

Java Web com Servlets e JSPs


Introdução ao Hibernate




cursos online




1

Tópicos Abordados


- Desvantagens do JDBC
- Frameworks ORM
- O que é o Hibernate
- Entidades
- Configuração do Hibernate
- SessionFactory e Session
- Associações
- HQL



2

Desvantagens do JDBC

- JDBC permite a integração entre aplicações Java e banco de dados
 - A forma de uso é a mesma para todos os bancos de dados
- JDBC expõe a linguagem SQL ao programador
 - SQL nem sempre é padronizado
 - Às vezes, as queries são complexas de montar



3

Frameworks ORM

- Em aplicações orientadas a objetos, normalmente são criadas classes para representar conceitos existentes na aplicação
- Estas classes são, no final das contas, mapeadas para tabelas do banco de dados, com o objetivo de persistir os dados
- Os frameworks do tipo **Object-Relational Mapping** facilitam este mapeamento

4

ORM

Aplicação

Produto

id
nome
valor

→

Banco de Dados

Produto		
ID	NOME	VALOR

Aplicação

Produto

id = 10
nome = XPTO
valor = 100.00

→

Banco de Dados

Produto		
ID	NOME	VALOR
10	XPTO	100.00



5

Hibernate

- O Hibernate é um framework ORM
- Gratuito e open source
- Site oficial
 - <http://www.hibernate.org>
- Pode ser utilizado em aplicações standalone e aplicações web

6

Entidades



- Entidades são classes Java que serão mapeadas para tabelas no banco de dados
- As entidades são classes POJO
 - Devem ter uma ou mais propriedades que serão utilizadas como chave na tabela
 - Atributos e métodos getters e setters
 - O Hibernate consegue mapear atributos sem getters e/ou setters definidos, mas utilizá-los é mais adequado
 - Presença de um construtor sem argumentos



7

Entidades



```
public class Produto {  
    private Long id;  
    private String nome;  
    private Double valor;  
  
    private void setId(Long id) {  
        this.id = id;  
    }  
  
    public Long getId() {  
        return this.id;  
    }  
  
    //Outros getters e setters  
}
```

A classe não precisa estender nenhuma classe ou implementar interfaces

Os atributos serão mapeados mais tarde para tabelas

O método `setId()` pode ser privado, pois o Hibernate gerará o ID



8

Mapeamento dos Atributos



```
Produto.hbm.xml  
  
<hibernate-mapping>  
  <class name="Produto">  
    <id name="id">  
      <generator class="native" />  
    </id>  
    <property name="nome" />  
    <property name="valor" />  
  </class>  
</hibernate-mapping>
```

Mapeamento do ID da entidade

Mapeamento das propriedades



9

Configurando o Hibernate

- Para que o Hibernate funcione adequadamente, ele deve ser configurado
 - Os JARs necessários devem ser colocados no classpath
 - Criação dos arquivos **.hbm.xml**
 - Criação do arquivo **hibernate.cfg.xml**
 - Define como o Hibernate será integrado com o banco de dados da aplicação

10

O Arquivo hibernate.cfg.xml

```
hibernate.cfg.xml<hibernate-configuration>
  <session-factory>
    <property name="connection.driver_class">
      com.mysql.jdbc.Driver
    </property>

    <property name="connection.url">
      jdbc:mysql://localhost/db
    </property>

    <property name="connection.username">root</property>
    <property name="connection.password">admin</property>

    <property name="dialect">
      org.hibernate.dialect.MySQLDialect
    </property>

    <mapping resource="db/Produto.hbm.xml" />
  </session-factory>
</hibernate-configuration>
```

11


SessionFactory

- A **SessionFactory** é responsável por criar os objetos Session

```
Configuration cfg = new Configuration().configure();
ServiceRegistry sr = new ServiceRegistryBuilder().
    applySettings(cfg.getProperties()).buildServiceRegistry();
SessionFactory factory = cfg.buildSessionFactory(sr);
```

- O ideal é usar apenas um objeto SessionFactory para toda a sua aplicação

12

Session


- A **Session** é utilizada para executar operações, que serão refletidas no banco de dados


```

Session session = factory.getCurrentSession();
session.beginTransaction();


Produto p = new Produto();
p.setNome("XPTO");
p.setValor(100.0);

session.save(p);
session.getTransaction().commit();
    
```


- A session pode ser utilizada para executar mais de uma operação




13

Operações da Session



Operação	Descrição
save()	Cria a entidade no banco de dados.
update()	Atualiza uma entidade existente no banco de dados.
delete()	Exclui uma entidade do banco de dados.
load()	Carrega uma entidade no banco de dados através do seu ID.
createQuery()	Cria um objeto Query a partir de uma expressão HQL, que possibilita executar queries no banco de dados.



14

Associações


- O Hibernate gerencia também as associações entre entidades, escondendo a linguagem SQL usada no processo
- Tipos de associação
 - Many-to-One (Muitos-para-Um)
 - One-to-Many (Um-para-Muitos)
 - One-to-One (Um-para-Um)
 - Many-to-Many (Muitos-para-Muitos)
- Associações podem ser unidirecionais ou bidirecionais



15

Associação Many-to-One

- Unidirecional

```

<class name="Cliente">
  ...
  <many-to-one name="endereco" column="endereco_id" />
</class>

<class name="Endereco">
  ...
</class>

```

16

Associação Many-to-One

- Bidirecional

```

<class name="Cliente">
  ...
  <many-to-one name="endereco" column="endereco_id" />
</class>

<class name="Endereco">
  ...
  <set name="clientes" inverse="true">
    <key column="endereco_id"/>
    <one-to-many class="Cliente"/>
  </set>
</class>

```

17

Associação One-to-Many

- Unidirecional

```

<class name="Endereco">
  ...
  <set name="clientes" inverse="true">
    <key column="endereco_id"/>
    <one-to-many class="Cliente"/>
  </set>
</class>

<class name="Cliente">
  ...
</class>

```

18

Associação One-to-Many

- Bidirecional

```

<class name="Endereco">
  ...
  <set name="clientes" inverse="true">
    <key column="endereco_id"/>
    <one-to-many class="Cliente"/>
  </set>
</class>

<class name="Cliente">
  ...
  <many-to-one name="endereco" column="endereco_id" />
</class>

```

19

Associação One-to-One

- Unidirecional

```

<class name="Cliente">
  <id name="id" column="id">
    <generator class="native"/>
  </id>
</class>

<class name="CartaoFidelidade">
  <id name="id" column="cliente_id">
    <generator class="foreign">
      <param name="property">cliente</param>
    </generator>
  </id>
  <one-to-one name="cliente" constrained="true" />
</class>

```

20

Associação One-to-One

- Bidirecional

```

<class name="Cliente">
  <id name="id" column="id">
    <generator class="native"/>
  </id>
  <one-to-one name="cartao" />
</class>

<class name="CartaoFidelidade">
  <id name="id" column="cliente_id">
    <generator class="foreign">
      <param name="property">cliente</param>
    </generator>
  </id>
  <one-to-one name="cliente" constrained="true" />
</class>

```

21

Associação Many-to-Many

• Unidirecional

```

<class name="Cliente">
...
<set name="produtos" table="produto_cliente">
  <key column="cliente_id" />
  <many-to-many column="produto_id" class="Produto" />
</set>
</class>

<class name="Pedido">
...
</class>

```

22

Associação Many-to-Many

• Bidirecional

```

<class name="Cliente">
...
<set name="produtos" table="produto_cliente">
  <key column="cliente_id" />
  <many-to-many column="produto_id" class="Produto" />
</set>
</class>

<class name="Pedido">
...
<set name="clientes" inverse="true" table="produto_cliente">
  <key column="produto_id" />
  <many-to-many column="cliente_id" class="Cliente" />
</set>
</class>

```

23

HQL

• **H**ibernate **Q**uery **L**anguage

• Linguagem similar ao SQL

• A HQL usa os conceitos de orientação a objetos


```

Query q = session.createQuery("from Cliente");
List result = q.list();

```

24

Cláusulas *from* e *as*




- *from*
 - Seleciona o tipo da entidade a ser retornada
- *as*
 - alias para a entidade

```
from Cliente
```


```
from Cliente as c
```

```
Retorno: List<Cliente>
```



25

Cláusula *select*




- Filtra os resultados, trazendo apenas os dados desejados das entidades

```
select c from Cliente as c
```

```
Retorno: List<Cliente>
```


```
select c.nome, c.cidade from Cliente as c
```

```
Retorno: List<Object[]>
```



26

Associações




- HQL é capaz de associar entidades automaticamente

```
select c.endereco from Cliente as c
```


```
Retorno: List<Endereco>
```

```
select c.endereco.rua from Cliente as c
```

```
Retorno: List<String>
```



27

Joins



- HQL permite fazer joins entre entidades

```
select c from Pedido as p
inner join p.cliente as c
```

Retorno: List<Cliente>

- Tipos de joins suportados
 - inner join
 - left outer join
 - right outer join

28

Cláusula *where*


- Estipula condições sobre os dados que devem ser selecionados pela query


```
from Cliente as c where c.idade > 18
```

Apenas clientes cuja idade é maior que 18 anos

```
select p from Pedido as p
inner join p.cliente c
where c.nome like '%José%'
```

Apenas pedidos cujo nome do cliente tem 'José' no meio

29

Cláusula *where*


```
from Cliente as c where c.endereco is not null
```


Apenas clientes que possuem endereço

```
select p from Pedido as p
where p.cliente.endereco.estado = 'SP'
```

Apenas pedidos cujo estado do cliente que fez o pedido é 'SP'

30

Funções de Agregação




- A HQL suporta algumas funções de agregação

Função	Descrição
count	Conta registros
min	Valor mínimo
max	Valor máximo
sum	Soma de valores
avg	Média de valores


```
select sum(p.valor) from Produto p
```

Retorna a soma dos valores de todos os produtos




31

Considerações sobre a HQL



- Permite escrever queries de forma orientada a objetos, sem pensar em termos de bancos de dados
- A HQL é bastante extensa, e o que foi visto aqui cobre o básico, que é o mais utilizado
- Para maiores informações, consulte a documentação do Hibernate



32
