

Zope Page Templates

Tutorial

© Marcello Peixoto Bax

Escola de Ciência da Informação – Universidade Federal de Minas Gerais (UFMG).

Abril de 2003.

O texto foi escrito a partir do trabalho de Thierry MICHEL.

SUMÁRIO

1	INTRODUÇÃO.....	3
2	CRIANDO UMA PÁGINA ZPT.....	5
3	EXPRESSÕES SIMPLES	6
4	INSERÇÃO DE TEXTO.....	7
5	ESTRUTURAS REPETITIVAS.....	7
6	ELEMENTOS CONDICIONAIS.....	8
7	DEFINIÇÃO DE VARIÁVEIS	9
8	MUDANÇA DE ATRIBUTOS	10
9	MISTURA DE ATRIBUTOS E SEQÜÊNCIA DE AVALIAÇÃO DAS DECLARAÇÕES.....	11
10	DECLARAÇÕES COM VÁRIAS PARTES	12
11	AS CADEIAS DE CARACTERES (STRINGS).....	12
12	EXPRESSÃO NOCALL	12
13	OUTRAS VARIÁVEIS INTERNAS	13
14	CAMINHOS (<i>PATHS</i>) ALTERNATIVOS	14
15	ELEMENTOS SIMPLES	14
16	UTILIZAÇÃO DA PALAVRA RESERVADA ESTRUTURA.....	15

1 Introdução

Zope Page Templates (ou linguagem de *templates Zope*) é um instrumento do servidor de aplicação Zope para a geração de páginas web dinâmicas. ZPT facilita a colaboração entre programadores e designers web aumentando a produtividade de páginas web dinâmicas que constituem a interface de qualquer aplicação Zope. Zope é um servidor de aplicações Web de código aberto, distribuído segundo a licença GPL. Nestes documentos descreve-se a ZPT para o atendimento de especificidades demandadas pela gestão de conteúdos.

Os designers web podem usar ZPT para a criação e o acompanhamento/manutenção de páginas dinâmicas, sem que para isso precisem abandonar seus instrumentos habituais de trabalho, e preservando, ao mesmo tempo, o esforço de programação feito para incluir as páginas numa aplicação.

O objetivo é proporcionar um fluxo de tarefas natural, sem que uma equipe (designers) atrapalhe o trabalho da outra (programadores). Um designer web utilizará um editor HTML WYSIWYG para criar o *template*, em seguida um programador poderá editá-lo para que este possa comunicar com a aplicação, introduzindo os comandos ZPT necessários.

Seguindo as especificações do projeto ou eventuais alterações, mais tarde o designer web poderá carregar o *template* no seu editor para fazer modificações sobre a sua estrutura e sobre a sua aparência. Tomando cuidado para preservar as mudanças efetuadas pelo programador, ele não correrá o risco de desorganizar a aplicação.

ZPT visa estes objetivos adotando 3 princípios:

1. Integração fácil com os editores HTML.
2. Apesar dos códigos de programa, a página aparece no editor WYSIWYG exatamente como apareceria em um navegador.
3. Manter o código mais complexo (regras do negócio) fora dos *templates*.

Uma página ZPT é um modelo, ou molde (do termo da língua inglesa *template*) para outras páginas que serão geradas a partir dela. Mais especificamente uma página ZPT é uma página HTML válida. Como o HTML é altamente estruturado, e os editores WYSIWYG preservam esta estrutura, existem limites muito estritos para que os programadores alterem uma página sem desrespeitar o primeiro princípio acima.

Para quem são feitas as páginas ZPT?

As páginas *templates* foram especialmente feitas para os programadores e designers web que têm a necessidade de trabalhar juntos para a criação de páginas web dinâmicas. Um programador que edita suas próprias páginas web com um editor de texto, pode não ter

necessidade de utilizar ZPT. No entanto, páginas ZPT podem ser mais simples de utilizar e mais facilmente compreensíveis do que páginas DTML.

Por que ter definido mais uma nova linguagem?

Existe um grande número de linguagens para a criação de *templates*, das quais algumas são bastante populares, como por exemplo, ASP, JSP e PHP. Por que inventar outra?

Em primeiro lugar, nenhuma destas linguagens foi criada para designers web, ou pensando neles. Uma vez que a página foi convertida em um *template* contendo scripts que geram conteúdos dinamicamente, a página não é mais uma página HTML válida, tornando-a de difícil manipulação fora da aplicação, por um editor HTML WYSIWYG.

Ou seja, nenhuma delas respeita o primeiro e segundo princípios do ZPT. Os programadores não deveriam ter de apropriar-se à "força" do conhecimento de designers web para alterar código HTML em aplicação; cada profissional deve dominar sua tarefa, especializando-se em design ou em programação.

XMLC, que faz parte do projeto Enhydra, compartilha esse ponto de vista, mas obriga os programadores a escrever uma grande quantidade de código Java na montagem do *template*. Este documento não trata desta ferramenta. O projeto Enhydra deixou de ser um projeto *Open Source*, após desentendimentos com a SUN Microsystems sobre o licenciamento da plataforma J2EE.

Aplicação dos princípios

ZPT tem uma linguagem denominada TAL (*Template Attribute Language*) que consiste de uma série de atributos especiais adicionados aos tags HTML. Por exemplo, uma página dinâmica ZPT poderia assemelhar-se à isto:

```
<title tal:content="here/title">Page Title</title>
```

O atributo `tal:content` é um enunciado de TAL. Perceba a presença de um espaço de nomes¹ XML (a parte `tal:`), a maior parte dos editores HTML não dá atenção, não se incomoda, com a parte que não é compreensível por eles e não suprime os atributos que considera estranhos. Dessa forma a estrutura ou a aparência da página não será alterada quando interpretada por um editor HTML ou um navegador Internet. As páginas TAL são interpretadas pelo servidor Zope, uma vez demandadas pelo navegador.

Em TAL, o nome `content` indica que o conteúdo do tag HTML `title` será atualizado, e `here/title` é uma expressão da qual o resultado será o texto a inserir no corpo do tag `title`.

¹ Um espaço de nomes XML é identificado por uma URI, e define os elementos válidos naquele escopo.

Este exemplo demonstra também o respeito ao segundo princípio: quando você vê o *template* num editor HTML, o texto do tag `title` ocupa o lugar do texto dinâmico, que será gerado quando a página for interpretada pelo Zope. O *template* pode então ser considerado como um exemplo fiel de representação dos documentos como aparecerão mais tarde no navegador, o que facilita bastante o trabalho dos designers.

Existem comandos TAL que substituem a totalidade do tag (incluindo suas marcas e seu conteúdo/contexto), apenas o seu conteúdo, ou simplesmente apenas seus atributos. Com comandos TAL, você pode fazer repetir várias vezes um tag ou retirá-lo inteiramente.

Todas estas características, juntas, permitem definir a estrutura dos documentos e gerar seu conteúdo dinamicamente. Porém, apesar de relativamente potente, utilizando TAL não se pode criar sub-rotinas ou classes, escrever laços (*loops*) complicados ou testes de múltipla escolha, ou realizar algoritmos complexos.

A linguagem TAL não é deliberadamente tão potente e de uso universal como uma linguagem de programação padrão. Com efeito, ela foi feita para ser utilizada dentro de uma arquitetura (como Zope, por exemplo) na qual outros objetos irão tomar para si a responsabilidade de gerar a lógica de negócio da aplicação, tarefas independentes da página com a diagramação visual HTML. Em geral os códigos de uma aplicação Zope são escritos na linguagem Python.

Por exemplo, TAL seria muito útil para a geração de uma fatura, criando uma linha para cada elemento, e inserindo a descrição, a quantidade, o preço, etc. Embora ela não será utilizada para criar o registro da fatura no interior da base de dados ou para interagir com um sistema de pagamento, por exemplo.

2 Criando uma Página ZPT

Aquelas pessoas que já trabalham com a criação de páginas web hospedadas em servidores Zope, utilizam provavelmente FTP ou WebDAV ao invés da *Zope Management Interface* (ZMI). Para os exemplos simples deste documento, é mais conveniente utilizar ZMI. Consulte seu administrador Zope para instruções sobre como aceder à ZMI.

Administradores ou programadores provavelmente já utilizam ZMI, pelo menos ocasionalmente. Pode-se também utilizar emacs, cadaver <http://www.webdav.org/cadaver> ou algum outro editor cliente. Confira o Guia de Administração de Zope para poder utilizar Zope com estes diferentes clientes.

Para executar os exemplos apresentados aqui, utilize seu navegador Internet para abrir uma sessão em ZMI. Escolha um diretório (*Folder*) qualquer, o `root`, por exemplo, e escolha "*Page Template*" da lista de escolhas dos componentes de Zope. Escreva `simple_page` nos campo `id`, e clique no botão *Add and Edit*.

A página de edição principal mostra então a nova *Página Template*. O título não está definido, o tipo do conteúdo é `text/HTML`, e o *template* criado por default está em edição. Vamos criar uma página dinâmica muito simples. Para isso basta inserir, por exemplo, as palavras `Simple Page` no campo `Title`. Em seguida, edite o conteúdo do *template* para obter isto:

```
This is <b tal:replace="template/title">the Title</b>.
```

Clique no botão *Save Changes*. A página de edição deveria emitir uma mensagem de confirmação, assinalando que as modificações foram registradas com sucesso. Se o texto contém na sua rubrica, `<- Page Template Diagnostics`, você deve verificar se a sintaxe do texto do exemplo acima está correta, em caso de erro corrigir e em seguida salvar novamente as alterações. A mensagem de erro desaparece automaticamente uma vez corrigido o erro.

Clique no link *Test*. O navegador deve mostrar: `This is Simple Page`.

Volte à página precedente (Botão *back* do navegador) e clique no link *Browse HTML Source* debaixo do campo *Content-Type*. Você deveria então ver `This is the Title`. Volte outra vez à página precedente, você está agora pronto para editar outros exemplos.

3 Expressões simples

O texto `template/title`, na página *template* simples da seção anterior, é uma expressão de caminho (*path expression*). É a expressão mais comumente utilizada entre os tipos definidos por *TAL Expression Syntax* (TALES). Ela procura o atributo `title` do *template*.

Eis outras expressões gerais:

1. `request/URL`: o URL do pedido web corrente.
2. `user/getUserName`: o nome de ligação do utilizador autenticado.
3. `container/objectIds`: a lista dos ids dos objetos contidos no mesmo diretório que a *Página Template*.

Cada *path* começa com o nome de uma variável. Se a variável contém o valor que desejado, para-se lá. Caso contrário, deve-se acrescentar `/` e o nome de outro objeto ou de um atributo/método. Você poderia ter necessidade de navegar através de vários objetos antes de encontrar o valor que procura.

Há um pequeno conjunto de variáveis básicas, como `request` e `user`, que serão descritas mais adiante no texto. Pode-se também definir variáveis próprias.

4 Inserção de texto

O *template* `simple_page`, utilizou o atributo `tal:replace` no tag HTML `Bold`. Quando do teste, a totalidade do tag foi substituída pelo título do *template*. Ao consultar o código fonte, você verificou que o texto do *template* estava em negrito. Utilizou o tag `Bold` para destacar a diferença. Para inserir texto dinâmico dentro de outros textos, você utilizará em geral a expressão `tal:replace`. Acrescente as linhas seguintes ao exemplo:

```
<br>
The URL is <span tal:replace="request/URL">URL</span>.
```

O tag `span` é estrutural e não visual, por conseguinte vamos obter: `The URL is URL`, olhando o a parti de um editor HTML ou em um navegador Internet. Atenção neste caso para não destruir o `span` ou inserir tags de formatação como `` ou `` dado que eles também serão substituídos.

Se quer inserir do texto dentro de um tag, mas deixar o tag ele mesmo único, deve utilizar `tal:content`. Para definir o título do vosso exemplo como o atributo `title` da página *template*, acrescentam as linhas seguintes acima do texto.

```
<head>
<title tal:content="template/title">The Title</title>
</head>
```

Abre-se a página clicando em *Test* numa outra janela, o título da janela estará `Simple Page`.

5 Estruturas repetitivas

Agora vamos acrescentar à nossa página uma lista de objetos que se encontram no mesmo diretório. Vamos criar uma tabela cujas linhas serão numeradas em uma coluna, além das colunas para `id`, `metatype` e `title`.

Acrescente esta linha à parte inferior do exemplo:

```
<table border="1" width="100%">
<tr>
<th>#</th><th>Id</th><th>Meta-Type</th><th>Title</th>
</tr>
<tr tal:repeat="item container/objectValues">
<td tal:content="repeat/item/number">#</td>
<td tal:content="item/id">Id</td>
<td tal:content="item/meta_type">Meta-Type</td>
<td tal:content="item/title">Title</td>
```

```
</tr>  
</table>
```

O atributo `tal:repeat` aplicada sobre as linhas significa: repita esta linha para cada objeto do *container*. O atributo `repeat` atribui os objetos da lista, um à um, à variável `item`, e criada uma cópia da linha para cada objeto utilizando esta variável. O valor `item/id` em cada linha é o atributo `id` do objeto para a linha corrente.

O nome `item` da variável não é especial; poder-se-ia utilizar qualquer nome desde começasse por uma letra e que contenha apenas letras, números ou `"_"`. A variável existe apenas no tag no qual encontra-se o atributo `tal:repeat`, no caso acima o tag `<tr>`; Se deseja utiliza-la o fora do tag, obterá um erro.

Cada linha no exemplo é numerada, o que implica o conhecimento do número corrente. Poderia também querer identificar se uma linha específica é a primeira ou a última ou outras informações similares. Em cada cópia da linha, a variável `item` representa um objeto da lista. Este objeto não está consciente do laço, sabe nada a respeito do loop, por conseguinte não existe atributo específico do objeto que permita extrair informações sobre o laço. A variável interna `repeat` foi criada exatamente para isso. Se põe o nome da variável de conteúdo (`item` no caso), após a variável `repeat`, pode aceder à contagem do laço a partir de zero (`index`), partir de 1 (`number`), partir de "A" (`Letter`), e de diferentes outras maneiras.

Por conseguinte, a expressão `repeat/item/number` é igual à 1 na primeira linha, 2 na segunda, etc. Dado que se pode inserir a expressão `tal:repeat` no interior de outra expressão `tal:repeat`, vários laços podem estar ativos por vez. Por essa razão deve-se usar `repeat/item/number` ao invés de `repeat/number`, ou seja, simplesmente para obter informações do `tal:repeat` correto, aquele que criou a variável `item`.

6 Elementos condicionais

Analise o *template*, e notará que a tabela está bastante simplificada. Fazamos de sorte a preencher alternativamente o fundo das linhas, isso vai melhorar o visual da tabela. Copie a segunda linha da tabela, para obter isto:

```
<table border="1" width="100%">  
<tr>  
<th>#</th><th>Id</th><th>Meta-Type</th><th>Title</th>  
</tr>  
<tbody tal:repeat="item container/objectValues">  
<tr bgcolor="#EEEEEE" tal:condition="repeat/item/even">  
<td tal:content="repeat/item/number">#</td>  
<td tal:content="item/id">Id</td>  
<td tal:content="item/meta_type">Meta-Type</td>
```

```
<td tal:content="item/title">Title</td>
</tr>
<tr tal:condition="repeat/item/odd">
<td tal:content="repeat/item/number">#</td>
<td tal:content="item/id">Id</td>
<td tal:content="item/meta_type">Meta-Type</td>
<td tal:content="item/title">Title</td>
</tr>
</tbody>
</table>
```

A expressão `tal:repeat` não se alterou, deslocamo-la simplesmente para um novo tag `tbody`. É um tag padrão HTML que permite agrupar o corpo das linhas de uma tabela. Há duas linhas no corpo, com colunas idênticas, das quais uma tem fundo cinza.

Visualize o código fonte do *template* e verifique as duas linhas. Se não acrescentar o atributo `tal:condition` às linhas, o *template* será gerado com duas linhas para cada objeto, o que não é bem o que se quer obter. O atributo `tal:condition` sobre a primeira linha assegura que esta será gerada apenas se o número da linha é par, ao passo que a condição da segunda linha permitirá gera-la apenas se o número é ímpar.

O atributo `tal:condition` não faz nada se o resultado da avaliação de sua expressão é verdadeiro, mas suprime o tag em totalidade, juntamente com seu conteúdo, se é falso. Os atributos `odd` e `even` de `repeat/item` são quer 0, quer 1. O número 0, uma cadeia vazia, uma lista vazia e a variável interna `nothing` são expressões falsas. Quase todos os outros valores são verdadeiros, incluindo os números não iguais a 0 e as cadeias de caracteres que contenham qualquer coisa (mesmo espaços!).

7 Definição de variáveis

O *template* mostra sempre pelo menos uma linha, dado que o próprio *template* ele mesmo é um dos objetos listados. Em outras circunstâncias, poder-se-ia desejar ter em conta o fato de que a tabela pudesse estar vazia, sem elementos. Neste caso o mais interessante seria simplesmente suprimir a tabela. Pode-se fazer isso posicionando uma condição no interior do tag `table`:

```
<table border="1" width="100%"
tal:condition="container/objectValues"
```

Agora, quando não há elementos, a tabela não será gerada. Porém quando há objetos, a expressão `container/objectValues` será avaliada duas vezes, o que não é muito eficaz. Além disso, se quiser alterar a expressão, deverá altera-la em 2 lugares. Para evitar estes problemas, pode-se definir uma variável para manipular a lista gerada, e utiliza-la nos

atributos `tal:condition` e `tal:repeat`. Altere a primeira linha da tabela para obter isto:

```
<table border="1" width="100%"
tal:define="items container/objectValues"
tal:condition="items">
<tr>
<th>#</th><th>Id</th><th>Meta-Type</th><th>Title</th>
</tr>
<tbody tal:repeat="item items">
```

O atributo `tal:define` cria a variável `items`, que poderá ser utilizada por toda a parte dentro do tag da `table`. Agora, suponha que queiramos apresentar uma mensagem e não simplesmente suprimir a tabela quando não há elementos. Para fazer isso, insira na tabela acima:

```
<h4 tal:condition="not:container/objectValues">There
Are No Items</h4>
```

Não se pode utilizar a variável `items` aqui, porque esta ainda não foi definida. Se a definirmos no tag `h4`, não ela poderá ser utilizada na tabela, porque tornar-se-ia uma variável local ao tag `h4`. Poderia-se defini-la dentro de um tag que englobaria os dois, mas existe uma solução mais simples. Utilizando a palavra-chave global na frente do nome da variável, define-se a variável do tag `h4` na parte inferior do *template*.

```
<h4 tal:define="global items container/objectValues"
tal:condition="not:items">There Are No Items</h4>
<table border="1" width="100%" tal:condition="items">
```

A expressão `not:` no primeiro atributo `tal:condition` pode ser colocada na frente de qualquer expressão. Se a expressão é verdadeira, `not:` a torna falsa e vice-versa.

8 Mudança de atributos

A maior parte, se não todos, os objetos listados pelo *template* têm um atributo *icon*, que contém o caminho para o ícone do tipo do objeto. De maneira a mostrar este ícone na coluna meta-type, seria necessário inserir este caminho no atributo `src` do tag `img`, alterando as duas colunas, dessa forma:

```
<td>
<span tal:replace="item/meta_type">Meta-Type</span>
</td>
```

O atributo `tal:attributes` substitui o atributo `src` de uma imagem com o valor `item/icon`. O valor de `em src` do *template* age como um substituto, assim a imagem será inserida sempre conforme o objeto, além do mais no seu tamanho correto. Dado que o atributo `tal:content` inserido na célula da tabela deveria substituir a totalidade do conteúdo da célula, a imagem incluída, pelo texto `meta-type`, seria suprimida. Assim, a solução é inserir `meta-type` diretamente da mesma maneira que a URL no exemplo do início do documento.

9 Mistura de atributos e seqüência de avaliação das declarações

Como se pôde ver no exemplo do *template*, é permitido inserir várias declarações TAL no mesmo tag. Contudo, deve-se atentar para 3 coisas:

- Só uma mesma ocorrência de declaração TAL pode ser utilizada num mesmo tag. Isso porque o HTML não permite se ter vários atributos com o mesmo nome. Por exemplo, não se pode ter mais de um `tal:define` no mesmo tag.

- As declarações `tal:content` e `tal:replace` não podem ser utilizadas no mesmo tag, já que entrariam em conflito óbvio.

- A ordem na qual se escreve os atributos TAL num tag não afeta a ordem na qual serão executados. Independentemente da ordem na qual são colocadas as declarações TAL, estas **sempre** serão executadas nesta ordem: `define`, `condition`, `repeat`, `content/replace`, `attributes`.

Para contornar estes limites, pode-se acrescentar outras tags e dividir as declarações entre estes tags. Se não há nenhum tipo de tag que corresponda ao mais adequado, pode-se utilizar os tags estruturantes `span` ou `div`.

Por exemplo, caso queira declarar uma variável para cada repetição de um determinado parágrafo, não pode inserir a declaração `tal:define` juntamente com `tal:repeat`, isso porque pela seqüência de avaliação padrão, a definição faz-se antes de todas as repetições. Para resolver este problema, pode escrever do seguinte modo:

```
<div tal:repeat="p phrases">
<p tal:define="n repeat/p/number">
Phrase número <span tal:replace="n">1</span> est
"<span tal:replace="p">Phrase</span>" .</p>
</div>
<p tal:repeat="p phrases">
<span tal:define="n repeat/p/number">
Phrase número <span tal:replace="n">1</span> est
"<span tal:replace="p">Phrase</span>" .</span>
</p>
```

10 Declarações com várias partes

Caso seja necessário definir vários atributos em um mesmo tag, não pode fazê-lo através de vários `tal:attributes` e, nesse caso, é obvio que coloca-los em vários tag diferentes não resolverá.

As declarações `tal:attributes` e `tal:define` podem ter ambas várias partes numa mesma declaração. Para isso deve-se separar estas partes por ";", se existe ";" dentro da expressão de uma destas declarações, é necessário então duplicar o sinal de ponto e vírgula: ";;". Eis um exemplo onde os atributos `alt` e `src` de uma imagem são definidos:

```

```

Eis um outro exemplo de definição de múltiplas variáveis:

```
<span tal:define = "global logo here/logo.gif; ids
here/objectIds">
```

11 As cadeias de caracteres (strings)

As expressões "cadeias de caracteres" permitem utilizar variáveis globais Zope (*path expression*). Cada uma destas variáveis deve ser precedida de "\$". Se há várias partes, ou se é necessário separa-las do texto que segue, devem ser postas entre "{}". Como o texto está dentro de um atributo, vocês não pode usar aspas (") para delimitar a variável. Como o sinal "\$" é utilizado para assinalar uma variável, o caracter "\$" deve ser escrito "\$\$". Eis alguns exemplos:

```
"string:Just text."
"string:© $year, by Me."
"string:Three ${vegetable}s, please."
"string>Your name is ${user/getUserName}!"
```

12 Expressão Nocal

Em geral, a chamada de uma URL tenta gerar o objeto correspondente. Isso significa que se o objeto é uma função (*Script, Method*), ou qualquer outra espécie de código executável, a chamada da URL avaliará o resultado do objeto chamado. É, em geral, o que se quer, mas não sempre. Por exemplo, caso se queira atribuir um Documento DTML à uma variável e poder aceder às suas propriedades, não se pode utilizar uma chamada de URL, porque esta voltará o documento como uma cadeia de caracteres. Ver Anotações na última página.

Caso seja inserida a expressão `nocall` no início do caminho (*path*), o objeto chamado não será avaliado, mas simplesmente retornado. Por exemplo:

```
<span tal:define="doc nocall:here/aDoc"  
tal:content="string:${doc/id}: ${doc/title}">  
Id: Title</span>
```

Este tipo de expressão é muito importante quando se quer definir uma variável que deve conter uma função, uma classe de um módulo, para seguidamente utiliza-la numa expressão Python.

13 Outras variáveis internas

Conforme já foi visto, alguns exemplos de variáveis internas, `template`, `gastar`, `repeat`, e `pedido`. Eis uma lista completa das outras variáveis internas e a sua utilização:

- `nothing`: este valor é falso, é semelhante a uma *string* vazia, que pode ser utilizado com `tal:replace` ou `tal:content` para apagar um tag ou o seu conteúdo. Se define um atributo como *nothing*, o atributo é retirado do tag, contrariamente um *string* vazio;
- `default`: É um valor especial que não altera nada quando é utilizado com `tal:replace`, `tal:content` ou `tal:attributes`. Deixa o texto do *template* inalterado;
- `option`: É um dicionário de valores, se existe, os valores passaram ao *template*.
- `attrs`: É um dicionário de atributos do tag que correm em do *template*. As chaves são os nomes dos atributos e os valores são os valores originais dos atributos no *template*.
- `root`: É objeto raiz de Zope. Utilizam este objeto quando quer aceder à objetos Zope a partir de uma posição fixa, independentemente do lugar do *template* ou a lugar da qual é chamado.
- `here`: É objeto no qual o *template* é chamado. Frequentemente é a mesma coisa que utilizar `container`, mas pode ser diferente se utiliza-se o mecanismo de aquisição Zope. Utilizar este objeto para aceder à objetos Zope que espera-se a encontrar em lugares diferentes dependentes da chamada o *template*.
- `container`: Em geral um *Folder* no qual encontra-se o *template*. Utilizado para aceder aos objetos Zope a partir de posições relativas ao lugar onde encontra-se o *template*.

14 Caminhos (*paths*) alternativos

O caminho `template/title` existe sempre que o *template* é utilizado, embora possa ser uma cadeia vazia. Certos caminhos como `request/form/x`, poderiam não existir quando o *template* foi elaborado. Isso causa em geral um erro na hora da avaliação do caminho.

Quando um caminho não existe, seria interessante ter-se um caminho padrão (default) a utilizar. Por exemplo, se `request/form/x` não existe, seria desejável utilizar `here/x` no lugar. Pode-se fazer isso listando os caminhos por ordem de preferência, separados por barras verticais "|":

```
<h4 tal:content="request/form/x | here/x">Header</h4>
```

Duas variáveis muito úteis a inserir em fim de listas de caminhos alternativos, são `default` e `nothing`. Utiliza-se `nothing` para apagar o resultado se nenhum dos caminhos for encontrado, ou `default` para deixar o texto por default.

Pode-se também testar do existência diretamente utilizando-se a palavra reservada `exists:`. Um caminho prefixado com `exists:` é verdadeiro se o caminho existe e falso no caso contrário. Estes dois exemplos afixam uma mensagem de erro apenas se alguma mensagem for passada pelo objeto `request`:

```
<h4 tal:define="err request/form/errmsg | nothing"
tal:condition="err" tal:content="err">Error!</h4>
<h4 tal:condition="exists:request/form/errmsg"
tal:content="request/form/errmsg">Error!</h4>
```

15 Elementos simples

Pode-se incluir elementos que são visíveis no *template* mas não no texto gerado utilizando a variável interna `nothing`, assim:

```
<tr tal:replace="nothing">
<td>10213</td><td>Example Item</td><td>$15.34</td>
</tr>
```

Isto pode ser muito útil para preencher partes de uma página de modo que assemelhe-se mais à página gerada que ao *template*. Por exemplo, uma tabela que deva ter 10 linhas, terá apenas uma linha no *template*. Acrescentando 9 outras linhas, o *template* assemelhar-se-á mais ao resultado final.

16 Utilização da palavra reservada estrutura

Em geral, as declarações `tal:replace` e `tal:content` convertem o conjunto dos caracteres especiais do HTML do texto que inserem, por exemplo substituem `<` por `<`. Se não se quer operar tais modificações sobre o texto, deve-se preceder a expressão pela palavra reservada `estrutura`. Dada a variável `copyright`, as 2 linhas seguintes:

```
<span tal:replace="copyright">Copyright 2000</span>  
<span tal:replace="structure copyright">Copyright 2000</span>
```

gerariam

© 2001 By **Me**" and "© 2001 By Me"

respectivamente.

Esta característica é muito útil quando se quer inserir um fragmento de código HTML armazenado numa propriedade, arquivo ou gerado por um objeto Zope. Por exemplo, pode-se ter elementos *news* que contém código HTML simples como texto em negrito e itálico quando gerado, e quer-se preservar isso ao inseri-los na página *Top News*. Neste caso, poder-se-ia escrever:

```
<p tal:repeat="article topnewsitems"  
tal:content="structure article">A News Article</p>
```

>> FIM <<

Anotações:

Expressão Nocall: Considere que os objetos do ZOPE possuem um método privado chamado `__call__()` que é executado quando estes objetos são requisitados pela URL, como por exemplo uma chamada a um DTML Method. A expressão `Nocall` faz com que não seja executado este método e portanto retorna a instância do objeto requisitado e não sua renderização(resultado da chamada de `__call__()`).

Pergunta: Como acessar objetos em TAL, através da sintaxe de dicionários?

Ex: `Request['atributo'].executaMetodo()`