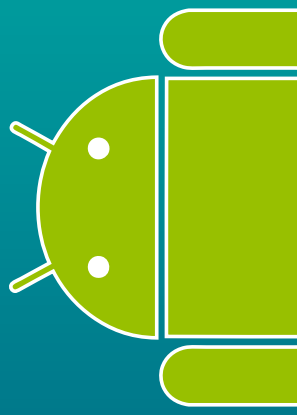
Nr.01 | Frühjahr 2012 | www.ijug.eu

Java aktuell

Das Magazin der Java-Community

Java aktuell

ANDROID in der Praxis



JavaOne 2011
Neuigkeiten und Trends

Oracle Public Cloud
Bereit für Wolke sieben

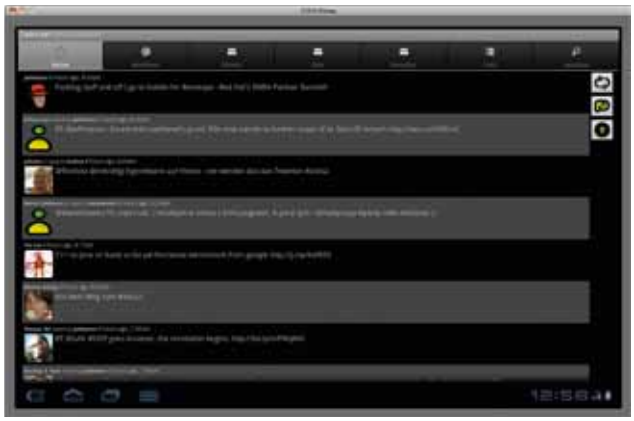
Adobe AIR
Anspruchsvolle Applikationen realisieren

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

Sonderdruck

iJUG Verbund

Inhalt



Android-Apps fit für die Zukunft machen, Seite 12

Fazit

Abseits der Spring-Community hat sich JCR zu einem weitverbreiteten Standard gemausert. Man merkt sehr deutlich, dass ein großes Interesse der Community vorhanden ist. Ein Indikator dafür ist die massive Anzahl von Tutorials, Mailing-Listen, Foren-Einträgen und Artikeln rund um das Thema JCR. Gerade diese umfangreiche Dokumentation erleichtert den Einstieg enorm.

Weitere JCR-Implementierungen sind unter [7] zu finden. Wer sich über den JCR-Standard hinaus noch mehr Flexibilität wünscht, besonders in Richtung SOAP und REST-Schnittstellen, sei auf „Content Management Interoperability Service" (CMIS) verwiesen. Dieser erweitert unter anderem auch den JCR-Standard und wird bereits von vielen namhaften ECM-Herstellern wie IBM, Microsoft, SAP oder Oracle unterstützt.

Die große Nachfrage spiegelt sich auch darin wider, dass im Moment die JCR-Version 2.1 in der Entwicklung ist (JSR-333). Hier wurde bereits der Early Draft veröffentlicht und daraus geht hervor, dass vor allem einige Änderungen und Vereinfachungen an der API vorgenommen werden, um die ohnehin nicht sehr hohen Einstiegshürden nochmals zu senken.

Literatur

[1] http://jackrabbit.apache.org

[2] http://www.jcp.org/en/jsr/detail?id=147

[3] http://www.oracle.com/technetwork/java/javaee/jta/index.html

[4] http://www.oracle.com/technetwork/java/jndi/index.html

[5] http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

[6] https://jira.springsource.org/browse/SEJCR

[7] http://wiki.apache.org/jackrabbit/JcrLinks

Dominic Weiser

dominic.weiser@esentri.com

Dominic Weiser beschäftigt sich bei esentri mit der Entwicklung von Enterprise-Social-Networking-Lösungen. Sein Spezialgebiet sind SaaS-Anwendungen auf Basis von JEE und Spring.

Unbekannte Kostbarkeiten des SDK: Dynamic Proxy

Bernd Müller, Ostfalia – Hochschule für angewandte Wissenschaften

Das Java SDK enthält eine Reihe von Features, die wenig bekannt sind. Damit könnten Entwickler viel Arbeit und manchmal sogar zusätzliche Frameworks einsparen. Wir wollen in dieser Reihe derartige Features des SDK vorstellen: die unbekannten Kostbarkeiten.

Das Proxy-Pattern wird im Buch der „Gang of Four" wie folgt definiert: „Provide a surrogate or placeholder for another object to control access to it". Neben der reinen Stellvertreter-Funktionalität kann auch der Zugriff auf das Proxy, also Methodenaufrufe, kontrolliert werden. Wir demonstrieren in einem kleinen Beispiel sowohl die Stellvertreter-Funktionalität als auch die Zugriffskontrolle. Dazu bauen wir einen Entity-Manager nach JPA-Vorbild, jedoch ohne wirkliche Funktionalität nach und kontrollieren die CRUD-Methodenaufrufe für das Transaktionsmanagement.

Proxy und InvocationHandler

Die Basis für dynamische Proxies bilden die Klasse „Proxy" und das Interface „InvocationHandler" im Package „java.lang.reflect". Beide sind seit Version 1.3 im Java SDK enthalten und gehören zu den unbekannten Kostbarkeiten. Die Klasse „Proxy" besitzt die beiden statischen Fabrikmethoden „getProxyClass()" und „newProxyInstance()". Die erste Methode liefert eine dynamisch erzeugte Klasse, die eine Menge von Interfaces implementiert. Die zweite Methode gibt eine Instanz einer solchen Klasse zurück.

Nachdem nun die Implementierung des Proxies möglich ist, folgt der zweite Teil des Proxy-Patterns, die Delegation des Methodenaufrufs. Hierzu wird das Interface „InvocationHandler" verwendet. Beim Einsatz von „getProxyClass()" erfolgt dies explizit, bei der Verwendung von „newProxyInstance()" implizit als Parameter der Methode. Wir werden für ein Beispiel die zweite Alternative benutzen, da der zu erstellende Code kompakter ist.

Das Beispiel

Als Beispiel bauen wir JPAs „EntityManager" nach, beschränken uns aber auf die „persist()"-Methode und bleiben die tatsächliche Realisierung schuldig. Das Interface „EntityManager" ist also recht einfach (siehe Listing 1).

Eine Implementierung ist der „SimpleEntityManager" (siehe Listing 2).

```
public interface EntityManager {
  void persist(Object entity);
}
```

Listing 1

```
public class SimpleEntityManager implements
EntityManager {
   @Override
   public void persist(Object entity) {
        System.out.println(„Persisting „ +
        entity);
  }
}
```

Listing 2

Ziel ist nun, ein Proxy zu erzeugen, das dasselbe Interface implementiert und die Kontrolle des Methodenaufrufs erlaubt. Im Beispiel soll ein Transaktionsmanagement

eingebaut werden. Es sind aber auch andere Möglichkeiten, etwa das Ändern der Methodenparameter oder der Abbruch des Methodenaufrufs denkbar. Die Klasse „EMProxy“ implementiert das Interface „InvocationHandler“ und hat das zu vertretende Objekt als Instanzvariable (siehe Listing 3).

```
public class EMProxy implements Invoca-
tionHandler {
  private EntityManager em;
  public EntityManagerInvocationHandler
  (EntityManager em) {
      this.em = em;
  }
  ...
}
```

Listing 3

Das Interface besitzt „invoke()“ als einzige Methode (siehe Listing 4). Hier ist „proxy“ die Proxy-Instanz, für die die Methode aufgerufen wurde, „method“ die aufzurufende Methode und „args“ deren Parameter. Wir implementieren „invoke()“ wie in Listing 5 gezeigt.

```
Object invoke(Object proxy,
          Method method,
          Object[ ] args) throws Throwable
```

Listing 4

```
@Override
public Object invoke(Object proxy, Method method,
   Object[ ] args)
 throws Throwable {
   Object result;
   try {
     System.out.println(„hier Transaktion starten“);
     result = method.invoke(em, args);
   } catch (InvocationTargetException e) {
     System.out.println(„hier Transaktion zurückrollen“);
     throw e.getCause();
   }
   System.out.println(„hier Transaktion committen“);
   return result;
}
```

Listing 5

Auch hier haben wir analog zur Methode „persist()“ mit „println()“ angedeutet, wo die eigentliche Implementierung zu realisieren ist. Hier wird mit der Methode „Method.invoke()“ die konkrete Methode über Reflection aufgerufen und an den entsprechenden Stellen das Transaktionsmanagement implementiert. Die Verwendung des Proxies erfolgt nun über die Methode „EMProxy.createProxy()“ (siehe Listing 6).

```
EntityManager em = EMProxy.createProxy(
                new SimpleEntityManager());
   em.persist(...);
```

Listing 6

In diesem Beispiel erfolgt die explizite Angabe der realen Klasse. Man kann sich aber leicht Alternativen mit einer Fabrikmethode, die über System-Property gesteuert wird, oder – moderner – die Verwendung einer Annotation vorstellen (siehe Listing 7).

```
EntityManager em = EMProxy.newInstance();
// oder
@Inject
EntityManager em;
```

Listing 7

Fazit

Die Möglichkeiten zur Realisierung eines dynamischen Proxies sind bereits seit Version 1.3 fester Bestandteil des SDK, jedoch leider wenig bekannt. Da die Entscheidung, auf welche konkrete Implementierung gemappt wird, erst zur Laufzeit vorgenommen wird, sind sehr dynamische und flexible Systeme realisierbar.

Bernd Müller
bernd.mueller@ostfalia.de

Bernd Müller ist seit März 2005 Professor für Software-Technik an der Fakultät Informatik der Hochschule Braunschweig/Wolfenbüttel, nachdem er sieben Jahre Professor für Wirtschaftsinformatik an der Hochschule Harz war.

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift *DOAG News* und vier Ausgaben im Jahr *Business News* zusammen für 75 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.