

Macros

Hasta ahora hemos visto como los *page templates* pueden ser usados para agregar contenido dinámico en páginas individuales. Otra característica de los *page templates* es la habilidad para reusar elementos a traves de varias páginas.

Por ejemplo, con *page templates* se puede conseguir un “look and feel” estándar para un sitio. Sin importar el contenido de la página, se puede conseguir que tenga un encabezado, barra lateral, pié de página, y otros elementos estándares. Este es un requerimiento muy común para sitios *web*.

Se pueden reusar elementos de presentación en distintas páginas usando macros. Los macros definen una sección de la página que puede ser reusada en otras páginas. Un macro puede ser una página completa, o solo un tramo de ella, como el encabezado. Luego de definidos uno o mas macros en un *page template*, se los puede usar desde otro *page template*.

Usando Macros

Se pueden definir macros con atributos en *tags* de forma similar a las declaraciones TAL. Estos atributos son llamados declaraciones METAL, por “Macro Expansion Tag Attribute Language”. Este es un ejemplo de una definición de un macro:

```
<p metal:define-macro="copyright">  
  Copyright 2001, <em>Foo, Bar, y Asociados</em> S.A..  
</p>
```

Esta declaración `metal:define-macro` define un macro llamado "copyright". El macro consiste de un elemento `p` (incluidos los elementos contenidos).

Usando Macros

Los Macros definidos en un *page template* son almacenados en los atributos macro del *template*. Se puede usar esos macros desde otros *page templates* refiriéndose a ellos a través de los atributos macro que están en la *page template* donde fueron definidos. Por ejemplo, supongamos que el macro “copyright” está en un *page template* llamado “master_page”. Así es como se usaría ese macro desde otro *page template*:

```
<b metal:use-macro="container/master_page/macros/copyright">  
  Aca va el macro  
</b>
```

En el siguiente *page template*, el elemento b será reemplazado completamente por el macro cuando Zope renderice la página:

```
<p>  
  Copyright 2001, <em>Foo, Bar, y Asociados</em> S.A.  
</p>
```

Usando Macros

Si se cambia el macro (por ejemplo, si el contenedor del copyright cambia) entonces todos los *page templates* que usan el macro reflejarán automáticamente el cambio.

Notar: como el macro está identificado por una expresión *path* usando la declaración “metal:use-macro”. La declaración “metal:use-macro” reemplaza el macro de la invocador por el macro invocado.

Ejercicio: crear un *template* declarando un macro y otro *template* que use ese macro.

Detalles de los Macros

Las declaraciones `metal:define-macro` y `metal:use-macro` son bastante simples. Sin embargo, hay algunas sutilezas que cabe mencionar.

El nombre de un macro debe ser único dentro del *page template* en el que es definido. Se pueden definir varios macros en un mismo documento pero todos deben tener distintos nombres.

Normalmente nos referiremos a un macro en una declaración `metal:use-macro` con una expresión *path*. Sin embargo, se puede usar una expresión de cualquier tipo mientras que ésta devuelva un macro. Por ejemplo:

```
<p metal:use-macro="python:here.getMacro()">
```

Reemplazado por un macro determinado dinámicamente,
que está ubicado en el script `getMacro`.

```
</p>
```

Detalles de los Macros

Se puede usar la variable *default* con la declaración `metal:use-macro`:

```
<p metal:use-macro="default">  
  Este contenido permanece – ninguna macro es usada  
</p>
```

El resultado es el mismo que usar *default* con `tal:content` o `tal:replace`. El contenido por defecto del *tag* no cambiará cuando sea renderizado. Esto puede ser útil si se necesita usar un macro condicionalmente o respaldarse en el contenido por defecto si el macro no existe.

Detalles de los macros

Si se intenta usar la variable *nothing* con `metal:use-macro` se obtendrá un error, dado que *nothing* no es un macro. Si se desea usar *nothing* para incluir condicionalmente un macro, en su lugar se debe enmarcar la declaración `metal:use-macro` con una declaración `tal:condition`.

Zope gestiona los macros primero cuando renderiza los *templates*. Luego Zope evalúa las expresiones TAL. Por ejemplo, considérese este macro:

```
<p metal:define-macro="title"  
  tal:content="template/title">  
  Título del template  
</p>
```

Detalles de los Macros

Cuando se use este macro insertará el título del *template* en el que el macro está siendo usado, no el título del *template* en el que el macro está definido. En otras palabras, cuando se usa un macro es como copiar el texto del macro dentro del nuevo *template* y renderizarlo en ese *template*.

Usando Slots

Los macros son mucho mas útiles si se puede sobrescribir parte ellos cuando se los usa. Esto se puede hacer definiendo *slots* en el macro que seran llenados cuando se usa el *template*. Por ejemplo:

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Productos</a></li>
    <li><a href="/support">Soporte</a></li>
    <li><a href="/contact">Contactenos</a></li>
  </ul>
</div>
```

Usando Slots

Ahora supongamos que queremos incluir alguna información adicional en la barra lateral para algunas páginas. Una forma de lograr esto es con *slots*:

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Productos</a></li>
    <li><a href="/support">Soporte</a></li>
    <li><a href="/contact">Contactenos</a></li>
  </ul>
  <span metal:define-slot="info_adicional"></span>
</div>
```

Usando Slots

Cuando se usa este macro se puede elegir llenar este *slot* como en:

```
<p metal:use-macro="container/master.html/macros/sidebar">  
  <b metal:fill-slot="additional_info">  
    No se olvide de mirar en <a href="/specials">especiales</a>.  
  </b>  
</p>
```

Usando Slots

Cuando este *template* se renderice mostrará la información extra provista en el *slot*:

```
<ul>  
  <li><a href="/">Home</a></li>  
  <li><a href="/products">Productos</a></li>  
  <li><a href="/support">Soporte</a></li>  
  <li><a href="/contact">Contactenos</a></li>  
</ul>  
<b>  
  No se olvide de mirar en <a href="/specials">especiales</a>.  
</b>
```

Notar como el elemento *span* que define el *slot* es reemplazado por el elemento *b* que define el *slot*.

Personalizando la Presentación por Defecto

Un uso común de *slots* es proveer la presentación por defecto que se puede personalizar. En el ejemplo anterior la definición del *slot* era un elemento vacío. Sin embargo se puede poner contenido dentro de la definición. Por ejemplo, consideremos el macro de barra lateral revisado:

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
    Enlaces
    <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/products">Productos</a></li>
      <li><a href="/support">Soporte</a></li>
      <li><a href="/contact">Contactenos</a></li>
    </ul>
  </div>
  <span metal:define-slot="info_adicional"></span>
</div>
```

Personalizando la Presentación por Defecto

Ahora la barra lateral es completamente personalizable. Se puede llenar el *slot* de “links” para redefinir los enlaces. Si se elige no ocupar el *slot* se devolverá el contenido por defecto.

Macros de Página Completa

En lugar de usar macros para porciones de presentación compartidas entre páginas, se pueden usar macros para definir páginas completas. Esto es posible con *slots*. Un ejemplo:

```
<html metal:define-macro="mipagina">
  <head>
    <title tal:content="here/title">título</title>
  </head>
  <body>
    <h1 metal:define-slot="headline"
      tal:content="here/title">title</h1>
    <p metal:define-slot="body">
      body
    </p>
    <span metal:define-slot="footer">
      <p>Copyright 2001</p>
    </span>
  </body>
</html>
```

Macros de Página Completa

Luego se puede usar ese macro en templates para distintos tipos de contenido, o diferentes partes del sitio. Por ejemplo, este es un template para noticias que puede usar el macro:

```
<html metal:use-macro="container/master/macros/mipagina">  
  
  <h1 metal:fill-slot="headline">  
    Prensa:  
    <span tal:replace="here/getHeadline">Titular</span>  
  </h1>  
  
  <p metal:fill-slot="body"  
    tal:content="here/getBody">  
    Cuerpo de la noticia aca.  
  </p>  
  
</html>
```

Macros de Página Completa

Este template redefine el *slot* “headline” para incluir las palabras “Prensa” y llama al método `getHeadline` en el objeto actual. También redefine el *slot* “body” llamando al método `getBody`.

El poder de este enfoque es que ahora se puede cambiar el macro “page” y la página de noticias será automáticamente actualizada. Por ejemplo se podría poner el cuerpo de la página en una tabla y agregar una barra lateral a la izquierda y estos elementos pasarían a formar parte de todo el sitio donde se use este macro.

Ejercicio: crear un *page template* que use el macro “mipagina”, reemplazando contenido en los *slots*.

Ejercicio: Plone provee un template principal con el cual formar las páginas identificar cual es el archivo donde está, partiendo desde “portal_skins”.

Cacheando Templates

Aunque la renderización de *page templates* normalmente es rápida, a veces no es lo suficientemente rápida. Para páginas accedidas con frecuencia, o páginas que toman mucho tiempo renderizar, tal vez se prefiera cambiar algo de comportamiento dinámico por velocidad. “Caching” nos deja hacer esto.

Se pueden cachear *page templates* usando un administrador cache de la misma forma que se pueden cachear otros objetos, asociándolos con el administrador de cache. Esto se puede hacer yendo al “cache view” del page template y eligiendo allí un administrador de cache (debe haber uno en el camino de adquisición del template para que se vea en el view, la instalación de Plone agrega un RAM cache y un HTTP cache por defecto). Otra forma es yendo a la vista de asociación (“associate view”) del administrador de cache elegido y localizar allí el *template* deseado.

Cacheando Templates

Este es un ejemplo de como cachear un *page template*. Crearemos un *script* llamado “long” con el siguiente contenido:

```
for i in range(500):  
    for j in range(500):  
        for k in range(5):  
            pass  
return 'Done'
```

El propósito de este *script* es hacer notar el tiempo de ejecución. Crearemos un *page template* que use el script, por ejemplo:

```
<html>  
  <body>  
    <p tal:content="here/long">resultado</p>  
  </body>  
</html>
```

Cacheando Templates

Cargemos la página, nótese el tiempo que toma en renderizarse. Ahora mejoremos ese tiempo usando cache. Podemos crear un objeto “RAM Cache manager” o usar uno que ya exista, debe estar en la carpeta raíz, en la del *template* o alguna en el camino de adquisición. Comprobemos que se ve en la vista de cache de nuestro *page template* y seleccionémoslo.

Seleccionemos el objeto de administrador cache que hemos creado y hagamos click en “Save Changes”. Click en el link de configuración (settings), por defecto almacenará los objetos durante una hora (3600 segundos). Conviene elegir este número de acuerdo a la aplicación.

Regresemos a la página de veámosla una vez mas. Tomará un tiempo en cargarse pero se guardará en cache. Ahora recargemos la página y notemos el cambio en la velocidad. Se pueden intentar varias recargas comprobando que el objeto está en cache.

Cacheando Templates

Si el page template es modificado será removido del cache. Luego la próxima vez que se renderice volverá a tomar un tiempo, luego de lo cual volverá a estar en cache.

Cachear es una forma simple y poderosa para mejorar el rendimiento. No hace falta saber mucho de ella para lograr resultados. Igualmente puede convenir investigar mas sobre el tema, familiarizándose con los objetos “RAM Cache manager” y “Accellerated HTTP Cache manager”.

Obteniendo Grandes Cantidades de Información por Tandas

Cuando un usuario consulta una base de datos y obtiene un resultados con muchos de items, a menudo es mejor mostrarlo en varias páginas de, digamos, veinte items en lugar de mostrar todo en una sola. Al proceso de separar una lista por tandas de le llama “batching”. Page templates soporta la separación por tandas usando un objeto especial Batch, provisto por el módulo ZTUtils. Un ejemplo:

```
<ul tal:define="lots python:range(100);
    batch python:modules['ZTUtils'].Batch(lots, size=10, start=0)">
  <li tal:repeat="num batch"
    tal:content="num">0
  </li>
</ul>
```

Obteniendo Grandes Cantidades de Información por Tandas

Este ejemplo muestra una lista con diez elementos (desde el 0 al 9). El objeto Batch corta una lista en tandas. Este ejemplo parte una lista de cien objetos en diez tandas de diez.

Se puede mostrar una tanda distinta de diez items pasando como parámetro un número distinto en *start*:

```
<ul tal:define="lots python:range(100);  
    batch python:modules['ZTUtils'].Batch(lots,  
        size=10, start=13)">
```

Esta tanda comenzará desde el item catorce y seguirá hasta el item veintitres. Es decir, muestra desde el número 13 hasta el 22. Notemos entonces que el argumento de inicio de la tanda (*start*) es el index del primer item. El index comienza desde cero. Python usa índices para referirse a los elementos de una lista.

Obteniendo Grandes Cantidades de Información por Tandas

Normalmente cuando usamos tandas queremos incluir elementos de navegación que nos permitan movernos de una tanda hacia otra. Este es un ejemplo:

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
  </head>
  <body tal:define="empleados here/getEmpleados;
    start python:int(path('request/start | nothing') or 0);
    batch python:modules['ZTUtils'].Batch(empleados,
      size=3,
      start=start);
    previous python:batch.previous;
    next python:batch.next">
```

Obteniendo Grandes Cantidades de Información por Tandas

```
<p>
  <a tal:condition="previous"
    tal:attributes="href string:${request/URL0}?start:int=${previous/first}"
    href="previous_url">previo</a>
  <a tal:condition="next"
    tal:attributes="href string:${request/URL0}?start:int=${next/first}"
    href="next_url">proximo</a>
</p>

<ul tal:repeat="empleado batch" >
  <li>
    <span tal:replace="empleado/name">Juan</span>
    recibe $<span tal:replace="empleado/salary">100,000</span>
    al año.
  </li>
</ul>

</body>
</html>
```

Obteniendo Grandes Cantidades de Información por Tandas

Copiar este código en un *script* “getEmpleados”:

```
return [ { 'name': 'Chris McDonough', 'salary':'5'},  
         { 'name': 'Guido van Rossum', 'salary': '10'},  
         { 'name': 'Casey Duncan', 'salary':'20' },  
         { 'name': 'Andrew Sawyers', 'salary':'30' },  
         { 'name': 'Evan Simpson', 'salary':'35' },  
         { 'name': 'Stephanie Hand', 'salary':'40' }, ]
```

Obteniendo Grandes Cantidades de Información por Tandas

Si vemos la declaración `tal:define` en el elemento *body*, define un conjunto de variables para la tanda. La variable “empleados” es una lista de objetos devuelta por el *script*.

La segunda variable, “start”, es o bien configurada con el valor de “request/start” o a cero si no hay una variable *start* en el *request*. La variable *start* lleva la cuenta de la posición actual en la lista de empleados. La tanda comienza en la posición especificada por la variable “start”.

Las variables “previous” y “next” se refieren a la pasada y a la próxima tanda respectivamente si existen. Todas estas variables quedan disponibles dentro del elemento *body*.

Obteniendo Grandes Cantidades de Información por Tandas

Veamos los enlaces de navegación. Se crean enlaces para navegar hacia atrás y hacia adelante en la sucesión de tandas. La declaración `tal:condition` primero prueba si existen las respectivas tandas, si es así, se renderiza el enlace (no aparece de lo contrario).

La declaración `tal:attributes` crea un enlace a las respectivas tandas. El enlace es el URL de la página actual (`request/URL0`) junto con un string de consulta indicando el índice “start” de la tanda.