

Desarrollo de aplicaciones PHP con Prado

RESCATE WEB



El entorno de desarrollo para PHP Prado nos permite la elaboración rápida de aplicaciones web. **POR PETER KREUSSEL**

Tanto *Google Docs* como otras herramientas de producción en línea revelan el potencial de las aplicaciones de escritorio apoyadas en el modelo de programación web. Al conservar la aplicación en el servidor, se minimiza la configuración a la vez que se optimiza el uso de recursos del sistema. Pero el modelo *Software como Servicio* no se limita a enormes y complicadas aplicaciones confinadas en los servidores de gigantes como Google. Podemos tener un servicio de software en forma de aplicación web personalizada ejecutándose a nivel local. Los desarrolladores suelen usar lenguajes como *PHP* para la creación de programas pequeños y eficientes del lado del servidor, cuyos resultados se muestran en un navegador.

Para el desarrollo de aplicaciones web eficientes pueden resultarnos útiles algunos de los bloques prefabricados de un entorno de trabajo. Existen varios entornos de desarrollo disponibles en *Linux*. Un ejemplo de esta nueva raza de herramientas de programación para *PHP* es Prado [1], un entorno de trabajo maduro y amplio

para aplicaciones web. *Prado*, que significa *PHP Rapid Application Development Object-oriented*, trabaja en muchas facetas del desarrollo de aplicaciones, ofreciendo componentes GUI que van desde simples campos de entrada hasta un interfaz de asistente. El lenguaje de plantillas *XML* incluido con *Prado* permite una sintaxis sencilla que puede gestionarse fácilmente desde cualquier editor de *HTML*. Todos los componentes para formularios guardan un registro de su propio estado, sin que el programador tenga que preocuparse por ellos. *Prado* también soporta controladores de validación del lado del cliente que comprueban los datos de entrada antes de enviar el formulario.

Diseño por Componentes

La arquitectura de *Prado Framework* se

basa en el modelo *MVC* (modelo-vista-controlador). Cada aplicación Prado está formada por una o más plantillas que definen el aspecto de la página.

Un archivo *PHP* proporciona la lógica del programa. Cada plantilla puede incluir archivos de código u otros archivos de plantilla.

Con una aplicación de ejemplo, que implementa un calendario de eventos basado en web, se muestra lo fácil que es programar con *Prado*. Las puertas del calendario se abren al pulsar con el ratón (Figura 1). Las que ya están

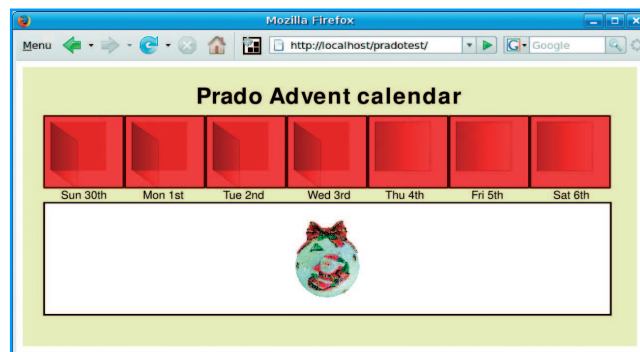


Figura 1: Una aplicación interactiva simple: las puertas del calendario se abren al pulsar con el ratón; el contenido aparece en el cuadro inferior.

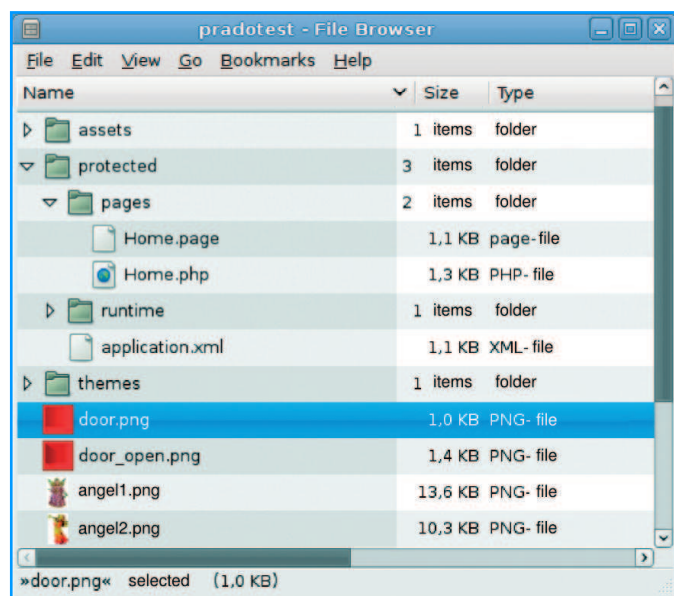


Figura 2: Siempre en pareja: La página de Prado consta de una plantilla XML y un archivo PHP con el mismo nombre que contiene la lógica de la aplicación.

abiertas no hacen nada, por lo que la aplicación debe guardar el estado de la puerta. El cuadro de debajo de las puertas muestra el contenido del compartimento del calendario.

La aplicación debería reusar tanto código como le sea posible; el código para las puertas, así como los componentes de la vista y del controlador, deberían aparecer sólo una vez. Debido a la limitación de espacio, sólo se muestra una puerta por semana.

Para Empezar

Para instalar *Prado* en un servidor web sólo hemos de descomprimirlo en la raíz de éste. Las aplicaciones Prado presuponen una estructura de directorios específica en la que sólo se accede a un archivo de índice central. El verdadero código se encuentra en el directorio *protected*, que se encuentra bloqueado por el archivo *.htaccess*.

Ejecutamos un script en *PHP* de la línea de comandos desde el directorio en el que se encuentra la aplicación usando el nombre de ésta como parámetro. Se crea entonces la estructura de directorios necesaria.

Los componentes de vista y controlador de la nueva aplicación vienen dados por un par de archivos de dentro del directorio protegido, llamados *Applicationname.page* y *Applicationname.php*. Los Listados 1 y 2 muestran los contenidos de la aplicación de calendario.

El componente de la vista (Listado 1) define un formulario *HTML* en la primera línea. Las líneas 8-14 crean una puerta. La etiqueta de plantilla `<com:TImageButton ... >` (línea 9) enlaza la imagen para las puertas cerradas.

Un bloque *div* con el atributo de estilo *CSS* `float:left` hace que los compartimentos del calendario se alineen correctamente. Los archivos de plantilla pueden obtener ambas etiquetas, las *Prado* (reconocibles por el prefijo `<com: >`) y las *HTML*.

Multiplicador

Una etiqueta *div* adicional, bajo la imagen, centra el texto de la fecha. El código PHP incluido se encarga del

texto, que cambia cada día. Las etiquetas `< %# ... % >` indican a *Prado* que el texto incluido es código fuente en *PHP* y no código de plantilla.

Para comprenderlo mejor podemos echar un ojo a la etiqueta `<com:TRepeater>` de la línea 4. Una etiqueta de repetición, tomada de *ASP.Net*, se parece a un bucle *foreach()*. Esto es, itera por el texto encerrado de acuerdo con el número de entradas de datos de la fuente de datos enlazada desde el código controlador. La variable `$this` referencia al repetidor en este contexto.

Cada repetidor de Prado proporciona el atributo `Data[]`, que no es más que un array que contiene los datos para la correcta iteración del bucle.

Los repetidores de Prado ofrecen una funcionalidad mayor que la de un simple *foreach()* de *PHP*: Las áreas de cabecera y pie de página se muestran una sola vez. Tan sólo se repite el área comprendida entre `<prop:ItemTemplate >` y `</prop:ItemTemplate >`.

Si un bloque está marcado con `<prop:ItemAlternatingTemplate >`, los contenidos de *ItemTemplate* y *ItemAlternatingTemplate* se alternan. Como

Listado 1: Componente de Vista de Advent Calendar

```

01 <!-- Formulario -->
02 <com:TForm>
03   <!-- Repetidor ("Datagrid") -->
04   <com:TR ID="Repeater"
05     OnItemCommand="open_door" > <!-- todos los eventos de
repetidor -->
06     <!-- Código iterado: Imagen, etiqueta y campo oculto -->
07     <prop:ItemTemplate>
08       <div style="float:left">
09         <com:TImageButton ID="door" ImageURL="door.png" />
10         <div align="center">
11           <com:TLabel ID="tbox" text="< %# $this->Data['text']
%>" />
12         </div>
13       </div>
14       <com:THiddenField ID="inside" data="< %#
$this->Data['inside'] %> " />
15     </prop:ItemTemplate>
16     <!-- Final del código iterado -->
17   </com:TRepeater>
18
19   <!-- Área de visualización del contenido del calendario -->
20   <div style="clear:left;border-width:2px; border-style:solid;
background-color:#ffffff;
21     width:650px; height:100px; text-align:center; padding:20px;
border-color:black;">
22     <com:TImage id="content" />
23   </div>
24 </com:TForm>

```

resultado, las líneas de la tabla alternan su color (ver el primer ejemplo del tutorial de inicio rápido de *Prado* [4]). En el ejemplo se repiten las líneas 11 a 14.

Fuentes de Datos

Las líneas 23 a 27 (Listado 2) enlazan al repetidor la fuente de datos. En ambos casos, *\$this* referencia a la clase actual, y por tanto al objeto de página. Dentro de un objeto de página, los métodos implementados van parejos a las etiquetas de *Prado*, cuyo indicativo es el atributo *ID*. Por tanto, *\$this->Repeater* apunta al repetidor, que podemos distinguir en *Home.page* por el atributo *ID="Repeater"*. El atributo *DataSource* de la clase *Repeater* contiene los datos. El código los asigna al valor que retorna el método *getData()*.

Las aplicaciones web más demandadas obtienen sus datos de una base de datos, que *Prado* enlaza a través de su *Object Relational Mapper* o de las llamadas *ActiveRecord*. En el ejemplo, los datos originados desde el array. La condición *!\$this->isPostBack* sólo se cumple cuando la aplicación carga la página por primera vez, no cuando se recarga tras la acción de un usuario. El repetidor se inicia por tanto a través del método *dataBind()* con valores

estándar sólo la primera vez que se carga la página. Dado que los componentes de *Prado* ya tienen completo su estado tras la inicialización, el repetidor conserva su estado, incluso cuando se recarga la página continuamente. La funcionalidad de las sesiones se gestiona automáticamente en el entorno de trabajo, por lo que no hace falta ningún código en los componentes de vista o de controlador.

El segundo método de la clase *open_door()* reemplaza la imagen de la puerta cerrada por la de la puerta abierta (Figura 1), algo que se consigue cambiando la imagen de una instancia replicada por el repetidor para la puerta en el calendario (Listado 1, línea 9).

Si *open_door()* cambia la URL, la variante cambiada permanece constante cada vez que se carga la página. Es decir, una vez abierta la puerta no vuelve a cerrarse.

¿Quién Apunta a Quién?

Cuando el usuario pincha en una puerta dada, ésta debe abrirse. Una forma de garantizar una correcta asociación es diferenciar las distintas instancias de los componentes del calendario en lo que se refiere a los *IDs* generados dinámicamente y que el manejador de eventos *OnClick* recibe

como parámetro. De todos modos, *Prado* nos proporciona una opción más simple: El manejador de eventos *OnItemCommand* (Listado 1, línea 5) capta los eventos de todos los elementos de control del repetidor.

Prado le pasa el objeto a la función llamada como segundo parámetro, *\$param* en este ejemplo. El método *getItem()* devuelve una referencia a la iteración del bucle en que se llamó al manejador. De este modo, uno puede referirse a instancias del objeto de imagen (creadas por la repetición) con el *ID* de puerta estático (Listado 1, línea 9), en términos de valor devuelto de *getItem()* e *ID* estático. *\$item->door->setImageURL()* siempre apunta a la imagen sobre la que pulsó el usuario. Entonces el método *setImageURL()* cambia las imágenes.

Durante la iteración, los repetidores pasan los datos a los objetos asociados, a quienes hacen peticiones mediante la llamada *\$this->Data['nombredelcampo']*. Después de mostrar la página, los datos dejan de estar disponibles en el repetidor. De cualquier modo, el campo oculto de la línea 14 (Listado 1) almacena los valores pasados. Podemos tratar este campo como las imágenes iteradas a través del objeto de repetición del bucle y su *ID* estático.

Listado 2: Componente de Modelo de Advent Calendar

```

01 <?php
02
03 // Clase de la página de inicio
04 class Home extends TPage {
05
06 // Esta función proporciona los datos
07 protected function getData()
08 {
09     return array( // texto y URLs a la imágenes de
10         array('text' => 'Sun 30th.', 'inside' =>
11             'angell.png'),
12         array('text' => 'Mon 1st', 'inside' =>
13             'bells.png'),
14         array('text' => 'Tue 3rd', 'inside' =>
15             'bug.png'),
16         array('text' => 'Wed 4th', 'inside' =>
17             'ball.png'),
18         array('text' => 'Thu 5th', 'inside' =>
19             'angel2.png'),
20         array('text' => 'Fri 6th', 'inside' =>
21             'star1.png'),
22         array('text' => 'Sat 7th', 'inside' =>
23             'star2.png'),
24     );
25 }
26
27 // Inicializar el repetidor (Data grid)
28 public function onLoad($param) {
29     if (!$this->isPostBack) { // sólo la primera
30         vez
31             // Asigna la función como fuente de datos
32             $this->Repeater->DataSource=$this->getData();
33             $this->Repeater->dataBind(); //Inicializar el
34             repetidor
35         }
36     }
37     // Cambiar la imagen al abrir la puerta
38     public function open_door($sender, $param)
39     { // Al pinchar en la puerta, el manejador
40         obtiene un objeto...
41         $item=$param->getItem(); // ... que contiene
42         el objeto llamado
43         $item->door->setImageURL('door_open.png'); //
44         nueva imagen de la puerta
45         $image=$item->inside->getData(); // URL de la
46         imagen para el repetidor ...
47         $this->content->ImageUrl=$image; // ...
48         mostrar bajo el calendario
49     }
50 }
51 }
52 }
53 }
54 }
55 }
56 }
57 }
58 }
59 }
60 }
61 }
62 }
63 }
64 }
65 }
66 }
67 }
68 }
69 }
70 }
71 }
72 }
73 }
74 }
75 }
76 }
77 }
78 }
79 }
80 }
81 }
82 }
83 }
84 }
85 }
86 }
87 }
88 }
89 }
90 }
91 }
92 }
93 }
94 }
95 }
96 }
97 }
98 }
99 }
100 }

```

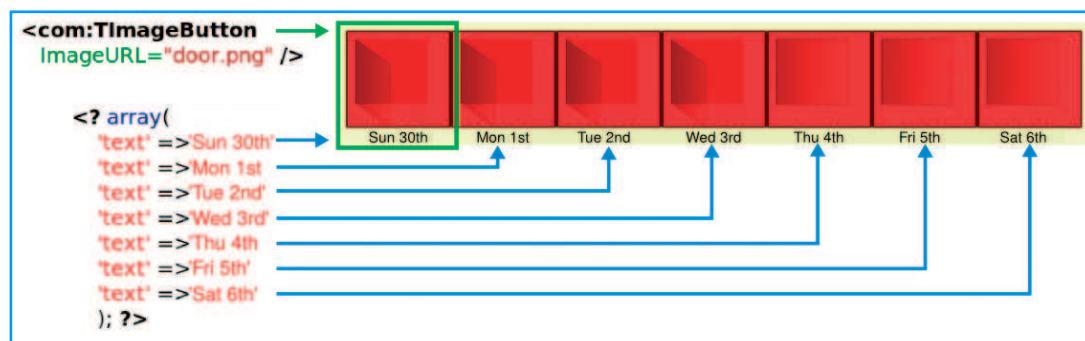


Figura 3: El lenguaje de plantilla de Prado contiene estructuras de control que replican los elementos de visualización y proporcionan las instancias individuales con datos obtenidos de arrays o listas.

`$item->inside->getData()` lee su contenido, mientras que `$this->content->ImageURL` define la URL de la hasta ahora vacía etiqueta `<com:TImage id="content" />` de la línea 22. Los contenidos del compartimento recién abierto aparecen en el calendario.

Amplia Base

Con el ejemplo del calendario de eventos sólo tenemos una primera impresión del extenso entorno de trabajo para PHP. Prado soporta localización, temas y pieles.

Prado Framework ofrece también acceso a servicios web SOAP, además contiene clases que facilitan el manejo de errores e integra SafeHTML Parser [5], útil para evitar ataques de XSS.

Prado almacena el estado de la página en campos de formulario ocultos que el usuario puede ver y manipular. Como contramedida, dependiendo de la configuración, Prado contrasta estos formularios con sus campos de datos internos y cookies basadas en claves guardadas en el servidor.

Una caché integrada guarda el análisis de las plantillas cada vez que se carga la página. Con un entorno de trabajo tan extenso como Prado, PHP tiene que cargar muchos archivos de Include cada vez que se genera una página. El archivo `pra-dolight.php` contiene todo el entorno de trabajo en un único archivo que, liberado de todos los comentarios, ocupa sólo unos 2.500 KB.

La aplicación de calendario de eventos funciona sin Ajax, recargando la página con cada cambio de estado. A mediados de 2007 se intro-

dujeron, con Prado 3.1.0, controles activos. El manejo de estos controles no siempre resulta sencillo. Cuando los insertamos individualmente en una plantilla como elementos estáticos, con la excepción de la recarga de la página guardada, se comporta como las clásicas etiquetas de Prado, que necesitan recargar la página con cada actualización.

La cosa se complica cuando el desarrollador desea combinar los controles activos con los repetidores y sus propios manejadores de eventos. Esta capacidad no funciona tal cual, como lo puedan hacer los elementos estáticos de control. Es más, el desarrollador debe iniciar el contacto asíncrono con el servidor de forma manual. No se documenta sistemáticamente el funcionamiento en el sitio web de Prado. Hay muchos enlaces en las páginas del tutorial que no llevan a ninguna parte. Con un poco de suerte, sólo una búsqueda en el foro nos proporcionó la información necesaria.

Con unos 4000 usuarios registrados, el foro de Prado ofrece al desarrollador una valiosa asistencia. De todos modos, éste no sustituye a un documento conciso, especialmente para los novatos. La mayor parte de lo que hay consiste en documentación de la clase del entorno creada automáticamente. En algunos casos, son una o dos las frases que explican el funcionamiento de la clase.

También hay disponible un tutorial de inicio rápido que describe los aspectos más importantes de Prado y que contiene además una aplicación de ejemplo. Para resolver problemas específicos, especialmente con los elementos de Ajax, el tutorial no profundiza lo suficiente. Aparte de las

carencias relacionadas con la funcionalidad de Ajax, disponible únicamente desde hace aproximadamente año y medio, la documentación enturbia la que de otro modo sería una impresión general positiva sobre Prado.

Curiosamente, Ohloh.net [7] certi-

fica que el entorno de trabajo cuenta con un equipo relativamente grande, con 11 desarrolladores principales, además de con un código fuente bien comentado.

Alternativas

Prado es un entorno de trabajo amplio que facilita las áreas más importantes del desarrollo de aplicaciones con PHP. Con su eficiente lenguaje de plantillas y su capa de abstracción para bases de datos, Prado ofrece una arquitectura MVC estable. Sin embargo, los elementos Ajax (o Active Controls) no están bien integrados aún en el entorno. Alternativas a Prado, como Cakephp [8] o SilverStripe [9], integran un CMS así como un entorno de desarrollo de aplicaciones. ■

RECURSOS

- [1] Prado: <http://www.pradosoft.com>
- [2] Conexiones a bases de datos: <http://www.pradosoft.com/demos/quickstart/?page=Database.DAO>
- [3] Active Record: <http://www.pradosoft.com/demos/quickstart/?page=Database.ActiveRecord>
- [4] Repetidor: <http://www.pradosoft.com/demos/quickstart/?page=Controls.Repeater>
- [5] SafeHTML: <http://pixel-apes.com/safehtml>
- [6] Controles Activos: <http://www.pradosoft.com/demos/quickstart/?page=ActiveControls.Home>
- [7] Prado en Ohloh.net: <http://www.ohloh.net/projects/3182>
- [8] Cakephp: <http://cakephp.org>
- [9] SilverStripe: <http://www.silverstripe.com>