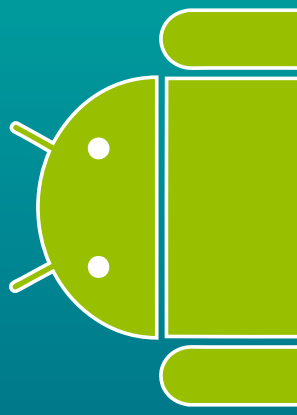
Nr.01 | Frühjahr 2012 | www.ijug.eu

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

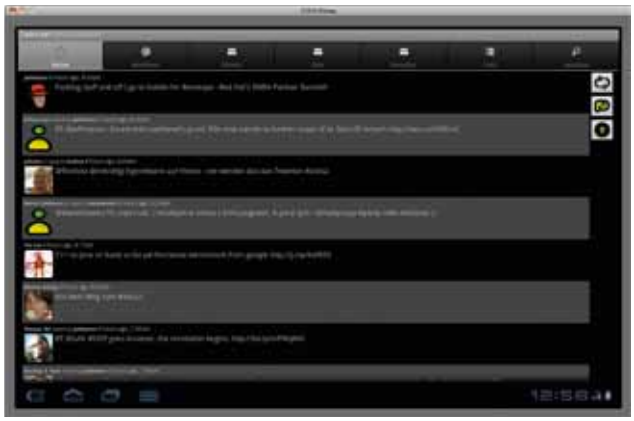
Adobe AIR
Anspruchsvolle Applikationen realisieren

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

Weaving, Instrumentation, Enhancement: Was ein JPA-Provider so alles macht

Marc Steffens und Bernd Müller, Ostfalia - Hochschule für angewandte Wissenschaften

Ein JPA-Provider muss die in der Spezifikation beschriebene Semantik von Assoziationen gewährleisten. Dazu müssen beispielsweise aus einer einfachen Return-Anweisung eines Attributs eine komplexe SQL-Anweisung erzeugt, diese ausgeführt und das Ergebnis aufbereitet werden. Um dies zu unterstützen, stellt JPA im Interface „ClassTransformer" Möglichkeiten bereit, eine Entity-Klasse auf Byte-Code-Ebene zu transformieren, die überarbeitete Version in die JVM zu laden und der Anwendung zur Verfügung zu stellen.

Der Artikel zeigt allgemein die Aufgabenstellung eines JPA-Providers bezüglich Assoziationen und die Realisierung durch die drei populären JPA-Provider EclipseLink, Hibernate und OpenJPA. Da die jeweiligen Realisierungen relativ aufwändig und komplex sind, geben wir diese nur im Ansatz wieder, versuchen aber trotzdem, dem Leser einen Eindruck der von den JPA-Providern geleisteten Aufgaben zu geben.

Zur Beschreibung von Assoziationen existieren in JPA die vier Annotationen „@OneToOne", „@OneToMany", „@ManyToOne" und „@ManyToMany". Mit JPA 2.0 kamen Möglichkeiten für Element-Collections hinzu. Um den Umfang nicht zu sprengen, konzentrieren wir uns exemplarisch auf 1:1-Beziehungen. Als Beispiel dient ein Ausschnitt aus dem Hochschulalltag: Zwischen den Entity-Klassen „Student" und „Studentenausweis" besteht eine 1:1-Beziehung, die, wie im folgenden Beispiel dargestellt, realisiert werden kann (siehe Listing 1).

```
public class Student {
  ...
  @OneToOne
  Studentenausweis ausweis;
  ...
  public Studentenausweis getAusweis() {
    return ausweis;
  }
  ...
}
```

Listing 1

JPA sieht für Objekt-wertige Assoziationsziele das frühe Laden (eager loading), für Collection-wertige Assoziationsziele das späte Laden (lazy loading) als Default für die Implementierung der Assoziation vor. Beim späten Laden einer Assoziation wird der JPA-Provider in der Regel eine Select-Anweisung an das Datenbank-System absetzen, um das Assoziationsziel in die JVM zu laden. Statt der einfachen Return-Anweisung im Quell-Code muss also eine SQL-Anweisung erzeugt und ausgeführt sowie das Ergebnis in entsprechende Java-Objekte eingepackt werden. Um dies zu unterstützen, sieht JPA das Interface „ClassTransformer" im Package „javax.persistence.spi" vor. Die einzige Methode „transform()" dieses Interface repliziert die Methode desselben Namens im Interface „ClassFileTransformer" im Java-SE Package „java.lang.instrument" (siehe Listing 2).

Die drei populären JPA-Provider EclipseLink, Hibernate und OpenJPA verwenden diese und andere Möglichkeiten, um Transformationen am Byte-Code von Entity-Klassen vorzunehmen, benutzen aber verschiedene Bezeichnungen, um diese Transformationen zu beschreiben. EclipseLink nennt dies „Weaving", Hibernate „Instrumentation" und OpenJPA „Enhancement".

Der Artikel soll einen Eindruck davon vermitteln, was ein JPA-Provider tun muss, um seiner Aufgabe gerecht zu werden. Da die Aufgaben relativ umfangreich sind, werden nur die ersten Schritte beschrieben; es wird nicht bis zur SQL-Generierung vorgedrungen. Bei Interesse am generierten SQL raten die Autoren, den Log-Level der drei Provider entsprechend zu erhöhen und das Log zu studieren.

Wir schauen uns im Folgenden die jeweiligen Vorgehensweisen zur Veränderung des Byte-Codes an. Um das Beispiel möglichst einfach zu halten, beschränken wir uns auf die 1:1-Beziehung zwischen „Student" und „Studentenausweis" und hier wiederum auf den Getter. Da eine solche Beziehung im Default früh geladen wird, spätes Laden aber aufwändigere und für unsere Darstellung geeignetere Mechanismen benötigt, definieren wir explizit das späte Laden: „@OneToOne(fetch

```
byte[ ] transform(java.lang.ClassLoader loader,
        java.lang.String className,
        java.lang.Class<?> classBeingRedefined,
        java.security.ProtectionDomain protectionDomain,
        byte[ ] classfileBuffer)
        throws
    java.lang.instrument.IllegalClassFormatException
```

Listing 2

```
<target name="weaving" description="weaving">
  <weave source="build/project.jar" target="woven.jar"
     persistenceinfo="../src" loglevel="FINEST">
    <classpath>
      <fileset dir="${lib.dir}" includes="**/*.jar" />
    </classpath>
  </weave>
</target>
```

Listing 3

```
public Studentenausweis getAusweis() {
    return _persistence_get_ausweis();
}

public Studentenausweis _persistence_get_ausweis() {
    _persistence_checkFetched(„ausweis");
    _persistence_initialize_ausweis_vh();
    this.ausweis = ((Studentenausweis)this._persistence_ausweis_
vh.getValue());
    return this.ausweis;
}
```

Listing 4

= FetchType.LAZY)". Die Realisierung des späten Ladens erfordert eine Byte-Code-Manipulation von EclipseLink und OpenJPA, während Hibernate auch ohne eine solche Transformation spät laden kann.

EclipseLink: Weaving

EclipseLink nennt die Byte-Code-Transformation „Weaving" und unterscheidet zwischen der statischen Transformation nach dem Kompilieren mithilfe einer Ant-Task beziehungsweise der Kommandozeile und der dynamischen Transformation zum Ladezeitpunkt mithilfe eines Java-Agenten. Als Werkzeug wird die Bibliothek „ASM" [1] verwendet.

Zur statischen Transformation bietet EclipseLink eine Ant-Task mit Namen „StaticWeaveAntTask" an, die hier „weave" heißt und die in den Attributen „source" und „target" das Quell- und Ziel-Jar der Transformation erwartet (siehe Listing 3).

Die dynamische Variante wird über einen Java-Agenten, der sich im Jar der EclipseLink-Implementierung befindet, realisiert.

```
java -javaagent:eclipselink.jar MainClass
```

Weitere Details sowohl zum statischen als auch zum dynamischen Weaving findet man in [2].

Der folgende Code zeigt das Ergebnis der Klassentransformation für die Methode „getAusweis()" (siehe Listing 4).

Um das späte Laden von Attributen zu ermöglichen, verwendet EclipseLink einen sogenannten „Value-Holder". Das Interface „ValueHolderInterface" im Package „org.eclipse.persistence.indirection" definiert Methoden zum Lesen und Schreiben von „Object"-Instanzen. Das oben verwendete Attribut „persistence_ausweis_vh" ist wie in Listing 5 deklariert. Das verwendete Interface ist ein Sub-Interface von „ValueHolderInterface". Beim Zugriff auf das Attribut wird geprüft, ob bereits ein Laden aus der Datenbank stattgefunden hat. Falls nicht, wird dies nachgeholt.

```
protected WeavedAttributeValueHolderIn-
terface
    _persistence_ausweis_vh;
```

Listing 5

Hibernate: Instrumentation

Hibernate nennt die Klassentransformation „Instrumentation" und stellt ebenfalls eine Ant-Task hierfür bereit. Intern wird die Bibliothek „Javassist" [3] verwendet, ebenfalls ein JBoss-Projekt. Das folgende Listing zeigt die Verwendung dieser Ant-Task. Eine alternative Realisierung mit einem Java-Agenten existiert bei Hibernate nicht (siehe Listing 6). Bemerkung: Hibernate unterstützt das späte Laden von 1:1-Beziehungen auch ohne explizite Instrumentierung. Diese bietet jedoch bestimmte Optimierungen und wird zum Beispiel für das korrekte Verhalten von Fetch-Groups zwingend benötigt.

Die „InstrumentTask" erweitert die Klassen um einen sogenannten „Field-Handler" (Interface „FieldHandler"). Das Lesen und Schreiben der Attribute wird nun über diesen „FieldHandler" realisiert. Listing 7 zeigt die Modifikation des Beispiel-Getters durch die „InstrumentTask".

Ein „FieldHandler" stellt für die primitiven Datentypen Lese- und Schreibmethoden bereit: „readInt()", „writeInt()", „readDouble()", „writeDouble()" etc. Die entsprechende Lesemethode für Objekte „readObject()" gibt Listing 8 wieder. Man erkennt hier, dass die Arbeit auf ein Proxy verlagert wird, das letztendlich für den SQL-Zugriff verantwortlich ist.

OpenJPA: Enhancement

Die Transformation von Entity-Klassen wird von OpenJPA „Enhancement" genannt. OpenJPA unterstützt sowohl die Transfor-

```
<target name="instrument">
        <taskdef name="instrument" classname=
„org.hibernate.tool.instrument.javassist.InstrumentTask">
                <classpath refid="classpath"/>
        </taskdef>
        <instrument verbose="true">
          <fileset dir="../build/classes">
            <include name="**/*.class"/>
          </fileset>
        </instrument>
</target>
```

Listing 6

```
public Studentenausweis getAusweis() {
      return $javassist_read_ausweis();
}
public Studentenausweis $javassist_read_ausweis(){
   if (getFieldHandler() == null) {
     return this.ausweis;
   }
   return (Studentenausweis) this.getFieldHandler
         .readObject(this, „ausweis", ausweis);
}
```

Listing 7

```
public Object readObject(Object target, String name,
                  Object oldValue) {
  Object value = intercept( target, name, oldValue );
  if (value instanceof HibernateProxy) {
    LazyInitializer li =
     ((HibernateProxy)value).getHibernateLazyInitializer();
    if ( li.isUnwrap() ) {
      value = li.getImplementation();
    }
  }
  return value;
}
```

Listing 8

```
  <target name="enhance">
    <path id="jpa.classpath">
      <pathelement location="../build/classes"/>
        <fileset dir="../lib">
          <include name="**/*.jar"/>
        </fileset>
    </path>
    <taskdef name="enhancer"
          classname="org.apache.openjpa.ant.PCEnhancerTask">
      <classpath refid="jpa.classpath"/>
    </taskdef>
    <enhancer>
      <classpath refid="jpa.classpath"/>
    </enhancer>
  </target>
```

Listing 9

```
public Studentenausweis getAusweis() {
   if (this.pcStateManager == null) {
         return pcgetAusweis();
   }
   int i = pcInheritedFieldCount + 0;
   this.pcStateManager.accessingField(i);
   return pcgetAusweis();
}
```

Listing 10

mation mit einer Ant-Task als auch die über einen Java-Agenten. Als Realisierungshilfe wird „Serp" [5] verwendet. Der folgende Ausschnitt eines Ant-Build-Files gibt die Struktur zur Verwendung der Task wieder.

Für das späte Laden von 1:1-Beziehungen muss das Enhancement erfolgen (siehe Listing 9).

Eine vollständige Übersicht über die Möglichkeiten des Enhancement von Entity-Klassen findet man in [6] und [7]. Das zentrale Interface von OpenJPA zur Zustandsverwaltung ist der „StateManager" im Package „org.apache.openjpa.enhance". Die überarbeitete Version des Beispiel-Getters zeigt der folgende Code-Ausschnitt (siehe Listing 10).

Wird auf den Getter der Entity-Klasse zugegriffen, so sorgt der „StateManager" durch die Methode „accessingField()" für das Laden des Attributs. Die auf den ersten Blick merkwürdige Berechnung des Wertes „i" durch Addition von 0 ist der Implementierung von OpenJPA geschuldet. Die Variable „i" repräsentiert einen Index in einem Array, in dem Eigenschaften wie Namen und Typen der Attribute hinterlegt sind. Die addierte Konstante (hier 0) wird zum Zeitpunkt des Enhancement berechnet und beschreibt den Index für dieses Attribut innerhalb des Arrays.

Gegenüberstellung

Tabelle 1 stellt die Möglichkeiten der statischen und dynamischen Byte-Code-Manipulation der drei JPA-Provider gegenüber. Außerdem sind die zur Byte-Code-Manipulation verwendeten Werkzeuge beziehungsweise Bibliotheken angegeben.

Fazit

Wir haben in diesem Artikel motiviert, dass JPA-Provider zur korrekten Imple-

	EclipseLink	Hibernate	OpenJPA
Byte-Code-Manipulation	Ant / Java-Agent	Ant	Ant / Java-Agent
Bibliothek	ASM	Javassist	Serp

Tabelle 1: Gegenüberstellung der JPA-Provider

mentierung der JPA-Spezifikation eine Reihe von Änderungen an Entity-Klassen vornehmen müssen. Diese Änderungen sollten sinnvollerweise nicht auf Quell-Code-Ebene, sondern im Byte-Code erfolgen. JPA sieht hierzu das Interface „ClassTransformer" vor. Um die eigentlichen Code-Transformationen vorzunehmen, verwenden die Provider verschiedene Werkzeuge, namentlich ASM, Javassist und Serp. Am Beispiel einer 1:1-Beziehung haben wir uns die oberste Ebene der entsprechenden Transformationen näher angeschaut und durchaus unterschiedliche Ansätze bei EclipseLink, Hibernate und OpenJPA erkennen können.

Weitere Informationen

[1] http://asm.ow2.org/
[2] http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_%28ELUG%29#What_You_May_Need_to_Know_About_Weaving_JPA_Entities
[3] http://www.jboss.org/javassist
[4] http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/performance.html
[5] http://serp.sourceforge.net/
[6] http://openjpa.apache.org/enhancement-with-ant.html
[7] http://openjpa.apache.org/docs/latest/ref_guide_pc_enhance.html#ref_guide_pc_enhance_unenhanced_types

Marc Steffens
m-th.steffens@ostfalia.de

Bernd Müller
bernd.mueller@ostfalia.de

Marc Steffens studiert IT-Management an der Ostfalia und beschäftigt sich in seiner Bachelor-Arbeit unter anderem mit den Möglichkeiten, die Java-Bytecode-Manipulationen bieten.

Bernd Müller ist Professor für Software-Technik an der Ostfalia. Er ist Autor mehrerer Bücher zu den Themen JSF, JPA und JBoss Seam.

Die iJUG-Mitglieder auf einen Blick

iJUG Verbund

Java User Group Deutschland e.V.
http://www.java.de

DOAG Deutsche ORACLE Anwendergruppe e. V.
http://www.doag.org

Java User Group Stuttgart e.V. (JUGS)
http://www.jugs.de

Java User Group Köln
http://www.jugcologne.eu

Java User Group München (JUGM)
http://www.jugm.de

Java User Group Metropolregion Nürnberg
http://www.source-knights.com

Java User Group Ostfalen
http://www.jug-ostfalen.de

Java User Group Saxony
http://www.jugsaxony.org

Sun User Group Deutschland e.V.
http://www.sugd.de

Swiss Oracle User Group (SOUG)
http://www.soug.ch

Der iJUG möchte alle Java-Usergroups unter einem Dach zu vereinen. So können sich alle interessierten Java-Usergroups in Deutschland, Österreich und der Schweiz, die sich für den Verbund interessieren und ihm beitreten möchten, gerne beim iJUG melden unter: office@ijug.eu

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.