Zope Page Templates

Aitzol Naberan Burgaña
CodeSyntax

CodeSyntax

Qué es

- Es una herramienta que ofrece Zope para la generación de páginas web
- Ayuda a los diseñadores y a los desarrolladores
  - Amigable para las herramientas de edición
  - Lo que ves es “muy parecido” a lo que obtienes
  - El código está fuera de la plantilla (salvo elementos estructurales)

CodeSyntax

Cómo funcionan

- Template Attribute Language (TAL)
  - Define una serie de atributos

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

- Los editores no entienden el *namespace tal*
  - Eliminan esa parte
  - Los diseñadores pueden trabajar ya que se define un valor por defecto para el contenido.

Cómo funcionan

- Aqui se demuestra que “lo que ves es 'muy parecido” a lo que obtienes”
  - La parte *here/title* es dinamica
  - Cambiará en tiempo de ejecución
  - El contenido de la etiqueta title hace que tengamos una idea de como aparecerá el contenido

CodeSyntax

Cómo funcionan

- Tenemos estamentos para:
  - Reemplazar etiquetas
  - Reemplazar el contenido de una etiqueta
  - Elementos de repetición
  - Elementos condicionales
  - Definición de bloques
  - ...

Cómo funcionan

- NO es posible:
  - Generar clases o funciones
  - Ejecutar flujos de control complejos
  - Expresar algoritmos complejos
- Si es necesaria alguna de las caracteristicas anteriores es el momento de empezar a pensar en python.
- Esta es una limitación intencionada, para mantener el código (logica) fuera de las plantillas.

CodeSyntax

Hello world!

```
<html>
    <head>
        <title tal:content=”template/title”>Page Title</title>
    </head>
    <body>
        <h1 tal:content=”template/title”>
            Aqui vamos a insertar el encabezado
        </h1>
    </body>
</html>
```

CodeSyntax

Expresiones simples

- *Path explessions*
  - template/title
  - request/URL
  - user/getUserName
- Siempre empiezan con un nombre de variable.
- Si la primera variable tiene el valor que necesitamos terminamos, de lo contrario se sigue con '/'
  - P.e.: template/title: es la propiedad *title* de *template*

CodeSyntax

Expresiones simples

- *request* y *user* son variables predefinidas por Zope.
- Es posible definir nuestras propias variables.

CodeSyntax

Insertar texto

- *tal:replace*
  - Reemplaza toda la etiqueta
- *tal:content*
  - Reemplaza el *contenido* de la etiqueta

```
<html>
<body>
	<h1>Ejemplo de inserción</h1>
	Esto reemplaza la etiqueta 'strong' :
	<strong tal:replace=”template/title”>Título</strong>
	Esto reemplaza el contenido:
	<strong tal:content=”template/title”>Título</strong>
</body>
</html>
```

Insertar estructuras

- Tanto *tal:content* como *tal:replace* escapan las entidades html
- Para insertar html necesitamos utilizar la palabra clave *structure*

CodeSyntax

Insertar estructura

```
<html>
  <head>
    <title tal:content="template/title">The title</title>
  </head>
  <body>
    <h1 tal:content="template/title">Título de la página</h1>
    <p>Con <strong>structure</strong>:
       <span tal:replace="structure here/returnHTML">
          Contenido de la plantilla</span>
    </p>
    <p>Sin <strong>structure</strong>:
       <span tal:replace="here/returnHTML">
          Contenido de la plantilla</span>
    </p>
  </body>
</html>
```

CodeSyntax

Estructura de repetición

- *tal:repeat*
  - Es el equivalente al bloque *for* de python
  - *tal:repeat=”variable_de_repeticion elementos”*
  - Información disponible sobre el número de repetición
    - *index :* indice de la iteración (cuenta desde 0)
    - *number* : número de la iteración (cuenta desde 1)
    - *letter* : cuenta usando caracteres (desde 'a')
    - *Letter* : cuenta usando caracteres (desde 'A')

CodeSyntax

Estructura de repetición

- Información disponible sobre caracteristicas de la repetición
  - *even:* True si la repetición es par
  - odd*:* True si la repetición es impar
  - *start:* True si es la primera repetición
  - *end:* True si es la última repetición
  - *length:* Número total de repeticiones
- Es posible anidar repeticiones

CodeSyntax

Estructuras de repetición

```
<html>
   <head>
      <title>Ejemplo de repetición</title>
   </head>
   <body>
   <h1>Ejemplo de repetición</h1>
   <table>
   <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>
```

CodeSyntax

Estructuras condicionales

- *tal:condition*
  - Es el equivalente al bloque *if* de python
  - *tal:condition=”expresion”*
  - Ejecuta el bloque si la condición se cumple
  - No existe bloque *else*
    - Se necesita otro bloque *condition* con la negación de la condición original
      - *tal:condition=”not:expresion”*

CodeSyntax

Estructuras condicionales

```
<html>
   <head>
      <title>Ejemplo de repetición</title>
   </head>
   <body>
   <h1>Ejemplo de repetición</h1>
   <table tal:condition=”container/objectValues”>
   <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>
```

CodeSyntax

Definir variables

- *tal:define*
  - Define una variable que es valida dentro del ambito de la etiqueta en la que se define
  - *tal:define=”variable valor_de_la_variable”*

```
<table tal:define=”items container/objectValues”
       tal:condition=”items”>
   <tr>
    <th>Number</th><th>Id</th><th>Meta-Type</th><th>Title</th>
   </tr>
   <tr tal:repeat="item items">
    <td tal:content="repeat/item/number">#</td>
    <td tal:content="item/getId">Id</td>
    <td><img src=”path.gif” tal:attribures="src item/icon">
        <span tal:replace=”item/meta_type”>Meta-Type</span></td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
```

CodeSyntax

Definir variables

- Para definir una variable global es necesario utilizar la palabra clave *global* antes de la definición de la variable.

```
...
<table tal:define=”global items container/objectValues”>
...
```

- Es posible definir más de una variable en la misma sentencia.

```
...
<table tal:define=”global items container/objectValues;
                   container_icon container/icon”>
...
```

CodeSyntax

Variables predefinidas

- *nothing*: Es un valor nulo
- *default:* No modifica el texto de la etiqueta
- *options:* Es la variable que guarda los argumentos basados en palabras clave (solamente disponible cuando se llama a la plantilla desde código python)
- *attrs:* Diccionario de atributos de la etiqueta actual
- *root*: La raiz de Zope

Variables predefinidas

- *here*: El objeto desde el que la plantilla es llamado (contexto)
- *container*: El objeto en donde la plantilla esta guardado.
- *modules*: Lista de módulos python disponibles.

Modificar atributos

- *tal:attributes*
  - Es posible modificar los valores de los atributos de las etiquetas
  - *tal:attributes=”atributo nuevo_valor”*

```
...
<td><img src=”path_a_la_imagen.gif”
         tal:attributes=”src item/icon”>
    <span tal:replace=”item/meta_type”>Meta-type</span>
</td>
```

- Se ha cambiado *tal:content* por una etiqueta *span* y un *tal:replace*, para poder tener tanto el texto como la imagen

CodeSyntax

Modifica atributos

- Es posible modificar mas de un atributo en una misma sentencia

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

CodeSyntax

Cambiar atributos

```
<html>
   <head>
      <title>Ejemplo de repetición</title>
   </head>
   <body>
   <h1>Ejemplo de repetición</h1>
   <table tal:condition=”container/objectValues”>
   <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><img src=”path.gif” tal:attribures="src item/icon">
        <span tal:replace=”item/meta_type”>Meta-Type</span></td>
    <td tal:content="item/title">Title</td>
  </tr>
</table>
</body>
</html>
```

CodeSyntax

Manejo de errores

- Si ocurre un error, es posible capturarlo y mostrar un mensaje notificando de ello. En caso de que no se capture se mostrará el error genérico de Zope.
- *tal:on-error=”mensaje_en_caso_de_error”*

```
<a href="link"
   tal:define=”item here/estoFallaSeguro”
   tal:on-error=”string:Ha ocurrido un error”>
```

Relaciones de orden

- Es posible especificar mas de una expresión *tal* en una etiqueta.
- En estos casos las expresiones se evaluan en un orden concreto:
  - *define*
  - *condition*
  - *repeat*
  - *content / replace*
  - *attributes*
  - *omit-tag*

CodeSyntax

Relaciones de orden

```
<p tal:define="x /root/a/long/path/x | nothing"
   tal:condition="x"
   tal:content="x/txt"
   tal:attributes="class x/class">Ex Text</p>
```

- Solamente se puede utilizar un elemento de cada tipo en una etiqueta
- *tal:replace* y *tal:content* no se pueden utilizar en una misma etiqueta
- El orden en el que se incluyen no afecta al orden en el que se evaluan.

CodeSyntax

Relaciones de orden

- Este código no funciona

```
<ul>
  <li tal:repeat="n python:range(10)"
      tal:condition="python:n != 3"
      tal:content="n">
    1
  </li>
</ul>
```

- La condición siempre se evalua antes que la repetición, por tanto, a la hora de evaluar la condición la variable *n* no existe.

Relaciones de orden

- Este código funciona

```
<ul>
  <div tal:repeat="n python:range(10)"
       tal:omit-tag="">
    <li tal:condition="python:n != 3"
        tal:content="n">
      1
    </li>
  </div>
</ul>
```

- Al estar englobados en distintas etiquetas, primero se evalua la expresión *repeat* y después la expresión *condition*.

CodeSyntax

Expresiones string

- Con las expresiones string podemos mezclar cadenas de caracteres con variables definidas en la plantilla.
- Para insertar un string utilizaremos *string:*
- Para inserta una variable de plantilla en una expresion string utilizaremos *$*
- Si la expresión que queremos utilizar tiene mas de un elemento englobaremos la llamada entre { *y* }

CodeSyntax

Expresiones string

```
"string:Just text. There's no path here."
"string:copyright $year by Fred Flintstone."
"string:Your name is ${user/getUserName}!"
```

CodeSyntax

Mas expresiones

- Expresiones de path alternativas

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

- Expresiones *nocall*

```
<h1 tal:define=”metodo nocall:here/Metodo”>
```

CodeSyntax

Expresiones python

- Usando las expresiones *python* podemos ejecutar cualquier sentencia que se pueda ejecutar en python
- No es muy conveniente abusar de estas expresiones, ya que es una tentación muy grande que nos llevaría a mezclar la lógica de la aplicación en la plantilla.
- Es necesario utilizarlas cuando una función que queremos utilizar necesita parámetros.

CodeSyntax

Expresiones python

- Para utilizar expresiones python tenemos que usar *python:*
- Al utilizar las expresiones python tenemos que usar el operador '.' en vez del operador '/'

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

```
"python:here.myscript(1, 2)"
"python:here.myscript('arg', foo=request.form['x'])"
```

Macros

- Mediante los macros podemos definir bloques de sentencias que podremos reutilizar en diferentes plantillas.
- Estos bloques son necesarios ya que en una aplicación web debemos de mantener una coherencía entre todas las páginas que la componen.
  - Encabezado
  - Píe de pagina
  - Bloques de patrocinadores

CodeSyntax

Definir macros

- Para definir macros utilizaremos estamentos METAL (Macro Extensión Tal Attribute Language)

```
<p metal:define-macro=”licencia”>
   Creative Commons SA
</p>
```

- El código anterior define una macro llamada *licencia* que está compuesta por una etiqueta *<p>* (y todo su contenido)

CodeSyntax

Usar macros

- Las macros definidas en una plantilla se guardan en la variable *macros* de la plantilla.

```
<!-- suponiendo que la macro esta definida en master_template -->
<b metal:use-macro=”container/master_temple/macros/licencia”>
   El contenido de la macro va aqui
</b>
```

CodeSyntax

Slots

- Las macros ofrecen la opción de sobreescribir partes de su código cuando son utilizadas.
- Para ello se definen *slots* dentro de las macros.

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/support">Support</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
</div>
```

CodeSyntax

Definir slots

- Para definir un *slot* tenemos que utilizar *metal:define-slot* dentro de una definición de macro.
- Podemos utilizar todas la exprsiones que hemos visto con anterioridad.

```
<div metal:define-macro="sidebar">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/support">Support</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
  <span metal:define-slot="additional_info"></span>
</div>
```

CodeSyntax

Macros y slots para definir la estructura de las páginas

```
<html metal:define-macro="page">
  <head>
    <title tal:content="here/title">The title</title>
  </head>

  <body>
    <h1 metal:define-slot="headline"
        tal:content="here/title">title</h1>

    <p metal:define-slot="body">
      This is the body.
    </p>

    <span metal:define-slot="footer">
      <p>Copyright 2001 Fluffy Enterprises</p>
    </span>

  </body>
</html>
```

CodeSyntax

Usar slots

- Cuando utilizamos una macro que define un slot tenemos la opción de rellenarlo.

```
<!-- la macro está definida en master_html -->
<p metal:use-macro="container/master_html/macros/sidebar">
  <b metal:fill-slot="additional_info">
    Make sure to check out our <a href="/specials">specials</a>.
  </b>
</p>
```

CodeSyntax

Usar slots

- Es posible definir *slots* con un contenido predeterminado que se puede sobreescribir si es necesario.

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/support">Support</a></li>
    <li><a href="/contact">Contact Us</a></li>
  </ul>
  </div>
  <span metal:define-slot="additional_info"></span>
</div>
```

CodeSyntax

Slots dentro de slots

- Es posible definir slots dentro de slots, para tener mas opciones de personalización.

```
<div metal:define-macro="sidebar">
  <div metal:define-slot="links">
  Links
  <ul>
    <li><a href="/">Home</a></li>
    <li><a href="/products">Products</a></li>
    <li><a href="/support">Support</a></li>
    <li><a href="/contact">Contact Us</a></li>
    <span metal:define-slot="additional_links"></span>
  </ul>
  </div>
  <span metal:define-slot="additional_info"></span>
</div>
```

CodeSyntax

Macros y slots para definir la estructura de las páginas

- Podemos definir una página maestra que defina las macros y slots.
- De esta manera definimos la estructura básica que se utilizará en la aplicación.

CodeSyntax

Referencias

- ZPT básico
- ZPT avanzado