Capítulo 6: Introdução Avançada a Templates e Scripts

Traduzido por Rafahela Garcia Bazzanella

O capítulo anterior abordou como o sistema de Zope Page Templates funciona. Para entender os page templates, o Capítulo 5 também abordou a hierarquia dos objetos, aquisição, e a Sintaxe de Expressão da Linguagem de Padrão de Atributos (Template Attribute Language Expression Syntax (TALES)). Usando o código do capítulo anterior, você era capaz de gerar páginas Web dinâmicas. O capítulo também mostrou um exemplo de page template que juntou o código, abordou a construção de blocos do sistema de templates no Plone, e forneceu informações chaves que você vai precisar para usar o Plone.

Agora é hora de passarmos para algumas das características mais avançadas dos page templates e dos templates do Plone em geral. Primeiro vou introduzir os namespaces da Macro Expansion Template Attribute Language (METAL) e da Internacionalização (I18N). Como os namespaces da TAL, eles oferecem funcionalidades ao desenvolvedor do site. Para aqueles que querem saber exatamente como as páginas do Plone são vinculadas, a seção Hooking Into Plone Using METAL oferece muitas respostas.

Até agora eu mostrei como você pode usar simples expressões do Python em page templates. É claro, algumas vezes uma expressão Python de uma linhas não é o suficiente. Assim na seção Scripting Plone with Python, vou mostrar como elevar o nível do Python e melhorar o poder dos seus scripts.

Ao final, vou abordar um exemplo comum, mostrando como colocar um formulário no Plone. Este exemplo demonstra os conceitos aprendidos nos capítulos anteriores e vincula tudo enquanto mostra exatamente como o Plone trata os formulários.

Entendendo os Templates Avançados do Plone

Um dos elementos mais legais dos page templates é que funções diferentes são claramente separadas em namespaces diferentes. No capítulo anterior, você viu os namespaces da TAL. Eles não são os únicos namespaces que o page template oferece; outros dois namespaces são importantes para o Plone.

O primeiro é o METAL. Como o nome sugere, ele é similar ao TAL no fato de ser uma linguagem de atributos e que insere-se em atributos do elemento. Entretanto, o objetivo principal é assegurar que você possa reusar pedaços de códigos de outros page templates. Ele faz isto usando funções de slots e macros.

O segundo é o I18N, que permite que você traduza o conteúdo dos page templates. Isto é usado no Plone para localizar a interface do Plone em mais de 30 línguas e para muitos usuários é uma das características chaves do Plone. Como você verá, a habilidade de localizar o texto é de interesse de todos os usuários, até mesmo para aqueles que controem sites em uma só língua. Você começará com o METAL.

Hooking Into Plone Using METAL

Até agora você viu como usar TAL para criar dinamicamente partes de páginas. Entretanto, isto realmente não permite que você faça templates muito complexos. Não há realmente um mecanismo para colocar um cabeçalho padrão no topo de cada página, a não ser usando uma condição TAL. METAL é um método que permite o pré-processamento de templates e oferece algumas funções mais poderosas que o TAL. Todas as funções METAL começam com o prefixo *metal:*.

metal:define-macro

O comando *metal:define-macro* permite que você defina um elemento para referenciar de outro template. O nome do pedaço referenciado é o nome da macro. A seguir temos um exemplo que define *boxA* como um pedaço que você quer usar em algum lugar:

```
<div metal:define-macro="boxA">
    ...
</div>
```

O elemento *div* é agora uma macro que pode ser referenciada de outros templates. A macro referencia somente a parte da página referenciada pelo elemento, que, neste caso, é a tag *div*. Assim, é comun usar múltiplas *macro:defines* em uma página e para a página ser uma página HTML (Hypertext Markup Language) válida, fazemos assim:

```
<html xmlns:tal="http://xml.zope.org/namespaces/tal"
    xmlns:metal="http://xml.zope.org/namespaces/metal"
    i18n:domain="plone">
    <body>
        <div metal:define-macro="boxA">
            ...
        </div>
        <div metal:define-macro="boxB">
            ...
        </div>
    </body>
</html>
```

Correspondendo aos objetivos anteriores dos page templates, esta página é uma página HTML válida que pode ser editada por um designer. Quando a macro é chamada, o HTML fora das tags *div* será descartado.

metal:use-macro

O comando *metal:use-macro* usa uma macro que foi definida usando o *define-macro*. Quando um template define uma macro usando o comando *define-macro*, ele se torna acessível a outros templates através da propriedade *macros*. Por exemplo, se você quer retirar a macro *portlet* do template *portlet_login*, você pode fazer o seguinte:

```
<div metal:use-macro="context/portlet_login/macros/portlet">
    The about slot will go here
```

```
</div>
```

Isto buscará a macro e inserirá o resultado em seu lugar. Como foi mostrado, o comando *use-macro* dá um caminho para a expressão que aponta para o template e então para a macro específica no template.

Exemplo: Usando as Macros use-macro e define-macro

Como exemplo disto, o que temos a seguir é um template chamado *time_template*. Este template mostra a data e hora do servidor atual do Plone. Esta é uma função muito útil para se ter, assim você pode colocá-la em uma macro para ser reusada. Este é o exemplo de um page template contendo o *define-macro*:

```
<html>
    <body>
        <div metal:define-macro="time">
            <div tal:content="context/ZopeTime">
                The time
            </div>
        </div>
    </body>
</html>
```

Se o seu template é chamado *time_template*, então você pode referenciar esta macro em outro template. Você pode agora referenciar esta macro em vários templates. Este é um exemplo do template:

```
<html>
    <body>
    <div metal:use-macro="context/time_template/macros/time">
      If there is a message then the macro will display i here.
    </div>
    </body>
</html>
```

Quando este template é renderizado, o HTML produz através do Plone o seguinte:

```
<html>
    <body>
    <div>
       <div>2004/04/15 17:18:18.312 GMT-7</div>
    </div>
    </body>
</html>
```

metal:define-slot

Uma *slot* é uma seção de uma macro que o autor do template espera não ser levada em conta por outro template. Podemos dizer que é como um buraco no seu page template que você espera ser preenchido com alguma coisa. Todos os comandos *define-slot* devem estar contidos dentro de um *define-macro*. Por exemplo:

```
<div metal:define-macro="master">
```

```
   <div metal:define-slot="main">
   ...
   </div>
 </div>
```

metal:fill-slot

Isto completa um slot que foi definido com o comando *use-slot*. Um *fill-slot* deve ser definido com um comando *use-macro*. Quando a parte *define- macro* é chamada, a macro preencherá todos os define slots com o *fill-slots* apropriado. Aqui temos um exemplo usando *fill-slot*:

```
 <div metal:use-macro="master">
   <div metal:fill-slot="main">
     The main slot will go here
   </div>
 </div>
```

Exemplo: Usando Macros e Slots

Voltando ao exemplo anterior, você poderá melhorar ele um pouco. Se você queria colocar uma mensagem padão no começo da hora, então você tem que adicionar uma slot no começo do *time_template*, dentro de *define-macro*. A slot é chamada *time* e fica assim:

```
 <html>
     <body>
         <div metal:define-macro="time">
             <div metal:define-slot="msg">Time slot</div>
             <div tal:content="context/ZopeTime">
                  The time
             </div>
         </div>
     </body>
 </html>
```

Agora, chamando o page template, você pode chamar o *fill-slot*:

```
 <html>
     <body>
     <div metal:use-macro="context/time_template/macros/time">
       <div metal:fill-slot="msg">The time is:</div>
       If there is a message then the macro will display i here.
     </div>
     </body>
 </html>
```

O resultado final é que você verá *time-slot* preenchido assim:

```
 <html>
     <body>
     <div>
        <div>The time is: </div>
```

```
        <div>2004/04/15 17:18:18.312 GMT-7</div>
     </div>
     </body>
 </html>
```

Como o Plone usa Macros e Slots

Ambos macros e slots são similares bem como ambos extraem conteúdo de outros templates e iserem conteúdo, mas eles fazem isto de maneiras diferentes. A diferença está em como eles são usados: Macros são elementos de um template que são explicitamente chamados, mas slots são como buracos em um template que você espera que outros templates preencham o vazio para você. Por exemplo, no caso do Plone, as portles tais como calendário, navegação, e assim por diante são macros que são chamadas explicitamente.

De fato, se em uma Interface de Gerenciamento do Zope (ZMI) você olhar o arquivo clicando em *portal_skins*, clicando em *plone_templates*, e então clicando em *main_template*, você verá que as páginas inteiras consistem de macros e slots. Neste estágio, é provavelmente um pouco confuso, mas quando chamadas, elas rodam através de uma série de macros e colcam tudo junto. Isto permite que um usuário altere facilmente qualquer parte de um site Plone sem levar em conta aquela macro, como você verá no próximo capítulo. Por exemplo:

```
 ...
 <div metal:use-macro="here/global_siteactions/macros/site_actions">
     Site-wide actions (Contact, Sitemap, Help, Style Switcher etc)
 </div>

 <div metal:use-macro="here/global_searchbox/macros/quick_search">
     The quicksearch box, normally placed at the top right
 </div>
 ...
```

Continuando a olhar o arquivo *main_template*, você encontrará alguns define slots. Brevememnte vou recaptular como uma página é renderizada no Plone. Quando um objeto é mostrado, um template para a visualização do conteúdo é mostrado. Quando você visualiza uma imagem, o template *image_view* é mostrado, e aquele template controla como a imagem é mostrada. Para realizar esta tarefa, o template da imagem preenche a slot *main*. Se você observar o template *image_view*, você verá o seguinte código definido naquele template:

```
 <div metal:fill-slot="main">
 ...
 </div>
```

Se você voltar ao *main_template*, você verá que ele contém uma definição *define-slot* para a slot *main*:

```
 <metal:bodytext metal:define-slot="main" tal:content="nothing">
     Page body text
 </metal:bodytext>
```

Cada tipo de conteúdo tem um template diferente, e cada tempate define como eles usarão, de maneira diferente, a slot principal. Assim, cada tipo de conteúdo tem seu próprio look and feel do template. Apenas um elemento é perdido na equação. Por alguma razão, quando você chamou o *image_view*, o template sabia que ele deveria usar o *main_template*. No template *image_view*, você usa a macro do *main_template*. Isto é definido no HTML seguinte:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en-US"
    lang="en-US"
    metal:use-macro="here/main_template/macros/master"
    i18n:domain="plone">
```

Neste caso, o *main_template* tem uma slot *main* preenchida pela slot definida como *main* no template que foi renderizado. O que segue é uma timeline de como a página é construída para a imagem ser visualizada:

image_view main_template image_view main_template define-slot="main" fill-slot image_view fill-slot main_template main_template

Isto permite que o Plone seja flexível em termos de como cada página é definida. Por exemplo, *main_template* define mais que aquela primeira slot; há também uma slot para inserção de código de Folhas de Estilos Encadeados (Cascading Style Sheets (CSS)):

```
<metal:cssslot fill-slot="css_slot">
    <metal:cssslot define-slot="css_slot" />
</metal:cssslot>
```

Se a visualização precisa de um conjunto padrão de CSS, você pode definir esta slot na visualização, e ela será preenchida quando renderizada. Alguns macros no *main_template* definem slots nelas também, e então preenchem elas novamente no *main_template* assim se você realmente precisar, você pode também preencher aquelas slots. Esta é uma técnica avançada, entretanto, você terá que certificar-se que você tem a base antes de cair na estrada.

Introdução a Internacionalização

O Plone está sempre se esforçando para manter uma grande qualidade em suas traduções. O fato é que o Plone oferece uma interface acessível para o usuário em mais de 30 línguas, que é o ponto estratégico do Plone. Isto também significa que o I18N é uma característica chave dos templates. Para facilitar isto, o namespace do I18N é um namespace extra como o TAL ou METAL, que tem condições específicas.

Esta sessão detalha o que os usuários precisam saber a respeito dos templates. Em um template você pode adicionar uma tag *i18n* a um elemento que permitirá a tradução de um atributo ou de seu conteúdo. Existem seis condições: *attributes*, *data*, *domain*, *source*, *target*, e *translate*. O modelo básico é envolver o pedaço de texto que você quer traduzir e adicionar o atributo *i18n* apropriado. Por exemplo, se você quer traduzir o seguinte:

```
<i>Some text</i>
```

poderia ser o seguinte:

```
<i i18n:translate="some_text_label">Some text</i>
```

Cada localização oferece uma tradução de *Some text*, e a ferramenta de tradução verifica uma tradução para o usuário. Quando a tradução é executada, cada string a ser traduzida deve ter um ID da mensagem único que identifica o item a ser traduzido. Por exemplo, uma string como *Search* pode ter um ID da mensagem como *search_widget_label*. O ID da mensagem permite que a sting seja identificada unicamente e a tradução pode ser repetida.

i18n:translate

Isto traduz o conteúdo de um elemento, com um ID da mensagem opcional passado como um parâmetro. Por exemplo, a seguir será criado um ID da mensagem *title_string*:

```
<h1 i18n:translate="title_string">This is a title</h1>
```

Este exemplo é para um pedaço de texto que é estático e não muda. Entretanto, em algumas situações, o pedaço de texto deve ser pego do banco de dados ou de um objeto, ficando dinâmico. Deixando a condição *translate* em branco, o ID da mensagem é composta do valor no campo. No seguinte exemplo, se o título retornado pelo caminho da expressão *here/title* for *Alice in Wonderland*, então aquele título deve ser passado para a ferramenta de tradução. Se nenhuma existir, o valor original será inserido:

```
<h1
    tal:content="here/title"
    i18n:translate="">
      This is a title.
</h1>
```

O comando de tradução é provavelmente uma das tags *i18n* mais comuns que você usará, e você verá ela nos templates do Plone. Ela não somente possibilita que você traduza partes estáticas de seu site, como rótulos de formulários, mensagens de ajuda, e descrições, mas também traduza as partes mais dinâmicas do seu site que podem mudar mais frequentemente, como títulos das páginas.

i18n:domain

Isto configura o domínio da tradução. Para prevenir conflitos, cada site pode ter múltiplos domínios ou grupos de traduções; por exemplo, pode haver um domínio para o Plone e um para sua aplicação padrão. O Plone usa o domínio *plone*, que é geralmente o domínio padrão no Plone:

```
<body i18n:domain="plone">
```

Você não deveria usar muito esta tag; entretanto, se você está escrevendo uma aplicação padrão, você pode achá-la útil para ter um domínio que não tenha conflito com outros domínios.

Ii8n:source

Isto configura a linguagem para o texto a ser traduzido. Não é usado no Plone:

```
<p i18n:source="en" i18n:translate="">Some text</p>
```

i18n:name

Isto oferece uma maneira de preservar elementos em um grande bloco de texto assim o bloco de texto pode ser reordenado. Em muitas linguagens, não somente as palavras mudam mas também a ordem. Se você tem que traduzir um parágrafo inteiro ou sentença que contenham pedaços pequenos de texto que deveriam ser traduzidos, então eles podem ser passados assim:

```
<p i18n:translate="book_message">
    The
    <span
       tal:omit-tag=""
       tal:content="book/color"
       i18n:name="age">Blue</span>
    Book
</p>
```

Isto irá produzir a seguinte string de mensagem:

```
The {color} Book
```

Se a linguagem alvo exige que esteja em uma ordem diferente, então elas podem ser modicas e ainda ter o conteúdo dinâmico no lugar correto. Em Francês, isto deve ser traduzido assim:

```
Le Livre {color}
```

i18n:target

Isto configura a linguagem alvo para o texto a ser traduzido. Não é usado no Plone.

i18n:attributes

Isto permite a tradução de atributos dentro de um elemento, ao invés do conteúdo. Por exemplo, uma tag *image* tem o atributo *alt*, que mostra uma representação alternada da imagem:

```
<img
    href="/someimage.jpg"
    alt="Some text"
    i18n:attributes="alt alternate_image_label" />
```

Atributos múltiplos devem ser separados com ponto-e-vírgula, como *tal:attributes* funciona.

i18n:data

Isto oferece uma maneira de traduzir outras coisas que não sejam strings. Um exemplo é o objeto *DateTime*. Uma condição *i18n:data* requer uma determinada consição *i18n:translate* assim um ID da mensagem é disponibilizado. Por exemplo:

```
<span i18n:data="here/currentTime"
    i18n:translate="timefmt"
    i18n:name="time">2:32 pm</span>... beep!
```

Translation Service

Agora que abordamos as tags, abordarei o mecanismo para executar a tradução. Por padrão o Plone vem com um mecanismo I18N. Ele permite que você internacionalize a interface do usuário assim as mensagens, abas e formulários podem ser traduzidos. Neste momento, não abordamos o conteúdo atual que os usuários adicionam. Se você adicionar um documento em Inglês e visualizar a página querendo-a em Francês, você vai obter um documento em Inglês com o texto ao redor em Francês (veja a Figura 6-1).

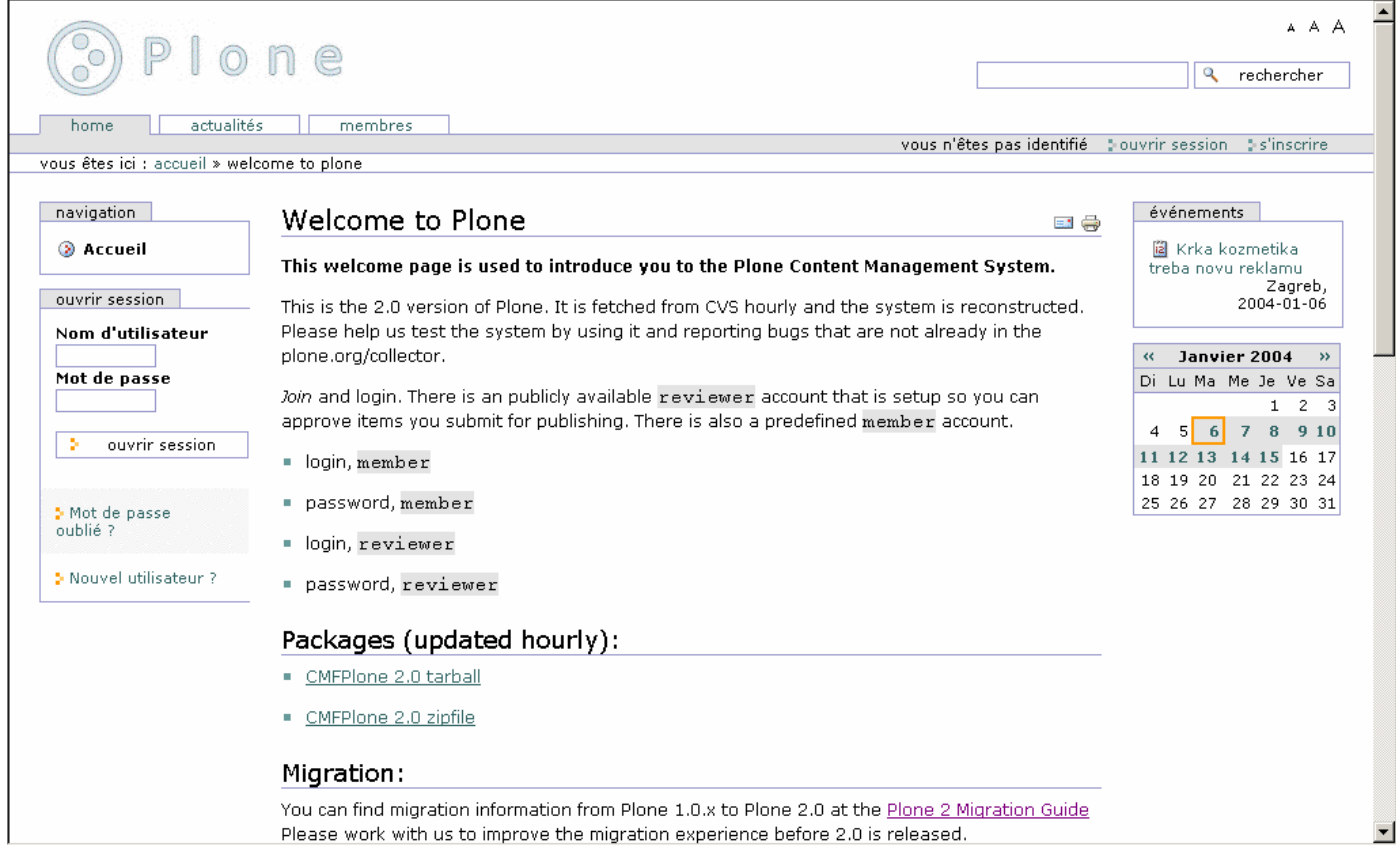

Figura 6-1. Plone.org em Francês

O Plone lê o cabeçalho HTML que o browser envia para o cliente requisitando uma linguagem. Se seu browser está em Inglês, então você não verá muito.

Para mudar a linguagem configurada no Internet Explorer, faça o seguinte:

Depois que você fizer isto, abra e visite o seu site preferido do Plone com o seu browser.

As traduções para o Plone são tratadas através da ferramenta chamada Placeless Translation Service (PTS). Você pode localizar a ferramenta PTS no painel de controle do Zope; no final da página você verá uma opção para Placeless Translation Service. Clique nela, e abrirá todas as traduções que existem. Estas traduções são lidas do sistema de arquivos; clique em uma tradução para ver a informação sobre a linguagem, como o translator, o encoding, e o caminho para o arquivo. Todos os campos estão atualmente armazenados no diretório *i18n* do diretório *CMFPlone*.

As traduções são tratadas usando dois arquivos para uma tradução, um arquivo *.po* e um *.mo*. Por exemplo, *plone-de.po* contém as traduções para o Alemão (*de* é o código para Alemanha). O arquivo *.mo* é a versão compilada do arquivo *.po* e é usado pelo Plone para apresentação. Você nunca precisará usar o arquivo *.mo*, assim você apenas ignora-o. O *.po* é o arquivo que você pode editar para alterar a tradução. Se você abrir este arquivo em um editor de texto, você verá várias linhas começando com o texto *msgid* ou *msgstr*. Acima o *msgid* é atualmente o código onde os comandos *i18n* acontecem, assim você pode ver que parte da página você está traduzindo. Por exemplo:

```
#: from plone_forms/content_status_history.pt
#.   <input attributes="tabindex tabindex/next;" value="Apply"
class="context" name="workflow_action_submit" type="submit" />
#.
#: from plone_forms/personalize_form.pt
#.   <input attributes="tabindex tabindex/next;" tabindex=""
value="Apply" class="context" type="submit" />
#.
msgid "Apply"
msgstr "Anwenden"
```

Nas duas partes do page template anterior, a palavra *Apply* será traduzida para *Anwenden* para os usuários Alemães. O que vai ser traduzido é determinado pelas tags *i18n* que foram inseridas nos page templates, como você viu anteriormente. Se você quiser alterar aquela tradução ou adicionar sua própria variação, então apenas mude o arquivo *.po*. Se não encontrar nenhum *msgstr*, então a tradução padrão em Inglês será encontrada. Depois que você fizer as alterações, reinicie o Plone. Quando isto acontece, o Plone recompilará aquele arquivo dentro da versão *.mo*, e a sua tradução será atualizada.

Para o Plone, se nenhuma linguagem é dada ou nenhuma tradução está disponível, a tradução padrão é sempre o arquivo que tem a tradução em Inglês. De fato, o arquivo *plone-en.po* está em branco, assim nenhuma tradução será disponibilizada. Por isto o Plone faz o fallback final e não traduz, e mostra o texto no page template. O texto em todos os page templates estão em Inglês pelo fato da maioria dos desenvolvedores falarem Inglês. Resumindo não há tradução para o Inglês.

Por isto, você pode fazer uma nova tradução copiando o arquivo *plone.pot* para um novo arquivo nomeando-o para *plone-xx.po*. O valor de *xx* deve ser o código do país que você está traduzindo. Você pode encontrar uma lista com os códigos das linguagens em *http://www.unicode.org/onlinedat/languages.html*. Depois que você começou a tradução, configure os valores no topo, incluindo o código da linguagem, e comece a traduzir. Se você fez um novo arquivo para a linguagem,

então a equipe do I18N do Plone ficará feliz em aceitá-la e vai ajudar você a completá-la. A lista da equipe do Plone está em *http://sourceforge.net/mailarchive/forum.php?forum_id=11647*

Traduzir o conteúdo que as pessoas adicionam é uma tarefa árdua e é algo em que o Plone está trabalhando, mas atualmente não se tem nada. A melhor abordagem até o momento é usar dois produtos, *PloneLanguageTool* e *i18nLayer*, ambos podem ser encontrados no SourceForge (*http://sf.net/projects/collective*). Entretando, ambos produtos são para desenvolvedores mais experientes que possam entender e integrar plenamente; espero que algo sobre isto esteja na próxima edição deste livro.

Exemplo: Mostrando Informação a Vários Usuários

No Capítulo 5 você usou comandos simples do TAL para mostrar as informações dos usuários mais detalhadamente. Aquele template tem alguns drawbacks; um deles é que ele mostra somente um usuário por vez. Você viu que um simples *tal:repeat* possibilita que você repita conteúdo, mas você usará agora uma macro para fazer esta página mais modular.

Você vai alterar o page template *user_info* para que ele liste todas páginas dos membros do site. Ao invés de procurar um usuário passando em um request, você vai usar a função *listMembers*, que retorna uma lista de todos os membros no site:

```
<div metal:fill-slot="main">
  <tal:block
   tal:define="
   getPortrait nocall: here/portal_membership/getPersonalPortrait;
   getFolder nocall: here/portal_membership/getHomeFolder
   ">
   <table>
     <tr tal:repeat="userObj here/portal_membership/listMembers">
         <metal:block
          metal:use-macro="here/user_section/macros/userSection" />
     </tr>
   </table>
 </tal:block>
</div>
```

Você vai ver que o código para *user_info* é agora menor. O membro retornado pelo *listMembers* é passado pelo *tal:repeat*. Para cada membro, há uma linha da tabela e então uma macro para mostrar a informação para o usuário. Naquela linha da tabela, a variável *userObj* definida localmente agora contém as informações do usuário. É claro que você precisa fazer uma macro chamada *userSection* em um page template, assim você criará um page template chamado *user_section* como referenciado na macro. Este template contém todo o código que estava entre as tags *row* da tabela. Novamente, você pode encontrar uma lista completa para este page template no Apêndice B:

```
<div metal:define-macro="userSection"
     tal:define="userName userObj/getUserName">
     ...
```

A única alteração real é que o *use-macro* no page template tem que ser removido e uma nova macro definida para que esta macro possa ser definida. Pelo fato do usuário não ser passado explicitamente, você precsa obter o usuário do objeto usuário usando o método *getUserName*. Para testar a página resultante, vá até *http://yoursite/user_info*, e você verá uma lista de usuários.

Agora a página é amigável, mostrando vários usuários em uma página. O código é mais modular, renderizando a informação do usuário em uma macro separada que pode ser alterada independentemente. Esta página não está ainda prefeita mas será melhorada nos capítulos posteriores.

Exemplo: Criando um Novo Portlet com Anúncios do Google

No Capítulo 4 você viu como editar facilmente as portlets em um site Plone; adicionar sua própria portlet não é difícil. Para escrever sua própria slot, você precisa fazer um novo page template com uma macro dentro dele. Então uma expressão TALES que aponta para a macro será adicionada para a lista do portlet, renderizando o portlet para a página.

O template básico para uma portlet é este:

```
<div metal:define-macro="portlet">
    <div class="portlet">
      <!-- Enter code here -->
    </div>
</div>
```

Tudo o que você precisa fazer é inserir algum código adequado dentro do portlet. O Google configurou um sistema de anúncios baseado em texto em 2003 que coloca texto em seu site. Os anúncios são baseados no que o Google acha que o seu site é sobre, baseado nos resultados das buscas para o seu site. O sistema do Google está disponível em *http://www.google.com/adsense*. Para mostrar os anúncios (e ser pago por eles), você terá que se registrar no Google. No site do Google, ele perguntará sobre cores e estilo. Depois você colocará isto em uma slot, eu recomendo o 'skyscraperâ€ sizeâ€”tall and thin. Faça uma cópia do JavaScript que o site produziu.

Depois, você tem que criar um portlet:

portal_skins/custom googleAds googleBox O resultado final deve ser algo parecido com a Listagem 6-1; entretanto, sua versão terá um valor válido para *google_ad_client*, ao invés de *yourUniqueValue*. Aquele valor diz ao Google que site pediu este anúncio e como pagar. Curiosamente suficiente, se você não tem um valor válido lá, o Google mostrará avocê assim mesmo os anúncios mas não pagará você!

Listagem 6-1. Mostrando Anúncios do Google

```
<div metal:define-macro="portlet">
    <div class="portlet">
<script type="text/javascript"><!--
google_ad_client = "yourUniqueValue";
google_ad_width = 120;
```

```
google_ad_height = 600;
google_ad_format = "120x600_as";
//--></script>
<script type="text/javascript"
  src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
</script>

   </div>
</div>
```

Para então incluir isto em seu site, como foi detalhado no Capítulo 4, adicione o seguinte portlet na sua lista de portlets:

```
here/googleAds/macros/portlet
```

Scripts Plone com Python

Até o momento existem quatro níveis diferentes no Plone para criar lógica. O nível mais simples para usar Python no Plone é a expressão do Python TALES que eu discuti no capítulo anterior. Entretanto, uma expressão Python permite que você faça somente uma linha de código; frequentemente você ia querer fazer algo mais complicado.

Ainda mais comum é o problema que você realmente não quer colocar toda a lógica no templage. Colocar lógica em seu template é uma má idéia em geral; na hora que você quer mover qualquer coisa que não está explícita na lógica de apresentação do template, você se livrou de uma dor de cabeça. Separar lógica e apresentação faz com que você permita que pessoas diferentes trabalhem em diferentes partes do projeto, e melhora o reuso de código. As outras camadas de adição de scripts no Plone acontecem aproximadamente nesta ordem:

- **Template attribute expressions**: Oferece expressões e uma maneira de inserir pequenos pedaços de lógica ou caminhos simples em muitos lugares.
- **Script (Python) objects**: Estes são simples scripts que executam no Plone em um ambiente restrito.
- **External method objects**: TEste são módulos mais complicados que não executam em ambientes restritos.
- **Python products**: Esta é a fonte chave na qual o CMF e o Plone estão escritos; isto oferece acesso para tudo no Plone. Produtos Python são um assunto avançado e são abordados no Capítulo 14.

Depois de uma expressãom, o próximo nível de complexidade é o objeto Script (Python). Este objeto permite que sejam feitas várias linhas de código Python, e você pode chamá-lo de uma expressão. Quando você chama um objeto Script (Python), você está sujeito a pequenas quantidades de overheads extras já que o Plone faz um chaveamento naquele objeto. Entretanto, este overhead é mínimo porque há uma troca entre clareza, separação e execução. Minha sugestão é colocar o máximo de lógica possível no Python e deixar os page tempates mais simples e mais claros possível. Assim é fácil voltar atrás se há ums queda na performance. mas agora você vai entender o que estava acontecendo.

Usando Objetos Script (Python)

Um objeto Script (Python) é o melhor que você pode pensar em scripts para Plone. É um pedaço de Python que você pode escrever e então chamar de outros templates ou diretamente pela Web. O Plone atualmente tem um grande número destes scripts para executar várias funções chave. Um Script (Python) é o meio termo entre uma expressão e um método externo em termos de capacidade.

Para adicionar um objeto Script (Python), vá até a ZMI, selecione Script (Python) do menu de adição, e clique Add, como mostra d Figura 6-2.

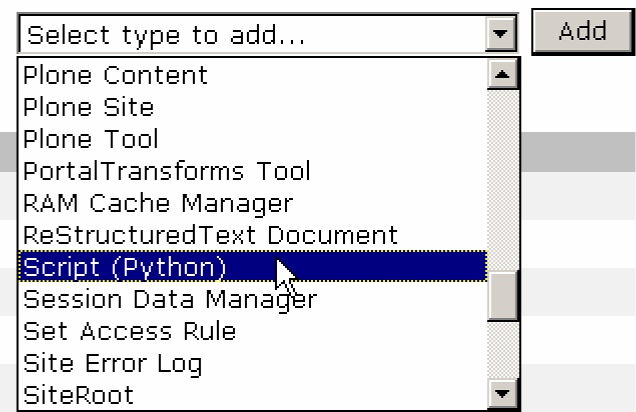

Figura 6-2. Adicionando um objeto Script (Python)

Dê ao script o ID **test_py** e então clique em Add and Edit. Abrirá a página de edição para o objeto Script (Python), que se parece como a Figura 6-3.

Figura 6-3. Editando um objeto Script (Python)

Você pode editar o script diretamente através da Web. Se você errar a sintaxe, você será avisado depois que clicar em Save Changes, como mostra a Figura 6-4.

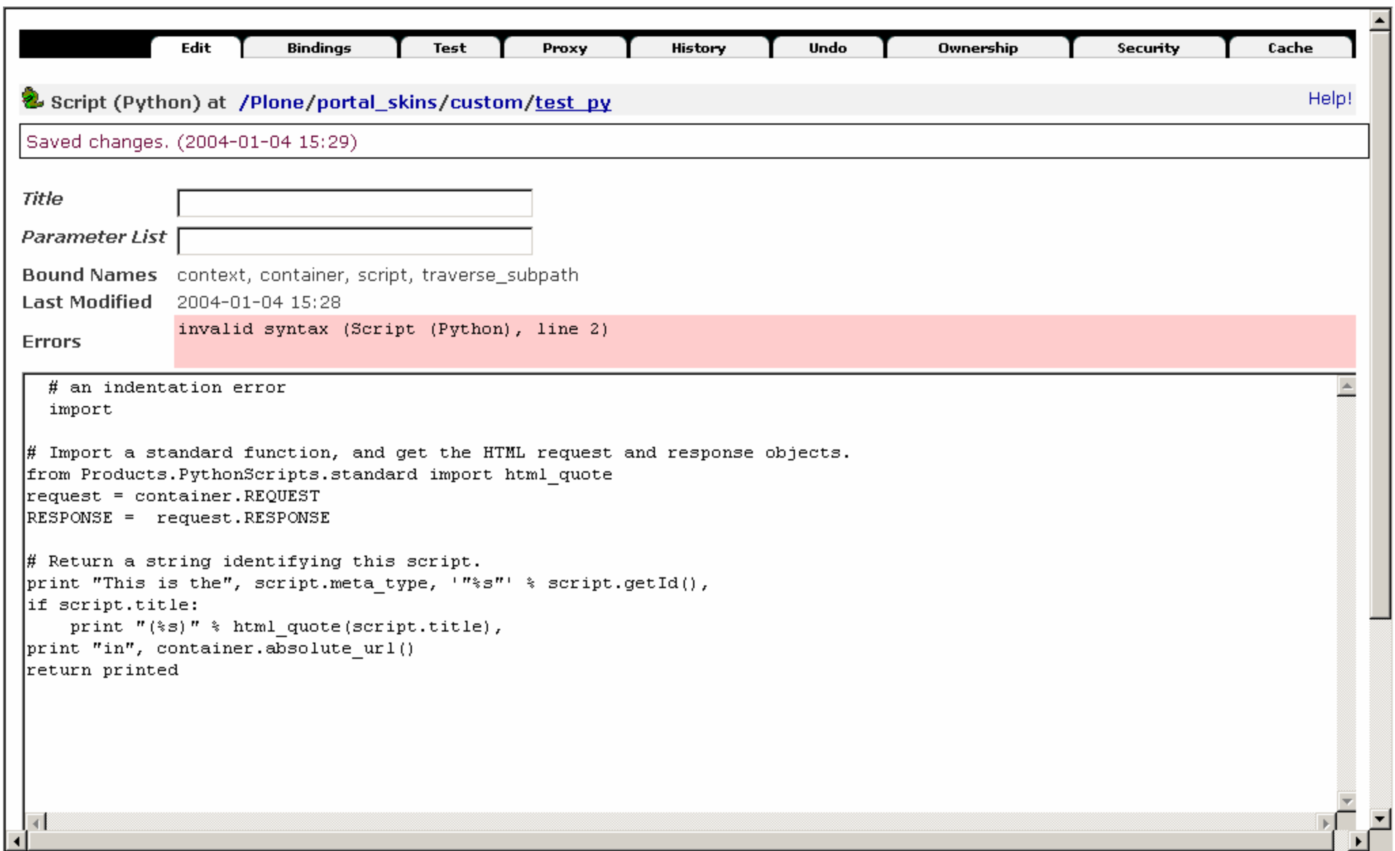

Figura 6-4. Um erro de identação premeditado noobjeto Script (Python)

Se o seu Script (Python) não tem erros, você pode clicar na aba Teste para ver a saída. Neste caso, o exemplo ainda é maçante; ele mostra o seguinte texto:

```
This is the Script (Python) "test_py" in
http://gloin:8080/Plone/portal_skins/custom
```

Um script também tem as seguintes opções:

Title: O formulário de edição tem uma opção Title, que é para você dar um título ao script. Ele será mostrado na ZMI, assim dever ser fácil para lembrar o que faz. **Parameter List**: Esta é uma lista de parâmetros que o script tem, como *variableA* ou *variableB=None*. De fato, esta é uma lista padrão de parâmetros que você espera em uma função padrão do Python.Alguns parâmetros já estão definidos para você neste objeto, entretanto; você pode vê-los clicando a aba Bindings. Nesta aba, você verá uma lista de variáveis que já estão limitados ao objeto, que devem ter nomes familiares.

A seguir temos as variáveis limitadas ao script que estão acessíveis em um objeto Script (Python):

- **context**: Este é o objeto no qual o script é chamado.
- **container**: Este é o objeto contido para este script.
- **script**: Este é o próprio objeto Script (Python); o equivalente no Zope Page Templates é o *template*.

- **namespace**: Este é por quem este script é chamado através do Document Template Markup Language (DTML), que é algo que não acontece no Plone.
- **traverse_subpath**: Esta é o caminho do Uniform Resource Locator (URL) depois do nome do script, que é uma característica avançada.

Eu vou mostrar agora um exemplo simples que liga estes tópicos no sistema de Zope Page Templates, usando o exemplo que eu dei de uma expressão Python no capítulo anterior que soma dois números. Como você viu, você pode fazer um page template para isto que se parece assim:

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

O equivalente usando um objeto Script (Python) se parece com o seguinte. Mude o script *test_py* para a seguinte linha:

```
return 1+2
```

Como você viu no começo do capítulo anterior, você chama um objeto dando seu caminho como uma expressão. Assim, no page template, você pode agora fazer o seguinte:

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

O objeto *test_py* é adquirido e chamado no caminho da expressão, e então retorna o Python de volta ao template e o imprime. Você agora chamou um script do seu template! Isto é obviamente um exemplo muito simples, mas o meu objetivo maior é dizer que você pode fazer muitas coisas em um objeto Script (Python) que você não pode fzer em um page template.

Em um objeto Script (Python), você pode especificar o título, parâmetros, e bindings setting usando a notação ## no topo de um script. Quando você salva um script com aquele pedaço de texto no topo, o Plone removerá aquelas linhas e em troca vai alterar o valor apropriado do objeto. Esta sintaxe é muito usada no objeto Script (Python) neste livro para termos certeza que você tem o título certo e os parâmetros. Assim, você pode reescrever o script anterior assim:

```
##title=Returns 1+2
##parameters=
return 1+2
```

Script do Plone

Scripting Plone is a rather complicated subject because as soon as you're able to script Plone, you have to take into account the Application Programming Interface (API) of all the objects and tools you may want to use. Explaining APIs is beyond the scope of this book; instead, I'll demonstrate how to do some simple tasks using Script (Python) objects. Once you're comfortable with them, I'll describe more API-specific functions.

Page templates podem ir pelos dicionários Python e listas muito bem. Mas se você não tiver dados em um destes formatos convenientes, então você precisa pular para um objeto Script (Python), formatar os dados, e então passá-los de volta para o page template.

O formato de dados mais conveniente é uma lista de dicionários, que combina o poder de um *tal:repeat* o os caminhos da expressão em uma função. Como exemplo, você verá uma função que pega uma lista de objetos. Cada um deste objetos é atualmente um objeto em uma pasta. Para cada um destes objetos, você verá se o objeto foi atualizado nos últimos 5 dias. A Listagem 6-2 mostra um portlet pequeno e útil que eu coloquei junto em um site que precisava localizar este tipo de informação e então marcar exatamente estes itens.

Listagem 6-2. Retornando Objetos com mais de Cinco Dias

```
##title=recentlyChanged
##parameters=objects
from DateTime import DateTime

now = DateTime()
difference = 5 # as in 5 days
result = []

for object in objects:
  diff = now - object.bobobase_modification_time()
  if diff < difference:
    dct = {"object":object,"diff":int(diff)}
    result.append(dct)

return result
```

Neste objeto Script (Python) eu introduzi alguns novos conceitos. Primeiro, você importa módulos de *DateTime* s=do Zope usando a função *import*. O módulo *DateTime*, abordado no Apêndice C, é um módulo que oferece acesso a datas. É muito simples, mas se você fizer um novo objeto *DateTime* sem parâmetros, então você vai obter a data e hora atuais; esta é a variável *now*. Quando você subtrai dois objetos *DateTime*, você terá o número de dias. Você pode dizer que a diferença (You can compare that to the difference a user wants to monitor and, if it's longer,) adicioná-lo a lista de resultado. O resultado disto é uma lista de objetos dicionário, que se parace com a Listagem 6-3.

Listagem 6-3. O Resultado da Litagem 6-2

```
[
  {
      'diff': 1,
      'object': <PloneFolder instance at 02C0C110>
  },
  {
l      'diff': 4,
      'object': <PloneFolder instance at 02FE3321>
  },
  ...
```

Agora que você tem os resultados certos, você precisa de um page template que passará na lista dos objetos e processará os resultados. Um exemplo disto:

```
<ul>
```

```
   <li tal:repeat="updated python:
context.updateScript(context.contentValues())">
```

Este template tem uma chamada *tal:repeat* no topo que chama o script (neste caso, chamado *updateScript*). Naquela função ele passa um valor, uma lista de *contentValues* do contexto atual. Antes você chamou o objeto Script (Python) usando um caminho da expressão; você pode fazer aquilo aqui com *context/updateScript*. Entretanto, você não pode passar parâmetros para o script chamado naquela sintaxe, assim você pode fazer uma expressão Python, que é *python: context.updateScript()*. A função *contentValues* retorna uma lista de todos os objetos de conteúdo em uma pasta. Olhe o código para cada interação:

```
   <a href="#"
      tal:attributes="href updated/object/absolute_url"
      tal:content="updated/object/title_or_id">
      The title of the item</a>
   <em tal:content="updated/diff" /> days ago
   </li>
 </ul>
```

Como foi mostrado, você pode ir por esta lista de valores, e você pode então usar caminhos de expressões para acessar primeiro o valor repetido (*updated*), então o objeto (*object*), e então um método daquele objeto (*title_or_id*). Este é um exemplo de como pegar o processamento da lógica complicada e passá-la para um objeto Script (Python).

Python Restrito

Eu mencionei várias vezes que os objetos Script (Python) e expressões TAL do Python rodam em modo *restricted Python*. Python restrito é um ambiente que tem algumas funções removidas. Estas funções podem ser potencialmente perigosas em um ambiente Web como o Plone. A razão original é que você pode ter usuários não confiáveis (mas autenticados) escrevendo Python em seu site. Se você abre uma conta para os muitos hosts free na Web para o Zope, você vai achar que você pode fazer isto. Entretanto, se você deu as pessoas direitos para fazerem isto, você não vai querer que elas acessem certas coisas como o sistema de arquivos.

No Python restrito, algumas funções comuns do Python foram removidas por razões de segurança, *dir* e *open* não estão disponíveis. Isto significa que, como os objetos Script (Python), eles não podem ser introspectivos, e o acesso é limitado ao sistema de arquivos. Alguns módulos do Python são disponibilizados ao usuário. A maioria destes são para desenvolvedores experientes; para mais informações, veja a documentação ou o código do módulo:

- **string**: Este é o módulo de strings do Python (*http://python.org/doc/current/lib/module-string.html*).
- **random**: Este é o módulo randômico do Python (*http://python.org/doc/current/lib/module-random.html*).
- **whrandom**: Este é o módulo *whrandom* do Python. Você deve usar *random* agora (*http://python.org/doc/current/lib/module-whrandom.html*).
- **math**: Este é o módulo de matemática do Python (*http://python.org/doc/current/lib/module-math.html*).
- **DateTime**: Este é o módulo de *DateTime* próprio do Zope.

- **sequence**: Este é um módulo do Zope para ordenar sequências facilmente.
- **ZTUtils**: Este é um módulo Zope que oferece várias utilidades.
- **AccessControl**: Este dá acesso ao módulo *Access* do Zope.
- **Products.PythonScripts.standard**: Este dá acesso às funções de processamento de string do DTML como *html_quote*, *thousands_commas*, e assim por diante.

Se você quer importar módulos que não estão na lista anterior, então você pode achar instruções excelentes no módulo *PythonScript*. Você achará eles em *Zope/lib/python/Products/PythonScripts/module_access_examples.py*. Entretanto, um método mais simples é disponibilizar a você um â€" usando um método externo.

Usando Objetos Método Exteno

Um *external method* é um módulo Python escrito no sistema de arquivos e então acessado no Plone. Pelo fato de ser escrito no sistema de arquivos, ele não roda em modo restrito do Python, e assim ele obedece as configurações padrão de segurança do Plone.

Isto significa que você pode escrever um script que faz tudo o que você quer e então chama-o de um page template. Tarefas comuns incluindo a abertura e fechamento de arquivos, acesso a processos e executáveis, e execução de tarefas no Plone ou no Zope que você simplesmente não pode executar de nenhuma outra maneira. Por razões óbvias, quando você está escrecendo um script que pode fazer isto, você pderisa ter certeza que você não está fazendo nada de perigoso, como escrever arquivo do senhas para seu servidor ou deletar um arquivo que você não quer deletar.

Para adicionar um método externo, você vai até o sistema de arquivos da sua página inicial da instância e procura o diretório *Extensions*. Neste diretório, adicione um novo arquivo de script Python; por exemplo, Figura 6-5 mostra que eu adicionei *test.py* a um diretório em meu computador Windows.

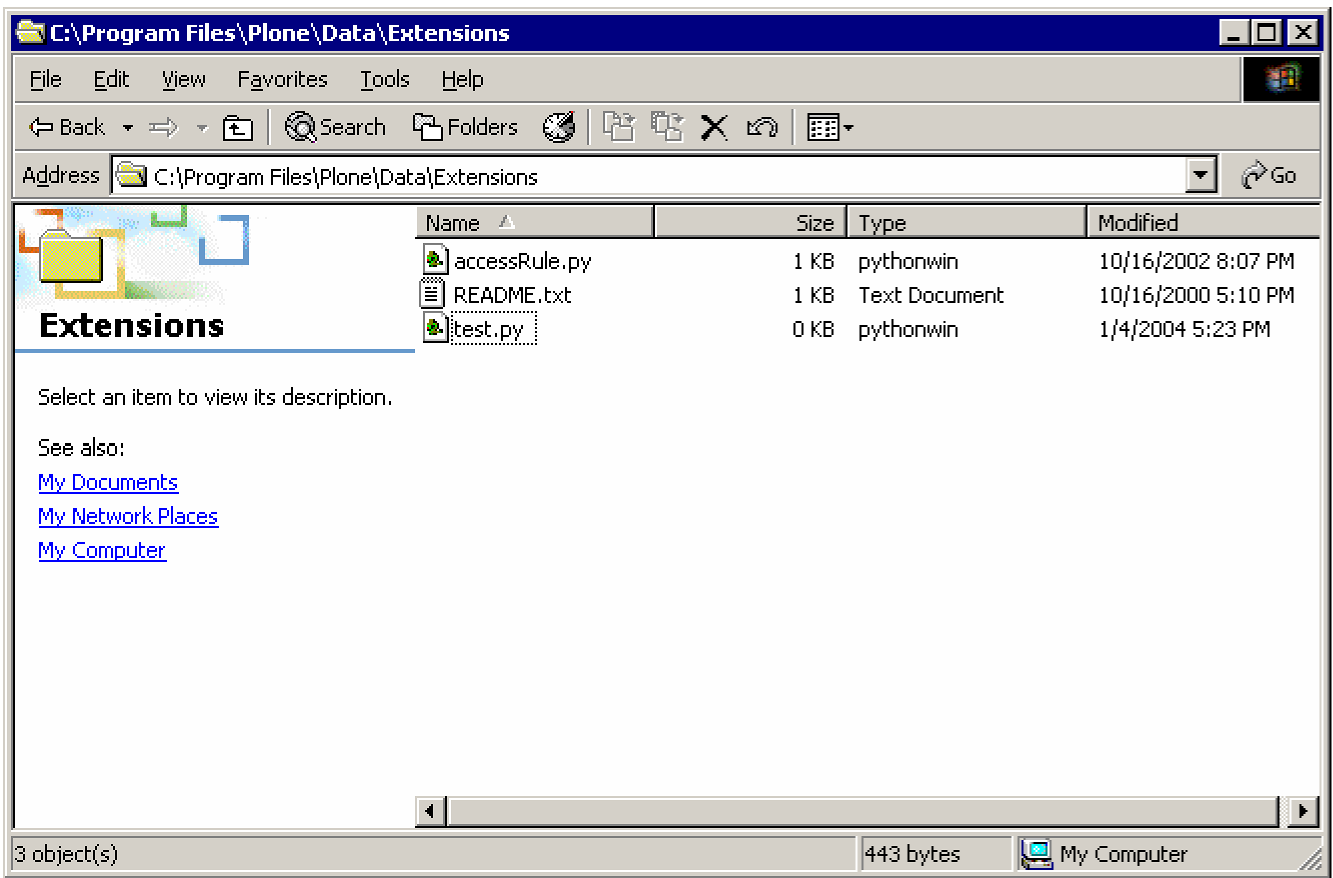

Figura 6-5. Um novo método externo, *test.py*

Você pode agora abrir *test.py* e editá-lo com seu conteúdo, escrevendo qualquer código Python que você quiser. A única restrição é que você deve ter uma função de entrada que tenha pelo menos um argumento, *self*. Este argumento é o objeto método externo no Plone que você irá adicionar brevemente. A seguir temos uma função de entrada como exemplo que lê o arquivo *README.txt* do mesmo diretório *Extensions* e devolve ao usuário (você terá que alterar o caminho que está apontando para seu arquivo):

```
def readFile(self):
   fh = open(r'c:\Program Files\Plone\Data\Extensions\README.txt', 'rb')
   data = fh.read()
   return data
```

Agora que você fez isto, você precisa mapear um método externo para este script. Este é um objeto do Zope, portanto retorne a ZMI, clique em *portal_skins*, e então clique em *custom*. Finalmente, selecione External Method da lista Add New Items. Quando você adiciona um método externo, você precisa dar o nome do módulo (sem o *.py*) e a função de entrada, assim neste caso o fomulário de adição se parece com a Figura 6-6.

Add External Method Help!

External Methods allow you to add functionality to Zope by writing Python functions which are exposed as callable Zope objects. The *module name* should give the name of the Python module without the ".py" file extension. The *function name* should name a callable object found in the module.

Id	test_external
Title	
Module Name	test
Function Name	readFile
	Add

Figura 6-6. O método externo recentemente adicionado

Depois de clicar em Save Changes, você pode clicar na aba Teste para ver o que acontece quando ele roda. Neste caso, você deve obter uma linha ou duas de texto. Depois de ter o módulo do External Method no Plone, você pode acessá-lo do page template da mesma forma que qualquer outro objeto. Um caminho da expressão para *here/test_external* deve fazer a mágica neste caso. Por exemplo:

```
<h1>README.txt</h1>
<p tal:content="here/test_external" />
```

O verdadeiro poder disto é que você pode passar o código para o mpdp irrestrito do Pyrhon e dali para qualquer função que você queira, sem ter que se preocupar com seguraça. Embora possa parecer como uma função controlada, métodos externos não são usados para grandes procedimentos no Plone porque a lógica complicada é geralmente passada em um objeto Product, e lógicas simples são matidas em um objeto Script (Python). Se você se encontra usando muito o objeto External Method, considere uma das ferramentas discutidas no Capítulo 12.

Dicas Úteis

Pelo fato de page templates serem Extensible Markup Language (XML) válida e poderem ser usados independentemente do Zope ou Plone, você tem vários scripts

úteis para limpar o código do page templatee executar checagem de sintaxe. Estas são as ferramentas e checagens adicionais; o Zope atualmente executa todas as checagens necessárias quando você carrega um page template. Para um projeto como o Plone, isto pode ser útil para rodar automaticamente as checagens em seu código ou verificá-lo localmente antes de salvar as alterações.

Para rodar estas checagens, você precisa ser capaz de editar estas ferramentas localmente e ter o Python instalado em seu computador. Para mais informações sobre Editores Externos, um método para editar código remoto localmente, veja o Capítulo 10.

Introdução aos Namespaces do XML

Page templates usam namespaces XML para gerar código. Os programadores podem usar as regras dos namespaces do XML para facilitar a vida. No topo do page template você verá uma declaração do namespace na tag inicial:

```
<html xmlns="http://www.w3.org/1999/xhtml"...
```

Isto configura o namespace padrão para Extensible HTML (XHTML). Para qualquer elemento contido, se o namespace não for definido, ele usa aquele namespace padrão. Por exemplo, você sabe que o próximo elemento é o XHTML porque não tem prefixo:

```
<body>
```

Normalmente para os elementos e atributos do TAL e METAL, você está adicionando o profixo *tal:* e *metal:* para definir o namespace. O código seguinte é algo que deve ser familiar agora:

```
<span tal:omit-tag="" tal:content="python: 1+2" />
```

This will render *3*. However, the following is an alternative:

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

Usando o o prefixo *tal:* no elemento, você definiu o namespace padrão para todo este elemento como *tal*. Se nenhum outro prefixo for dado, o namespace *tal* é usado. No exemplo, usando tags *span*, o namespace padrão é o XHTML, assim você terá que definir especificamente o prefixo *tal:* quando usando a aba Conteúdo.

Observe que o elemento nome é descritivo e pode ser que não tinha definido ainda o namespace *tal* (por exemplo, *content* ou *replace*). Porque *tal:number* não é um elemento XHTML válido, a tag atual não mostrará, mas o conteúdo willâ€” fazendo a *omit-tag* desnecessária. Esta técnica é muito usada no Plone para fazer código menor, mais simples de depurar e mais semântico.

Introdução ao Código Tidying

HTML Tidy é uma ferramenta excelente para teste e limpeza do código HTML que pode executar algumas tarefas úteis. Existem versões do HTML Tidy para todos os sistemas operacionais; você pode baixá-lo de *http://tidy.sourceforge.net*. Para os usuários Windows, procure o download apropriado para sua versão do Windows,

dezipe o arquivo *tidy.zip*, e coloque o *tidy.exe* no seu *PATH* (geralmente o diretório do seu Windows, *C:\up WINNT*).

O HTML Tidy pode dizer se há erros no XHTML do seu page template. Para estas finalidades, uma flag pode fazer a diferença: *-xml*. Isto diz para o HTML Tidy processar os arquivos como XML e reportar qualquer erro no XML. Dando o exemplo 'badâ€ o page template mostrado na Listagem 6-4, você pode ver alguns erros. Não somente a não indentação do código, mas é esquedido de fechar oe elementos e tem conjuntos de tags inválidos.

Listagem 6-4. Um exemplo de Page Template Quebrado: *bad_template.pt*

```
<!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">
<head>
<title></title>
</head>
<body>
<p>
<div>
This is bad HTML,
XHTML or XML...<a tal:contents="string: someUrl"></a>
</p>
<img>
Further it isnt indented!
</body>
</html>
```

Se você todar a Listagem 6-4 com o HTML Tidy, você verá os erros no template e obterá uma boa indentação do código, como mostra a Listagem 6-5.

Listagem 6-5. A saída do HTML Tidy

```
$ tidy -q -i bad_template.pt
line 11 column 1 - Warning: <img> element not empty or not closed
line 10 column 1 - Warning: missing </div>
line 10 column 39 - Warning: <a> proprietary attribute "tal:contents"
line 11 column 1 - Warning: <img> lacks "alt" attribute
line 11 column 1 - Warning: <img> lacks "src" attribute
line 9 column 1 - Warning: trimming empty <p>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta name="generator" content=
  "HTML Tidy for Linux/x86 (vers 1st August 2003), see www.w3.org" />

  <title></title>
</head>

<body>
```

```
  <div>
    This is bad HTML, XHTML or XML...<a tal:contents=
    "string: someUrl"></a> <img />Further it isnt indented!
  </div>
</body>
</html>
```

As queixas sobre atributos proprietários pode ser um pouco aborrecido. Para checar que seu page template é um XML válido, passe a flag *-xml*. A saída é menos verbosa e apenas aponta as tags predidas:

```
$ tidy -q -xml bad_template.pt
line 15 column 1 - Error: unexpected </body> in <img>
line 16 column 1 - Error: unexpected </html> in <img>
```

Conduzindo Checagens de Sintaxe

Quando você edita um page template na ZMI, o Zope executa uma checagem na sintaxe no documento para verificar tags inválidas. Se uma tag é inválida, um erro aparecerá no template enquanto você estiver editanto ele através da Web. Se como eu (e como demostro no Capítulo 7), você escreve a maioria dos seus page templates no sistema de arquivos, então uma simples checagem na sintaxe é relamente útil. A Listagem 6-6 é um Python que está no seu sustema de arquivos e roda independentemente do Zope.

Para rodar isto, você dever ter um interpretador Python, e o módulo do Python *PageTemplate* deve ser importável. Para fazer o *PageTemplate* importável para o seu interpretador Python, você deve adicionar o diretório *Products* da sua instalação Zope no caminho do Python. Você tem várias maneiras de fazer isto (abordado no Apêndice B).

Listagem 6-6. Checagem de Erro no Page Template

```
#!/usr/bin/python
from Products.PageTemplates.PageTemplate import PageTemplate
import sys

def test(file):
    raw_data = open(file, 'r').read()
    pt = PageTemplate()
    pt.write(raw_data)
    if pt._v_errors:
        print "*** Error in:", file
        for error in pt._v_errors[1:]:
            print error

if __name__=='__main__':
    if len(sys.argv) < 2:
        print "python check.py file [files...]"
        sys.exit(1)
    else:
        for arg in sys.argv[1:]:
            test(arg)
```

Para cada arquivo passado pelo script, a ZMI vai compilar o page template e ver se há algum erro na TAL. Usando o arquivo *bad_template.pt* da Listagem 6-4, você obterá um erro:

```
$ python zpt.py /tmp/bad_template.pt
*** Error in: /tmp/bad_template.pt
TAL.TALDefs.TALError: bad TAL attribute: 'contents', at line 10, column 39
```

Neste caso, ele pegou a ortografia incorreta do *tal:content* como *tal:contents*. Este erro não é pego pelo HTML Tidy. Infelizmente, o processo para no primeiro erro de sintaxe. Se há mais erros, somente o primeiro é visto, significando que você terá que checar a sintaxe várias vezes.

Usando Formulários

Os formulários são a parte integrla de qualquer site, e quase todos precisam criar um método para criar e alterar formulário no seu site Plone. Com o framework de formulário no Plone, você pode alterar a validação que processa os formulários, onde eles pegam o usuário, e assim por diante. Este framework não é especificamente desenvolvido para formulários stand-alone que executam tarefas simples, como uma requisição de senha, login, e assim por diante. O framework também funciona para todos os tipos de conteúdo para tarefas como edição de um tipo de conteúdo, que será abordado mais tarde neste livro nos Capítulos 11â€" e 13.

Todos os formulários básicos têm ao menos dois componentes que você já viu antes: um objeto Page Template para mostrar o formulário para o usuário, e um objeto Script (Python) para verificar os resultados executar algumas ações sobre os resultados.

O formulário de controle do framework no Plone introduz alguns tipos novos de objetos que são equivalentes aos tipos que você viu neste capítulo. Este são o objeto de Controle dos Page Templates, o objeto de Controle do Script, e o objeto de Controle do Validador. Estes novos objetos têm seus objetos equivalentes, que são mostrados na Tabela 6-1. Estes novos objetos têm mais propriedades e ações um pouco diferentes dos objetos equivalentes.

Tabela 6-1. Novos tipos de Objetos que os Controladores Oferecem

Tipo de Objeto Equivalente ao Objeto no Zope

Controller Filesystem Page Template Page Template

Controller Python Script Python Script

Controller Validator Python Script

Para adicionar um destes objetos usando a ZMI, vá até a caixa de seleção, e selecione o nome.

O controlador do framework cria uma sequência de eventos para um formulário que um usuário pode então definir. A seguir está a sequência de eventos quando um formulário é executado:

Quando esta sequêncua de eventos ocorre, um objeto estado é passado, ele contém informações sobre o status do objeto, o sucesso de qualquer validação, e qualquer mensagem que era para ser passada.

As sessões seguintes rodam por estes passos para mostrar como um formulário pode ser validado, e então mostrarão um exemplo completo no Exemplo do E-Mail: Enviando um E-Mail para o Webmaster â€ sessão.

Criando Formulário de Exemplo e Scripts Associados

O início deste processo é um formulário. Apesar deste ser atualmente um objeto Controlador de Page Template, é escrito usando código TAL padrão. Para adicionar um, selecione um Page Template da, agora familiar, caixa de seleção e dê a ele o ID **test_cpt**.

Um formulário no Plone é atualmente um pedaço um tanto longo de código se você quer utilizar todas as opções disponíveis para você. Este pedaço de código é reproduzido completamente no Apêndice B e é o código usado no exemplo posterior:

```
<form method="post"
      tal:define="errors options/state/getErrors"
      tal:attributes="action template/id;">
    ...
    <input type="hidden" name="form.submitted" value="1" />
</form>
```

Olhando para este código, você deve notar que para funcionar em um framework, existem diferenças muito pequenas entre este e aqthis and whuele que você pode considerar um formulário padrão. Primeir, o formulário é configurado para submeter-se a ele mesmo; *não é* opcional. Segundo, existe uma variável especial escondida chamada *form.submitted*.

O objeto Controlador do Page Template checa a variável de requisição para o valor *form.submitted* para ver se o formulário foi submetido. Se, ao invés, ele foi apenas acessado dâ€” por exemplo, via linkâ€" ele *não é *opcional. No início do formulário, você pode configurar as variáveis de erro. O dicionário de *errors* vem do objeto estado que é passado nos templates. O objeto *state* é um objeto comum para todos os templates e scripts neste sistema.

Criando Validadores

Depois que o usuário clicar o botão Submit no seu formulário, os dados rodarão através dos validadores e serão validados. Validadores são opcionais. Os Dados não necessitam ser validados, mas é claro que qualquer aolicação deve fazer o que é mais apropriado. A aba Validator para um objeto Controlado do Page Template dá um link para os validadores possíveis.

Um script de validação é o mesmo que um objeto normal Script (Python) que tem uma variável extra, *state*. A variável *state* é como você faz para passar resultados da validação. A Listagem 6-7 mostra um script simples de validação para checar se você deu um número.

Listagem 6-7. Validandno Se um Número Foi Passado

```
##title=A validation script to check we have a number
##parameters=
num = context.REQUEST.get('num', None)
try:
    int(num)
except ValueError:
    state.setError("num", "Not a number", new_status="failure")
except TypeError:
    state.setError("num", "No number given.", new_status="failure")
if state.getErrors():
    state.set(portal_status_message="Please correct the errors.")
return state
```

Este objeto *state* contém informações básicas sobre o que está acontecendo durante o processo de validação. O objeto *state* armazena os erros para cada campo, o status, e qualquer outro valor. Por exemplo, se o número dado não pode ser tranformado em um inteiro, você configura o status para failure (fracasso) e dá uma mensagem de erro para o campo usando o método *setError*. Depois esta mensagem de erro será mostrada para o campo. No final do script, qualquer erro retornado até agora são recuperados pelo método *getErrors*.

Para adicionar o script anterior, clique em *portal_skins*, clique em *custom*, e selecione Controller Validator da caixa de seleção. Digite o ID **test_validator**. Você pode retornar agora para a aba Validação do seu objeto Controller Page Template e adicionar um apontamento para este script de validação, como mostra a Figura 6-7.

Figura 6-7. Adicionando o *test_validator* para o objeto Controller Page Template

Você teve algumas escolhas para a validação. No exemplo você ignorou-as porque elas não eram relevantes, mas a seguir temos uma lista de opções:

contextType: Este é o tipo do contexto do objeto, no qual o template foi executado. Este é um shortcut para o tipo de conteúdo do contexto do objeto. Se você precisava somente desta validação para acontecer em um link, então você deve configurar este valor no Link.

button: Este é o botão, onde clicamos para submeter um formulário. Você deve ter diferentes botões em um formulário (por exemplo, um botão Submit e um Cancel). Cada um deste botões devem então mapear uma ação diferente; se clicar em Cancel você deve ser levado a um lugar, e clicando Submit a outro.

validators: Esta é uma lista separada em vírulas de validadores, que são objetos Controller Validator que o template adquirirá. No exemplo anterior, você usou o ID validador de *test_validator*.

NOTA Quando escrever um script de validação, use o objeto Controller Validator ao invés de objetos Script (Python). Objetos Controller Validator são como objetos Script (Python) comuns com a adição de uma aba Ações da ZMI.

Especificando Ações

Ações são as ações finais depois dos validadores terem sido rodados, e elas dependem do status que é retornado pelos validadores. A aba Ações para um objeto Controller Page Template mostra todas as ações para o page template em questão. Você pode especificar ações como o mesmo tipo de opções de especialização como descrito anteriormente através de um formulário da Web, como mostrado na Figura 6-8.

Figura 6-8. Adicionando uma Ação

Você tem as quatro escolhas a seguir para a ação resultante atual:

redirect_to: Isto redireciona para a URL especificada no argumento (uma expressão TALES). A URL pode ser tanto absoluta como relativa.

redirect_to_action: Isto redireciona para a ação especificada no argumento (uma expressão TALES) para o conteúdo do objeto atual (por exemplo, *string:view*). Neste estágio eu não abordei as ações ainda, mas cada conteúdo do objeto tem ações como visualização e edição. O Capítulo 11 aborda as ações para um objeto.

traverse_to: Isto traverses para a URL especificada em um argumento (uma expressão TALES). A URL pode ser tanto absoluta ou relativa.

traverse_to_action: Isto traverses para a ação especificada no argumento (uma expressão TALES) para o conteúdo do objeto atual (por exemplo, *string:view*).

Um exemplo disto é se a conclusão do formulário é um sucesso, você traverse para um objeto Controller Python Script que você escreveu, que processa o resultado do formulário. Se a página é uma falha, você traverse back ao template e mostra o erro.

A diferença entre um redirect e um traversal é que o redirect é um redirecionamento HTTP enviado para o browser do usuário. O borwser processa-o e então envia o usuário para a próxima página. Assi, as ações de redirecionamento perdem todos os valores passados por uma requisição original.Se você precisa examinar os conteúdos do formulário orignal, então esta não é a melhor solução. Ao invés, eu recomendo a opção traversal. O resultado é o mesmo; mas a opção traversal faz tudo isto no servidor. Assim preserva as variáveis da requisição e permite que você exemine elas em um script.

Exemplo de E-Mail: Enviando E-Mail para o Webmaster

Agora você vai ver um exemplo real e dispender o resto deste capítulo constuindo-o. O que queremos é um formulário que envia e-mail para o Webmaster. Você construirá este tipo de formulário nas seguintes sessões. Os scrips completos, page templates, e código associado estão diponíveis no Apêndice B. Se você realmente não quer digitar todos eles, você pode ver este exemplo online no site do livro; ele pode ser baixado coimo um arquivo comprimido do site do livro do Plone (*http://plone-book.agmweb.ca*) e do site da Apress (*http://www.apress.com*), Assim você oide aoenas instalá-lo e executá-lo. Este exemplo tem somente dois campos no formulário: o e-mail da pessoa que submeterá o formulário e alguns comentários daquela pessoa. Para este formulário, o e-mail da pessoa deve ser obrigatório assim você pode responder os seus comentários.

Construindo o Formulário

O formulário é a maior parte e a mais complicada deste procedimento, porque há tanto trabalho a ser feito para suportar o tratamento de erros. Este formulário é um objeto Controller Page Template chamado *feedbackForm*. Para assegurar que

ele está dentro do template principal, eu vou começar o formulário no método padrão:

```
<html
    xmlns="http://www.w3.org/1999/xhtml"
    xml:lang="en-US"
    lang="en-US"
    i18n:domain="plone"
    metal:use-macro="here/main_template/macros/master">
  <body>
    <div metal:fill-slot="main"
         tal:define="errors options/state/getErrors;">
```

Um adicional aqui é o *errors options/state/getErrors*, que que colocará todo e qualquer erro na variável de erro local para ser usada mais tarde.

Por causa da obrigação de o formulário ter que postar de volta o e-mail, você configura esta ação no TAL, com a expressão *template/id*. Este caminho vai publicar o ID do template e inserí-lo em uma ação, assim este caminho sempre vai funcionar, até mesmo se você renomear o template. Observe que você está também adicionando as tags *i18n* que você viu anteriormente, para assegurar que este formulário pode ser localizado:

```
<form method="post"
      tal:attributes="action template/id;">

<legend i18n:translate="legend_feedback_form">
    Website Feedback
</legend>
```

A seguir está o começo da linha para o endereço de e-mail. Você definirá uma variável aqui chamada *error_email_address* que está configurada para uma string de erro se houver uma string adequada nos dicionários de *erros*. Aquele valor do erro gerado pelo validador deve ser um erro:

```
<div class="field"
     tal:attributes="class python:test(error_email_address,
                                  'field error', 'field')">
     tal:define="error_email_address errors/email_address|nothing;">
```

A seguir temos a etiqueta para o campo de endereço de e-mail. Esta etiqueta incluirá um *div* para o texto de ajuda. O elemento *span* se transformará no familiar ponto vermelho próximo a etiqueta que o usuário sabe que é obrigatório:

```
<label i18n:translate="label_email_address">Your email address</label>
<span class="fieldRequired" title="Required">(Required)</span>
<div class="formHelp"
     i18n:translate="label_email_address_help">
     Enter your email address.
</div>
```

Aqui você adicionará o elemento atual:

```
     <div tal:condition="error_email_address">
         <tal:block i18n:translate=""
                    content="error_email_address">Error
         </tal:block>
     </div>
     <input type="text" name="email_address"
            tal:attributes="tabindex tabindex/next;
                            value request/email_address|nothing" />
 </div>
```

No topo deste bloco, você testa para ver se há um erro. Se há, a classe para o elemento alterado será a classe *field error* ; esta classe mostrará uma bela caixa laranja ao redor do campo. Depois, se um erro ocorreu neste campo (como você já testou), a mensagem correspondente será mostrada. Finalmente, você mostrará o elemento do formulário, e se já houver um valor para *email_address* na requisição, você colocar no formulário aquele valor.

A *tabindex* é uma ferramenta muito útil no Plone. Ela contém um número sequencial que é incrementado para cada elemento, e cada vez ele configura um novo valor HTML para *tabindex* para cada elemento em um formulário. Este é uma ótima característica da interface do usuário; significa que cada elemento do formulário pode ser movido com segurança sem ter que se preocupar em lembrar os números das *tabindex* porque vai acontecer automaticamente.

É muito trabalho para um elemento, mas é um código mais boilerplate; você pode facilmente copiá-lo ou mudá-lo. Você pode encontrar o restante co formulário no Apêndice B.

Criando um Validador

No exemplo você tem somente um elemento obrigatório (o e-mail), assim ele é um simples pedaço de Python chamado *validEmail.vpy* que faz o trabalho. Os conteúdos deste script são os seguintes:

```
 email = context.REQUEST.get('email_address', None)
 if not email:
     state.setError('email_address', 'Email is required',
                    new_status='failure')
 if state.getErrors():
     state.set(portal_status_message='Please correct the errors.')
 return state
```

Se nenhum endereço de e-mail for encontrado, este script adiciona um erro ao dicionário de erros com a chave do *email_address* e uma mensagem. Esta chave é usada no page template para ver se um erro ocorreu naqiele campo em particular.

Processando o Script

Este exemplo tem um simples script para o e-mail que pega os valores (que já estão validados) e forma um e-mail fora deles. Este é um objeto Controller Python Script; é como um objeto padrão Script (Python) exceto por ele ter uma variável *state*, e, como o Controller Page Template, você pode dar a ela ações para quando ela ter sucesso:

```
mhost = context.MailHost
emailAddress = context.REQUEST.get('email_address')
administratorEmailAddress = context.email_from_address
comments = context.REQUEST.get('comments')

# the message format, %s will be filled in from data
message = """
From: %s
To: %s
Subject: Website Feedback

%s
URL: %s """

# format the message
message = message % (
    emailAddress,
    administratorEmailAddress,
    comments,
    context.absolute_url())

mhost.send(message)
```

Você viu agora um simples script para enviar e-mail. Este é um script comum que você verá várias vezes. Basicamente, o objeto *MailHost* no Plone pegará um e-mail como uma string, enquanto aja de acordo com a especificação Request for Comment (RFC) para o e-mail que tem os endereços *From* e *To*.

Neste e-mail, você tem o endereço do administrator que você especificou na configuração do portal e enviou o e-mail para aquela pessoa. A única parte extra neste script é a adição da configuração do estado. Isto vai configurar uma mensagem que oferece um feedback para o usuário:

```
screenMsg = "Comments sent, thank you."
state.setKwargs( {'portal_status_message':screenMsg} )
return state
```

Juntando as Três Partes

Até o momento, entratanto existem três entidades separadas: um formulário, um validador, e um script para a ação. Eles precisam ser colocados juntos para formar um laço, então você vai voltar ao objeto Controller Template. Clique na aba Validação, e digite um novo validador que aponta para o script *validEmail*. Você também vai adicionar uma ação de sucesso se o processamento estiver correto para traverse ao script *sendEmail*. No script *sendEmail*, você pode agora adicionar outro traversal de retorno ao *feedbackForm* que depois que *sendEmail* funcionar corretamente, o usuário será enviado para a página original.

NOTA Um script de validação de e-mail mais completo aparece no Plone, o *validate_emailaddr*, que checa se o e-mail está em um formato correto. Se você quer usar este script, você pode apontar o validador para este script.

Está pronto! Você deve agora ser capaz de testar o formulário no site do livro. Para ficar mais fácil, eu fiz uma aba Feedback, que aponta para o template *feedbackForm*, e de lá você pode me dar feedbacks sobre este livro!