Considerações Sobre um Sistema Interativo para Execução de Algoritmos Econométricos em Java

Carlos Duarte do Nascimento
(aluno do Bacharelado em Matemática
Aplicada e Computacional do IME/USP)

Objetivo

Este documento visa documentar e elaborar consdierações sobre os pontos discutidos nas reuniões ocorridas em 04/Abril/2007 e 12/Abril/2007 com o Prof. Dr. Cicely Moitinho Amaral, nas quais foi discutida a estrutura de um sistema que disponibilizaria, através de um browser, um conjunto de algoritmos econométricos, permitindo ao usuário conhecer os algoritmos e observar sua execução, tanto em exemplos prontos quanto em dados digitados no próprio sistema.

Muitas das idéias discutidas aqui têm em vista o trabalho já desenvolvido pelo Professor (EconoJava), bem como seus estudos na conversão de algoritmos codificados em FORTRAN para uso em ambientes baseados em Java. Entretanto, tais aspectos não foram explicitamente considerados, tendo em vista a natureza introdutória do documento.

Arquitetura

O sistema seria desenvolvido em Java, como uma *web application*, isto é, rodando em um *web container* como o Apache Tomcat ou o container embutido no JAVA EE SDK.

Para manter o projeto simples, pode-se usar simplesmente servlets para os processamentos e JSPs para a exibição dos resultados. Caso o projeto venha a crescer no futuro (a ponto de esta organização se tornar problemática), ele pode ser organizado sob o formato Model-View-Controller[1], usando um framework como Struts.

O ideal é que classes de apoio façam as tarefas unitárias das servlets (ex.: gerar HTML para campos de entrada, executar os algoritmos, desenhar gráficos), reservando-se a essas uma quantidade reduzida de código, que seja diretamente traduzida das necessidades da mesma[2].

É interessante que os componentes lógicos sejam refletidos como classes, indepdente do seu armazenamento em banco. Ex.: Algoritmo, Parametro/Parametros, Usuario, Resultado. Isto também ajuda a manter o código das servlets livre de detalhes de implementação, e facilita o reuso em diferentes interações (ex.: a execução do algoritmo pode ser usada tanto na exibição de um exemplo quanto na interação direta com o algoritmo).

Um mecanismo de persistência como Hibernate pode facilitar a operação com o banco de dados (e tornar a aplicação indepdente do mesmo).

Para permitir o estudo e alteração dos algoritmos existentes, bem como a inserção de novos algoritmos, as características de entrada do mesmo (ex.: parâmetros e seus tipos de dados) devem ser armazenadas em banco (e não codificadas diretamente no sistema).

Além disso, é preciso que o sistema possa compilar e executar os algoritmos diretamente no servidor, sem exigir uma re-instalação ou qualquer procedimento especial. Estas necessidades especiais são discutidas nos tópicos seguintes.

---

1 O padrão de projeto *Model-View-Controller* está catalogado no *Core J2EE Patterns* (vide bibliografia). Outra obra recomendada neste sentido é o *Design patterns: elements of reusable object-oriented software.*

2 Este padrão de projeto é conhecido como *View Helper*, e também consta no *Core J2EE Patterns.*

Armazenamento de Dados

O ideal seria que o projeto armazenasse os dados de cada algoritmo em um registro de uma tabela. Nesta constariam campos como:

- Identificador
- Nome
- Descrição (HTML) e outros materiais de apoio à explicação (ex.: Links)
- Código-java para execução

Os parâmetros de entrada seriam armazenados em uma segunda tabela, com os campos:

- Nome do parâmetro
- Identificador do algoritmo ao qual pertence
- Tipo (inteiro, real, matriz)
- Valor de exemplo (pode ficar aqui num primeiro momento)

Uma terceira tabela (não necessariamente na primeira fase do projeto) poderia armezenar as execuções já feitas por cada usuário, contendo:

- Identificador do usuário (possivelmente vindo de uma quarta tabela, a de usuários)
- Identificador do algoritmo
- Parâmetros utilizados (um campo texto, n campos ou uma tabela subordinada)

Execução dos Algoritmos

Sendo a idéia armazenar os algoritmos no banco de dados (permitindo a inserção dinâmica de novos algoritmos, e, numa fase futura, algoritmos customizados para cada usuário), é preciso definir um mecanismo de compilação e execução dos mesmos.

O Java 1.6 permite chamar o compilador Java através de API específica[3] (independente da plataforma onde se está executando).

Este sistema poderia ser usado (direcionando a saída para uma pasta visível no CLASSPATH). Aliando-o a uma regra para nomes de classes de nomes conveniente (ex.: formado parcialmente pelo identificador do algoritmo, prefixado por "Algoritmo") permite à aplicação compilar o algoritmo apenas se o mesmo já não tiver sido compilado.

Um teste simples é chamar a classe do algoritmo e tratar a ClassNotFoundException, como no exemplo mostrado na Listagem 1 (que supõe que os algoritmos descendem da classe Algoritmo, o que é uma idéia razoável).

3 Vide exemplo em *http://www.javalobby.org/java/forums/t44534.html*

```
// Aqui supomos que ja sabemos id_algoritmo e que o código do banco já foi
// descarregado para um arquivo temporario, prefixado e sufixado com
// a declaração básica da classe, e o nome do arquivo temporario (.java)
// está em arq_tmp

Algoritmo a = null;
while (a == null) {
  try {
    // Tenta criar uma nova instância do algoritmo
    a = (Algoritmo)Class.forName(
          "pacote.do.projeto.Algoritmo" + id_algoritmo
        ).newInstance();
  } catch (ClassNotFoundException e) {
    // Não foi compilado ainda, vamos compilar:
    JavaCompilerTool compilador = ToolProvider.defaultJavaCompiler();
    compilador.setOutputDirectory(
      new File("c:/pasta/que/esta/no/classpath/"));
    resultado = compilador.run(out, new File(arq_tmp));
    if (resultado) {
      // Compilação deu certo, vamos iterar o loop e dessa vez ele vai existir
      continue;
    } else {
      // Procedimento para tratar o erro (ex.: avisar o usuário) e decidir se
      // vai tentar novamente (continue) ou desistir (break)
    }
}
// Aqui eu já posso chamar o método padronizado, contido em a. Ex.:
a.calcula(lista_de_argumentos);
```

Listagem 1 – Compilação e chamada dinâmicas do alogritmo.

O código garante que a classe seja compilada apenas se ainda não o tiver sido. É preciso sofisticar um pouco para o caso de um algoritmo ser alterado (o programa precisa descarregar a classe, e para isso é necessário fazer uso de um ClassLoader customizado).

Caso não se use o Java 1.6, pode-se chamar manualmente o compilador (javac), mas este método é bem mais sensível à configuração do servidor.

Bibliografia Recomendada

Alur, Malks, Crupi: ***Core J2EE Patterns: Best Practices and Design Strategies, Second Edition****, Pearson Education, 2003.*

Gamma et al: ***Design patterns: elements of reusable object-oriented software****, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 1995*

Linwood, Minter: ***Pro Hibernate 3 (Expert's Voice)****, Apress, 2005*