View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Infoteca-e

Documentos
Dezembro, 2003 39

ISSN 1677-9274

Uma Introdução a Tecnologia de Reflexão Java

Embrapa

República Federativa do Brasil

Luiz Inácio Lula da Silva
Presidente

Ministério da Agricultura, Pecuária e Abastecimento

Roberto Rodrigues
Ministro

Empresa Brasileira de Pesquisa Agropecuária – Embrapa

Conselho de Administração

José Amauri Dimárzio
Presidente

Clayton Campanhola
Vice-Presidente

Alexandre Kalil Pires
Dietrich Gerhard Quast
Sérgio Fausto
Urbano Campos Ribeiral
Membros

Diretoria Executiva da Embrapa

Clayton Campanhola
Diretor-Presidente

Gustavo Kauark Chianca
Herbert Cavalcante de Lima
Mariza Marilena T. Luz Barbosa
Diretores-Executivos

Embrapa Informática Agropecuária

José Gilberto Jardine
Chefe-Geral

Tércia Zavaglia Torres
Chefe-Adjunto de Administração

Sônia Ternes Frassetto
Chefe-Adjunto de Pesquisa e Desenvolvimento

Álvaro Seixas Neto
Supervisor da Área de Comunicação e Negócios

Empresa Brasileira de Pesquisa Agropecuária
Embrapa Informática Agropecuária
Ministério da Agricultura, Pecuária e Abastecimento

ISSN 1677-9274
Dezembro, 2003

Documentos 39

Uma Introdução a Tecnologia de Reflexão Java

Silvio Roberto Medeiros Evangelista

Campinas, SP
2003

Embrapa Informática Agropecuária
Área de Comunicação e Negócios (ACN)
Av. André Tosello, 209
Cidade Universitária "Zeferino Vaz" Barão Geraldo
Caixa Postal 6041
13083-970 Campinas, SP
Telefone (19) 3789-5743 Fax (19) 3289-9594
URL: http://www.cnptia.embrapa.br
e-mail: sac@cnptia.embrapa.br

Comitê de Publicações

Carla Geovana Nascimento Macário
José Ruy Porto de Carvalho
Luciana Alvim Santos Romani (Presidente)
Marcia Izabel Fugisawa Souza
Marcos Lordello Chaim
Suzilei Almeida Carneiro

Suplentes

Carlos Alberto Alves Meira
Eduardo Delgado Assad
Maria Angelica de Andrade Leite
Maria Fernanda Moura
Maria Goretti Gurgel Praxedis

Supervisor editorial: *Ivanilde Dispato*
Normalização bibliográfica: *Marcia Izabel Fugisawa Souza*
Editoração eletrônica: *Área de Comunicação e Negócios (ACN)*

1ª edição on-line - 2003
Todos os direitos reservados.

Evangelista, Silvio Roberto Medeiros.
Uma introdução a tecnologia de reflexão Java / Silvio Roberto Medeiros Evangelista — Campinas: Embrapa Informática Agropecuária, 2003.
19 p. : il. — (Documentos / Embrapa Informática Agropecuária; 39).

ISSN 1677-9274

1. Reflexão Java. 2. Linguagem Java. I .Título. II. Série.

CDD - 005.133 (21st ed.)

© Embrapa 2003

Autor

Silvio Roberto Medeiros Evangelista
Dr. em Engenharia Elétrica, Técnico de Nível Superior da Embrapa Informática Agropecuária, Caixa Postal 6041, Barão Geraldo - 13083-970 Campinas, SP
Telefone (19) 3789-5740
e-mail: silvio@cnptia.embrapa.br

Apresentação

Este documento apresenta a tecnologia de reflexão existente na linguagem JAVA. A Reflexão Java admite que uma aplicação instancie e utilize objetos Java dinamicamente, sem que o programador tenha conhecimento das classes que serão empregadas.

O mecanismo de reflexão oferece acesso programático a construtores, métodos e campos de uma classe que são desconhecidos no momento da compilação do código fonte de um programa. Desta forma, é admissível que uma classe utilize outra, mesmo que a última ainda não exista no momento da compilação da primeira.

A reflexão fornece mecanismos poderosos que permitem a construção de programas sofisticados. Podem-se citar como exemplos: programas de testes, navegadores de classes, inspetores de objetos, ferramentas de análise de código, sistemas interpretados, programas concorrentes, programas com chamadas remotas, aplicações que envolvam uma camada de persistência com um banco de dados, etc.

Este documento visa orientar especialistas envolvidos no desenvolvimento de ferramentas computacionais baseadas em componentes reusáveis. Neste sentido, os conceitos e os exemplos apresentados são baseados em um caso real de uma ferramenta reusável que fornece a funcionalidade de mapeamento entre objetos de um programa computacional e de um banco de dados relacional. Este mapeamento é também conhecido como camada de persistência.

José Gilberto Jardine
Chefe-Geral

Sumário

Introdução 9

Persiste: uma Camada de Persistência 10

Descrição da Tabela de Nome Climadia..............................10
Descrição da Classe para Mapeamento com a Tabela..............11

Reflexão Java: Estratégia de Implantação da Classe Persiste...14

Considerações Finais..17

Referências Bibliográficas..19

Uma Introdução a Tecnologia de Reflexão Java

Silvio Roberto Medeiros Evangelista

Introdução

Este documento tem por objetivo apresentar a tecnologia de reflexão existente na linguagem JAVA (Wu & Schwiderski, 1997). Esta exposição será realizada a partir de uma implementação real de uma classe Java que fornece a funcionalidade de mapeamento entre objetos de um programa computacional e de um banco de dados relacional. Este mapeamento é também conhecido como camada de persistência.

A reflexão foi apresentada pela primeira vez por Maes (1987), e pode ser definida como qualquer atividade executada por um sistema computacional sobre si mesmo, no intuito de obter informações sobre suas próprias tarefas ou objetos. Em outras palavras, a reflexão é a capacidade de um programa reconhecer detalhes internos em tempo de execução que não estavam disponíveis no momento da compilação do programa.

O mecanismo de reflexão oferece acesso programático a construtores, métodos e campos de uma classe que são desconhecidos no momento da compilação do código fonte. Desta forma, é possível que uma classe utilize outra, mesmo que a última ainda não existisse quando a primeira foi compilada. É um mecanismo poderoso, que permite a construção de programas sofisticados. Podem-se citar como exemplos: programas de testes, navegadores de classes, inspetores de objetos, ferramentas de análise de código, sistemas interpretados, programas concorrentes, programas com chamadas remotas, aplicações que envolvam uma camada de persistência com um banco de dados, etc.

O exemplo utilizado neste trabalho para descrever os recursos de reflexão da linguagem Java implementa uma camada de persistência, a partir de agora chamada de Persiste, que é uma infra-estrutura objeto-relacional que permite a recuperação e o armazenamento de objetos em bancos de dados relacionais (RDBMS). Ela é composta por alguns componentes que cooperam entre si para permitir que programas escritos na linguagem Java possam acessar, de forma

simples e direta, banco de dados relacionais. Esta ferramenta suporta os princípios de encapsulamento, hierarquia e abstração, características importantes que são suportadas pela tecnologia de reflexão.

Persiste: uma Camada de Persistência

Esta classe foi inicialmente implementada para dar suporte de persistência ao projeto Agência de Informação (Embrapa Informática Agropecuária, 2003). Neste projeto, a implementação da camada de negócio (requisitos funcionais) ficou separada da implementação dos requisitos não funcionais ou de infra-estrutura (por exemplo: da camada de persistência). Um dos problemas enfrentados era a inexistência de tempo hábil para se esperar a conclusão da camada de persistência (chamada de Persiste) para se começar a implementação das classes da aplicação e, por outro lado, a implementação da classe Persiste não poderia ficar atrelada ao desenvolvimento das classes de negócio. Assim, a única informação compartilhada entre os dois grupos de desenvolvimento era a regra de formação de uma classe na camada de negócio, isto é:

- o nome da classe corresponderia ao nome de uma tabela no banco de dados;
- os atributos da classe representariam as colunas da tabela, uma coluna (não importa o seu tipo: número, data, texto, etc.) seria representada por uma STRING;
- um atributo especial com o nome formado pelas letras "id" seguido do nome da classe (por exemplo, idClima, idCadastro, ...) representaria a chave primária da tabela;
- as classes possuiriam funções para atribuição e recuperação do valor de um atributo. Estas funções deveriam possuir os seguintes nomes: *setNomedoAtributo* e *getNomedoAtributo.*

O exemplo a seguir mostra uma descrição de tabela de um banco de dados e uma classe Java (*ClimaDia*) que foi escrita conforme as regras definidas.

Descrição da Tabela de Nome Climadia

```
IDCLIMADIA       NOT NULL VARCHAR2(32)
IDESTACOES                VARCHAR2(16)
TMAX                      NUMBER
TMIN                      NUMBER
PRECIPITACAO              NUMBER
DATA             NOT NULL DATE
```

Descrição da Classe para Mapeamento com a Tabela

```
public class ClimaDia {
        String idClimaDia;
        String idEstacoes;
        String data;
        String tmax;
        String tmin;
        String precipitacao;

        public void setIdClimaDia (String v) {
                idClimaDia = v;
        }
        Public String getIdClimaDia () {
                return idClimaDia;
        }
        public void setIdEstacoes (String v) {
                idEstacoes = v;
        }
        public String getIdEstacoes () {
                return idEstacoes;
        }
        public void setData (String valor) {
                data = valor;
        }
        public String getData () {
                return data;
        }
        public void setTmax (String valor) {
                tmax = valor;
        }
        public String getTmax () {
                return tmax;
        }
        public void setTmin (String valor) {
                tmin = valor;
        }
        public String getTmin () {
                return tmin;
        }
        public void setPrecipitacao (String v) {
                precipitacao = v;
        }
        public String getPrecipitacao () {
                return precipitacao;
        }
}
```

O desenvolvimento da classe Persiste foi pautado na simplicidade de configuração do banco a ser utilizado pela aplicação e na facilidade de uso para recuperação e para alteração dos dados armazenados no banco de dados.

Esta camada deveria oferecer as seguintes características para manutenção da consistência entre os objetos da aplicação e o banco de dados relacional:

1. vários tipos de mecanismos de persistência: implementar a inclusão, a remoção e a atualização de objetos no banco de dados relacional;
2. encapsulamento completo dos mecanismos de persistência: executar a mensagem enviada (como armazenar, remover, recuperar) sem necessidade de se conhecer detalhes do banco de dados ou mesmo a linguagem SQL;
3. Transações: controlar as ações por meio de transações que terminam com um *commit* (caso tenha sido realizada com sucesso) ou um *rollback* (caso uma falha tenha ocorrido durante a transação);
4. capacidade de recuperação de objetos ou lista de registros: retornar registros ou uma lista de objetos como resultado de uma determinada ação de leitura no banco de dados;
5. várias versões de banco de dados: facilitar o processo de mudança dos mecanismos de persistência para outro fornecedor ou para outra versão de banco de dados;
6. SQL queries: permitir a escrita de *queries* SQL diretamente na aplicação orientada a objetos (violação do encapsulamento). Todavia, o uso direto de códigos SQL pode ser utilizado, caso seja requerido, para aprimorar o desempenho da aplicação. A utilização direta de comandos SQL deve ser a exceção e não a regra de implementação.

A Fig. 1 apresenta a abordagem de persistência adotada pela Persiste para mapear objetos para um banco de dados relacional de tal maneira que pequenas mudanças em seu esquema não afetem os mecanismos de persistência, permitindo que alterações possam ser feitas no banco sem que isto afete o comportamento da classe Persiste. A vantagem desta abordagem é que os programadores não necessitam conhecer os esquemas adotados no banco de dados. De fato, eles nem precisam saber que os dados estão armazenados em um banco relacional. Esta característica viabiliza o desenvolvimento de aplicações em larga escala por uma determinada organização. A desvantagem é a eventual redução do desempenho da aplicação, que pode ser minorada a partir do uso adequado da camada de persistência.

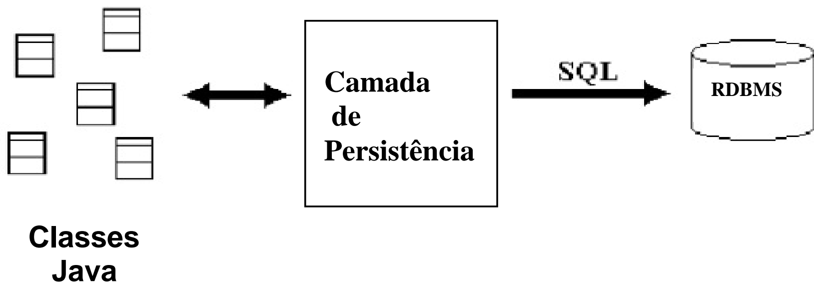

Fig. 1. Camada de Persistência Fornecida pela Persiste.

A utilização do banco de dados é totalmente transparente ao programador, uma vez que a Persiste reconhece uma classe em tempo de execução, descobrindo o seu nome e seus atributos e gerando comandos SQLs de acordo com a mensagem de persistência requerida pelo programador. A Fig. 2 ilustra esquematicamente a Persiste.

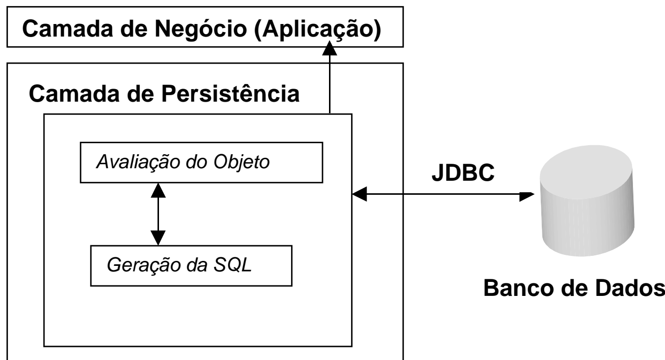

Fig. 2. Classe Persiste.

Como exemplificação de uso da Classe Persiste, considere o exemplo a seguir que requisita a gravação de um objeto (clima) no banco de dados:

Persiste.*rdGrava((Object) clima)*

Deve-se observar neste exemplo a invocação de uma função rdGrava da classe Persiste, cujo parâmetro representa o objeto a ser gravado, previamente preenchido pela aplicação do usuário, e que só a camada de negócio conhece. A Persiste, por sua vez, não conhece a classe que o objeto representa. Ela recebe um objeto genérico, do tipo *Object*, e descobre o seu nome, os seus atributos de classe e os seus métodos para recuperação ou alteração dos valores dos atributos, utilizando os recursos de reflexão da linguagem Java. Além disso, os valores dos atributos são gravados no banco de dados.

A classe Persiste possui inúmeros outros métodos para manipulação de um banco de dados, os quais fogem do objetivo deste trabalho. Todo o restante do trabalho será focado na implementação do método *rdGrava,* o que será suficiente para descrever e exemplificar a tecnologia de reflexão Java.

Reflexão Java: Estratégia de Implementação da Classe Persiste

A Reflexão Java permite que uma aplicação instancie e utilize objetos Java dinamicamente, sem que o desenvolvedor tenha conhecimento das classes que serão utilizadas. Através do pacote *java.lang.reflect* tem-se acesso aos elementos de uma classe: Atributos, Métodos e Construtor (Sun Microsystems, 2003a, 2003b, 2003c, 2003d).

O processo para utilização da reflexão é razoavelmente simples e coerente. Os exemplos que se seguem permitem a obtenção de informações sobre a classe recebida como parâmetro. São basicamente as informações que um programador precisaria saber para utilizar uma classe convenientemente.

Para a obtenção do nome de um objeto é necessário a invocação do método *getClass* para recuperação de uma classe associada a um objeto e a chamada do método *getName* para a obtenção do nome desta classe recuperada (Sun Microsystems, 2003a, 2003b). Estes dois passos podem ser sintetizados no exemplo a seguir que imprime o nome da classe de um objeto recebido como parâmetro.

```
void imprimeNomeClasse(Object obj) {
            System.out.println("O nome da classe é " +
            obj.getClass().getName());
}
```

Os dados a respeito dos campos públicos que compõem uma classe podem ser obtidos da seguinte forma:

```
  void imprimeNomeCamposDeclarados(Object obj) {
     try{

       Field[] fl = obj.getClass().getDeclaredFields();
       for (int i=0; i<fl.length;i++){
              // exibindo o nome da variável
              System.out.println(" Nome:" + l[i].getName();
              // exibindo o tipo de definição da variável
              System.out.println(" Tipo: " +
              fl[i].getType().getName();
       }
     }
    catch(ClassNotFoundException e)
              {System.out.println("Classe não encontrada ");}
}
```

Os métodos de um objeto podem ser obtidos a partir da invocação *getDeclaredMethods* da classe em questão (Sun Microsystems, 2003c, 2003d). O exemplo a seguir recupera o nome, o tipo e o valor de cada um dos métodos de um objeto.

```
void imprimeMetodosDeclarados(Object obj) {

    try{
      Method[] fl = obj.getClass().getDeclaredMethods();
      for (int i=0; i<fl.length;i++){
            // exibindo o nome do método
            System.out.println(" Nome: " + fl[i].getName();
            // exibindo o tipo de definição do método
            String tipo = fl[i].getType().getName();
            System.out.println(" Tipo: " + tipo);
            // exibindo o valor associado com a invocação de um
            // método
            System.out.println( (String) Method[i].invoke(obj, null));
      }
    }
    catch(ClassNotFoundException e)
             {System.out.println("Classe não encontrada ");}
}
```

Os três exemplos anteriores mostram detalhadamente como recuperar as informações relacionadas a uma classe Java. Os métodos ali exemplificados serão utilizados em um exemplo mais complexo que implementa o método rdGrava da classe Persiste.

Este método tem como funcionalidade básica a gravação de um objeto em um banco de dados relacional. Ele recebe um parâmetro do tipo *Object*, cria uma classe, descobre o seu nome, recupera o valor correspondente ao campo *idNomedaClasse* (que representa o valor da chave primária), descobre quantos e quais são os atributos do objeto recebido e recupera o valor de cada um deles. Durante o processamento dos atributos, uma STRING com a SQL desejada será montada para posterior execução. Observa-se que no código a seguir os comandos relacionados com a reflexão estão destacados em negrito.

```
import java.lang.reflect.*;

public void rdGrava(Object obj) throws SQLException {
    Method aMetodo;
    String id;
    String nomeID = null;
    String nomeClasse;
    String SQL;
    String valor;
    String virgula = " ";
    Class cl;

 try {
```

```
cl = obj.getClass();  // recupera a classe do objeto recebido como parâmetro

nomeClasse = cl.getName(); // Descobre o nome da classe.

// Faz a invocação do método getIdNomedaClasse e recupera o valor do campo
try {
      nomeID = "Id" + nomeClasse;
      aMetodo = cl.getMethod("get" + nomeID,  null);  // recupera o endereço do método
                                                      // cujo nome é passado como
                                                      //parâmetro
      id = (String) aMetodo.invoke(obj, null);  //  invoca o método para recuperar o valor da
                                                // chave primária
 }catch (Exception e) {
          throw new SQLException("\nErro de mapeamento rdGrava(1): "+
                "get" + nomeID + ".\n"+ e.toString());
}

// Recuperar todos os campos públicos da classe

Field[] publicFields = cl.getDeclaredFields();    // Recuperada todos os campos declarados

SQL = " UPDATE " + nomeClasse + " SET ";

// Percorre todos os campos e gera o comando SQL

for (int i = 0; i < publicFields.length; i++) {

  String fieldName = publicFields[i].getName();// Nome do atributo

  Class typeClass = publicFields[i].getType();

  String fieldType = typeClass.getName();//Nome do Tipo do tributo

  try {
         // Invoca as interfaces do objeto para recuperar valores a serem inseridos

         aMetodo = cl.getMethod("get" + upperFirst(fieldName), null);

         valor = (String) aMetodo.invoke(obj, null); // Recupera valor do
                                                     // atributo

         if (valor == null) valor = "";

  }Catch (NoSuchMethodException e) {
       throw new SQLException("\nErro de mapeamento grava: "+
                              "get" +
                               upperFirst(fieldName)
                               + " "+ e.toString());
  }
  // Criar coluna para mudar valor, exceto da coluna ID

  SQL = SQL + virgula;

  SQL = SQL + virgula;

  SQL = SQL + fieldName + " = '" + valor.replace('\'','`') + "'";

  Virgula = " , ";

}
SQL = SQL + " WHERE " + "TRIM("+nomeID + ")" + "='" + id.trim() + '";

.... Executar SQL com comandos JBDC
```

O método rdGrava recebe como parâmetro um objeto que foi instanciado pela aplicação. Caso o objeto a ser utilizado tenha que ser carregado do disco e instanciado a partir de um nome, pode-se utilizar as seguintes linhas de comandos:

```
Class cl = Class.forName("teste"); // Carrega a classe de nome teste
Object obj = cl.newInstance(cl);  // Cria um objeto a partir da classe
                                  // carregada
```

O comando *forName* carrega uma classe que está gravada como arquivo em disco e o método *newInstance* cria um objeto desta classe (Sun Microsystems, 2003a, 2003b).

Do exemplo anterior pode-se concluir que os requisitos não funcionais de uma aplicação podem ser isolados em classes externas (meta-classe), incorporando assim capacidades inerentes à orientação a objetos, como reutilização e encapsulamento. Essas metas-classe podem ser novamente utilizadas em outras aplicações que as necessitem. Esta possibilidade torna-se importante no incremento da velocidade de desenvolvimento de sistemas, permitindo ao programador incorporar tais características a seus programas de forma rápida, simples e eficiente. Basta selecionar as classes que fornecem suporte às características desejadas no sistema e introduzi-las no mesmo, sem necessidade de recodificação ou mesmo entendimento de seu funcionamento interno, pois o encapsulamento demanda do programador apenas o conhecimento sobre os métodos a serem chamados. Eventuais melhorias na implementação da classe reusável podem ser feitos através da simples substituição ou evolução da meta-classe, sem que haja necessidade de alteração no código da aplicação.

Considerações Finais

Um dos problemas de manutenção enfrentados por desenvolvedores de softwares é que, uma vez que os programas estão desenvolvidos e em operação, torna-se extremamente difícil e complexo alterar o seu comportamento sem modificação de seu código fonte. As demandas que ocorrem na prática podem exigir do sistema uma flexibilidade muitas vezes inatingível por meio dos mecanismos tradicionais existentes nas linguagens de programação. Mantendo-se este cenário em mente, percebe-se o enorme potencial de aplicação de um paradigma de reflexão computacional.

A reflexão é um mecanismo poderoso, mas tem desvantagens que precisam ser consideradas antes da sua utilização:

- perde-se o benefício das verificações de tipo que acontecem em tempo de compilação, incluindo checagem de exceções;

- o código necessário para realizar a reflexão é mais complexo que o tradicional;
- existe um impacto de performance, no Java 1.4, por exemplo, o tempo necessário para a invocação de um método é o dobro se comparado ao acesso normal.

A reflexão foi inserida na linguagem Java para permitir a construção de ferramentas baseadas em componentes reusáveis. Isto implica que objetos não devem ser acessados por reflexão em uma aplicação normal sem que haja necessidade.

Referências Bibliográficas

EMBRAPA INFORMÁTICA AGROPECUÁRIA. Projeto Agência. Disponível em: <http://alphard.cnptia.embrapa.br/agencia/>. Acesso em: 10 dez. 2003.

MAES, P. Concepts and experiments in computational reflection. ACM Sigplan Notices, v. 22, n. 12, p. 147-155, Dec. 1987.

SUN MICROSYSTEMS. Class (Java 2 Platform SE v1.4.2). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Class.html>. Acesso em: 10 dez. 2003a.

SUN MICROSYSTEMS. Object (Java 2 Platform SE v1.4.2). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Object.html>. Acesso em: 10 dez. 2003b.

SUN MICROSYSTEMS. Method (Java 2 Platform SE v1.4.2). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/reflect/Method.html>. Acesso em: 10 dez. 2003c.

SUN MICROSYSTEMS. Field (Java 2 Platform SE v1.4.2). Disponível em: <http://java.sun.com/j2se/1.4.2/docs/api/java/lang/reflect/Field.html>. Acesso em: 10 dez. 2003d.

WU, Z.; SCHWIDERSKI, S. Reflective, making java even more flexible. Cambridge, UK: ANSA,1997. (Technical Report APM 1936.02). ANSA Phase III.

Informática Agropecuária

Ministério da Agricultura, Pecuária e Abastecimento
Governo Federal