Acessando Dados com Java: Parte 3 - Hibernate Annotations

Publicado por jesuino em 25/09/2010 - 84.963 visualizações

comentários: 8

Acompanhe a parte 1(apresentação de um simples, mas defeituoso, acesso a dados usando JDBC) e a parte 2(Possíveis melhorias para a parte 1).

Objetivo: Mostrar um exemplo simples de acesso a dados com Java usando Hibernate Annotations.

Requerimentos: Para realizar esse tutorial você deve ter os seguintes JARs no seu projeto(No eclipse, de preferência), mais o jar para o driver do seu banco de dados utilizado:
0
Baixar Projeto do Eclipse

Persistindo Pessoas
Iremos utilizar o mesmo exemplo de entidade dos tutoriais anteriores, a entidade pessoa, que ficou como segue:

> **Quote:**
>
> **Pessoa:**
> ID, RG, NOME, IDADE, CIDADE, ESTADO

Crie sua tabela no banco de dados. Nesse tutorial foi utilizado MySql como banco de dados, se também utilizar MySql, pode criar a tabela com o script abaixo:

```
CREATE TABLE pessoa (
  pessoa_id int(11) NOT NULL auto_increment,
  pessoa_rg varchar(20) default NULL,
  pessoa_nome varchar(20) default NULL,
  pessoa_idade int(2) default NULL,
  pessoa_cidade varchar(20) default NULL,
  pessoa_estado varchar(2) default NULL,
  PRIMARY KEY  (pessoa_id)
)
```

Hibernate Annotations permite que você realize o mapeamento ORM sem utilizar XML, somente Annotations. Para isso você deve trabalhar com classes POJOs, ou seja, atributos privados, acessados pelos famosos métodos GET e SET.
Crie um pacote model e dentro a seguinte classe anotada:

```java
package model;
import javax.persistence.*;

@Entity
@Table(name="PESSOA")
public class Pessoa {
   private int id;
   private String rg;
   private String nome;
   private int idade;
   private String estado;
   private String cidade;

   @Id
   @GeneratedValue
   @Column(name="PESSOA_ID")
   public int getId() {
      return id;
   }
   public void setId(int id) {
      this.id = id;
   }

   @Column(name="PESSOA_RG", nullable=false)
   public String getRg() {
      return rg;
   }
   public void setRg(String rg) {
      this.rg = rg;
   }

   @Column(name="PESSOA_NOME", nullable=false)
   public String getNome() {
      return nome;
   }
   public void setNome(String nome) {
      this.nome = nome;
   }

   @Column(name="PESSOA_IDADE")
   public int getIdade() {
      return idade;
   }
   public void setIdade(int idade) {
      this.idade = idade;
   }

   @Column(name="PESSOA_ESTADO")
   public String getEstado() {
      return estado;
   }
   public void setEstado(String estado) {
      this.estado = estado;
   }

   @Column(name="PESSOA_CIDADE")
   public String getCidade() {
      return cidade;
   }
   public void setCidade(String cidade) {
      this.cidade = cidade;
   }
}
```

A seguir uma explicação básica sobre cada anotação:
@Entity: Usamos para marcar uma classe como entidade do banco de dados. Esta classe deve estar em um pacote e não ter argumentos em seu construtor.
@Table: Essa anotação serve para indicar em qual tabela iremos salvar os dados. Se você não usar essa anotação, o Hibernate usa o nome da classe para a tabela. O atributo **name** refere-se ao nome da tabela.
@Id: Usamos para mostra o identificador único(chave primária) de nossa classe persistente. No nosso caso, o identificador único é o campo ID.
GeneratedValue: Indica que o valor para o identificador único será gerado automaticamente. Você pode configurar a forma de geração dos valores através do atributor **strategy**. Se você não colocar uma estratégia, será usada a estratégia AUTO.
0
@Column: Utilizamos para especificar uma coluna da tabela do banco de dados. No exemplo acima, especificamos que o campo *RG* e *nome* . Se você não mapear a coluna, o nome da propriedade será usado como nome da coluna.
0

Configurando a fonte de dados
O próximo passo é configurar nossa fonte de dados. Nesse ponto usamos mapeamento XML, o único. Nos iremos adicionar os dados da fonte e as classes mapeadas. Observe bem os parâmetros da fonte de dados, essa parte é muito comum apresentar erros. Meu XML ficou assim:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">

<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">com.mysql.jdbc.Driver</property>
        <property name="connection.url">jdbc:mysql://localhost:3306/simples_tutorial_annotations</property>
        <property name="connection.username">root</property>
        <property name="connection.password">senha</property>

        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>

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

        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>

        <!-- Mapping files -->
        <mapping class="model.Pessoa"/>
    </session-factory>
</hibernate-configuration>
```

A fábrica de sessões
Quem já mexeu com Hibernate conhece muito bem a famosa fábrica de Sessões e a classe HibernateUtil. Crie um pacote chamado util e uma classe chamada **HibernateUtil**. Abaixo nossa classe HibernateUtil:

```
package util;

import org.hibernate.SessionFactory;
import org.hibernate.cfg.AnnotationConfiguration;
public class HibernateUtil {
   private static final SessionFactory sessionFactory;
   static {
      sessionFactory = new AnnotationConfiguration().configure()
            .buildSessionFactory();
   }

   public static SessionFactory getSessionFactory() {
      return sessionFactory;
   }
}
```

Agora nos resta testar para verificar se tudo que foi feito até aqui está certo. Uma classe de teste:

```
import model.Pessoa;
import org.hibernate.Session;
import util.HibernateUtil;
import org.hibernate.Transaction;

public class Teste {
   public static void main(String[] args) {
      Session sessao = HibernateUtil.getSessionFactory().openSession();
Transaction t = sessao.beginTransaction();

      Pessoa pessoa = new Pessoa();
      pessoa.setNome("William");
      pessoa.setRg("123456");
      pessoa.setCidade("São José dos campos");
      pessoa.setEstado("SP");
      pessoa.setIdade(21);
      sessao.save(pessoa);
      t.commit();
      sessao.close();

   }
}
```

Perceba que neste teste simplesmente estamos instanciando uma pessoa e inserindo no banco de dados. Não há um DAO para abstrair as operações CRUD para com o banco. O próximo passo é criar esse DAO, que possibilitará maior flexibilidade e encapsulamento das operações com o banco de dados.

Conclusão
Apresentamos uma pequena amostra de persistência usando Hibernate Annotations. O foco desse tutorial, assim como dos outros, foi a simplicidade. É claro que quem conseguir terminar este tutorial não será o mestre Hibernate Annotations, mas terá um impulso para começar seus estudos.
Esperamos que todos possam usufruir desse tutorial, e lembramos que o fórum está aberto para retirada de dúvidas e compartilhamento de experiências.

Baixar Projeto do Eclipse

Futuro
Esse será o acesso básico para as próximas partes do nosso tutorial, onde mostraremos como montar interfaces para que um usuário comum possa utilizar-se do sistema.

Fonte: Vanilla Hibernate Annotation

Artigos desta série:

Acessando dados com Java: 1º parte - Simples Dao
Acessando Dados com Java: Parte 2 - Prevendo problemas
Acessando Dados com Java: Parte 3 - Hibernate Annotations

Leia também:
Acessando Banco de Dados em Java (PARTE 1)
Acessando Banco de Dados em Java (PARTE 2)
Acessando Banco de Dados em Java (PARTE 3)
Hibernate 3 com Spring e DAO Generico

Quer aprender mais sobre Java?
O que é Java?
Características Básicas
Orientação a Objetos

Tutoriais para Certificação Java
Fundamentos da Linguagem
Modificadores
Operadores e atribuições
Controle de Fluxo
Orientação a Objetos
Java Lang e Wrappers
Objetos e Conjuntos
Classes Internas
Threads (Segmentos)

Baixar: Propriedades Coluna.jpg
Tamanho: 18 KB

Baixar: valores.jpg
Tamanho: 15 KB

Baixar: Jars.jpg
Tamanho: 16 KB

comentários: 8