

Capítulo 5: Introdução Básica ao Template

Trad.: José Marcelo de Miranda e Michael Almeida Lucas de Souza

Introdução Básica a Template

Plone une três camadas para criar uma página. Python e páginas templates criam um Hypertext Markup Language (HTML), que é enviado para o navegador. Lá, alguns CSS - Cascading Style Sheets formatam para uma melhor visualização. O código Python e páginas templates, são as principais áreas de discussão neste capítulo e no Capítulo 6.

Para compreender como se faz para gerar e editar um Plone template, você tem que primeiro aprender alguns conceitos chave sobre alguns fundamentos. Alguns desses conceitos são particularmente únicos do Plone, e embora forneça grandes vantagens, isto leva algum tempo para compreendê-los.

Neste capítulo, começarei explicando a publicação de um objeto, como interagir com objetos dentro do Plone e também como construir expressões. Explicarei como Plone junta tudo isso ao publicar uma página. No fim do capítulo, você criará uma página dentro de seu site Plone utilizando as técnicas que aprendeu.

Este capítulo fará mais sentido se você já possui conhecimentos em Python. De qualquer forma, em cada estágio eu explicarei o conceito atrás do código, assim mesmo não conhecendo Python, você não vai ter problemas. O restante do livro se refere a TALEs - Template Attribute Language Expression e Script (Python). Você deve ter uma vantagem: no capítulo anterior utilizamos o TALEs para gerar portlets e ações.

Entendendo o que está por traz do mecanismo de Template

Começar a falar em como o Plone templating funciona provavelmente o deixaria confuso, então começarei por explicar o que está por traz do mecanismo de teplate. Em um mundo ideal, isso é uma coisa com a qual você não teria de se preocupar, mas na prática descobri que esse é o primeiro obstáculo que as pessoas encontram tentando aprender a utilizar o Plone.

Tudo em Plone é um objeto, se você está familiarizado com o conceito de objeto, não tem muito o que saber; um *objeto* é "algo" que envolve uma ação. Cada objeto tem métodos que você pode chamar no objeto. Um exemplo é o mouse do computador. Um mouse de computador pode ter muitos métodos tais quais mover, clicar com o botão esquerdo, clicar com o botão direito etc.

Em Plone, um documento é um objeto de tipo particular, significando que o documento não é apenas uma parte de texto estático; Mas sim, algo mais complicado e muito mais útil. Um documento em Plone, por exemplo, tem um método 'descrição' que vai te dar a descrição que o usuário informou. Quando utilizando o sistema de templating, você irá ver em mais detalhes que tudo é um objeto. Irá ver primeiramente alguns dos princípios básicos de publicação de objeto.

Introdução a Publicação de Objeto

Em plone você está publicando objetos que estão localizados no Zope; a maioria deles são objetos persistentes no banco de dados de objeto. O conceito é mais complicado do que a interface padrão do CGI - Common Gateway Interface, onde um script é executado e passa por uma série de variáveis requeridas. Tudo no PLONE, ZOPE e Python são objetos. Até agora eu tenho evitado usar a palavra *objeto*; e utilizar palavras como *template*, *script*, e *item* que são apenas objetos que possuem ações diferentes.

Quando você requisita um endereço (URL - Uniform Resource Locator) do Plone, um objeto é chamado. O Plone faz isto traduzindo o endereço/URL em um caminho. Então, se o URL é */Plone/Login_form*, o que o Plone vai fazer é partir o URL em caminhos e procurar por objetos no banco de dados. Ele vai encontrar o objeto *Plone* e em seguida o objeto *Login_form*. Este método de procura pelo caminho é chamado *traversal*; Essencialmente, Zope chama o último objeto identificado no caminho (URL).

Quando o Zope chama o objeto *Login_form*, o objeto é executado dentro de seu contexto. A forma *contexto* é algo que você irá escutar muito em Plone. É simplesmente o contexto no qual o objeto atual está sendo executado, no caso é o */Plone*. O contexto muda muito conforme você se movimenta através do site Plone. Se você chamou a URL */Plone/Members/Login_form* em um navegador, então o contexto pode ser */Plone/Members*.

Como mencionado, *traversal* é como você pode acessar programaticamente objetos no Plone do mesmo modo que se faz com uma URL. Isto é similar aos itens de acesso em um arquivo de sistema, se você quer acessar uma figura em *My Documents* no Windows, você ia entrar em um diretório tal como o **c:\Documents and Settings\andy\My Documents\My Portrait.jpg**. Você pode acessar um objeto no Plone entrando no **Members/andy/My Portrait.jpg**. Este pode trabalhar se você tiver uma série de pastas e objetos que é o seguinte:

```
Members
  |_ andy
     |_ My Portrait.jpg
```

Na versão do sistema de arquivo, você passa através do disco rígido dos computadores, diretório por diretório. No Plone, acontece à mesma coisa; isto é apenas *Members* e *andy* são objetos.

Uma pegadinha é que o Zope é sensível a caracteres maiúsculos e minúsculos. No Windows, tanto faz, você pode digitar **My Portrait.jpg** ou **my portrait.jpg**. No Plone, entretanto; você tem que digitar exatamente igual ao ID do objeto. Por está razão, é recomendado que você mantenha todas URLs em minúsculas (lowercase) assim seus usuários terão menor possibilidade de erro.

Plone e Zope adicionaram um mecanismo, chamado **aquisição**, para esse sistema de publicação. O conceito de aquisição é de retenção: Objetos estão situados dentro de outros objetos chamados *containers*. No exemplo anterior, o objeto *andy* é um container dentro do objeto *Members* que é um container dentro do site Plone (que por sua vez é um container da aplicação Zope).

Em um ambiente orientado a objeto, um objeto herda o comportamento de seu pai. Em Plone e Zope, um objeto herda também o comportamento de seu container. Um objeto passa por uma hierarquia de container para entender como adquirir estes comportamentos.

Assim, veja o exemplo, acessando *Members/andy/My Portrait.jpg*. Se o objeto *Some Image.jpg* não existe na pasta *andy* mas ao invés disso existir em uma hierarquia superior? Bem, o mecanismo de aquisição o encontraria para você. Observe a seguinte hierarquia:

```
Members
  |_ andy
  |_ My Portrait.jpg
```

Neste caso, se você executasse a URL, Plone faria o *traverse* até *andy* e então tentaria encontrar *My Portrait.jpg*, mas como ele não está no container (*andy*), procuraria pela hierarquia, que é a pasta *Members*, e encontraria *My Portrait.jpg*, retornando a imagem.

Porém, se você comparar este com o exemplo anterior onde a imagem estava contida na pasta *andy*, você perceberia as seguintes diferenças chaves:

- no primeiro, o contexto é o mesmo, embora o objeto esteja em um lugar diferente, está baseado no local de onde o objeto é chamado.
- no segundo, o container é diferente, o container do *My Portrait.jpg* é *Members*, não *andy*.

Assim, qual é o ponto chave de tudo isso? Bem, você agora pode pôr um objeto na raiz de um site Plone, e qualquer outro objeto podem adquiri-lo porque é visualizado por aquisição.

Embora provavelmente faça sentido, a aquisição pode ser muito complicada, especialmente olhando através da hierarquia do contexto. Se você quiser aprender mais sobre este assunto, você pode ler uma excelente discussão mediada pelo líder desenvolvedor do Zope Jim Fulton's em <http://www.zope.org/Members/jim/Info/IPC8/AcquisitionAlgebra/index.html>.

Introdução as Expressões de Template

Antes de mergulhar no sistema de Zope Page Templates, você tem que entender o TALES. Frequentemente em uma aplicação você precisa escrever expressões que podem ser avaliadas dinamicamente. Estas expressões não são scripts; mais simples expressões que podem fazer algo simples e fácil em uma única linha de código.

Uma expressão é avaliada juntamente com uma série de variáveis locais. Estas variáveis são determinadas pelo que a expressão está chamando. O sistema de Workflow fornece um conjunto de variáveis, e o sistema Zope Page Templates fornece um outro. No momento, usaremos exemplos voltados ao *context*. Como discutido, o *context* é o contexto no qual um objeto é pedido.

A pouco vimos algumas expressões de TALES, como *string:\${portal_url}/Software*. Porém, isto é somente um exemplo de uma grande variedade de expressões. O

principal uso de TALES está no Zope Page Template, é o sistema de geração de código HTML para o Plone. Embora seu nome possa sugerir que somente é apropriado em templates, muitas ferramentas no Plone usam esta sintaxe para gerar expressões simples, como ações, workflow, e segurança. Existem diferentes tipos de expressões, falaremos de cada um.

Usando Expressões Path

A expressão path é o padrão e é a mais expressão utilizada. Ao contrário de todas as outras expressões, não exigem um prefixo para denotar o tipo de expressão. A expressão inclui um ou mais caminhos. Cada caminho está separado pelo símbolo pipe (|). Cada caminho é uma série de variáveis separadas por contra-barras (/). Veja os exemplos:

```
context/message
context/folderA/title
context/Members/andy/My Portrait.jpg
```

Quando a expressão é avaliada, o caminho é dividido pela contra-barras, começando pelo valor mais a esquerda fazendo o *traverse* para encontrar o objeto, método, ou valor. Ele então coloca o objeto na pilha atual e move sobre o próximo valor; ele repete este processo até que chegue ao fim da expressão ou até que não consiga achar um valor. Se o objeto encontrado for um dicionário Python ou *mapping object*, chamará aquele valor do dicionário. Uma característica legal de uma expressão path em que o único caractere restringido é o /, então nomes podem conter espaços e pontos e ainda assim podem ser avaliados.

Quando o fim é alcançado, chamará o objeto (se puder ser chamado). Se for um objeto não acessível, adquirirá o valor da string do objeto, e isto é o que será retornado. Se ocorrer algum erro nesta procura (o mais comum é que o atributo pedido não existe), então moverá para a expressão alternativa, se houver uma. Você pode especificar uma expressão alternativa separando-a com o símbolo pipe.

Por exemplo:

```
context/folderA/title|context/folderB/title
```

O exemplo acima retornará o título *folderA* se existir ou título *folderB* se o primeiro não existir. Repetirá este processo para cada expressão, até que não haja mais expressões ou até que avalie uma com sucesso.

Utilizando Expressões Not

Este tipo de expressão tem o prefixo *not*: é simplesmente o inverso da avaliação de uma expressão TALES que segue o prefixo. O sistema Zope Page Templates não possui uma declaração *if*, assim você poderá utiliza-la para testar o oposto de uma condição anterior.

Por exemplo:

```
not: context/message|nothing
```

Utilizando Expressão Nocal1

Quando uma expressão do tipo path alcança o último item na seqüência do caminho, se possível, irá chamar o item. O prefixo *nocal1*: impede que isto aconteça. Uma expressão nocal1 é raramente usada em Plone, mais isso tem usos ocasionais. Por exemplo, você pode usar isto para referenciar outro objeto. Veja o exemplo:

```
nocal1: context/someImage
```

Utilizando Expressões String

Expressões string permitem misturar texto e variáveis em uma única expressão, estas expressões começam com o prefixo *string:*. Está é uma função útil, e você verá que vai utiliza-la bastante. O texto pode conter qualquer coisa que é legalmente permitido dentro de um atributo, que contenha essencialmente caracteres alfanuméricos e espaços. Contidas dentro do texto podem ter variáveis, prefixadas com um sinal (\$). Aqui estão alguns exemplos:

```
string: This is some long string
string: This is the $title
```

No último exemplo, a variável *\$title* é avaliada, ela realmente pode ser qualquer expressão path, se essa variável possuir /, então tem que ser envolvida com {} para identificar o começo e o fim da expressão.

Por exemplo:

```
string: This is the ${context/someImage/title}.
```

Se um símbolo dollar (“ \$ “) precisa ser retirado no texto, utilize outro símbolo dollar (“ \$ “) antes do outro símbolo que você precisa retirar.

Por exemplo:

```
string: In $$US it costs ${context/myThing/cost}.
```

Utilizando Expressões Python

Expressões Python avaliam uma linha de código Python, todas começam com prefixo *python:* e contém uma linha de código Python.

Por exemplo:

```
python: 1 + 2
```

O código Python é avaliado utilizando o mesmo modelo de segurança que um objeto Script (Python) utiliza (como discutido no capítulo 6). Por estas razões, o código Python aqui utilizado deveria ser simples e limitado à funcionalidade de apresentação, como para formatar strings e números ou executar condições simples.

Mais adiante veremos que quase todas as expressões TALES mencionadas podem ser envolvidas e chamadas através do código Python. São as expressões:

- `path(string)`: Avalia uma expressão `path`
- `string(string)`: Avalia uma expressão `string`
- `exists(string)`: Avalia uma expressão `string`
- `nocall(string)`: Avalia uma expressão `nocall`

Por exemplo, o seguinte código:

```
python: path('context/Members')
```

é equivalente ao seguinte:

```
context/Members
```

Algumas funções foram adicionadas para dar assistência aos desenvolvedores. A função `test` pega três parâmetros: uma declaração para avaliar e as condições `true` e `false`. A declaração é avaliada, e o valor apropriado é retornado. Por exemplo:

```
python: test(1 - 1, 0, 1)
```

A função `same_type` pega duas variáveis e as compara se elas são iguais. Por exemplo:

```
python: same_type(something, '')
```

Alguns desenvolvedores desaprovam utilizar o Python dentro do sistema Zope Page Templates porque significa adicionar lógica na apresentação de templates. Frequentemente, o desenvolvedor, para cada parte de Python adicionado, deve-se perguntar se aquela parte de código ficaria melhor se colocada em um objeto Script (Python) separado. Isto não significa que você pode mover toda parte do Python, apenas pense sobre isto antes de adicionar qualquer coisa.

Web Site do Livro: Revisitando as Ações

No Capítulo 4, você adicionou uma ação apontando para a parte de software do site, aparecendo como uma aba `portal`. Nesta ação, você adicionou a expressão `string string: ${portal_url}/Software`. Isto pode fazer um pouco mais de sentido agora que foi explicado a variável `portal_url`. Esta é a URL para seu portal, que pode variar dependendo se você estiver usando um host virtual. Isto se faz utilizando o mecanismo de aquisição para adquirir o objeto `portal_url` e inserir o valor resultante na string. O resultado é que você sempre adquirirá um link para a pasta `Software`.

Misturando Python e Strings

Todas as expressões são diferentes, assim você não pode colocar o `path` como expressões dentro de uma expressão Python. Por exemplo, a expressão `python: here/Members + "/danae"` não faz sentido. A expressão inteira é interpretada como Python, assim Plone tentará fazer a divisão de `here` por `Members`, e vai ocorrer um erro. Esta é uma situação ideal para usar uma expressão string (que

deixa fazer a substituição de variáveis), assim uma variável pode conter uma expressão path. Assim, você pode usar a expressão *string*:
``${here/Members}/danae`.`

Utilizando o sistema Zope Page Templates

Agora que você entende da publicação de objeto e expressões, vamos falar do ponto chave do sistema, Page Templates Zope. Este é o sistema templating que o Plone utiliza para gerar o HTML.

Muitos sistemas de geração de HTML estão disponíveis, e alguns dos mais conhecidos são o JavaServer, Active Server Pages (ASP), e PHP. Para usuários de outros sistemas, o sistema Zope Page Templates pode parecer de início bastante estranho, mas rapidamente você verá que é um sistema extremamente poderoso.

O template mais simples se parece com:

```
<p tal:content="here/message">The title</p>
```

O template acima resulta na mensagem:

```
<p>Hello, World!</p>
```

Irei explicar alguns dos principais pontos e mostrar o que aconteceu aqui. Um parágrafo foi escrito em HTML, contudo o conteúdo desse parágrafo não é o texto (Hello, World!) mostrado na saída. Ao abrir a tag de parágrafo, um atributo *tal:content* foi adicionado, e a expressão *here/message* foi escrita para este atributo. O conteúdo do parágrafo foi emitido, porém, com o valor da variável *message* (neste caso, *Hello, World!*).

No momento, o template é avaliado, e o atributo *tal:content* é chamado. O *tal* representa Template Attribute Language (TAL) e possui um conjunto de comandos, incluindo o comando *content*. Mais adiante, você verá todos estes comandos, e com eles poderá manipular quaisquer tags de HTML, bem como criar loops, alterar tags, alterar atributos, remover tags, e assim por diante. Antes do template ser executado, ele se parecerá como um HTML Extensível (XHTML) válido e vai aparecer em um editor HTML como um parágrafo o texto Hello, World!.

Todas as páginas templates são XHTML válidos. Este é um padrão HTML e é um código XML - Extensible Markup Language válido. Para voce manter estes padrões siga as regras:

- Todas as tags devem ser em minúsculas.
- Atributos sempre devem estar entre aspas (por exemplo, `**`).
- Elementos vazios devem ser terminados com a contra-barra (por exemplo, `*
, not
*)`).

Para definir uma página como XHTML, você tem que declarar o DOCTYPE e tem que utilizar o namespace XML inserido na tag *html*. O Plone usa, no topo de toda página, a seguinte declaração:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
```

Para mais informações sobre a especificação de XHTML, vá para <http://www.w3.org/TR/xhtml1/#xhtml>.

Outro sistema de geração de HTML?

Nos primeiros anos da Web, os programadores eram os principais criadores de HTML, rapidamente lançaram sistemas para gerar, através da programação, o HTML. Com ferramentas tal como módulos CGI de Perl, os programadores conseguiam escrever complicados códigos.

Logo todos estavam gerando conteúdo, e esse processo teve que ser feito de um modo mais simplificado. Isto provocou uma onda de subterfúgios que codificam linguagens. Estas linguagens usaram um tipo especial de remarcação de HTML que era processada para produzir a saída (página HTML). Como mencionado, alguns dos mais populares são Active Server Pages (ASP), JavaServer Pages, e PHP. Zope seguiu esta tendência com Document Template Markup Language (DTML).

Estes sistemas levam HTML e intercalam com tags como `<%.%>` ou `* *`. Estes sistemas eram populares porque era de fácil entendimento, e os usuários que já conheciam o HTML básico poderiam pegar a idéia de mais tags. Os designers poderiam ignorar o conteúdo destas tags e poderiam deixar os programadores lidarem com elas. Programadores poderiam alterar as partes de código pertinentes sem desorganizar o conteúdo.

Porém, estes sistemas têm problemas:

- Os templates de HTML podem ser difíceis de modificar a medida que cada vez mais e mais conteúdo é acrescentado ao script. As Páginas rapidamente ficam enormes e difíceis de administrar. * A lógica e conteúdo não são separados nitidamente. Eles podem ser separados como alguns destes sistemas; porém, a habilidade para intercalar qualquer HTML com um pedaço de código programado é muito fácil. Frequentemente, o conteúdo, a apresentação, e a lógica se tornam uma grande bagunça. * As páginas não podem facilmente serem editadas. Frequentemente as páginas ou os templates vêm com a informação "just leave these bits alone..." pois se forem editadas quebraria o código. Editores WYSIWYG (O que você vê é o que você tem) podem ser configurados para não alterar algumas tags, mas eles podem quebrar outras facilmente. Em grandes organizações, todos os usuários com diferentes funções têm que editar a mesma página. * Pode ser difícil visualizar um resultado padrão. Pegue, por exemplo, uma busca (query) no banco de dados que mostra o resultado em uma tabela. Como pode um designer visualizar o resultado da busca sem executar aquele trecho de código de fato?

Por estas razões, foi criado o sistema Zope Page Templates. As páginas templates apresentam um acesso moderno; em vez de prover um outro método de codificação do tipo `<%.%>`, o código é acrescentado aos atributos das tags existentes. Este mecanismo não é somente utilizado pelo Zope, atualmente, versões desse sistema existem em Python, Perl, e Java.

Introdução as Page Templates e Conteúdo

Como você está ciente, Plone é um sistema de administração de conteúdo onde os usuários acrescentam conteúdo para um site Plone pela Web. Esses conteúdos são armazenados dentro do Plone e são mostrados usando páginas templates.

Retornando ao exemplo anterior de acessar */Members/andy/My Portrait.jpg*, explicarei o que de fato acontece ao conteúdo no Plone. Primeiro, Plone encontra e chama o objeto *Meu Portrait.jpg*; é chamado porque não há nenhum método específico que chama o objeto. Quando um tipo de conteúdo é chamado, um certo template é encontrado e executado. O contexto para aquele template será a imagem que você quer acessar, e o template será esse para aquela imagem.

Se uma ação diferente estivesse sendo chamada na imagem, como */Members/andy/My Portrait/image_edit*, então a ação *image_edit* seria observada para aquele objeto, e o template correspondente seria retornado. O Capítulo 11 discute como isto funciona em mais detalhes.

Assim, em todos os templates do Plone, você irá ver uma orientação para *here* ou *context*, que é o contexto do conteúdo que está sendo acessado. Em um template, você pode agora dizer *context/something or other*, e isto será o *something or other* relativo a parte do conteúdo, e não do template. Você criará agora seu primeiro template em Plone.

Criando seu primeiro page template

A forma padrão de se criar um page template é através da ZMI - Zope Management Interface. Infelizmente, isso quer dizer que você vai editar o template através de uma área de texto de um browser, a ZMI é também é mais complicada de se usar como desenvolvedor. A área de texto oferece funcionalidades limitadas quando comparado com a maioria dos editores; A falta de certas funções tais como linhas numeradas, realce da sintaxe, e assim por diante. No Capítulo 9, mostrei como usar o editor externo para editar conteúdo; isto permite a você editar conteúdos da Web em editores instalados localmente como o Macromedia Dreamweaver ou Emacs. No Capítulo 6, vou mostrar como fazer o Plone ler um page template, e aí você pode usar qualquer ferramenta que quiser.

Para criar um template, vá para a ZMI, clique *portal_skins*, clique *custom*, e então selecione Page template da caixa/menu drop-down (veja Figura 5-1). Clique no botão Add, e você verá a página mostrada na Figura 5-2.

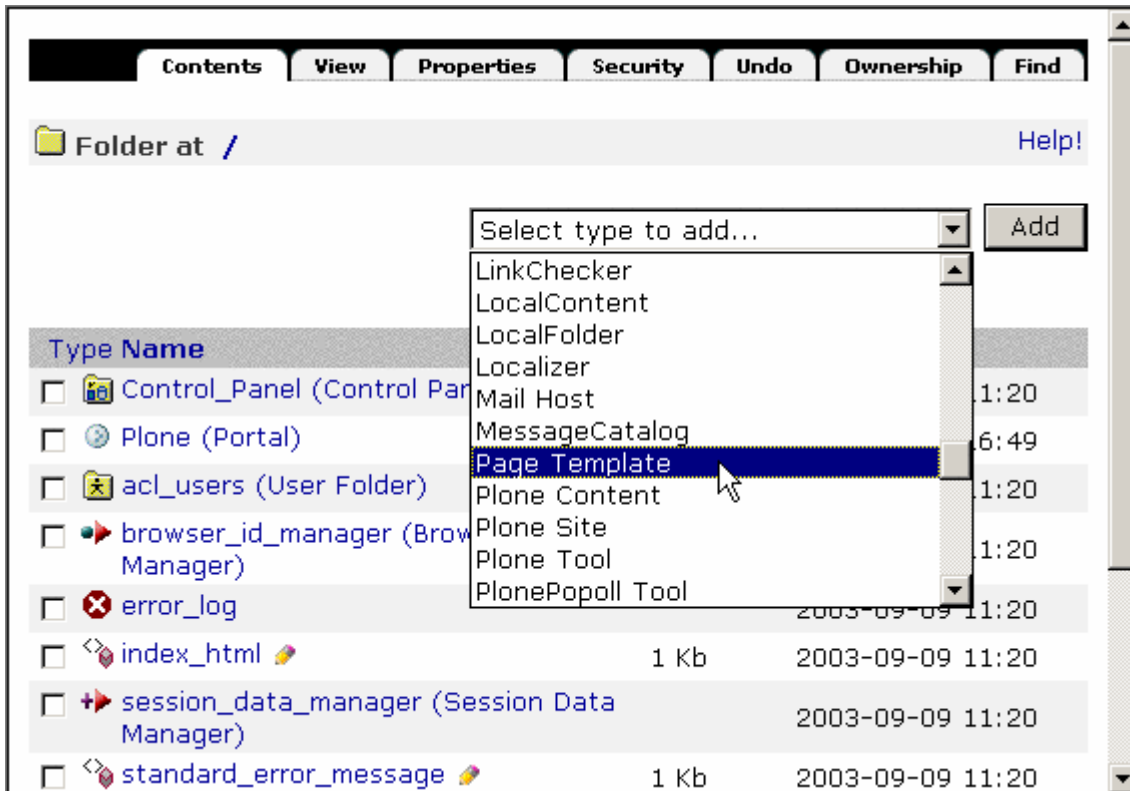


Figura 5-1. Selecionando a opção Page Template

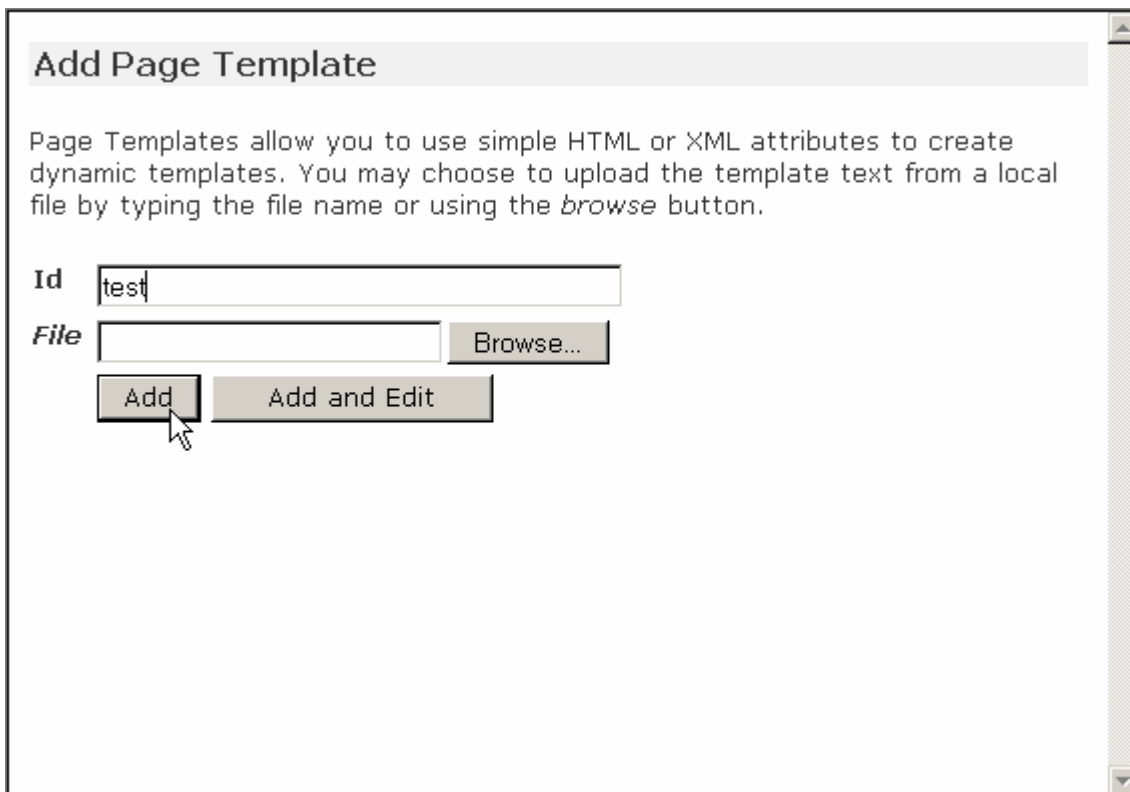


Figura 5-2. Adicionando uma Page Template

Entre **test** para a ID da page template. Então clique em Add e no botão Edit, que mostra a tela de gerenciamento para você (veja Figure 5-3). Daí você pode editar o template pela web usando um editor de texto e clicando em Save Changes para salvar suas mudanças.

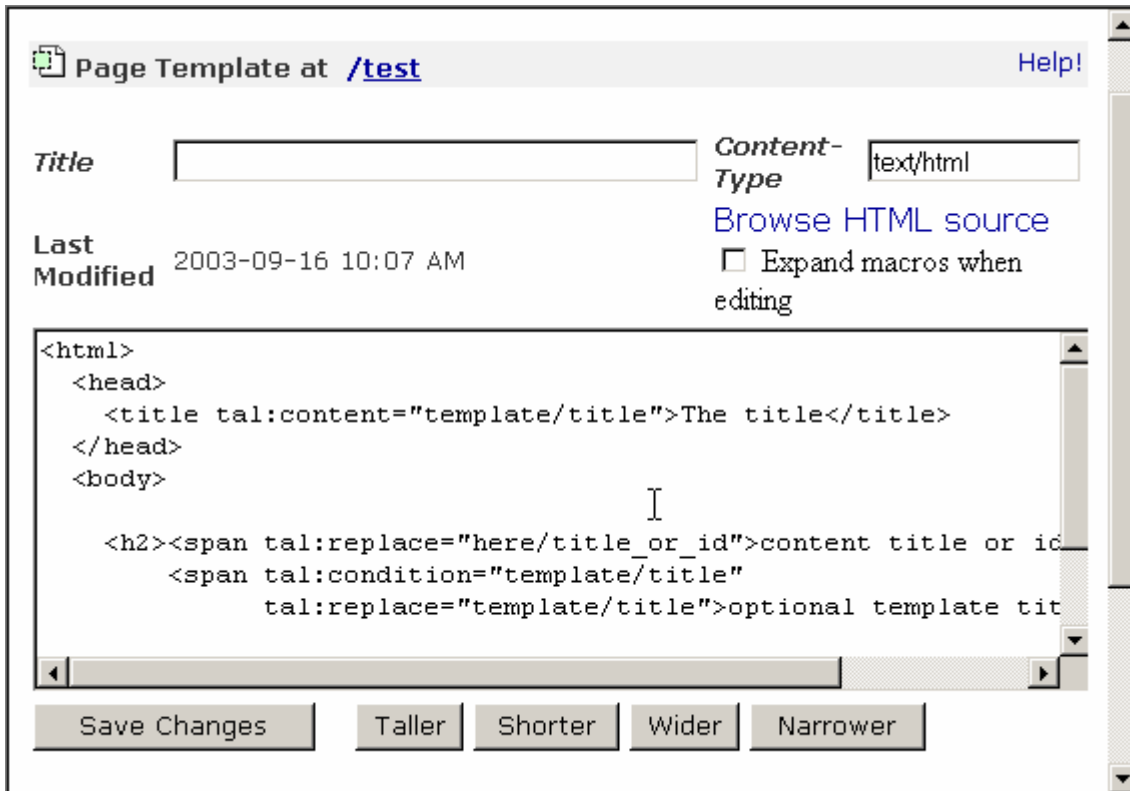


Figura 5-3. Editando uma page template

NOTA Antes do Plone 2, todas os page templates passaram pela variável *here*, que é equivalente a *context*. Se você ver *here* em qualquer código em uma page template, isto significa *context*. A nova variável *context* foi adicionada para ser mais clara e criar uma certa compatibilidade entre as page templates com objetos do Script(Python).

Após clicar Save Changes, a page template irá ser compilada. Se você tiver cometido qualquer erro no template, você os verá destacados no topo da página. A figura 5-4 mostra o erro com a tag *h1* que não foi fechada. (Como já mencionado, os templates da página devem ser XHTML válidos.)

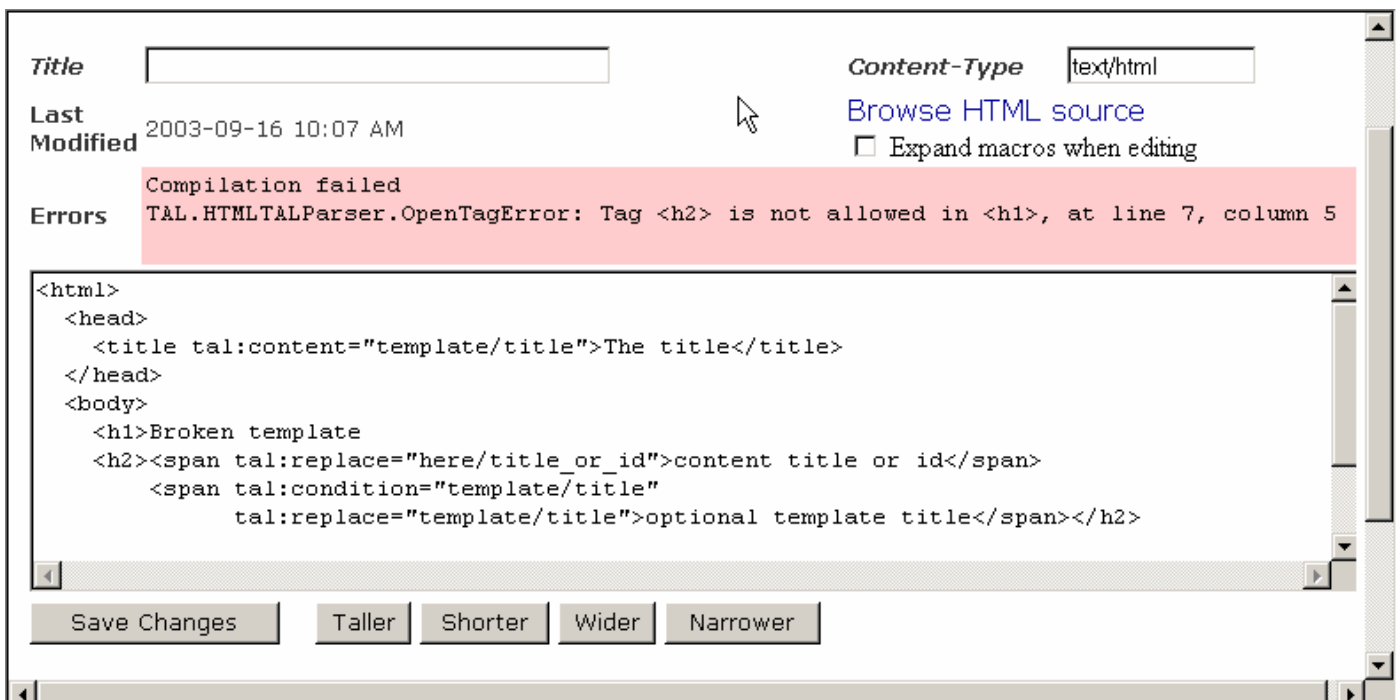


Figura 5-4. Erro na page template

Uma vez que você salvou a page template com sucesso, você pode clicar na aba Test para ver o template executado. Na Figura 5-5, você irá ver que o cabeçalho foi trocado pelo ID do template, e o título principal inclui agora o ID do template.

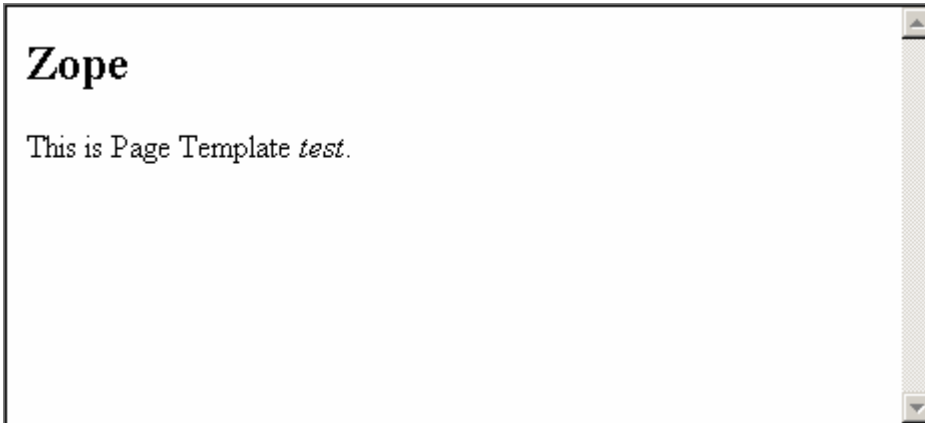


Figura 5-5. Gerando uma página

A tela de administração para a page template tem também as seguintes características importantes:

Title: Título para este template (opcional). Se você mudar isto no exemplo anterior, por exemplo, após clicar Test, você vai notar que o HTML resultante mudou.

Content-Type: Tipo de conteúdo para o template; normalmente é *text/html*.

Browse HTML source: Vai passar o template não processado como HTML. É como o template vai aparecer se for carregado em um editor do HTML.

Test: Isto vai processar e executar o template.

Expand macros when editing: Essa caixa de verificação (checkbox) vai tentar ampliar macros. Recomendo deixar sem essa verificação na maioria das vezes. Os macros são uma característica avançada e irão ser discutidas no capítulo 6.

Agora que você criou uma page template, você fará algumas modificações nele. Isto vai demonstrar os pontos discutidos até o momento neste capítulo. Por exemplo, se você quer que sua page template demonstre 1+2, você pode acrescentar a seguinte linha na sua page template:

```
<p>1+2 = <em tal:content="python: 1+2" /></p>
```

Então clique na aba Test para ver se está funcionando. Você deve ver o seguinte:

1+2 = 3

Para ver um exemplo de um “path traversal”, imprima o logotipo de seu site Plone. Você pode incluir uma expressão dentro do logotipo de seu site Plone adicionando o seguinte:

```
<p tal:replace="structure context/logo.jpg" />
```

Isto criará o HTML apropriado para a imagem que será mostrada na página.

Entendendo a Sintaxe Básica do page template

Agora que você viu como fazer um page template, Eu irei explicar sua sintaxe básica. Você pode quebrar a sintaxe de page templates em alguns componentes diferentes, que eu explicarei a seguir.

Introduzindo Variáveis Embutidas

Você viu a sintaxe da expressão, então agora você aprenderá sobre as variáveis que são passadas quando você faz um page template. Dentro do contexto de acessar a imagem *Some Image.jpg* na pasta *Members/andy*, chamada com a URL, */Members/andy/Some Image.jpg* acontece o seguinte:

container: É o container dentro do qual o template esta situado. No Plone geralmente é a pasta *portal_skins*. Você deve evitar a utilização do container *portal_skins* porque ele pode fazer ações inesperadas para o propósito de um container (por exemplo, uma referência para a pasta *andy*).

context: é o Contexto em que o template está sendo executado. No Plone este é o objeto que está sendo visto se você estiver vendo um objeto portal (por exemplo, uma referência para o objeto *Some Image.jpg*).

default: Algumas declarações possuem comportamentos padrões. É notável em cada uma das declarações, e está variável é um indicador desse comportamento.

here: É equivalente para *context*.

loop: É equivalente para *repeat*.

modules: Este é o container para os módulos importados. Por exemplo, *modules/string/atoi*. *atoi* é uma função do módulo *string* do Python. Isto inclui todos os módulos que são seguros para serem importados/utilizados nos Page Templates. Para mais informações, veja "Scripting Plone utilizando Python" no Capítulo 6.

nothing: Isto é o equivalente ao *None* de Python.

options: São as opções passadas para um template, acontece quando o template é chamado de um script ou por outro método, não pela Web.

repeat: É a repetição de um elemento; veja o elemento *tal:repeat* na seção "Introdução a sintaxe de comandos TAL" deste capítulo.

request: é a solicitação/pedido de um browser/cliente (todos os valores do pedido são visíveis usando o script abaixo de teste do contexto). Todos os

parâmetros *GET* e *POST* são ordenados em um dicionário para um fácil acesso. Aqui estão alguns exemplos:

production:* os códigos abaixo fazem parte da lista.**

```
request/HTTP_USER_AGENT # the users browser
request/REMOTE_ADDR # the users browser
request/someMessage # the value of some message, in the query string
```

****root**:** Esta é a raiz objeto Zope. Por exemplo, **root/Control_Panel** lhe dá o painel de controle do Zope.

****template**:** É o template que é chamado. Por exemplo, **template/id** é o ID do template que está sendo executado.

****traverse_subpath**:** Contém uma lista dos elementos que ainda vão ser 'transversed'. Esta é uma variável de alto nível, e é recomendada primeiro entender os conceitos 'transversal' e de aquisição antes de utiliza-la.

****user**:** É o objeto usuário atual. Por exemplo, **user/getUserName** é o username do usuário.

****CONTEXTS**:** Lista da maioria destes valores.

****NOTA ****Com a exceção do **CONTEXTS**, algumas destas variáveis podem ser redefinidas na declaração **tal:define**. Entretanto, isto pode ser confuso para qualquer um que esteja lendo o código e não é uma prática recomendada.

O page template *test_context* mostra todos os valores destas variáveis, mais as posições de alguns objetos (veja a lista 5-1). Pode ser útil para eliminar erros e descrever variáveis. Adicione-o como o page template chamado *test_context*, e então clique test para ver o resultado.

Listing 5-1. *test_context*

```
<html>
  <head />
  <body>
    <h1>Debug information</h1>
    <h2>CONTEXTS</h2>
    <ul>
      <tal:block
        tal:repeat="item CONTEXTS">
        <li
          tal:condition="python: item != 'request'"
          tal:define="context CONTEXTS;">
          <b tal:content="item" />
          <span tal:replace="python: context[item]" />
        </li>
      </tal:block>
    </ul>
    <h2>REQUEST</h2>
    <p tal:replace="structure request" />
```

```
</body>
</html>
```

O page template `test_context` produzirá a saída mostrada na Figura 5-6.

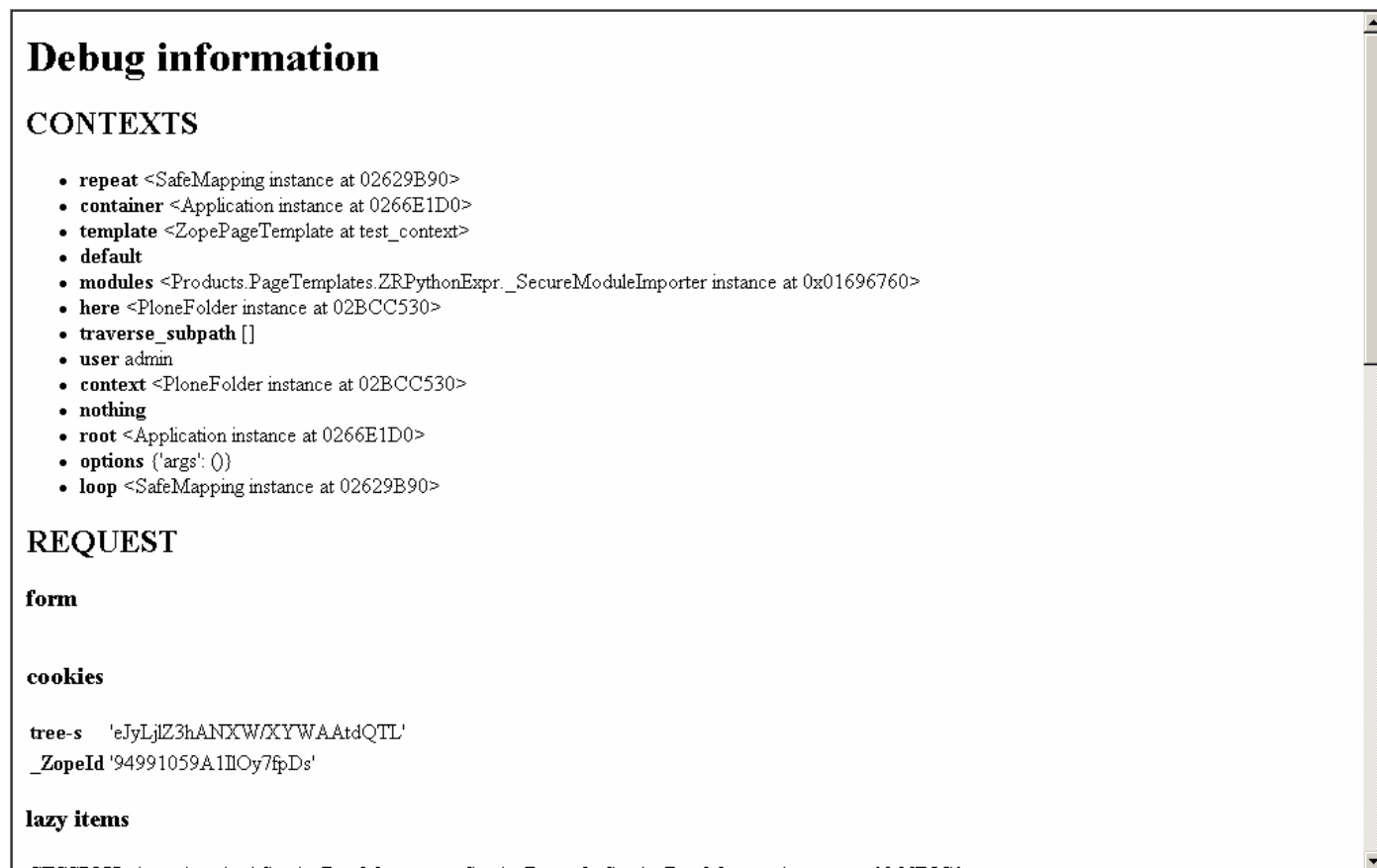


Figura 5-6. Um exemplo de todas as variáveis padrões de um script

Introdução a sintaxe de comandos TAL

O TAL (Template Attribute Language) fornece todos os blocos básicos de construção para uma apresentação dinâmica, definindo oito declarações: *attributes*, *condition*, *content*, *define*, *omit-tag*, *on-error*, *repeat*, e *replace*.

Sabendo que as páginas templates são XML válidos, todos os atributos TAL devem ser minúsculos (lowercase). Mais, cada elemento pode ter a declaração só uma vez. Nos exemplos a seguir, inseri novas linhas no elemento para aumentar a legibilidade; este código é perfeitamente válido e totalmente comum no Plone. Entretanto, isto é opcional e não é requerido.

tal:attributes: Mudando os Atributos de um Elemento

O *tal:attributes* permite que você substitua um ou mais atributos de um elemento. Uma declaração contém o atributo para ser mudado, separado por um espaço da declaração. Por exemplo:

```
<a href="#"
  tal:attributes="href context/absolute_url">
  Clique aqui
</a>
```

Isto mudará o atributo *href* do link para o resultado da expressão *here/absolute_url*. O atributo *href* já foi definido neste elemento, então se um designer abrir a página, o designer irá ver um elemento válido (embora o link não faça sentido até que a página seja processada). Segue um exemplo de saída:

```
<a href="http://plone.org/Members/andy/book">Clique aqui</a>
```

Sabendo que cada elemento possa ter múltiplos atributos, *tal:attributes* permite que você altere um ou mais atributos simultaneamente tendo declarações múltiplas. Para mudar múltiplos atributos de uma vez, separe as declarações com ponto-e-vírgula (;). Se o atributo ou a declaração conter ponto-e-vírgula, você pode deverá digitar outro ponto-e-vírgula (;;). Por exemplo, para mudar ambos os elementos *href* e *title*, faça o seguinte:

```
<a href="#"
  tal:attributes="href context/absolute_url;
                 title context/title_or_id">Link</a>
```

O exemplo da saída será:

```
<a href="http://plone.org/Members/andy/book">Plone Book</a>
```

As tags *tal:attributes* e *tal:replace* são mutuamente exclusivas desde que *replace* elimine o elemento. Se o sistema Zope Page Templates detecta isto, mostrará um aviso, e ignorará a tag *tal:attributes*. Por exemplo:

```
<a href="#"
  tal:attributes="href
                 python:request.get('message', 'change', default)"">
  Link</a>
```

Neste exemplo, estou usando a função *get* no objeto *request*. Se o pedido para a página conter a variável *message*, então o primeiro valor será usado, que é naturalmente *change*. Se a variável *message* não estiver presente, então o segundo valor, *default*, será usado. Conseqüentemente, só passando o parâmetro *message* para a mudança ocorrer.

tal:condition: Avaliando Condições

A declaração *tal:condition* permite que a condição seja testada antes de exibir o elemento. Por exemplo:

```
<p tal:condition="request/message">
  Minha mensagem
</p>
<p tal:condition="not: request/message">
  Nenhuma mensagem
</p>
```

O parágrafo com o texto para mensagem será mostrado somente se a variável *request* conter um atributo (solucionado como *true*). Sendo capaz de testar uma condição seria uma insensatez o teste da condição oposta, o que a expressão *not*

permite. O prefixo *not:* inverte a declaração, assim *not: request/message* resulta em *true* se a mensagem da variável pedida resulta *false*.

Em TAL, os seguintes resultados resultam em *false*:

- O número zero
- Qualquer float ou complex que resultam em zero (por exemplo, *0.0*)
- Strings com zero caracteres de tamanho (por exemplo, *""*)
- Uma lista ou tupla vazia
- Um dicionário vazio
- Valor Python *None*
- Valor TALEX *nothing*

Os seguinte resultados resultam em *true*:

- O valor padrão
- Qualquer número diferente de zero
- Strings que não estão vazios
- Strings que são apenas espaços (por exemplo, *" "*)
- Qualquer outra coisa

tal:content: Adicionando Texto

A declaração *tal:content* é geralmente a declaração mais usada em um page template. Esta declaração é também uma das mais simples, substituindo o conteúdo de um elemento com o valor especificado. Por exemplo:

```
<i tal:content="context/title_or_id">Some title</i>
```

A saída do exemplo é como segue:

```
<i>The title</i>
```

Isto substituirá o texto *Some title* com o valor da expressão *context/title_or_id*. Se o texto a ser colocado possuir elementos HTML, aqueles elementos serão retirados. Por padrão, o texto a ser substituído é NÃO HTML (escaped); o prefixo *structure* permitirá o HTML ser incorporado sem os elementos que estão sendo suprimidos. Por exemplo:

```
<i tal:content="structure here/title_or_id">Do not escape HTML</i>
```

Se o elemento com o atributo *tal:content* contém outros elementos, então todos esses elementos vão ser substituídos. As tags *tal:content* e *tal:replace* são mutuamente exclusivas; ambas não podem ser substituídas no mesmo elemento, e um erro será mostrado se voce tentar fazer isto. Se o valor é *default*, o conteúdo é inalterado.

tal:define: Definindo Variáveis

A declaração *tal:define* permite a criação e a reutilização de variáveis dentro dos templates. Por exemplo:

```
<p tal:define="title here/title_or_id">
```

```
... <i tal:content="title">The title</i> ...
</p>
```

Neste exemplo, o título da variável é criado e determinado pelo resultado de *here/title_or_id*; depois a variável *title* é usada na declaração *tal:content*. Por padrão a variável é criada localmente dentro do namespace do elemento atual. Assim, no exemplo anterior, só elementos dentro da tag do parágrafo podem usar a variável *title*. Você pode redefinir a variável em qualquer lugar dentro da declaração ou reutilizá-la em outros elementos quantas vezes forem necessárias.

Para criar uma variável para ser usada globalmente, você pode usar o prefixo *global*. Isto permitirá acesso para a variável de qualquer lugar dentro do template, não apenas definindo dentro do elemento. Por exemplo:

```
<p tal:define="global title string:Foo bar">
  ... <i tal:content="title">The title</i> ...
</p>
<i tal:content="title">We still have a title</i>
```

O Plone define um grande número de variáveis globais (*defines*) de modo que os usuários possam facilmente utilizá-las em seus scripts. Como qualquer variável, são sujeitas a mudanças, assim você deve usá-las com cuidado. Estes **defines** resultam em um grande número de variáveis globais disponíveis. Por exemplo, para adquirir o título do seu site Plone, você pode chamar o seguinte:

```
<p tal:content="portal_title" />
```

Você pode encontrar estes *defines* na ZMI clicando em *portal_skins*, clicando em *plone_templates*, e então clicar em *global_defines*. Você pode achar uma lista com todos os *defines*, e uma explicação deles, no Appendix A.

tal:omit-tag: Removendo elementos

O *tal:omit-tag* é bastante incomum, ele permite a remoção de uma tag. O sistema Zope Page Templates requer o uso de tags HTML, páginas complexas podem precisar freqüentemente de muitos elementos e podem resultar em tags extras. Para esta declaração, a tag é removida, deixando apenas o conteúdo das tags. Por exemplo:

```
<p tal:omit-tag="">This is some text</p>
```

A saída é como a seguinte:

```
This is some text
```

Neste exemplo, o texto *This is some text* será passado; porém, a tag não será passada. Opcionalmente, a declaração *tal:omit-tag* pode receber uma expressão como um argumento. Se a expressão for avaliada *false*, então a *tal:omit-tag* não ocorre. Por exemplo:

```
<p tal:omit-tag="nothing">This is some text</p>
```

Uma alternativa para usar *tal:omit-tag* é usando o namespace *tal*, como discutido na seção 'Dicas Úteis' do Capítulo 6.

tal:on-error: Executando a Manipulação de Erro

A declaração *tal:on-error* fornece um método para manipular erros. Funciona com o comando *tal:content* que promove a substituição do conteúdo da tag, mas somente quando ocorre um erro.

```
<p tal:content="request/message"
  tal:on-error="string: No message">Message</p>
```

Se houver um erro na avaliação da expressão *request/message*, então o atributo *on-error* será ativado. Isto faz com que o conteúdo da tag seja substituído com o texto *No message*.

Infelizmente, a declaração *on-error* é bastante limitada. A tag não consegue distinguir entre os diferentes tipos de erro e permite apenas que uma expressão seja avaliada e usada. Esta limitação é de projeto assim a tag não pode ter um uso excessivo. A manipulação de erro deve realmente ser segura na lógica de sua aplicação.

Felizmente, para todas as expressões, você pode fornecer alternativas na declaração se a primeira parte da declaração resultar algo diferente de *true* ou *false* (em outras palavras, se um erro é mostrado). Cada alternativa é separada por um caractere pipe (*|*), e múltiplas alternativas podem aparecer em uma declaração. Se você está apoiado nas variáveis da requisição que chega, então sempre adicione um */nothing* no final para assegurar que o erro atribuído não seja mostrado.

Por exemplo:

```
<p
  tal:content="request/message"
  tal:condition="request/message|nothing">
  There's a message
</p>
<p tal:condition="not: request/message|nothing">
  No message
</p>
```

Este segundo exemplo é mais trabalhoso, porém mais desejável por um par de razões:

- O programador pode ver a condição positiva e negativa.
- Você pode controlar uma condição de erro mais complicada do que apenas através da impressão de uma string.

tal:repeat: Executando Looping

O *tal:repeat* permite fazer o looping através dos objetos e é uma das declarações mais complicadas. Uma declaração contém o valor para ser atribuído

para cada iteração dos resultados, separada por um espaço dos resultados que estão sendo iterados.

```
<table>
  <tr tal:repeat="row context/portal_catalog">
    <td tal:content="row/Title">Title</td>
  </tr>
</table>
```

Neste exemplo, a expressão *here/portal_catalog* retorna a lista de resultados. O *repeat* começa na tag da tabela *row*, para cada *row* na lista de resultados, uma nova linha na tabela será criada. Assim como *tal:define*, cada iteração dos resultados são atribuídos para uma variável local (neste caso, *row*).

Você pode acessar algumas variáveis úteis da declaração *repeat*, tal como o número da iteração atual. Você pode acessa-las pela variável *repeat*, que é acrescentada ao namespace. Por exemplo, para acessar o número atual, você usa o seguinte:

```
<table>
  <tr tal:repeat="row context/portal_catalog">
    <td tal:content="repeat/row/number">1</td>
    <td tal:content="row/Title">Title</td>
  </tr>
</table>
```

Segue uma relação das variáveis disponíveis no *repeat*:

- **index**: Número de iterações, a partir de zero.
- - ***number**: Número de iterações, a partir de um.
- - ***even**: É *true* para iteração de números pares (por exemplo, 0, 2, 4, ...).
- - ***odd**: é *true* para uma iteração de números ímpares (por exemplo, 1, 3, 5, ...).
 - **start**: É *true* para a primeira iteração.
 - **end**: É *true* para a última iteração.
 - - ***length**: É o total do números de iterações.
 - **letter**: Número de iteração com escrita em minúscula (por exemplo, *a-z*, *aa-az*, *ba-bz*, ..., *za-zz*, *aaa-aaz*, e assim por diante), a partir de um.
 - - ***Letter**: É a versão maiúscula de **letter**.
 - - ***roman**: Número numeral Romano em minúscula (*i*, *ii*, *iii*, *iv*, *v*, e assim por diante), a partir de um.

Dois outros valores são disponíveis no namespace de *repeat* que são bastantes incomuns e raramente usados, *first* e *last*. Estas duas variáveis permitem que você guarde informação sobre os dados na iteração. Usando o valor que você quer guardar na expressão, assim um valor Booleano será retornado. Para a variável *first*, *true* indica que é a primeira vez que o valor ocorreu na iteração. Do mesmo modo, para a variável *last*, *true* indica que é a última vez que o valor ocorreu na iteração.

Aqui esta um exemplo disto:

```

<ul>
  <li tal:repeat="val context/objectValues">
    First: <i tal:content="repeat/val/first/meta_type" />,
    Last: <i tal:content="repeat/val/last/meta_type" />:
    <b tal:content="val/meta_type" />,
    <b tal:content="val/title_or_id" />
  </li>
</ul>

```

tal:replace: Adicionando Texto

A declaração *tal:replace* é similar para *tal:content* com uma diferença, esta remove a tag inteira.

Por exemplo:

```
<p tal:replace="context/title_or_id">Some title</p>
```

Isto fará o resultado da expressão *context/title_or_id* mas removerá do resultado as tags do parágrafo. Isto é o equivalente a:

```

<p
  tal:content="here/title_or_id"
  tal:omit-tag="">Some title</p>

```

Se a declaração *tal:replace* contém outros elemento, então todos os elementos serão substituídos. Você não pode usar a declaração *tal:replace* com *tal:attributes* ou *tal:content*; eles são mutuamente exclusivos, e um erro será mostrado se você coloca-los no mesmo elemento.

Introduzindo a 'Execução de Ordem'

A ordem na qual os atributos TAL foram escritos não é a que eles serão executados porque eles são na realidade elementos XML (e XML não se preocupa com ordens atribuídas). A ordem em que eles são executados está a seguir.

define condition repeat content replace attributes omit-tag Você não pode usar as declarações *content* e *replace* no mesmo elemento porque eles são mutuamente exclusivos. Usando a declaração *attributes* no mesmo elemento como um *replace* ou um *omit-tag* seria inócuo desde que os atributos são removidos. A tag *on-error* não é mencionada porque será usada quando o primeiro erro ocorrer em alguns dos elementos anteriores.

Example: Exibindo Informações do Usuário

Para ilustrar o que você aprendeu até agora, você criará um page template que execute uma tarefa simples: exibir informações sobre o usuário do sistema.

Neste exemplo, uma companhia está usando o Plone internamente como uma intranet. Cada empregado é registrado no Plone e lhe é dado um login; entretanto, não há nenhuma página que mostre o nome dos empregados ou como entrar em contato com eles. Você criará uma página simples contendo informações

do usuário: o endereço e-mail do usuário, home page, fotos, e quando foi a última vez que se logaram.

O primeiro protótipo desta página é facilmente realizado com TAL, TALES, e um pouco de conhecimento básico das ferramentas Content Management Framework (CMF). Infelizmente, porque as APIs - Application Programming Interfaces são bastante complicadas para estas ferramentas, alguns destes códigos são um pouco maiores do que devem ser. Não se preocupe muito com as ferramentas da API; estes serão explicados no Capítulo 9. Se você apenas utiliza a API para se garantir até o momento, você pode se concentrar no TAL.

Primeiro, você precisa criar um page template; clique em *portal_skins*, clique *custom*, adicione uma page template, e forneça um ID *user_info*. Segundo, você edite-o como segue. Você encontrará no Apêndice A a listagem completa deste page template. Examinando a listagem completa, você verá que o page template foi iniciado com HTML e tags body.

Por conveniência você colocará as definições principais dentro de uma tag *div*:

```
<div
  tal:omit-tag=""
  tal:define="
    userName request/username|nothing;
    userObj python: here.portal_membership.getMemberById(userName);
    getPortrait nocall: here/portal_membership/getPersonalPortrait;
    getFolder nocall: here/portal_membership/getHomeFolder
  ">
```

A tag *div* tem quatro definições: uma para pegar o nome do usuário passado pelo objeto pedido e outra para traduzir este nome do usuário de um objeto usuário. As duas últimas definições asseguram que você tenha referências válidas para os métodos que lhe darão acesso a foto e a pasta do usuário; isto é conveniente porque fazem dos códigos anteriores algo mais simples. É bastante comum num page template utilizar uma tag *div* ou outra tag *tal* como está que contém uma série de definições. Isto simplesmente faz com que o código se torne mais simples.

Logo, você inclui duas condições simples para se certificar de que você tem um usuário logado:

```
<p tal:condition="not: userName">
  No username selected.
</p>
<p tal:condition="not: userObj">
  That username does not exist.
</p>
```

Se nenhum username foi determinado na requisição, a expressão *request/username/nothing* resultará no nome do usuário (*userName*) isso é *nothing* e conseqüentemente falha o teste. Mais adiante, se o username não é válido, o *userObj* resultará em *None*, e as mensagens de erro serão impressas para ambas as condições.

Agora você está pronto para processar o usuário:

```
<table tal:condition="userObj">
  <tr>
    <td>
      <img src=""
        tal:replace="structure python: getPortrait(userName)" />
    </td>
```

Voce precisa se assegurar que um usuário foi encontrado, inserindo um teste (*tal:condition="userObj"*) na tag da tabela. Para mostrar a foto de um usuário, você vai usar o método *getPortrait* definido anteriormente. Esta função retorna a tag completa, então a tag *structure* assegura que a imagem seja passada corretamente. Em seguida, você quer mostrar algumas propriedades como *name* e *email*, como mostrado a seguir, pegando da pasta *home*:

```
<li
  tal:define="home python: getFolder(userName)"
  tal:condition="home">
  <a href=""
    tal:attributes="href home/absolute_url"
    >Home folder</a>
</li>
```

Primeiramente, você usa um *define* para pegar a pasta e atribuindo-a a variável *home*. No site Plone, a criação de uma pasta *home* para o usuário é opcional, assim você tem que estar certo de que se está linkando para uma pasta que existe. Felizmente, por causa da ordem de execução dos comando TAL, o *define* vem antes da condição. Seguindo isto, você mostra um link para a pasta usando o atributo *absolute_url* de uma pasta.

O page template passa por mais algumas linhas procurando outras propriedades úteis para mostrar. Como na maioria das coisas em Plone, o segredo é encontrar a chamada correta da API e então processar a saída adequadamente.

Finalmente, a página termina fechando todas as tags relevantes. Se tudo correr bem, você deve ser capaz de chamar a página acessando a URL

http://yoursite/user_info?userName=[someuser] onde someuser é um usuário válido no seu site Plone.

No momento, esta page template é bem limitada. Apenas um usuário, com a permissão de administrador, pode ver esta página, e pode ela mostrar somente um membro de cada vez além da informação do usuário ser bastante reduzida. No Capítulo 6, mostrarei como expandir este exemplo e adicionar alguns componentes reutilizáveis, como também a possibilidade de traduzir o texto em outra idioma.