KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

11/2012

Jahrgang 5

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

Hätten Sie's gewusst?

Was muss man als Software-Entwickler eigentlich alles wissen? Bei der Frage geht es nicht um die Dinge, die mit der täglichen Arbeit zu tun haben, sondern um diejenigen Sachen, die darüber hinaus präsent sein sollten – die Grundlagen sozusagen. Aber was ist denn überhaupt unsere Wissensbasis?

Irgendwie hat man doch immer wieder mit Datenansammlungen zu tun: Listen, Mengen, Maps und so weiter. Jetzt sind die in der Regel aber abstrakt definiert, so dass man die Implementierung austauschen kann. Das bedeutet natürlich auch, dass ich die Implementierungen und deren Vor- und Nachteile kennen muss. Und hier geht es schon los: Muss man die Vor- und Nachteile wirklich kennen?

Bei einer Liste könnte ich immer eine auf Arrays basierende Implementierung wählen und ich mache da kaum etwas falsch – zumindest nichts Offensichtliches. Damit muss ich auch nicht wissen, welche Eigenschaften diese Implementierung etwa beim Einfügen, Durchsuchen, Lesen und Löschen an den Tag legt. Bei Tests mit ausreichend geringer Elementzahl legen die verschiedenen Implementierungen auch kein merklich unterschiedliches Verhalten an den Tag.

Muss ich also wissen, wie das asymptotische Verhalten der verschiedenen Implementierungen ist? Muss ich die Groß-O-Notation kennen? Muss ich wissen, was der Unterschied zwischen $O(n)$, $O(n \log n)$ und $O(n^2)$ ist, oder gar $O(2^n)$ kennen? Muss ich außer den in der Standardbibliothek vorhandenen Implementierungen noch andere kennen, damit ich nach denen im Fall der Fälle suchen kann? Muss ich wissen, was eine Skip-List ist und muss ich sie gar implementieren können?

Wenn Sie von einem Teilausdruck wissen müssen, ob er negativ werden kann, reicht dann ein bisschen Intuition oder Probieren aus oder müssen Sie das beweisen können? Und muss man wissen, wann Probieren nicht mehr ausreicht? Wenn man einen Algorithmus auf Basis von 16-Bit-Wörtern mit allen möglichen Kombinationen erfolgreich getestet hat, funktioniert der auch fehlerfrei für eine 32- oder 64-Bit-Variante?

Muss ich mich unter Java mit Swing auskennen, wenn ich nur Server-Anwendungen programmiere? Muss ich unter .NET die Windows Communication Foundation (WCF) kennen, wenn ich nur Oberflächen programmiere? Muss ich TeX kennen? Muss ich mehr als ein Web-Framework kennen? Muss ich eine zweite Programmiersprache beherrschen? Muss ich Design-Patterns kennen? Muss ich wirklich wissen, was eine Äquivalenzrelation ist?

All diese Fragen, die sich nur mit einem Bruchteil dessen beschäftigen, mit dem wir es täglich zu tun haben könnten, lassen sich ohne Kontext wahrscheinlich nicht beantworten. Aber irgendwie müsste es doch ein Minimum geben. Mir würde eine Liste gefallen, auf der ich mein vorhandenes Wissen abhaken und mir Ideen über fehlendes Wissen holen könnte.

Aber vielleicht genügt ja auch die Fähigkeit, zu wissen, wo und wie man etwas sucht, findet und nachlesen kann – zumindest wenn man genügend Zeit hat.

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 Alexandra Specht via E-Mail unter alexandra.specht@bookware.de oder telefonisch unter 09131/8903-14.

Copyright

Impressum

KaffeeKlatsch Jahrgang 5, Nummer 11, November 2012
ISSN 1865-682X
Bookware – eine Initiative der MATHEMA Software GmbH

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

Klettergarten

Common Table Expressions und rekursive Abfragen in SQL 8
von Andreas Heiduk

Selbst in einfachen Schemata verstecken sich sehr oft einige hierarchische Strukturen wie Schritt, Unterschritt und Teilschritt. Ebenso oft werden diese mit *SQL* „platt geklopft“ und ausprogrammiert. Dabei geht es auch anders: Flexibel!

Web-Anwendungen auf Klick

Einführung in Apache Click 12
von Rustam Khakimov

Kein *MVC* und *JSP* sondern ein seitenorientiertes Design und „natürliche“ *Rich-Client*-Programmierungsweise bei der Entwicklung von Web-Anwendungen. *Apache Click* bietet hierfür eine interessante Lösung. Dieser Artikel stellt das Framework vor und zeigt anhand von zahlreichen Beispielen Entwicklung und Test einer Web-Applikation.

Fakultativ

Des Programmierers kleine Vergnügen 18
von Michael Wiedeking

Die Fakultätsfunktion läuft einem im Rahmen der Ausbildung eigentlich immer über den Weg. Sie scheint sich dazu zu eignen, einem sowohl die Rekursion als auch die Iteration näher zu bringen. Aber leider hinterlässt sie dann den Eindruck, dass es getan wäre und man diese so programmieren müsse. Aber weit gefehlt.

Leserbriefe

Leserbrief

von Lars Hupel

bezogen auf

Var-um? – Über Sinn und Unsinn variabler Typen in C#
von Tobias Krügel, KaffeeKlatsch 2012/10

Hallo,

die Lektüre des Artikels lies mich verdutzt zurück. Zunächst einmal möchte ich darauf hinweisen, dass die Bezeichnung von *'var'* als „variablen Typen" irreführend ist. *'var'* kennzeichnet nämlich gar keinen Typen. *'var'* signalisiert, wie auch später korrekt beschrieben wurde, dem Compiler, dass der Typ automatisch eingesetzt werden soll.

Dieses Verhalten ist naheliegend, denn bei der Definition einer lokalen Variable *'var x = foo()'* liegen dem Compiler sämtliche Informationen vor, um den Typen von '*x*' zu bestimmen.

Alles andere, nämlich statt *'var'* einen konkreten Typen zu notieren, wäre eine Verletzung des bekannten und bewährten Prinzips „Don't Repeat Yourself" (DRY). Die Sprache macht es – äußerst limitiert zwar – möglich, also sollte man es auch nutzen.

Es gibt auch Sprachen, die tun das von Anfang an so. Als Beispiel sei *Scala* genannt. Dort ist es der Normalfall, bei lokalen Variablen die Typannotation wegzulassen. Die explizite Notation sieht hingegen so aus: *'val x: Foo = …'*

Ich habe mir mal einen größeren Korpus von Scala-Code genommen, hier *Akka* (ca. 73 Tsd. Zeilen Code), und stupide nach diesem Muster gegreppt.

```
akka (master)$ git grep -E 'va[lr] [a-zA-Z]*: .* =' | wc -l
608
akka (master)$ git grep -E 'va[lr] [a-zA-Z]* =' | wc -l
4457
```

Insgesamt werden also knapp 90 % aller Variablen ohne Typannotation definiert. (Dieser Wert ist etwas zu niedrig, da hier nicht nur lokale sondern auch Klassenvariablen mitgezählt werden, bei denen es generell üblich ist, die Typen explizit auszuschreiben.)

Und das sind nur die Zahlen für eine Sprache mit lokaler Typinferenz, ein System, was beileibe nicht so stark ist wie die Inferenz von Sprachen wie *Haskell* oder *OCaml*. Tatsächlich ist mir in diesen Sprachen noch bei keiner lokalen Variable eine Typannotation untergekommen.

Die Lesbarkeit leidet darunter kein bisschen. Bei all der Kritik, die z. B. an Scala geäußert wird („zu komplex", „zu funktional", „zu akademisch"), tauchen die fehlenden Annotationen nie auf. Ganz im Gegenteil: Es werden oft die Schwächen der Typinferenz kritisiert; es könnten ja auch in der Implementation eines Interfaces die Methodenparameter inferiert werden, etc. pp.

Die Stärke der Typinferenz ist, dass statisch typisierte Programme so kompakt aussehen wie dynamisch typisierte – allerdings mit statischen Garantien. Dass diese Erkenntnis nun auch in den etablierten objektorientierten Sprachen ankommt, kann ich nur begrüßen. Jedenfalls darf die Abwesenheit von Typannotationen nicht mit der Abwesenheit von Typen verwechselt werden. Deshalb sehe ich in *'var'* auch weder Gefahr für Typsicherheit noch Lesbarkeit.

Viele Grüße
Lars Hupel

Sehr geehrter Herr Hupel,

vielen Dank für Ihre Anregungen. Gerne nehme ich Stellung dazu.

Typinferenz

In Ihrem Leserbrief schreiben Sie, dass Typinferenz den Vorteil hat, statisch typisierte Sprachen wie dynamisch typisierte aussehen zu lassen. Doch genau hier liegt meiner Meinung nach das Problem. Durch übermäßigen Gebrauch variabler Typen präsentiert sich *C#* als vermeintlich dynamisch typisiert. Obendrein verhält sich die Sprache dann noch nicht einmal ihrer scheinbaren Außenwirkung entsprechend. Ich denke nicht, dass die Erreichung dieses Umstandes das Ziel halbwegs geistesgegenwärtiger Sprachdesigner gewesen sein kann. Den einzigen wirklichen Vorteil – abgesehen von der vereinfachten Handhabung anonymer Typen – den ich in der Verwendung variabler Typen sehe, ist die Förderung der Bequemlichkeit von Entwicklern. In anderen Sprachen, wie etwa Scala, mögen andere Bedingungen vorherrschen. Dies kann ich aufgrund mangelnder Kenntnisse jedoch nicht beurteilen. Es sollte aber grundsätzlich hinterfragt werden, ob es sinnvoll ist, den Inhalt des Artikels verallgemeinernd auf Scala anzuwenden.

Verletzung des DRY-Prinzips

Die Deklaration konkreter Typen stellt in C# keineswegs eine Verletzung des *DRY*-Prinzips dar. Schließlich handelt es sich bei Variablendeklarationen nicht um redundanten Code. Die Angabe eines Typs ist in C# an dieser Stelle unumgänglich. Und das ist völlig unabhängig davon ob es sich um einen variablen oder einen konkreten Typ handelt. Sicherlich kann man die Überlegung anstellen, ob die Deklaration eines konkreten Typs bei gleichzeitiger Instanziierung einer Variable über einen Konstruktor

eine Redundanz darstellt und somit das DRY-Prinzip verletzt. Allerdings halte ich dies für überinterpretiert. Denn einer solchen Argumentation zu folge würde so manch andere streng typisierte und objektorientierte Sprache, welche keine Typinferenz unterstützt, implizit gegen das DRY-Prinzip verstoßen.

Die Bezeichnung „variabler Typ“

Microsoft bezeichnet *var* selbst als „variablen Typ“ bzw. als „impliziten Typ“. Ähnliches trifft im Übrigen auch auf das Schlüsselwort *dynamic* zu. Hier spricht Microsoft vom statischen Typ *dynamic*. Da diese Namenskonventionen offiziell propagiert werden und somit der C#-Gemeinde vertraut sein dürften, habe ich im Artikel ebenfalls darauf zurückgegriffen. Syntaktisch werden die Schlüsselwörter exakt wie Typen angewendet, weswegen ich deren Bezeichnung als eben solche durchaus schlüssig finde. Aus Sicht der Runtime stimmt das natürlich nicht und es lässt sich deshalb in der Tat darüber streiten ob diese Konvention generell geeignet ist.

Typsicherheit

Ich stimme Ihnen voll und ganz zu, dass Typinferenz die Typsicherheit von C# keineswegs gefährdet. Der Artikel will deshalb auch zu keinem Zeitpunkt die Abwesenheit von Typdeklarationen mit der Abwesenheit von Typen gleichstellen. Eventuell. liegt hier ein Missverständnis vor.

Lesbarkeit

Betrachten wir folgenden Quelltext: *var f = Foo()*;

In diesem Beispiel ist nicht ersichtlich, welcher Typ sich hinter der Variable *f* verbirgt. Die Methode *Foo()* lässt keinerlei Rückschlüsse zu. Eine konkrete Typdeklaration würde die Lesbarkeit an dieser Stelle selbst dann verbessern, sollte der konkrete Typ nicht exakt dem des zurückgelieferten Objekts entsprechen (Liskovsches Substitutionsprinzip).

Der Artikel soll durchaus provozieren und ist nicht zuletzt deswegen ironisch gefärbt. Dennoch spiegelt er voll und ganz meine Meinung wider. Allerdings bin ich fundierten Argumenten gegenüber aufgeschlossen und lasse mich gerne eines Besseren belehren. Schließlich handelt es sich um meine persönliche Sichtweise, die sich sicherlich nicht mit der Anderer decken muss. Über Stil und Qualität von Code kann man bekanntlich unendlich diskutieren. Die pauschale inhaltliche Übertragung des Artikels auf andere Sprachen als C# halte ich indes für wenig geeignet. Zu unterschiedlich können Syntax und Paradigmen ausgeprägt sein.

Viele Grüße
Tobias Krügel

Klettergarten

Common Table Expressions und rekursive Abfragen in SQL

von Andreas Heiduk

Selbst in einfachen Schemata verstecken sich sehr oft einige hierarchische Strukturen wie Schritt, Unterschritt und Teilschritt. Ebenso oft werden diese mit *SQL* „platt geklopft“ und ausprogrammiert. Dabei geht es auch anders: Flexibel!

Für hierarchische Abfragen in SQL gibt es die *Common Table Expressions (CTE)*. Genau genommen gibt es zwei Arten von CTE mit jeweils eigenen Anwendungsmöglichkeiten: „normal“ und „rekursiv“.

Beide Arten wurden in SQL-1999 [1] standardisiert und werden von den üblichen Verdächtigen unterstützt: *Oracle, DB2, MS SQL Server* auf der kommerziellen Seite, *PostgreSQL*, *HSQLDB* und *Firebird* auf der Open Source-Seite. *MySQL* fehlt in diesem Club [2]. Die folgenden Beispiele benutzen PostgreSQL, aber ohne spezielle Eigenheiten zu verwenden.

Normale CTE

Die einfacheren, nicht-rekursiven CTE sehen im Kern wie folgt aus.

```
WITH
  query_name1 AS ( SELECT … ) [,
  query_name2 AS ( SELECT … ), … ]
SELECT …
  FROM query_name1, query_name2 …
```

Das heißt, man kann auf einfache Art und Weise *Subqueries* aus der eigentlichen *Query* herausziehen und mit einem Namen versehen. Alleine diese Möglichkeit von CTE hat schon viele Vorteile:

- Vereinfachung komplexer Queries
- Vermeidung temporärer Tabellen[1] oder *Views*
- mehrfache Verwendung einer Subquery
- keine Mehrfachauswertung

[1] Damit sind explizit angelegte temporäre Tabellen gemeint. Was die DB intern zu brauchen denkt, …

Der letzte Punkt bedeutet, dass *WITH*-Queries nur einmal ausgewertet werden, auch wenn sie mehrfach in der Haupt-Query oder in anderen *WITH*-Queries angesprochen werden. Dadurch lassen sich einerseits Seiteneffekte vermeiden und andererseits lässt sich gezielt Einfluss auf die Performance nehmen.

Als Ausblick sei noch erwähnt, dass PostgreSQL sowohl in den *WITH*-Queries selbst als auch in der Haupt-Query nicht nur *SELECT* erlaubt, sondern auch die modifizierenden Statements *INSERT*, *UPDATE* und *DELETE*. Damit kann man lustige Sachen machen [3], hilft uns aber bei Hierarchien nicht weiter.

Rekursive Abfragen

Eigentlich ist es ja erstaunlich, wie lange der SQL-Standard keine Unterstützung für rekursive Abfragen bereitstellte – vor allem wenn man bedenkt, dass das zugrunde liegende relationale Datenmodell als Nachfolger des hierarchischen und des Netzwerkmodels entstand. Da aber die Realität nicht unbedingt auf Standardisierungsgremien wartet, sind – natürlich – zwei Dinge geschehen:

- Es wurden Mechanismen entwickelt, um rekursive und baumartige Strukturen mehr oder weniger effizient auf vorhandene SQL-Möglichkeiten abzubilden. In „The Art of SQL“ [4] finden sich einige davon.
- DB-Hersteller kochten ihre eigene Syntax. Insbesonders Oracle unterstützte bis zur Version 11gR2 nur die eigene *CONNECT BY*-Syntax.

Aber braucht man das überhaupt? Wo in einer 08/15-Enterprise- oder Web-Anwendung sind denn im Datenmodell Hierarchien und Baumstrukturen zu finden?

Ganz offensichtliche Beispiele sind:

- Artikelhierarchie in Webshops
- Organisationsstrukturen
- Archive / Logistik (Regal, Palette, Box,...)
- Konversationen in E-Mails, Foren oder Newsgruppen

Ein weiteres, nicht so offensichtliches Beispiel ist die *Bill of Material (BOM)*, bei der alle Bauteile von Komponenten und deren Unterkomponenten berechnet werden sollen.

Aber ganz so weit bzw. ganz so speziell muss es oft gar nicht werden. Viel öfter begegnen mir „kleine" Hierarchien der Art „Schritt, Teilschritt, Unterschritt". Seltsamerweise sind mir dabei nie mehr als drei Stufen begegnet – auch wenn meistens fachlich einige Verrenkungen gemacht werden mussten um die Realität in genau diese zwei oder drei Stufen einzuteilen.

In der Realität dieses Artikels soll es um die Warenhierarchie eines kleinen Ladens in einem fiktiven Spiel gehen. Derartige Hierarchien eigenen sich besonders zum Experimentieren, da hier die Äste von vorne herein unterschiedlich lang sein können.

Apropos experimentieren: Bei *SQL-Fiddle* liegt für Sie unter *http://sqlfiddle.com/#!12/63fcb/0* das Schema und die Beispieldaten schon bereit. Sie können dort einfach und online die Beispiele ausprobieren oder ändern.

key	parent_key	item	value
1		clothing	
2		shoes	
3		armour	
4		weapons	
5		equipment	
10	1	leather	
11	1	cloth	
20	2	boots	15
21	2	low shoe	10
22	2	sandal	2
30	3	iron plate	100
31	3	bronze plate	50
32	3	copper plate	25
40	4	swords	
41	4	knifes	
42	4	clubs	
400	40	bronze sword	20
401	40	iron sword	50
402	40	mag. sword	60
410	41	iron knife	10

Die grundlegende Syntax für rekursive Abfragen lautet:

```
WITH RECURSIVE
query_name AS (
    SELECT … FROM raw_table …
    UNION [ALL]
    SELECT … FROM query_name …
)
SELECT … FROM query_name;
```

Jede rekursive Query enthält eine *UNION* (bzw. *UNION ALL*), deren erster Teil der nicht-rekursive Teil ist, von dem aus die Abfrage gestartet wird. Dieser Teil liefert also die Wurzel(n) für die Rekursion. Der zweite Teil der *UNION* enthält den rekursiven Teil, in dem die Rekursion aufgebaut wird – man hat in diesem Zweig Zugriff auf *query_name*. Damit kann man schon mal rekursiv abfragen, welche Artikel es unterhalb eines bestimmten Knotens gibt:[2]

```
WITH RECURSIVE
cte AS (
    SELECT store.*, 1 as i
    FROM store
    WHERE item = 'weapons'
  UNION ALL
    SELECT store.*, i+1
    FROM cte
    JOIN store
      ON cte.key = store.parent_key
)
SELECT * FROM cte;
```

key	p..	item	value	i
4		weapons		1
40	4	swords		2
41	4	knifes		2
42	4	clubs		2
400	40	bronze sword	20	3
401	40	iron sword	50	3
402	40	mag. sword	60	3
410	41	iron knife	10	3

Zu beachten ist, dass das Selektions-Kriterium, die Auswahl der Kategorie *'weapons'*, nicht im Hauptteil der Query steht sondern im nicht-rekursiven Teil von *cte*. Ausgehend von den so gefundenen Elementen werden rekursiv nach unten passende Kind-Elemente gesucht. Dabei wird, nur zur Übersicht, ein Ebenenzähler *i* mit nach unten propagiert.

[2] http://sqlfiddle.com/#!12/63fcb/8/0

Anstatt *Top-Down* kann man auch *Bottom-Up* suchen, um z. B. direkt das magische Schwert zu finden. Man geht diesmal im nicht-rekursiven Teil vom gewünschten Gegenstand aus und hangelt sich dann nach oben.[3]

```
WITH RECURSIVE
cte AS (
    SELECT store.*, 1 as i
    FROM store
    WHERE item = 'mag. sword'
  UNION ALL
    SELECT store.*, i+1
    FROM cte
    JOIN store
      ON cte.parent_key = store.key
)
SELECT * FROM cte;
```

```
 key | p...|    item    | value | i
-----+-----+------------+-------+---
 402 |  40 | mag. sword |    60 | 1
  40 |   4 | swords     |       | 2
   4 |     | weapons    |       | 3
```

So weit so gut. Jedoch sollte die Frage „Wo ist das magische Schwert?“ mit einer klaren Antwort beantwortet werden und nicht mit dreien wie von einem Orakel. Das heißt man will für jeden gefundenen Gegenstand gleich den Pfad zu diesem Gegenstand.

Den Schlüssel dazu haben wir eigentlich schon mit dabei: Bisher propagieren wir das *i* als einfache *Debug*-Hilfe durch die Hierarchie. Was hindert uns daran, stattdessen

- die auf dem Weg liegenden Schlüssel als Pfad aufzusammeln und daneben
- mehr als ein Datum zu progagieren?

Daher ändern wir die Query einfach so ab, dass Bottom-Up alle relevanten Informationen inklusive des bisher berechneten Pfades propagiert.[4]

```
WITH RECURSIVE
cte AS (
    SELECT
      store.key,
      store.parent_key,
      store.item,
      store.value,
      ARRAY[key] AS path
    FROM store
    WHERE item = 'mag. sword'
  UNION ALL
    SELECT
      cte.key,
      store.parent_key,
      cte.item,
      cte.value,
      store.key || cte.path
    FROM cte
    JOIN store
      ON cte.parent_key = store.key
)
SELECT * FROM cte;
```

```
key|par| item     |value|    path
---+---+----------+-----+------------
402| 40|mag. sword|  60 | {402}
402|  4|mag. sword|  60 | {40,402}
402|   |mag. sword|  60 | {4,40,402}
```

Besondere Sorgfalt muss darauf gelegt werden, dass nicht einfach alles propagiert wird (*store.**), sondern dass zumindest das Verbindungskriterium (*parent_key*) erhalten bleibt. Ansonsten hat man eine endlose Rekursion am Hals. Aus diesem Grund wurden die bisher knappen *SELECT store.**-Ausdrücke durch explizite Angaben ersetzt.

Als Nebenbemerkung für nicht PostgreSQL-Benutzer: Um den Pfad einfach handhabbar zu machen, wurde ein *Array* verwendet, das mit *element* || *array* erweitert wird. Stattdessen kann man auch einfach einen kommaseparierten String aufbauen.

Zum Ergebnis selbst ist zu sagen: Die relevanten Informationen inklusive vollständigem Pfad zum gewünschten Gegenstand stehen nun in der letzten Zeile zur Verfügung. Man kann die überflüssigen Zeilen und die überflüssige Spalte *parent_key* natürlich am Ende der Query herausfiltern[5]:

```
…
SELECT key, item, value, path
FROM cte
WHERE parent_key IS NULL ;
```

```
key|  item    |value|    path
---+----------+-----+------------
402|mag. Sword|   60| {4,40,402}
```

[3] http://sqlfiddle.com/#!12/63fcb/1/0

[4] http://sqlfiddle.com/#!12/63fcb/5/0

[5] http://sqlfiddle.com/#!12/63fcb/9/0

Dennoch sollte man sich bewusst sein, dass im allgemeinen Fall viele temporäre *Tupel* erzeugt werden. Dies geht zwar nicht auf die Kosten der Latenz und der Bandbreite zwischen Applikation und Datenbank, aber bei großen Datenmengen ggf. auf den Speicherverbrauch bzw. auf die IO-Kanäle der Datenbank selbst. Also sollte man diese Größe im Blick behalten.

Fazit

Damit sind eigentlich die Brot-&-Butter-Fälle auch schon abgedeckt! Navigieren Top-Down und Bottom-Up, Propagieren von Werten und Berechnen von Pfaden. Das alles ist relativ einfach möglich, ohne für jedes Tupel bzw. für jede Hierarchie-Ebene einen Roundtrip zwischen Applikation und Datenbank zu benötigen. Es werden auch keinerlei Annahmen über die Tiefe der Bäume getroffen.

Neben den rekursiven Möglichkeiten bietet *WITH* auch so gute Möglichkeiten, komplexe Anfragen verständlich zu zerlegen.

Wenn Sie also demnächst ebenfalls über eine Struktur wie „Schritt“, „Teilschritt“ und evtl. auch „Unterschritt“ stolpern, denken sie einfach „Das kann ich auch mit beliebigen Ebenen“.

Referenzen

[1] ISO/IEC 9075-2:1999
www.iso.org/catalogue_detail.htm?csnumber=26197

[2] Wikipedia *Hierarchical and recursive queries in SQL*
http://en.wikipedia.org/wiki/Hierarchical_query#Common_table_expression

[3] PostgreSQL *7.8.2. Data-Modifying Statements in WITH*
http://www.postgresql.org/docs/current/interactive/queries-with.html#QUERIES-WITH-MODIFYING

[4] Faroult, Stephane *The Art of SQL*, O'Reilly, 2006

Kurzbiographie

Andreas Heiduk (andreas.heiduk@mathema.de) ist als Senior Consultant für die MATHEMA Software GmbH tätig. Seine Themenschwerpunkte umfassen die Java Standard Edition (JSE) und die Java Enterprise Edition (JEE). Daneben findet er alle Themen von hardware-naher Programmierung bis hin zu verteilten Anwendungen interessant.

Web-Anwendungen auf Klick

Einführung in Apache Click

von Rustam Khakimov

Kein *MVC* und *JSP* sondern ein seitenorientiertes Design und „natürliche“ *Rich-Client-*Programmierungsweise bei der Entwicklung von Web-Anwendungen. *Apache Click* bietet hierfür eine interessante Lösung. Dieser Artikel stellt das Framework vor und zeigt anhand von zahlreichen Beispielen Entwicklung und Test einer Web-Applikation.

Allgemeines

Das Click-Framework ist ein Open Source-Projekt und wurde 2010 zum *Apache Top Level Project* befördert. Seitdem wird Click unter der *Apache Licence* veröffentlicht. Das Framework steht als Archiv zur Verfügung. Im Archiv sind *Source-Code*, Beispiele, Dokumentation und Binäres zu finden. Apache Click verwendet keine Model View Controller (MVC)-Paradigmen sondern ist seitenorientiert und hat ein *Event*-basiertes Programmiermodel, das man aus der Rich Client-Programmierung mit z. B. *Swing* kennt. Für das *Template-Rendering* wird standardmäßig die *Apache Velocity Template Engine* eingesetzt. Die Verwendung von anderen Template-Engines wie z. B. JavaServer Pages (JSP) oder *Freemarker* ist auch möglich. Als *Servlet-Container* verwendet das Framework *Apache Tomcat* oder *Jetty*. Apache Click bietet Mechanismen für die client- und serverseitige Validierung an. Außerdem unterstützt es *Ajax-Requests* mit der serverseitigen Verarbeitung. Alle Elemente in Click sind *stateless*, aber es ist möglich, leichtgewichtige Objekte in der *Session* zu speichern. Eine testgetriebene Entwicklung mit Click ist auch kein Problem. Dazu gibt es eine *Mock-API*, die es ermöglicht *Unit-Tests* ohne *Web-Container* auszuführen.

Für die Entwicklung mit Apache Click kann das *Eclipse* Plug-in *ClickIDE* heruntergeladen werden. *ClickIDE* erleichtert die Handhabung des Frameworks und man kann sich komplett der Entwicklung von Anwendungsfällen widmen.

Architektur von Apache Click

Das Herz des Frameworks ist ein zentraler *Request-Dispatcher*, der in der Klasse *org.apache.click.ClickServlet* implementiert ist. Beim Eintreffen von einem *Browser-Request* erzeugt *ClickServlet* eine Instanz vom Typ *org.apache.click.Page*. Zu jeder solchen *Page*-Instanz existiert immer ein Template. Dieses Template wird durch die Template-Engine verarbeitet und mit der Page-Instanz zusammengefügt. Die fertige HTML-Seite wird danach zum Browser zurückgeschickt:

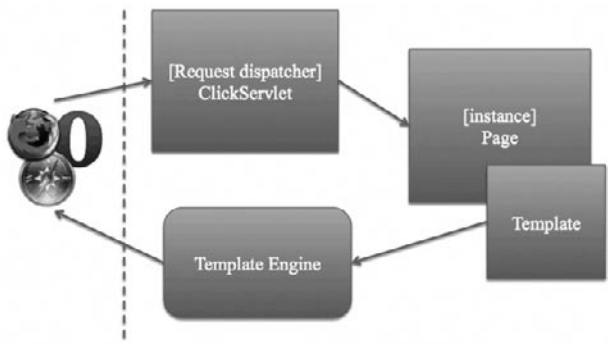

Abbildung 1: Apache Click Architektur [6].

Beispiel 1: Hallo Welt!

Schauen wir mal, wie die „Hallo Welt“-Anwendung in Apache Click aussieht.

Abbildung 2: Webapplikation „Hallo Welt!“

Schritt 1: Click-Projekt erstellen

Das neue Projekt wird mit Hilfe der *ClickIDE* erstellt. Dafür gehen wir in den Wizard für *Dynamic Web Project* unter File -> New -> Other... und wählen da in der Configuration „Apache Click“ (siehe Abb. 3).

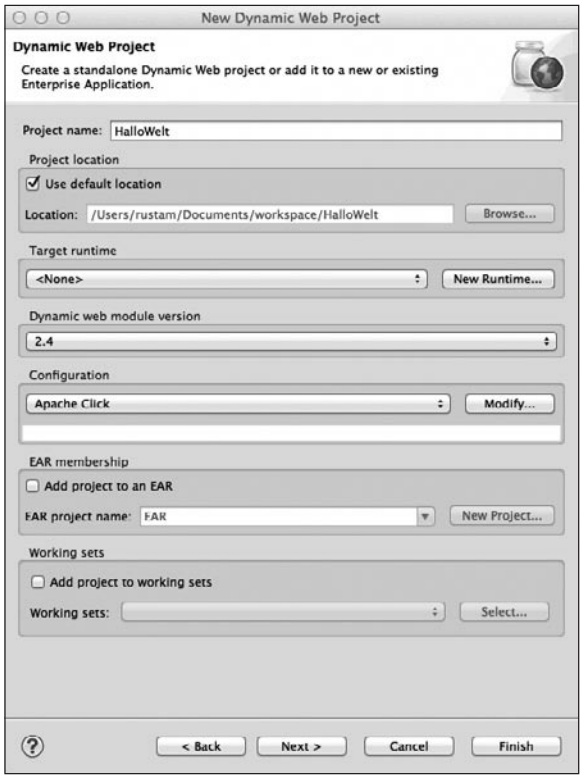

Abbildung 3: Apache Click Projekt erstellen

Nach der Eingabe des Projektnamens „Hallo Welt“ können wir das Projekt erstellen. Das neu angelegte Click-Projekt (siehe Abb. 4) besteht aus drei Teilen: *Java Resources*, *JavaScript Resources* und *WebContent*. Die ersten zwei Bereiche sind selbsterklärend, sie beinhalten Java- bzw. JavaScript-Quellcode. Im *WebContent* befinden sich Templates für Web-Seiten und zwei Konfigurationsdateien, *./WEB-INF/click.xml* und *./WEB-INF/web.xml*, für die Click- bzw. Servlet-Konfiguration.

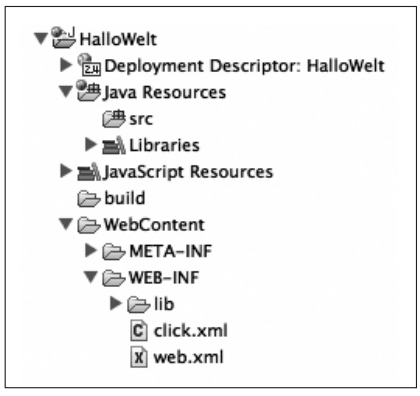

Abbildung 4: Projekt HalloWelt

Schritt 2: Click-Seite erstellen

Unsere erste Click-Seite erstellen wir auch mit Hilfe der *ClickIDE*. Im Wizard für die Seitenerstellung tragen wir unseren Template-Namen *HalloWeltSeite.htm* und den Klassennamen *HalloWeltSeite* ein, dann setzen wir den Hacken *Add mapping to click.xml* um das Template und die Java-Klasse aufeinander zu mappen (siehe Abb. 5). Die anderen Felder bleiben unverändert.

Abbildung 5: Click-Seite erstellen

Das Eclipse Plug-in legt für uns eine Java-Klasse *de.HalloWeltSeite* im Verzeichnis *./Java Resources/src/* und ein *Velocity*-Template *HallWeltSeite.htm* unter dem *./WebContent/* an. Außerdem erweitert es *click.xml* um einen Mapping-Eintrag (siehe Abbildung 6 Zeile 02).

```
    …
01  <pages package="" autobinding="annotation">
02  <page path="HalloWeltSeite.htm"
    classname="de.HalloWeltSeite"/>
03  </pages>
    …
```

Abbildung 6: *click.xml* Seiten-Mapping

Jetzt bauen wir in die *HalloWeltSeite.java* unsere Ausgabe „Hallo Welt“ ein (siehe Code Zeile 06 – 09). Mit der Annotation *@Bindable* machen wir unsere Instanzvariablen *halloWelt* und *datum* für das Template sichtbar.

```
04  public class HalloWeltSeite extends Page {
05
06      @Bindable
07      private String halloWelt = "Hallo Welt!";
08      @Bindable
09      private Date datum = new Date();
10
11      public HalloWeltSeite(){
12      }
13  }
```

Abbildung 7: *HalloWeltSeite*.java

In das Template *HalloWeltSeite.htm* fügen wir Zeile 20 und 21 ein (siehe Abb. 8). Wenn ein entsprechender *Request* eintrifft, erzeugt *ClickServlet* eine Instanz vom Typ *de.HalloWeltSeite* und *Velocity* ersetzt die Template-Variablen *$halloWelt* und *$datum* durch entsprechende Werte aus dieser Instanz. In der Zeile 21 (Abb. 8) verwenden wir noch zusätzlich einen *Formatter org.apache.click.util.Format*, den uns das Click-Framework zur Verfügung stellt um das Datum in einem bestimmten Format auszugeben.

```
14  <html>
15    <head>
16      <META HTTP-EQUIV="Content-Type"
        CONTENT="text/html;charset=UTF-8">
17    <title>Blank</title>
18  </head>
19  <body>
20    <H1>$halloWelt</H1>
21    ${format.date($datum, "dd.MM.yyyy 'um' HH:mm")}
22    </body>
23  </html>
```

Abbildung 8: *HalloWeltSeite.htm*

Schritt 3: Startseite definieren

Bevor wir unsere Anwendung starten, müssen wir dem *ServletContainer* noch die Startseite unserer Web-Applikation mitteilen. Dafür wird *web.xml* angepasst (siehe Abb. 9 Z. 25).

```
    ...
24  <welcome-file-list>
25   <welcome-file>HalloWeltSeite.htm</welcome-file>
26  </welcome-file-list>
    ...
```

Abbildung 9: *web.htm* Startseite

Fertig! Jetzt können wir unsere Click-Applikation deployen und starten.

Der erste Schritt ist getan. „Hallo Welt“ läuft. Obwohl unser Beispiel sehr klein und einfach war, haben wir vieles erreicht. Wir haben ein Click-Projekt aufgesetzt, eine Seite erstellt, kleine Konfigurationsänderungen vorgenommen und die *ClickIDE* kennengelernt. Als Nächstes schauen wir uns *Pages*, die wichtigste Komponente von Apache Click, detaillierter an.

Seiten (Pages)

Apache Click ist ein seitenorientiertes Framework. Eine logische Click-Seite ist eine Kombination aus einer Java-Klasse und einem Template. Diese Zuordnung geschieht entweder nach Namenskonvention oder durch Mapping in der *click.xml* (siehe Abb. 6). Die Java-Klasse muss von *org.apache.click.Page* abgeleitet sein und den *default*-Konstruktor implementieren (siehe Abb. 7). Für das Template können mehrere Formate verwendet werden. Standardmäßig arbeitet das Framework mit Apache Velocity, kann aber für die Verarbeitung z. B. von JSP- oder Freemarker-Templates konfiguriert werden. Die Click-Seiten sind statuslos.

Die Oberklasse *org.apache.click.Page* stellt zahlreiche leere *Handler*-Methoden zur Verfügung:

- *onSecurityCheck()* wird sofort nach der Objekterzeugung aufgerufen. Hier kann z. B. geprüft werden, ob der Benutzer autorisiert ist diese Seite zu öffnen.
- *OnInit()* kann z. B. für die Initialisierung der Seiten-Ressourcen bzw. Business-Objekte verwendet werden.
- *onGet()/onPost()*. Nach der Verarbeitung von Control-Elementen (siehe unten Controls) der Seite wird die *onGet*- bzw. *onPost*-Methode (abhängig davon, ob es ein *Get*- oder *Post-Request* ist) aufgerufen. Diese Methoden können z. B. für das *direct-Rendering* von Templates verwendet werden, d. h. die Templates werden dynamisch zur Laufzeit erstellt und der Seite zugeordnet.
- *OnRender()* wird unmittelbar vor dem Template-Rendering aufgerufen. Hier können z. B. fehlende Web-Ressourcen geladen werden.
- *OnDestroy()* wird immer aufgerufen, unabhängig von aufgetretenen Fehlern während der Seitenverarbeitung. Sie wird verwendet um Seiten-Ressourcen freizugeben.

Seitennavigation

Zur Navigation zwischen den Seiten bietet Apache Click drei unterschiedliche Verfahren an: zwei übliche per *Forward* und *Redirect* sowie ein drittes mittels Template-Austausch. Die Oberklasse *org.apache.click.Page* stellt drei Methoden für Forward und Redirect zur Verfügung (siehe Abb. 10). In den Zeilen 27 und 31 wird eine Weiterleitung auf die nächste Seite durch den Template-Pfad realisiert. In den Zeilen 28 und 32 navigiert man über die Java-Klasse zur nächsten Seite. In der Zeile 29 wird dagegen die nächste Seite vor dem Forwarding zuerst instanziiert. Die Zeilen 34 – 36 zeigen einen Redirect-Aufruf mit übergebenen Request-Parametern.

```
    ...
27  setForward("NextPage.htm");
28  setForward(NextPage.class);
29  setForward(new NextPage());
30
```

```
31  setRedirect("NextPage.htm");
32  setRedirect(NextPage.class);
33
34  Map<String, Integer> reqParams =
      new HashMap<String, Integer>();
35  reqParams.put("id", 42);
36  setRedirect(NextPage.class, reqParams);
    ...
```

Abbildung 10: Navigation mittels *setForward()* / *setRedirect()*

Mit Hilfe der Methode *setPath* kann das zur Java-Klasse durch das Mapping zugeordnete Template zur Laufzeit ausgetauscht werden (siehe Abb. 11).

```
    ...
37  setPath("NextTemplate.htm");
    ...
```

Abbildung 11: Navigation mittels *setPath()*

Master Template

Mit Apache Click ist es sehr leicht, unserer Web-Anwendung ein standardisiertes Look & Feel zu geben. Dafür wird eine Oberklasse für alle unsere weiteren Seiten erstellt. Diese Oberklasse soll dann von *org.apache.click.Page* abgeleitet sein und die Methode *getTemplate()* so überschreiben, dass sie den Pfad vom Mastertemplate zurückliefert (siehe Abb. 12).

```
38  class MasterPage extends Page {
39    @Override
40    public String getTemplate() {
41      return "./MasterPage.htm";
42    }
43  }
```

Abbildung 12: Master-Seite

Das einfachste Mastertemplate sieht folgendermaßen aus (siehe Abb. 13). Durch den Velocity-Befehl *#parse* wird das aktuelle Template eingefügt. Die Variable *$path* wird durch *ClickServlet* in den Velocity-Kontext eingefügt und enthält den Pfad zum aktuellen Template.

```
44  <html>
45    <head>
46      <title>$title</title>
47      $imports
48    </head>
49    <body>
50      <H2 class="title">$title</H2>
51      #parse($path)
52    </body>
53  </html>
```

Abbildung 13: Mastertemplate

Steuerelemente (Controls)

Controls ermöglichen Apache Click vieles in Java zu implementieren anstatt sich mit HTML, Apache Velocity oder JSP herumzuschlagen. Sie werden clientseitig gerendert und serverseitig verarbeitet. Das Framework stellt Mechanismen für die client- und serverseitige Validierung von Controls zur Verfügung. Wie das alles funktioniert, sehen wir in Beispiel 2.

Beispiel 2: Web-Anwendung auf Klick

Stellen wir uns vor, dass wir die Besucher unserer Web-Anwendung registrieren wollen. Dazu erstellen wir uns ein Formular, in dem sich Besucher mit dem Namen und ihrer E-Mail-Adresse registrieren können. Diese Daten werden an unseren Webserver geschickt und da abgespeichert.

Zuerst legen wir ein Click-Projekt an und erstellen eine Click-Seite. Wie das geht, wissen wir bereits. Deswegen starten wir sofort mit der Programmierung unseres Formulars.

Schritt 1: Formular erstellen

In der zur Click-Seite gehörigen Java-Klasse, die eine Unterklasse von *org.apache.click.Page* ist, überschreiben wir die *onInit*-Methode (siehe Abb. 14).

```
54  @Override
55  public void onInit() {
56    final Form form = new Form("formUserRegistration");
57    addControl(form);
58
59    TextField fieldUserName =
        new TextField("userName", "Ihr Name:", true);
60    fieldUserName.setFocus(true);
61    fieldUserName.setMinLength(2);
62    fieldUserName.setMaxLength(30);
63
64    EmailField fieldUserMail =
        new EmailField("userMail", "Ihre E-Mail:", true);
65
66    Submit submit = new Submit("submit", "Registrieren");
67
68    form.add(fieldUserName);
69    form.add(fieldUserMail);
70    form.add(submit);
71
72    form.setJavaScriptValidation(true);
73    form.setValidate(true);
74
75    submit.setActionListener(new ActionListener() {
76      private static final long serialVersionUID = 1L;
77
78      @Override
```

```
79        public boolean onAction(Control source) {
80            if(form.isValid()) {
81             String userName =
              ((TextField)form.getControl("userName")).getValue();
82            String userMail =
              ((EmailField)form.getControl("userMail")).getValue();
83
84            storeUserInDb(userName, userMail);
85            return true;
86          }
87          return false;
88          }
89      });
90  }
```

Abbildung 14: *onInit*-Methode

In den Zielen 56 – 57 (Abb. 14) instanziieren wir unser Formular mit dem Namen *formUserRegistration* und registrieren es mittels *addControl()* bei unserer Seite. Durch den Formularnamen können wir im Template auf das Formular zugreifen. In den Zeilen 59 – 66 erstellen wir weitere Controls:

- ein TextField *fieldUserName* mit dem Namen *userName* und dem Label „Ihr Name“. Dieses Feld ist ein Muss-Feld, d. h. eine Eingabe ist erforderlich (siehe Abb 14 Z. 59). Da wir unsere Anwendung benutzerfreundlich gestalten möchten, setzen wir mit der Methode *setFocus(true)* den Fokus ins Feld, um das Ausfüllen des Formulars zu erleichtern (siehe Abb. 14 Zeile 60). Beim Rendern des Feldes wird eine dazu entsprechende *JavaScript Routine* von Apache Click generiert, die den Fokus ins Feld setzt sobald die Seite in einem Browser geladen wird. Zum Schluss setzen wir noch die minimale und maximale Länge fürs Feld.
- ein EmailField *fieldUserMail* mit dem Namen *userMail* und dem Label „Ihre E-Mail“. Dieses Feld ist auch ein Muss-Feld (siehe Abb. 14 Z. 64) und
- einen *Submit-Button* mit dem Label „Registrieren“ (siehe Abb. 14 Z. 66).

Unsere Steuerelemente fügen wir dem Formular bei (siehe Abb. 14 Z. 68 – 70). Als Nächstes schalten wir Formular-Validierung an. Zu Demonstrationszwecken machen wir beides, client- und serverseitige Validierung (siehe Abb. 14 Z. 72 – 73). Bei der clientseitigen Validierung erzeugt das Framework beim Rendern des Formulars eine JavaScript Routine. Diese JavaScript-Validierung wird beim *Submit* im Browser ausgeführt. Bei der serverseitigen Validierung wird der Formularinhalt im Server geprüft.

Wir möchten die gewonnenen Informationen von registrierten Benutzern z. B. in einer Datenbank speichern.

Wie schon erwähnt wurde, hat Apache Click ein eventbasiertes Programmiermodel, d. h. wir können ähnlich wie bei der Rich-Client-Programmierung vorgehen. Registrieren wir beim Submit-Button einen *ActionListener* (siehe Abb. 14 Z. 75). Jetzt bleibt uns nur die Methode *onAction(Control source)* zu überschreiben (siehe Abb. 14 Z. 79 – 88), die serverseitig ausgeführt wird, sobald der Submit-Button im Browser gedrückt wird. Wenn die Formularvalidierung erfolgreich war, werden die Werte aus den Formularfeldern *userName* und *userMail* in der Datenbank gespeichert. Ansonsten brechen wir die weitere Request-Verarbeitung ab, in dem wir ein *false* zurückliefern.

Schritt 2: Template erstellen

Das Template zu unserer Seite sieht sehr einfach aus (siehe Abb. 15). Mit der Variable *$formUserRegistration* fügen wir das gesamte Formular ins Template ein. Die Variable *$imports* liefert uns alle *Head*-Elemente inklusive der JavaScript- und *css*-Elemente. In unserem Fall den JavaScript-Code und *css*-Klassen für die Fokussetzung und Validierung.

```
091 <html>
092   <head>
093     <title>Benutzer registrieren</title>
094     $imports
095   </head>
096   <body>
097 .....<H1>Benutzer registrieren<H1>
098   $formUserRegistration
099   </body>
100 </html>
```

Abbildung 15: Template

Schritt 3: Starten und Testen

Unser Formular ist fertig. Wir können die Anwendung starten (siehe Abb. 16). Die Eingabefelder sind als Muss-Felder entsprechend im Browser gekennzeichnet und der Fokus ist ins Feld „Ihr Name“ gesetzt.

Abbildung 16: „Benutzer registrieren“ Eingabeformular

Die clientseitige Validierung funktioniert auch (siehe Abb. 17). Beim Drücken auf den Knopf „Registrieren“ wurde die JavaScript Routine ausgeführt, die erkannt hat, dass das Muss-Feld „Ihr Name“ leer war und die E-Mail-Adresse nicht das gültige Format hatte. Falls wir uns nur für die serverseitige Validierung entschieden hätten, wäre das Formular zum Server verschickt und erst da validiert worden. Das Validierungsergebnis wäre dann gleich. Die ausgegebenen Meldungen sind Standardmeldungen, die bei Bedarf entsprechend geändert werden können.

Abbildung 17: „Benutzer registrieren“ validieren

Testen

Apache Click ist nicht nur ein Framework das uns beim Entwickeln von Web-Applikationen unterstützt, es hilft uns auch das richtig testgetrieben zu machen. Das Framework stellt eine Mock-API zur Verfügung, die es ermöglicht Tests ohne Webserver auszuführen.

Beispiel 3: Testen auf Klick

In Beispiel 2 konnten wir die serverseitige Validierung schlecht testen, weil die JavaScript Routine beim Submit verhindert hat, dass wir ungültige Daten an den Server senden. Wir haben angenommen, dass die serverseitige Validierung auch richtig funktioniert. Jetzt prüfen wir, ob es wirklich so ist.

```
101 MockContainer container =
      new MockContainer(
        WORKSPACE_PATH +
        "Click/UserRegistration/WebContent"
      );
102 container.start();
103
104 InputUserPage inputPage =
      (InputUserPage)container.testPage("InputUserPage.htm");
105
106 container.setParameter(
      Form.FORM_NAME, "formUserRegistration"
    );
107 container.setParameter("userName", "");
108 container.setParameter("userMail", "rustam % mathema.de");
109 container.setParameter("submit", "");
110 InputUserPage nextPage =
      (InputUserPage)container.testPage("InputUserPage.htm");
111
112 Form from =
      (Form)nextPage.getModel().get("formUserRegistration");
113
114 assertFalse(container.getHtml().contains("Error Page"));
115 assertEquals(2, form.getErrorFields().size());
```

Abbildung 18: Unit-Test mit Apache Click Mock API

Für die Testdurchführung stellt die Apache Click Mock-API einen *MockContainer* zur Verfügung (siehe Abb. 18). In den Zeilen 101 – 102 wird er instanziiert und gestartet. Die Methode *testPage()* simuliert hier einen Browser-Request für die Seite *InputUserPage* (siehe Abb. 18 Z. 104). Danach wird ein Submit simuliert (siehe Abb. 18 Z. 106 – 110). Dabei werden der Formularname, ein leerer Wert anstatt dem Benutzernamen, eine ungültige E-Mail-Adresse und der Submit-Befehl als Request-Parameter übergeben. Nach dem „Verschicken“ des Submit bekommen wir eine neue Seite, die wir uns für die nachfolgende Analyse merken. In der Zeile 112 holen wir aus der Seite unser Formular und prüfen ob die Validierung des Formulars erfolgreich war (siehe Abb. 18 Z. 115). Die Methode *getErrorFields()* liefert alle mangelhaften Felder während der Validierung. Da Apache Click eine eigene Fehlerbehandlungsroutine besitzt und bei Parserfehlern in Templates bzw. aufgetretenen *Exceptions* im Java-Code eine Fehlerseite inklusive Fehlerbericht generiert, wird in der Z. 114 geprüft, ob es sich hier um keine Fehlerseite handelt.

To Click or Not to Click: That is the question!

Wenn man auf der Suche nach einem Web-Framework ist, lohnt es sich einen kurzen Blick auf Apache Click zu werfen. Man wird angenehm überrascht sein, wie leicht es ist, das Framework zu lernen und wie handlich es ist. Ob Click dann Klick macht, soll jeder für sich selbst entscheiden.

Referenzen

[1] Apache Click *home*, http://click.apache.org
[2] Apache Click *ClickIDE*, http://click.apache.org/docs/click-ide.html
[3] Apache Click *User Guide*, http://click.apache.org/docs/user-guide.html
[4] Apache Click *Mock API*, http://click.apache.org/docs/mock-api/overview-summary.html
[5] Apache Click *Examples*, http://click.avoka.com/click-examples/home.htm
[6] Stäuble, Markus *Apache Click: Dann eben ohne MVC* http://it-republik.de/jaxenter/artikel/Apache-Click-Dann-eben-ohne-MVC-2970.html
[7] Wikipedia *Apache Click*, http://en.wikipedia.org/wiki/Apache_Click

Kurzbiografie

Rustam Khakimov ist als Senior Software-Entwickler und Consultant für die MATHEMA Software GmbH tätig. Seit 2003 beschäftigt er sich mit Design und Entwicklung von verteilten Systemen. Sein Schwerpunkt liegt dabei auf dem Einsatz von Java EE-, Web- und Rich Client-Technologien.

Des Programmierers kleine Vergnügen

Fakultativ

von Michael Wiedeking

Die Fakultätsfunktion läuft einem im Rahmen der Ausbildung eigentlich immer über den Weg.

Sie scheint sich dazu zu eignen, einem sowohl die Rekursion als auch die Iteration näher zu bringen. Aber leider hinterlässt sie dann den Eindruck, dass es getan wäre und man diese so programmieren müsse. Aber weit gefehlt.

Die Fakultätsfunktion stammt aus der Kombinatorik. Sie gibt nämlich an, wie viele Möglichkeiten man hat, unterscheidbare Kugeln aus einer Urne zu entnehmen, ohne dass man sie wieder zurücklegt. Hat man etwa eine Urne mit fünf Kugeln, so hat man beim ersten Mal fünf Möglichkeiten, eine Kugel zu ziehen. Beim nächstem Mal sind ja nur noch vier in der Urne, also hat man nur noch ebenso viele Möglichkeiten. Dann drei, dann zwei und schließlich bleibt nur noch eine Kugel bzw. Möglichkeit.

Die Gesamtzahl der Möglichkeiten bei n Kugeln ergibt sich dann aus dem Produkt der jeweiligen Möglichkeiten und das nennen wir die Fakultät *n!*. Das Schöne an der Fakultät ist, dass sie sich so einfach beschreiben lässt. Für positives, ganzzahliges n definiert man also:

$$1! = 1$$
$$n! = n \cdot (n-1)!$$

Damit ist etwa $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$. Diese rekursiv definierte Funktion[1] lässt sich dann relativ einfach „natürlich“ rekursiv programmieren:

```
int fac(int n) {
  if (n == 1) {
    return 1;
  } else {
    return n * fac(n – 1);
  }
}
```

So weit so gut. Leider wächst die Fakultätsfunktion enorm schnell, sodass bereits 17! bei einem 32-Bit-*int* zu einem Überlauf führt. Die Funktion wächst so schnell, dass hier auch eine Verdopplung der Wortbreite kaum Abhilfe schaffen kann: ein 64-Bit-*long* läuft schon bei 21! über.

Um die Fakultät sinnvoll berechnen zu können, bedarf es also einer deutlich größeren Zahl, idealerweise einer, die nur durch den Speicher im Rechner begrenzt ist. Soll etwa 1 000 000! berechnet werden, so ist das eine Zahl mit über fünfeinhalb Millionen Dezimalziffern. Das ist eine ziemlich große Zahl und – bevor es irgendwelche Einwände gibt – sie wird tatsächlich auch gelegentlich gebraucht.

C# und *Java* beispielsweise bieten mit *BigInteger* eine passende Zahl, die beliebig groß werden kann. Der Code bleibt dabei im Prinzip unverändert, jetzt muss lediglich dieser neue Datentyp verwendet werden.

```
BigInteger fac(BigInteger n) {
  if (n.equals(BigInteger.ONE)) {
    return BigInteger.ONE;
  } else {
    return n.multiply(
      fac(n.subtract(BigInteger.ONE))
    );
  }
}
```

Wegen der anderen Schreibweise[2] ist die Funktion nicht mehr ganz so gut zu lesen, aber das Prinzip ist immer noch gut erkennbar. Allerdings lässt sich 10519! bei mir schon nicht mehr berechnen, denn dann gibt es einen *Stack*-Überlauf. Das ließe sich sicherlich irgendwie beheben, aber alleine die Tatsache, dass es diese Grenze gibt, macht doch nervös.

Eine Lösung dieses Problems wäre eine *Tail*-Rekursion, die vom *Compiler* dahingehend optimiert werden kann, dass der rekursive Aufruf einfach durch einen Sprung ersetzt werden kann. Das würde den Stack nicht mehr belasten und im Prinzip wie eine Schleife funktionieren.

[1] Aus praktischen Gründen wird normalerweise auch noch $0! = 1$ definiert, aber das soll bei unseren Betrachtungen keine Rolle spielen – lässt es sich doch durch einen trivialen Test implementieren.

[2] Leider gibt es die Operatoren-Schreibweise nur in C#.

```
private BigInteger fac(
  BigInteger accu, BigInteger k
) {
  if (k.equals(BigInteger.ONE)) {
    return accu;
  } else {
    return fac(
      accu.multiply(k),
      k.subtract(BigInteger.ONE)
    );
  }
}

BigInteger fac(BigInteger n) {
  return fac(n, BigInteger.ONE);
}
```

Hier werden zwei Parameter betrachtet: das sich reduzierende *n* und der Akkumulator, der das aktuelle Produkt bis zum jeweiligen *n* enthält. Da der rekursive Aufruf der letzte Befehl ist und dieser keine „Hilfe“ mehr aus der Funktion benötigt, kann dieser wie beschrieben durch den Sprung ersetzt werden. In Java nutzt das leider nichts, weil die *Java Virtuelle Maschine* diese *Tail-Calls* nicht unterstützt; in C# sollte das allerdings funktionieren.[3]

In Java müssen wir die *Tail*-Rekursion also von Hand in eine Iteration umwandeln. Das bekommen wir sozusagen geschenkt, weil wir die Rekursion einfach durch eine Schleife ersetzten, um den Sprung zu vollziehen, wobei die Bedingung zur Abbruchbedingung der Schleife wird.

```
private BigInteger fac(BigInteger accu, BigInteger k) {
  while (!k.equals(BigInteger.ONE)) {
    accu = accu.multiply(k);
    k = k.subtract(BigInteger.ONE);
  }
  return accu;
}

static BigInteger fac(BigInteger n) {
  return fac(BigInteger.ONE, n);
}
```

Mit der Schleife brauchen wir natürlich die Hilfsfunktion nicht mehr. Dann darf der Zähler natürlich noch ein *int* sein. Darüber hinaus kann man noch anstatt den Zähler zu vermindern, diesen auch erhöhen. Das hat insbesondere den Vorteil, dass die Produkte zu Beginn nicht so groß sind, was sich schon merklich auf die Performanz auswirkt. So benötigt das hundertmalige Berechnen von 5000! absteigend 33,11 ms, während aufsteigend nur noch 29,18 ms benötigt wurden.

[3] In Scala beispielsweise funktioniert das auch, wenn die Funktion entsprechend annotiert und der Compiler die *Tail*-Rekursion erkennen und umsetzen kann.

```
BigInteger fac(int n) {
  BigInteger accu = BigInteger.ONE;
  for (int i = 2; i <= n; i++) {
    accu = accu.multiply(BigInteger.valueOf(i));
  }
  return accu;
}
```

Das ist wieder einmal ein Beispiel dafür, dass etwas signifikant von der Reihenfolge abhängen kann, ohne das Ergebnis zu beeinflussen. Sind A und B zwei 100-stellige Zahlen und c eine 10-stellige Zahl, so benötigt $(A \cdot B) \cdot c$ deutlich mehr Zeit als $A \cdot (B \cdot c)$. Da eine Multiplikation im naiven Ansatz einen quadratischen Aufwand bezüglich des Maximums der Längen beider Operanden hat, sind hier $100^2 = 10\,000$ Operationen nötig um $(A \cdot B)$ zu berechnen. Das Ergebnis ist dann eine 200-stellige Zahl. Die Multiplikation mit c führt ebenso zu quadratischem Aufwand, diesmal also $200^2 = 40\,000$ Operationen, zusammen sind es 50 000 Operationen.

Nimmt man die andere Variante, berechnet man zunächst $(B \cdot c)$ und erhält damit eine nur 110-stellige Zahl. Multipliziert man nun dieses Produkt mit A, so werden hier nur $110^2 = 12\,100$ Operationen benötigt. Damit stehen für $A \cdot (B \cdot c)$ lediglich 22 100 Operationen den 50 000 für $(A \cdot B) \cdot c$ gegenüber.

Zum Glück ist der Aufwand für Multiplikationen deutlich kleiner.[4] Es ändert aber nichts an der Tatsache, dass eine sorgfältige Auswahl des Algorithmus signifikanten Einfluss auf die Perfomanz hat. Um den Unterschied noch einmal zu verdeutlichen, sei an dieser Stelle noch erwähnt, dass die 100-malige Berechnung von 10 000! aufwärts nur 117,3 ms benötigt, während abwärts schon 141,5 ms benötigt werden. Dagegen benötigt die einmalige (!) Berechnung von 100 000! aufwärts 16,34 s und abwärts 18,04 s (ja Sekunden, nicht Millisekunden).

Das Vergnügen hat übrigens noch gar nicht richtig angefangen, wenngleich dies ja schon recht vergnüglich war. Schließlich will man ja auch 1 000 000! berechnen können, und das ist mit dem aktuellen Algorithmus nicht gut genug möglich. Unter mit oben vergleichbaren Bedingungen benötigt diese Berechnung nämlich mehr als 34 Minuten. Das nächste Mal gibt es zum Thema Fakultät also noch ein bisschen mehr, und dann sollte es gelingen, die selbe Aufgabe auch unter vier Minuten (!) zu erledigen.

[4] Es gibt eine Algorithmenfamilie, die für die Multiplikation $x \cdot y$ mit $l = \min\{\text{Wortlänge}(x), \text{Wortlänge}(y)\}$ statt der $O(l^2)$ nur $O(l \cdot \log l \cdot \log\log l)$ benötigt – eine unvorstellbare Beschleunigung.

Deutsch für Informatiker

Eselsbrücken

von Alexandra Specht

Wenn man mit sich mit einer Sprache beschäftigt, muss man auch mit deren Tücken zurecht kommen. Damit man sich gelegentlich etwas leichter mit den Besonderheiten tut, gibt es viele Eselsbrücken. Und diese gibt es natürlich nicht nur für fremde Sprachen, sondern auch für die eigene.

Die Eselsbrücke, die mir am häufigsten in den Sinn kommt, ist „Wer nämlich mit *h* schreibt ist dämlich". Es ist nicht so, dass ich mir das in den letzten Jahrzehnten nicht habe merken können, aber jedes Mal wenn ich „nämlich" schreibe, geht mir dieser Satz durch den Kopf. Vermutlich bin ich entsprechend konditioniert. Von dieser Eselsbrücke gibt es allerdings noch eine Variante, die mir bisher unbekannt war: „Wer nämlich und ziemlich mit *h* schreibt, ist nämlich ziemlich dämlich".

Ähnlich geht es mir, wenn ich „Geburtstag" schreibe, denn dann muss ich immer an den Spruch „Trenne nie *s*-*t*, denn es tut ihm weh!" denken. Vom dem gibt es noch die etwas martialischere Alternative „*s* und *t* wird nie getrennt, auch wenn das ganze Schulhaus brennt". Auch diese war mir unbekannt, was vielleicht dem Umstand geschuldet ist, dass ich auf einer Klosterschule war.

Jetzt hat diese Regel im Zuge der neuen Rechtschreibung ihre Bedeutung verloren – wenngleich es mir immer noch schwer fällt, *s* und *t* zu trennen. Aber bezüglich der neuen Regeln bin ich auch fündig geworden: „Trenne forsch das *s* vom *t*, denn das ist jetzt ganz ok. Trenne nie das *c* vom *k*, denn die beiden sind ein Paar!"

Allgemeine Hinweise zur neuen Trennung gibt es natürlich auch:

Trenne bitte doch nur dort,
wo Sprechpausen sind im Wort.
S und t darfst du trennen,
Vokale nicht alleine nennen.
Trennst du s-c-h, c-h, c-k
sträubt sich mir das Nackenhaar.

Apropos neue Rechtschreibung: Ich meine mich daran zu erinnern, dass Harry Rowohlt mal als Reformgegner gesagt hat, „Wer liebhaben auseinander schreibt, der ist es nicht wert, dass man ihn lieb hat". Das habe ich allerdings wirklich nur gehört und habe deswegen keine Ahnung, wie ich diesen Satz als Zitat aufschreiben soll. Leider konnte ich das auch nicht verifizieren, wollte es aber, weil es mir so gut gefällt, nicht unerwähnt lassen.

Ein Freund machte mich während des Studiums mehrfach darauf aufmerksam, dass ich wohl gelegentlich ein „zu" wegoptimierte. „Wer brauchen ohne *zu* gebraucht, braucht *brauchen* überhaupt nicht zu gebrauchen!" Auch in diesem Fall gehe ich davon aus, dass diese Optimierung der Gegend geschuldet ist, aus der ich komme. Wobei mich die Tatsache, dass es diese Eselsbrücke gibt, dahingehend beruhigt, dass ich nicht der einzige bin, der sie braucht – was die Sache natürlich an und für sich nicht besser macht.

Zur Rechtschreibung habe ich noch einige Eselsbrücken gefunden, die mir sämtlich unbekannt waren: „Nach *l*, *n*, *r* das merke ja, steht nie *t*-*z* und nie *c*-*k*", „Nach *a*, *e*, *i*, *o*, *u* schreib ein *t*-*z* und ein *c*-*k* dazu. Nimm die Regel mit ins Bett: Nach *ei*, *au*, *eu* steht nie *t*-*z*!" Allerdings habe ich auch „Nach *l*, *m*, *n*, *r* …" gefunden und bin jetzt natürlich dementsprechend verunsichert.

„Wenn wider nur dagegen meint – dann ist das *e* dem *i* stets Feind! Wenn wieder nur noch einmal meint – dann sind dort *i* und *e* vereint!" löst ein Problem, dass man erfahrungsgemäß frühestens in der Mittelstufe hat, derweil „Den Tiger schreib mit langem *i*, jedoch mit *i*-*e* schreib ihn nie!" und „Doppel-*a*, das ist doch klar, sind in Waage, Haar und Paar!" schon in der Grundschule ihren Zweck erfüllen können.

Im Zusammenhang mit der Groß-/Kleinschreibung kannte ich nur sich nicht reimende Eselsbrücken: „Nach *vom*, *zum* oder *beim* schreib niemals klein", „Endet ein Wort auf *-ung*, *-heit*, *-keit*, *-schaft* oder *-nis*, die Großschreibung sei Dir gewiss" und „Sei nicht dumm und merk dir bloß: Namenwörter schreibt man groß!"

Bei *das(s)* mit einfachem oder mit scharfem *s*, wobei es ja heute nicht mehr ganz so scharf ist, habe ich zwei

reimende Varianten gefunden: „Das *s* bei das muss einfach bleiben, wenn du dafür kannst welches schreiben“ oder „Das *s* in *das* muss einsam bleiben, kannst du auch *dieses* oder *welches* schreiben“. In der mir bekannten Version hieß es sogar „*dieses*, *jenes*, *welches*“.

Es gibt sogar etwas zum Thema Gänsefüßchen: „Da, wo man redet, sagt und spricht, vergiss die kleinen Zeichen nicht“. Hier hätte ich mir vor Jahren noch eine Eselsbrücke gewünscht, die mich daran erinnert hätte, ob ich im Super-User-Modus der Sun-Terminals mit Display Postscript kleine Sechsen ‘ oder kleine Neunen ’ verwenden musste, um einen Ausdruck in der Shell korrekt zu quoten, wenn Sie wissen, was ich meine.

Im Zusammenhang mit der Grammatik bin ich auch fündig geworden. Mir gänzlich unbekannt, kann man diese Eselsbrücke von vielen erfahren, die Deutsch als Fremdsprache kennen:

An, auf, hinter, neben, in,
über, unter, vor und zwischen
stehen mit dem 4. Fall,
wenn man fragen kann „wohin“?
Mit dem 3. steh’n sie so,
dass man nur kann fragen „wo“?

Es gibt noch viel mehr von diesen Eselsbrücken, die sich mehr oder weniger gut reimen und metrisch oft eine Katastrophe sind. Ob sie für einen routinierten Schreiber noch eine Bedeutung haben, ist fraglich, aber für einen Lernenden sind sie sicher eine große Hilfe, um Unsicherheiten zu vermeiden. Und irgendwie sind diese kleinen Reime viel eingängiger als die entsprechende Regel.

Übrigens kann man die eine oder andere Eselsbrücke auch singen. Falls Sie Probleme mit Präpositionen haben und die Melodie von Frère Jacques (Bruder Jakob) kennen, können Sie sie sich folgendes merken:

aus bei mit nach
aus bei mit nach
seit von zu
seit von zu
immer mit dem Dativ
immer mit dem Dativ
gegenüber auch
gegenüber auch

Und wenn sie musikalisch genug sind, können Sie das dann auch im Kanon singen – aber nicht alleine.

Kaffeesatz

Das Unsuchbare suchen

von Michael Wiedeking

Stellen Sie sich vor, Sie gehen auf eine Bushaltestelle zu, zücken Ihr Handy und es sagt Ihnen, ob da jetzt ein Bus kommt. Oder Sie wollen zum Metzger und Ihr Telefon sagt Ihnen, wo gerade die kürzeste Schlange wartet.

Bekanntermaßen wissen wir sehr gut darüber Bescheid, was wir wissen. Bei dem was wir wissen, kennen wir sehr gut unsere Lücken, also die Dinge, über die wir nichts wissen. Leider können wir keine Aussagen darüber machen, was wir überhaupt nicht wissen. Schon Sokrates sagt man nach, er habe gewusst, dass er nichts weiß. Wobei eine kurze Suche bei Google zu dem Ergebnis führt, dass er eigentlich nur *nicht* weiß, nicht *nichts* weiß [1].

Aber Google will es jetzt wissen. Google will uns auch nach den Dingen suchen lassen, die wir noch nicht wissen. Google will wissen, wonach man den ganzen Tag sonst noch sucht, um ihr Angebot den Bedürfnissen mobiler Anwender anzupassen [2]. So hat Google in einer Studie 150 Personen zufällig acht mal täglich danach gefragt, wonach sie gerade suchen. Damit wollen sie herausfinden, ob sie Fragen beantworten können, von denen bisher noch keiner auf die Idee gekommen ist, diese online zu stellen.

Die Idee dahinter klingt eigentlich recht verlockend: Auf Basis der aktuellen Position werden sämtlich in Frage kommenden Informationen schon mal vorbereitet, so dass sie bei einem Abruf sofort zur Verfügung stehen. Dazu gehören dann nicht nur Informationen darüber, ob und wann der nächste Bus fährt, sondern auch, ob es beim Bäcker eine lange Warteschlange gibt.

Dabei darf man natürlich vergessen, dass Google nicht etwa die Welt verbessern will, sondern nur ganz schnöde ans Geldverdienen denkt. So werden wir in diesem Fall sicherlich auch beim Betreten der Bäckerei darüber informiert, dass es bei dem Bäcker auf der anderen Straßenseite heute drei Krapfen für den Preis von zwei im Angebot gibt.

Zu wissen, wann der Bus kommt, scheint mir ja noch nützlich zu sein. Und hierfür brauche ich Google nicht, dafür genügt mir die App des lokalen Verkehrsunternehmens. Mir graust es bei dem Gedanken, dass Google dann nicht nur weiß, wann und wo ich überall gewesen bin, sondern auch noch, wo ich überall einkaufen gehe, damit mich deren Konkurrenten mit Werbung überhäufen können.

Dabei darf auch nicht vergessen werden, dass es im Falle der Warteschlange einer entsprechenden Infrastruktur bedarf, mit der man eine solche überhaupt feststellen kann. Vielleicht eine von Google gestiftete Kamera, deren Bilder dann auch direkt an Google übertragen werden. Oder das Warenwirtschaftssystem, das direkt an Google gekoppelt ist, damit ich gleich weiß, ob das Sonderangebot beim Aldi um die Ecke noch da ist oder nicht.

Vielleicht kann ich dann ja auch nach neuen Mitarbeitern suchen und Google zeigt mir dann schon mal das Kurzprofil mit Bild (inklusive Beispiel-Code, absolvierten Projekten, Liste seiner Freunde und deren Bewertungen) von einem potenziellen Kandidaten, der in der selben Einkaufspassage herumlungert wie ich und zeigt mir dann noch seinen vermuteten Pfad unter Angabe des Ladens in dem ich – ausgehend von meiner aktuellen Position – am unverfänglichsten mit ihm ins Gespräch kommen kann.

Referenzen

[1] Wikipedia *Ich weiß, dass ich nichts weiß*
http://de.wikipedia.org/wiki/Ich_wei%C3%9F,_dass_ich_nichts_wei%C3%9F

[2] Simonite, Tom *How Google Plans to Find the UnGoogleable*
http://www.technologyreview.com/news/507451/how-google-plans-to-find-the-ungoogleable/

Kurzbiographie

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

User Groups

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

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

Java User Groups

Deutschland

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

Java User Group Saxony
Java User Group Dresden
http://www.jugsaxouy.de
Kontakt: Herr Torsten Rentsch (torsten@jugsaxony.de)
Herr Falk Hartmann (falk@jugsaxony.de)
Herr Kristian Rink (kristian@jugsaxony.de)

rheinjug e.V.
Java User Group Düsseldorf
Heinrich-Heine-Universität Düsseldorf
Universitätsstr. 1, 40225 Düsseldorf
http://www.rheinjug.de
Kontakt: Herr Heiko Sippel (info@rheinjug.de)

ruhrjug
Java User Group Essen
Glaspavillon Uni-Campus
Universitätsstr. 12, 45127 Essen
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 asc-Dienstleistungs GmbH
Ehrengard-Schramm-Weg 11, 37085 Göttingen
http://www.java.de (office@java.de)

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

JUG Karlsruhe
Java User Group Karlsruhe
Universität Karlsruhe, Gebäude 50.34
Am Fasanengarten 4, 76131 Karlsruhe
http://jug-karlsruhe.de
jug-karlsruhe@gmail.com

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

jugm
Java User Group München
Jupiterweg 8, 85586 Poing
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 Alexandra Specht
(alexandra.specht@jug-n.de)

JUG Ostfalen
Java User Group Ostfalen
(Braunschweig, Wolfsburg, Hannover)
Siekstraße 4, 38444 Wolfsburg
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
Schönaicherstraße 3, 70597 Stuttgart
http://www.jugs.org
Kontakt: Herr Dr. Micheal Paus (mp@jugs.org)
Herr Hagen Stanek (hs@jugs.org)

Schweiz

JUGS
Java User Group Switzerland
Postfach 2322, 8033 Zürich
http://www.jugs.ch (info@jugs.ch)
Kontakt: Frau Ursula Burri

.Net User Groups

Deutschland

.NET User Group OWL
http://www.gedoplan.de/cms/gedoplan/ak/ms_net
c/o GEDOPLAN GmbH
Stieghorster Str. 60, 33605 Bielefeld

.Net User Group Bonn
.NET User Group "Bonn-to-Code.Net"
Langwartweg 101, 53129 Bonn
http://www.bonn-to-code.net (mail@bonn-to-code.net)
Kontakt: Herr Roland Weigelt

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

.net Usergroup Frankfurt
c/o Thomas Sohnrey, Agile IService
Mendelssohnstrasse 80, 60325 Frankfurt
http://www.dotnet-ug-frankfurt.de
Kontakt: Herr Thomas 'Teddy' Sohnrey
(thomas.sohnrey@gmx.de)

.NET DGH
.NET Developers Group Hannover
Landwehrstraße 85, 30519 Hannover
http://www.dotnet-hannover.de
Kontakt: Herr Friedhelm Drecktrah
(friedhelm@drecktrah.de)

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

DNUG-Köln
DotNetUserGroup Köln
Goldammerweg 325, 50829 Köln
http://www.dnug-koeln.de
Kontakt: Herr Albert Weinert (info@der-albert.com)

.Net User Group Leipzig
Brockhausstraße 26, 04229 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
Sachsenstr. 24, 26121 Oldenburg
http://www.dotnet-oldenburg.de
Kontakt: Herr Hilmar Bunjes
(hilmar.bunjes@dotnet-oldenburg.de)
Yvette Teiken (yvette.teiken@dotnet-oldenburg.de)

.NET User Group Paderborn
c/o Net at Work Netzwerksysteme GmbH,
Am Hoppenhof 32, 33104 Paderborn
http://www.dotnet-paderborn.de
(raacke@dotnet-paderborn.de)
Kontakt: Herr Mathias Raacke

.Net Developers Group Stuttgart
Tieto Deutschland GmbH
Mittlerer Pfad 2, 70499 Stuttgart
http://www.devgroup-stuttgart.de
(GroupLeader@devgroup-stuttgart.de)
Kontakt: Frau Catrin Busley

.net Developer-Group Ulm
c/o artiso solutions GmbH
Oberer Wiesenweg 25, 89134 Blaustein
http://www.dotnet-ulm.de
Kontakt: Herr Thomas Schissler (tschissler@artiso.com)

Österreich

.NET Usergroup Rheintal
c/o Computer Studio Kogoj
Josefgasse 11, 6800 Feldkirch
http://usergroups.at/blogs/dotnetusergrouprheintal/default.aspx
Kontakt: Herr Thomas Kogoj (thomas@kogoj.com)

.NET User Group Austria
c/o Global Knowledge Network GmbH,
Gutheil Schoder Gasse 7a, 1101 Wien
http://usergroups.at/blogs/dotnetusergroupaustria/default.aspx
Kontakt: Herr Christian Nagel (ug@christiannagel.com)

Software Craftsmanship Communities

Deutschland

Softwerkskammer – Mehrere regionale Gruppen unter einem Dach, http://www.softwerkskammer.de

MATHEMA

meet the experts of enterprise infrastructure

Spring Framework
JavaEE ganz ohne EJB
9. – 11. Januar 2013,1.315,– € (zzgl. 19 % MwSt.)

Einführung in die Unified Modeling Language 2.x
Darstellung komplexer OO-Modelle mit der UML
21. – 22. Januar 2013, 835,– € (zzgl. 19 % MwSt.)

Programmierung mit Java
Einführung in die Java-Technologie und die Programmiersprache Java
14. – 18. Januar 2013, 1.645,– € (zzgl. 19 % MwSt.)

Enterprise JavaBeans (EJB)
Enterprise JavaBeans im Detail
14. – 18. Januar 2013, 1.870,– € (zzgl. 19 % MwSt.)

Einführung in XML
Strukturierte Darstellung von Dokumenten und Daten
4. – 5. Februar 2013, 835,– € (zzgl. 19 % MwSt.)

Testkonzepte
Software-Tests zur kontinuierlichen Sicherung der Qualität
11. – 13. Februar 2013,
1.315,– € (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

Software-Entwickler (m/w)
Software-Architekt (m/w)

Arbeiten Sie gerne selbstständig, motiviert und im Team? Haben Sie gesunden Ehrgeiz und Lust, Verantwortung zu übernehmen?

Wir bieten Ihnen erstklassigen Wissensaustausch, ein tolles Umfeld, spannende Projekte in den unterschiedlichsten Branchen und Bereichen sowie herausfordernde Produktentwicklung.

Wenn Sie ihr Know-how gerne auch als Trainer oder Coach weitergeben möchten, Sie über Berufserfahrung mit verteilten Systemen verfügen und Ihnen Komponenten- und Objektorientierung im .Net- oder JEE-Umfeld vertraut sind, dann lernen Sie uns doch kennen.

Wir freuen uns auf Ihre Bewerbung!

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

Das Allerletzte

```
public class StubFactoryFactoryProxyImpl
  extends StubFactoryFactoryDynamicBase
{
  public PresentationManager.StubFactory makeDynamicStubFactory(
    PresentationManager pm,
    PresentationManager.ClassData classData,
    ClassLoader classLoader
  ){
    return new StubFactoryProxyImpl(classData, classLoader);
  }
}
```

Dies ist kein Scherz!
Dieser Code 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 Dezember.

Herbstcampus

Wissenstransfer par excellence

2.– 9. September 2013

in Nürnberg