ARTIGO:

Tirando o Maximo da Struts – Parte I

(ActionMapping e LookupDispatchAction)

Por: Paulo Alvim

Março/2004

Powerlogic
Tecnologia Orientada ao Cliente

(31) 3286-1691
www.powerlogic.com.br
plc@powerlogic.com.br

Índice

TIRANDO O MAXIMO DA STRUTS

(ActionMapping e LookupDispatchAction)
Nível: Intermediário

Introdução

Segundo os autores do livro "Professional Struts Applications", desenvolver uma *framework in-house* nos moldes da Struts custaria algo entre U$ 500.000,00 e U$ 1.000,000,00.

Pelo fato de ter sido encabeçada por Craig R. McClanahan, engenheiro da SUN responsável por importantes projetos J2EE como o Tomcat 5, e de já ter recebido contribuição de alguns dos maiores *experts* em MVC/J2EE da Indústria (cada um dos quais comandando altos salários em suas empresas), não se pode duvidar muito deste número...

Mais ainda, devido ao fato de a Struts ter-se tornado um padrão em seu segmento, soma-se uma impressionante comunidade mundial de colaboradores e usuários que lhe conferem estabilidade e robustez inestimáveis.

No entanto, mais que saber o "quanto podemos deixar de investir" por usar *frameworks open-source* como a Struts, desejamos descobrir "o quanto podemos obter de benefício" com seu uso apropriado.

Este é nosso objetivo neste artigo. Para tal, ele é voltado para desenvolvedores e arquitetos de software com experiência em Struts. Aos iniciantes, recomendamos que busquem antes por artigos introdutórios.

jCompany – Framework Open-Source de Última Milha

Para procurar atingir este objetivo, nos valemos de nossa intensa experiência na utilização da Struts durante os últimos dois anos, como um dos principais componentes do projeto **jCompany.**

O jCompany (mais informações em www.powerlogic.com.br), é uma *framework* de "última milha" que integra ainda dezenas de outros projetos *open-source* como Hibernate, JasperReports, Log4j, Eclipse/MyEclipse, CVS, Ant, Junit, JFreeChart, Quartz e Lucene, e traz implementadas as abordagens de extensão descritas neste artigo. Na verdade, costumamos batiza-la como "penúltima milha", já que sempre resta alguma generalidade a ser estentida por cada empresa (ver Figura 1).

Figura 1. Esquema Final de Arquitetura Open-Source, com Penúltima e Última Milhas.

Dentro de uma estratégia completa como o jCompany, a Struts desempenha um papel de **infra-estrutura de controle** da arquitetura MVC. Em outras palavras, a Struts é um excelente alicerce que resolve e nos abstrai de diversos **problemas de J2EE**! Mas, como veremos adiante, os maiores resultados que obtivemos para resolver **problemas de negócio** advieram das extensões criadas acima deste alicerce.

Nos próximos tópicos, iremos ilustrar algumas destas extensões, que resolvem, genericamente, problemas típicos de aplicações corporativas.

Evitando Subestimações Frequentes

Então, como tirar o máximo de benefícios da Struts? Primeiramente, precisamos dissipar os exageros de expectativas!

Subestimações provocadas por introduções apaixonadas ao tema muitas vezes levam o interessado à ansiedade de imaginar a Struts como um produto "*out-of-box*", pronto para ser usado como está. Dentre simplificações importantes, alguns *slogans* recorrentes são:

1. Com a Struts você está livre para se dedicar às suas regras de negócio!

2. Com a IDE X ou Y (ou *plug-ins*) você pode ganhar produtividade "gerando" a maior parte dos artefatos da Struts.

Na realidade, como sabem os mais experimentados, ambas as afirmações não representam a realidade! E o pior da subestimação não é somente levar a empresa a frustrações desnecessárias, mas muitas vezes sabotar a oportunidade que esta possui de realizar um investimento em estratégias de produtividade mais profícuas e duradouras.

Se todos estão satisfeitos com o uso da Struts sem extensões - perigo! Podem estar dando como satisfatório uma utilização medíocre da *framework*, que está longe de trazer seu melhor benefício.

Mas se todos superam a frustração e a empresa está disposta e encarar um "dever de casa" ou, como chamamos, um trabalho de "última milha", as recompensas subseqüentes serão exponencialmente maiores.

Desenvolvendo a "Última Milha"

Para um bom trabalho de "última milha" precisaremos:

1. Tomar importantes decisões preliminares acerca do uso da Struts.
2. Obter um conhecimento razoável de seus *extension-points* e estendê-los eficientemente.
3. Usar as boas e velhas práticas de OO!

Como vimos, não vamos explorar nada relacionado a "geradores de códigos" ou "assistentes" ou "drag-and-droppers": os ganhos de **rapidez** obtidos a partir destas abordagens existem, mas são realmente marginais quando comparados aos ganhos obtidos pela OO. Por isso, preferimos não chamá-los de **ganhos de produtividade**.

O foco inteligente deve estar mais em "**eliminar** a necessidade de código-fonte" do que em "**gerar** código-fonte **rapidamente**"!

Decisões Preliminares

No momento em que este artigo está sendo redigido, a Struts se encontra com versão oficial 1.1.0 (junho de 2003), e com versão beta 1.2.0, sendo que o "road map" já aponta planos tanto para a família Struts 1.x (1.3, 1.4, etc.) quanto para a Struts 2.0. Datas para as versões estáveis não são apresentadas, já que a prática é de que estas datas "aconteçam" naturalmente, na medida em que os *bugs* sejam aceitos como resolvidos pela comunidade.

Para, neste estágio, tirarmos o máximo da Struts, as 10 (dez) decisões iniciais mais importantes são:

1. Privilegiar o desenvolvimento declarativo em XML
2. Utilizar a versão 1.1 da Struts
3. Não criar classes Java para Form-Beans
4. Utilizar o Validator
5. Utilizar o Tiles
6. Utilizar o Tratamento de Exceção da Struts
7. Utilizar as tag-libs JSTL em lugar de "bean" e "logic" da Struts
8. Especializar LookupDispatchAction para classes de controle
9. Utilizar módulos
10. Não utilizar a Struts antes de especializar todos os seus Extension-Points

Vejamos uma análise destas decisões mais em detalhe:

1. **Privilegiar desenvolvimento declarativo em XML**. O padrão J2EE introduz o uso de alguns arquivos XML, em especial o "web.xml", que prestam um importante serviço ao desenvolvimento, permitindo que atributos de contexto, segurança, tratamento de erros, *filters*, *listeners* e outras facilidades sejam **declaradas (o quê)** ao invés de **programadas (como)**. Os ganhos são muitos: maior qualidade, documentação automática e representação de conhecimento em "campo neutro" (regras em XML facilitam uma futura migração, por estarem "desamarrados" de uma linguagem específica, de repositórios de geradores, CASE ou fornecedores específicos).

   A Struts evolui no uso de XML com arquivos como "struts-config.xml", "tiles-page.xml", "tiles-menu.xml", "validation.xml" e "validator-rules.xml", sendo que o principal deles, o "struts-config.xml", pode ser estendido pelo usuário para conter atributos próprios através da especialização do ActionMapping (como veremos a seguir).

   Privilegiar este tipo de abordagem é um dos primeiros grandes passos na obtenção de grandes benefícios no uso da Struts.

2. **Utilizar a versão 1.1**. Após a versão 1.0, os desenvolvedores responsáveis pela Struts descontinuaram alguns caminhos (muito embora os mantenham presentes, para compatibilidade), e muito dos diferenciais que veremos adiante somente são possíveis a partir da versão 1.1.

3. **Não criar classes Java para Form-Beans**. Desde a versão 1.1, estas classes Java são totalmente dispensáveis e substituíveis com méritos por declarações em XML. Uma grande economia!

4. **Utilizar o Validator**. Se não teremos classes Form-Bean, precisamos utilizar o Validator para eliminar também todo o código Java de consistência da entrada de dados, em prol de validação declarativa em XML.

   ”Minha aplicação já usa o Validator?” Se existe programação Java específica para qualquer validação de entrada de dados que não vá à camada de Modelo (exs: campo obrigatório, data válida, validação entre campos), os benefícios do Validator não estão sendo utilizados.

5. **Utilizar o Tiles**. Esta *framework* de Gestão de Layout oferece grandes oportunidades de ganhos de produtividade e qualidade de aplicações com foco na camada de Visualização, tanto que serão motivo de artigo futuro (“Tirando o Máximo do Tiles”).

6. **Utilizar o Tratamento de Exceção da Struts**. Conseguimos isso especializando o ExceptionHandler para garantir tratamento mínimo de exceções. Exceções podem ser difíceis de se tratar ou, o que é mais comum, podem simplesmente não ser tratadas pelo desenvolvedor, trazendo “ilusões” em produção. (Alguém aí conhece aquela aplicação que não apresenta erros para os usuários, mas cuja console não pára de exibir *stack trace*?)

7. **Utilizar JSTL em lugar de tag-libs “bean” e “logic” da Struts**. Com a liberação da versão 1.0 da JSTL, a equipe da Struts cuidou de evoluir as tag-libs da família “html” para funcionarem integradas com as tag-libs “c” e “fmt” da JSTL dispensando, respectivamente, as tag-libs “bean” e “logic”. As tag-libs JSTL são mais simples, poderosas e se tornarão inerentes aos *containers* na versão “servlet 2.4”.

8. **Utilizar descendentes de LookupDispatchAction**. Apesar de ser exemplo na maioria dos artigos introdutórios, não devemos especializar classes de controle diretamente de Action! Problemas como “rótulos de botão atrelados a nomes de métodos” ou “método *execute* abarrotado de *ifs*”, indicam especialização direta de Action, na maior parte das vezes indesejável. A especialização de *lookup* resolve este problema.

9. **Utilizar módulos**. Qualquer projeto corporativo - mesmo pequeno – deveria exigir, tipicamente, a criação de um ou mais módulos Struts genéricos, para disponibilização de forma elegante (em “.jar”, tipicamente), de ações (Actions) reutilizáveis por toda a empresa. O uso de módulos costuma ser citado como recurso para evitarmos arquivos XML imensos mas, na verdade, é praticamente compulsório para organizações que pretendam fomentar a reutilização de código (nossa meta!).

10. **Não utilizar a Struts antes de especializar todos os seus Extension-Points**. Os desenvolvedores da Struts prestaram um enorme serviço em prover uma *framework* de base, como vimos, mas bastante genérica. Sem o uso das especializações, somos incapazes de “tirar o máximo da Struts”. A Struts provê os pontos ideais para desenvolvermos este trabalho e seus autores recomendam que isso seja feito. No próximo tópico, veremos algumas práticas neste sentido.

Especializando a Struts

Nos grandes provedores de internet, as fibras óticas estão folgadas. Mas grande parte das pessoas continua com acessos lentos, baseados em Modem. A qualidade da **última milha -** o trecho final da conexão do backbone à residência dos usuários - é o gargalo decisivo para a velocidade de navegação.

No mundo Open-Source a última milha **é igualmente decisiva para a obtenção de resultados!** Para este trabalho, os projetistas da Struts reservaram os chamados pontos de extensão, sendo os principais:

1. ActionMapping
2. LookupDispatchAction
3. TilesRequestProcessor
4. Validation
5. ExceptionHandler
6. Tiles

Em seguida, vamos conceituar os dois primeiros pontos e dar alguns exemplos práticos implementados com sucesso no jCompany através de um roteiro passo a passo para uma extensão básica (tipo "hello world. Em artigos futuros, iremos explorar os demais *Extension-Points*.

Obs: nos exemplos de código, os "imports", "javadocs" e "loggers" foram suprimidos e boa parte simplificada, mas todos eles foram extraídos de exemplos de produção. As classes utilizadas podem ser importadas de "jars" da Struts ou dos projetos Commons.

1. ActionMapping

(Descendente de ActionConfig)
Guarda uma representação Java das informações de cada *action* declaradas em XML no arquivo de configuração Struts.

Por quê estender? Através da extensão desta classe, podemos criar propriedades específicas nos XMLs da Struts. É um dos principais instrumentos de extensão para ganhos de produtividade. Permite que implementemos lógicas declarativas que eliminam redundâncias de códigos procedimentais.

Exemplos de extensão: Criação de lógicas de controle genéricas em classes "Action" que se baseiem em atributos XML declarados pelo desenvolvedor no "struts-config.xml". Os exemplos são inúmeros, podendo ir desde chamada genérica de uma página de ajuda declarada no XML, até estereótipos de lógicas comerciais complexas (CRUD, Máster-Detail, Reports, etc.), que dispensem programação.

No nosso tutorial, vamos estender o "action-mapping" para permitir aos desenvolvedores declararem um atributo no qual indicará qual VO-Value Object (ou DTO-Data Transfer Object, para alguns) será criado e populado genericamente a partir do Form-Bean, dispensando essas lógicas "burras" mais conhecidas como "de-para". (Obs: Para atingir este objetivo, o presente roteiro precisa contar com a especialização adicional do LookupDispatchAction, que será vista a seguir.)

Passo a passo:

1.1. Criar uma classe que estenda ActionMapping e implemente o atributo "valueObject", com métodos "getter" e "setter". No nosso exemplo, esta classe se chamará PlcActionMapping.

```
public class PlcActionMapping  extends ActionMapping {

    String valueObject = "";

    public String getValueObject(){
                return valueObject;
        }

     public void setValueObject(String novo){
                valueObject=novo;
        }
}
```

1.2. No dia a dia, o desenvolvedor deve indicar, na declaração de "action-mapping", no elemento "action" do arquivo de configuração da Struts, que deseja utilizar como classe de mapeamento a extensão de "última milha".

```
<action path="/cliente"
        name="clienteForm"
        className="com.powerlogic.jcompany.controle.PlcActionMapping"
...
```

1.3. Em seguida, deve declarar, na seção "set-property" do "action-mapping", a classe de VO que receberá os dados do Form-Bean.

```
<action path="/cliente"
      name="clienteForm"
     className="com.powerlogic.jcompany.controle.PlcActionMapping"
     … >
<set-property property="valueObject"
                          value="com.empresa.app.vo.ClienteVO">
</set-property>
<forward name="mesmaPagina" path="def.cliente"/>
     …
</action>
```

Na especialização do controle, veremos como complementar esta extensão.

2. LookupDispatchAction

(Descendente de Action / DispatchAction)
As classes que contém lógica de controle, descendentes de Action, são as mais comumente utilizadas pela comunidade Struts. Atualmente, a melhor opção é criar uma classe base de controle, herdando da classe de controle mais especializada da Struts, que é o LookupDispatchAction, pelas vantagens citadas nas decisões preliminares.

Por quê estender? Para generalizar lógicas de controle para funções de navegação padrão, chamada da validação, exibição de mensagens, segurança de interface com o usuário e comunicação com a camada Modelo. Importante:

Estas classes não são feitas para conter lógicas do negócio!

Exemplos de extensão: Diversas lógicas genéricas de controle, aliadas com a extensão do ActionMapping, permitindo o uso de desenvolvimento declarativo em XML. (No caso do jCompany, todas as lógicas padrões mais comuns em sistemas comerciais, como CRUD, QBE and Search, Clone, Print, Help, etc..., com qualidade de produção, foram generalizados na classe PlcBaseAction, dispensando a criação de classes Action específicas para grande parte dos casos.)

Passo a passo: Vamos utilizar o atributo especifico criado no PlcActionMapping para realizar uma lógica que, genericamente, copia os dados do Form-Bean para o VO declarado para ela.

2.1. Criar uma classe que estenda LookupDisptachAction e implemente o método "grava", disparado a partir de um botão padrão de "submit". Este botão deverá ser utilizado em todas as aplicações. O método "getKeyMethodMap" isola o "label" do botão de *submit* do nome do método, inclusive viabilizando a internacionalização destes botões, segundo o padrão da Struts. O importante, aqui, é entender que o método "grava" será chamado para tratar submissões realizadas com o botão que possua, como *label*, a chave "jcompany.evt.gravar".

```
public class PlcBaseAction extends LookupDispatchAction {

 protected Map getKeyMethodMap() {
   Map map = new HashMap();
   map.put("jcompany.evt.gravar","grava");
   return map;
  }

  public ActionForward grava (ActionMapping mapping,
    ActionForm form, HttpServletRequest request, HttpServletResponse response)
          throws Exception {

   // Código deste método será exibido adiante

  }
}
```

2.2. Criar a chave no arquivo de "properties". Exemplo para arquivo "ApplicationResource_pt_BR.properties"

```
...
jcompany.evt.gravar=Gravar
...
```

2.3. Criar uma ".jsp" genérica (preferencialmente utilizada em um layout Tiles que contenha o botão de gravar.

```
...
<html:submit styleId="botao_menu" property="evento">
        <bean:message key="jcompany.evt.gravar"/>
</html:submit>
...
```

2.4. Implementar a lógica genérica do método grava. A lógica abaixo elimina uma quantidade excepcional de código Java, de menor importância para o negócio, mas ainda atendendo a exigências de *Design Patterns* e da arquitetura MVC:

```
...
  public ActionForward grava (ActionMapping mapping,
    ActionForm form, HttpServletRequest request, HttpServletResponse response)
          throws Exception {

     // 1
     DynaValidatorActionForm f = (DynaValidatorActionForm) form;
     PlcActionMapping plcMapping = (PlcActionMapping) mapping;

     // 2
     ActionErrors errors = f.validate(mapping,request);

     if (errors.isEmpty()) {

        // 3
        Object voGenerico
                    = Class.forName(plcMapping.getValueObject()).newInstance();

        // 4
        BeanUtils.copyProperties(voGenerico,f);

         // 5. Pega interface Façade registrada em escopo de aplicação
         IPlcFacade plc = (IPlcFacade) request.getSession().getServletContext().
                      getAttribute(PlcConstantes.INTERFACE_PERSISTENCIA_KEY);

         // 6. Grava
         plc.grava(voGenerico);

      } else
          request.setAttribute(Globals.ERROR_KEY,errors);

  }
...
```

1. Nestas linhas, fazemos o casting do form para o nosso form dinâmico e da classe de mapeamento padrão para a nossa classe, criada no item anterior.

2. Nestas linhas, disparamos a validação de entrada declarativa, que irá checar as regras informadas no XML para o Form-Bean dinâmico e, somente em caso positivo, prosseguir na gravação.

3. Nesta linha, instanciamos um Value Object baseado no nome da classe declarado pelo desenvolvedor no XML. O valor declarado é recuperado com o comando "plcMapping.getValueObject()".

4. Aqui utilizamos o utilitário disponível nos "Commons" do Jakarta para cópia por introspecção do DynaBean (Bean utilizado internamente pela Struts para armazenar os Form-Beans dinâmicos definidos no XML) para o VO (Bean criado pelo desenvolvedor). A cópia de todas as propriedades é feita baseada nos seus nomes.

5 e 6. Nas linhas adicionais, a *Interface* com a camada de Modelo é capturada e o método que irá persistir o VO chamado.

Conclusão

Esperamos ter conseguido dar uma percepção sobre o ganho potencial de produtividade advindo de abordagens OO sobre a Struts. Em um artigo futuro, iremos ver a especialização do *TilesRequestProcessor*, do *Validation* e do *ExceptionHandler...*

E tudo isso ficará ainda mais interessante quando explorarmos outros universos, em "Tirando o Máximo do Tiles" e "Tirando o Máximo da Hibernate". Até lá!

Bibliografia Rápida

- Struts Fast Track by Vic Cekvenich
- Programming Jakarta Struts by Chuck Cavaness
- Jakarta Pitfalls by Bill Diudney, Jonathan Lehr
- Struts in Action by Ted Husted, Cedric Dumoulin
- Struts Survival Guide by Skikanth Shenoy, Nithin Mallya
- Professional Struts Applications by John Carnell, Jeff Linwood
- Patterns of Enterprise Application Architecture by Martin Fowler
- **http://jakarta.apache.org/struts**

Autor

- Paulo Alvim é Diretor de Tecnologia da Powerlogic Consultoria e Sistemas S.A., e responsável pelo projeto jCompany. Pode ser contatado pelo email **alvim@powerlogic.com.br**