

Hibernate ORM



Prof. Dr. João Paulo Lemos Escola
Copyright© 2022

Tópicos da aula

- Nesta aula vamos conhecer o framework Hibernate;
- Com ele podemos manipular banco de dados sem escrever código SQL;
- Possibilita aumento da produtividade do projeto de software.

O que é JPA

- Java Persistence API;
 - É um documento que estabelece diretrizes para serem empregadas por implementações de acesso a dados.

O que é Hibernate

- É uma biblioteca que implementa JPA;
- Trata-se da biblioteca mais utilizada no mundo para implementação de DAOs Java.

Configuração do Hibernate

- Realizada por meio do arquivo persistence.xml;
 - Declaração dos models a serem gerenciados pelo Hibernate;
 - Configuração de acesso ao BD;
 - Configuração do Hibernate.

Exemplo persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
  version="2.0">

  <persistence-unit name="ExemploPU">

    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

    <class>model.Estado</class>

    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://192.168.56.102:3306/Loja5"/>
      <property name="javax.persistence.jdbc.user" value="root"/>
      <property name="javax.persistence.jdbc.password" value=""/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>

      <property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5InnoDBDialect" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.format_sql" value="true" />

      <property name="hibernate.hbm2ddl.auto" value="update" />
    </properties>

  </persistence-unit>

</persistence>
```

Controller Genérico

- Utilizada como pai de todos os controllers do projeto.

```
public class Controller<T> {  
    private Dao dao = null;  
  
    public Controller() {  
        dao = Dao.getIntance();  
    }  
  
    public boolean salvar(T t) {  
        return dao.salvar(t);  
    }  
  
    public boolean excluir(T t){  
        return dao.excluir(t);  
    }  
  
    public List<T> listar(Class c, String campo, String valor){  
        return dao.listar(c, campo, valor);  
    }  
  
    public T get(Class c, int id){  
        return (T)dao.get(c, id);  
    }  
}
```

Controller Estado

- Herda de Controller;
- Pode sobrescrever os métodos do Controller;

```
public class EstadoController extends Controller<Estado>{  
  
    @Override  
    public boolean salvar(Estado t) {  
        return super.salvar(t);  
    }  
  
    @Override  
    public boolean excluir(Estado t) {  
        return super.excluir(t);  
    }  
  
    public List<Estado> buscar(String campo, String valor) {  
        return super.listar(Estado.class, campo, valor);  
    }  
  
    public Estado get(int id) {  
        return super.get(Estado.class, id);  
    }  
  
    public List<Estado> listar() {  
        return super.listar(Estado.class, "nome", "");  
    }  
}
```


Dao genérico

```
public class Dao<T> {  
  
    private EntityManagerFactory factory = null;  
    private EntityManager manager = null;  
    private static Dao instance = null;  
  
    protected Dao() {  
        factory = Persistence.createEntityManagerFactory("ExemploPU");  
        manager = factory.createEntityManager();  
    }  
  
    public static Dao getIntance() {  
        if (instance == null) {  
            instance = new Dao();  
        }  
        return instance;  
    }  
  
    public void fechar() {  
        manager.close();  
        factory.close();  
    }  
  
    public boolean salvar(T t) {  
        try {  
            manager.getTransaction().begin();  
            manager.persist(t);  
            manager.getTransaction().commit();  
        } catch (Exception e) {  
            e.printStackTrace();  
            return false;  
        }  
        return true;  
    }  
}
```

```
    public boolean excluir(T t) {  
        try {  
            manager.getTransaction().begin();  
            manager.remove(t);  
            manager.getTransaction().commit();  
        } catch (Exception e) {  
            e.printStackTrace();  
            return false;  
        }  
        return true;  
    }  
  
    public List<T> listar(Class c, String campo, String valor) {  
        List<T> lista = null;  
        try {  
            manager.getTransaction().begin();  
            CriteriaBuilder builder = manager.getCriteriaBuilder();  
            CriteriaQuery<T> criteria = builder.createQuery(c);  
            Root<T> root = criteria.from(c);  
            criteria.select(root);  
            criteria.where(builder.like(root.get(campo), "%" + valor + "%"));  
            lista = manager.createQuery(criteria).getResultList();  
            manager.getTransaction().commit();  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
        return lista;  
    }  
  
    public T get(Class c, int id){  
        manager.getTransaction().begin();  
        T t = (T)manager.find(c, id);  
        manager.getTransaction().commit();  
        return t;  
    }  
}
```

Model Estado

- Os Models, como Hibernate, precisam ser ajustados com anotações:
 - @Entity: a classe é uma entidade do Hibernate;
 - @Id: o campo é identificação do registro;
 - @GeneratedValue: o campo é auto_increment;

```
@Entity
public class Estado implements Serializable {

    @Id
    @GeneratedValue
    private int id;
    private String nome;
    private String sigla;

    public Estado() {
    }

    public Estado(int id, String nome, String sigla) {
        this.id = id;
        this.nome = nome;
        this.sigla = sigla;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getSigla() {
        return sigla;
    }

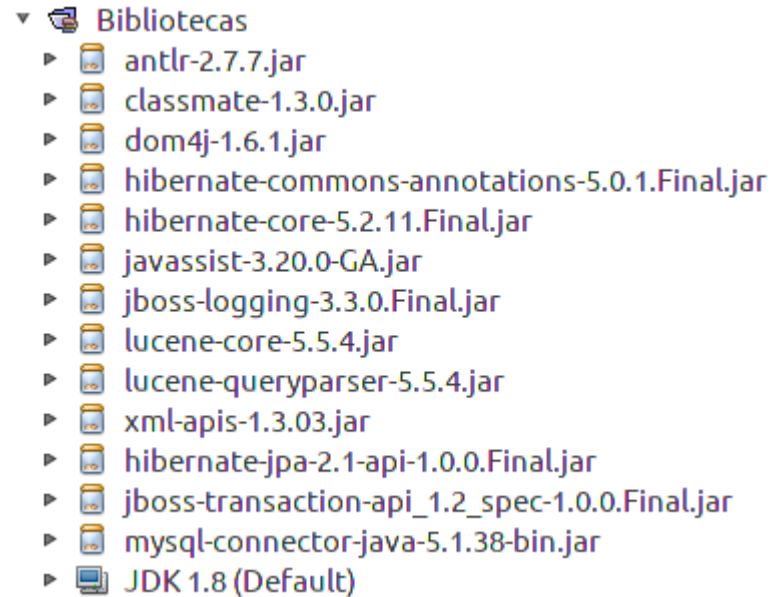
    public void setSigla(String sigla) {
        this.sigla = sigla;
    }
}
```

Botando a mão na massa

- Crie um projeto Java chamado 'ExemploHibernate';
- Crie os pacotes Dao, Model, View e Controller.

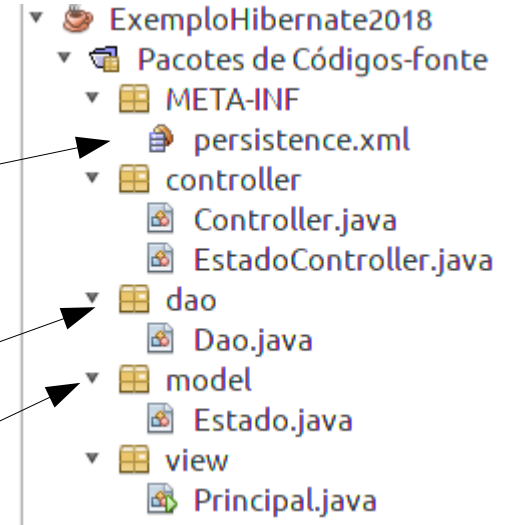
Bibliotecas

- Baixe a biblioteca em
 - <https://sourceforge.net/projects/hibernate/files/hibernate-orm/>
- Inclua em seu projeto:



Estrutura básica

- `persistence.xml`
 - Onde são feitas as configurações do hibernate.
- Pacotes MVC
 - Para utilizar JPA, basta configurar a camada Dao e ajustar a camada model com as anotações do hibernate;
 - As demais camadas não precisam ser ajustadas.



View

- Vamos criar um objeto controler e salvar um novo estado;
- Em seguida listar os estados cadastrados.

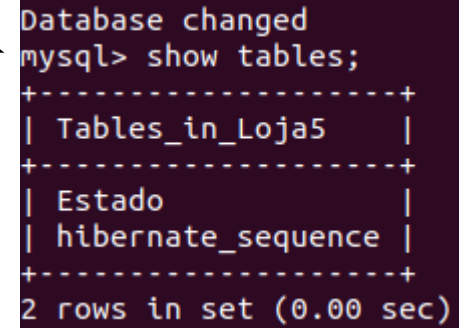
```
EstadoController c = new EstadoController();
try {
    c.salvar(new Estado(0, "Alagoas", "AL"));

    for (Estado e : c.buscar("nome", "")) {
        System.out.println("" + e.getNome());
    }

} catch (Exception e) {
    e.printStackTrace();
}
```

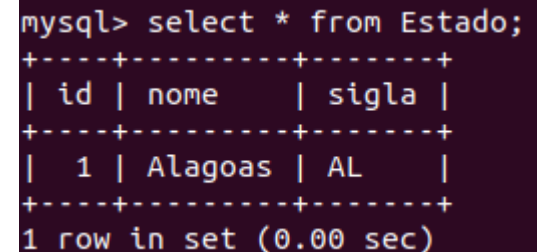
Primeira execução

- Veja que as tabelas foram criadas:
- E os registros incluídos:



Database changed
mysql> show tables;
+-----+
| Tables_in_Loja5 |
+-----+
| Estado |
| hibernate_sequence |
+-----+
2 rows in set (0.00 sec)

An arrow points from the first bullet point to this terminal output. Another arrow points from the second bullet point to the terminal output below.



mysql> select * from Estado;
+----+-----+-----+
| id | nome | sigla |
+----+-----+-----+
| 1 | Alagoas | AL |
+----+-----+-----+
1 row in set (0.00 sec)

An arrow points from the second bullet point to this terminal output.

Por quê o programa não termina?

- Estamos acostumados a ver um...

```
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)
```

...ao final de nossos programas.

- Com hibernate o processamento fica ativo;
- Para resolver isso, vamos incluir um **System.exit(0)** ao final do método **main** para que seja finalizado ao final da View.

Model Cidade

- Agora vamos incluir um novo model no nosso projeto;
- Veja que agora temos uma nova anotação:
 - @ManyToOne: este objeto tem relacionamento N para 1 com o model atual;
 - Isso vai permitir que você consiga buscar o objeto Estado a partir de um objeto Cidade.

```
@Entity
public class Cidade implements Serializable {

    @Id
    @GeneratedValue
    private int id;
    private String nome;

    @ManyToOne
    private Estado estado;

    public Cidade() {
    }

    public Cidade(int id, String nome, Estado estado) {
        this.id = id;
        this.nome = nome;
        this.estado = estado;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public Estado getEstado() {
        return estado;
    }

    public void setEstado(Estado estado) {
        this.estado = estado;
    }
}
```

Controller Cidade

- CidadeController herda de Controller;
- Mesmo padrão do controller de Estado.

```
public class CidadeController extends Controller<Cidade>{

    @Override
    public boolean salvar(Cidade t) {
        return super.salvar(t);
    }

    @Override
    public boolean excluir(Cidade t) {
        return super.excluir(t);
    }

    public List<Cidade> buscar(String campo, String valor) {
        return super.listar(Cidade.class, campo, valor);
    }

    public Cidade get(int id) {
        return super.get(Cidade.class, id);
    }

    public List<Cidade> listar() {
        return super.listar(Cidade.class, "nome", "");
    }
}
```

Testando na View

- Criando dois objetos cidade:

```
EstadoController ce = new EstadoController();
CidadeController cc = new CidadeController();
try {
    Estado alagoas = ce.get(1);
    cc.salvar(new Cidade(0, "cidade1", alagoas));
    cc.salvar(new Cidade(0, "cidade2", alagoas));
} catch (Exception e) {
    e.printStackTrace();
}
```

- Verificando o resultado no BD:

```
mysql> select * from Cidade;
+----+-----+-----+
| id | nome   | estado_id |
+----+-----+-----+
|  2 | cidade1 |          1 |
|  3 | cidade2 |          1 |
+----+-----+-----+
2 rows in set (0.00 sec)
```

Buscando na View

- Exemplo da View:

```
public static void main(String[] args) {  
    EstadoController ec = new EstadoController();  
    CidadeController cc = new CidadeController();  
    try {  
        Cidade c1 = cc.get(1);  
        System.out.println("cidade: "+c1.getNome());  
        System.out.println("estado: "+c1.getEstado().getNome());  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
    System.exit(0);  
}
```

@OneToMany

- A anotação @OneToMany serve para relacionar entidades 1 para N:

```
@OneToMany(mappedBy="estado", fetch = FetchType.LAZY)
private List<Cidade> cidades;

public List<Cidade> getCidades() {
    return cidades;
}

public void setCidades(List<Cidade> cidades) {
    this.cidades = cidades;
}
```

Buscando Cidades do Estado

- Agora que declaramos o @OneToMany, podemos listar as cidades de um objeto estado:

```
EstadoController ec = new EstadoController();
CidadeController cc = new CidadeController();
try {
    Estado al = ec.get(1);
    for (Cidade c : al.getCidades())
        System.out.println("cidade: "+c.getNome());
} catch (Exception e) {
    e.printStackTrace();
}

System.exit(0);
```

Possíveis Exceções

- Base de dados não existe, precisa ser criada:

```
Caused by: com.mysql.jdbc.exceptions.jdbc4.MySQLSyntaxErrorException: Unknown database 'Loja4'  
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)  
    at sun.reflect.NativeConstructorAccessorImpl.newInstance(NativeConstructorAccessorImpl.java:62)  
    at sun.reflect.DelegatingConstructorAccessorImpl.newInstance(DelegatingConstructorAccessorImpl.java:45)  
    at java.lang.reflect.Constructor.newInstance(Constructor.java:423)
```

- Falta o .jar do mysql-connector:

```
Caused by: org.hibernate.boot.registry.classloading.spi.ClassLoadingException: Unable to load class [com.mysql.jdbc.Driver]  
    at org.hibernate.boot.registry.classloading.internal.ClassLoaderServiceImpl.classForName(ClassLoaderServiceImpl.java:348)  
    at org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl.loadDriverIfPossible(DriverManagerConnectionProviderImpl.java:160)  
    at org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl.buildCreator(DriverManagerConnectionProviderImpl.java:116)  
    at org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl.buildPool(DriverManagerConnectionProviderImpl.java:100)
```

O que aprendemos?

- Framework Hibernate.

Na próxima aula...

- Classes de componentes gráficos;
- Desenvolvimento de UI no NetBeans;
- Eventos.