Introduction to Hibernate 2.0

Steve Peterson
Acorn Bay Software, Inc.
http://www.acornbay.com

Presentation Overview

- Hibernate overview
- Building an application
- Other useful stuff
- Hibernate vs. JDO
- Wrap up

About Steve Peterson

- System architect, designer, and project manager
- 23 years experience in software development
- Client list includes:
  - Macromedia
  - 3M
  - NASA
  - Solectron

Facts about Hibernate

- Java object-relational mapping tool
- One step up from direct JDBC access
- Not based on a standard
- Only OSS Data Access tool selected as JavaWorld Editor's Choice for 2003
- Popular

Technologies in the "objects to database" space

- Standardized API
  - JDO implementations
  - J2EE CMP
  - OJB
- Proprietary API
  - TopLink
  - CocoBase O/R
  - Hibernate

RDBMS support

Tested with:

- DB2 7.1, 7.2;
- MySQL 3.23;
- PostgreSQL 7.1.2, 7.2, 7.3;
- Oracle 8i, 9i;
- Sybase 12.5 (JConnect 5.5);
- Interbase 6.0.1 (Open Source) with Firebird InterClient 2.01;
- HypersonicSQL 1.61, 1.7.0;
- Microsoft SQL Server 2000;
- Mckoi SQL 0.93
- Progress 9
- Pointbase Embedded 4.3
- SAP DB 7.3

Thought to be compatible with the latest versions of:

- Informix
- Ingres
- FrontBase

Tool support

- XDoclet
- Modeling tools/code generators
  - Middlegen
  - AndroMDA
- IDE plug-in support: Eclipse, IDEA

Creating your persistent class


```
public class Customer {
        private Account    _account = new Account(  );
        private CreditCard _creditCard = new CreditCard(  );
        private String     _email = "";
        // ...

        public Customer() { }

        public void setAccount(Account acct) { _account = acct; }
        public Account getAccount() { return _account; }

        public void setCreditCard(CreditCard cc) { _creditCard = cc; }
        public CreditCard getCreditCard() { return _creditCard; }

        public void setEmail(String email) { _email = email; }
        public String getEmail() { return _email; }

        // and so on

}
```

Rules for your class

- Must have Bean-style `getXXX()` and `setXXX()`
- Must have default constructor
- Not `final`
- Identifier recommended (not required)
  - any primitive type
  - any primitive wrapper
  - String
  - Date
  - user-defined class (good for existing tables with composite keys)

Creating the mapping file

- Defining a class
  - <class>
  - <id>
  - <generator>
  - <property>
- Defining relationships
- Tools support

<class>


```
<class name="name.of.class"                  can be an interface
        table="name_of_table"                default: class name
        schema="database_schema"             default: none
        proxy="name.of.proxy.class"          default: no proxy i/f
        discrimator-value="val"              default: class name
        mutable="true"
        polymorphism="implicit"
        dynamic-update="false"
        dynamic-insert="false"
        persister="PersisterClass">

        ...

</class>
```


<id>


```
<class name="xpetstore.domain.Customer" table="T_CUSTOMER"/>
        <id name="userId" length="10">
                <generator class="assigned" />
        </id>
</class>
```

Additional <id> attributes:

```
type="typename"
column="column_name"
unsaved-value="something"          options: any, none, null, value
```

<generator> options

Class	Description	Types
vm	Intra-VM identifier	long, short, int
identity	DB-supported identity column	long, short, int
sequence	DB-supported sequence	long, short, int
hilo	Database-unique values	long, short, int
seqhilo	hi/lo using sequences	long, short, int
uuid.hex	128 bit UUID algorithm incorporating IP address	32 digit hex string
uuid.string	same	16 character ASCII string
native	Varies	best of identity, sequence, or hilo, based on database capabilities
assigned	Application assigned	Any supported type
foreign	Uses identifier of associated object, for one-to-one associations	Any supported type

<property>


```
<class name="xpatstore.domain.Customer" table="T_CUSTOMER">
        <id name="userId" length="10">
                <generator class="assigned" />
        </id>
        <property name="email" length="100" not-null="false"
                unique="false"/>
        ...
</class>
```

Additional <property> attributes:

```
type="typename"
column="column_name"
update="true"
insert="true"
```


<property> types

- Java primitives & wrappers
- string
- date, time, timestamp
- calendar, calendar_date
- big_decimal
- locale, timezone, currency (maps to VARCHAR)
- class (maps to name)
- binary (byte array)
- serializable
- clob, blob

Creating the mapping file

- Defining a class ☑
- Defining relationships
  - <one-to-one>
  - <one-to-many>
  - <many-to-many>
  - <collection>
  - <map>
- Tools support

One-to-one mapping

Java usage:

```
Bar myBar = Foo.getBar();
```

Mapping file:

```
<class name="Foo">
        ...
        <one-to-one name="bar" class="Bar" />
</class>
```

Schema:

Foo
id

Bar
id

One-to-many mapping

Java usage:

```
Set myBars = Foo.getBars();
```

Mapping file:

```
<class name="Foo">
        ...
        <set role="bars" table="bar" inverse="true" lazy="true">
                <key column-"foo_id" />
                <one-to-many class="Bar">
        </set>
</class>
```

Schema:

Foo
id

Bar	
Id	foo_id

Many-to-many mapping

Java usage:

```
Set myBars = Foo.getBars();
```

Mapping file:

```
<class name="Foo">
        ...
        <set role="bars" table="Foo_Bar">
                <key column="foo_id" />
                <many-to-many column="bar_id" class="Bar">
        </set>
</class>
```

Schema:

Bar
id

Foo
id

Foo_Bar	
foo_id	bar_id

Collection

Java usage:

```
Set myPeople = Foo.getPeople();
```

Mapping file:

```
<class name="Foo">
        ...
        <set role="people" table="Person">
                <key column="foo_id" />
                <element column="name" type="string">
        </set>
</class>
```

Schema:

Foo
id

Person	
foo_id	name

Map

Java usage:

```
Map myAges = Foo.getAges();
```

Mapping file:

```
<class name="Foo">
        ...
        <map role="ages" table="Ages">
                <key column="id" />
                <index column="name" type="string" />
                <element column="age" type="string">
        </map>
</class>
```

Schema:

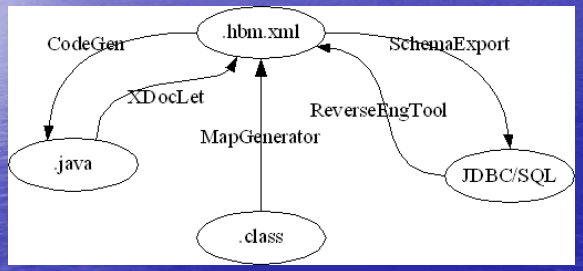
Foo
id

Ages		
foo_id	name	age

Creating the mapping file

- Defining a class ☑
- Defining relationships ☑
- Tools support

Mapping file tools support



hibernate.properties file

```
# example hibernate.properties

# put me in the root of the classpath
# or do Configuration.setProperties(startupProperties)
# or set System properties using -Dproperty=value

# Example configuration for Oracle

hibernate.dialect net.sf.hibernate.dialect.OracleDialect
hibernate.connection.driver_class oracle.jdbc.driver.OracleDriver
hibernate.connection.username SCOTT
hibernate.connection.password TIGER
hibernate.connection.url jdbc:oracle:thin:@localhost:1521:test

hibernate.show_sql false
```

Manipulating objects

- Initialization
- Creating/loading objects
- Updating/deleting objects
- Updating stale objects
- Filtering results
- Other query features

Initialization


```
// Loading the configuration
// Looks for mapping files in same classpath location as class

Configuration cfg = new Configuration()
        .addClass(eg.Vertex.class)
        .addClass(eg.Edge.class);

// This binds you to a database configuration
SessionFactory sessions = cfg.buildSessionFactory();

// obtain a JDBC connection and instantiate a new Session

Session sess = sessions.openSession();

// start a new transaction (optional)
Transaction tx = sess.beginTransaction();
```


Creating and loading objects


```
// Creating an object

Customer customer = new Customer();
customer.setAccount(new Account());
customer.setEmail("speterson@acornbay.com");
// ...
Long id = (Long) sess.save(customer);

// Later
// Loading an object

// If we know the ID
Customer customer = (Customer) sess.load(Customer.class, id);

// Querying
List customers = sess.find(
        "from Customer as customer where customer.email = ?",
        email,
        Hibernate.STRING);
```


Updating and deleting objects


```
// Updating object saved/loaded in current session

Customer customer = (Customer) sess.load(Customer.class, id);
customer.setEmail("abc@123.com");
sess.flush();

// Deleting an object

sess.delete(customer);

// Deleting many objects

sess.delete("from customer as Customer where amount_purchased < ?",
        new Double(10.0),
        Hibernate.DOUBLE);
```

Updating stale objects


```
// Updating objects from a previous session

// First session
Customer customer = (Customer) s1.load(Customer.class, id);
Account newAccount = new Account();
s1.save(newAccount);

// Somewhere else in the application after s1 is defunct
customer.setAccount(newAccount);

// Second session
s2.update(customer);
s2.update(newAccount);
```


Filtering results


```
// Filtering query results

List customers = sess.find(
        "from Customer as customer where account_balance > ?",
        new Double(10.0),
        Hibernate.DOUBLE);

// Give me them in customer.name order
List sortedUsers = sess.filter(customers, "order by this.name");

// Give me only those who owe more than 100.00
List lotsDueFromThem = sess.filter(customers,
        "where this.account_balance > 100");

// Iteration over query results

Iterator it = sess.iterate("from Customer customer order by "
        + " customer.account_balance");
```


Queries

- Table joins expressed as property paths
- Support for SQL aggregate functions sum, avg, min, max, count
- Support for left|right outer join, full join
- Support for group by, having and order by
- Support for subqueries (on databases with subselects)
- Queries may return tuples of objects or scalar values
- Batch delete

Other useful stuff

- Extension points
- Interfaces
- Performance
- Hibernate vs JDO

Extension points

- Transaction definition
- RDBMS support
- Database connection management
- Primary key generation

Other useful interfaces

- Persistent object lifecycle callbacks
- Validatable
- Interceptor
- Metadata API

Managing performance

- < 10% slower than straight JDBC in nearly all cases
- Options for managing performance
  - Lazy initialization (proxies)
  - Caching

Hibernate vs. JDO

Hibernate

- Single "vendor"
- More expressive QL
- Open source
- Uses standard build process

JDO

- JCP Standard
- Commercial (but viable OSS approaching)
- Modified class files (extra compile step)
- Support for nonrelational data stores
- Mapping not standard

HQL vs JDOQL

JDO:

```
Query query = pm.newQuery(pm.getExtent(GameObject.class, false));
Query.setFilter("age > ageParam");
Query.declareParameters("int ageParam");
Collection oldGameObjects = (Collection)
    q.execute(newInteger(25));
```

Hibernate:

```
List oldObjects =
    sess.find("from GameObjects go where go.age > ?",
        age, Hibernate.INTEGER);
// Note:  Hibernate supports JDO-style queries too
```


Wrap up

- Demo applications
  - xPetstore - http://xpetstore.sourceforge.net
  - Struts-resume - http://static.raibledesigns.com/downloads
  - Struts-hibernate - http://struts.sourceforge.net
- Where to find out more
  - Type "hibernate" into Google

Presentation Sources

- Hibernate Reference Manual
  - http://hibernate.bluemars.net/5.html
- Tom Sedge – Hibernate Association Styles
  - http://www.xylax.net/hibernate