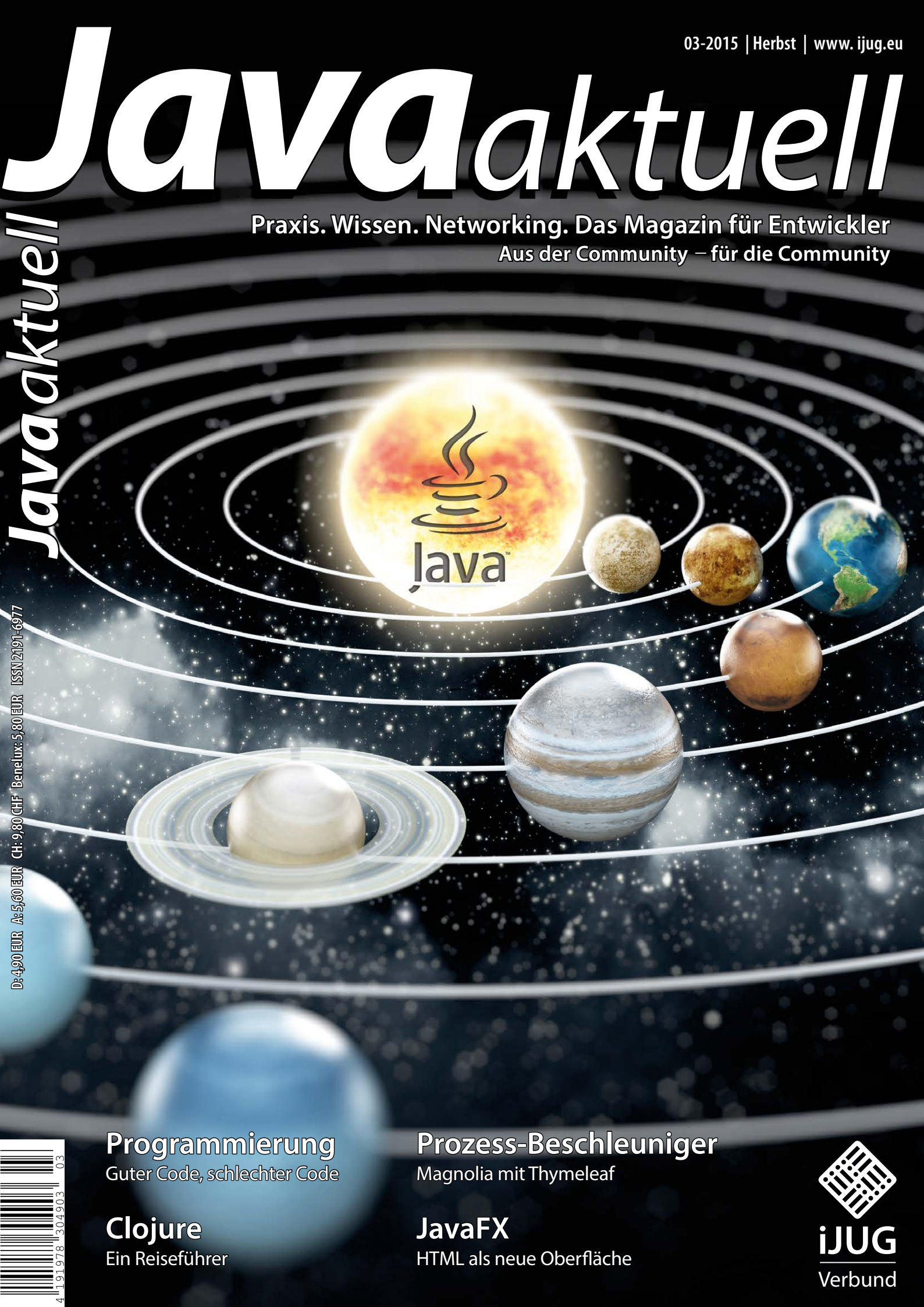
03-2015 | Herbst | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell



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

Programmierung
Guter Code, schlechter Code

Clojure
Ein Reiseführer

Prozess-Beschleuniger
Magnolia mit Thymeleaf

JavaFX
HTML als neue Oberfläche

iJUG Verbund

Kunstprojekt im JavaLand 2015

Seit Java 1.5 erlaubt die Java Virtual Machine die Registrierung sogenannter „Java-Agenten"

Bei Mgnolia arbeiten Web-Entwickler und CMS-Experten mit ein und demselben Quellcode

Ein Heim-Automatisierungs-Projekt

in der Einleitung genannte Access-Log-Beispiel in drei Varianten zur Verfügung. Die ersten zwei Varianten benutzen WSO2-CEP, die dritte verwendet Siddhi direkt innerhalb eines Java-Programms. Die beiden WSO2-CEP-Beispiele unterscheiden sich darin, dass im ersten Fall die grafische Konfigurations-Oberfläche genutzt wird und im zweiten Fall XML-Konfigurationsdateien erzeugt werden, die direkt auf dem Server eingesetzt werden können.

Fazit

Siddhi ermöglicht den Einstieg in CEP und somit in die zeitnahe Verarbeitung von Events. Siddhi kann dabei als Teil des WSO2 Complex Event Processor genutzt werden oder direkt in eigene Applikationen eingebunden sein. Welche der zwei Varianten letztlich benutzt wird, hängt von den jeweiligen Anforderungen des Projektes ab. Als CEP-Schnellstart eignet sich der WSO2-CEP besser, da er dank grafischer Administrationsoberflächen eine deutlich geringere Einstiegshürde bietet.

Referenzen und Quellen

[1] https://github.com/wso2/siddhi
[2] http://wso2.com/products/complex-event-processor/
[3] http://www.eaipatterns.com/docs/EDA.pdf
[4] http://srinathsview.blogspot.ch/2014/12/real-time-analytics-with-big-data-what.html
[5] https://docs.wso2.com/display/CEP310/WSO2+Complex+Event+Processor+Documentation
[6] http://wso2.com/products/complex-event-processor
[7] https://github.com/ungerts/siddhi-cep-example

Tobias Unger
tobias.unger@yenlo.com

Tobias Unger (Dipl.-Inf.) verfügt über mehr als 10 Jahre Erfahrung in den Bereichen Enterprise Architecture, BPM und Java Enterprise. Sein aktueller Fokus liegt auf Design und Implementierung von Integrationsplattformen auf Basis von WSO2-Middleware. Er ist Autor zahlreicher Publikationen und als Sprecher auf Konferenzen aktiv.

http://ja.ijug.eu/15/3/16

Unbekannte Kostbarkeiten des

Heute: Dateisystem-Überwachung

Bernd Müller, Ostfalia

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Wären sie bekannt und würden sie verwendet, könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir stellen in dieser Reihe derartige Features des SDK vor: die unbekannten Kostbarkeiten.

Einige Systeme und Werkzeuge beobachten das Dateisystem, um bei Änderungen bestimmte Aktionen auszuführen. Seit Java 7 ist eine derartige Möglichkeit zur Überwachung im SDK eingebaut und es ist sehr einfach, sie zu verwenden.

Motivation

Der WildFly-Application-Server beobachtet ein bestimmtes Verzeichnis, um beim Erzeugen, Verändern oder Löschen einer Datei mit einer der Endungen .jar/.war/.ear ein Deployment, Redeployment oder Undeployment durchzuführen. Andere Systeme oder Werkzeuge besitzen ähnliche Funktionalitäten, um beispielsweise den Überlauf eines Dateisystems zu verhindern beziehungsweise frühzeitig Alarm zu schlagen. Seit Java 7 enthält das SDK

Interfaces und entsprechende Implementierungen, die es Entwicklern erlauben, derartige Funktionalitäten sehr einfach zu nutzen. Grundlage dafür sind die Interfaces „WatchService" und „WatchKey" im Package „java.nio.file".

„WatchService" und „WatchKey"

Ein „WatchService" beobachtet Änderungen an Dateisystem-Objekten, die zuvor registriert wurden. Änderungs-Events werden indirekt durch einen „WatchKey" repräsentiert. Indirekt deshalb, weil ein derartiger „WatchKey" einen Zustand besitzt, der abgefragt werden kann; es gibt jedoch keine Events und Event-Listener, wie es sonst in Java üblich ist. Ein Grund hierfür ist, dass Dateisystem-Events eventuell in einer höheren Frequenz entstehen, als sie abgearbeitet werden können. Man hat sich daher für einen Abfragemodus entschieden, der es zum einen erlaubt, Events aufzureihen, aber auch, Dateisystem-Events einfach „zu vergessen".

Die Verwendung

Nachdem eine „WatchService"-Instanz erzeugt wurde, wird sie für ein bestimmtes Verzeichnis registriert. Bei der Registrierung sind die interessierenden Events anzugeben: „ENTRY_CREATE", „ENTRY_DELETE" und „ENTRY_MODIFY", Konstanten der Klasse „WatchEvent.Kind<Path>", um auf Dateisystem-Events zu registrieren, oder „OVERFLOW", eine Konstante der Klasse „WatchEvent.Kind<Object>", um auf verlorene beziehungsweise verworfene Events zu registrieren.

Mit den Methoden „poll()" oder „take()" lassen sich „WatchKeys" erfragen. Die zweite Methode ist blockierend, wartet also auf das nächste Event. Nachdem ein „WatchKey" erfragt wurde, folgt die eigentliche Verarbeitung des Events. Im einführenden Beispiel geht es um das Deployment einer EE-Anwendung oder die Reaktion auf zur Neige gehenden Speicherplatz. Nach der Verarbeitung wird der „WatchKey" zurückgesetzt. *Listing 1* zeigt das beschriebene Verfahren exemplarisch.

Die Verarbeitung des Events besteht im Beispiel aus der Ausgabe der Event-Art (Methode „kind()"), also einer der oben genannten Konstanten, und der Ausgabe des Namens der Datei (Methode „context()"), die sich entsprechend verändert hat. Man kann sich sicher leicht vorstellen, wie man hier seine eigenen Anforderungen realisiert. Ein umfangreicheres Beispiel, bei dem rekursiv ein ganzer Verzeichnisbaum beobachtet wird, steht im Java-Tutorial *[1]*.

Plattform-Abhängigkeiten

Nach dem Javadoc des Interface „WatchService" soll eine Implementierung die nativen Möglichkeiten des zugrunde liegenden Betriebssystems für die Überwachung des Dateisystems verwenden und nur, falls das Betriebssystem eine derartige Funktionalität nicht anbietet, Polling oder andere Alternativen als Fall-Back-Lösung verwenden. Die im OpenJDK und Oracle-SDK verwendete Implementierung für Linux, die Klasse „sun.nio.fs.LinuxWatchService", realisiert ganz offensichtlich die direkte Betriebssystem-Verwendung, da auf dem Fedora-Rechner des Autors keinerlei Rechenaufwände des Prozesses feststellbar sind, selbst wenn zum Beispiel das Log-Verzeichnis des WildFly-Application-Servers überwacht wird.

Fazit

Seit Java 7 gibt es im SDK die Möglichkeit, Änderungen im Dateisystem, also das Neuanlegen, Löschen oder Ändern von Dateien, zu überwachen und beim Eintreten eines solchen Ereignisses mit einem beliebigen Methodenaufruf darauf zu reagieren. Das API ist äußerst schlank und sehr einfach zu verwenden.

Literatur

[1] Watching a Directory for Changes: http://docs.oracle.com/javase/tutorial/essential/io/notification.html

Bernd Müller
bernd.mueller@ostfalia.de

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

http://ja.ijug.eu/15/3/17

```
WatchService watcher = FileSystems.getDefault().newWatchService();
Path path = Paths.get("."); // zu beobachtendes Verzeichnis
path.register(watcher, ENTRY_CREATE, ENTRY_DELETE, ENTRY_MODIFY);
for (;;) {
   WatchKey key = watcher.take();
   for (WatchEvent<?> event : key.pollEvents()) {
       System.out.println("Name: " + event.kind());
       System.out.println("Path: " + event.context());
   }
   key.reset();
}
```

Listing 1

Alle unbekannten Kostbarkeiten

Bernd Müller veröffentlicht seit Jahren die "Unbekannten Kostbarkeiten des SDK" in der Java aktuell. Die früheren Beiträge sind auf seiner Website zu finden *(siehe „http://www.pdbm.de/person/publikationen.html#Artikel+in+Zeitschriften+und+Tagungsb%C3%A4nden")*.