# HIBERNATE

## Eine Einführung

# Inhalt

# Einleitung (1) - Geschichtliches

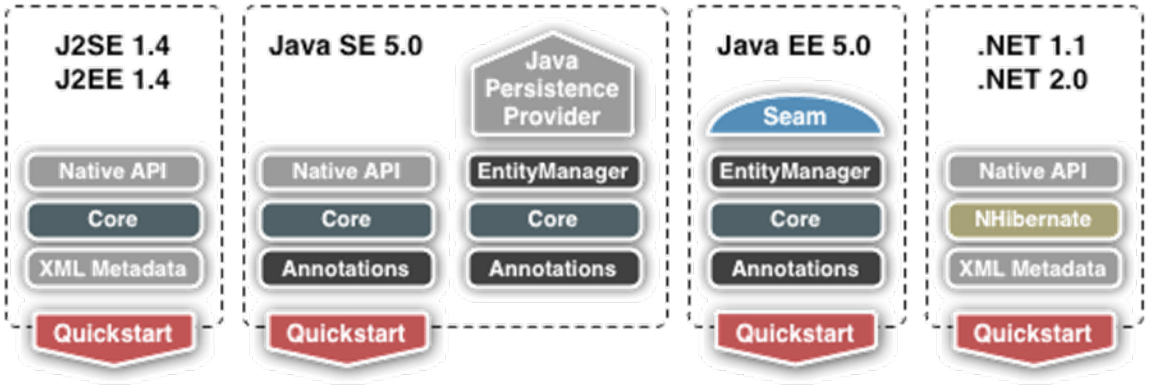


- 2001 empfand Gavin King die Standardpersistenzlösung (CMP Entity Beans) als unbefriedigend und begann eigene Entwicklung einer Persistenzlösung
- Diese Anforderungen nahmen überhand (gerechnet auf ein sehr kleines Entwickler-Team)
- Übergabe Hibernates an JBoss Inc. und fortan full-time Entwicklung
- Hibernate hat den EJB3 – Standard (JPA) maßgeblich mit beeinflusst

# Einleitung (2)



- Hibernate ist ein modular aufgebauter O/R – Mapper und JPA – Engine
- Hibernate unterstützt die Arbeit mit bestehenden DB – Schemata, oder erstellt diese auch selbstständig
- Hibernate ist unabhängig, von der Art der Anwendung (Client/Server, Stand – Alone)



# Konfiguration mit Annotations...



```
@Entity
@org.hibernate.annotations.Enity{
   optimisticLock = org.hibernate.annotations.OptimisticLockType.ALL}
public class Item {
   @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
   protected long SKU;
   @Column(name="description")
   protected String description;
   @Column(name="name")
   protected String name;
   @Column(name="category")
   protected String category;
   public Item() {}
   // Getter und Setter
   ...
}
```

# und mit XML – Mappingdateien (1)



```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD
    3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
< hibernate-mapping package =" de.tfh.hibernateexample. db.entities ">
    <class name =" Item" >
        <id name =“SKU" type =" long">
            <column name =“SKU" />
            <generator class =" identity " />
        </id >
        <property name =“description " type =" string “>
            <column name =" description " length =“100" />
        </ property >
        ...
```

# XML – Mappingdateien (2)



```
        <property name =“ name" type =" string “>
                <column name =“name " length ="30" />
        </ property >
        <property name =“category" type =" string„>
                <column name =“category " />
        </ property >
        <version name=“version“ access=“field“ column=“version“/>
    </ class >
</ hibernate-mapping >
```

# hibernate.cfg.xml (1)



```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>

    </session-factory>
</hibernate-configuration>
```

# hibernate.cfg.xml (2)



```
<property name="connection.driver_class“>org.hsqldb.jdbcDriver</property>
<property name="connection.url“>jdbc:hsqldb:hsql://localhost</property>
<property name="connection.username“>sa</property>
<property name="connection.password“></property>
<property name="dialect“>org.hibernate.dialect.HSQLDialect</property>

<property name="current_session_context_class">thread</property>
<property name="transaction.factory_class">
    org.hibernate.transaction.JDBCTransactionFactory
</property>
```

# hibernate.cfg.xml (3)



```
<property name="hibernate.c3p0.min_size">5</property>
<property name="hibernate.c3p0.max_size">20</property>
<property name="hibernate.c3p0.timeout">300</property>
<property name="hibernate.c3p0.max_statements">50</property>
<property name="hibernate.c3p0.idle_test_period">3000</property>

<property name="show_sql">true</property>
<property name="hibernate.format_sql">true</property>

<property name="hbm2ddl.auto">create</property>
```

# hibernate.cfg.xml (4)



```
<mapping class="de.tfh.hibernateexample.db.entities.Inventory"/>
<mapping class="de.tfh.hibernateexample. db.entities.Item"/>
<mapping class="de.tfh.hibernateexample. db.entities.Order"/>
```

# JPA nach Hibernate



- aus EntityManagerFactory wird SessionFactory
- aus EntityManager wird Session
- aus EntityTransaction wird Transaction

# Ablauf einer Transaktion



```
try {
   AnnotationConfiguration configuration =
        new AnnotationConfiguration();
   SessionFactory sessionFactory =
        configuration.configure().buildSessionFactory();
   Session session = sessionFactory.openSession();
   Transaction transaction = session.beginTransaction();
   ...
   Transaction.commit();
   session.close();
} catch (Exception ex) {}
```

# Abfragen (1) – Natives SQL



- Unterstützung der SQL – Syntax
- Funktioniert ähnlich wie Abfragen mit JDBC
- Hibernate unterstützt Arbeit mit Resultsets

Beispiel:

```
List l = session.createSQLQuery(
   “select * from ITEM where ITEM.NAME = ‘HardDrive120‘ )
   .list();
```

# Abfragen (2) - HQL



- Hibernate Query Language ist ein objektorientiertes SQL
- Alle Abfragen sind automatisch polymorph
- Beinhaltet fast alle Möglichkeiten, die auch SQL bietet

Beispiel:

```
List l = session.createQuery(
   ”from Item i where i.name= ‘HardDrive120’ “).list();
```

# Abfragen (3) – Criteria API



- API zur Programmatischen Erzeugung von Abfragen
- Deutlich komfortabler als Manipulation von Stringqueries
- Funktionsumfang entspricht HQL

Beispiel:

```
Criteria crit = session.createCriteria(Item.class );
Criterion restriction = Restrictions .eq("name","HardDrive120"));
crit.add( restriction );
List l = crit.list();
```

# Abfragen (4) – Example API



- Eine weitere Programmierschnittstelle um dynamische Abfragen zu generieren
- Verwendung gleicht der Criteria API
- Prinzip: man definiert über das Criteria-Interface ein Objekt, das

  bestimmte Eigenschaften hat und sucht dann nach allen Objekten, die in diesen Eigenschaften übereinstimmen

Beispiel:

```
Item beispiel = new Item();
beispiel. setName(“HardDrive120“);
Criteria crit = session.createCriteria(Item.class);
crit.add(Example.create(beispiel)
     .enableLike (MatchMode.EXACT ));
List l = crit.list ();
```

# Das ist doch wie JPA...



JPA ist für die meisten Mappingaufgaben gut gerüstet, allerdings ist zu beachten:

Je spezieller die Mappings werden, desto deutlicher erkennt man die Grenzen von JPA.

# Aber dann doch mehr!



- Verwendung  Formel-  und datenbankgenerierter Werte möglich
- Mapping von Arrays, sowie aller Collectiontypen (einschließlich Collections von Werttypen und Komponenten) möglich
- Automatische Löschung verwaister Entity – Instanzen
- Automatische Generierung von gemeinsamen OneToOne - Primärschlüsseln
- Transaction – API kann für JDBC und JTA konfiguriert werden (JPA Entsprechung: EntityTransaction – API ist nur für resource – local Transaktionen nützlich)
- Reattach von Objekten möglich
- Datenfilter für dynamische Viewerstellung vorhanden
- Feingranulierte Fetching – Strategie möglich
- Existenz einer Collections – Filter – API

# Zusätzliche Ressourcen



- Hibernate Tools for Eclipse and Ant (http://www.hibernate.org/255.html)
  - Mapping Editor
  - Console
  - Reverse Engineering
  - Wizards
  - Anttask
- Using Hibernate with the Java Persistence API (Tutorial mit Netbeans) (http://www.netbeans.org/kb/articles/hibernate-javaee.html)
- Tutorial auf hibernate.org (http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html)

# Quellen



- Java Persistence mit Hibernate
- http://hibernate.org
- http://www.netbeans.org/kb/articles/hibernate-javaee.html
- http://www.hibernate.org/hib_docs/v3/reference/en/html/tutorial.html

Vielen Dank für die Aufmerksamkeit.