JPA & Hibernate

Gustavo Ansaldi Oliva
{golivax@gmail.com, GOLIVA@ime.usp.br}

10 de agosto de 2011

Instituto de

Agenda

- Introdução
- Java Persistence API (JPA)
- Annotations
- Entity Manager
- JPQL
  - Comandos

Introdução

- Grande maioria das aplicações utiliza bancos de dados relacionais para armazenar seus dados
- Dados são modelados de maneira diferente em sistemas orientados a objetos e em bancos de dados relacionais
- JDBC (Java Database Connectivity)

Object-Relational Impedance Mismatch

- Granularidade
  - O modelo de classes é mais granular que o modelo relacional
- Subtipos (herança)
  - O conceito de herança
- Identidade
  - Em modelos relacionais, a noção de

JPA

- Especificação de um framework para mapeamento objeto-relacional
- Solução mais simples e eficiente que as existentes até o momento
- Divulgado no Java EE 5 como parte do EJB 3

JPA

- Frameworks que motivaram a criação da JPA (Hibernate, Toplink, etc), adaptaram-se para implementar esta especificação

Anotações básicas

```
@Entity
public class Funcionario implements
Serializable{
   @Id
   private int id;
   private String nome;
```

Anotações básicas

```
@Entity
 public class Funcionario implements Serializable{
  private int id;
  private String nome;
  private String nascimento;
  @Id
  public int getId(){
    return id;
  }
  @Temporal (TemporalType.DATE)
  public Date getNascimento(){
```

Anotações básicas - @Id

```
@Id

@Generated Value

private int id;
```

@Id - Define qual campo que é a chave primária.

@GeneratedValue - Define como que a JPA

Anotações básicas - @Id

```
@Id

@Generated Value
(strategy=GenerationType.AUTO)

private int id;
```

strategy=GenerationType.AUTO - O JPA tenta descobrir qual é a melhor estratégia.

Anotações básicas - @Id

```
@Id

@Generated Value
(strategy=GenerationType.IDENTITY)

private int id ;
```

GenerationType.IDENTITY - O banco de dados usa uma coluna de auto-

Anotações básicas - @Id

```
@Id

@Sequence Generator
(name="Funcionario Seq"
,sequenceName=" FuncGen")

@GeneratedValue(strategy=GenerationT
ype.SEQUENCE,generator="FuncionarioS
eq")
```

Anotações básicas - @Id

```
@Table Generator(name="Funcionario
Seq", table="Contadores",
pkColumnName="Nome",
valueColumnName="Proximo",
pkColumnValue="Funcionario")

@Id

@Generated Value
(strategy=GenerationType.TABLE
```

Anotações básicas - @Temporal

- Para tipos de data, hora e data/hora:
  - java.sql.Date mapeia para "data".
  - java.sql.Timestamp mapeia para "data/hora".
  - java.sqlTime mapeia para "hora".
  - java.util.Date e java.util.Calendar mapeiam de acordo com a annotation @Temporal.

Anotações básicas - @Enumerated

- Campos ou getters de tipos enum devem ter a anotação @Enumerated
- @Enumerated tem dois valores: ORDINAL, que é o padrão e diz que o método ordinal() do enum será usado para persisti-lo, ou STRING que diz que o método name() será usado para persisti-lo.

Anotações básicas - @Lob, @Basic

```
@Lob

@Basic (optional=true, fetch=FetchType.LAZY)

private char[ ] fotografia;
```

- FetchType.EAGER - Indica que o campo sempre é carregado para a memória

Anotações básicas - @Entity, @Table

```
@Entity (name="truck")

@Table (name="tbcaminhoes")

public class Caminhao implements
Serializable {
```

Anotações básicas - @Column

```
@Entity

public class Carro implements
Serializable{

   @Id

   @Column (name="carro placa")

   private String placa;
```

Mapeando relacionamentos - @OneToOne

```
@Entity
public class Customer implements Serializable {
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name="passport_fk")
    public Passport getPassport() {
        ...
    }

@Entity
public class Passport implements Serializable {
    @OneToOne(mappedBy = "passport")
    public Customer getOwner() {
    ...
}
```

Mapeando relacionamentos - @ManyToOne

```
@Entity()
public class Flight implements Serializable {
    @ManyToOne( cascade = {CascadeType.PERSIST, CascadeType.MERGE} )
    @JoinColumn(name="COMP_ID")
    public Company getCompany() {
        return company;
    }
    ...
}
```

Mapeando relacionamentos - @OneToMany

- Unidirecional

```
@Entity
public class Trainer {
    @OneToMany
    @JoinTable(
            name="TrainedMonkeys",
            joinColumns = @JoinColumn( name="trainer_id"),
            inverseJoinColumns = @JoinColumn( name="monkey_id")
    )
    public Set<Monkey> getTrainedMonkeys() {
    ...
}

@Entity
public class Monkey {
    ... //no bidir
}
```

Mapeando relacionamentos - @OneToMany

- Bidirecional

```
@Entity
public class Troop {
    @OneToMany(mappedBy="troop")
    public Set<Soldier> getSoldiers() {
    ...
}

@Entity
public class Soldier {
    @ManyToOne
    @JoinColumn(name="troop_fk")
    public Troop getTroop() {
    ...
}
```

Mapeando relacionamentos - @ManyToMany

```
@Entity
public class Employer implements Serializable {
    @ManyToMany(
        targetEntity=org.hibernate.test.metadata.manytomany.Employee.class,
        cascade={CascadeType.PERSIST, CascadeType.MERGE}
    )
    @JoinTable(
        name="EMPLOYER_EMPLOYEE",
        joinColumns=@JoinColumn(name="EMPER_ID"),
        inverseJoinColumns=@JoinColumn(name="EMPEE_ID")
    )
    public Collection getEmployees() {
        return employees;
    }
    ...
}
```

Mapeando relacionamentos - @ManyToMany

```
@Entity
public class Employee implements Serializable {
    @ManyToMany(
        cascade = {CascadeType.PERSIST, CascadeType.MERGE},
        mappedBy = "employees",
        targetEntity = Employer.class
    )
    public Collection getEmployers() {
        return employers;
    }
}
```

Mapeando relacionamentos - Herança

```
@MappedSuperclass
public class BaseEntity {
    @Basic
    @Temporal(TemporalType.TIMESTAMP)
    public Date getLastUpdate() { ... }
    public String getLastUpdater() { ... }
    ...
}

@Entity class Order extends BaseEntity {
    @Id public Integer getId() { ... }
    ...
}
```

- Ver outros tipos em

Entity Manager

Funcionalmente, divide-se em:

- Transaction Association
- Entity Lifecycle Management
- Entity Identity Management
- Cache Management
- Query Factory

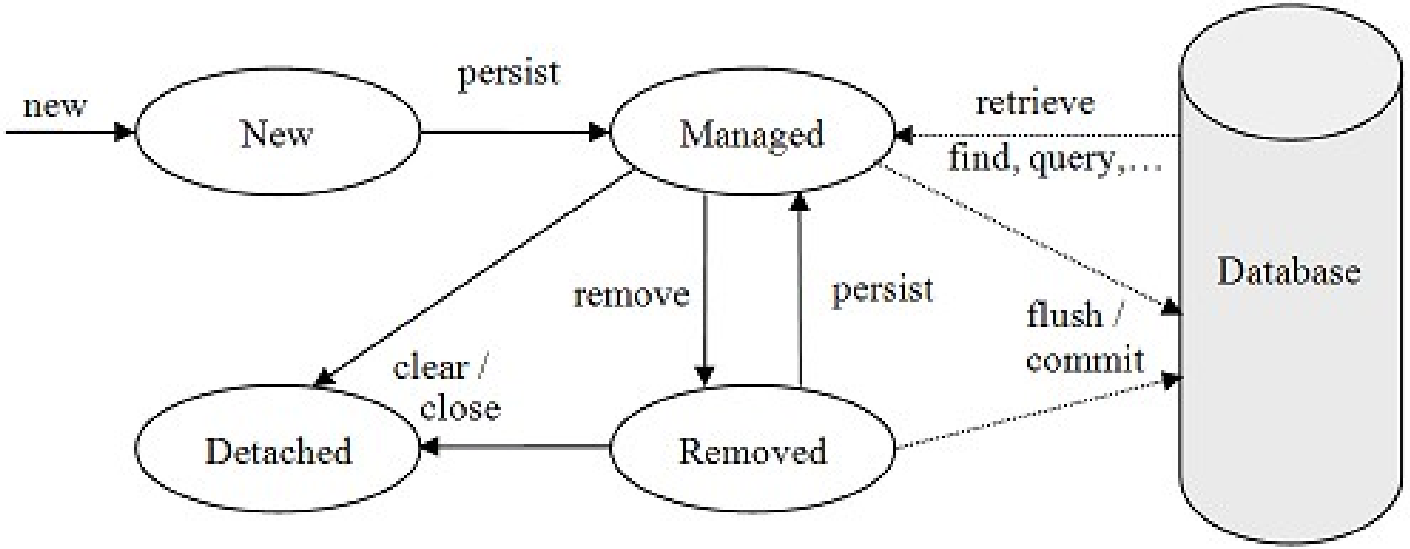
Transaction Association

- `public EntityTransaction getTransaction();`
  - Utilizado para iniciar e fechar uma transação

Entity Lifecycle Management

- `public void persist(Object entity);`
- `public void remove(Object entity);`
- `public Object merge(Object entity);`
- `public void refresh(Object entity);`

Entity Lifecycle Management



Entity Identity Management

- `public <T> T find(Class<T> cls, Object oid);`
  - Retorna o objeto buscado
- `public <T> T getReference(Class<T> cls, Object oid);`
  - Retorna um proxy do objeto buscado
- `public boolean contains(Object`

Cache Management

- `public void flush();`
- `public FlushModeType getFlushMode();`
- `public void setFlushMode(FlushModeType flushMode);`
  - 嬪 COMMIT

Query Factory

- `public Query createQuery(String query);`
- `public Query createNamedQuery(String name);`
- `public Query createNativeQuery(String sql);`
- `public Query`

Closing

- `public boolean isOpen();`
- `public void close();`

JPQL

- Similar a SQL
- Usado para definir buscas de entidades
- Independente do mecanismo usado para persistência
- Estende EJB QL
- Declaradas estaticamente ou dinamicamente

Comandos

- SELECT campos FROM entidade [WHERE condição] [GROUP BY cláusula] [HAVING condição] [ORDER BY cláusula]
- UPDATE entidade SET campo atualizado=valor [WHERE condição]
- DELETE entidade [WHERE condição]

Exemplo

```
@Entity
@Table (name="CUSTOMER TABLE")
public class Customer implements Serializable{
  @Id
  @Column (name="ID")
  private Integer customer id ;
  @Column (name="CITY")
  private String city;
  @Column (name="NAME")
  private String name;
  @Enumerated (ORDINAL)
  @Column (name="STATUS")
```

Exemplo

```
@Entity
@Table(name="ORDER TABLE")
public class Order implements Serializable{
  @Id
  @Column(name="ID”)
  private Integer orderId
  @Column(name="QUANTITY")
  private int quantity;
  @Column (name="TOTALPRICE")
  private float totalPrice
  @ManyToOne
```

- Definindo a query de busca
  - ```
    String ejbql = "SELECT c FROM
      Customer c WHERE c.name = 'Joao' ";
    ```
- Executando
  - ```
    Query query = em.createQuery(ejbql);
    ```
  - ```
    List result = query.getResultList();
    ```
    ```

- Definindo a Named Query

```
@NamedQuery
(name="findCustomerByName",

query="SELECT c FROM Customer c WHERE
c.name = :name")})
```

- Executando

```
List customers =
```

- SELECT simples

```
"SELECT c FROM Customer c WHERE c.name
= 'Joao'
```

- SELECT com JOIN implícito

```
"SELECT o FROM Order o WHERE
o.costumer.name = 'Joao'
```

- SELECT com JOIN explícito

Hibernate

- Um dos principais frameworks que inspiraram a JPA
- Criado a partir de iniciativa open-source
- Hoje é mantido pela JBoss

Utilizando a API do Hibernate

- É possível utilizar a API exclusiva do Hibernate mesmo quando o utilizamos como *provider* para o JPA
- O método `getDelegate()` retorna uma instância que implementa a interface `org.hibernate.Session`

Hibernate Query API

- Criteria Query API
  - Maneira mais simples de obter dados
  - Queries são feitas programaticamente através de uma API Java
- Hibernate Query Language
  - Sintaxe e funcionalidades bastante parecidas com JPQL

Como utilizar a Criteria API

- Crie um objeto `org.hibernate.Criteria` através do *factorymethod* `createCriteria(...)` da Session
  - Passe a classe do objeto persistente ou o nome da entidade para o método createCriteria(...)
  - Chame o metodo list() do objeto Criteria

Paginação

- Hibernate cuida da paginação
  - Retornar um numero fixo de objetos
- Dois métodos da classe Criteria
  - setFirstResult() - seta a primeira linha do resultado
  - setMaxResult() - seta a quantidade de linhas a retornar

Restrictions

- Utilizada para obter objetos que satisfaçam determinadas condições
  - Exemplo: Objetos do tipo Pessoa cuja idade seja maior do que 20 anos
- Adicione restrições para o objeto Criteria com o metodo add()
  - O método `add()` recebe um objeto `org.hibernate.criterion.Criterion` que

Métodos da classe Restriction

- `Restrictions.eq("name", "Shin")`
- `Restrictions.ne("name", "NoName")`
- `Restrictions.like("name", "Sa%")`
- `Restrictions.ilike("name", "sa%")`
- `Restrictions.isNull("name");`
- `Restrictions.gt("price",new`

Exemplo de Restriction

```
// Resgata objetos do tipo Pessoa
// cujo nome segue um padrao e cuja
// idade é 10 ou null (em branco)

List people =
sess.createCriteria(Person.class)
```

Ordenando resultados

- É possível ordenar resultados utilizando `org.hibernate.criterion.Order`

```
List cats =
sess.createCriteria(Cat.class)
.add( Restrictions.like("name", "F
%")
```

Associações

- É possível especificar constraints em entidades relacionadas utilizando createCriteria(...)

```
List cats =
sess.createCriteria(Cat.class)
.add(Restrictions.like("name", "F%")
```

Projeções e Aggregate Functions

- A classe `org.hibernate.criterion.Projections` é um *factory* para instâncias de Projection
- Aplique uma projeção em uma query através do método setProjection() da classe Criteria

Projeções e Aggregate Functions

- rowCount()
- avg(String propertyName)
- count(String propertyName)
- countDistinct(String propertyName)
- max(String propertyName)
- min(String propertyName)

Exemplo

```
// Retorna uma lista com um array de
Object

// como primeiro elemento. O array
de Object contem

// todos os valores em ordem

Criteria crit =
sess.createCriteria(Product.class);
```

Query by Example

- Prove um outro "estilo“ de busca
- Como realizar uma QBE
  - Parcialmente popule uma instancia de um objeto
  - Deixe o Hibernate construir uma Criteria utilizando a instancia como exemplo
- A classe

Exemplo

```
// Recupera objetos do tipo Pessoa
// através de um exemplo
Criteria crit = sess.createCriteria(Person.class);
Person person = new Person();
person.setName("Shin");
Example exampleRestriction = Example.create(person);
crit.add(exampleRestriction);
List results = crit.list();
```

Hibernate Criteria API

- Facil de usar, poderosa e elegante
- Especialmente útil quando e necessário utilizar Dynamic Query Generation
- Por outro lado, queries estaticas externas possuem algumas
- vantagens
  - Queries externalizadas podem ser