Hibernate

Advanced Concepts

Agenda

1. Advanced Features

2. Best Practices

3. Design Patterns

Advanced Features

Advanced Features

- Java maps can be used instead of entity classes to create a representation of a relational table in Hibernate

- In XML configuration file the `entity-name` property should be specified additionally

```
<hibernate-mapping>
      <class entity-name="employee">
      …
      </class>
</hibernate-mapping>
```

Advanced Features

- The main benefit in using maps instead of POJOs is for creating quick prototypes
- The main drawback is that compile-type type checking is lost

Advanced Features

- Tupilizers can be used to customize mapping of particular fields (components) or entire entities in Hibernate

- Tupilizers are specified for the particular entity/field in the Hibernate mapping XML/annotations configuration

Advanced Features

- Entity name resolves can be used to specify a strategy for resolving the entity name for an entity class

- Entity name resolvers can be registered either by:

  - Creating a tupilizer and returning them as part of the `getEntityNameResolvers` method

  - Passing them to the `registerEntityNameResolver` method of the Hibernate session factory

Advanced Features

- Hibernate does not differentiate between the usage of tables and views

- Some RDBMS does not support creation of views - in this case you can use Hibernate to link an entity to a query thus simulating a view creation in Java

Advanced Features

- Example (using the `@Subselect` annotation):

```
@Entity
@Subselect("select Name, Phone from Employees"
@Synchronize( {"Employees"} ) //tables impacted
public class Employee {
      @Id public String getId()
      {
            return id;
      }
... }
```

- Example (using XML configuration):

```
<class name="Employee">
      <subselect>select Name, Phone from Employees
      </subselect>
      <synchronize table="Employees"/>
      <id name="id"/>
      ...
</class>
```

Advanced Features

- You can also simulate a virtual column by specifying a formula/SQL fragment that assigns a value to the field (Hibernate uses it instead of mapping to a column in the entity's table)
- Example:

```
@Entity
@Table(name="EMPLOYEES")
public class Employee {
      @Formula("upper(Name)")
      public String getUpperName()
      {
            return upperName;
      }
... }
```

Advanced Features

- You can implement application-level triggers using interceptors (instances of `org.hibernate.EmptyInterceptor`)

- Interceptors can be used to inspect and manipulate properties of persistent entities once they are loaded, saved, updated or deleted

Advanced Features

- Hibernate supports batch processing of queries by using the JDBC batch mechanisms

- Batch size is set using the `hibernate.jdbc.batch_size` property

Advanced Features

- You can mark a column with the `@Version` annotation on a version number or date/timestamp column in order to enable optimistic locking
- Optimistic locking in that manner allows Hibernate to determine if an entity has been changed by another transaction in order to provide transaction consistency
- Version columns are typically generated automatically by Hibernate or the underlying database

Advanced Features

- A property can also be a collection of simple types instead of a collection of entities

- Example:

```
@Entity
public class Employee
{
      @ElementCollection
      @CollectionTable(name="Nicknames", joinColumns
=@JoinColumn(name="user_id"))
      @Column(name="nickname")
      public Set<String> getNicknames() { ... }
}
```

Advanced Features

- A property can also be a collection of embeddable types (components) instead of a collection of entities

- Example:

```
@Entity
public class Employee {
      @ElementCollection
      @CollectionTable(name="Addresses", joinColumns
     =@JoinColumn(name="user_id"))
      @AttributeOverrides({@AttributeOverride(name="
     city", column=
      @Column(name="col_city"))   })
      public Set<Address> getAddresses() { ... }
}
@Embeddable
public class Address {
      public String getCity() {...}  }
```

Advanced Features

- There are several strategies for persisting Java class hierarchies of entities:
  - Single table per class strategy - a single table hosts all the instances of a class hierarchy
  - Joined subclass strategy - one table per class and per subclass is present and each table persists the properties specific to a given subclass
  - Table per class strategy - one table per concrete class and subclass is present and each table persist the properties of the class and its superclasses - the state of the entity is then stored entirely in the dedicated table for its class

Advanced Features

- Hibernate uses a fetching strategy to retrieve objects if the application needs to navigate the association

- Fetch strategies can be declared in the O/R mapping metadata, or over-ridden by a particular HQL or Criteria query

Advanced Features

- Common fetching strategies:
  - JOIN fetching - associated instance/collection is retrieved as part of the `SELECT` that retrieves the parent instance
  - SELECT fetching - a separate `SELECT` is used to retrieve child entities - executed only when children are requested (unless `lazy = "false"` is set)

Best Practices

Best practices

- Define ID fields for entities
- If using XML mapping files define each entity class in a separate XML file
- Use named/positional bind variables in queries

Design Patterns

Design Patterns

- Entity classes may also be called data transfer objects (DTO) objects - use a consistent naming schema for Hibernate entities

- Defining data access classes for the mapped entities with following functionality:
  - Finding entities by primary key
  - Creating and updating entities
  - Deleting existing entities
  - Finding entities by criteria with support for paging and sorting

Design Patterns

- Data access classes are also called data access object (DAO) classes
- An abstract DAO class or interface can be defined to provide the common functionality of all DAOs (e.g. abstract method that must be implemented by all DAOs such as `find(Long id)` or `listAll()`)

Questions?