04-2014 | Winter | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Aus der Community – für die Community

Java aktuell

Mit Lambda-Ausdrücken einfacher programmieren

Go for the Money
Währungen und Geldbeträge in Java

Oracle-Produkte
Die Vorteile von Forms und Java vereint

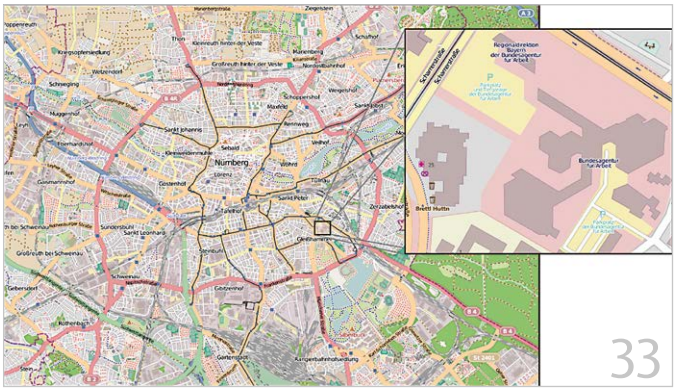
Pimp my Jenkins
Noch mehr praktische Plug-ins

Sonderdruck

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

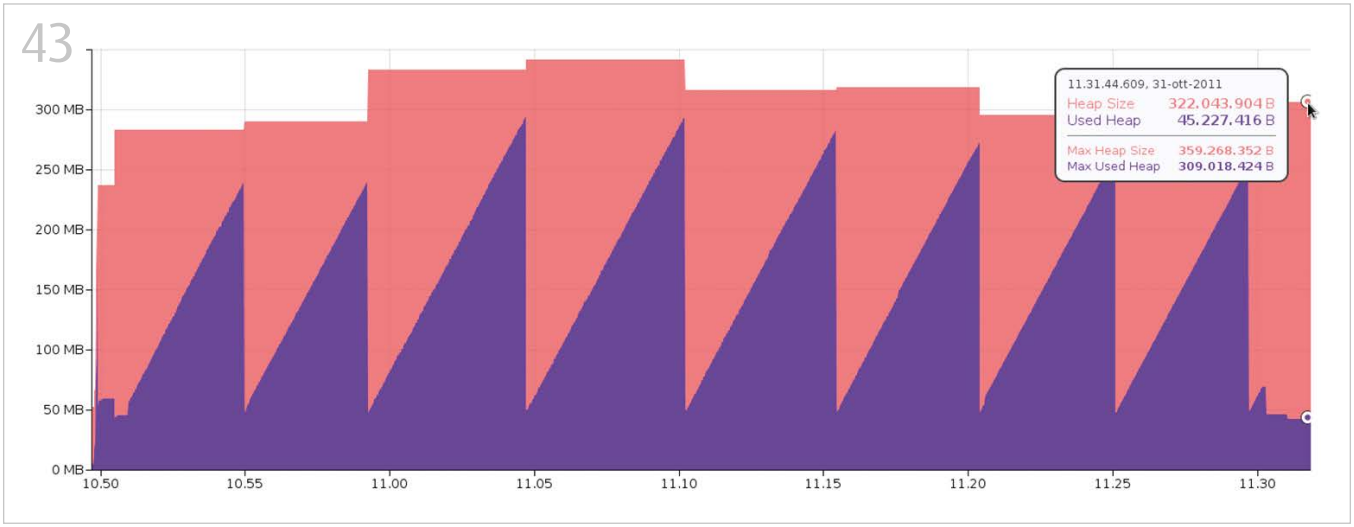
4 191978 304903 04

iJUG Verbund



OpenStreetMap ist vielen kommerziellen Diensten überlegen, Seite 33

Alles über Währungen und Geldbeträge in Java, Seite 20



WURFL ist verglichen mit selbst dem nicht reduzierten OpenDDR-Vokabular deutlich speicherhungriger, Seite 43

Fazit

Clientseitige Integration ist ein vielversprechender Ansatz, um Web-Anwendungen zu integrieren, wo dies serverseitig nicht sinnvoll realisierbar ist. Die „Same Origin Policy" ist ein sinnvoller und notwendiger Mechanismus, um sich gegen schädliche Skripte und unbefugten Zugriff auf die eigenen Daten zu schützen. Sie steht jedoch einer clientseitigen Integration grundsätzlich als größtes Hindernis im Weg.

Die aufgezeigten Mechanismen schaffen ein vielfältiges Repertoire an Möglichkeiten, mit diesem Hindernis umzugehen. Was bleibt, ist die Entscheidung, das geeignete Mittel mit Blick auf Technologie und Sicherheit für den konkreten Anwendungsfall auszuwählen und einzusetzen. Darüber hinaus kann das Wissen um diese Mechanismen helfen, bei der Konzeption von Serveranwendungen bestimmte Mechanismen (wie JSONP oder CORS) zu berücksichtigen und damit von vornherein die notwendigen Voraussetzungen für eine clientseitige Integration zu schaffen.

Links

[1] http://de.wikipedia.org/wiki/Same-Origin-Policy
[2] http://bigdesk.org
[3] http://www.elasticsearch.org/guide/en/elasticsearch/reference/current/common-options.html#_jsonp
[4] http://www.w3.org/TR/cors
[5] http://jquery.com
[6] http://easyxdm.net/wp
[7] http://ternarylabs.github.io/porthole

Sascha Zak
sascha.zak@adesso.de

Sascha Zak ist Senior Software Engineer bei der adesso AG im Competence Center Telematik in Berlin. Schwerpunktmäßig beschäftigt er sich mit Architektur und Entwicklung von webbasierten Systemen sowie Anwendungen im Umfeld von Smartcards und der elektronischen Gesundheitskarte.

http://ja.ijug.eu/14/4/15

Unbekannte Kostbarkeiten des SDK Heute: Die Klasse „Objects"

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.

Die Klasse „java.lang.Object" ist hinreichend bekannt, da sie die Oberklasse aller Java-Klassen ist. Die mit Java SE 7 eingeführte „java.util.Objects" ist hingegen vielen Entwicklern unbekannt, obwohl sie durchaus sinnvoll verwendet werden kann.

Die Klasse „Objects"

Die Klasse „java.util.Objects" ist im API-Doc [1] beschrieben als: „This class consists of static utility methods for operating on objects. These utilities include null-safe or null-tolerant methods for computing the hash code of an object, returning a string for an object, and comparing two objects."

Die hinter der Klasse steckende Motivation ist also, den Umgang mit „Null"-Referenzen zu vereinfachen. Dies hat ganz offensichtlich auch die in Java SE 8 eingeführte Klasse „java.util.Optional" zum Ziel. Wir wollen in unserer Reihe der unbekannten Kostbarkeiten allerdings keine neuen, kürzlich eingeführten Features vorstellen, sondern solche, die schon einige Zeit existieren, aber nicht genutzt werden, einer breiten Öffentlichkeit bekannt machen.

Die Klasse „Objects" kann nicht instanziiert werden und enthält in der Version Java SE 7 neun statische Methoden, die vor allem dem Vergleich von Objekten und der Berechnung des Hash-Codes beziehungsweise der String-Repräsentation eines Objekts dienen. Wir wollen uns hier nur exemplarisch die „equals()"-Methode anschauen, bitten jedoch den Leser, einen Blick auf die komplette Klasse zu werfen – es lohnt sich.

```
public final class Pair<L, R> {

  private final L left;
  private final R right;

  public Pair(L left, R right) {
    this.left = left;
    this.right = right;
  }
  ...
}
```

Listing 1

```
public boolean equals(Object obj) {
  if (this == obj)
    return true;
  if (obj == null)
    return false;
  if (getClass() != obj.getClass())
    return false;
  Pair<?, ?> other = (Pair<?, ?>) obj;
  if (left == null) {
    if (other.left != null)
      return false;
  } else if (!left.equals(other.left))
    return false;
  if (right == null) {
    if (other.right != null)
      return false;
  } else if (!right.equals(other.right))
    return false;
  return true;
}
```

Listing 2

```
public boolean equals(Object obj) {
  if (Objects.equals(getClass(), obj.getClass())) {
    Pair<?, ?> other = (Pair<?, ?>) obj;
    return Objects.equals(left, other.left)
        && Objects.equals(right, other.right);
  } else {
    return false;
  }
}
```

Listing 3

Bevor wir auf die Code-Ebene eines Beispiels hinabsteigen, ist hier zunächst noch eine ganz allgemeine Bemerkung zu „Null"-Referenzen angebracht. Charles Antony Richard Hoare, meist als „C.A.R. Hoare" oder „Tony Hoare" abgekürzt, ist ein Informatiker, den jeder Informatikstudent zumindest dem Namen nach als Erfinder des Quicksort-Algorithmus kennt. Er hat im Jahr 1965 die „Null"-Referenz in die Sprache „ALGOL W" eingeführt. Er gilt damit als deren geistiger Vater. Unter [2] findet man ein Video, in dem Tony Hoare seine Einführung der „Null"-Referenz als „Billion Dollar Mistake" beschreibt.

Verwendung von „Objects.equals()"

Wir demonstrieren an einem Beispiel, wie die Verwendung der „equals()"-Methode von „java.util.Objects" Code deutlich vereinfachen kann. Dazu dient die generische Klasse „Pair<L, R>", deren Instanzen gleich sein sollen, wenn der linke und rechte Teil des Paares jeweils gleich sind. Listing 1 zeigt, wie eine einfache Implementierung aussehen könnte. Die von Eclipse generierte „equals()"-Methode fällt recht umfangreich aus (siehe Listing 2). Die Überarbeitung unter Verwendung von „Objects.equals()" reduziert den Code erheblich (siehe Listing 3).

Wir haben dabei alle Fallunterscheidungen des ursprünglichen Codes übernommen, entweder explizit oder implizit über die „Objects.equals()"-Implementierung. Einzige Ausnahme ist das erste „if" des ursprünglichen Codes, auf das wir verzichtet haben, da es durch die darauffolgenden „equals()" impliziert wird. Der Leser wird sicher zustimmen, dass diese Variante deutlich kürzer und wesentlich besser zu verstehen ist.

Fazit

Die Klasse „java.util.Objects", die seit Version 7 in Java SE enthalten ist, enthält Utility-Methoden, die bei der Verwendung von Objektreferenzen diese implizit auf „Null"-Referenzen überprüfen. Dies erspart explizite Tests darauf, sodass der resultierende Code unter Verwendung der Objects-Methoden zum Teil erheblich schlanker ist als ohne deren Verwendung. Neben der verwendeten „equals()"-Methode enthält die Klasse „Objects" weitere „Null"-tolerante Methoden, wie etwa „compare()", „deepEquals()", „hash()" und „hashCode()", die man sich unbedingt anschauen sollte.

Literatur/Videos

[1] http://docs.oracle.com/javase/7/docs/api/java/util/Objects.html

[2] http://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor des Buches „JavaServer Faces 2.0" und Mitglied in der Expertengruppe des JSR 344 (JSF 2.2).

http://ja.ijug.eu/14/4/16