KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

ISSN 1865-682X

01/2016

Jahrgang 9

KaffeeKlatsch

Das Magazin rund um Software-Entwicklung

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

abo@bookware.de

abonnieren.

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

Editorial

Unberechenbar

Eigentlich haben wir ja den allerbesten Job der Welt. Wir können den ganzen Tag beschäftigt sein, ja vielleicht sogar jahrelang, und das Ergebnis unserer Arbeit lässt sich immer bequem auf einem USB-Stick umhertragen.

Zugegeben, wir können außer dem Stick nichts herzeigen und haben auch so unsere Nöte, wenn wir erklären müssen, was wir eigentlich genau den ganzen Tag machen. Aber zumindest hat sich inzwischen herumgesprochen, dass einige, wenige von uns, es durch Programmieren zum Milliardär schaffen.

Unabhängig davon erlaubt es unsere Fähigkeit zu Programmieren, Probleme zu lösen, die keinen praktischen Nutzen haben. So versuchen wir Außerirdische zu entdecken, die wir tendenziell nicht besuchen können; Primzahlen zu finden, die so groß sind, dass man sie noch nicht einmal für eine Verschlüsselung verwenden könnte; oder π auf mehrere Billionen Nachkommastellen zu berechnen, falls man mal einen Kreis um die Milchstraße ziehen will.

Aber allen Unkenrufen zum Trotz gibt es ja doch auch wirklich nützliche Dinge, deren Berechnung lohnt. So wurde jetzt endlich, nach 2500 Jahren, die Frage beantwortet, wie viele gültige Stellungen auf einem Go-Brett mit 19 × 19 Feldern möglich sind:

208 168 199 381 979 984 699 478 633 344 862 770
286 522 453 884 530 548 425 639 456 820 927 419
612 738 015 378 525 648 451 698 519 643 907 259
916 015 628 128 546 089 888 314 427 129 715 319
317 557 736 620 397 247 064 840 935

Jetzt können viele wieder ruhig schlafen. Denn ohne dieses Wissen war man ja völlig aufgeschmissen.

Mich persönlich beschäftigt da noch ein anderes Problem, dass mir praktisch täglich zu schaffen macht: Auf wie viele Arten kann man n Damen auf einem überdimensionalen $n \times n$ Schachbrett so unterbringen, dass sie sich nicht gegenseitig bedrohen. Für den Anfang würde genügen, das für ein Brett mit $n = 27$ auszurechnen. So weit ich weiß, hat man das für $n = 26$ zuerst am 11. Juli 2009 herausgefunden:

22 317 699 616 364 044

Um diese Zahl herauszufinden, hat man an der Uni Dresden mit Hilfe speziell dafür angefertigter FPGAs am 14. Oktober 2008 mit dem Rechnen angefangen und war dann am besagten 11. Juli 2009 damit fertig.

Bei $n = 24$ haben übrigens die von *nQueens@Home* kläglich versagt. Die weltweit zusammengetragenen Teilergebnisse ergaben bei denen 226 732 487 925 864 Stellungen statt der erforderlichen 227 514 171 973 736. Vermutlich hat es da beim Zusammenzählen einen Überlauf gegeben, denn die Differenz beträgt exakt 781 684 047 872 = 0xB600000000 = $182 \cdot 2^{32}$. Ach ja, sind schon goldig unsere Klassiker.

Wir müssen uns bei all diesen Erfolgen aber keinesfalls Sorgen machen: Es gibt keine größte Primzahl, π hat unendlich viele zu erforschende Nachkommastellen und Außerirdische hat man ja auch noch nicht gefunden.

Ach, haben wir nicht einen wunderbaren Beruf!?

Ihr Michael Wiedeking
Herausgeber

Beitragsinformation

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

Beiträge

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

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

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

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

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

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

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

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

Leserbriefe

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

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

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

Werbung ist Information

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

- Stellenanzeigen
- Seminaranzeigen
- Produktinformation und -werbung

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

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

Abonnement

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

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

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

Copyright

Das Copyright des KaffeeKlatschs liegt vollständig bei der Bookware. Wir gestatten die Übernahme des KaffeeKlatschs in Datenbestände, wenn sie ausschließlich privaten Zwecken dienen. Das auszugsweise Kopieren und Archivieren zu gewerblichen Zwecken ohne unsere schriftliche Genehmigung ist nicht gestattet.

Sie dürfen jedoch die unveränderte PDF-Datei gelegentlich und unentgeltlich zu Bildungs- und Forschungszwecken an Interessenten verschicken. Sollten diese allerdings ein dauerhaftes Interesse am KaffeeKlatsch haben, so möchten wir diese herzlich dazu einladen, das Magazin direkt von uns zu beziehen. Ein regelmäßiger Versand soll nur über uns erfolgen.

Bei entsprechenden Fragen wenden Sie sich bitte per E-Mail an copyright@bookware.de.

Impressum

KaffeeKlatsch Jahrgang 9, Nummer 1, Januar 2016
ISSN 1865-682X
Bookware – eine Initiative der
MATHEMA Verwaltungs- und Service-Gesellschaft mbH

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

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

Inhalt

Editorial 3

Beitragsinfo 4

Inhalt 5

Jahreinhaltsverzeichnis 2016 19

User Groups 25

Werbung 27

Das Allerletzte 28

Artikel

Kommentar Kommentar

Böse Java Annotations 6

Entwicklung im Schatten

Ein Einblick ins Shadow DOM 8

Strukturprogramm

Log Daten als Input für Regressionstests 14

Kolumnen

Séparée

Des Programmierers kleine Vergnügen 18

Kommentar Kommentar

Böse Java Annotations 6
von Róbert Bräutigam

Java Annotationen wie *@Override* oder *@Test* sind inzwischen – auch losgelöst von Java Enterprise – unentbehrliche Werkzeuge, um in Java zu Entwickeln. Nicht alle Annotationen sind aber gleich, es gibt *hilfreiche*, *harmlose* und eine ganze Menge *böse* Annotationen!

Entwicklung im Schatten

Ein Einblick ins Shadow DOM 8
von Frank Goraus

Kennen Sie das Shadow DOM? Nie gehört? Das Shadow DOM ist eine noch recht junge Technologie und befindet sich derzeit noch in der Spezifikation. Dennoch ist bereits klar was es damit auf sich hat und es sind bereits große Teile implementiert. Dieser Artikel soll somit einmal aufzeigen, wozu man das Shadow DOM braucht, und die Grenzen des aktuell machbaren ausloten. Das ultimative Ziel ist es natürlich, die Technologie selbst für sich zu nutzen.

Strukturprogramm

Log Daten als Input für Regressionstests 14
von Florina Sas

Wenn Logdateien als Daten betrachtet werden können, dann werden sie zu wertvollen Informationen. Wie wäre es also, wenn wir die Logdateien nicht nur ab und zu für die Fehlersuche, sondern auch für das Durchführen von Regressionstests verwenden würden? Die Idee dahinter ist, die Logdateien der Produktionsumgebung in Regressionstests zu verwenden, um Fehler oder unbekannte Szenarien zu identifizieren. Dies ist vor allem bei einem neuen Release oder einer Änderung der Konfiguration wichtig. Dieser Artikel betrachtet das Thema Parsen von Logdateien, wenn die Logdateien strukturiert sind, um Input-Daten für Regressionstests vorzubereiten.

Kommentar Kommentar

Böse Java Annotations

von Róbert Bräutigam

Java Annotationen wie *@Override* oder *@Test* sind inzwischen – auch losgelöst von Java Enterprise – unentbehrliche Werkzeuge, um in Java zu Entwickeln. Nicht alle Annotationen sind aber gleich, es gibt *hilfreiche*, *harmlose* und eine ganze Menge *böse* Annotationen!

Wie kann ein Annotation *böse* sein?

Eine Annotation ist zumindest *harmlos*, wenn es keinerlei Wirkungen auf die Laufzeit hat. Das bedeutet, wenn die Annotation nicht da wäre, würde die Applikation exakt gleicht funktionieren, wie es beispielsweise bei *@Deprecated* der Fall ist.

Eine *hilfreiche* Annotation ist *harmlos*, bietet aber zusätzlich eine Funktion die zur Übersetzungszeit irgend einen statischen Fehlerfall verhindert, wie etwa *@Override*.

Alle anderen, also nicht-*harmlose* Annotationen, sind *böse*. Sie sind böse, weil sie Funktionalität, Bedeutung und Logik implizit machen. Sie verändern die annotierte „Unit" (also etwa die Klasse), die dadurch sehr schwierig zu testen sein wird.

Checker Framework

Hilfreiche Annotationen findet man z. B. in dem Checker Framework [1]. Es beinhaltet zusätzliche Annotation-Prozessoren die das Java-Typsystem erweitern und zusätzliche statische Prüfungen möglich machen.

Es kann beispielsweise mit den *@NotNull*, *@Nullable* Annotationen verfolgen, ob eine Variable *null* sein kann und durch einen Compiler-Fehler melden, wenn einer potentieller *null*-Wert einer Variable zugewiesen wird, die nicht *null* sein darf. *@NotNull* ist dabei Default, wodurch das ganze meistens auch ohne es hinzuschreiben funktioniert.

Es kann aber auch komplexere Fälle abdecken. Mit den *@Tainted* und *@Untainted* Annotationen kann man „verschmutzte" bzw. „nicht-verschmutzte" Werte im Typsystem unterscheiden ohne dafür zusätzliche Klassen zu bauen. So kann man etwa vermerken, dass direkte Werte von Benutzereingaben „verschmutzt" sind und die Business-Logik nur „nicht-verschmutzte" Werte empfangen darf:

```
public @Tainted String getUserInput() { ... }
public void doLogic(@Untainted String userInput) { ... }
```

Im diesen Fall wird folgender Code nicht kompilieren:

```
doLogic(getUserInput());
```

Obwohl beide Methoden mit String arbeiten, schaffen die Annotationen effektiv zwei neue Typen die inkompatibel sind. Man muss zuerst den User-Input „bereinigen", also etwa *Cross-site Scripting (XSS)* und *SQL Injection* ausfiltern, und dann *@Tainted* auf *@Untainted* casten.

Dasselbe funktioniert auch für verschlüsselte oder nicht-verschlüsselte Daten, validierte oder nicht-validierte Daten, usw.

Auch diese Annotationen sind harmlos, weil sie zur Laufzeit nichts machen; und sie sind hilfreich, weil sie verschiedenste Arten von Programmier-Fehlern verhindern können.

Die Enterprise Falle

Mit der Einführung der Java Enterprise Edition 5 hat Oracle leider *böse* Annotationen „legalisiert". In JEE 5 sind nämlich „Dependency Injection", „Persistence" und andere Aspekte ebenfalls durch Annotation steuerbar:

```
public class ShoppingCart {
  @EJB
  private InvoiceService invoiceService;
}
```

„*@EJB*" ist eindeutig *böse*. Falls man es nämlich löschen (oder ändern) würde, könnte die Klasse nicht mehr funktionieren. Was noch schlimmer ist, es würden sehr wahrscheinlich noch nicht einmal Unit-Tests fehlschlagen.

Das Problem war, dass Oracle die Komplexität aus Enterprise Java nicht mehr ausbauen konnte; sie haben also versucht diese so gut wie möglich zu verstecken. Ein *Plain Old Java Object* (POJO) mit Annotationen zu schreiben, scheint einfach zu sein, die Komplexität aber ist immer noch da, implizit versteckt. Das Problem ist also noch immer da. Es ist nicht modular, es ist nicht kompatibel mit normalem Java-Code und man kann es nicht mit anderen, beliebigen Technologien kombinieren.

Was sind die Alternativen?

Es gibt Entwickler die sagen, dass Konfiguration nicht in den Code gehört. Es klingt soweit logisch, dass beispielsweise die „*@EJB*"-Annotation nicht in die *ShoppingCart* - Klasse oben gehört. Wieso sollte die Klasse wissen was für ein *Dependency-Injection-Framework*[1] (DI) später zur Laufzeit benutzt wird? Sollte es für die *ShoppingCart* nicht egal sein?

Die Alternative muss aber nicht unbedingt eine externe Datei (beispielsweise in XML) sein. Die Konfiguration kann sehr wohl in Java geschrieben werden, es sollte eben nur nicht in die *ShoppingCart*-Klasse. Es kann ja eine eigene Klasse für die Initialisierung sein, ähnlich einer *Main*-Klasse, die nur eine *main*-Methode hat, um das System starten können. Solche „Gerüst-Klassen" können sehr wohl Teil der Applikation sein.

Das bezieht sich natürlich nicht nur auf DI, sondern auch auf JPA und andere APIs die *böse* Annotationen verwenden.

Wiederverwendbarkeit

Die Unterscheidung zwischen *bösen* und *harmlosen* Annotationen ist natürlich nicht willkürlich.

Was wäre wenn die *String*- oder *Date*-Klasse eine *@PostConstruct*-Methode, oder gar *@Inject*-Annotation enthielte? Die Klassen wären für Nicht-Enterprise-Applikationen unbenutzbar. Dann etwas mit *String* oder *Date* zu testen wäre ein Albtraum.

Zugegebenermaßen sind die für die Business-Logik relevanten Klassen vielleicht etwas komplizierter und haben mehr Abhängigkeiten; das entschuldigt aber nicht, Funktionalität hinter Annotationen zu verstecken und damit auch implizit irrelevante Frameworks einzubinden.

Enterprise Java versucht wiederverwendbare Business-Komponenten zu schaffen. Java ist aber auf Offenheit und Alternativen aufgebaut. Es wird immer eine signifikante Menge an Entwicklern geben, die Enterprise Java nicht nutzen oder nicht das „mainstreamige" Unit-Testing-Framework, die gerade angesagte IDE, das gehypte Logging-Framework usw. verwenden. Wiederverwendbarkeit in Java muss also bei technischer Unabhängigkeit anfangen; und das schließt *böse* Annotationen eindeutig aus, da wir die Laufzeitumgebung nicht unter Kontrolle haben.

Fazit

Annotationen sind ein wichtiges Werkzeug. Nicht alle sind aber gleich und es gibt auch viele *böse* Annotationen. Leider ist die Verwendung von *bösen* Annotationen durch Java Enterprise die Norm geworden, daher ist es sehr schwierig sie zu vermeiden.

Unser Aufgabe als Entwickler oder Architekten ist es, den Code richtig zu designen und diesbezügliche Entscheidungen sorgfältig abzuwägen. Dieser Artikel hilft vielleicht zu erkennen, dass manche Entscheidungen bezüglich der Annotationen, auch wenn sie Präzedenz haben, vielleicht doch nicht die besten sind.

[1] *Dependency Injection*: Abhängigkeitsauflösung

Referenzen

[1] checkerframework.org *The Checker Framework* http://checkerframework.org

Kurzbiografie

Robert Bräutigam (Robert.braeutigam.de) ist als Senior Consultant für MATHEMA Software GmbH tätig. Er ist seit 1999 als Entwickler und Architekt im Java Enterprise-Umfeld beschäftigt und interessiert sich für leichtgewichtige Lösungen im technischen Bereich sowie im Projektmanagement.

Entwicklung im Schatten

Ein Einblick ins Shadow DOM

von Frank Goraus

Kennen Sie das Shadow DOM? Nie gehört? Das Shadow DOM ist eine noch recht junge Technologie und befindet sich derzeit noch in der Spezifikation. Dennoch ist bereits klar was es damit auf sich hat und es sind bereits große Teile implementiert. Dieser Artikel soll somit einmal aufzeigen, wozu man das Shadow DOM braucht, und die Grenzen des aktuell machbaren ausloten. Das ultimative Ziel ist es natürlich, die Technologie selbst für sich zu nutzen.

Im Laufe der letzten Jahre hat sich viel getan im Web. Die einst für Skript-Kiddies verpönte Umgebung hat sich zur Ernst zu nehmenden Entwicklungslandschaft gewandelt. JavaScript erlebte einen richtiggehenden Hype und ist von der Sprache, die Web-Seiten so bunt und blinkend macht, hin zur populärsten Programmiersprache des abgelaufenen Jahres aufgestiegen. Web-Seiten sind dank HTML5, CSS3 und der standardkonformität der Browser nicht mehr eine Ansammlung diverser Hacks.

Und dennoch ist nicht alles so schön und glitzernd wie man vermuten mag. Unter der Haube schlummern noch veraltete Technologien, sowie Denk- und Programmierweisen, die an der Performanz so manch einer Web-Anwendung nagen. Doch es gibt auch genug neue Entwicklungen, die dem entgegen wirken. Eine davon ist der Umstand, dass es neben dem gemein hin bekannten DOM noch ein zweites, das so genannte Shadow DOM, gibt. Doch was bringt das und wie kann man sich dieses zu Nutze machen?

Shadow DOM? Nie gehört...

Müsste man das Shadow DOM in einem Satz erklären, würde so etwas herauskommen wie „Naja, das ist der Teil von HTML-Elementen, der auch gerendert wird, aber der im DOM nicht zu sehen ist.“ Doch das wird dem bei weitem nicht gerecht, darum eins nach dem anderen.

Eine der Grundursachen warum JavaScript früher von Enterprise-Entwicklern immer belächelt wurde, ist seine offene Struktur. Man konnte alles irgendwie irgendwo hinschreiben und es funktionierte auch irgendwie. Denn es vereint Konzepte von prozeduraler, objektorientierter und funktionaler Programmierung unter einem Hut. Doch was fehlt ist eine klare Struktur, und so kommt meist verteilter Spaghetti-Code raus.

Jetzt liegt es an der Disziplin der Entwickler, dass man sich diese Struktur erarbeitet und so klare Schnittstellen zwischen dem Code und seinen Consumern schafft. Denn speziell eines der Grundprinzipen der OOP, die Datenkapselung, gibt es von Haus aus nicht. Es gibt Pattern für JavaScript, wie man durch geschickte Nutzung von *Closures* und dem *Function Scope* diese Kapselung schaffen kann. Doch für das Document selbst? Da gab es mal das *iframe*, doch das ist schon aus Sicherheitsgründen seit dem letzten Jahrtausend wieder „out“.

Die Entwicklung ist zum Glück nicht stehen geblieben, und im Zuge der Entwicklung der HTML5-Komponenten hatten die Browser-Hersteller eine geniale Idee. Was nach außen hin zum Beispiel nur ein *video*- oder *audio*-Element ist, ist intern dennoch eine Ansammlung von HTML-Elementen, die zusammen gerendert das eigentliche Element ergeben. Und hier schließt sich die Brücke zum einleitenden Satz. Nach außen im DOM ist nur das entsprechende <*audio* /> oder <*video* /> Tag zu sehen. Der Rest der Implementierung versteckt sich im Shadow DOM.

Zugriff auf das Shadow DOM

Es gibt also diese versteckte Implementierung, doch was heißt das jetzt für mich als Entwickler? Schauen wir uns erstmal ein paar Details an.

In der kleinen Testseite aus Abbildung 1 sieht man, dass mit dem *-Selektor im CSS versucht wurde, jeglichem HTML-Element ein *padding* zu geben. Das funktioniert auch für alle Elemente im DOM (*html*, *body*, *div* und *audio*), die Steuerelemente des *audio*-Elements sind davon jedoch unbeeindruckt. Warum? Prinzipiell kann man erstmal nicht auf das Styling dieser Unterelemente im Shadow DOM zugreifen und somit auch nicht deren Design zerstören. Stellt man sich das ganze als zwei Ebe-

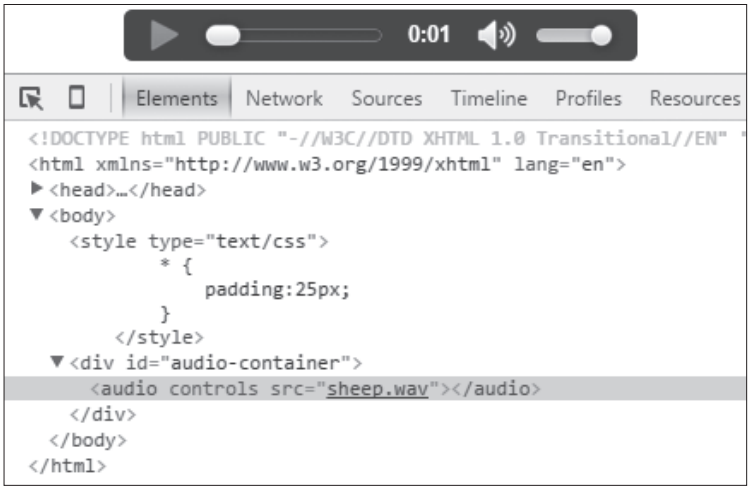


Abbildung 1

nen vor, so kann also das Stylesheet auf die erste Ebene – das DOM – Einfluss nehmen. Es kann jedoch nicht auf die zweite Ebene – das Shadow DOM – gelangen und dort ebenso aktiv werden.

Genauso verhält es sich mit JavaScript-Ereignissen (Events). Würde man an das *div*- oder auf das *audio*-Element einen Click-Event-Listener binden und nach dem Target des Events fragen, so würde man als Antwort nur das *div*- oder das *audio*-Element bekommen und nie eines der internen Elemente aus dem Shadow DOM. Allerdings bemerkt man hier bereits einen Unterschied im Verhalten allein zwischen Chrome und Firefox. In ersterem bekommt man kein Click-Event, sofern man auf den internen Bereich des *audio*-Elements klickt, bei zweiterem schon (siehe Abbildung 2). Somit scheint im Chrome das interne Click-Event nicht weiter propagiert zu werden, während der Firefox es tut. (Kleine Anmerkung dazu: Man kann im Chrome schon auch das Click-Event auf das *audio*-Element registrieren, dann allerdings nur in den äußeren Bereichen des Box-Modells: *margin*, *border*, *padding*; Im *Content*-Bereich befindet sich der Shadow DOM Code und da gilt vorherige Feststellung.)

Es scheint also eine unüberwindbare Grenze zwischen DOM und Shadow DOM zu geben, die weder im DOM, CSS oder JavaScript durchbrochen werden kann.

Aber Moment mal, es hieß doch, dass man sich das als Entwickler zu Nutze machen könne? Wie soll das gehen, wenn es da eine unüberwindbare Grenze gibt? Nun, Grenzen sind da um überwunden zu werden. Man muss einfach nur anders an die Sache heran gehen.

Denken wir nochmal zurück an die Sache mit der Kapselung in OOP. Ein wichtiger Punkt dieses Prinzips ist es die eigene Implementierung zu verstecken und nach außer hin nur eine wohldefinierte Schnittstelle anzubieten. Und genau das ist auch hier der Fall.

Abbildung 2

Im Falle von CSS passiert dies über Pseudo-Klassen, welche auf dem Element existieren können. Mit ein wenig Web-Recherche [1] stößt man schnell auf diese Pseudo-Selektoren. (Die natürlich alle – noch – Browser-spezifisch sind, da die Implementierung den entsprechenden Browser-Herstellern obliegt; eine nette API-Dokumentation oder offizielle Dokumentation lassen sie jedoch noch missen.) Versuchen wir als einfaches Beispiel etwa die Hintergrundfarbe des Audio-Players im Chrome zu verändern. Allen Pseudo-Selektoren gemein ist, dass sie mit zwei vorangestellten Doppelpunkten referenziert werden. Und so lautet unsere CSS-Angabe wie folgt:

```
audio::-webkit-media-controls-panel {
  background-color: lightblue;
}
```

Abbildung 3

Gegen meine Versuche, die Steuerelemente noch entsprechend hübsch einzufärben, hat sich der Chrome zumindest teilweise erfolgreich gewehrt. So gelang es mir zwar Einfluss auf die Größen und Hintergrundfarbe zu nehmen, jedoch steht das angezeigte Steuerelement im direkten Zusammenhang mit der Angabe *-webkit-appearance*, welche ein Set von definierten Controls beinhaltet, auf die man keinen (optischen) Einfluss hat. Wie man in Abbildung 4 sieht, kann man diese jedoch auch nach Belieben austauschen, so dass statt dem *Play Control* nun das *Mute Control* zu sehen ist.

Wer die Abbildung aufmerksam betrachtet hat, wird auch bemerkt haben, dass hier das *Input*-Element für den *Play*-Button hervorgehoben und im DOM (unterhalb eines ominösen *#shadow-root*) zu sehen ist. (Ebenso sieht man hier auch die Pseudo-Selektoren als Attribut *pseudo* an den verschiedenen HTML-Elementen der internen Implementierung.) Das ist eines der netten Features der Chrome Entwicklertools. Hier gibt es nämlich in den Einstellungen unter *General* die Option „Show user agent shadow DOM", welche bei Aktivierung diesen tiefen Einblick in das interne Shadow DOM erlaubt und somit die Entwicklung erleichtert.

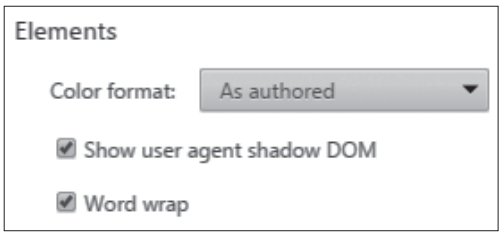


Abbildung 5

Leider sucht man solch ein Feature bei Firefox' eigenen Entwicklertools und Firebug (noch) vergeblich. Es ist jedoch schon auf der Wunschliste von Firebug, bedarf aber wohl noch etwas Vorarbeit der Firefox-Entwickler [5].

Bevor wir zu dem Teil kommen, in dem wir selbst versuchen auf dem Shadow DOM zu arbeiten, vollenden wir noch schnell die Kür und schauen uns den JavaScript-Teil zu solchen Shadow-Elementen an. Als Beispiel soll wieder das Audio-Element genügen. Oben haben wir bereits ausprobiert wie sich ein einfacher Click-Event-Handler verhält, doch eigentlich interessiert uns viel mehr

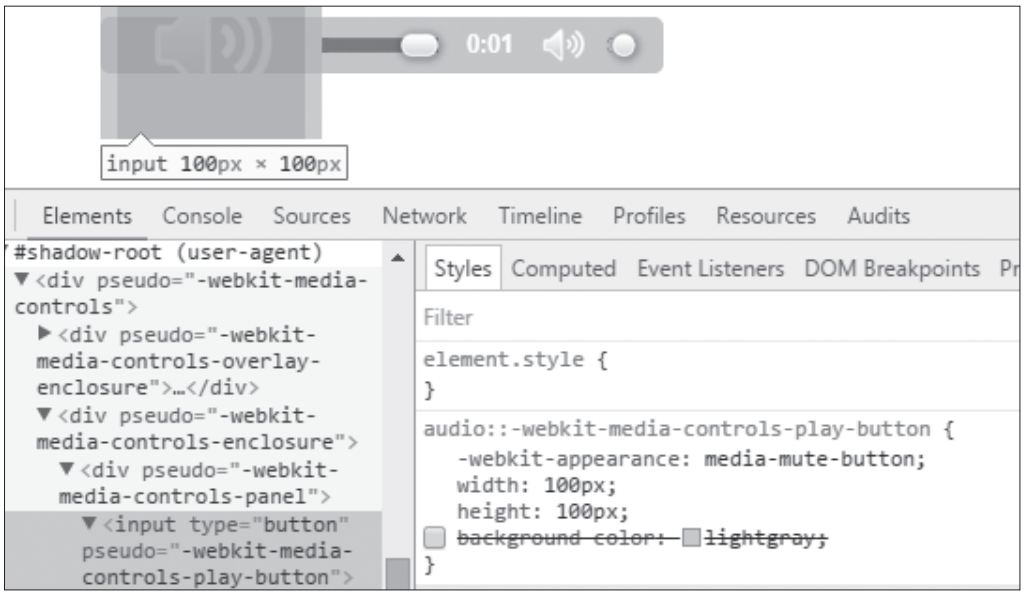


Abbildung 4

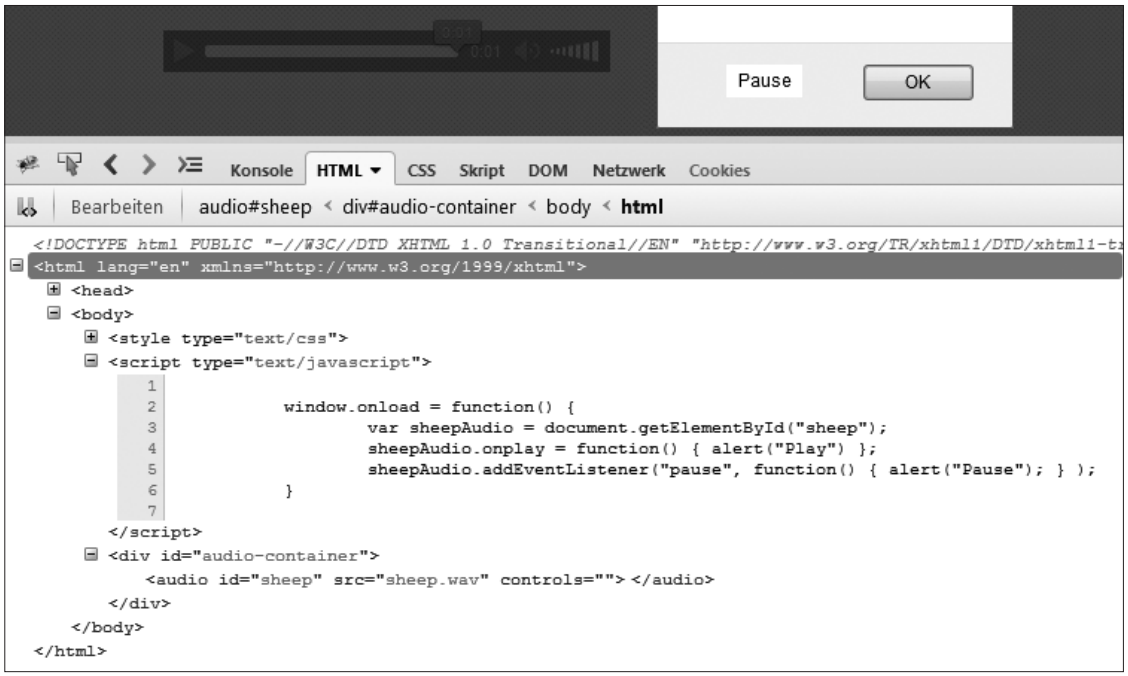


Abbildung 6

was denn der Anwender mit dem Player macht und wo er geklickt hat. Dazu bieten die Player verschiedene Events an [2], welche durch die verschiedenen Aktionen ausgelöst werden. In Abbildung 6 sieht man eine beispielhafte Implementierung, um über die Play- und Pause-Aktionen benachrichtigt zu werden.

Hierzu bedienen wir uns der *play*- und *pause*-Events, welche durch das Element zur Verfügung gestellt werden, und implementieren auf zwei mögliche Arten entsprechende Listener. (Anmerkung: Wenn die Wiedergabe vollständig beendet wurde, so zählt dies auch als Pause.)

Ziehen wir an dieser Stelle ein kurzes Resümee, dann stellen wir fest, dass alle Anforderungen an die Kapselung gegeben sind. Die interne Implementierung der Elemente ist versteckt (bzw. nur mit Tricks sichtbar) und stattdessen gibt es durch Pseudo-Klassen und Javascript-Events eine definierte Schnittstelle, mit der man mit dem Element interagieren kann.

Abbildung 7

I got the power!

Nun schauen wir uns einmal an, wie man sich das Shadow DOM selbst zu Nutze machen kann. Das ist auf den ersten Blick wesentlich einfacher als man gedacht hätte, da einem die Browser-Hersteller im gleichen Zug eine neue JavaScript-Methode geschenkt haben, die man auf Elementen aufrufen kann:

```
Element.createShadowRoot();
```

Damit erstellt man solch eine Shadow Root innerhalb des Elements, wie wir sie bereits in den Chrome Developer Tools gesehen haben, und bekommt ein Referenz darauf zurück. In dieser kann man dann wiederum das *innerHTML* beschreiben. Zum Test legen wir uns, wie in Abbildung 7 zu sehen, einen Button umschlossen von einem *div* im Shadow DOM an.

Doch auch hier stoßen wir schon wieder auf Hürden! Denn der Code funktioniert so im Chrome, im Firefox jedoch nicht. Vorerst…

Was ist los beim Firefox? Nun, das Shadow DOM ist noch kein offizieller Web-Standard und somit gilt auch die Benutzung als eher experimentell. Schaut man sich im MDN die Dokumentation zu der *createShadowRoot*-Methode an [3] so ist diese als *deprecated* markiert. Das ist jedoch nur die halbe Wahrheit. Das Shadow DOM als Technologie ist Teil der *Web Components* [4] und bei der W3C noch in Entwicklung. Besser gesagt ein Working Draft. Darum versteckt Mozilla diese experimentelle Schnittstelle hinter einer Option, die erst explizit aktiviert werden muss. (Abbildung 8)

Hat man diese aktiviert, funktioniert obiger Code dann nämlich auch im Firefox. Das ist eher ernüchternd. Schließlich kann man doch nicht jeden Anwender bitten, erst in den Optionen rumzubasteln.

Nun auch hier kann man sich etwas Abhilfe dank eines *Polyfills* holen. Denn es gibt das *Polymer Project*, dessen Ziel es ist, eine Bibliothek zur Verfügung zu stellen, um auf einfache Weise *Web Components* entwickeln zu können. Da deren Spezifikation, wie oben erwähnt, noch nicht abgeschlossen ist, mussten sie unter anderem auch einen *Polyfill* für das Shadow DOM erstellen. Dieses wurde inzwischen aus dem Projekt ausgegliedert und ist nun Teil des *webcomponents.js* [5], eine Sammlung von *Polyfills* für die *Web Component* Technologien.

Bindet man diese Bibliothek nun in sein Test-Projekt ein, dann hat man zwei Möglichkeiten. Entweder man bindet nur den Shadow DOM *Polyfill* mittels der *ShadowDOM.js* ein, dann verhalten sich Chrome und Firefox (und IE11) gleich. Jedoch hat man dann keine Shadow DOM sondern eine Shady DOM Implementierung, auf die ich gleich zu sprechen komme. Die Alternative wäre das kumulative *webcomponents.js* einzubinden. Dann wird in Chrome die native Shadow DOM Implementierung genutzt, welche wir mit unserem eigenen Code schon erlebt haben. Im Firefox hingegen wird die Shady DOM Implementierung genutzt.

Es gäbe übrigens noch eine dritte Variante, bei der man die *webcomponents-lite.js* einbindet. Diese enthält jedoch alle Polyfills bis auf die des Shadow DOMs und ist damit für unsere Zwecke genauso gut geeignet, wie gar keine Bibliothek einzubinden.

Was ist denn nun dieses Shady DOM? Wo ist den der Unterschied zu dem Shadow DOM? Das Problem bei der Implementierung des Shadow DOM Polyfills ist, dass dieser enorm an der Performanz (speziell im mobile Safari) zieht, um entsprechend alle Features des Shadow DOM (andere Event Targets, veränderter Komponentenbaum usw.) korrekt abzubilden [6]. Deswegen entstand das Shady DOM als eine Art Shadow DOM „Lite“. Hier werden die HTML-Elemente nicht im Baum des Shadow DOM (der ja nicht in jedem Browser zur Verfügung steht) abgelegt, sondern ganz normal ins DOM gerendert. Jedoch wird der Komponentenbaum angepasst, so dass man über die Shady DOM API trotzdem diese Art des scoped component tree sieht, wie man ihn mit echtem Shadow DOM hätte [7].

Für uns konkret heißt das nun, dass wir zwar besser die Shady DOM Variante nutzen sollten, doch damit auch auf einige der Vorzüge des Shadow DOM verzich-

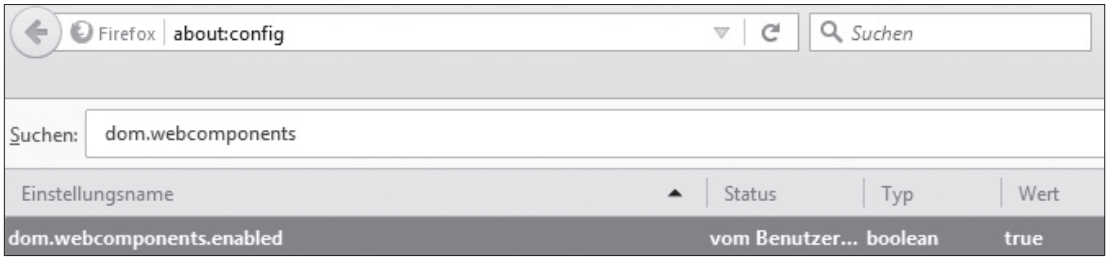


Abbildung 8

ten müssen. Wie der Test in Abbildung 9 zeigt, funktioniert unsere Lösung immer noch – halbwegs.

Jedoch sieht man im Vergleich zu Abbildung 7 auch, dass wir (im Chrome) keinen Shadow Root mehr haben und somit die Elemente auf unseren *padding*-Test (** { padding: 25px }*) anfällig sind. Damit haben wir leider nur einen halben Sieg errungen, denn eigentlich wollten wir ja ein Element haben, was sich nicht durch das CSS beeinflussen lässt.

Nun gut, an dieser Stelle beenden wir vorerst und leicht ernüchtert unseren Ausflug ins Shadow und Shady DOM. Im Zuge der Web Components werden wir jedoch einen zweiten Blick darauf werfen und dann soll sich herausstellen, dass bereits diese Shady DOM Implementierung einen Vorteil bietet.

Vorrübergehendes Fazit

Wie man trotz der Stolpersteine gesehen hat, entsteht hier eine mächtige Technik, die uns voraussichtlich in Zukunft noch öfters über den Weg laufen wird. Doch aktuell ist ihre Nutzung noch ebenso von Schatten überzogen wie der Name vermuten lässt. Die interne Implementierung der Browser-Hersteller funktioniert schon mal bestens. Nun bleibt zu hoffen, dass der Standard möglichst bald definiert und es so auch zu einer native Unterstützung in allen Browsern kommen wird, welche die Technologie für uns Entwickler vollständig nutzbar macht.

Im nächsten Artikel wird es dann mehr um das Thema *Polymer* und *Web Components* gehen, was wir jetzt nur am Rande berührt haben.

Referenzen

[1] GitHubGist *webkit-pseudo-elements.md*, https://gist.github.com/afabbro/3759334

[2] W3Schools *HTML Audio/Video DOM Reference,* http://www.w3schools.com/tags/ref_av_dom.asp

[3] MDN *Element.createShadowRoot()*, https://developer.mozilla.org/de/docs/Web/API/Element/createShadowRoot

[4] MDN *Web Components*, https://developer.mozilla.org/en-US/docs/Web/Web_Components

[5] WebComponents.org, http://webcomponents.org

[6] fbug *Issue 6625: Display shadow DOM elements*, https://code.google.com/p/fbug/issues/detail?id=6625

[7] Polymer Project *What is shady DOM?*, https://www.polymer-project.org/1.0/articles/shadydom.html

Weiterführende Literatur

- W3C *Shadow DOM,* http://www.w3.org/TR/shadow-dom
- Polymer Project, https://www.polymer-project.org

Kurzbiografie

Frank Goraus (frank.goraus@mathema.de) ist Senior Developer bei der MATHEMA Software GmbH in Erlangen. Seit 2006 beschäftigt er sich bereits mit der Entwicklung von JEE-Anwendungen, u. a. in Verbindung mit einem Portal-Server. Seine Liebe zum Detail verwirklicht er mit seinen Web-Design-Kenntnissen. In seiner Freizeit beschäftigt er sich außerdem mit Android-Entwicklung, verschiedensten Web-Frameworks und einem eigenen Projekt für eine Sammlungsverwaltung.

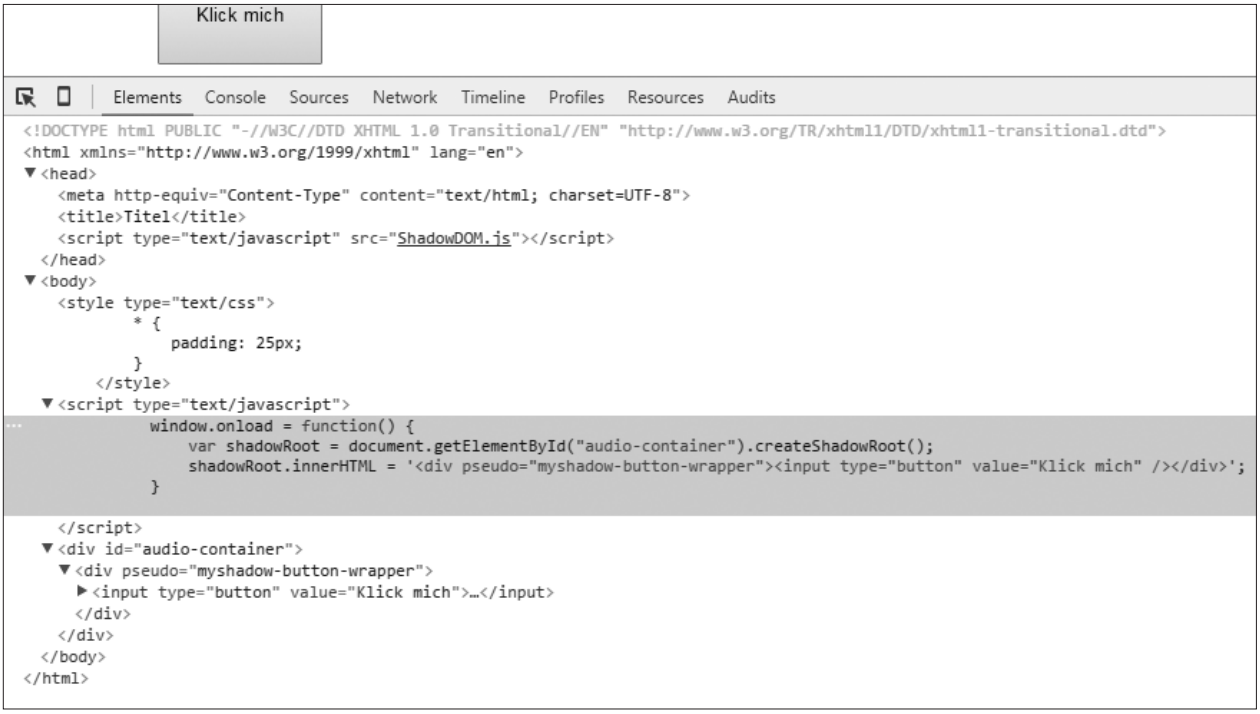


Abbildung 9

Strukturprogramm

Log Daten als Input für Regressionstests

von Florina Sas

Wenn Logdateien als Daten betrachtet werden können, dann werden sie zu wertvollen Informationen. Wie wäre es also, wenn wir die Logdateien nicht nur ab und zu für die Fehlersuche, sondern auch für das Durchführen von Regressionstests verwenden würden? Die Idee dahinter ist, die Logdateien der Produktionsumgebung in Regressionstests zu verwenden, um Fehler oder unbekannte Szenarien zu identifizieren. Dies ist vor allem bei einem neuen Release oder einer Änderung der Konfiguration wichtig. Dieser Artikel betrachtet das Thema Parsen von Logdateien, wenn die Logdateien strukturiert sind, um Input-Daten für Regressionstests vorzubereiten.

Strukturierte Logdateien

Viele Systeme produzieren heutzutage eine große Menge an Logdateien, egal ob es sich dabei um Software-Anwendungen, Datenbanken, Netzwerkgeräte oder Betriebssysteme handelt. Man stellt sich dann die Frage, warum diese Logs nur ab und zu für die Fehlersuche und nicht auch für die Analyse der Schwachstellen dieser Systeme verwendet werden. Mittels einer solchen Analyse können wir nämlich nicht nur eine stärkere Sicherheit und bessere Leistung, sondern auch eine klare Entscheidungsfindung und Statistik für unsere Systeme erreichen.

Um so eine umfangreiche Analyse besser und schneller zu machen, können wir unsere Logging, dort wo es möglich ist, mittels strukturiertem Logging verbessern. Anhand des strukturierten Loggings können unsere geloggten Meldungen mehr Bedeutung bekommen und nicht nur eine Kollektion von Strings bleiben.

Mit strukturiertem Logging ist ein festgelegtes Nachrichtenformat, mit einer wohldefinierten semantischen Bedeutung für die Logdateien gemeint. Um gut strukturierte Logdateien zu erhalten, müssen Nachrichtenformate verwendet werden, die idealerweise für beide, Mensch und Maschine, lesbar sind. Dabei ist es vorteilhaft, wenn diese die folgenden Eigenschaften erfüllen:

- Verwendung von Name-Werte-Paaren
- Zeitstempel mit ausreichender Auflösung am Anfang jedes Eintrags
- Angabe von eindeutigen Identifikatoren (ID) (auch für Benutzer- oder Transaktions-ID)
- Verwendung von Kategorien für die Bedeutung und Schwere eines Eintrags (wie INFO, WARNING, ERROR oder DEBUG)
- Identifizierung der Logquelle (etwa die Klasse oder Methode)
- Einträge möglichst jeweils auf einer eigenen logischen „Zeile“
- In einem einfach von Mensch und Machine lesbaren Formaten (wie etwa JSON oder XML)

Ein für diesen Zweck geeignetes Format ist beispielsweise JSON:

```
[{
   "timestamp":"1376681196470",
   "level":"INFO",
   "message":"Text1"
},{
   "timestamp":"1376681196471",
   "level":"ERROR",
   "message":"Text2",
   "exception":"java.lang.IllegalArgumentException
}]
```

Logmanagement Tools

Eine wertvolle und umfangreiche Loganalyse bieten verschiedene Logmanagement Tools, wie beispielsweise SPLUNK [1], an. Dieses Tool kann alle Logarten analysieren, kann diese durchsuchen, beobachten oder Berichte über die Loggingdaten erstellen. Die Analyse wird in Form von Charts, Reports oder einfach als Suchergebnis in einer Webseite dargestellt. Wenn die Logdateien schon strukturiert sind, dann können wir bei der Analyse die vorgegebenen Formate, die SPLUNK unterstützt, verwenden.

Wie wäre es nun, wenn wir solche Tools wie SPLUNK auch für das Durchführen von Regressionstests verwenden würden? Die Idee dabei ist, die Logdaten der Produktionsumgebung innerhalb des Regressionstests eines Software-Systems zu verwenden. Dieses hat den Zweck, Fehler oder unbekannte Szenarien herauszufinden. SPLUNK bietet SDKs für die Sprachen Java, Python, JavaScript, PHP, Ruby und C# an. Zusätzlich wird ein Eclipse-Plug-in angeboten. Dieses enthält Code-Templates, um es direkt an einer SPLUNK-Instanz anzubinden. Die Logdateien können dann über SPLUNK eingelesen werden. Mittels verschiedener Suchkriterien, die exemplarisch vom Java-Code aus durchgeführt werden, werden diese Daten in den Regressionstests eingespielt. Und dies alles ist möglich, ohne zusätzlichen Code für das Parsen der Logdateien zu benötigen.

Implementierung

Als Beispiel für den oben genannten Ansatz wird unten eine Java Electronic Trading Anwendung betrachtet. Ein Teil der Regressionstests dieser Anwendung wurde mit Hilfe der Logdateien aus der Produktionsumgebung erstellt. Spezifisch für eine solche Anwendung sind die große Anzahl von Szenarien und Events an einem Börsenhandelstag, die häufigen Deployments in der Produktionsumgebung, die Konfigurationsänderungen, aber auch das Logformat, das den oben geforderten Eigenschaften entspricht.

Die Anwendung baut TCP-Verbindungen zu verschiedenen *Counterparties* auf, um Trades zu schicken und Trade Executions zurück zu bekommen. Trade-Daten die durch diese Verbindungen getauscht werden, sind in eine Fix-Nachricht eingepackt. Diese hat ein standardisiertes Format, das in der FIX-Protokoll Spezifikation [2] beschrieben ist. Das Loggingformat einer solchen Nachricht enthält am Anfang jeder Zeile den Zeitstempel, mit einer Granularität im Bereich von Millisekunden, gefolgt von Name-Werte-Paaren für alle andere Felder, die in dem FIX-Protokoll verwendet werden.

```
20150302-16:20:11.010 SNT-000093:8=FIX.4.2#9=232#
35=D#52=20150302-07:20:11.010#49=SOURCE#
50=CLIENT:user#34=93#57=DERIV#56=DEST#
38=100#167=FUT#60=20150302-07:20:11.010#
100=OS#48=SYM#15=EUR#11=VSiajU1t3VQAAA9U#
40=1#207=OS#59=0#55=SYM#54=1#1=TAG#22=5#
200=201503#21=2#10=062
```

Die Logdateien sind vorher in einer SPLUNK-Instanz, die auf dem lokalen Rechner (*localhost*) läuft, als Data Sources importiert worden, um danach mittels Java-Code darauf Suchen durchführen zu können. Beim Import der Logdateien wurde in SPLUNK ein Custom SourceType namens *fix* definiert. Dies liegt daran, dass die vorgegebene Formate nicht ganz zu diesem Format passen. Eine Verbindung zur SPLUNK Instanz bekommen wir wie im Beispiel unten:

```
public static void main(String[] args) {
  // Erzeugen der Login-Parameter.
  ServiceArgs serviceArgs = new ServiceArgs();
  serviceArgs.setUsername("admin");
  serviceArgs.setPassword("admin");
  serviceArgs.setHost("localhost");
  serviceArgs.setPort(8089);

  // Erzeugen einer Service-Instanz und mit
  // den Login-Parametern.
  Service service = Service.connect(serviceArgs);
  // Der Suchparameter
  Args oneshotSearchArgs = new Args();
  ....
}
```

Wir wollen jetzt nach neuen Trades und deren IDs suchen. In dem FIX-Protokoll sind neue Trades dadurch zu erkennen, dass die Fix-Nachricht das Name-Werte-Paar „35=D“ enthält. Weiterhin bekommen wir eine Trade-ID vom Name-Werte-Paar „11=<trade_id>“. Die Query, die wir in der Suche verwenden, kann so gebaut werden, dass die Trade-ID direkt in einem Feld namens ClOrdID zurück geliefert wird:

```
search source="fix.log" host="hostName"  sourcetype="fix"
"35=D" | rex field=_raw "11=(?<ClOrdID>.[^#]+)"
```

Im Fix Protokoll ist die Namenskonvention für die Trade-ID ClOrdID.

```
InputStream resultsStream = service.oneshotSearch(
  "search source=\"fix.log\" host=\"localhostName\" " +
  "sourcetype=\"fix\" \"35=D\" | " +
  "rex field=_raw \"11=(?<ClOrdID>.[^#]+)\"",
  oneshotSearchArgs
);
```

```
List<String> ordIDs = new ArrayList<String>();
try {
  ResultsReaderXml resultsReader =
    new ResultsReaderXml(resultsStream);
  for (Event event : resultsReader) {
    for (String key: event.keySet()) {
      if ("ClOrdID".equals(key)){
        ordIDs.add(event.get(key));
      }
    }
  }
  resultsReader.close();
} catch (IOException e) {
  e.printStackTrace();
}
```

Nun, da wir eine Liste mit allen Trades und deren entsprechenden IDs vorliegen haben, können wir im nächsten Schritt alle geloggten Events, die zu einem Trade gehören, suchen.

```
// map vom Trade ID zu einer Liste von Strings
// (FIX-Nachrichten) die zu diesem Trade gehören
Map<String, List<String>> ordMsgs = new HashMap<>();

for(String id : ordIDs){
  List<String> msgs = new ArrayList<String>();
  InputStream resultsStream = service.oneshotSearch(
    "search source=\"fix.log\" " +
    "host=\"localhostName\" " +
    "sourcetype=\"fix\" \"" + id +"\"",
    oneshotSearchArgs
  );
  try {
    ResultsReaderXml resultsReader =
      new ResultsReaderXml(resultsStream);
    for (Event event : resultsReader) {
      for (String key: event.keySet()) {
        if ("_raw".equals(key)){
          msgs.add(event.get(key));
          break;
        }
      }
    }
    resultsReader.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
  ordMsgs.put(id, msgs);
}
```

Die Suchergebnisse enthalten jetzt alle Trades und deren entsprechenden Trade Executions oder andere FIX-Nachrichten, die mit einem Trade in Verbindung stehen.

Alternativ kann die Query so angepasst werden, dass sie Felder für alle Name-Werte-Paare enthält. Diese werden wiederum in FIX-Nachrichten-Objekte zusammengebaut. Das Beispiel unten zeigt, wie die Felder Menge(*OrderQty*) oder Währung(*Currency*) direkt aus der Query extrahiert werden können, ohne zusätzliches Parsing.

```
Map<String, List<FixMessage>> fixMsgs =
  new HashMap<String, List<FixMessage>>();
for(String id : ordIDs){
  //Liste mit allen Fix Messages die zu einem Trade gehören
  List<FixMessage> msgs =
    new ArrayList<FixMessage>();
  InputStream resultsStream = service.oneshotSearch(
    "search source=\"fix.log\" " +
    "host=\"localhostName\" " +
    "sourcetype=\"fix\" \"" + id +"\" | " +
    "rex field=_raw \"38=(?<OrderQty>.[^#]+)\" | " +
    "rex field=_raw \"15=(?<Currency>.[^#]+)\"",
    oneshotSearchArgs
  );
  String qty = "";
  String ccy = "";
  try {
    ResultsReaderXml resultsReader =
      new ResultsReaderXml(resultsStream);
    for (Event event : resultsReader) {
      FixMessage fixMessage =
        new FixMessage(id);
      for (String key: event.keySet()) {
        if ("OrderQty".equals(key)) {
          fixMessage.setQuantity(event.get(key));
        }
        if ("Currency".equals(key)) {
          fixMessage.setCurrency(event.get(key));
        }
      }
      msgs.add(fixMessage);
    }
    resultsReader.close();
  } catch (IOException e) {
    e.printStackTrace();
  }
  fixMsgs.put(id, msgs);
}

public class FixMessage {
  private String symbol = "";
  private String quantity = "";
  private String currency = "";
  private String ID = null;
  public FixMessage() {
  }
  public FixMessage(String ID) {
    this.ID = ID;
  }
  public String getSymbol() {
    return symbol;
  }
```

```
    public void setSymbol(String symbol) {
      this.symbol = symbol;
    }
    public String getQuantity() {
      return quantity;
    }
    public void setQuantity(String quantity) {
      this.quantity = quantity;
    }
    public void setID(String ID) {
      this.ID = ID;
    }
    public String getID() {
      return ID;
    }
    public String getCurrency() {
      return currency;
    }
    public void setCurrency(String currency) {
      this.currency = currency;
    }
  }
```

Anhand von anderen Name-Werte-Paaren können die Log-Nachrichten weiter in Sender- oder Receiver-Gruppen geteilt werden, so dass die in den Regressionstests eingespielt werden können.

Zusammenfassung

Die klassische Alternative zu der oben beschriebenen Lösung wäre das Parsen von Logdateien mittels Java-Code gewesen. Eine solche Aufgabe ist nicht trivial und Tools wie SPLUNK bieten eine schnellere und bessere Alternative an. Egal ob das Parsen mittels Java-Code oder mit Hilfe eines Log-Management-Tools gemacht wurde, es wird alles einfacher, wenn die geloggten Nachrichten und Events strukturiert sind. Wenn die Logs als Daten betrachtet werden können, dann erhalten wir wertvolle Informationen – unabhängig davon, woher diese kommen oder wie sie verwendet werden.

Referenzen

[1] Splunk Inc. *Operational Intelligence, Log Management, Application Management, Enterprise Security and Compliance | Splunk*
http://www.splunk.com

[2] quickfixj.org *QuickFIX/J – 100% Java Open Source FIX Engine*
http://www.quickfixj.org

Kurzbiografie

Florina Sas (florina.sas@mathema.de) ist seit Januar 2015 als Software-Entwicklerin für MATHEMA Software GmbH in Erlangen tätig. Ihre Themenschwerpunkte umfassen die Java Standard und Enterprise Edition (JavaSE und JavaEE). Neben ihrer Entwicklungstätigkeit hatte sie auch Teamleiter- und ScrumMaster-Aufgaben. Derzeit entwickelt sie eine integrierte Anwendung zur Bearbeitung von systemgesteuerten Baufinanzierungen mittels JavaEE.

MATHEMA

Lesen bildet.
Training
macht fit.

Des Programmierers kleine Vergnügen

Séparée

von Michael Wiedeking

Eines der großen Probleme beim Unicode ist die schiere Menge der Zeichen. Dadurch gestaltet sich etwa die Bestimmung der Art eines Zeichens zu einem kleinen Abenteuer. Mit Hilfe der Surrogatpaare lässt sich aber zumindest bei Einzelfällen einiges optimieren.

Das Arbeiten mit dem Unicode kann schon sehr erquicklich sein [1]. Will man beispielsweise einen Unicode-Text formatieren, so stellt sich die Frage, wie man das am besten anstellt. Natürlich gibt es Formate wie XML oder HTML, aber das mag aus irgendwelchen Gründen nicht in Frage kommen, weil man vielleicht lieber etwas Binäres haben möchte.

Ideal wären da spezielle Zeichen, mit denen man etwa Formattrenner markieren könnte. Oft bedient man sich hier eines Fluchtsymbols, um unterschiedliche Formatarten einzuleiten.

```
Hallo \iItalic\I
```

Hier wäre also der Backslash „\“ das Fluchtsymbol und der nachfolgende Buchstabe markiert das Attribut – exemplarisch der Kleinbuchstabe den Anfang und der Großbuchstabe das Ende. Bei Fluchtsymbolen hat man nun aber das Problem, dass dieses, wenn es selbst vorkommt, irgendwie gefluchtet werden muss. Das ist aber nicht nur unschön, sondern erfordert auch noch das Prüfen eines Spezialfalls.

Abhilfe schafft da die *Private Use Area* (PUA), die aus den Zeichen U+E000 – U+F8FF, U+F0000 – U+FFFFD und U+100000 – U+10FFFD besteht. Der Unicode erlaubt es, diese Zeichen nach Belieben für eigene Zwecke zu nutzen. Definiert man etwa U+E000 für den Anfang und U+E001 für das Ende eines Format-Bereichs, so kann man gänzlich auf ein Fluchtsymbol verzichten.

Sammelt man nun diese Formatzeichen f in einem zusammenhängenden Bereich von U_1 bis U_n, so lässt sich ein solches recht einfach durch eine entsprechende Bedingung ausfindig machen:

```
f = in.read();                    // UTF-16 oder Unicode lesen
if (U1 <= f && f <= Un) {
   …f…
}
```

Wie wir aus einem vergangenen Vergnügen [2] wissen, lässt sich diese Bedingung immer in einen einzigen (vorzeichenlosen) Test umwandeln und so müssen wir hier nicht um die Performanz fürchten. Bei einem UTF-16-Strom müssen allerdings gegebenenfalls die Zeichen erst in Unicode-Zeichen umgewandelt werden, wenn die Formatzeichen nicht in nur einem UTF-16-Zeichen dargestellt werden können [3]. Aber zum Glück gibt es auch hier eine Alternative, mit der gänzlich auf die Konvertierung zu Unicode-Zeichen verzichtet werden kann und die ebenfalls mit exakt einem Test auskommt.

Sucht man sich nämlich einen Bereich aus der PUA, der nicht mehr mit nur einem UTF-16-Zeichen darstellbar ist, so muss dieses als Surrogatpaar dargestellt werden. Reserviert man sich eine beliebige Ebene für seine Formatzeichen, so lässt es sich einrichten, dass das *High*-Surrogat immer gleich ist.

Genügen einem 1024 solcher Formatzeichen, könnte man diese beispielsweise in den Bereich U+100000 – U+1003FF legen. Das dazugehörige Surrogatpaar sähe dann wie folgt aus:

$$high = (1101\ 1011\ 1100\ 0000)_2 = (DBC0)_{16}$$
$$low = (1101\ 11yy\ yyyy\ yyyy)_2$$

Man braucht dann also nur nach dem „Flucht-Surrogat“ suchen und kann dann mit dem (entsprechend maskierten) nächsten Zeichen herausfinden, um welches Formatzeichen es sich handelt – ohne das Surrogatpaar erst in ein Unicode-Zeichen umwandeln zu müssen.

```
if (in.readUtf16() == 0xDBC0) { // high lesen
   f = in.readUtf16() - 0xDC00;  // low lesen; 0 – 1023
   …f…
}
```

Bei den Formatzeichen könnte man dann noch ein dediziertes Bit auswählen, das einem sagt, ob es sich um ein öffnendes oder schließendes Formatzeichen handelt. Aber das ist nun eine ganz andere Geschichte und will vielleicht ein andermal behandelt werden.

Referenzen

[1] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Ersatzweise KaffeeKlatsch*, Jahrgang 8, Nr. 11, S. 14f, Bookware, November 2015
http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2015-11.pdf

[2] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Voll beschränkt!* KaffeeKlatsch, Jahrgang 3, Nr. 4, S. 13f, Bookware, April 2010
http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2010-04.pdf

[3] Wiedeking, Michael *Des Programmierers kleine Vergnügen – Paarweises Vergnügen,* KaffeeKlatsch, Jahrgang 8, Nr. 12, S. 10f, Bookware, Dez. 2015
http://www.bookware.de/kaffeeklatsch/archiv/KaffeeKlatsch-2015-12.pdf

Jahresinhaltsverzeichnis 2015

Artikel

Januar

SharpCity
Städtebau in .NET
von Thomas Haug 01/2015, Seite 6

Zur Visualisierung von analysierter Software gibt es eine Reihe von Werkzeugen, die sich unterschiedlicher Darstellungsformen bedienen. Eine sehr mächtige Plattform zur Analyse und Visualisierung ist die MOOSE-Plattform [1], die sowohl zwei- als auch dreidimensionale Darstellungen unterstützt. Da diese Plattform lediglich *Smalltalk*, *C(++)* und *Java* unterstützt, hat der Autor den *FamixGenerator* implementiert, der aus *.NET-Assemblies* FAMIX-Modelle produziert, welche in die MOOSE-Platt- form importiert werden können [2]. Der Einsatz dieses Tools ist in [2a] und [2b] beschrieben.

Zurückgelehnt
Design eines RESTful HTTP-Clients
von Róbert Bräutigam 01/2015, Seite 13

REST hat sich schon mehr oder weniger als nützliche Technologie etabliert, und der Hype hat auch ein bisschen nachgelassen. Man findet eine Unmenge von Ressourcen um RESTful HTTP-Services zu implementieren, aber es mangelt vielleicht an Einführungen, die sich um die Benutzerseite kümmern.

Februar

Des Pudels Kern
Plattformübergreifend arbeiten
von Frank Ganske 02/2015, Seite 6

Um das große Ganze zu erkennen, muss man auf die Unterschiede im Detail achten. Wie richtet man seinen Arbeitsplatz ein, um mit alternativen Systemen arbeiten zu können? Wie organisiert man, dass man in dem anderen System nahtlos weitermachen kann?

Somewhere over the Rainbow
Verschlüsselung
von Nikolas May 02/2015, Seite 11

Es wird in der heutigen Zeit vermehrt auf Verschlüsselung gesetzt, doch ist oft nicht bekannt warum diese „sicher“ ist. Die Sicherheit eines Verschlüsselungsverfahrens beruht, sofern es keine Schwachstelle besitzt, auf einer einfachen mathematischen „Verschlüsselungs“-Funktion, welche eine komplexe Umkehrfunktion besitzt. Natürlich muss es auch eine einfache Funktion zum Entschlüsseln geben (solange man das Geheimnis kennt).

März

Sendest Du noch oder informierst Du schon?
Gedanken zur Informationsvermittlung
von Ilker Yümsek 03/2015, Seite 6

Wir sind soziale Wesen und kommunizieren jeden Tag in unterschiedlichen Kontexten. Im Rahmen dieser Kommunikation werden Informationen zwischen den Teilnehmern übermittelt (von einem Sender zu einem oder mehreren Empfängern). Ein wichtiger Aspekt dabei ist die Interpretation der Informationen durch den Empfänger. In diesem Artikel geht es darum, wie wesentlich dieser Aspekt die Kommunikation beeinflusst und wie wichtig er im Rahmen unserer alltäglichen Kommunikation ist.

Wo soll’s denn hingehen?
Navigation in JSF aus Sicht eines Neulings
von Frank Lauterwald 03/2015, Seite 8

JavaServer Faces (*JSF*) ist die Standard-Technologie für Webbasierte Frontends in *Java*. Von einem guten Framework er-

wartet man, dass es einfache Aufgaben einfach und komplexe Aufgaben möglich macht. In diesem Artikel wird eine Navigationsaufgabe vorgestellt, die auf den ersten Blick einfach wirkt, für den JSF-Neuling jedoch überraschende Komplexität bereithält.

April

Wrtzlprmpft

Mit GnuPG eine sichere Kommunikation aufbauen

von Thomas Bertz 04/2015, Seite 7

Edward Snowden hat uns gezeigt, dass bei elektronischer Kommunikation im großen Stil abgehört und mitgelesen wird. Die Verteidigung gegen Abhören lautet Verschlüsselung, und die Technik dahinter ist seit vielen Jahren bekannt und bewährt. Trotzdem verschlüsseln aus den unterschiedlichsten Gründen viele Menschen nicht – mit sich allmählich einschleichenden, aber doch weitreichenden Folgen. In diesem Artikel soll anhand von *GnuPG* gezeigt werden, wie der praktische Alltag beim sicheren Datenaustausch von elektronischen Dokumenten aussehen kann: Schlüsseltausch, Unterschreiben, Verschlüsseln, Entschlüsseln und Verifizieren.

Aus, Sitz, Platz!

CDI-Komponenten ohne Server testen

von Werner Eberling 04/2015, Seite 11

Unit-Testing ist im Rahmen der Java Enterprise-Entwicklung schon immer ein unbeliebtes Stiefkind gewesen. Mit dem CDI-Standard bietet sich hier jedoch wieder die Möglichkeit zur Rückbesinnung auf klassische, lokale *Test-First*-Ansätze, wie sie aus der POJO-Entwicklung bekannt sind. Dieser Artikel zeigt wie einfach lokales Testen mit CDI sein kann.

Der Umzug von XML in die Java-Welt

JAXB im Einsatz

von Ángel Salces Sillero 04/2015, Seite 15

XML ist ein Format für den Datenaustausch, das vielfältig genutzt werden kann. Unter anderem für Konfigurationsdateien, *Log*-Dateien und *Web Services*. Häufig ist es notwendig, die Struktur und den Inhalt einer *XML*-Datei in die Java-Welt umzuwandeln. Es gibt mehrere *Java-API*s für die Verarbeitung von XML-Dateien. Dieser Artikel widmet sich der *JAXB-API* und zeigt, wie einfach man XML-Dateien in *Java-Beans* (und umgekehrt) umwandeln kann.

Tortenschlacht

Dynamisches D3 mit Knockout.js

von Sandy Stehr 04/2015, Seite 19

Die schon in der Ausgabe 10/2014 beschrieben, lassen sich mit der *JavaScript*-Bibliothek *D3* auf einfache Weise schöne Diagramme und Visualisierungen erstellen. Sollen diese aber auch auf Nutzereingaben reagieren und dynamisch und wiederverwendbar sein, dann kann der Code mit D3 allein schnell aufwändig und unübersichtlich werden. An dieser Stelle setzt die JavaScript-Bibliothek *Knockout.js* an, die mit einem sauberen darunterliegenden Datenmodel arbeitet, Änderungen registriert und die Benutzeroberfläche automatisch aktualisiert.

Mai

Erbsenzählerei

Mikrobenchmarks mit JMH

von Andreas Heiduk 05/2015, Seite 6

Eine der dunkleren Künste von Java sind korrekt geschriebene *Benchmarks*, die der JVM auf den Zahn fühlen. Da trifft es sich gut, dass es unterstützende Frameworks wie JMH (*Java Microbenchmark Harness*) gibt.

Seifenkisten

Nutzung von SOAP-Services unter Android

von Thomas Künneth 05/2015, Seite 12

Der Zugriff auf *REST-Services* ist unter *Android* mit sehr wenig Aufwand möglich. Entwickler nutzen analog zu klassischem Java die Klassen rund um *HttpURLConnection*. Wie aber sieht es mit den etwas schwergewichtigeren *SOAP-Services* aus? Solche Dienste werden vor allem innerhalb von Unternehmen nach wie vor gerne eingesetzt. Dieser Artikel zeigt, wie Android-Clients über SOAP mit einem *Backend* kommunizieren können.

Juni

PrimeFaces Tree

… und sein Zustandslosigkeitsproblem

von Zeeshan Ali 06/2015, Seite 7

Die PrimeFaces Tree-Komponente ist ein nützliches Werkzeug, hat aber keinen eingebauten Zustands-mechanismus. Diese Eigenschaft führt zu Problemen für die Benutzer und muss mit einer Sonderlösung umgangen werden.

Entwickeln mit Kapuze

Hoodie einmal vorgestellt

von Frank Goraus 06/2015, Seite 13

Web-Entwicklung ist inzwischen ein breit gefächertes Gebiet mit zig Betätigungsfeldern und unterschiedlich komplex gestalteten Anwendungen. Dabei verteilen sich die Entwickler auf verschiedene Rollen und unterschiedliche technologische Schwerpunkte. Ein Teil davon sind die Web-Entwickler, welche sich um das UI der Anwendung kümmern, oder gar eine Anwendung ohne hochgradig verzahnte Backends entwickeln. Damit ihnen diese Arbeit erleichtert wird, gibt es zum Beispiel Frameworks wie Hoodie, welches mit Schlagworten wie *Offline first*, *NoBackend* oder *Dreamcode* für sich wirbt. Doch was muss man sich darunter genau vorstellen?

HTML5 – und Alles ist gut...?

Eine kleine Fallstudie mit JSF und HTML5

von Frank Eichfelder 06/2015, Seite 21

Eigentlich ist HTML5 für Entwickler eine feine Sache: Vieles, was in HTML 4.01 nur auf Umwegen, mit JavaScript oder durch den Einsatz von Plugins zu erreichen war, ist jetzt in den HTML5-Standard integriert. An einem kleinen JSF-Beispiel soll gezeigt werden, dass in der Praxis noch längst nicht Alles so reibungslos funktioniert, wie man sich das vorstellt.

Juli

Fabrikneu

Das Factory Pattern im CDI-Zeitalter

von Marc Spanagel 07/2015, Seite 6

Muster oder Patterns gehören zum Alltag des Software-Entwicklers wie der Speck zur Mausefalle. Oft bieten sie schon vorgefertigte Implementierungsansätze, die in manchen Situationen jedoch bereits überholt sind. Ein Beispiel ist das Fabrikmuster oder Factory Pattern, das in seiner Reinform Probleme bei der Zusammenarbeit mit CDI hat.

August

Scaling Scrum

Können wir Scrum skalieren und dabei noch Scrum „by the book“ folgen?

von Tim Bourguignon 08/2015, Seite 7

Eigentlich ist die Scrum-Methodologie schnell erklärt. So lässt sich in wenigen Minuten eine ausreichend verständliche Einführung in Scrum geben. Schwieriger ist es da schon, Scrum einzuführen und konsequent umzusetzen. Muss Scrum dann zusätzlich noch skaliert werden, stellt sich die Frage, ob das überhaupt möglich ist, ohne dabei die Treue zu Scrum zu opfern.

(All)gemeine Trickserei

Zustände mit Java Generics

von Róbert Bräutigam 08/2015, Seite 11

Java Generics sind bei weitem nicht so mächtig wie in Scala oder wie in einer „richtigen“ funktionalen Sprache wie Haskell. Nichtsdestoweniger kann man sehr interessante – wenn auch vielleicht weniger praktische – Lösungen damit programmieren.

Do it yourself

Eigene JPA Mappings mit *Attribute Convertern*

von Werner Eberling 08/2015, Seite 14

Das letzte Update des JPA Standards wartet mit einer Reihe interessanter neuer Features auf, die in der Entwicklergemeinde bisher leider weitestgehend unbeachtet geblieben sind. Um dies zu ändern soll hier eine dieser Neuerungen, der *Attribute Converter*, mit seinen Anwendungsmöglichkeiten, aber auch seinen Grenzen, genauer vorgestellt werden.

September

SE-DI

CDI in SE-Anwendungen mit Apache DeltaSpike

von Nikolas May 09/2015, Seite 6

Apache DeltaSpike ist eine Sammlung von CDI-Erweiterungen. Es besteht aus einem Kernmodul und einer Vielzahl an optionalen Modulen, mit denen man neue Funktionalität in Enterprise-Anwendungen bringen kann. Neben diesen Erweiterungen kann man mit DeltaSpike die Vorteile eines CDI-Containers in die SE-Welt bringen.

Des Pudels Kern

Teil 2: Plattformübergreifend arbeiten, System-übergreifendes Wissen

von Frank Ganske 09/2015, Seite 9

Es kann sich ändern, soviel wie es will, aber am Ende bleibt doch alles beim Alten. Manche sagen, in der Informatik habe sich in den letzten Jahrzehnten nichts Wesentliches geändert. Gleichzeitig gibt es doch permanent Änderungen, für die wir Umgebungen aktualisieren müssen. Und Änderungen können eben immer auch fehlschlagen.

Oktober

Transferleistung

Datenmigration im Finanzsektor

von Ilker Yümsek 10/2015, Seite 6

Im Rahmen einer Datenmigration werden Daten von einer bestimmten – sich produktiv im Einsatz befindenden Quelle – in eine neue Datenstruktur transportiert und dabei auch entsprechend der neuen Datenstruktur geändert. Durch diese Tatsache berührt die Datenmigration zwei Themenbereiche, die bei der Regulierung der Finanzunternehmen eine große Rolle spielen: Datenschutz und Risikomanagement. In diesem Artikel geht es um die Regulierungsbasis innerhalb dieses Gebiets und um deren Auswirkungen auf eine Datenmigration.

November

Erste Wörter mit einem neuen Buchstaben

Lambda-Ausdrücke für die Alltagsprogrammierung

von Ángel Salces Sillero 11/2015, Seite 6

Wenn man eine neue Sprache lernt, ist es oftmals der Fall, dass man einen neuen Buchstaben benutzen muss. Was man am Anfang als aufwändig empfinden kann, führt normalerweise zu einer Bereicherung der sprachlichen Fähigkeiten. Für die Mehrheit der Leser ist Java eine bereits bekannte Sprache, allerdings wurde sie in der Version 8 um Lambda-Ausdrücke ergänzt. Dieser Artikel zeigt, wie man sie unkompliziert in der Alltagsprogrammierung einsetzen kann und vermittelt grundlegende Konzepte rund um das durch sie in Java neu eingeführte Programmierparadigma.

Alles ist drin

Java-Programme inklusive Abhängigkeiten in ein Über-Archiv packen

von Zeeshan Ali 11/2015, Seite 10

n einem Java Archive (JAR) können Java-Klassen zu Bibliotheken oder sogar zu Programmen zusammengefasst werden. So können sie natürlich auch andere JAR-Dateien beinhalten, aber leider werden diese, wenn es sich dabei um Laufzeit-Bibliotheken handelt, einfach nicht als solche erkannt. Da bedarf es schon eines Über-JARs, das eine solche Fähigkeit mitbringt.

Dezember

Globales Zeitgeschenk

Global Day of Coderetreat

Von Thomas Bertz 12/2015, Seite 6

Ganz genau 31 Stunden dauerte die Veranstaltung, die heuer am Samstag, den 14. November stattfand – 31 Stunden und das an einem Tag? Wie kann das sein?

Kolumnen

Des Programmierers kleine Vergnügen

Jahrestag

von Michael Wiedeking 01/2015, Seite 17

Das Herumhantieren mit Kalenderdaten ist mitunter nicht ganz einfach. Abhilfe schafft da die Nummerierung der Tage innerhalb des Jahres, da sich auf diese Weise sämtliche Berechnungen mit Ganzzahlarithmetik erledigen lassen.

Wochentag

von Michael Wiedeking 02/2015, Seite 13

Mit der Tagesnummer in einem Jahr ist es natürlich ein Leichtes, herauszufinden, um welchen Wochentag es sich dabei handelt. Wenn man denn nur wüsste, auf welchen Jahrestag der erste Tag des Jahres fällt.

Frohe Ostern

von Michael Wiedeking 03/2015, Seite 13

Demnächst ist ja Ostern. Aber warum eigentlich? Warum nicht eine Woche früher, oder einen Monat später? Grund genug, sich damit zu beschäftigen, wie Ostern überhaupt berechnet wird.

Tagestag

von Michael Wiedeking 04/2015, Seite 22

Die Umwandlung eines beliebigen Datums in eine Tagesnummer innerhalb des Jahres ist besonders nützlich, wenn man Tage relativ zueinander bestimmen möchte. Allerdings nutzt das nichts, wenn man nicht in der Lage ist, die Tagesnummer wieder in ein entsprechendes Datum umzuwandeln.

Julianisch, nicht julianisch

von Michael Wiedeking 05/2015, Seite 17

Die bisher auf Tagesbasis vorgestellten Kalender-Algorithmen hatten das Problem, dass sie jeweils am Anfang eines Jahres beginnen und mit demselben auch enden. Das ist nicht nur hilfreich, wenn man etwa wissen will wie viele Tage man alt ist. Es muss also irgendeine Tagesbasis her, die jahresübergreifend ist und darüber hinaus auch noch weit genug zurück liegt, so dass man auch viele Jahre damit erfassen kann.

Unvergnüglich

von Michael Wiedeking 06/2015, Seite 20

Das nutzt das beste Julianische Datum, wenn man es nicht wieder in eines des Gregorianischen Kalenders umwandeln kann? Also bedarf es alleine schon der Vollständigkeit halber eines entsprechenden Algorithmus, auch wenn dieser nicht ganz so vergnüglich ist.

Aufrunden, bitte!

von Michael Wiedeking 07/2015, Seite 9

Ganz besonders wenn es um das Thema Ressourcen geht, benötigt man gelegentlich die nächst höhere Zweierpotenz. Diese zu berechnen ist im Grunde ganz einfach, ist aber schwieriger, als man es von einer Zweierpotenz erwarten würde.

Die spinnen, die Römer!

von Michael Wiedeking 08/2015, Seite 17

Auch wenn Obelix mit seinem berühmten Ausspruch eher die Taten der in Gallien stationierten, römischen Legionäre gemeint haben mochte, so lässt sich seine Aussage problemlos auch auf die römischen Zahlen übertragen. Zum Glück lässt sich eine Zahl trotzdem relativ leicht in sein römisches Pendant umwandeln.

SPQR – Sono pazzi questi romani!

von Michael Wiedeking 09/2015, Seite 15

Wer glaubt, SPQR stehe für *Senatus Populusque Romanus*, also fürSenat und Volk von Rom, der irrt. Denn wie wir seit dem Umgang mit römischen Zahlen wissen, steht es doch eher für *sono pazzi questi romani*, was sich ganz im Sinne von Obelix etwa als „die spinnen, die Römer“ übersetzen lässt. Nichtsdestoweniger müssen natürlich auch die „spinnerten“ römischen Zahlen in dezimale umgewandelt werden.

Zwischenergebnis

von Michael Wiedeking 10/2015, Seite 10

Gelegentlich will man wissen, ob zwei Zahlen nah genug beieinander liegen. Das kann man wie immer naiv machen, wenngleich das – auch wie immer – den Eindruck hinterlässt, man könne es schneller machen. Grund genug sich das genauer anzusehen.

Ersatzweise

von Michael Wiedeking 11/2015, Seite 14

Der Unicode ist der Zeichensatz schlechthin – zumindest für das Web und XML-basierte Informationen. Und das ist sicherlich auch seiner Cleverness bei der Kodierung zu verdanken.

Paarweises Vergnügen

von Michael Wiedeking 12/2015, Seite10

Im Zusammenhang mit den verschiedenen Varianten des Unicode Transformation Formats (UTF) ergeben sich verschiedene Probleme, die auf unterschiedlichste Art und Weise gelöst werden können. Eines dieser Aufgaben ist die Umwandlung eines Surrogatpaars unter UTF-16 in seinen dazugehörigen Unicode. Und das natürlich so schnell wie möglich.

Kaffeesatz

Ab in die Zukunft

von Michael Wiedeking 01/2015, Seite 20

Mit dem Jahreswechsel stellt sich immer wieder die Frage, wie wohl die nähere Zukunft aussehen mag. Hat man es dabei mit einer markanten Jahreszahl zu tun, wirft man auch gerne mal einen Blick zurück. Und das ist dieses Jahr der Blick „*Zurück in die Zukunft*“.

Prinzessinnenliberalisierung

von Michael Wiedeking 02/2015, Seite 15

Haben Sie eine Lieblingsprinzessin? Nun gut, möglicherweise sind Sie inzwischen aus dem Alter rausgewachsen. Aber hatten Sie einmal eine Lieblingsprinzessin? Schneewittchen? Dornröschen? Rapunzel? Belle? Aschenputtel? Jasmin?

Grenzwertig

von Michael Wiedeking 03/2015, Seite 15

Zensur war schon immer ein sehr sensibles Thema. Man sollte meinen, dass besonders im Zusammenhang mit den zunehmenden Demokratisierungsbemühungen in der Welt, die Meinungs- und Gestaltungsfreiheit ernster genommen wird, aber erstaunlicherweise ist oft das Gegenteil der Fall.

Höchst unwahrscheinlich

von Michael Wiedeking 04/2015, Seite 24

Deklarative Programmiersprachen zeichnen sich dadurch aus, dass nur noch das in Regeln beschrieben werden muss, was man erreichen möchte, aber nicht mehr wie. Und anscheinend kann man nun auch die Programmierung getrost dem Rechner überlassen.

Oh, wie schön

von Michael Wiedeking 05/2015, Seite 19

ber das Internet ist ein schier unbegrenzter Fundus an schönen Dingen zu finden. Dabei entstehen zwei große Probleme: Das Schöne überhaupt zu finden und es beizeiten wiederzufinden.

Das Ende der Langeweile

von Michael Wiedeking 06/2015, Seite 25

Seit es die Smartphones gibt, hat man immer etwas zu tun. Wem das wie ein Segen erscheint, dem sei warnend gesagt, dass er damit nicht nur die Langeweile erlegt, sondern auch das Warten auf die Liste der vom Aussterben bedrohten Arten verbannt.

Nie ohne

von Michael Wiedeking 07/2015, Seite 11

Wenn man über die neuesten Seltsamkeiten der Internet-Junkies hört und liest, dann passieren jene meist in fernen Ländern, in China etwa oder den USA. Und so nachvollziehbar diese Marotten auch sein mögen, sie haben etwas, das man eben nur in der Fremde vermutet. Aber weit gefehlt.

Es kommt alles wieder

von Lisa-Marie Wehlmann-Wiedeking 08/2015, Seite 19

Obwohl ich eine ungeklärte Abneigung gegen Schwarzweißfilme habe, habe ich mir vor kurzem einen Film aus den 30er Jahren angesehen. Eine der Figuren fragte, wer ein Telefon hätte und es waren wirklich wenige, die sich daraufhin meldeten.

Zweierlei Datenerfassung

von Michael Wiedeking 10/2015, Seite 11

Vor einiger Zeit habe ich einen wunderschönen Füller geschenkt bekommen. Sie wissen schon, so einen mit Seriennummer und so. Ich habe mich auch deswegen so über diesen Füller gefreut, weil ich immer noch sehr viel mit der Hand schreibe. Und das Schreiben mit Füller finde ich immer noch deutlich angenehmer als etwa das Scheiben mit einem Kugelschreiber.

User Groups

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

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

Java User Groups

Deutschland

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Schweiz

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

.NET User Groups

Deutschland

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

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

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

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

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

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

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

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

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

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

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

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

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

Österreich

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

Software Craftsmanship Communities

Deutschland, Schweiz, Österreich

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

Metropolregion Nürnberg
JUG

Die Java User Group
Metropolregion Nürnberg
trifft sich regelmäßig einmal im Monat.

Thema und Ort werden über
www.jug-n.de
bekannt gegeben.

Weitere Informationen
finden Sie unter:
www.jug-n.de

MATHEMA

meet the experts of enterprise infrastructure

Enterprise JavaBeans (EJB)
Entwicklung von Geschäftslogik-Komponenten
7. – 10. März 2016, 4. – 7. Oktober 2016,
1.650,– € (zzgl. 19 % MwSt.)

Anwendungsentwicklung mit der Java Enterprise Edition
14. – 17. März 2016, 4. – 7. Oktober 2016
1.650,– € (zzgl. 19 % MwSt.)

Fortgeschrittenes Programmieren mit Java
Ausgewählte Pakete der Java Standard Edition,
11. – 13. April 2016, 17. – 19. Oktober 2016
1.350,– € (zzgl. 19 % MwSt.)

Scrum Basics
2. – 3. Mai 2016, 950,– € (zzgl. 19 % MwSt.)

Scrum im Großen
Agiles Organisationsdesign nach LeSS
10. – 11. Mai 2016, 950,– € (zzgl. 19 % MwSt.)

Lesen bildet. Training macht fit.

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

MATHEMA

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

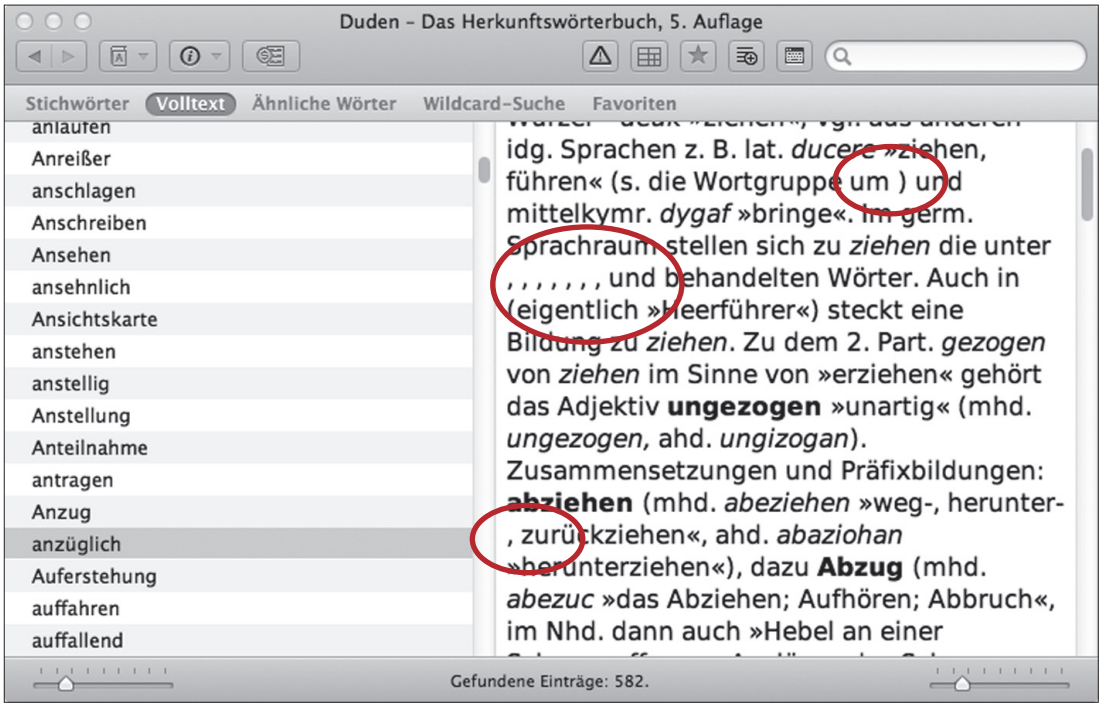
Thomas Bertz, Senior Consultant

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

join the experts of enterprise infrastructure

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

Das Allerletzte



Dies ist kein Scherz!
Dieser Eintrag wurde tatsächlich in der Applikation
des Meisters der deutschen Wörter 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.