Twig, el motor de plantillas para PHP que separa el código HTML

A la hora de llevar a cabo un desarrollo web, la arquitectura MVC (Modelo – Vista –Controlador) es la que más aceptación tiene entre la comunidad de desarrolladores. En esta arquitectura, la vista es la encargada de proporcionar la interfaz gráfica con la que interactuar con los usuarios y para desarrollar esta parte visual podemos optar por el uso de etiquetas HTML mezclados con un lenguaje de programación a nivel de servidor como PHP o ASP, o bien por un sistema de plantillas, como es el caso de Twig, protagonista de este White Paper y que es la utilizada en el framework Symfony 2, del cual ya hablamos el pasado mes de marzo en el libro blanco "Framework para el desarrollo ágil de aplicaciones".

Qué es Twig

Como hemos comentado anteriormente, Twig es un motor de plantillas desarrollado para el lenguaje de programación PHP y que nace con el objetivo de facilitar a los desarrolladores de aplicaciones web que utilizan la arquitectura MVC el trabajo con la parte de las vistas, gracias a que se trata de un sistema que resulta muy sencillo de aprender y capaz de generar plantillas con un código preciso y fácil de leer. Actualmente el código se distribuye bajo licencia BSD y es utilizado por el framework Symfony 2, aunque puede ser utilizado directamente con proyectos desarrollados en PHP en el que no interviene ese framework.

```
{% extends 'BloggerBlogBundle::base.html.twig' %}

{% block title %}{{blog.title}}{% endblock %}

{% block body %}
    <article class="blog">
        <header>
            <div class="date"><time datetime="{{blog.created|date('c')}}">{{blog.created|date('l, F j, Y')}}</div>
            <h2>{{blog.title}}</h2>
        </header>

        <img src="{{ asset(['images/', blog.image]|join) }}" alt="{{ blog.title }}" class="large" />

        <div>
            <p>{{blog.blog}}</p>
        </div>
    </article>
{% endblock %}
```

Cuando hablamos de plantilla, nos estamos refiriendo a un archivo de texto que puede arrojar resultados en formatos como HTML, XML, CSV… y que está formado por expresiones de control y variables, las cuales serán reemplazadas por valores una vez que la plantilla sea evaluada.

Cuando nos referimos al lenguaje PHP, una de las plantillas más utilizadas es una plantilla PHP, en la que se mezcla texto interpretado por PHP y en el que se mezclan etiquetas HTML y código PHP para formar la vista que verá el usuario.

Un ejemplo de plantilla creada con PHP sería el siguiente que os dejamos.

```
<html>
    <head>
        <title>Bienvenido a acens!</title>
```

```
    </head>
    <body>
      <h1><?php echo $titulo ?></h1>

      <ul id="navigation">
        <?php foreach ($listaValores as $elemento): ?>
          <li>
            <a href="<?php echo $elemento->getURL() ?>">
              <?php echo $ elemento ->getTexto() ?>
            </a>
          </li>
        <?php endforeach; ?>
       </ul>
    </body>
</html>
```

Esa misma plantilla escrita con Twig nos permite obtener un resultado más conciso y sencillo de entender.

```
<html>
    <head>
      <title>Bienvenido a acens!</title>
    </head>
    <body>
      <h1>{{ titulo }}</h1>

      <ul id="navigation">
        {% for elemento in listaValores%}
          <li><a href="{{ elemento.url}}">{{ elemento.texto}}</a></li>
        {% endfor %}
       </ul>
    </body>
</html>
```

En este caso, se han eliminado las estructuras típicas de PHP y se han utilizado las que proporciona Twig. Entre las que aparecen, podemos destacar dos sintaxis especiales que aparecen:

- **{{……}}**: Se utiliza para imprimir el valor de una variable por pantalla.
- **{%....%}**: Se trata de una etiqueta que controla la lógica de la plantilla. Es utilizado cuando tengamos que utilizar bucles, estructuras if – else…

Además de las dos anteriores, también está disponible **{#.....#}** que es utilizada para dejar comentarios en el código, comentarios que no serán visibles en la parte pública pero que pueden ser de gran ayuda a la hora de entender el código.

Las principales características de Twig son:

- **Rápido**: se trata de un sistema de plantilla muy rápido debido a que Twig compila los templates a código PHP optimizado.
- **Seguro**: Twig dispone de un módulo de sandbox para evaluar el código no verificado mejorando de esta forma la seguridad.

- **Flexible**: por defecto este motor de plantillas trae predefinidas un amplio conjunto de etiquetas que pueden ser utilizadas por el usuario, pero también ofrece la posibilidad de que el propio usuario pueda definir otras nuevas etiquetas.

¿Por qué trabajar con Twig?

Muchos de los que estáis acostumbrados a trabajar con código PHP directamente en las plantillas podréis pensar que si de esta forma todo funciona bien, por qué cambiar. Pero son muchos los motivos por los que dar el paso y empezar a trabajar con Twig puede ser beneficiosos a la hora de desarrollar vuestros proyectos. Veamos a continuación algunos de estos motivos.

a) Conciso

El lenguaje PHP puede llegar a ser extremadamente conciso a la hora de mostrar por pantalla cierta información. Por ejemplo, para mostrar una variable en la plantilla utilizaríamos alguno de los siguientes métodos:

```
<?php echo $var ?>
<?php echo htmlspecialchars($var, ENT_QUOTES, 'UTF-8') ?>
```

Por el contrario, para hacer eso mismo, Twig utiliza una sintaxis muy concisa.

```
{{ var }}
{{ var|escape }}
```

b) Plantillas orientadas a sintaxis

Twig ofrece accesos directos a ciertos patrones comunes como mostrar un determinado texto cuando al iterar un array éste se encuentra vacío. Iterar es recorrer el array de objetos, la estructura donde está almacenada la información. Por ejemplo, tenemos un array que contiene 8 usuarios. Cuando lo recorremos, en cada iteración se coge uno de esos elementos y se trabaja con él. Luego en la siguiente iteración se trabaja con el siguiente y así sucesivamente hasta que recorreos todos los elementos que hay en el array.

```
{% for user in users %}
   * {{ user.name }}
{% else %}
   No hay ningún usuario.
{% endfor %}
```

Para lograr esto mismo con PHP, lo primero que deberíamos hacer es comprobar si el array está vacío. Si lo estuviera, se mostraría el mensaje y si no lo está, entonces recorremos ese array.

c) Potencia

Twig ofrece todo lo necesario para poder crear complejas plantillas de forma muy sencilla. Para ello nos proporciona sistemas de herencia múltiple, bloques de contenidos, escape automático…

d) Fácil de aprender

Utiliza una sintaxis sencilla y optimizada buscando en todo momento facilitar la tarea al diseñador web.

e) Flexible

Twig es lo suficientemente flexible para abordar proyectos tanto sencillos como complicados, gracia sobre todo a su arquitectura abierta que permite a los desarrolladores crear sus propias etiquetas.

f) Estable

Se trata de un proyecto totalmente estable y donde ha sido testeada cada una de las funcionalidades que ofrece.

g) Seguro

Ofrece algunos elementos únicos de seguridad:

- **Sistemas de escape automático**: Permite habilitar el escape automático global o bien para un determinado bloque.

  {% autoescape true %}
  {{ var }}
  {{ var|escape }} {# var no será doblemente escapado, en este caso ignora "escape" #}
  {% endautoescape %}

- **Sandboxing**: Twig puede evaluar algunos templates en un entorno de desarrollo donde el usuario tenga acceso a ciertos tags, filtros y métodos definidos por el desarrollador. Esto se puede habilitar de forma global o únicamente para una plantilla.
  {{ include('pagina.html', sandboxed = true) }}

- **Mensajes de error claros**: Cada vez que tenga un problema de sintaxis en una plantilla, Twig emite un mensaje de ayuda con el nombre del archivo y el número de línea donde se produjo el problema, facilitando mucho la depuración.

h) Rapidez

Para ofrecer mejores resultados de velocidad de carga, Twig compila las plantillas optimizando el código PHP. Para ello cachea en clases PHP todo el contenido de las plantillas lo que favorece al rendimiento de nuestra aplicación.

i) Sistema de herencia

Se trata de una de las características más destacadas que ofrece Twig y cuyo principal objetivo es la reutilización de código. Gracias a este sistema de herencia, el desarrollador puede crear una estructura base de la cual heredarán el resto de plantillas del proyecto. Más adelante veremos cómo trabajar con este sistema.

Instalación de las plantillas Twig

Veamos a continuación el proceso de instalación de este motor de plantillas para utilizarlo en nuestros proyectos. Recordad que en el caso de que estemos utilizando Symfony 2, este framework lo trae configurado por defecto para ser utilizado.

Lo primero que debemos hacer es descargarnos la última versión estable de Twig. Para ello accederemos a la página del proyecto y nos descargaremos la versión comprimida en ZIP o en formato TAR. Una vez descargada, lo descomprimimos. De la estructura de directorios que nos aparecerá, nos quedaremos con el contenido que hay dentro de la carpeta "lib".

Dentro de "lib" nos encontraremos una carpeta con el nombre de "Twig". La copiamos y la movemos a nuestro proyecto.

El siguiente paso será crearnos dos nuevas carpetas en nuestro proyecto. Una la llamaremos "**templates**" y otra "**cache**". En la primera de ellas será donde iremos almacenando todos los templates que vayamos creando en nuestro proyecto. En la carpeta "**cache**" será donde Twig irá almacenando las plantillas compiladas en PHP y que estarán listas para ser utilizadas.

El siguiente paso será registrar su cargador automático. Esto lo conseguimos con las siguientes líneas de código.

```
require_once '/ruta/a/Twig/Autoloader.php';
Twig_Autoloader::register();
```

Lo primero que hacemos es incluir en nuestro proyecto el archivo Autoloader.php para después hacer el registro del cargador. En este punto decir que habría que cambiar "**/ruta/a/**" por la ruta donde hayamos almacenado la carpeta Twig.

A continuación de esto, lo que tendremos que hacer es indicar donde se almacenará las plantillas. Esto lo conseguimos con la siguiente línea.

```
$loader = new Twig_Loader_Filesystem('/ruta/a/templates');
```

Lo siguiente es crearse un objeto Twig que será el que utilicemos para realizar la renderización de las plantillas. A este objeto le tendremos que indicar el objeto loader que hemos creado anteriormente además de un array de opciones en las que hay que indicar la ruta hacia la carpeta cache que hemos creado.

```
$twig = new Twig_Environment($loader, array(
    'cache' => '/ruta/a/cache',
));
```

Con esto ya hemos terminado la configuración de Twig para ser utilizado en nuestro proyecto. Ahora para hacer la llamada a una plantilla tendríamos que hacer algo similar a lo que os dejamos a continuación.

echo $twig->render('index.twig.html', array('empresa' => 'acens'));

En este ejemplo, estamos llamando a la plantilla "index.twig.html" a la que le pasamos la variable "empresa" que contiene el valor "acens".

Usos básicos de Twig

En puntos anteriores ya hemos visto el uso de la instrucción **{{…}}** para mostrar la información de las variables. Veamos a ahora otras funcionalidades que ofrece Twig a los desarrolladores.

1.- Modificar información
Es habitual querer modificar la información antes de ser mostrada a los usuarios. Puede ser que queramos que alguna información se muestre toda en mayúsculas limpiar el código HTML que pudiera tener una variable ya que esto podría afectar a como se vería. Para lograr esto, Twig nos ofrece los filtros, que siempre van detrás del nombre de la variable y separados por el carácter "**|**".

{{ variable | filtro}}

Entre los filtros que nos podemos encontrar están:

- **upper**: Muestra el contenido de una variable en mayúsculas.
- **lower**: Muestra el valor de una variable en minúsculas.
- **striptags**: Elimina cualquier etiqueta HTML que pudiera tener la variable.
- **date**: Muestra la fecha en el formato indicado que se le indicara, siguiendo el mismo estilo que la función date de PHP.
- **nl2br**: Transforma los saltos de línea en etiquetas
.
- **trim**: Elimina los espacios en blanco que pueda tener la variable tanto al principio como al final.
- **Capitalize**: Cambia la primera letra del texto a mayúsculas y el resto a minúsculas.

Estos son sólo algunos ejemplos, aunque en la documentación oficial de Twig podréis encontrar un listado más amplio.

2.- Mostrar información de las variables
Como ya hemos comentado en varias ocasiones, para mostrar el valor contenido en una variable se utiliza la estructura **{{….}}**, pero esto no funciona en caso de que la variable se trate de un objeto.

En caso de que la variable sea un objeto, para mostrar la información correspondiente a una de sus propiedades, habría que hacerlo de la siguiente manera **{{ variable.propiedad }}**. De esta forma, se mostraría el valor que tiene ese objeto para la propiedad indicada.

Por ejemplo, supongamos que tenemos un objeto usuario que almacena su nombre, dirección y teléfono. Si quisiéramos pintar el valor de su nombre sería de la siguiente manera.

{{ usuario.nombre }}

3.- Estructura de control for

El principal uso del for es para iterar sobre los elementos que contiene una colección de variables. Su forma de uso es la siguiente:

```
{% for variable in colección %}
        …….
{% endfor %}
```

Por ejemplo, si tenemos una colección de artículos y queremos mostrar el precio de cada uno de ellos, lo haríamos de la siguiente manera.

```
 {% for articulo in artículos %}
        {{ articulo.precio }}
{% endfor %}
```

Dentro de esta estructura, nos encontramos una variante llamada for …. else, y cuya estructura es la siguiente:

```
 {% for articulo in artículos %}
        {{ articulo.precio }}
{% else %}
        No existen artículos
{% endfor %}
```

Lo que permite esta estructura es que si la colección, en nuestro ejemplo "artículos", es vacío, entonces se ejecuta lo que hay dentro del **{% else %}**.

Además del funcionamiento que hemos comentado, la estructura for crea en su interior una variable especial llamada "loop" con la que podemos obtener información de cada iteración por medio de una serie de propiedades. Veamos algunas de ellas.

- **loop.index**: Nos indica el número de iteración en la que estamos.
- **loop.revindex**: Nos indica el número de iteraciones que faltan por ejecutarse.
- **loop.first**: Devuelve true si es la primera iteración.
- **loop.last**: Devuelve true si es la última iteración.
- **loop.length**: Nos devuelve el número total de iteraciones.

Estas propiedades pueden ser útiles a la hora de montar nuestras plantillas. Por ejemplo, supongamos que en nuestro array de artículos, en el primer elemento, queremos mostrar su precio en negrita. El código quedaría de la siguiente manera.

```
{% for articulo in artículos %}
    {% if loop.first%}<b>{% endif %}
    {{ articulo.precio }}
    {% if loop.first%}</b>{% endif %}
{% else %}
        No existen artículos
{% endfor %}
```

4.- Estructura de control if – else

Se trata de una estructura que funciona de forma similar a como lo suele hacer en cualquier otro lenguaje de programación. Se evalúa una condición y si se cumple se ejecuta el código que hay dentro del if. Si por el

contrario no lo cumple, se ejecutaría el que hubiera en el bloque else. Cabe destacar que no siempre hay por qué utilizar la instrucción else, es decir, nos podemos encontrar sólo el if.

Veamos a continuación cómo sería esta estructura.

```
{% if variable. propiedad%}
    ……
{% endif %}
```

O con el uso del else.

```
{% if variable. propiedad%}
    ……
{% else %}
    ……
{% endif %}
```

Normalmente la estructura if se combina con los operadores "is" e "is not" y algunos tests que incluye por defecto Twig. Algunos de estos tests son los siguientes:

- **constant(valor)**: Comprueba si la variable contiene un valor igual a la constante indicada.
- **divisibleby(numero)**: Mira si la variable es divisible por el valor indicado.
- **empty**: Comprueba si la variable es vacía.
- **even**: Evalúa si la variable es un número par.
- **odd**: En este caso nos indica si la variable es un número impar.
- **null**: Comprueba si la variable es null.

Por ejemplo, queremos que en nuestro listado de artículos, aquellos que su precio sea divisible por 2 se destaque en negrita, mientras que el resto se muestre de forma normal. El código sería el siguiente:

```
{% for articulo in articulos %}
   {% if articulo.precio is divisibleby(2) %}
        <b>{{ articulo.precio }}</b>
   {% else %}
        {{ articulo.precio }}
   {% endif %}
{% else %}
        No existen artículos
{% endfor %}
```

Además de estos tests, se pueden utilizar los típicos operadores que se suelen utilizar en los lenguajes de programación como son AND, OR, NOT, ==, <=, >=, ….

5.- Herencia de plantillas

La herencia es la característica más destacada que nos ofrece Twig ya que gracias a este mecanismo, podemos crear una estructura multinivel. Para ello Twig proporciona los bloques (block) y el método extends para indicar de qué plantilla se hereda.

Gracias a la herencia, podemos crear una plantilla base en nuestro desarrollo que contenga los elementos comunes que contendrán todas las plantillas. Un ejemplo de plantilla base podría ser el siguiente.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>{% block title %}Welcome!{% endblock %}</title>
    {% block stylesheets %}{% endblock %}
    <link rel="shortcut icon" href="{{ asset('favicon.ico') }}" />
  </head>
  <body>
    {% block body %}{% endblock %}
    {% block javascripts %}{% endblock %}
  </body>
</html>
```

En el código anterior, nos encontramos varios bloques definidos con la etiqueta {% block %} que serán los que luego se podrán sobreescribir cuando otra plantilla herede de ésta. Lo veremos más claro con un ejemplo un poco más adelante.

Cuando una plantilla hereda de otra, su primera etiqueta debe ser {%extends 'nombre_plantilla.html.tiwg'%}. Una vez que hemos añadido esta etiqueta, en la plantilla sólo podremos rellenar los bloques definidos en la plantilla de la que hereda.

Por ejemplo, si tenemos una plantilla que hereda de la que hemos indicado anteriormente, sólo podrá crear contenido dentro de los bloques llamados "title", "stylesheets", "body" y "javascript". Si se intenta crear un nuevo bloque o añadir código HTML fuera de esos bloques, Twig lanzaría un error.

Para añadir el contenido en la plantilla que hereda de la base, habría que indicar los bloques y dentro de ellos añadir el contenido. Por ejemplo, si queremos mostrar un texto dentro del bloque "title", lo deberíamos hacer de la siguiente manera:

```
{% extends 'base.html.twig'%}
{% block title%} Título de esta nueva página {% endblock %}
```

Aunque la plantilla de la que se herede tenga más bloques, no es necesario sobreescribir todos. Si no se sobreescribe alguno de ellos, se mostrará la información que tuviera la plantilla padre en ese bloque.

Veamos a continuación un ejemplo. Supongamos que tenemos una aplicación que dependiendo del país de donde se acceda, mostraría una estructura u otra.

Lo primero que deberemos hacer es crearnos nuestra plantilla base.html.twig.

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <title>{% block titulo %}{% endblock %}</title>
    <link href="estilos.css" rel="stylesheet" type="text/css" />
  </head>

  <body>
    {% block contenido %}{% endblock %}
  </body>
</html>
```

Ahora nos crearemos una plantilla, a la que llamaremos, nacional.html.twig, que será la que cargue cuando se acceda desde España. Como hemos comentado anteriormente, sólo hay que sobreescribir los bloques. El

resto del código que esté fuera de los bloques será similar para todas las plantillas. Veamos el código de esta plantilla.

```
{% extends 'base.html.twig' %}
{% block titulo%}Portada para España{%endblock%}

{%block contenido%}
        <h1>Últimas noticias en España</h1>
        ……
{%endblock%}
```

Como podemos ver, esta segunda plantilla que ha heredado de la "base" es mucho más sencilla, ya que no hay que incluir todo el código HTML que forma parte del esqueleto.

Ahora veamos el código de la plantilla que se mostraría cuando se accediera desde fuera de España y que llamaremos internacional.html.twig.

```
{% extends 'base.html.twig' %}
{% block titulo%}Portada Internacional{%endblock%}

{%block contenido%}
        <div id="sidebar">
            ……
        </div>

        <div id="content">
            <h1>Últimas noticias Internacionales</h1>
        </div>
{%endblock%}
```

En este caso, como podemos ver, la plantilla es totalmente a la de nacional.html.twig. En este caso, dentro del bloque "contenido" hemos creado dos zonas: un sidebar y un content donde se mostrará las últimas noticias.

Este es un ejemplo sencillo de herencia, pero sirve para ver todo el potencial que nos ofrece este mecanismo.

A lo largo de todo este White Paper hemos visto los beneficios que podemos obtener al utilizar este motor de plantillas, así como algunas de las funcionalidades básicas que nos ofrece, funcionalidades que seguro iréis ampliando una vez que empecéis a trabajar con este motor de plantillas.