Hibernate (OR-Mapping)

Roland Furrer

Urs Frei

Inhalt

- Beispielanwendung
- Herkömmlicher Datenbankzugriff
- Hibernate
  - Idee
  - Bestandteile
  - Beispielanwendung
  - Collection
  - Vererbung
  - Datenbankabfragen
- Hibernate mit XDoclets
- Hibernate mit Annotationen

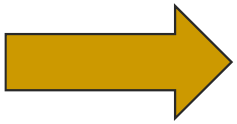
Beispielanwendung

Person
- Kürzel
- Vorname
- Nachname

Termine
- Datum
- Uhrzeit
- Titel
- Beschreibung

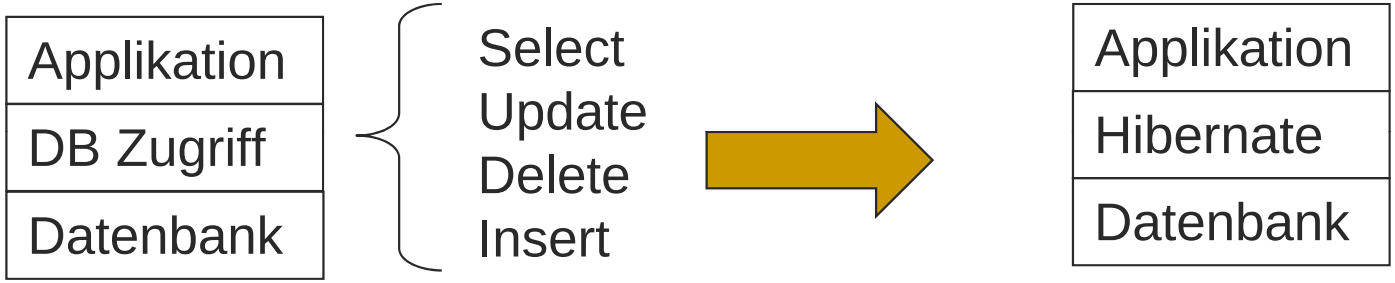
Herkömmlicher Datenbankzugriff

- JDBC
  - update
  - insert
  - delete
  - select
  - create
- Connectionverwaltung



Es geht auch einfacher!

Hibernate Idee



Zugriff auf DB wird nicht mehr selber programmiert!!

Grundelemente

Applikation	Anwendung in Java
Hibernate	Open-Source-Produkt unter www.hibernate.org
Datenbank	Open-Source-Produkt HSQLDB unter www.hsqldb.org

Bestandteile der Beispielanwendung

- DB config File (xml)
  - Definiert die Verbindung zur DB
  - Art der DB
- Mapping File (xml)
  - Welches Javaobjekt gehört zu welcher Tabelle
- Java Klassen (Java Beans)

Beispiel Hibernate Configfile

```
<hibernate-configuration>
	<session-factory>
		<property name="dialect">
			org.hibernate.dialect.HSQLDialect
		</property>
		<property name="connection.driver_class">
			org.hsqldb.jdbcDriver
		</property>
		<property name="connection.username">
			sa
		</property>
		<property name="connection.password"></property>
		<property name="connection.url">
			jdbc:hsqldb:file:db/termine
		</property>
		<mapping resource="net/sf/hibernatesample/einfach/Termin.hbm.xml" />
	</session-factory>
</hibernate-configuration>
```

Beispiel Hibernate Mappingfile

```
<hibernate-mapping package= "net.sf.hibernatesample.einfach">
    <class name="Termin">
        <id name="id">
            <generator class="native"/>
        </id>
        <property name="titel"/>
        <property name="beschreibung"/>
        <property name="zeitPunkt"/>
        <property name="ort"/>
    </class>
</hibernate-mapping>
```

Beispiel Java-Klasse

```
package net.sf.hibernatesample.einfach;
import java.util.Date;
public class Termin {
  private long id;
  private String titel;
  private String beschreibung;
  private String ort;
  private Date zeitPunkt;

  public String getTitel ()
        { return titel; }
  public void setTitel (String titel)
        { this.titel = titel; }

  public Date getZeitPunkt ()
        { return zeitPunkt; }
  public void setZeitPunkt (Date zeitPunkt)
        { this.zeitPunkt = zeitPunkt; }
.....}
```

Tabellen mit Hibernate erzeugen

Erzeugen der Tabelle in der Java-Anwendung:

```
Configuration configuration = new Configuration().configure();
SchemaExport export = new SchemaExport(configuration);
export.create(false, true);
```

Resultierendes Statement für HSQLDB:

```
create table Termin (
    id bigint generated by default as indentity (start with 1),
    titel varchar(255),
    beschreibung varchar(255)
    zeitPunkt timestamp,
    ort varchar(255)
    primary key (id)
)
```

Erzeugung eines persistenten Objekts

```
private long erzeugeTermin(String titel, String beschreibung, String ort, Date zeitPunkt) {
        Termin termin = new Termin();
        termin.setTitel(titel);
        termin.setBeschreibung(beschreibung);
        termin.setOrt(ort);
        termin.setZeitPunkt(zeitPunkt);
        Session session = null;
        Transaction transaction = null;
        try {
            session = sessionFactory.openSession();
            transaction = session.beginTransaction();
            session.save(termin);
            transaction.commit();
        }
        catch (HibernateException e) { ... }
         ....
         return termin.getID();
}
```

Laden, suchen, aktualisieren, löschen eines persistenten Objekts

Laden:

```
Termin termin = (Termin) session.load(Termin.class, id);
```

Suchen:

```
Query query = session.createQuery("from Termin where ort=‘ " + ORT + “ ‘ ");
List result = query.list();
Termin termin = (Termin) result.get(0);
```

Aktualisieren:

```
Termin termin = (Termin) session.load(Termin.class, id);
Transaction transaction = session.beginTransaction();
termin.setTitel("neuer Termin");
transaction.commit();
```

Löschen:

```
Termin termin = (Termin) session.load(Termin.class, id);
Transaction transaction = session.beginTransaction();
session.delete(termin);
transaction.commit();
```

Details der Mapping-Dateien

```
<hibernate-mapping package = "net.sf.hibernatesample.einfach">
  <class name="Termin" table="termin">
    <id name="id">
      <generator class="native"/>
    </id>

    <property name="titel"/>
    <property name="beschreibung"/>
    <property name="zeitPunkt" column="zeit" type="timestamp" />
    <property name="ort" type="string" length="40" />
  </class>
</hibernate-mapping>
```

Objekte als Klassenmember

Wir ändern die Klasse Termin so, dass der Zeitpunkt nicht mehr in einem java.util.Date gespeichert wird, sondern in einem Zeitpunkt.

```
public class Zeitpunkt {
        private int minute, stunde, tag , monat, jahr;
        .... //mit set- und get-Methoden
}

public class Termin {
        private Zeitpunkt zeit;
        ...//mit set- und get-Methode
}
```

Objekte als Klassenmember

Die Mapping-Datei ändert sich nun wie folgt:

```
<hibernate-mapping package = "net.sf.hibernatesample.einfach">
  <class name="Termin" table="termin">
    <id name="id">
      <generator class="native"/>
    </id>
    <property name="titel"/>
    <property name="beschreibung"/>
    <component name="zeit">
       <property name="minute" />
       <property name="stunde" />
       <property name="tag" />
       <property name="monat" />
       <property name="jahr" />
    </component>
    <property name="ort" type="string" length="40" />
  </class>
</hibernate-mapping>
```

Umgang mit Collections

Es soll nun möglich sein, einen Termin nicht an einem definierten Zeitpunkt abzuhalten sondern vorgängig alternative Zeitpunkte zu definieren:

```
public class Termin {
    private List<Zeitpunkt> alternativZeiten = new ArrayList<Zeitpunkt>();
    ...//mit set- und get-Methode
}
```

In der DB wird eine neue Tabelle ALT_ZEITPKT erzeugt. Die Tabelle hat einen Fremdschlüssel auf die Tabelle TERMIN. Ausserdem braucht die Tabelle eine Spalte LFD_NR, die für alle Elemente in der Liste festlegt, in welcher Reihenfolge sie stehen.

ALT_ZEITPKT						
id	lfd_nr	Minute	stunde	tag	monat	jahr
1	2	22	18	1	6	2006
1	0	0	12	1	6	2006
1	1	0	18	7	6	2006
2	0	0	20	7	6	2006

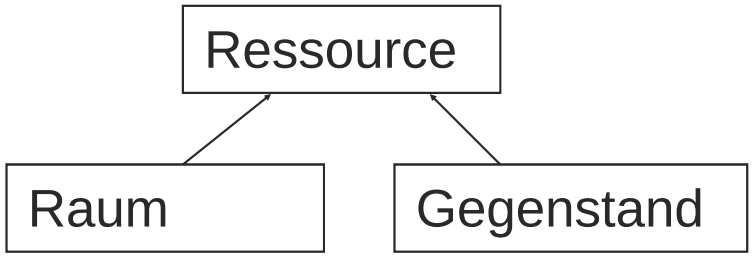
Umgang mit Collections

Die Mapping-Datei sieht dann wie folgt aus:

```
<hibernate-mapping package = "net.sf.hibernatesample.einfach">
  <class name="Termin" table="termin">
    ...
    <list name="alternativeZeiten" table="ALT_ZEITPKT">
       <key column="id"/>
       <list-index column="lfd_nr"/>
       <composite-element class="hibtest.entity.Zeitpunkt">
              <property name="minute" />
              <property name="stunde" />
              <property name="tag" />
              <property name="monat" />
              <property name="jahr" />
       </composite-element>
    </list>
    ....
 </class>
</hibernate-mapping>
```

Mit Set, SortedSet, Map, SortedMap, etc. wird jeweils entsprechend verfahren.

Vererbung in Hibernate




```
public abstract class Ressource {
    private long id;
    private String name;
    .../// set- und get-Methoden
```

```
public class Raum extends Ressource {
    private String gebaeude;
    .../// set- und get-Methode
```

```
public class Gegenstand extends Ressource {
    private double wert;
    .../// set- und get-Methode
```

Vererbung in Hibernate

Tabelle je Klassenhierarchie: Die ganze Vererbung in einer einzigen Tabelle!

Ressource				
id	disc	name	gebaeude	wert
1	r	Physikzimmer	C1	
2	g	Beamer		2500.00

```
<class name=„Ressource">
    <id name="id">
        <generator class="native"/>
    </id>
    <property name="name"/>
    <discriminator column=„disc“ type=„string“ lenght=„1“/>
    <subclass name=„Raum“ discriminator-value=„r“>
        <property name=„gebaeude“/>
    </subclass>
    <subclass name=„Gegenstand“ discriminator-value=„g“>
        <property name=„wert“/>
    </subclass>
 </class>
```

Vererbung in Hibernate

Tabelle je konkrete Klasse

Raum		
id	name	gebaeude
1	Physikzimmer	C1

Gegenstand		
Id	name	wert
2	Beamer	2500.00

```
<class name=„Ressource">
    <id name="id">
        <generator class="native"/>
    </id>
    <property name="name"/>
    <union-subclass name=„Raum“ table=„raum“>
        <property name=„gebaeude“/>
    </union-subclass>
    <union-subclass name=„Gegenstand“>
        <property name=„wert“/>
    </union-subclass>
</class>
```

Vererbung in Hibernate

Tabelle je Klasse

Ressource	
id	name
1	Physikzimmer
2	Beamer

Raum	
id	gebaeude
1	C1

Gegenstand	
id	wert
2	2500.00

```
<class name=„Ressource">
    <id name="id">
        <generator class="native"/>
    </id>
    <property name="name"/>
    <joined-subclass name=„Raum“ table=„raum“>
        <key column=„id“ />
        <property name=„gebaeude“/>
    </joined-subclass>
    <joined-subclass name=„Gegenstand“>
         <key column=„id“ />
        <property name=„wert“/>
    </joined-subclass>
</class>
```

Datenbankabfragen

- HQL sehr ähnlich SQL

```
Query query = session.createQuery("from Termine");
List<Termin> alleTermine = query.list();
```

- Criteria

```
Criteria crit = session.createCriteria(Termin.class);
crit.add(Restrictions.eq("ort", "Hamburg"));
List<Termin> termine = criteria.list();
```

- SQL auch möglich

Hibernate: Ziel erreicht?

- Vorteile
  - Keine SQL Statements
- Problem
  - Definitionsredundanz Java und XML
- Lösungsansätze
  - XDoclet
  - Annotationen

Hibernate mit XDoclet

- XDoclet:
  - XML Mappingfile generieren
  - Informationen für Generierung aus Javadoc

Hibernate mit XDoclet Bsp. Java

```
/**
 * @hibernate.class
 */
public class Termin {
        private long id;

        ...
        /**
         * @hibernate.property
         */
        public String getTitel() {
                return titel;}
        public void setTitel(String titel) {
                this.titel = titel;             }

        /**
         * hibernate.one-to-one
         * @hibernate.many-to-one
         */
        public Benutzer getInitiator() {
                return initiator;}
}
```

Hibernate mit XDoclet (3) Vor-Nachteile

- Vorteil:
  - Mapping Infos näher bei Sourcen
    - Einfacheres Ändern
    - Konfiguration an einem Ort
- Nachteil:
  - XML Mappingfile immer noch erforderlich
  - Muss z.B. mit Ant generiert werden

Hibernate Annotationen

- Annotationen
  - Ermöglicht Metainformationen im Code
  - Annotationsprozessoren bei Kompilierung
  - In Reflection verwenden
  - „Weiterentwicklung“ XDoclet
  - In Java 1.5 enthalten

Hibernate Annotationen Bsp.

```
@Entity
public class Termin {
	private long id;
	…
	public String getTitel() {
		return titel;}
	public void setTitel(String titel) {
		this.titel = titel;   }

	@ManyToOne(targetEntity = Benutzer.class)
	public Benutzer getInitiator() {
		return initiator;}
}
```

Hibernate Annotationen Vor-Nachteil

- Vorteil:
  - Kein XML Mappingfile mehr
- Nachteil:
  - Java 1.5 wird benötigt. Nachteil?

Warum Hibernate einsetzen?

Bsp. Programmieraufwand für eine Tabelle

Hibernate	JDBC
@Entity	Create Table
	Select
	Insert
	Delete
1 Codezeile	**xxx Codezeilen**
Es funktioniert	Funktioniert es????

Referenzen

- Hibernate www.dpunkt.de ISBN 3-89864-371-9
- www.hibernate.org
- java.sun.com
- www.hsqldb.org
- http://xdoclet.sourceforge.net/xdoclet/index.html