JavaFree.com.br

[Tutoriais] - Banco de Dados - Acessando Bancos de Dados em Java (PARTE 2)

aspirante - Qui Out 16, 2003 11:26 pm
Assunto: Acessando Bancos de Dados em Java (PARTE 2)

As chamadas de JDBC
No artigo passado, mostramos o primeiro exemplo de acesso a um banco de dados com JDBC. Neste artigo começaremos a explicar alguns motivos implementados no padrão utilizando o exemplo do artigo passado. O que nos chama a atenção logo de cara é que o código JDBC não se assemelha ao que o programador iniciante em Java está acostumado. Por exemplo, fazemos:

Código:

```
Connection con = DriverManager.getConnection(parametros da conexão);
```

ao invés de

Código:

```
Connection con = new Connection();
```

JDBC é um padrão onde está definida uma estrutura geral de acesso ao driver de banco de dados através de interfaces e o fornecedor se encarrega de implementar as classes que concretamente vão realizar o serviço. Ora, cada fornecedor tem o seu driver específico, construído como uma classe JDBC. A chamada à função forName(classe) registra a classe nomeada na JVM corrente:

Citação:

Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").getInstance(); **o Sun JDBC-ODBC bridge**

Citação:

Class.forName("COM.ibm.db2.jdbc.net.DB2Driver").newInstance(); **para o IBM DB2**

podemos ainda utilizar o método **registerDriver** da classe **DriverManager** para isso:

Citação:

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver()); **para o Oracle 8i**

O comportamento do driver JDBC é uma incógnita. Depende sobretudo da arquitetura de acesso oferecida pelo banco de dados. Por exemplo, alguns bancos de dados vão exigir uma conexão com um cliente local. Outros utilizam arquivos de configuração. Para tratar com as diferenças entre os diversos produtos, DriverManager utiliza o padrão Factory Method para instanciar Connection.
Este padrão é utilizado por todas as interfaces de JDBC.
Por exemplo:

Código:

```
Connection con = DriverManager.getConnection("jdbc:odbc:meusCdsDb","conta","senha");

Statement stm = con.createStatement();

ResultSet rs = stm.executeQuery(alguma query SQL);
```

O problema é simplesmente este: não sabemos, em tempo de compilação, qual a estrutura de classe terá a classe do driver que DriverManager irá carregar. Isto é feito dinamicamente, no momento em que o driver é registrado com DriverManager. Usando o padrão FactoryMethod a necessidade de verificar a estrutura do driver estaticamente é transformada numa chamada padrão de função que pode ser realizada em tempo de execução. As classes que implementam concretamente essas interfaces no DB2, por exemplo, são DB2Connection, DB2Statement e DB2ResultSet e estão presentes no pacote db2java.zip.
Para os programadores migrados de outros padrões, este padrão é equivalente, no mundo

dos objetos, às funções callback, tão comuns nos SDKs do Windows. A diferença é que um callback registra uma função a ser chamada. O FactoryMethod cria um objeto.
Se você deseja saber mais sobre o padrão FactoryMethod, recomendo "Design Patterns: Principles
of Reusable Object Oriented Software" de Erich Gamma e outros.

Java Morrendo ??

Uma pergunta lançada numa lista de discussão nos ultimos dias me deixou realmente intrigado. Alguém afirmou que a Microsoft estava parando o desenvolvimento em Java (provavelmente devido à recente decisão judicial no caso Sun X Microsoft) e perguntou se por causa disso Java não estaria morrendo. A resposta (que continua sendo postada) foi um sonoro NÃO. Os argumentos variaram desde os motivos religiosos, tipo "**M$ Jamais**", até decisões bem pensadas tipo "...adivinhe quantas companhias estão desenvolvendo uma JVM para a plataforma Wintel neste momento ?". Eu tenho minha humilde opinião a respeito.
A aceitação e compromisso do mercado em relação a Java deve-se a diversos fatores:
1. Claramente há uma profunda mudança de visão sobre como construir sistemas em andamento no mercado. O modelo cliente-servidor apoiado na plataforma Wintel é dispendioso, difícil de manter e não se ajusta às necessidades do mundo Internet. Irá desaparecer ? Não, certamente que não, mas irá sendo posicionado num nicho ao invés de ser utilizado como uma solução universal. O modelo Internet veio para ficar e mandar.
2. Java permite economia em escala quando seu modelo de portabilidade é bem utilizado. Mesmo em ambientes com padrões públicos e conhecidos, como C++, portar código é um desafio. Em outros ambientes ainda mais populares e mais conhecidos, portar código varia entre o impossível e o pesadelo. Empresas como a PointBase, de bancos de dados pervasivos, estão, nas palavras de seu CEO: "...empregando mais engenheiros de portabilidade de código que a Oracle, sem gastar um tostão na folha de pagamentos com isso. Onde estão eles ? Na Sun, Na IBM, Na HP..." (JDJ dez/2002).
3. Players pesados estão jogando neste mercado. Nomes como IBM, Oracle, HP, Inprise, (que, ao que parece, agora vai com tudo para o mercado Linux), o Apache.org e a própria Sun, para citar alguns, deixam o desenvolvedor e o empresário bastante tranqüilos quanto a seu futuro. Linux ainda é incipiente em relação a Java, mas a oferta de JVMs da IBM parece estar surtindo efeito também nesta plataforma.
4. XML vem com um modelo de representação de informação que, pela primeira vez, oferece um meio a baixo custo de expressar estruturas de dados complexas com a certeza de que serão lidas onde quer que sejam necessárias. A portabilidade dos dados demanda portabilidade do código para sua interpretação, pois, como dito no item dois, manter portabilidade custa caro. Java tem uma proposta bastante interessante neste campo.
Portanto, Java já ultrapassou a fronteira de sustentação que separa produtos de plataformas. O máximo que pode acontecer agora, IMHO, é a linguagem se estabelecer num nicho e aí ficar para sempre. Lembram do velho e bom FORTRAN ? Passa muito bem, obrigado.

Dalton Milkvicz de Camargo
dalton@javafree.com.br
