Objektorientierte Datenbanken

Vorlesung 3

Sebastian Iwanowski
FH Wedel

JDO: 1. Teil

Entstehungsgeschichte und Ziele von JDO

Überblick über den Leistungsumfang von JDO

JDO – Grundlagen

Im Detail: Persistenzkonzept von JDO

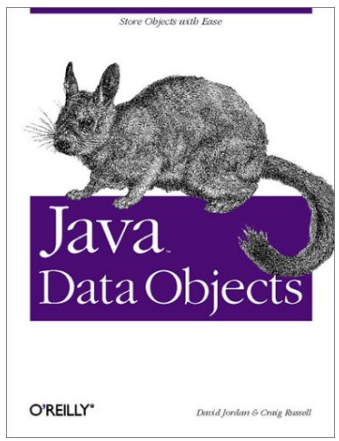
Entstehungsgeschichte von JDO

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

Neuer Standard seit 2005: JDO-2 (noch nicht verabschiedet)

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

  auf höherer Abstraktionsebene ersetzt durch OR-Mapper, heute meistgenutzt: Hibernate

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

  Java in verteilten Anwendungen (sehr komplex)

  weiterentwickelt zu EJB-2 als Teil von J2EE, zukünftiger Standard: EJB-3

Leistungsumfang von JDO

- **Persistenzkonzept**

  Persistenzfähige Klassen, objektspezifische Persistenz, Persistenz durch Erreichbarkeit

- **Transaktionskonzept**

  mehrere Transaktionsmagementstrategien

- **Anfragesprache (JDOQL)**

  Filtersprache mit objektorientiertem Fokus, Java-Syntax
  sehr verschieden von SQL

- **Konzept zum Management von Datenveränderungen**

  über so genannte Lebenszykluszustände von Daten
  definiert Mechanismen für den Lebenszyklusübergang

- **Datenidentitätskonzepte**

  berücksichtigt unterschiedliche Anforderungen von Datenbank und Programm

JDO - Grundlagen

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 (keys für Versant Object Database):

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

pmfProps.put("javax.jdo.PersistenceManagerFactoryClass",
             "com.versant.core.jdo.BootstrapPMF" );
         // names the class responsible for creating the
         // Versant persistence manager factory implementation
pmfProps.put("javax.jdo.option.ConnectionURL", "versant:myDatabase@host" );
         // names a connection to the database specified by the respective 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 Versant-JDO-API nachsehen !

Metadaten und Enhancement

Wie macht JDO Klassen persistenzfähig?

- **Spezifikation von persistenzfähigen Klassen in speziellen XML-Dateien (genannt *Metadaten*)**
- **Alle kompilierten persistenzfähigen Klassen müssen vor Benutzung durch den Persistenzmanager *enhanced* werden**

Alle modernen Persistenzstandards arbeiten mit XML-Metadaten und Enhancement (teilweise)

Ältere Konzepte (ODMG und noch älter):

- **Einrichtung von Spezialklassen für die Basisklassen (in JDO nicht erforderlich)**

Metadaten und Enhancement

Festlegung der persistenzfähigen Klassen in XML-Datei:

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

```
<?xml version="1.0" encoding="UTF-8"?>
<jdo>
  <package name="">
    <class name="someClass">
    </class>
  </package>
</jdo>
```

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`**

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 (Java binding):**
  - **Persistenz durch explizite Deklaration**
  - **Persistenz durch Erreichbarkeit**
  - **Datenbankoperationen nur in Transaktionen (Ausnahmen kommen später)**

Im Detail: Persistenzkonzept von JDO

Persistenzkonzept: Persistenzfähige Klassen

Definition in XML-Datei

Klassendefinitionen: (Ausschnitt der DTD von Sun: http://java.sun.com/dtd)

```
<!ELEMENT class (field|extension)*>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST class requires-extent (true|false) 'true'>
<!ATTLIST class persistence-capable-superclass CDATA #IMPLIED>
```

Persistenzkonzept: Persistenzfähige Klassen

Definition in XML-Datei

Attributmanagement (field management):

(Ausschnitt der DTD von Sun: http://java.sun.com/dtd)

```
<!ELEMENT field ((collection|map|array)?, (extension)*)?>
<!ATTLIST field name CDATA #REQUIRED>
<!ATTLIST field persistence-modifier
                        (persistent|transactional|none) #IMPLIED>
<!ATTLIST field embedded (true|false) #IMPLIED>
```

Persistenzkonzept: Attributmanagement

Drei Typen für die Persistenz durch Erreichbarkeit:

- **persistent**

  normale Persistenz durch Erreichbarkeit, **`rollback()`** bei Transaktionen wirksam

  Zugelassene Attributdatentypen:

  - alle einfachen Datentypen
  - zu einfachen Datentypen gehörende Objektdatentypen
  - alle selbst definierten persistenzfähige Klassen
  - weitere spezielle Klassen

- **transactional**

  Daten sind immer transient, aber **`rollback()`** bei Transaktionen wirksam

  Zugelassene Attributdatentypen: Alle

- **none**

  Daten sind immer transient, **`rollback()`** bei Transaktionen setzt sie nicht zurück

  Zugelassene Attributdatentypen: Alle

Persistenzkonzept: Attributmanagement

Drei Typen für die Persistenz durch Erreichbarkeit:

- **persistent**

  unveränderbare Objekte: Instanzen von

  - allen einfachen Datentypen (z.B. **`int`**)
  - zu einfachen Datentypen gehörenden Klassen (z.B. **`Integer`**)
  - **`Locale, BigDecimal, BigInteger, String`**
  - allen selbst definierten persistenzfähigen Klassen ohne Änderungsmöglichkeiten

  veränderbare Objekte: Instanzen von

  - **`Date, HashSet`**
  - weiteren optionalen Klassen wie **`ArrayList, Vector,`** etc.
  - allen selbst definierten persistenzfähigen Klassen mit Änderungsmöglichkeiten

Unterscheidung aus Anwendersicht:

Nur unveränderbare Objektinstanzen dürfen mit
gleicher Identität an mehreren Stellen verwendet werden !

Persistenzkonzept: Attributmanagement

Zwei Typen für die Objektreferenzierung in der Datenbank:

- **First Class Objects (`embedded = false`):**

  eigene ObjektId

  existiert auch von alleine in der Datenbank

  notwendig, falls mehrere Instanzen auf dieses Objekt zeigen

- **Second Class Objects (`embedded = true`):**

  keine eigene ObjektId

  existiert nur in Verbindung mit referenzierendem Objekt in der Datenbank

  nur möglich, wenn nur eine Instanz auf dieses Objekt zeigt

Die persistenzfähigen Systemklassen sind second class per default

in vielen JDO-Implementierungen gar nicht erlaubt als first class

für second-class-Systemklassen: **`deletePersistent()`** nicht nötig und nicht erlaubt

Persistenzkonzept: Collections

Definition in XML-Datei

Ausschnitt der DTD von Sun (`http://java.sun.com/dtd`):

```
<!ELEMENT field ((collection|map|array)?, (extension)*)?>

<!ELEMENT collection (extension)*>
<!ATTLIST collection element-type CDATA #IMPLIED>
<!ATTLIST collection embedded-element (true|false) #IMPLIED>

<!ELEMENT map (extension)*>
<!ATTLIST map key-type CDATA #IMPLIED>
<!ATTLIST map embedded-key (true|false) #IMPLIED>
<!ATTLIST map value-type CDATA #IMPLIED>
<!ATTLIST map embedded-value (true|false) #IMPLIED>

<!ELEMENT array (extension)*>
<!ATTLIST array embedded-element (true|false) #IMPLIED>
```