Objektorientierte Datenbanken

Vorlesung 5 vom 06.05.2004

Dr. Sebastian Iwanowski
FH Wedel

Offene Fragen vom letzten Mal:

Was passiert bei Dateninkonsistenzen ?

- **FastObjects vergleicht die Daten bei jedem enhance mit der aktuellen Spezifikation und löscht Daten, die nicht dazu passen.**

- **Aus diesem Grund erlaubt FastObjects Veränderungen nur an der lokalen Datenbank (und nicht auf entfernten)**

Von welcher Klasse sind die Rückgabewerte einer select-query ?

- **Die Rückgabewerte können anfragetypabhängig der Klasse SetOfObject, ArrayOfObject, String, Integer und weiteren Klassen angehören.**

  Daher sollte jedes result auf eines dieser Möglichkeiten geprüft werden.

Inhalt heute: Einführung in JDO

Entstehungsgeschichte und Ziele von JDO

Überblick über den Leistungsumfang von JDO

Grundlagen Persistenz / Transaktionen

Wiederholung / Vertiefung mit Arno Schmidhauser (HTA Bern)

Zusamenfassung: Erste Anwendung mit JDO

Entstehungsgeschichte und Ziele von JDO

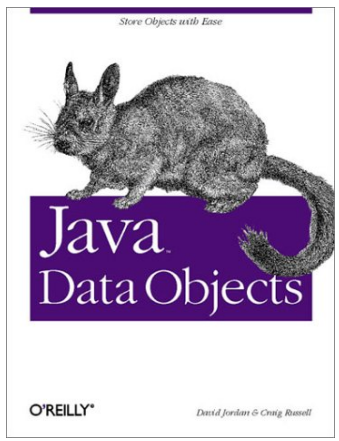
Entstehungsgeschichte von JDO

OODB-Zugriffsstandard, exklusiv für Java

entwickelt von Sun, mit Implementierung

Förderer: Rick Cattell (Sun, Leiter der ODMG 1991)

maßgeblich beteiligt: David Jordan, Craig Russell (Buchautoren)



entstanden 1999, eingeführt 2000, seitdem sukzessiv weiterentwickelt

Ziele von JDO

Vereinheitlichung und Verbesserung der folgenden älteren Persistenztechniken für Java:

- **Serialisierung**

  Kommunikation von Java mit anderer Software über Streams

- **JDBC (Java DataBase Connectivity)**

  SQL-Anfragen direkt aus dem Java-Programm heraus

- **EJB CMP (Enterprise Java Beans Container Managed Persistence)**

  Java in verteilten Anwendungen (sehr komplex)

Überblick über den Leistungsumfang von JDO

Leistungsumfang von JDO

- **Persistenzkonzept**

  Persistenzfähige Klassen, objektspezifische Persistenz, Persistenz durch Erreichbarkeit

- **Transaktionskonzept**

  mehrere Transaktionsmagementstrategien

- **Anfragesprache (JDOQL)**

  mit objektorientiertem Fokus, Java-Syntax

- **Konzept zum Management von Datenveränderungen**

  über so genannte Lebenszykluszustände von Daten
  definiert Mechanismen für den Lebenszyklusübergang

- **Datenidentitätskonzepte**

  berücksichtigt unterschiedliche Anforderungen von Datenbank und Programm

Unterschied zum ODMG-Standard

- **Persistenzkonzept**

  gleiches Prinzip wie Java-Anbindung der ODMG, aber genauere Beschreibungsvorgaben, in JDO wird die Zusatzinformation in XML festgelegt (entspricht ptj.opt-Datei bei ODMG).

- **Transaktionskonzept**

  wesentlich umfangreicher als ODMG: ODMG kennt nur ein Konzept, legt Details nicht fest.

- **Anfragesprache (JDOQL)**

  sehr verschieden von OQL: OQL ist eher SQL-basiert.
  Leistungsumfang von JDOQL mit OQL nicht vergleichbar.

- **Konzept zum Management von Datenveränderungen**

  in ODMG nicht festgelegt

- **Datenidentitätskonzepte**

  in ODMG nicht festgelegt

Grundlagen Persistenz / Transaktionen

Grundlagen Persistenz / Transaktionen

In JDO vorgeschriebene Java-Interfaces:

(alle in Package javax.jdo)

- **PersistenceManagerFactory**

  generiert Instanzen des Interfaces PersistenceManager,
  speichert die Verbindung zur zugehörigen Datenbank,
  legt die zu den Interfaces gehörenden Klassen des JDO-Anbieters fest.

- **PersistenceManager**

  generiert Instanzen der Interfaces Transaction und Extent,
  stellt damit die Funktionalität für Transaktionen, Anfragen und Persistenzoperationen zur Datenbank der zugehörigen PersistenceManagerFactory bereit.

- **Transaction**

  beginnt und beendet Transaktionen

- **Extent**

  liefert Extents zu einer gegebenen Klasse zur weiteren Bearbeitung

Grundlagen Persistenz / Transaktionen

Generierung einer PersistenceManagerFactory:

über die Klasse JDOHelper (in javax.jdo):

```
PersistenceManagerFactory pmf =
            JDOHelper.getPersistenceManagerFactory( properties );
```

- **`properties`** muss Instanz der Klasse java.util.Properties sein (ist Spezialisierung von Hashtable und damit von Dictionary).
- In **`properties`** wird die Zuordnung zur Datenbank und zu den Implementierungsklassen festgelegt.

Beispiel (für FastObjects):

```
java.util.Properties pmfProps = new java.util.Properties();

pmfProps.put("javax.jdo.PersistenceManagerFactoryClass",
             "com.poet.jdo.PersistenceManagerFactories" );
         // names the class responsible for creating the
         // FastObjects persistence manager factory implementation
pmfProps.put("javax.jdo.option.ConnectionURL", "fastobjects://LOCAL/MyBase" );
         // names a connection to the database specified by the URL
PersistenceManagerFactory pmf =
            JDOHelper.getPersistenceManagerFactory( pmfProps );
```

Grundlagen Persistenz / Transaktionen

Generierung der anderen benötigten Objekte:

- **PersistenceManager**

```
PersistenceManager pm = pmf.getPersistenceManager();
```

pmf muss das Interface javax.jdo.PersistenceManagerFactory implementieren

- **Transaction**

```
Transaction txn = pm.currentTransaction();
```

pm muss das Interface javax.jdo.PersistenceManager implementieren

- **Extent**

```
Extent ext = pm.getExtent( someClass.getClass(), true );
```

pm muss das Interface javax.jdo.PersistenceManager implementieren

Der 1. Parameter muss persistenzfähig sein. Der 2. Parameter legt fest,
ob die Instanzen der Unterklassen im Extent enthalten sein sollen oder nicht.

Grundlagen Persistenz / Transaktionen

Für den Anfang benötigte Methoden:

Interface PersistenceManagerFactory

public PersistenceManager getPersistenceManager ();

Interface PersistenceManager

public Transaction currentTransaction ();

public getExtent
(Class persistenceCapableClass,
***boolean subclasses**);*

public void makePersistent (Object obj);

Objektbindung durch Name geht nicht !

public void deletePersistent (Object obj);

Interface Transaction

public void begin();
public void commit();
*public void **rollback**();*

Interface Extent

public Iterator iterator ();

Interface java.util.Iterator

public boolean hasNext ();
public Object next ();

weitere Methoden: in FastObjects-JavaAPI nachsehen !

Grundlagen Persistenz / Transaktionen

Festlegung der persistenzfähigen Klassen in XML-Datei:

Beispiel (minimale XML-Datei, um someClass persistenzfähig zu machen)**:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jdo SYSTEM "jdo.dtd">
<jdo>
  <package name="">
    <class name="someClass">
    </class>
  </package>
</jdo>
```

FastObjects-spezifische DTD

Original-DTD für JDO von Sun in
`http://java.sun.com/dtd`

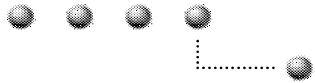
! ACHTUNG: someClass muss Konstruktor someClass () haben !

- Die Datei muss heißen: **`someClass.jdo`**

  Sie kann nur die Klasse **`someClass`** spezifizieren.

- Eine Datei zur Spezifikation mehrerer Klassen muss heißen: **`package.jdo`**

Wiederholung / Vertiefung
nach dem Skript von
Arno Schmidhauser (HTA Bern)



Berner Fachhochschule

Hochschule für Technik und Architektur Bern
Software-Schule Schweiz

Java Data Objects

Letzte Revision: Februar 2003

Dr. Arno Schmidhauser
Software-Schule Schweiz
Morgartenstr. 2c
3014 Bern
arno.schmidhauser@hta-be.bfh.ch
+41313355275

http://www.hta-be.bfh.ch/~schmd/jdo

Was ist JDO

1. Eine Allzweck-Architektur um Java-Objekte in einer Datenbank abzulegen
   1. unabhängig vom Datenmodell (relational, objekt-orientiert, ...)
   2. unabhängig von der Umgebung (Client-Server, Applikationsserver)

2. Beliebig auswechsel- und kombinierbare "Driver" für verschiedene Datenbanktypen und Hersteller.

3. Durch verschiedenste Interessengruppen abgesegnete Spezifikation.

JDO Anwendungskonzept

1. Typsystem von Java, keine speziellen Datentypen wegen Persistenz -> Arrays, Collections, Utility-Klassen, Benutzer-Klassen.

2. Modifikation von Objekten mit Java-Befehlen (Zuweisung), keine speziellen Operatoren wie in SQL.

3. Von der Applikation kontrolliert werden lediglich der Lebenszyklus von Objekten (persistent, transient) und die Transaktionsgrenzen (begin, rollback, commit).

4. Zugriff auf Objekte in der Datenbank via Extent einer Klasse, via bekannte Objekt-ID, via Abfragesprache.

Arbeiten mit JDO

Erzeugen persistenter Objekte

```
Properties p = new Properties();
...
PersistenceManagerFactory pmf;
pmf = JDOHelper.getPersistenceManagerFactory( p );
PersistenceManager pm = pmf.getPersistenceManager();
Transaction tra = pm.currentTransaction();
tra.begin();
PMQ pmq = new PMQ(  name );
pm.makePersistent( pmq );
tra.commit();
...
```

Automatische Persistenz

1. Jedes transiente Objekt, welches an ein persistentes Objekt angehängt wird, ist automatisch persistent.

```
...
PMQ pmq = new PMQ(  name );
pm.makePersistent( pmq );
Message m = new SimpleMessage( ... );
pmq.append( m );
...
```


m wird persistent !

2. Ein persistentes Objekt wird nur durch `PersistenceManager.deletePersistent()` wieder aus der Datebank entfernt

Aufsuchen persistenter Objekte

1. Der wichtigste Schritt ist das Holen von Objekten aus der Datenbank. Es gibt drei grundsätzliche Möglichkeiten, welche eine JDO-Implementation anbieten muss:

    1. Via Extent einer Klasse
    2. Via Query über eine beliebige Collection von Objekten
    3. Via ObjektID

2. Die Modifikation von persistenten Objekten geschieht 100% wie bei transienten Objekten ohne Verwendung einer Datenbank.

3. Zum Commit-Zeitpunkt werden modifizierte Objekte in die Datenbank zurückgeschrieben. Welche das sind, weiss das Datenbank-Framework, der Entwickler ist von der Buchführung darüber vollständig entlastet.

Extent

1. Der Extent ist ein *logisches* Gefäss für die Menge aller Objekte einer bestimmten Klassen, allenfalls auch ihrer Unterklassen.
2. Das Konstruieren des Extends beinhaltet noch nicht das Holen der Objekte aus der Datenbank.
3. Der Extent stellt einen Iterator zur Vefügung. Der Iterator bestimmt den Algorithmus für das Abholen der Objekte aus der Datenbank.
4. Queries können über einen Extent oder eine Collection durchgeführt werden. Queries über einen Extent werden *serverseitig* abgewicket, Queries über eine Collection *clientseitig*.

Extent, Beispiel

```
...
PersistenceManager pm = pmf.getPersistenceManager();
tra.begin();
Extent e = pm.getExtent( PMQ.class, true );
Iterator it = e.iterator();
if ( it.hasNext() ) {
  PMQ pmq = (PMQ) it.next();
  if ( pmq.getName().equals( "myQueue" ) ) {
    SimpleMessage sm = new SimpleMessage( ... );
    pmq.append( sm );
  }
}
tra.commit();
...
```

JDO Persistenzmodell

- Grundsatz / Anforderungen der Datenbank
- Enhancer
- Konfigurationsdate
- First Class / Second Class Objekte
- spezielle Felder

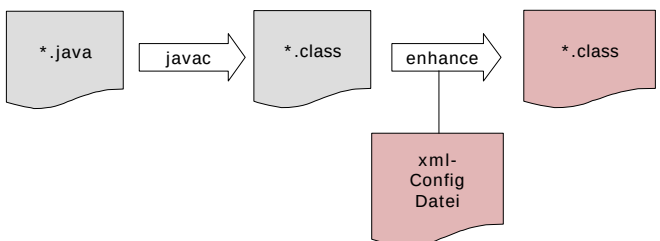
Grundsatz

- Für den Entwickler sollen sich das Arbeiten mit der Datenbank auf gewisse *logische* Aufgaben beschränken:
  - Objekte als persistent markieren oder löschen
  - Transaktion starten, comitten oder rollbacken
  - Objekte gezielt aus der Datenbank holen via Query oder Objekt-ID
- Für den programmiersprachlichen Umgang mit Objekten gilt:

  Arbeiten mit Datenbankobjekten = Arbeiten mit Java-Objekten

JDO Enhancer

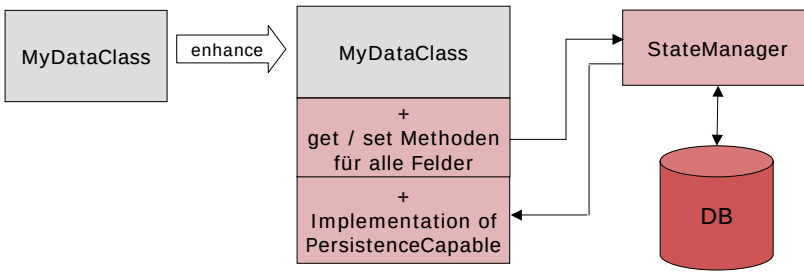
- Die Umprogrammierung des Java-Codes (Enhancement) findet auf Ebene Bytecode, also mit den Class-Files statt.



JDO Enhancer

- In Java-Klassen, welche mit persistenten Objekten *arbeiten*, werden die Operatoren, welche diese Objekte verändern, in Methoden*aufrufe* umprogrammiert.

- In Java-Klassen, welche *selber persistente Objekte* repräsentierten, werden die entsprechenden Methoden durch den Enhancer *implementiert*.



XML Konfigurationsdatei

- Das Konfiguartionsfile beschreibt die persistenten Klassen.
- Wird für den Postprocessor (Enhancer) und zur Laufzeit der Applikation benötigt.

Beispiel: Datei pmq.jdo

```
<jdo>
  <package name="pmq">
    <class name="PMQ"/>
    <class name="Message" identity-type="datastore"/>
    <class name="ImageMessage"
            persistence-capable-superclass="pmq.Message"/>
      <class name="SimpleMessage"
            persistence-capable-superclass="pmq.Message"/>
      <class name="CompositeMessage"
            persistence-capable-superclass="pmq.Message"/>
 </package>
</jdo>
```

Zusammenfassung: Erste Anwendung mit JDO

Zusammenfassung: Erste Anwendung mit JDO

- Persistenzfähige Klassen in XML-Datei beschreiben
- Datenbankanbindung über Properties an PersistenceManagerFactory
- Alle anderen Operationen über PersistenceManager erreichbar
- Vorerst können Objekte nur via Extent aus Datenbank geholt werden
  - Objektbindung durch Name geht nicht !
- Alle anderen Prinzipien wie bei ODMG:
  - Persistenz durch explizite Deklaration
  - Persistenz durch Erreichbarkeit
  - Datenbankoperationen nur in Transaktionen (Ausnahmen kommen später)
  - Persistenzfähige Klassen müssen nach jeder Änderung enhanced werden.