Desenvolvimento Estruturado de Aplicações Web com Cocoon

Márcio Ferreira, João Pias, Célio Abreu, Rui Alberto

PT inovação

{jpias,celio,rui-l-goncalves}@ptinovacao.pt

Telbit

mferreira@telbit.pt

12 de Janeiro de 2004

Resumo

Pretende-se com este documento apresentar os principais conceitos utilizados na framework de publicação Apache Cocoon. Não é pretendido explanar de forma detalhada a framework, mas sim apresentar as metodologias subjacentes à tecnologia.

1 Introdução

Inicialmente a WWW[1] foi criada como um meio de publicação de conteúdos estáticos. No entanto o conteúdo que é devolvido por um servidor aplicacional não necessita de ser obrigatoriamente estático, poderá ser o resultado da execução de um programa produzindo respostas de forma dinâmica.

O Apache Cocoon é uma framework de publicação, desenvolvida na linguagem Java baseada na Servelt API, podendo ser instalado em qualquer servlet container. Eleva a utilizaçao das tecnologias XML[2] e XSLT[3] para um novo nível no desenvolvimento de aplicações Web. As aplicações desenvolvidas nesta framework são de grande performance e escalabilidade, pois

[1] World Wide Web
[2] eXtended Markup Language
[3] eXtensible Stylesheet Language

baseiam-se no encadeamento de eventos SAX[4] e em mecanismos de cache, diminuindo assim os tempos de resposta. Os custos de desenvolvimento e manutenção são também mais reduzidos por existir uma separação clara entre lógica e apresentação.

Esta framework é baseada em componentes, sendo por isso bastante modular. Como esses componentes geram e/ou consomem eventos SAX, podem ser encadeados numa sequência para produzir os resultados pretendidos. Uma cadeia de componentes é designada por **pipeline**. O conceito de pipeline é semelhante ao conceito já existente nos sistemas unix, mas nos pipelines do cocoon são passados eventos SAX. Existe um conjunto de componentes genéricos prontos a ser utilizados num pipeline, no entanto o programador é livre de criar os seus próprios componentes para tarefas mais especificas.

2 Sitemap

O sitemap é o ficheiro de configuraço principal do cocoon, pois é onde se define o fluxo de uma determinada aplicação. Conforme o exemplo que se segue, o sitemap está dividido em duas secções, componentes e pipelines, dos quais passaremos a falar.

```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
  <map:components>
    <map:component ...
    ...
  </map:components>

  <map:pipelines>
    <map:pipeline ...
    ...
  </map:pipelines>
</map:sitemap>
```

2.1 Componentes

Um componente é um bloco de software que pode ser combinado com outros de forma a realizar a tarefa que se pretende. Na secção dos componentes são declarados todos os componentes necessários para a construção dos pipelines.

[4]Simple Api for XML

São geridos pelo Avalon Exclibur Component Manager, o qual é responsável pela gestão das pools, configuração e parametrização dos mesmos.

2.1.1 Generators

Este componente aceita qualquer coisa à entrada e gera uma sequência válida de eventos SAX, que são encaminhados para o próximo componente existente no pipeline. Este tipo de componente é utilizado na inicialização de um pipeline de forma isolada ou conjuntamente com outros generators para produzir um documento que é o resultado da agregação de cada um dos documentos produzidos pelos generators isoladamente.

2.1.2 Transformers

Componente opcional num pipeline que pode ser usado depois de um generator ou um transformer. Recebe como input eventos SAX e lança novos eventos SAX para o próximo componente no pipeline.

Um transformer não precisa de processar o documento inteiro, apenas precisa de focar a informação necessária á tarefa para a qual foi desenvolvido. Se o documento contém tags especificas para o transformer corrente, avalia essa informação e age para obter conteúdos adicionais ou para manipular o documento de outras formas. Por exemplo se o generator receber o seguinte documento XML:

```
<document>
 <sql:execute-query xmlns:sql="http://apache.org/
                                    cocoon/SQL/2.0">
   <sql:use-connection>
     conf.conn@cocoon.xconf
   </sql:use-connection>
   <sql:query>
     select id, name from users_table
   <sql:query>
 </sql:execute-query>
</document>
```

que especifica um statement SQL para uma base de dados, usando o SQL Tranformer, extrai a query SQL, executa-a e insere o resultado no documento. É possível que um componente gere comandos para o componente que se segue no pipeline. Por exemplo, o SQL Transformer fez uma query a base de dados e o seu resultado pode ser usado para formar um comando para o

próximo transformer na cadeia, como por exemplo o mail transformer, para enviar um e-mail ao administrador se algo correu mal.

2.1.3 Serializers

Todos os pipelines tem obrigatoriamente que terminar com um serializer. Recebe como input eventos SAX e serializa-os num determinado formato, dependendo do tipo de serializer.

2.1.4 Matchers

A função de um matcher é avaliar se o URI[5] que está a ser pedido deverá ser o resultado da execução do fluxo que matcher encapsula. Um matcher age como uma fechadura para um pipeline, só deixa passar se a chave o abrir. O sitemap é percorrido usando o URI que foi pedido como chave, essa chave é testada em cada um dos matchers até que seja encontrado um match para a respectiva chave. Uma vez que seja encontrado, é executado o respectivo pipeline. Por essa razão é importante a ordem pela qual se colocam os pipelines no sitemap. Matchers mais específicos devem ser colocados no inicio do sitemap. Se o URI não fizer match em qualquer pipeline é retornado um erro para o cliente.

Exemplo da utilização de um matcher:

```
<map:match pattern="today-*/*.xml"/>
  ....
</map:match>
```

2.1.5 Selectors

Tal como os matchers os selectors são componentes do sitemap que podem ser usados para determinar o fluxo atravéz do sitemap. Essas decisões poderão ser tomados com base em informações quer do cliente, quer do sistema.

O exemplo mais comum de um selector é o browser selector, que permite optar por fluxos distintos dependendo do tipo de cliente.

```
<map:generate src="calls.xml">
<map:select type="browser">
  <map:when test="explorer">
    <map:transform src="">
    <map:serialize/>
```

[5]Uniform Resource Identifier

```
    </map:when>
    <map:when test="netscape">
      <map:transform src=""/>
      <map:serialize/>
    </map:when>
    <map:otherwise>
      <map:transform src=""/>
      <map:serialize/>
    <map:otherwise>
  </map:select>
```

Note-se que o ficheiro lido não é dependente do browser, é o mesmo para todos os clientes. Isto permite uma verdadeira separação entre informação e apresentação, dando ao sitemap a mesma flexibilidade que temos nas linguagens de programação estruturada.

2.1.6 Actions

Os componentes até agora apresentados partilham uma funcionalidade comum, influenciam o resultado do pedido. No entanto quando se está a desenvolver uma aplicação precisam-se de efectuar tarefas que não influenciam directamente o documento, como por exemplo se quiseremos construir uma loja virtual com o Cocoon vamos necessitar de adicionar um novo utilizador á base de dados ou adicionar items para venda, é aqui que entram os actions. Vários actions são disponibilizados pelo cocoon para acesso a informação dos pedidos, como por exemplo os cookies, para comunicação com bases de dados e para autorizar utilizadores num portal. Um action pode também controlar o fluxo e fornecer informação a outro pipeline.

```
<map:act type="session-isvalid">
  <map:generate type="file" src="file.xml"/>
  <map:transform src="rules.xsl" type="xslt"/>
  <map:serialize type="html"/>
</map:act>
<map:read src="timeout.html"/>
```

O exemplo anterior define um action chamado session-isvalid que dá indicação do estado da sessão. Caso a sessão seja válida, então é retornado o HTML correspondente a transformação do documento file.xml com a stylesheet rules.xsl, senão será retornado o documento timeout.html.

2.1.7 Readers

Já se falou de componentes para processar XML ou para controlar o fluxo de processamento, no entanto as aplicações necessitam frequentemente de documentos que não são XML-based, como imagens, filmes, javascript. Estes documentos devem ser servidos para o cliente tal como são, sem qualquer tipo de manipulação. Os readers são usados para estes casos, em que o documento já se encontra no formato pretendido, apenas lê a fonte e passa a informação directamente á aplicação. Pode ser visto como a combinação de um generator especial e um serializer.

Exemplo típico da utilização de um reader:

```
<map:match pattern="*/*.gif">
  <map:read src="{1}/{2}.gif"/>
</map:match>
```

2.2 Pipelines

É considerado um pipeline uma sequência de componentes entre os quais fluem eventos SAX. Exemplo de definição de um pipeline num sitemap:

```
<map:pipelines>
  <map:pipeline>
    <map:match pattern="*.csv">
      <map:read type="xsp"
                src="generators/xsp/{1}.xsp" mime-type="text/csv"/>
    </map:match>

    <map:match pattern="*.xsp">
      <map:generate src="generators/xsp/{1}.xsp" type="xsp"/>
      <map:serialize/>
    </map:match>

    <map:match pattern="*.html">
      <map:read src="static/html/{1}.html" mime-type="text/html"/>
    </map:match>
  </map:pipeline>

  ...
</map:pipelines>
```

Um sitemap é composto por um conjunto de pipelines, cada pipeline contém um conjunto de matchers e cada matcher encapsula um fluxo de eventos.

2.2.1 Error handling

Durante o processamento de um pipeline poderão ocorrer execepções que impossibilitem a execução correcta do pedido. O mecanismo de error handling oferecido pelo cocoon permite que seja executado um fluxo distinto dependendo do tipo de erro.

```
<map:handle-errors>
  <map:select type="exception">
    <map:when test="connection-refused">
      <map:read src="connection_refused.html"
                mime-type="text/html"/>
    </map:when>
  </map:select>
</map:handle-errors>
```

2.2.2 Debugging facilities

No caso de o output retornado não ser o que se esperava o que fazer? O cocoon disponibiliza mecanismos que permitem efectuar o debug de forma simples.

- Usando o LogTransformer

```
...
  <map:trsnform type="sql"/>
  <map:transform type="log">
    <map:parameter name="logfile" value="logfile.log"/>
    <map:parameter name="append" value="no"/>
  </map:transform>
...
```

Neste exemplo o output do sql transformer é colocado no ficheiro definido no parâmetro logfile, o parâmetro append define se deve ser criado um novo ficheiro ou se deverá ser concatenado ao existente. Se não for definido nenhum parâmetro o output será enviado para o standard output.

- Definindo Views

  Uma view é um pipeline alternativo para um documento, começa como o original mas termina de modo diferente, ou seja, não é definido um generator mas podem-se definir transformers e um serializer. O atributo from-position permite definir o inicio do novo pipeline.

```
<map:view name="debug"
          from-label="debug">
  <map:serialize type="xml"/>
</map:view>
```

  Para que seja executada uma determinada view o utilizador deve efectuar o request passando na query string do URL cocoon-view=debug.

2.3 Mounting sitemaps

No desenvolvimento de uma aplicação podem estar envolvidas várias pessoas, executando tarefas distintas o que torna a gestão do sitemap complicada. Para facilitar a edição e manutenção do sitemap o Cocoon disponibiliza o conceito de subsitemaps. Um subsitemap é igual a um sitemap normal mas é referenciado no sitemap principal.

```
<map:match pattern="faq/*">
 <map:mount check-reload="yes" src="faq/sitemap.xmap"
   reload-mathod="synchron"/>
</map:match>
```

O atributo src define a localização do sitemap, o check-reload define se as alterações devem ser reflectidas e o reload-method especifica se a regenaração do sitemap deve ser sincrona ou assincrona. Os componentes declarados num sitemap são herdados pelos subsitemaps.

3 Conclusão

Com o Cocoon é possível construir aplicações sem que haja preocupações com o problema do crescimento. Pelo facto de o cocoon permitir um desenvolvimento bastante modular, a adiçao de novas funcionalidades apenas implica a adição de novos pipelines, quer seja num sitemap existente ou num novo sitemap. Esta modularidade e separação de funcionalidades permite

que o processo de desenvolvimento seja efectuado de forma paralela entre as equipas de desenvolvimento, sem que posteriormente venham a existir problemas de integração.

Os mecanismos de cache existentes no cocoon permitem aumentar o desempenho das aplicações por ele suportadas.

Devido a diversidade de componentes oferecidos pela framework, dificilmente será necessário recorrer ao desenvolvimento de componentes específicos, no entanto essa possibilidade é também deixada em aberto para casos mais particulares.

Para além de todas as vantagens já enumeradas, o cocoon herda todas as vantagens das tecnologias que lhe estão subjacentes.

4 Referêcias

[MC02] Matthew Langham and Carsten Ziegeler. *Cocoon: Building XML Applications*, New Riders, Jun 2002.

[HM01] Elliotte Rusty Harold and W. Scott Means. *XML in a Nutshell*, O'Reilly, Janeiro 2001.

[Wl01] Heather Williamson. *XML: the complete reference*, Osborne/McGraw-Hill, 2001.

[CA03] The Apache Cocoon Project. `http://cocoon.apache.org/2.1/`, 1999-2003

[AA03] The Apache Avalon Project. `http://avalon.apache.org/`, 1997-2003