

Zope Page Templates

Pablo Ambrosio pandres@menttes.com

Emanuel Sartor emanuel@menttes.com

Función

Zope favorece la separación entre presentación y lógica proveyendo objetos destinados expresamente a la presentación.

Definimos como presentación la tarea de definir dinámicamente la estructura de las páginas web. Los objetos de presentación generan código HTML.

Zope provee dos métodos para presentación Zope Page Templates (ZPT) y Document Template Markup Language (DTML).

ZPT apunta a directamente a generar código HTML válido insertando código XML para definir el comportamiento dinámico.

DTML también permite generar páginas web insertando tags especiales, pero estas no son HTML válido.

¿Que son?

Los Zope Page Templates son documentos XHTML, lo que significa que pueden ser vistos y editados usando herramientas compatibles con XHTML.

ZPT (Zope Page Templates) esta basado en los lenguajes TAL (Template Attribute Language) y METAL (Macro Expansion Template Attribute Language), ambos son específicos de Zope. estos lenguajes proveen facilidades para extraer elementos de la base de datos y mostrar sus propiedades en HTML.

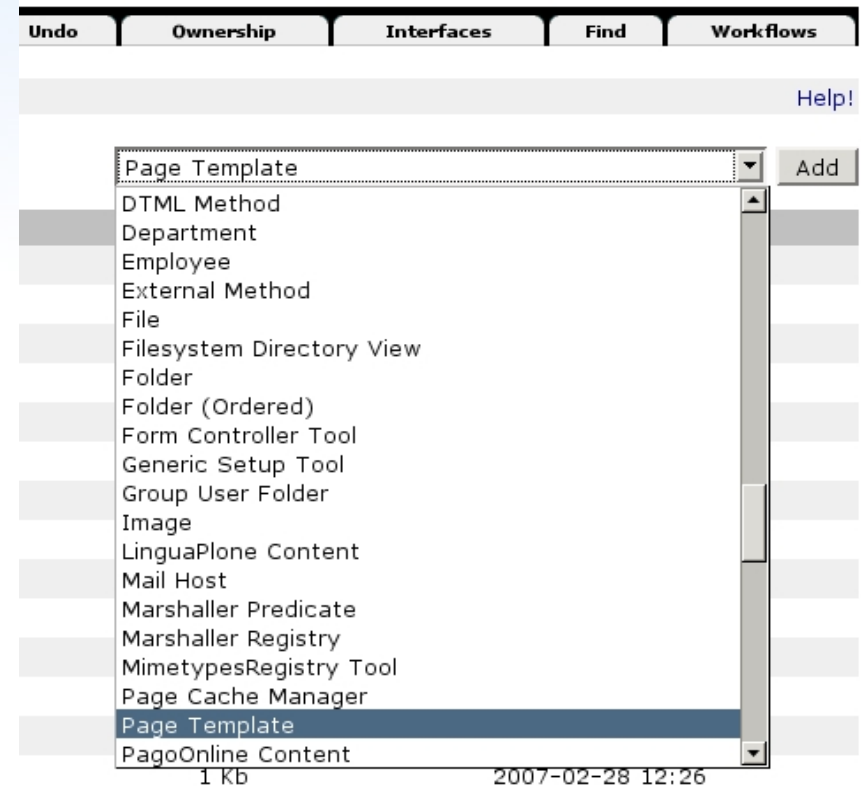
La implementación de ZPT es la misma tanto para Zope 2 como para Zope 3.

Creando un Page Template

Crear una carpeta “Demo” en el *root*.

Ingresar a “Demo” y crear un *Page Template* eligiendo el item en la lista desplegable de arriba a la derecha.

Elegir el *id* (ej: demo) y hacer click en *Add* o *Add and Edit* para acceder al cuadro de edición.



Add Page Template

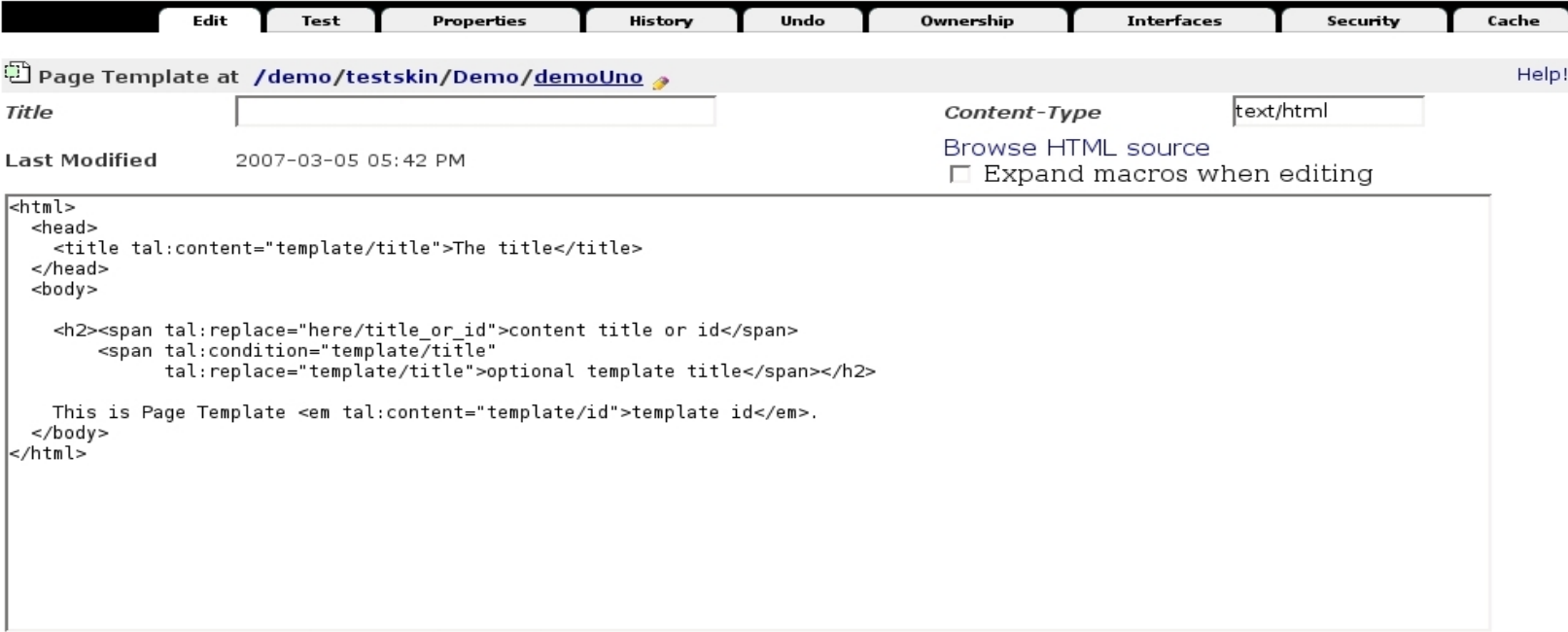
Page Templates allow you to use simple HTML or XML attributes local file by typing the file name or using the *browse* button.

Id

File

Editando un Page Template

Luego de hacer *click* en “Add and Edit”, o haciendo *click* sobre un *page template* preexistente en la *folder* donde se esta trabajando ingresamos a la pantalla de edición *web*. Si no se ha ingresado código luego de la creación la interfaz mostrará un esqueleto de código básico sobre el que se puede comenzar a trabajar.



The screenshot shows the 'Page Template at /demo/testskin/Demo/demoUno' editor. The interface includes a menu bar with options: Edit, Test, Properties, History, Undo, Ownership, Interfaces, Security, and Cache. Below the menu, there is a title input field, a 'Content-Type' dropdown set to 'text/html', and a 'Last Modified' timestamp of '2007-03-05 05:42 PM'. A 'Browse HTML source' link and an 'Expand macros when editing' checkbox are also present. The main area is a code editor containing the following HTML skeleton:

```
<html>
<head>
  <title tal:content="template/title">The title</title>
</head>
<body>
  <h2><span tal:replace="here/title_or_id">content title or id</span>
    <span tal:condition="template/title"
      tal:replace="template/title">optional template title</span></h2>
  This is Page Template <em tal:content="template/id">template id</em>.
</body>
</html>
```

At the bottom of the editor, there are buttons for 'Save Changes', 'Taller', 'Shorter', 'Wider', and 'Narrower'. Below the editor, a note states: 'You can upload the text for demoUno using the following form. Choose an existing HTML or XML file from your local computer by clicking *browse*. You can also click here to view or download the current text.'

Una vez realizados los cambios que se requieran hacer *click* en el botón “Save Changes” para que estos tengan efecto.

Ejercicio: Editar un Page Template

- Crear un nuevo *Page Template*.
- Editarlo, reemplazando el texto por defecto por algún código HTML simple, como el siguiente:

```
<html>  
  <body>  
    <h1>¡Hola Mundo!</h1>  
  </body>  
</html>
```

Subiendo un Page Template

Si se quiere editar un template desde un editor externo o simplemente cargar el código del *template* desde el sistema de archivos, desde la página de creación se puede hacer *click* en el botón “Browse“, desde donde elegiremos el archivo a agregar.

Add Page Template

Page Templates allow you to use simple HTML or XML attributes local file by typing the file name or using the *browse* button.

Id

File

Subiendo un Page Template

You can upload the text for demoDos using the following form. You can also [click here](#) to view or download the current text.

File

Encoding

En la parte inferior de la página de edición se puede acceder a la opción de navegar el sistema de archivos desde donde elegiremos de donde tomar el código para el *template*, una vez elegido se debe hacer *click* en “Upload File”.

Ejercicio: Subir un Zope Page Template

Crear con su editor preferido el siguiente archivo demoUno.pt:

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

Este es el Page Template `id`.

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

y subirlo al sitio Plone en la carpeta /demo.

Vista de un Page Template

Un *Page Template* puede ser visto haciendo *click* en la pestaña (*tab*) *Test* desde la pantalla de edición del *template*.



También puede ser visto visitando su URL de Zope asociado, ej:
<http://misitio/puntodemontaje/miportal/micarpeta/mitemplate> .

Algunas propiedades de los Page Template

El mecanismo de XML y HTML de Zope:

- conserva el *template* como código XML bien formado
- intenta ser no invasivo usando atributos registrados como *namespaces* (TAL)
- provee soporte para macros (METAL)
- provee soporte de para internacionalización

Template Attribute Language (TAL)

El **Template Attribute Language (TAL)** es un lenguaje expresado como atributos en las etiquetas (*tags*) HTML.

Las etiquetas tienen la forma:

```
<p tal:comando="expresion">Texto</p>
```

Todas las declaraciones en TAL consisten de atributos en etiquetas cuyos nombres comienzan con “tal:” , y todas tienen valores asociados, que siempre van entre comillas.

Estas etiquetas son código HTML válido, o sea que estos documentos se pueden editar con cualquier editor de HTML.

Template Attribute Language Expression Syntax (TALES)

Las expresiones TALES evalúan código de un tipo específico y devuelven su resultado. Las distintas expresiones son manejadas por el motor de TALES. En una expresión TALES su tipo se especifica al comienzo. Estas son algunas de las expresiones mas comunes:

- “path: ” - Toma una cadena con pinta de URL para evaluarla y recorrerla hasta obtener un objeto. Es la expresión por defecto en el motor de TALES.
- “string: ” - Devuelve un *string*, pero tambien puede interpretar un *path* interpolado.
- “python: ” - Devuelve el resultado de una expresión *python*.
- “not: ” - Si el resultado de la expresion evaluada es un booleano, niega ese resultado.
- “exists: ” - Determina si un objeto existe dado un *path*.

Algunas Expresiones Simples

La expresión "template/title" en nuestro ejemplo de edición de *Page Templates* es de tipo *path*. Este es el tipo más común de expresión. Existen varios otros tipos de expresiones definidas en la especificación de *TAL Expression Syntax* (TALES).

La expresión "template/title" obtiene la propiedad *title* del *template*. Estas son algunas otras expresiones de tipo *path* comunes:

- 'request/URL': EL URL del *request* actual.
- 'user/getUserName': El nombre de *login* del usuario autenticado.
- 'container/objectIds': Una lista de los Ids de los objetos en la misma carpeta que el *template*.

Todos los *path* comienzan con un nombre de variable. Si la variable contiene el valor buscado, podemos detenernos allí. Si no, agregamos una barra (/) y el nombre del sub-objeto o propiedad. Habitualmente se recorre un camino de varios sub-objetos hasta llegar al valor deseado.

Insertando Contenido

Para insertar texto dinámico dentro de otro texto, típicamente se usa *tal:replace* en un *tag span*. El comando *replace* reemplaza el *tag* donde esta con el resultado de evaluar su expresión.

Agregemos las siguientes líneas a nuestro template de ejemplo:

```
<br>
```

```
El URL es <span tal:replace="request/URL">URL</span>.
```

El *span tag* es estructural, no visual, luego esto lucirá como "El URL es URL." si vemos el código en un navegador. La version renderizada se verá como:

El URL es <http://localhost:8080/demoUno>.

Insertando Contenido

Hay que tener cuidado cuando se edita de no romper el *tag span* o colocar *tags* de formato como *b* o *font* dentro de él, ya que estos también serían reemplazados.

Si se quiere insertar texto en un *tag* pero dejar el propio *tag* intacto, se puede usar el comando *tal:content*. Para mostrar el título de nuestro ejemplo como la propiedad título del *template*, agregamos las siguientes líneas a nuestro código:

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

Si abrimos el *tab* "Test" en una nueva ventana podremos ver los cambios.

Estructuras de Repetición

Agregemos un poco más de contenido a nuestro *template*, por ejemplo, una lista con los objetos que está en la misma carpeta que nuestro *template*. Escribamos una tabla con filas numeradas por cada objeto y columnas con el id, el meta-type y el título. Agregemos estas líneas al final de nuestro *template*:

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

Estructuras de Repetición

El comando *tal:repeat* en la fila de la tabla significa "repetir esta fila para cada item en mi contenedor de la lista de valores de objetos". El comando *repeat* pone los objetos de la lista en la variable *item* de a uno por vez (a esta variable se le llama la variable de repetición), y hace una copia de la fila usando esa variable. El valor de "item/getId" en cada fila es el Id del objeto para esa fila, lo mismo ocurre con "item/meta_type" e "item/title".

```
<tr tal:repeat="item container/objectValues">
```

Cualquier nombre puede ser usado para la variable de repetición ("item" es solo un ejemplo), siempre que comience con una letra y contenga solo letras, números y guiones bajos (_). La variable de repetición solo esta definida para el *tag* donde se hace la repetición. Si se intenta usar por encima o debajo del *tag tr* ocurrirá un error.

```
<td tal:content="item/getId">Id</td>
```

Estructuras de Repetición

La variable de repetición también se puede usar para obtener información sobre la repetición actual. Insertándola delante de la variable predefinida *repeat* en un *path*, se puede acceder a la cuenta de la repetición desde cero (index), desde uno (number), desde "A" (Letter), y de varias otras formas. Luego la expresión

```
<td tal:content="repeat/item/number">#</td>
```

vale 1 para la primera fila, 2 para la segunda y así sucesivamente.

Dado que un ciclo *tal:repeat* puede ser insertado dentro de otro, puede haber más de uno activo a la vez. Por eso se usa la expresión "repeat/item/number" en lugar de "repeat/number", se debe especificar a cuál ciclo se hace referencia incluyendo el nombre de la repetición.

Ejercicio

Crear un *page template* con el código de repetición visto.

Mostrar la página, notar como lista todos los objetos que estan en la misma carpeta que el *template*.

Agregar y borrar objetos de la carpeta y notar como la página refleja estos cambios.

Elementos Condicionales

Usando la declaración *tal:condition* se pueden chequear toda clase de condiciones. La declaración *tal:condition* deja el *tag* y su contenido si la expresión devuelve el valor *True*, y las remueve si el valor es *False*.

Zope considera al número cero, al *string* vacío, a la lista vacía, ya la variable incorporada *nothing* como valores falsos. Cualquier otro valor es verdadero, incluyendo números distintos de cero y *strings* no vacíos (también espacios).

Otro uso común de condiciones es probar una secuencia para ver si es vacía antes de iterar sobre ella.

Ahora veremos como agregar un control en la página de forma que si la lista de objetos es vacía no se muestre la tabla.

Elementos Condicionales

Agregemos esto al final de nuestro *Page Template*:

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

Ejercicio: Elementos Condicionales

Agregar tres carpetas llamadas “1”, “2” y “3” a la carpeta “/demo” en donde se encuentran nuestros *templates*. Abrir el *template* “demoUno” y ver la salida que genera usando el *tab* “Test”.

Veremos una tabla parecida a esto:

Number	Id	Meta-Type	Title
1	demoUno	Page Template	demoUno
2	1	Folder	1
3	2	Folder	2
4	3	Folder	3

Notar que si la expresión “container/objectValues” fuera falsa (si no hubiese objectValues por ejemplo), toda la tabla sería omitida.

Ejercicio: cambiar la expresión por “container/1/objectValues” y observar la salida.

Cambiando Atributos

El comando `tal:attribute` nos permite asignar dinámicamente valores a los atributos del *tag*.

La mayoría sino todos los objetos listados por el *template* tienen una propiedad *icon* que contiene el path al ícono de ese tipo de objeto.

Para mostrar este ícono en la columna *meta-type*, es necesario este *path* en el atributo *src* de un *tag img*.

Cambiando Atributos

Ejercicio: editar en la tabla la celda de la columna *meta-type* del ejemplo anterior con las siguientes líneas:

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

La declaración `tal:attributes` reemplaza el atributo `src` del tag `img` con el valor de “`item/icon`”.

Insertando Texto Estructurado

Normalmente las declaraciones `tal:replace` y `tal:content` convierten *tags* HTML en el texto donde son insertadas en una forma “escapada” que se muestra en la salida como como texto plano en lugar de HTML. Por ejemplo el caracter '`<`' se reemplaza por '`<`'. Si se desea insertar texto como parte de la estructura HTML evitando esta conversión se debe preceder la expresión con la palabra clave *structure*.

Esto es útil para cuando se quiere insertar un fragmento HTML o XML que esta almacenado en una propiedad o generado por otro objeto Zope. Por ejemplo, podemos tener un item de noticias que contiene código HTML simple (como mostrar partes en negrita o itálica) y se lo quiere preservar cuando se lo inserte en la página de noticias. En ese caso se podría usar algo como:

```
<p tal:repeat="newsItem here/topNews"  
  tal:content="structure newsItem">  
  Item de noticias con codigo <code>HTML</code>.  
</p>
```

Insertando Texto Estructurado

Realemente no importa si el texto reemplazante tiene código HTML o no, la palabra *structure* solo dice “dejar este texto sin tocar”.

Este compartamiento no es el por defecto ya que la mayoría del texto insertado no contiene HTML, mientras que puede tener caracteres que interfieran con la estructura de la página.

Ejercicio: Probar en un *template* los siguientes fragmentos de código

```
<span tal:content="string:<b>hola</b>">texto</span>
```

```
<span tal:content="structure string:<b>hola</b>">texto</span>
```

Elementos de Maqueta

Se pueden incluir elementos en la página que sean visibles en el *template* pero no en el texto generado usando la variable incorporada *nothing* :

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

Esto es útil para completar partes de la página que serán pobladas con contenido dinámico. Por ejemplo, una tabla de diez filas en el *template* tiene solo una (combinada en un *tal:repeat*). Agregando nueve filas *dummy* la estructura del *template* va a semejarse mucho al resultado final.

Contenido *Default*

Se pueden dejar los contenidos de un elemento (*tag*) intactos usando la expresión *default* junto con un `tal:content` o un `tal:repeat`, por ejemplo:

```
<p tal:content="default">Spam</p>
```

Esto se renderiza como:

```
<p>Spam</p>
```

Esto se usa para incluir contenido por defecto en forma selectiva, como en:

```
<p tal:content="python:here.getFood() or default">Spam</p>
```

Si el método “getFood” evalúa verdadero entonces se mostrará su resultado, si es falso se insertará la opción por defecto “Spam”.

Repetición Avanzada

Las variables de repetición vienen incorporadas al mecanismo de *template* y proveen información sobre la iteración actual. Son:

- `index` – índice de la repetición, empezando de cero.
- `number` – índice (o número) de la repetición, empezando desde uno.
- `even` – verdadera para iteraciones con índice par (0, 2, 4, ...).
- `odd` - verdadera para iteraciones con índice impar (1, 3, 5, ...).
- `start` – verdadera para la repetición inicial (índice 0).
- `end` – verdadera para la última repetición.
- `length` – longitud de la secuencia, que es el número total de repeticiones.
- `letter/Letter` – cuenta las repeticiones con letras en minúscula/mayúscula respectivamente ("a" - "z", "aa" - "az", "ba" - "bz", ...)

Repetición Avanzada

Los contenidos de una variable se repetición se pueden acceder usando expresiones *path* o expresiones Python.

En una expresión *path* que conste de tres partes que sean el nombre *repeat*, el nombre de la variable de repetición, y el nombre de la información que se desea, por ejemplo “repeat/item/start”.

En una expresión Python se usa la notación habitual de diccionarios para obtener la variable *repeat*, luego el acceso a atributos para obtener la información, por ejemplo, “python:repeat['item'].start”. La razón por la que no se usa “repeat/start” es que los tal:repeat pueden estar anidados, así que necesitamos una forma de especificar a cual de ellos le estamos pidiendo la información.

Ejercicio: probar distintas variables en el *template* de pruebas de repetición, por ejemplo, colorear distinto a las iteraciones pares.

Trucos con Repeticiones

A veces se necesita repetir una parte de una página, pero no hay candidato natural para el tag. En este caso se puede elegir un elemento para contener la declaración, pero se quiere evitar que aparezca en la página renderizada. Esto se puede hacer usando un `tal:omit-tag` :

```
<div tal:repeat="section here/getSections"
  tal:omit-tag="">
  <h4 tal:content="section/title">Título</h4>
  <p tal:content="section/text">contenido</p>
</div>
```

No se trata solo de ahorro de caracteres en la salida. Incluir los *tags div* puede modificar el aspecto de la página, especialmente si usa plantillas de estilo. La declaración `tal:omit-tag` desincluira los *tags de div* (el que abre y el que cierra) conservando su contenido.

Trucos con Repeticiones

Un ejemplo de tal:repeat anidado. Cada declaración tiene una variable de repetición con distinto nombre:

```
<table border="1">  
  <tr tal:repeat="x python:range(1, 13)">  
    <td tal:repeat="y python:range(1, 13)"  
      tal:content="python:'%d x %d = %d' % (x, y, x*y)">  
      X x Y = Z  
    </td>  
  </tr>  
</table>
```

Ejercicio: ver que linda queda en la pantalla.

Trucos con Repeticiones

Una forma de ordenar las listas es usando la función *sequence.sort*. Acá un ejemplo de como ordenar una lista de objetos por títulos y luego por fecha de modificación:

```
<table tal:define="objects here/objectValues;
      sort_on python:(('title', 'nocase', 'asc'),
                      ('bobobase_modification_time', 'cmp', 'desc'));
      sorted_objects python:sequence.sort(objects, sort_on)">
  <tr tal:repeat="item sorted_objects">
    <td tal:content="item/title">título</td>
    <td tal:content="item/bobobase_modification_time">modificado</td>
  </tr>
</table>
```

En este ejemplo se definió el criterio de ordenación de los argumentos separado para mas prolijidad. La función *sequence.sort* toma una secuencia y una descripción de como ordenarla, en este caso la descripción esta definida en la variable “sort_on”.

Control de Atributos Avanzado

La declaración `tal:attributes` se puede usar para reemplazar dinámicamente atributos. Se puede reemplazar más de un atributo a la vez separándolos con una “;”.

```
<a href="link" tal:attributes="href here/getLink;  
                             class here/getClass">link</a>
```

Ejercicio: Insertar el siguiente código en un *template* y probar el resultado. “here” hace referencia al lugar desde donde es llamado el *template*.

```
<a tal:attributes="href here/absolute_url">enlace</a>
```

Adquisición y navegación de URLs

Zope separa un URL y lo compara con la jerarquía de objetos, recorriéndola hasta que encuentra una coincidencia para cada parte. Este proceso se conoce como caminata de URL. Por ejemplo, cuando Zope recibe el URL “Zoo/Animales/leon/alimentar”, comienza desde el directorio raíz buscando un objeto llamado Zoo. Luego se mueve a la carpeta Zoo y busca un objeto llamado Animales. Se mueve a la carpeta Animales y busca un objeto llamado leon. Se mueve dentro de leon y busca el objeto llamado alimentar. Supongamos que el *script* alimentar no se encuentra en el objeto leon y es encontrado en la carpeta Zoo usando el mecanismo de adquisición. Zope siempre comienza a buscar un objeto comenzando por el último objeto recorrido, en este caso leon. Como leon no contiene nada, Zope vuelve al contenedor de leon, Animales. El *script* alimentar no está allí, así que Zope vuelve un paso más en la jerarquía hasta el contenedor Zoo, donde encuentra el *script*.

Adquisición y navegación de URLs

Ahora Zope ha alcanzado el final del URL y ha encontrado coincidencias de objetos para los elementos del URL. Zope reconoce que el último objeto hallado, alimentar, se puede llamar (es una función o método), y lo llama en el contexto del anteúltimo objeto encontrado, leon. Así es como el script alimentar es invocado para el objeto leon.

Así mismo se podría llamar al método bañar en el objeto leon visitando el URL "Zoo/Animales/leon/bañar". Este método podría adquirirlo de la carpeta Animales.

Adquisición y Navegación de URLs

La adquisición es un asunto de contenedores

El concepto detras de la adquisición es:

- Los objetos estan situados dentro de otros objetos. Estos objetos actúan como sus “contenedores”.
- Los objetos pueden adquirir el comportamiento de sus contenedores.

Ejercicio:

- Correr los *templates de prueba* desde los URLs de “/demo” y de “/demo/1” y observar las diferencias.
- En el template de repetición cambiar “container/objectValues” por “here/objectValues” y correrlo en las carpetas indicadas arriba.

Definiendo Variables

Podemos definir nuestras propias variables usando el atributo `tal:define`. Un uso común de esto es evitar escribir repetidamente algunas expresiones en la página. Otra es hacer una llamada a un método computacionalmente caro una sola vez dentro de la página (y almacenar su valor para usos posteriores). Las variables definidas dentro de un *tag* pueden usarse todas las veces que sea necesario siempre que estén en elementos abarcados por el *tag* donde se definió. En este ejemplo se define una variable para luego testearla e iterar sobre ella:

```
<ul tal:define="items container/objectIds"
  tal:condition="items">
  <li tal:repeat="item items">
    <p tal:content="item">id</p>
  </li>
</ul>
```

Ejercicio: evaluar las diferencias contra el ejemplo de `tal:repeat`.

Definiendo Variables

Como vimos la variable definida sólo está disponible en el *tag* o elementos abarcados por este. Si ponemos la palabra clave *global* delante del nombre de la variable podemos hacer que la variable dure desde donde se la define hasta el final del *template*:

```
<span tal:define="global items container/objectIds"></span>  
<h4 tal:condition="not:items">No hay items</h4>
```

Se pueden definir tantas variables como se quiera en un `tal:define` separándolas con un “;”. Cada variable puede tener su propio rango local o global. También se puede referenciar una variable definida previamente en la misma definición. For example:

```
<p tal:define="title template/title;  
            global containerTitle here/title;  
            tlen python:len(title);">
```

Manejo de Errores

Si ocurre un error en nuestro *page template*, podemos atrapar ese error y mostrar un mensaje útil al usuario. Supongamos que definimos una variable usando información de un *form*:

```
<span tal:define="global prefs request/form/prefs" tal:omit-tag="" />
```

Si Zope encuentra un problema, como que la variable buscada no existe en el *form*, la página se rompería y se mostraría una página de error en su lugar. Se puede evitar ese comportamiento con la funcionalidad `tal:on-error`, que provee manejo de errores limitado:

```
<span tal:define="global prefs here/scriptToGetPreferences"  
  tal:omit-tag=""  
  tal:on-error="string:Ha ocurrido un error con las preferencias">
```

Manejo de Errores

Cuando ocurre un error Zope busca una declaración tal:on-error en el elemento actual, si no lo encuentra sigue con el inmediatamente superior y así. Cuando encuentra un manejador de errores reemplaza el contenido de ese elemento con la expresión del manejador (en el ejemplo anterior, un mensaje de error).

Típicamente se define un manejador en un elemento que encierra una expresión lógica, por ejemplo una tabla.

Para un manejo de errores mas flexible se puede llamar a un *script*. Por ejemplo:

```
<div tal:on-error="structure here/capturarError">
```

```
...
```

```
</div>
```

Notar el uso de la palabra clave *structure*.

Manejo de Errores

El script puede examinar el error y tomar distintas acciones dependiendo de su valor. El script tiene acceso al error a través de la variable error definida predefinida. Por ejemplo (el script debe recibir los valores “errType”, “errValue”):

```
if errType==ZeroDivisionError:  
    return "<p>Can't divide by zero.</p>"  
else:  
    return ""<p>An error occurred.</p>  
        <p>Error type: %s</p>  
        <p>Error value: %s</p>"" % (errType, errValue)
```

El *script* puede tomar varias acciones, por ejemplo, puede logear el error o mandarlo por *email*. La declaración tal: on-error no es de propósito general, si se desea hacer validación de *forms* se recomienda otras herramientas, como los “Controller Page Templates”.

Interacciones Entre Comandos TAL

Cuando hay una declaración TAL por elemento, el orden en el que se ejecutan es simple. Comenzando desde la raíz, la declaración de cada elemento es ejecutada y luego la de sus elementos hijos siguiendo el orden en que aparecen.

Sin embargo, es posible tener mas una declaración TAL en un mismo elemento. Cualquier combinación de declaraciones puede aparecer en un mismo elemento, con la excepción de tal:content y tal:replace, que nunca pueden estar juntos.

Interacciones Entre Comandos TAL

Cuando un elemento tiene múltiples declaraciones, se ejecutan en el siguiente orden:

1. define
2. condition
3. repeat
4. content o replace
5. attributes
6. omit-tag

Dado que la declaración `tal:on-error` sólo se invoca si ocurre un error, no aparece en la lista.

Interacciones Entre Comandos TAL

El razonamiento en este orden es algo así: a menudo se desea asignar variables para su uso en otras declaraciones, así que *define* va primero. Lo próximo por hacer es decidir si el elemento se incluirá o no, así que sigue el condicional; dado que la condición puede depender de las variables seteadas. Puede ser útil poder reemplazar las distintas partes de un elemento con valores en cada iteración de un *repeat*, así que éste viene antes de *content*, *replace* y *attributes*. *Content* y *replace* no pueden ser usados juntos así que ocupan el mismo lugar en la jerarquía. *Omit-tag* viene último dado que ninguna otra declaración depende de ella y que debe aparecer luego de *define* y de *repeat*.

Interacciones Entre Comandos TAL

Hay tres restricciones de las que hay que estar conciente al combinar declaraciones TAL en elementos:

- Solo un tipo de cada declaración puede ser usada en cada *tag*. Dado que HTML no permite atributos múltiples con el mismo nombre.
- Tanto `tal:content` como `tal:replace` no pueden ser usadas en el mismo *tag*, dado que sus funcionalidades conflictúan.
- El orden en el que se presentan los atributos TAL en un *tag* no afecta su orden de ejecución.

Expresiones

Las expresiones proveen valores a las declaraciones TAL, y solo pueden funcionar dentro de ellas.

Las variables son nombres que pueden ser usados en las expresiones. Los *Page Templates* proveen variables predefinidas. Estas son:

- **nothing**: es un valor falso, similar al *string* vacío. Se puede usar en un `tal:replace` o `tal:content` para borrar un elemento o su contenido. Si un atributo es definido como *nothing*, es removido del tag, mientras que un *string* vacío inserta el *tag* con valor vacío.
- **default**: es un valor especial que no cambia nada cuando es usado con `tal:replace`, `tal:content` o `tal:attributes`. Dejando el texto en su lugar.

Expresiones

- **here**: el objeto desde donde el *template* esta siendo llamado. A menudo esto es lo mismo que *container*, pero puede variar si se esta usando adquisición. Se usa para obtener objetos Zope que se espera encontrar en distintos lugares dependiendo de donde se llame el *template*. Es igual a la variable *context* que se usa en scripts Python.
- **container**: el contenedor (usualmente una carpeta) en donde se encuentra el *template*. Se usa para obtener objetos Zope desde ubicaciones relativas a la posición del *template*. Las variables *container* y *here* se refieren al mismo objeto cuando un *template* es llamado desde su ubicación normal. Sin embargo cuando el *template* es aplicado a otro objeto (por ejemplo en un método ZSQL) ambas variables se referirán a distintos objetos.
- **modules**: la colección de módulos Python disponibles en los *templates*.

Path Alternativos

La existencia del camino “template/file” está garantizada cada vez que se usa el *template*, aunque su valor sea el *string* vacío. Algunos caminos, como “request/form”x”, pueden no existir durante la renderización del *template*. Normalmente esto causa un error cuando Zope evalúa la expresión *path*.

Ante la situación de que un camino no exista conviene tener un camino alternativo o valor para usar en su lugar. Por ejemplo, si “request/form/x” no existe, se podría usar “here/x” en su lugar. Esto se puede hacer listando los caminos por orden de preferencia y separándolos con una barra vertical “|”:

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

Dos variables que son útiles como alternativas para *path* son *nothing* y *default*.

Path Alternativos

Como parte final de de una expresión *path* alternativa tambien se puede usar una expresión que no sea de tipo *path*:

```
<p tal:content="request/form/edad|python:18">edad</p>
```

En este ejemplo, si el camino “request/form/age” no existe, el valor es el número 18. Esta *form* permite especificar valores por defecto que no pueden ser expresados como caminos. Notar que solo se puede usar una expresión de otro tipo en la última alternativa.

También se puede probar directamente la existencia de un camino prefijando con una expresión de tipo *exists*.

Expresiones String

Las expresiones *string* permiten combinar fácilmente expresiones *path* y texto. Todo el texto que aparece luego del prefijo “string:” es chequeado en busca de expresiones *path*. Cada expresión *path* debe estar precedida por un signo de dolar '\$'. Algunos ejemplos:

```
"string:Solo texto. No hay un path aqui.."
```

```
"string:copyright $year por Gomez Bolaños."
```

Si la expresión *path* tiene mas de una parte (si contiene una barra '/'), o necesita ser separada del texto que la sigue, debe ser encerrada por corchetes '{}'. Ejemplo:

```
"string:Tres ${manzana}s por favor."
```

```
"string:Su nombre es ${user/getUserName}."
```

Expresiones String

Dado que el texto esta dentro del valor de un atributo, solo se puede incluir una comilla doble usando la entidad sintáctica '"'. Como el símbolo dolar se usa para para marcar una expresión de tipo *path*, un símbolo de dolar literal debe ser escrito como dos símbolos de dolar '\$\$'. Unos ejemplos:

```
"string:Favor de pagar $$$pesos_debidos"
```

```
"string:Ella dijo, &quot;Hola gente.&quot;"
```

Las operaciones de formateo de *string* complejas, como buscar y reemplazar o cambiar a mayúsculas, no puede ser hecha fácilmente con expresiones *string*. Para esos casos se deben usar expresiones Python o *scripts*.

Ejercicio: Probar distintas combinaciones de expresiones *string* en *templates* de prueba.

Expresiones Not

Una expresión not permite negar el valor de otra expresión:

```
<p tal:condition="not:here/objectIds">  
  No hay objetos contenidos.  
</p>
```

Las expresiones *not* devuelven el valor verdadero cuando la expresión a la que están aplicados devuelve falso y viceversa. En Zope, cero, el *string* vacío, las secuencias vacías, *nothing* y *None* se consideran falsas, mientras que todo lo demás se considera verdadero. Las expresiones *path* no existentes no tienen valor de verdad y la aplicación de un 'not:' sobre ellas fallará.

Expresiones Nocall

Una expresión *path* común intenta renderizar el objeto que invoca. Esto significa que si el objeto es una función, *script*, método, o alguna otra cosa ejecutable, la expresión va a evaluar al resultado del objeto llamado. Esto es lo que se desea usualmente, pero no siempre. Supongamos que se quiere definir una variable para contener una función o clase de un módulo (de Python), para ser usada en una expresión Python. Podemos usar la expresión *nocall* para esto:

```
<p tal:define="join nocall:modules/string/join">
```

Esta expresión define la variable *join* como una función (*string.join*), en vez de ser el resultado de llamar a la función.

Las expresiones pueden ser usadas tanto en objetos como en funciones.

Expresiones Exists

Una expresión *exists* es verdadera cuando su *path* asociado existe. La siguiente es una forma de mostrar un mensaje de error solo si éste es pasado en el *request*:

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

Se puede lograr lo mismo mas fácilmente con una expresión *exists*:

```
<h4 tal:condition="exists:request/form/errmsg"  
  tal:content="request/form/errmsg">¡Error!</h4>
```

Expresiones Exists

Se pueden combinar expresiones exists con expresiones not, por ejemplo:

```
<p tal:condition="not:exists:request/form/number">  
Por favor ingrese un número entre 0 y 5</p>
```

Ejercicio: probar la expresión *exists* con combinaciones con la expresión *not* en un *template* de prueba.

Expresiones Python

Una expresión Python puede contener cualquier cosa que el lenguaje Python considere una expresión. No se pueden usar declaraciones como *if* o *while*. Además, Zope impone algunas restricciones de seguridad para evitar el acceso a información protegida o crear problemas como *loops* infinitos.

Expresiones Python: Comparaciones

Un lugar donde las expresiones Python son necesarias es en las declaraciones `tal:condition`. Usualmente se quiere comparar entre dos strings o dos números, y no hay soporte en las expresiones TAL para hacer esto sin expresiones Python. Se pueden usar los operadores de comparación `<` (menor que), `>` (mayor que), `==` (igual que), `!=` (distinto que). También se pueden usar los operadores booleanos *and*, *not* y *or*. Ejemplo:

```
<p tal:repeat="widget widgets">
  <span tal:condition="python:widget.type == 'gear'">
    Gear #<span tal:replace="repeat/widget/number">1</span>:
    <span tal:replace="widget/name">Name</span>
  </span>
</p>
```

Este ejemplo itera sobre una colección de objetos, imprimiendo información acerca de los *widgets* que son de tipo *gear*.

Expresiones Python: Comparaciones

A veces se quiere elegir entre diferentes valores dentro de una única declaración basándose en una o mas condiciones. Esto se puede hacer con la función *test* de la siguiente manera:

```
Usted <span tal:define="name user/getUserName"
      tal:replace="python:test(name=='Anonymous User',
                              'necesita logearse', default)">
      está logeado como
      <span tal:replace="name">Nombre</span>
</span>
```

Si el usuario es *Anonymous*, luego el elemento *span* es reemplazado con el texto “necesita logearse”. De otra forma el contenido por defecto es usado.

Expresiones Python: Comparaciones

La función `test` funciona como un “if/then/else”. Este es otro ejemplo de como usar la función `test`:

```
<tr tal:define="oddrow repeat/item/odd"  
    tal:attributes="class python:test(oddrow, 'oddclass',  
                                'evenclass')">
```

Esto asigna al atributo `class` los valores `'oddclass'` y `'evenclass'` alternadamente a las filas de la tabla, permitiendo dar distintos estilos a la salida HTML.

Sin la función `test` necesitaríamos escribir dos elementos `tr` con diferentes condiciones, una para columnas pares y otra para impares.

Expresiones Python: Otros Tipos De Expresiones

Se pueden usar otros tipos de expresiones dentro de una expresión Python. Cada tipo expresión tiene una función correspondiente con el mismo nombre, incluyendo: *path()*, *string()*, *exists()*, and *nocall()*. Esto permite escribir expresiones como:

```
"python:path('here/%s/thing' % foldername)"  
"python:path(string('here/$foldername/thing'))"  
"python:path('request/form/x') or default"
```

El último ejemplo tiene un significado levemente distinto a la expresión *path* "request/form/x | default", dado que usa el texto *default* si "request/form/x" no existe o si es falso.

Expresiones Python: Objetos Zope

Buena parte del potencial de Zope involucra lograr que interactúen objetos especializados. Nuestros *Page Templates* pueden usar *scripts*, métodos SQL, catálogos y objetos de contenido customizados. Para poder usar todos estos objetos tenemos que saber como accederlos dentro de los *Page Templates*.

Las propiedades de los objetos son atributos, luego podemos acceder el título de un template con la expresión "template.title". La mayoría de los objetos Zope soportan adquisición, lo que nos permite obtener atributos desde objetos "padre" (anteriores en la jerarquía). Esto significa que la expresión "here.Control_Panel" traerá el objeto "Control Panel" desde la carpeta raíz. Los métodos de los objetos son atributos, como "here.objectIds" y "request.set".

Expresiones Python: Objetos Zope

Los objetos contenidos en una carpeta pueden ser accedidos como atributos de la carpeta, pero como a menudo tienen Ids que no son identificadores Python válidos, no se puede usar la notación habitual.

Por ejemplo, no se puede acceder el objeto “penguin.gif” con la siguiente expresión:

```
"python:here.penguin.gif"
```

en su lugar se debe usar:

```
"python:getattr(here, 'penguin.gif')"
```

dado que Python no reconoce nombres de atributos con puntos '.' en ellos.

Expresiones Python: Objetos Zope

Algunos objetos, como request, módulos, y carpetas Zope proveen soporte para acceso de items a la Python, por ejemplo:

```
request['URL']  
modules['math']  
here['thing']
```

Cuando se usa el acceso de items en una carpeta, no intenta usar adquisición sobre el nombre, luego solo tendrá éxito en la búsqueda si hay un objeto con ese identificador en la carpeta.

Usando Scripts

Los objetos *script* se usan para encapsular lógica de negocios y manejo de datos complejos. Cada vez que nos encontremos escribiendo muchas declaraciones TAL con expresiones complicadas en ellas, debemos considerar usar un *script*. De esta forma se pueden mantener páginas mas simples y prolijas.

Cada *script* tiene una lista de parámetros que espera que se le de cuando es llamado. Si esta lista es vacía, entonces se puede llamar al *script* usando una expresión *path*. De lo contrario, se deberá usar una expresión Python para poder pasarle los parámetros:

```
"python:here.myscript(1, 2)"
```

```
"python:here.myscript('arg', foo=request.form['x'])"
```

Usando Scripts

Si se desea devolver mas de un dato desde un *script* a un *Page Template*, es buena idea hacerlo con un diccionario. De esa forma, se puede definir un avariable para contener todos los datos, y usar expresiones *path* para referirse a cada uno de ellos. Por ejemplo, supongamos que el *script* `getPersona` devuelve un diccionario con las claves `nombre` y `edad`:

```
<span tal:define="persona here/getPersona"
      tal:replace="string:${persona/nombre} tiene ${person/edad}">
</span> años..
```

Por supuesto, tambien se puede devolver un objeto Zope o una lista Python.

Ejercicio: Hacer dos *script* simples (que devuelvan un string por ejemplo), uno que reciba parámetros y otro que no.

Módulos Python

El lenguaje de programación Python tiene una gran cantidad de módulos, que proveen mucha variedad de funcionalidades para los programas Python. Cada módulo es una colección de funciones Python, datos, y clases relacionadas con un mismo propósito, como cálculos matemáticos o expresiones regulares.

Varios módulos, incluyendo “math” y “string”, están disponibles para expresiones Python por defecto. Por ejemplo, se puede obtener el valor de pi desde el módulo “math” escribiendo “python:math.pi”. Para accederlo desde una expresión path se usa la variable *modules*, “modules/math/pi”.

Módulos Python

El módulo “string” está escondido en las expresiones Python por la función del tipo de expresión “string”, así que para acceder al módulo se debe usar la variable *modules*. Esto se puede hacer directamente en la expresión en la que se use, o definiendo una variable global para ello, como en:

```
tal:define="global mstring modules/string"  
tal:replace="python:mstring.join(slist, ':')"
```