kein Thema sein. Was aber wichtig erscheint, ist dass jede Seite für sich schnell und bedienbar ist, dass also das native Gefühl da ist. Vielleicht lässt sich so etwas mit JQM doch machen.

Sollte es nicht der Fall sein, werde ich außerhalb von JQM suchen müssen. Ich habe schon kurz anhand von *Adobe Topcoat* [13] und *Zepto* [14] einen Prototyp erstellt. Topcoat bietet CSS für saubere und schnelle Web-Applikationen und Zepto ist eine minimalistische und schlanke Alternative zu JQuery, die für das mobile Web gedacht ist.

Beide sind also Performanzorientiert und könnten uns vielleicht helfen. Leider bin ich durch meinen Prototyp schon auf Browser-Unterschiede gestoßen, die JQM für mich glatt gezogen hatte und an denen ich hier selber arbeiten müsste. Habe ich fast alle dieser Unterschiede entdeckt oder ist das nur die Spitze des Eisberges? Vielleicht muss ich ja UI-Frameworks aufgeben und es wirklich selber implementieren?

Ich habe noch etwas Arbeit vor mir...

Referenzen

[1] Heiduk, Andreas *Vom Duke zum Droid*, Lehren eines Android-Anfängers, KaffeeKlatsch, 2013, November, http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2013-11.pdf

[2] Eberling, Werner *Mind the gap! Plattformübergreifende mobile Entwicklung mit PhoneGap*, KaffeeKlatsch 2012, Januar, http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2012-01.pdf

[3] Adobe *home* http://www.adobe.com

[4] Apache Cordova *home* http://cordova.apache.org

[5] Bd *PhoneGap Build,* https://build.phonegap.com

[6] Telerik *Telerik's Icenium* http://www.icenium.com

[7] Bootstrap http://getbootstrap.com

[8] JQueryMobile *home* http://jquerymobile.com

[9] Eberling, Werner *Web2touch, Web-Entwicklung für mobile Endgeräte mit jQuery Mobile*, KaffeeKlatsch, 2012, August, http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2012-08.pdf

[10] Chrome web store *Ripple Emulator (Beta)* https://chrome.google.com/webstore/detail/ripple-emulator-beta/geelfhphabnejjhdalkjhgipohgpdnoc

[11] Chrome web store *Mobile/RDW Tester,* https://chrome.google.com/webstore/detail/mobilerwd-tester/elmekokodcohlommfikpmojheggnbelo

[12] github *Fastclick*, https://github.com/ftlabs/fastclick

[13] Adobe Topcoat http://topcoat.io

[14] Zepto.js, http://zeptojs.com

Kurzbiographie

Timothée Bourguignon ist als Senior Developer für die MATHEMA Software GmbH tätig. Sein Spezialgebiet ist die Desktop- und Web-Programmierung mit dem .NET Framework. Daneben beschäftigt er sich mit den Neuerungen der .NET-Welt und deren Communities. Als Agilist liegt sein Schwerpunkt auf agilen Methoden, guter Teamarbeit und hoher Arbeitsqualität. Hierbei setzt er insbesondere auf die Philosophie des Software Craftmanships.

Des Programmierers kleine Vergnügen

Ohne Vorzeichen

von Michael Wiedeking

Worüber die C*-Programmierer nur müde lächeln können, versetzt die Java-Gemeinde in jähes Entsetzen. Das Rechnen mit vorzeichenlosen Zahlen ist Letzteren nämlich nicht vergönnt. Aber zum Glück kann auch diesen geholfen werden, man muss sich nur zu helfen wissen.

All zu oft muss man dann doch mal mit vorzeichenlosen Zahlen rechnen. Während das bei den meisten C-basierten Sprachen kein Problem darstellt – verfügen sie doch über *unsigned*-Varianten in allen erdenklichen Größen –, muss man in Java ohne auskommen. Glücklicherweise lassen sich alle vorzeichenlosen Operationen auch mit vorzeichenbehafteten Zahlen implementieren, wenngleich man da auch mal etwas mehr nachdenken muss.

Sämtliche Bit-Operationen sind dankbarer Weise unabhängig vom Vorzeichen. Bei den *shift*-Operationen wird ja das logische >> und arithmetische Schieben >>> unterschieden. Da Letzteres das Vorzeichen immer unberücksichtigt lässt und Nullen nachschiebt, ist so die Art des Schiebens unabhängig von der Vorzeichenbehaftung des beteiligten Typen.

Bei der Addition und der Subtraktion ändert sich auch nichts, da deren Implementierung im vorzeichenbehafteten wie im vorzeichenlosen Fall fast identisch ist. Einziger Unterschied ist das Überlaufverhalten, das aber in den meisten Programmiersprachen unbemerkt bleibt.

Bei einem 32-Bit-*int* etwa ist 0x7FFFFFFF die größte vorzeichenbehaftete Zahl. Die nächst „höhere“ Zahl ist dann 0x80000000. Interpretiert man diese Zahl als vorzeichenlose Zahl, so entspricht sie dem Wert 2 147 483 648; vorzeichenbehaftet erhält man –2 147 483 648. Also findet hier nur im vorzeichenbehafteten Fall ein Überlauf statt.

Bei der Verminderung von 0x00000000 um 1 erhält man 0xFFFFFFFF und interpretiert das im vorzeichenbehafteten Fall als –1, im vorzeichenlosen Fall als 4 294 967 295. Dementsprechend findet der Überlauf hier bei den vorzeichenlosen Zahlen statt.

```
int uAdd(int a, int b) {
  return a + b;
}

int uSub(int a, int b) {
  return a – b;
}
```

Bei der Multiplikation stellt sich die Sache zwar nicht mehr ganz so einfach dar, aber zum Glück können wir uns auch hier der vorzeichenbehafteten Multiplikation bedienen. Aber das liegt nur daran, dass wir auf die obere Hälfte des Ergebnisses nicht zugreifen können. Multipliziert man nämlich zwei 32-Bit-Zahlen miteinander, so erhält man eigentlich ein 64-Bit-Ergebnis, das dann in Abhängigkeit des Vorzeichens der beiden Faktoren ein Vorzeichen hat oder nicht. Schneidet man nun die obere Hälfte einfach ab, wie es die meisten Programmiersprachen machen, so erhält man ein Bit-Muster, das nur dann ein korrekt gesetztes Vorzeichen-Bit hat, wenn es keinen Überlauf gegeben hat. Wenn also ein Überlauf stattfindet, dann wirkt sich dieser nur auf die obere Hälfte aus, und so erhält man analog zur Addition und Subtraktion immer das „korrekte“ vorzeichenlose Ergebnis.

```
int uMul(int a, int b) {
  return a * b;
}
```

Die Division ist – wie immer – nicht trivial. Schließlich muss es ja einen Grund geben, warum diese bisher so stiefmütterlich behandelt wurde. Aber auch hier gibt es wieder eine sehr einfache und gleichermaßen elegante Lösung: Man wandelt die Zahlen vor der Division einfach in ein *long* um, wobei man die vorzeichenbehaftete Erweiterung unterdrückt.

```
long toLong(int i) {
  return i & 0x00000000FFFFFFFFL;
}
```

(Dabei darf man das *L* am Ende der Konstante auf keinen Fall vergessen. Vergisst man dies nämlich, so sieht die Zahl zwar wie eine *long*-Konstante aus, ist aber keine. Die führenden Nullen werden natürlich ignoriert und es kommt deswegen auch nicht zu einem Überlauf, der einen auf den Fehler aufmerksam machen könnte. Das Ergebnis ist dann ein unverändertes *int*, das implizit zu einem vorzeichenbehafteten *long* umgewandelt wird –

was bei einem negativen *int* leider dem gewünschten Ziel widerspricht.)

Damit erhält man zwar vorzeichenbehaftete Zahlen, allerdings sind diese wegen des ausreichend großen Zahlenraums des *long* nie negativ. Da es bei dieser Division nicht zu Überläufen kommt, kann mit deren Hilfe auch dieser Fall trivial gelöst werden (und analog auch der Rest der Division) .

```
int uDiv(int a, int b) {
   return (int) (toLong(a) / toLong(b));
}

int uMod(int a, int b) {
   return (int) (toLong(a) % toLong(b));
}
```

Mit Vergleichen von vorzeichenlosen Zahlen hatten wir uns schon im März 2010 in einem Vergnügen beschäftigt [1]. Dort hatten wir festgestellt, dass man einen vorzeichenlosen Vergleich recht leicht bewerkstelligen kann, indem man das höchstwertigste Bit mit Hilfe einer Subtraktion mit 2^{31} = 0x80000000 (allgemein 2^{n-1} für Maschinenwörter der Breite n) zu korrigieren versucht.

```
boolean uLess(int a, int b) {
   return a – 0x80000000 < b – 0x80000000;
}

boolean uLessOrEqual(int a, int b) {
   return a – 0x80000000 <= b – 0x80000000;
}
```

Die vorzeichenlosen Varianten für > und ≥ erhält man, indem man bei < bzw. ≤ die Parameter vertauscht.

Soweit so gut. Das war ja alles recht einfach. Aber was macht man denn, wenn man – wie im Falle der Division – eine nächste Wortgröße gar nicht zur Verfügung hat, weil man etwa unter Java auch mit *long* vorzeichenlos rechnen will?

Na, dann muss man sich eben etwas mehr Mühe geben. Aber das ist dann doch eine eigene Kolumne wert.

Referenzen

[1] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Vergleichsweise einfach,* KaffeeKlatsch, Jahrgang 3, Nr. 3, S. 21, Bookware, März 2010 http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2010-03.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.

Kaffeesatz

Wunderpille

von Lisa-Marie Wehlmann-Wiedeking

Das die Smartphones die Modeindustrie noch nicht vollkommen auf den Kopf gestellt haben, ist ein Wunder – in Anbetracht der Größe der *Multifunktionäre* müsste der Hosentasche eigentlich eine Wissenschaft gewidmet werden.

Wenn man mit der U-Bahn fährt und neben jemandem telefoniert, kann man ihn kaum noch sehen, ist doch die Hälfte des Gesichts mit dem kleinen Computer bedeckt.

Abgesehen von der Tatsache, dass sich immer mehr Menschen, die sich im selben Raum befinden, zur Kommunikation dem Handy zuwenden, sind die Smartphones natürlich nicht mehr – wie das olle Mobiltelefon aus der Steinzeit – nur zum Telefonieren da. Neben zahlreichen *Chats* und sozialen Netzwerken tummeln sich Millionen von anderen – bei näherer Betrachtung nutzlosen – Apps, die zur Unterhaltung dienen.

Aber wer stundenlang vor dem *Screen* abhängt, der bekommt schon mal ein bisschen Flimmern auf der Netzhaut und Kopfschmerzen sind auch nicht selten. Bitte lesen Sie die Packungsbeilage und fragen Sie Ihren Arzt oder Apotheker.

Die Lösung des Problems sind *Phablets*. Was klingt wie ein Trendprodukt, das die Pharmaindustrie ausgespuckt hat, ist zwar ein innovatives Produkt und auch *in*, aber nichts was speziell für die Gesundheit kreiert wurde. Die Pharmaindustrie ist natürlich hauptsächlich – das tut hier aber nichts zur Sache – auch auf Geld aus, und das Produkt, von dem hier die Rede ist, soll auch die Kassen klingeln lassen.

Es kann auch tatsächlich klingeln, so ein *Phablet* und mit der richtigen App kann es auch vermeintlich zur Gesundheit beitragen, Kalorien oder Schritte zählen, *chatten*, SMSen, telefonieren, surfen und Videos abspielen.

Ein *Phablet*. Das ist der *Missing Link* zwischen Phone und Tablet.

Auf die Idee sind kluge Geschäftsleute gekommen, die erkannt haben, dass viele Kunden Geld hinblättern, wenn auf der Packung das Wort größer steht.

Das *Phablet* verbindet gekonnt die Qualitäten eines Tablets – wie beispielsweise das augenschonende Schauen von Filmen und Musikvideos, das Erledigen der Arbeit wie am Laptop (was wiederum das Tablet vom PC geklaut und durch die Aufschrift *Touchscreen* salonfähig gemacht hat), das Checken von E-Mails und noch vieles mehr – mit den Qualitäten des Smartphones: Chatten, Spielen, Musik hören usw. Ein zu klein geratenes Tablet also, mit dem man wenigstens telefonieren kann.

Zwar schont der riesige *Screen* die Augen, aber unhandlich ist es doch ein wenig. Um gewöhnliche Smartphones kann sich ja schon nicht jede beliebige Hand herumlegen, und die *Phablets* benötigen meist zwei.

Vielleicht erfinden die Pharmaindustriellen ja ein Mittelchen, das die Hosentaschen wachsen lässt. Oder die *Phablet*-Hüllen werden demnächst mit Umhängetasche geliefert.

Kurzbiographie

Lisa-Marie Wehlmann-Wiedeking ist fast sechzehn Jahre alt und Zehntklässlerin. Wenn sie ein schönes Thema findet, das in den KaffeeKlatsch passt, schreibt sie sehr gerne Artikel. Sie hat sich immer noch vorgenommen zu bloggen (allerdings nicht zu Software-Entwicklungs-Themen) und möglichst bald eines ihrer Bücher fertig zu stellen.

Jahresinhaltsverzeichnis 2013

Artikel

Januar

Moderne Gesichter
Neues in JSF 2.2

von Rüdiger Keller 01/2013, Seite 6

Mit *JavaEE 7* wird auch die neue Version 2.2 von *JavaServer-Faces* erscheinen. Dieser Artikel stellt zwei der größeren Neuerungen vor: Unterstützung für *HTML 5* und *Faces Flows*.

Lichtjahre voraus
Einfach Domänenspezifische Sprachen (DSL's) entwerfen mit JParsec

von Róbert Bräutigam 01/2013, Seite 9

Die funktionale Welt benutzt *Parser*-Kombinatoren schon seit einer Weile. Zum Glück hat die Java-Welt funktionale Entwicklung wieder entdeckt, und damit stehen mehr und mehr dieselben funktionalen Werkzeuge auch Java-Entwicklern zur Verfügung. *JParsec* ist ein solches Werkzeug, obwohl es ohne *Closures* ein bisschen umständlich ist.

Februar

Zweites Date mit Nancy
und Web-Entwicklung mit .NET macht wieder Spaß

von Timothée Bourguignon 02/2013, Seite 8

Sagen wir mal, dass wir eine *Web-API* für eine bestehende Datenbank bauen wollen – und zwar schnell – und dass wir dafür .NET verwenden möchten. Wir könnten *ASP.NET* (*MVC3* oder *MVC4 Web API*) oder lieber ein Lightweight Framework wie *Nancy* verwenden. Im letzten Artikel wurde gezeigt wie man *Nancy*, *Simple.Data* und *MongoDB* kombinieren, konfigurieren und miteinander verbinden kann. Dabei konnte man sehen, wie *Nancy* auf eine bestimmte Route reagiert, einen *View* anzeigt und *View*-Daten von und zum *View* übermittelt. Ebenso wurde gezeigt wie man *Simple.Data* verwendet, um Daten zu schreiben bzw. von einer Datenbank zu lesen. Dazu wurde *MongoDB* genutzt und demonstriert, wie über die *Interactive Shell* Objekte hinzugefügt und gelesen werden können. Das Ganze innerhalb von zehn C#- und kaum mehr *cshtml*-Zeilen. Das war ein erster Blick auf den „Super-Duper-Happy-Pfad". Ich nehme an, dass wir diese Kenntnisse anwenden wollen!

Der Typ ist sicher!
Constraint Code Generator für Java

von Heiner Kücker 02/2013, Seite 14

Moderne Programmiersprachen wie *Haskell* oder *Scala* beeindrucken mit hochentwickelten Typ-Systemen und versprechen die Vermeidung von Fehlern bereits zur Kompilierzeit. Mit einem Code-Generator ist manches davon auch mit dem (guten) alten Java möglich.

März

Web unchained!
Barrierefreie Web-Entwicklung

von William Siakam 03/2013, Seite 6

Seit Ende 2005 sind deutsche Behörden durch §11 Abs. 1 des Behindertengleichstellungsgesetzes verpflichtet, ihre Internetauftritte und Angebote barrierefrei zu gestalten. Nach Schätzung der Weltgesundheitsorganisation (WHO) leben ungefähr 1,2 Millionen blinde und sehbehinderte Personen in Deutschland. Obwohl Barrierefreiheit für die Meisten kein Fremdwort ist, stellt man immer wieder fest, dass viele unserer Produkte, u. a. Web-Seiten, implizite Zäune für Blinde und Sehbehinderte enthalten. Worauf Sie beim Entwickeln Ihrer Web-Seite achten können, um sie zugänglich zu machen, erfahren Sie in diesem Artikel.

April

Verflixte Sieben?
Ein Blick auf den JBoss AS 7 – Von Domänen und Modulen

von Werner Eberling 04/2013, Seite 6

In regelmäßigen Abständen scheint *JBoss* (seit einigen Jahren ein Teil von *RedHat*) die Notwendigkeit zu sehen, den gleichnamigen *Application-Server* auf technisch neue Beine zu stellen. Mit Version 7 stand wieder ein solcher *Re-Write* ins Haus. Einige der Neuerungen, die diese neue Version mit sich bringt, sollen in diesem Artikel beleuchtet werden.

Ist Amazon wirklich an allem schuld?
Ein Kommentar

von Prof. Dr. Svenja Hagenhoff 04/2013, Seite 11

Schon im Vorfeld der Leipziger Buchmesse wurde in der Presse erneut verstärkt über das Dilemma des stationären Buchhandels berichtet: rückläufige Umsätze, schließende Ladengeschäfte und kleiner werdende Verkaufsflächen sind die beklagten Phänomene.

Frisch gemixt
Unit-Tests mit Mockito und PowerMock

von Rustam Khakimov 04/2013, Seite 13

Schreiben Sie *Unit-Tests* oder ist Ihre Anwendung dafür zu komplex, weil viel zu viele Abhängigkeiten zu anderen Komponenten bzw. Systemen bestehen? Wie diese Abhängigkeiten zur Testlaufzeit aufgelöst werden können, erfahren Sie in diesem Artikel.

Mai

Warum?
reicht manchmal nicht!

von Andreas Heiduk 05/2013, Seite 6

Fragen ist eigentlich ganz einfach. Im Zweifel reicht ein einfaches „Warum?“. Möchte man aber eine sinnvolle und brauchbare Antwort, so wird es komplizierter.

Juni

Simsalabim!
Pragmatische Code-Generierung mit Ant und Velocity am Beispiel eines Android-Projekts

von Frank Ganske 06/2013, Seite 6

Das Thema Code-Generierung ist nicht neu, sondern hilft Entwicklern seit *yacc*, *bison* oder in Java *antlr*, *sablecc* usw. gleichförmigen Code zu erstellen, ohne weiter darüber nachdenken zu müssen. Man investiert Arbeit in die Vorbereitung und erhält Flexibilität und Geschwindigkeit als Ergebnis. So das Ziel. Jedes komplexe Problem kann ja durch Einführung einer zusätzlichen Indirektionsschicht gelöst werden – solange diese Schichten nicht zum komplexen Problem werden. Ansätze wie *Model Driven Architecture* haben das Konzept auf die Spitze getrieben. Oft wurde versucht, weitgehend alles zu generalisieren. Das wird leicht wieder zum Hemmschuh.

Nicht nur Spinnen bauen Netze
Web-Entwickung für Java-Entwickler – Teil 2

von Frank Goraus 06/2013, Seite 10

Jeder fängt mal klein an. Und so kann einen die Vielfalt an Technologien in der Web-Welt förmlich erschlagen. Hinzu kommt noch, dass man bei Web-Anwendungen teils anders an Probleme herangehen muss als dies bei *Desktop-/Rich Client*-Anwendungen der Fall ist. Im Folgenden soll es um einige der fortgeschritteneren Basis-Technologien gehen, welche man zur Umsetzung einer Web-Anwendung verwenden kann.

Juli

Unbeabsichtigt monadisch
Monaden entwickeln in Java

von Róbert Bräutigam 07/2013, Seite 6

Funktionale Entwicklung ist wie die dunkle Seite der Macht. Begibt man sich einmal auf diesen Pfad, kontrolliert sie die Gedanken des Entwicklers für immer, scheinbar auch wenn es um Java-Code geht. Folgendes beruht auf einer wahren Begebenheit.

August

Dynamisch und gefährlich?
C# Dynamics in freier Wildbahn

von Timothée Bourguignon 08/2013, Seite 7

Was passiert genau, wenn das C#-Schlüsselwort *dynamic* verwendet wird? Manche sagen, dass die Hölle ausbrechen wird. Andere, dass für jede Verwendung ein Kätzchen stirbt – und wir sprechen von echt süßen Kätzchen! Bei dessen Einführung in C# 4.0 im Jahr 2010 hat sich der Autor bemüht, die Kätzchen zu retten, also keine *dynamics* zu verwenden. Aber das gilt seit 2012 nicht mehr. Wenn Kätzchen wirklich in Gefahr wären, wenn man *dynamics* verwendet, würde er nämlich bald Ärger mit Tierrechtsorganisationen bekommen.

Höher, schneller, weiter
Erweiterungen für CDI entwickeln
von Werner Eberling 08/2013, Seite 40

Der inzwischen nicht mehr ganz so neue *CDI*-Standard kommt mittlerweile in immer mehr Projekten zum Einsatz. Doch was tun, wenn die *out-of-the-box*-Funktionalität des „Neuen“ nicht ausreicht? Für solche Situationen bietet *CDI* einen eingebauten Erweiterungsmechanismus, der dem Entwickler interessante Möglichkeiten zum Ausbau des Frameworks bietet. Wie dies vonstatten geht, soll in diesem Artikel anhand zweier Beispiele genauer betrachtet werden.

September

Himbeerkuchen
Faszination Einplatinenrechner – Vorstellung des Raspberry Pi
von Sascha Gross 09/2013, Seite 6

Das neue Sternchen am *Hardware*-Himmel ist nicht ein Superrechner oder ein plattes Tablet, sondern ein unscheinbarer erschwinglicher Einplatinenrechner, der *Raspberry Pi* mit der Grundfläche einer Kreditkarte. Von seinen Machern gedacht um Kindern die Programmierung nahezubringen, erfreut er sich in der Bastler- und Entwicklergemeinde großer Beliebtheit. Mit 25 $ für Model A beziehungsweise 35 $ für das Model B, mit der besseren Ausstattung, kann der Spaß beginnen.

Oktober

Moderne Gesichter (Teil 2)
Weitere neue Features in JSF 2.2
von William Siakam 10/2013, Seite 6

Die *JavaEE 7*, die im Juni diesen Jahres released wurde, enthält die *Java Server Faces*-Spezifikation in der Version 2.2. Im Frühjahr wurden zwei bedeutsame Neuerungen in einem Artikel vorgestellt. Nun möchten wir uns weitere Features ansehen, die diese neue Version mit sich bringt.

November

Vom Duke zum Droid
Lehren eines Android-Anfängers
von Andreas Heiduk 11/2013, Seite 6

Wenn man für *Android* entwickeln möchte, gibt es im Netz genügend gute Tutorials, in denen die APIs und deren Verwendung sehr gut erklärt werden. Einem Umsteiger von *JSF*, *Swing* oder *SWT* wird aber leider nicht verraten, welche „höheren“ Design-Prinzipien auf der neuen Plattform effizient sind und auf welche man nicht mehr bauen sollte. In diesem Artikel werden einige dieser „Fallen“ an einer Beispiel-App erläutert.

Getaktete Passwörter
von Wolfgang Wolf 11/2013, Seite 10

Trotz neuer Entwicklungen, wie die Auswertung biometrischer Merkmale (Fingerabdruck, Handflächen- oder Irisscan), ist das Passwortverfahren gegenwärtig noch immer die gängigste Authentifizierungsmethode. Aus kryptoanalytischer Sicht hängt die Sicherheit eines Passwortes von dessen Länge und von der Anzahl verfügbarer Zeichen ab. *Server*-seitig kann das Passwort gesalzen werden, um so die Sicherheit zu erhöhen. Aber auch *client*-seitig gibt es noch bisher ungenutzte (getaktete) Spielräume zur Verbesserung der Sicherheit.

Dezember

Wenn Murphy mal wieder Recht behält
Datenbankskripte übergabesicher gestalten
von Marc Spanagel 12/2013, Seite 7

Murphy erfuhr es schon früh: „Wenn es mehrere Möglichkeiten gibt, eine Aufgabe zu erledigen, und eine davon in einer Katastrophe endet oder sonstwie unerwünschte Konsequenzen nach sich zieht, dann wird es jemand genau so machen.“

Dies gilt insbesondere bei der Übergabe eines, oder schlimmer, mehrerer *SQL*-Skripte in die Produktion. In diesem Artikel soll gezeigt werden, wie man diese Gefahren minimieren kann.

Nicht nur Spinnen bauen Netze
Web-Entwickung für Java-Entwickler – Teil 3
von Frank Goraus 12/2013, Seite 12

Jeder fängt mal klein an. Und so kann einen die Vielfalt an Technologien in der Web-Welt förmlich erschlagen. Hinzu kommt noch, dass man bei Web-Anwendungen teils anders an Probleme herangehen muss als dies bei *Desktop-/Rich-Client*-Anwendungen der Fall ist. Im Folgenden soll es um *Sessions* und die *Servlet*-Filter gehen. Am Ende gibt es eine Beispielanwendung, die selbst implementiert werden kann und anhand derer aufgezeigt wird, wie man beide Techniken einbinden kann.

Kolumnen

Des Programmierers kleine Vergnügen

Altlasten entsorgen

von Michael Wiedeking 01/2013, Seite 15

Man sagt ja, dass wenn der Keller vollsteht, man einfach alles wegwerfen soll, was man länger als ein Jahr lang nicht mehr gebraucht hat, weil man es eh nie wieder brauchen wird. Das ist in der Informatik nicht anders: hat man nur einen begrenzten Platz zur Verfügung, so ist man gut bedient das Teil zu entsorgen, das man am längsten nicht mehr benutzt hat, weil auch hier wahrscheinlich die selbe Regel gilt, wie für das Gerümpel im Keller. Aber wie kann man so etwas elegant implementieren?

Bloomige Angelegenheit

von Michael Wiedeking 02/2013, Seite 22

Bit-Mengen scheinen immer eine gute Idee zu sein, wenn die abzubildende Menge entsprechend überschaubar ist. Wird aber die zu verwaltende Menge zu groß, würde auch die Bit-Menge zu groß werden. Abhilfe schafft hier die Idee des Herrn Bloom, anstatt für jedes Element je ein Bit zu verwenden, die einzelnen Bits geschickt mehrfach zu vergeben.

Zum Weglaufen

von Michael Wiedeking 03/2013, Seite 12

Gelegentlich möchte man etwas komprimieren, Bilder beispielsweise. Damit haben sich natürlich schon viele kluge Köpfe beschäftigt, aber deren Resultate sind gleichermaßen effizient wie kompliziert. Unter Umständen geht es aber auch ganz einfach.

Über Mandelbrote, Apfelmänner, Elefanten und Seepferdchen

von Michael Wiedeking 04/2013, Seite 19

Eigentlich ist alles ganz einfach. Da hat ein Herr Mandelbrot festgestellt, dass sich die Folge $z_{n+1} = z_n^2 + c$ mit dem Startwert $z_0 = c$ sehr interessant verhält, wenn c eine komplexe Zahl ist. Nun sind komplexe Zahlen Paare reeller Zahlen. Und was kann man mit Zahlenpaaren wunderbares machen? Man kann sie schön darstellen.

Pappsatt

von Michael Wiedeking 05/2013, Seite 8

In der Bildbearbeitung tritt gelegentlich das Problem auf, dass man ein Bild aufhellen will. Das lässt sich eigentlich recht leicht bewerkstelligen, indem man auf sämtliche Intensitäten eines Bildpunktes etwas addiert. Damit wäre das Problem auch schon gelöst, wenn es nicht die Überläufe gäbe, durch die ein Punkt nicht mehr heller sondern ganz dunkel erscheinen würde. Abhilfe schafft hier die Sättigungsarithmetik, bei der ein genutzter Zahlenraum niemals unter- oder überschritten wird.

Alle meine Teilmengen

von Michael Wiedeking 06/2013, Seite 15

Kleine Mengen können sehr effizient als Bit-Sequenzen implementiert werden, denn viele der typischen Mengenoperationen lassen sich dann in nur einer Instruktion umsetzen. Gelegentlich möchte man aber auch über alle Teilmengen einer Menge iterieren. Und hier zeigt sich wieder, wie gut es ist, wenn man sich mit Bit-Frickeleien auskennt.

Weiter so!

von Michael Wiedeking 07/2013, Seite 9

Gelegentlich hat man es mit Daten zu tun die sowohl paarweise als auch allein auftreten. Speichert man diese etwa in einem *Array* ab, so ergibt sich das Problem, dass man zwar die Stelle I bearbeitet, aber das nächste Element an einer geraden oder ungeraden Stelle liegen muss. Wie aber findet man effizient zum nächsten geraden oder ungeraden Index?

Dünn besät

von Michael Wiedeking 08/2013, Seite 44

Gelegentlich hat man das Problem, dass ein *Array* nur verhältnismäßig wenige Einträge hat. Dieses Problem lässt sich etwa – wie schon in früheren Vergnügen gesehen – teilweise mit einem *Bit-Array* lösen, womit zumindest geklärt ist, ob ein Element überhaupt vorhanden ist. Jetzt stellt sich nur die Frage, wie man das dazugehörige Element effizient finden kann.

Grüppchenzählung

von Michael Wiedeking 09/2013, Seite 9

Hat man sich einmal dafür entschieden Informationen in einer Bit-Sequenz unterzubringen, so besteht manchmal der Bedarf zusammenhängende Einser-Gruppen zu zählen. Selbstverständlich kann man dies einfach mit Hilfe einer Schleife bewerkstelligen, aber es geht auch verzweigungsfrei – ganz ohne Bedingungen.

Gruppenzwang

von Michael Wiedeking 10/2013, Seite 9

Zusammenhängende Einsergruppen zu zählen, hat sich im letzten Vergnügen als nicht all zu schwierig herausgestellt. Gelegentlich interessieren einen aber nur zusammenhängende Gruppen mit einer bestimmten Mindestgröße.

Der allerletzte Wächter

von Michael Wiedeking 11/2013, Seite 14

Wenn der Bedarf einmal besteht, kann eine Instruktion mehr oder weniger erfolgsentscheidend sein. Deshalb ist es gelegentlich sinnvoll auf Bedingungen zu verzichten, wenn man im Gegenzug sicherstellt, dass eine Andere sicher eintritt.

Apropos Wächter

von Michael Wiedeking 12/2013, Seite 19

Das Prinzip des im letzten Vergnügen vorgestellten Wächters kann natürlich nach Belieben erweitert werden. Mein persönlicher, ungeschlagener Favorit ist dabei aber der Einsatz in einer verketteten Liste.

Deutsch für Informatiker

Zitate

von Alexandra Specht 01/2013, Seite 17

Wie wir in den letzten Monaten erfahren haben, kann man eine Menge falsch machen, wenn man etwas aus den Werken anderer verwendet. Falls Sie selbst zitieren möchten, hilft Ihnen dieser Artikel wenigstens dabei, die Satzzeichen richtig zu verwenden.

Am bestensten gemeint und doppeltgemoppelt

von Alexandra Specht 02/2013, Seite 24

Wir erhielten doch diesen unterhaltsamen Leserbrief von Herrn Förster, den wir Ihnen letzten Oktober abgedruckt haben. Darin ging es um den „bestbewertesten aller vergangenen Herbstcampusse“. Nachdem es bei der Steigerung von Adjektiven und Adverbien und auch bei anderen Stilmitteln, die der Steigerung des Gesagten dienen sollen, bemerkenswerte Tatsachen gibt, heute ein Artikel über ebensolche Steigerungen.

Null Komma Nichts

von Michael Wiedeking 03/2013, Seite 14

Eigentlich hat man es relativ oft mit Zahlen zu tun: Festplattengröße, Taktfrequenz, Mitarbeiter, Jahresumsatz et cetera. Aber in den seltensten Fällen muss man solche Zahlen ausschreiben. Was ist aber, wenn man sie doch einmal ausschreiben muss? Wie schreibt man eigentlich (insbesondere große) Zahlen?

Vereinheitlichte Differenzierung

von Michael Wiedeking 04/2013, Seite 24

Als die Rechner noch klein und überschaubar waren, war alles kein Problem. So glaubte man, dass es für mehr als fünf Großrechner weltweit keinen Bedarf gäbe und Niemand mehr als ein Megabyte bräuchte. Heute weiß man es besser, und die Datenmengen sind so astronomisch groß, dass man sich auch Gedanken darüber machen muss, wie man diese Beträge darstellt. Dabei ergibt sich aber ein neues Problem: Besteht ein Kilobyte nun aus 1000 oder 1024 Byte?

Lehnen

von Alexandra Specht 05/2013, Seite 10

Neulich saß ich in der U-Bahn in Berlin und habe nicht nur die Leute betrachtet, sondern auch ein paar Zeilen des Buches meiner unbekannten Sitznachbarin gelesen. Ich laß: “...sein Kopf lehnte gegen das Fenster...“. Hmm, das klingt aber komisch, dachte ich mir. Daraufhin wollte ich hier in dieser Kolumne etwas über reflexive Verben schreiben.

Sammelsurium III

von Alexandra Specht 06/2013, Seite 17

Ob ich einen halb garen oder halbgaren Artikel in dieser Ausgabe abgeliefert habe oder ob er wenigstens lesenswert ist, können Sie nach der Lektüre desselben hoffentlich – positiv natürlich – beantworten. Ebenso, ob er hoch interessant oder hochinteressant ist.

me <3 #omnom

von Golo Roden 07/2013, Seite 11

Geeks und Nerds haben eine eigene Sprache, eigenen Humor – letztlich eine ganz eigene Subkultur. Diese ist von Wortspielereien, Katzenfotos und anderen auf Außenstehende seltsam anmutenden Dingen geprägt. Dies ist der Versuch einer Erklärung, und ein Plädoyer für mehr Toleranz.

Redewendungen

von Alexandra Specht 08/2013, Seite 46

August ist, zumindest hier in Bayern respektive Franken, die Haupturlaubszeit. Die Straßen sind erfreulich frei am Morgen und man findet genügend freie Parkplätze in Wohngebieten. Sicher sind auch von unseren Leserinnen und Lesern viele gerade im Urlaub. Und vielleicht auch im Ausland. Für diesen Fall oder einen späteren Urlaub möchte ich Ihnen nun ein paar nützliche Redewendungen an die Hand geben, mit denen Sie, wenn alles gut läuft, Einheimische beeindrucken können.

Eine Hommage an den Babel Fish

von Michael Wiedeking 09/2013, Seite 8

Leser von Douglas Adams' *Per Anhalter durch die Galaxis* wissen, dass der Babel Fish ein sehr nützliches Tierchen ist, welches, einmal in den Hörgang eingeführt, dem Träger ermöglicht, sämtliche gesprochenen Sprachen zu verstehen. Da es sich dabei – leider – um ein rein fiktives Lebewesen handelt, muss natürlich technisch Abhilfe geschaffen werden.

Über Larts, Ingrid und das Heisen

von Michael Wiedeking 10/2013, Seite 10

Das bemerkenswerteste an Sprache ist sicherlich ihre Anpassungsfähigkeit. Diesbezüglich leistet ganz besonders die Jugend ihren Beitrag. Und weil es immer wieder eine neue Jugend mit einem eigenem, neuen Wortschatz gibt, hat der Duden jetzt Abhilfe geschaffen.

Reise nach Engadin

von Michael Wiedeking 11/2013, Seite 15

Deutsch ist selbstverständlich meine Lieblingssprache. Aber wenn es um das Thema Internationalisierung und Lokalisierung geht, verliert Deutsch ganz klar an Punkten. Und das lässt sich relativ leicht begründen.

Kaffeesatz

Batteries not included

von Lisa-Marie Wehlmann-Wiedeking 01/2013, Seite 18

Wenn man sich etwas Teureres zu Weihnachten wünscht, muss man entweder den Wunschzettel kürzer ausfallen lassen oder Andere überzeugen, dass sie das Gewünschte auch benötigen. Wenn man ein Einzelkind ist, hat man zwar keine Geschwister zur Verfügung, aber in diesem speziellen Fall konnte der Vater aushelfen. Ein Nintendo 3DS!

Machtworte

von Michael Wiedeking 02/2013, Seite 26

Es ist schon unglaublich, was man mit Hilfe des Internets alles erreichen kann. Und dazu bedarf es – wie in jüngster Vergangenheit noch nicht einmal einer großen Gruppe. So schafft es heutzutage schon ein einzelner Mensch, ohne das Haus zu verlassen, beliebig großen Schaden anzurichten.

Ich weiß genau, wo du bist

von Andreas Schubert 03/2013, Seite 15

Ich gebe es zu, ich bestelle gerne im Internet. Nicht nur, da ich dann zu jeder Tages- und Nachtzeit einkaufen kann, sondern auch, weil ich die Sendungen bequem dahin geliefert bekomme, wo ich sie gerne hätte. In meinem Fall ist das meist nicht meine Wohnung, da ich während der üblichen Zustellzeiten der Paketdienste selten zu Hause bin, sondern die Arbeit.

Analoges Rechtemanagement

von Michael Wiedeking 04/2013, Seite 25

In der letzten Zeit liest man immer wieder darüber, was wohl wäre, wenn es sich bei dem Buch um ein neues Medium handeln würde. Da wird dann über die Vorzüge dieser neuen Erfindung geschwärmt, allerdings auch auf deren Gefahr hingewiesen. Vorzüge hin, Gefahren her: Das Buch gibt es jetzt tatsächlich neu erfunden – artgerecht für das digitale Zeitalter.

Ein ganz heißer Tipp

von Michael Wiedeking 05/2013, Seite 12

Neulich haben wir Freunde in der Fremde besucht. Eigentlich wollten wir diese hauptsächlich deswegen besuchen, damit wir uns mal das neue Haus ansehen können. Obwohl der Fertigstellungstermin zur Zeit unseres Besuchs zwei Monate zurückliegen sollte, mussten wir bei unserer Ankunft feststellen, dass das Haus gerade erst fertiggestellt worden war. Das mag vielleicht daran gelegen haben, dass es sich bei dem Hausbesitzer um einen Software-Entwickler handelt.

Stubenrein?

von Lisa-Marie Wehlmann-Wiedeking 06/2013, Seite 18

Streicheln, füttern, gut zureden – das sind Dinge, die man mit einem Haustier machen kann. Oder mit einem Handy, wenn man es gut genug leiden kann.

Entschleunigung

von Michael Wiedeking 07/2013, Seite 13

Mit der Einführung des Internets sind bei der Kommunikation Geschwindigkeiten erreicht worden, die ihresgleichen suchen. Wobei sich natürlich die Frage stellt, ob dies wirklich immer für den Inhalt – quantitativ wie qualitativ – förderlich ist. Und, ob dabei nicht auch etwas auf der Strecke bleibt.

Altmodisch bewährt sich

von Lisa-Marie Wehlmann-Wiedeking 08/2013, Seite 47

Bei technischen Geräten weiß man ja nie. Fehlermeldungen, Verschicken ganzer Adressbücher und mehr oder weniger abhörbare Korrespondenzen. Es scheint als hätte es früher weniger Überfälle auf Kutschen mit versiegelten Staatsangelegenheitsbriefen gegeben, als es heute Hacker gibt.

Der Weg nach Hause – wenn das Navi nicht mehr weiterhilft

Über ein Gewitter, fehlende Umleitungen und was passiert, wenn die Technik uns im Stich lässt

von Kerstin Wolf 09/2013, Seite 11

Neulich in einem beschaulichen Ort in Bayern (zufälligerweise dem Hauptsitz meines Arbeitgebers)... Gegen 18 Uhr kommt an einem heißen und sonnigen Tag auf einmal Wind auf, der erste Vorbote des angekündigten Gewittersturms. Ich raffe meine Sachen zusammen, und nachdem ich glücklicherweise auf dem Parkplatz mein Auto wiedergefunden habe, geht es bei beginnendem Regen in Richtung Heimat. Ich habe es eilig, denn ich würde gerne noch halbwegs trockenen Fußes die Wohnung erreichen – was sogar realistisch wäre, denn ich wohne ja nur 18 Minuten von der Arbeit entfernt, und über meinem Wohnort sieht es derzeit sogar noch einigermaßen hell aus.

Ausgetrickst

von Michael Wiedeking 10/2013, Seite 11

Als Alan Turing 1950 vorschlug einen Test zu entwickeln, um die Intelligenz eines Computers mit der eines Menschen zu vergleichen, konnte er nicht ahnen, dass schon 2013 die Maschine dabei besser abschneiden würde.

Suchmaschine sei Dank

von Michael Wiedeking 11/2013, Seite 16

Mit der Zeit gewöhnt man sich ja an ziemlich viel. Und oft verinnerlicht man diese Dinge so sehr, dass man sich gar nicht mehr richtig daran erinnern kann, wie es denn früher einmal gewesen ist.

Ping

von Michael Wiedeking 12/2013, Seite 21

Sendet man ein Signal aus, dass irgendwo reflektiert wird, so kann man aus dem, was zurückkommt, Rückschlüsse ziehen, die beispielsweise Aufschluss über Ort oder Entfernung geben können. Jetzt macht man sich dieses Verfahren auch im Rahmen der SEPA-Umstellung zu nutze.

Fehlerteufelchen

Konfiguration wo bist du? Gerade warst du noch da!

von Sascha Gross 03/2013, Seite 9

Konfigurationen für Anwendungen gehören zum A und O der Entwicklung von Software. Nur dumm wenn sie „einfach“ verschwinden. Fast jede Anwendung braucht irgendeine Art von Konfiguration, sei es für die Anpassung an die Umgebung, für das Freischalten/Sperren von Funktionalität oder für benutzerspezifische Einstellungen, um ein paar Anwendungsfälle aufzuzeigen. Im einfachsten Fall kann die Konfiguration auf Schlüsselwert-Paare (*Properties*) abgebildet werden, die im Dateisystem hinterlegt sind. Im etwas fortgeschrittenen Fall liegen diese in einer Datenbank und können über eine Administrationsoberfläche geändert werden, die im Folgenden eine Web-Oberfläche sein wird.

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 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
Tieto Deutschland GmbH
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
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

Einführung in die objektorientierte Programmiersprache Java

Eine praxisnahe Einführung
10. – 14. Februar 2014, 30. Juni – 4. Juli 2014,
2.150,– € (zzgl. 19 % MwSt.)

Einführung in C# und .NET

Einstieg in C# und die .NET-Plattform für Programmieranfänger
10. – 14. März 2014, 21. – 25. Juli 2014,
2.150,– € (zzgl. 19 % MwSt.)

Entwicklung mobiler Anwendungen mit iOS

7. – 9. April 2014, 15. – 17. September 2014,
1.250,– € (zzgl. 19 % MwSt.)

HTML5, CSS3 und JavaScript

31. März – 3. April 2014, 22. – 25. September 2014,
1.650,– € (zzgl. 19 % MwSt.)

Fortgeschrittenes Programmieren mit Java

Ausgewählte Pakete der Java Standard Edition (J2SE)
5. – 7. Mai 2014, 13. – 15. Oktober 2014,
1.350,– € (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

„Die Herausforderung, jeden Tag etwas Neues zu lernen, habe ich gesucht und bei MATHEMA gefunden."

Tim Bourguignon, 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

Herbstcampus

Wissenstransfer par excellence
1. – 4. September 2014, Nürnberg

Beitragsaufruf

(Call for Papers)

Der Herbstcampus ist eine technologie-orientierte Konferenz, die Software-Entwicklern, Architekten und Projektleitern eine konkrete Hilfe bieten soll. Schwerpunkte sind .NET und Java.

Interessante Themen sollen in verschiedenen Darbietungen behandelt werden: als Einführung, Erfahrungsbericht oder problemlösender Vortrag.

Haben Sie ein passendes Thema oder Interesse daran, einen Vortrag zu halten? Schicken Sie uns Ihre Vorschläge:
beitragsaufruf@herbstcampus.de

Haben Sie noch Fragen? Dann schreiben Sie uns eine E-Mail.
info@herbstcampus.de

Den vollständigen Beitragsaufruf und weitere Informationen zum Herbstcampus finden Sie unter
www.herbstcampus.de

Das Allerletzte

Sehr geehrte Web-Besucherin, sehr geehrter Web-Besucher!

null

Wenn Sie Fragen zu dieser Fehlermeldung haben, speichern Sie bitte diese Fehlerseite und schicken Sie diese als Anhang in einer E-Mail mit der Fehlerbeschreibung an den Webmaster (support@xyz.de) .

Dies ist kein Scherz!
Diese Fehlermeldung 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 Februar.

Herbstcampus

Wissenstransfer par excellence

1. – 4. September 2014

in Nürnberg