

Drupal 7 Desarrollo de Módulos

Crea tus propios Drupal 7 módulos desde cero

**Matt Carnicero**

**Greg Dunlap**

**Matt Farina**

**Larry Garfield**

**Ken Rickard**

**Juan Albin Wilkins**



BIRMINGHAM - MUMBAI

Drupal 7 Desarrollo de Módulos

Copyright © 2010 Packt Publishing

Todos los derechos reservados. Ninguna parte de esta publicación puede ser reproducida, almacenada en un sistema de recuperación o transmitida en cualquier forma o por cualquier medio, sin el permiso previo y por escrito del editor, excepto en el caso de citas breves incluidas en artículos críticos o revisiones.

Cada esfuerzo se ha hecho en la preparación de este libro para garantizar la exactitud de la información presentada. Sin embargo, la información contenida en este libro se vende sin garantía, ya sea expresa o implícita. Ni los autores, ni Packt Publishing y sus concesionarios y distribuidores serán responsables de cualquier daño causado o presuntamente causado directa o indirectamente por este libro.

Packt Publishing ha esforzado en proporcionar información acerca de la marca de todas las compañías y productos mencionados en este libro por el uso apropiado de los capitales. Sin embargo, Packt Publishing no puede garantizar la exactitud de esta información.

Primera edición: Diciembre de 2010

Producción de Referencia: 1301110

Publicado por Packt Publishing Ltd. 32 Lincoln Road

Olton

Birmingham, B27 6PA, Reino Unido. ISBN 978-1-849511-16-2

www.packtpub.com

Imágen de Portada por Vinayak Chittar (vinayak.chittar @ gmail.com)

Crédits

**Autores**

Matt Carnicero

Greg Dunlap

Matt Farina

Larry Garfield

Ken Rickard

Juan Albin Wilkins

**Revisores**

Davy Van Den Bremt

David Myburgh

Jojodae Ganesh Sivaji

**Editor de Adquisición**

Sarah Cullington

**Editores de Desarrollo**

Mayuri Kokate

Susmita Panda

**Editores Técnicos**

Vanjeet D'souza

Harshit Shah

**Copy Editor**

Neha Shetty

**Editorial Team Leader**

Akshara Aware

**Jefe de Equipo**

Priya Mukherji

**Coordinador del Proyecto**

Srimoyee Ghoshal

**Corrector de pruebas**

Aaron Nash

**Indexadores**

Tejal Daruwale

Hemanjini Bari

**Gráficos**

Nilesh R. Mohite

**Coordinador de Producción**

Aparna Bhagat

**Cubierta de Trabajo**

Aparna Bhagat

Prefacio

Drupal tiene sus raíces en el proyecto de pasatiempo humilde de Dries Buytaert, líder del proyecto Drupal, entonces un estudiante universitario. En un principio se creó un pequeño sitio de noticias y

tablón electrónico de modo que él y sus amigos pudieran mantenerse en contacto. Cuando llegó el momento de la graduación, este pequeño script fue puesto en la red pública, y un pequeño pero vibrante comunidad de desarrolladores, hackers, manitas, y los innovadores comenzó a reunirse allí. El script capaz de alimentar el sitio web fue abierta proviene de "Drupal" en enero de 2001, y atrajo la atención debido a su extensibilidad y la arquitectura modular.

Desde entonces, tanto el proyecto Drupal y su comunidad han explotado en crecimiento. La comunidad se compone actualmente de más de 700.000 personas en todo el mundo. Drupal también ahora los poderes más de 1% de la red, incluidos los sitios web de nombres tan conocidos como

whitehouse.gov y grammy.com.

Mi posición actual en la comunidad Drupal es la del Administrador de estreno de la última versión de Drupal, la versión 7. Dries Buytaert y trabajo conjunto con el equipo colaborador principal para ayudar a priorizar las iniciativas, animar a la gente atacando a problemas similares para trabajar juntos, actúan como revisores de control de calidad finales sobre parches, y en última instancia, confirmó cambios en el proyecto una vez que estén listos.

Drupal 7 representa un enorme salto hacia adelante de las versiones anteriores. El equipo contribuyente núcleo juntos tomó una mirada muy seria a las limitaciones de Drupal, desde casi todos los ángulos. La investigación Las pruebas de usabilidad se hizo en varias universidades, destacando los problemas de larga data con la interfaz de usuario de Drupal, y un equipo de usabilidad surgió para hacer frente a los problemas. Ingenieros colaboraron juntos para identificar y diseccionar severas limitaciones de la API que habían plagado las versiones anteriores. El equipo de control de calidad puesto enormes esfuerzos detrás de la integración de las pruebas automatizadas en nuestro flujo de trabajo de desarrollo, mejorando enormemente nuestra capacidad para refactorizar partes del sistema. Comunidad de diseño de Drupal intensificó y se convirtió en vocal acerca de las limitaciones de Drupal en el lado de la tematización que provocan que se vayan acudiendo en masa a otros marcos.

Un equipo de la accesibilidad surgió, no sólo empujando hacia adelante parches para mejorar el cumplimiento de las Pautas de Drupal, sino también educar a los miembros de la comunidad acerca de la accesibilidad. Drupal 7 es una versión notable por varias razones, pero sobre todo por la diversidad del equipo involucrado en su creación.

Como resultado de todo este esfuerzo, sin embargo, hay muy poco en Drupal 7 que no ha cambiado con respecto a versiones anteriores. La capa de abstracción de base de datos ha sido completamente reescrito y ahora está basado en los PHP Data Objects (PDO) de la biblioteca, lo que introduce una nueva sintaxis orientada a objetos para consultas. Además de las formularios y de cierto contenido, como nodo y los datos del usuario, en Drupal 7 la totalidad de la página se basa en matrices renderizable, que tiene enormes (y emocionante) Consecuencias para los temas.

Adición de campos de metadatos a varias entidades del sistema, ahora es posible a través del campo integrado de Drupal 7 y API entidad, que requiere previamente un módulo adicional, y se limitó a sólo ser capaz de ampliar el contenido. Hay literalmente cientos de otras mejoras bajo el capó.

El libro Drupal Desarrollo Módulo 7 ofrece un enfoque basado en proyectos que guía a los lectores a través de los conceptos más importantes, nuevas y modificadas en profundidad, lo que le pongas esto en práctica. Los autores de esta edición del libro tiene mucho más que "simplemente" un profundo conocimiento de Drupal 7 internos-en muchos casos, que, literalmente, escribió los mismos parches que ponen los internos en su lugar. Larry Garfield es el principal arquitecto detrás de la nueva capa de abstracción de base de datos orientada a objetos de Drupal 7, y de mantenimiento del sistema de base de datos del núcleo de Drupal. Juan

Wilkins ha diseñado la mayor parte de las mejoras a los archivos de plantillas y componentes internos del sistema el tema en Drupal 7, basado en gran parte en su trabajo de vanguardia en el tema del zen. Ken Rickard lanza con cabeza de numerosas mejoras en el sistema de acceso del nodo de Drupal 7 después de explorar sus límites exteriores de su contribuyó Dominio acceso y módulos de menú de acceso. Matt Farina ayudó con numerosas mejoras de JavaScript fundamentales, incluidas la alterabilidad de CSS y JavaScript, y el rendimiento para el usuario. El trabajo de Greg Dunlap con documentación de la API de núcleo tiene muchas veces descubrían particularmente difíciles de encontrar errores.

Es mi sincera esperanza que este libro encuentra muchos lectores ávidos que son capaces de no sólo ampliar Drupal 7 para satisfacer sus necesidades específicas, sino también unirse a nuestra comunidad de desarrollo dinámico de contribuir de nuevo lo que han aprendido y ayudar a que Drupal aún mejor.

**Angela Byron**

Drupal 7 Core Mantenedor

Secretario de la Asociación Drupal

Acerca de los autores

**Matt Carnicero** es un desarrollador web y autor. Ha escrito otros cinco librospor Packt, incluyendo Drupal 6 JavaScript y jQuery y Aprendizaje Drupal 6 Módulo de Desarrollo. Matt es un desarrollador senior en ConsumerSearch.com (a About.Com / empresa New York Times), donde trabaja en uno de los mayores sitios de Drupal en el mundo. Matt está activo en la comunidad de Drupal, la gestión de varios módulos. Él también lleva un par de proyectos de Open Source: QueryPath.



Me gustaría dar las gracias a Larry, Ken, Sam, Matt, Greg, y John para trabajar conmigo en el libro. Son un fantástico grupo de personas para trabajar. También me gustaría agradecer a los revisores técnicos de este libro, todos los cuales han contribuido a hacer de este un mejor trabajo.

También me gustaría dar las gracias a Austin Smith, Brian Tully, Chachi Kruel, Marc McDougall, Theresa Summa, y ​​el resto de la ConsumerSearch. equipo com por su apoyo. La gente de Palantir.net fueron fundamentales para conseguir este libro fuera de la tierra, y yo siempre estoy agradecido por su apoyo. Finalmente, Angie, Anna, Claire, y

Katherine han sacrificado algunos fines de semana y por las noches conmigo en beneficio de este libro. A ellos, les debo la mayor deuda de gratitud.



**Greg Dunlap** es un ingeniero de software con sede en Estocolmo, Suecia. Durante la última15 años, Greg ha estado involucrado en una amplia variedad de proyectos, incluyendo las aplicaciones de bases de datos de escritorio, kioscos, software embebido de pinball y máquinas tragamonedas, y sitios web en más de una docena de lenguajes de programación. Greg ha estado muy involucrado con Drupal por tres años, y es el encargado del mantenimiento de los módulos de implementar y Servicios, así como un orador frecuente en conferencias de Drupal. Greg es actualmente un director Desarrollador de Software en nodeone.



Varias personas jugaron un papel crucial en mi desarrollo como contribuyente Drupal, proporcionando apoyo y aliento justo cuando más lo necesitaba. Mi más profundo agradecimiento a Gary Love, Jeff Eaton, Boris Mann, Angie Byron, y Ken Rickard por ayudarme a patear a un nivel superior. Agradecimiento especial adicional a la encantadora Roya Naini para poner

con noches y fines de semana perdidos en el servicio de terminar mis capítulos.



**Matt Farina** ha sido un desarrollador Drupal desde 2005. Él es un front-end de alto niveldesarrollador, ingeniero y director técnico de Palantir.net, donde trabaja en una amplia variedad de proyectos que van desde museos hasta grandes sitios interactivos. Es colaborador de núcleo de Drupal, así como un encargado del mantenimiento de múltiples contribuido módulos de Drupal.

Matt escribió su primer programa de ordenador, cuando estaba en el quinto grado. Desde entonces, ha programado en más de una docena de idiomas. Él tiene una licenciatura en Ingeniería Eléctrica de la Universidad Estatal de Michigan.

**Larry Garfield** es un Arquitecto Superior e Ingeniero en Palantir.net, un líder en Drupalempresa de desarrollo con sede en Chicago. Él ha sido la construcción de sitios web desde los 16 años, lo cual es más de lo que le gustaría admitir, y ha estado trabajando en PHP desde 1999.

Encontró Drupal en 2005, cuando Drupal 4.6 todavía estaba nuevo y fresco, y nunca se fue. Él es el arquitecto principio y mantenedor del subsistema de base de datos de Drupal, entre otras iniciativas esenciales y módulos de terceros.

Anteriormente, Larry era un desarrollador de Palm OS y un periodista que cubre el sector de la electrónica móvil y fue el editor técnico para la construcción de sitios web potente y robusto con Drupal 6, también de Packt. Posee una Licenciatura y Maestría en Ciencias de la Computación de la Universidad DePaul.



Si tuviera que agradecer a todas las personas que hicieron posible este libro se necesitarían varias páginas, como la lista colaborador Drupal 7 era bastante más de 700 personas, la última vez que lo comprobé. En lugar de ello me limitaré a decir gracias a toda la comunidad por ser tan vibrante, de apoyo, y todos-en torno increíble que todavía trae una lágrima a mis ojos, a veces incluso después de media década.

Extra especial agradecimiento a Dries Buytaert, no sólo por ser nuestro líder del proyecto, pero para sentarse en el suelo a mi lado en

DrupalCon Sunnyvale y animarme a correr con esta loca idea que tenía, sobre el uso de esta cosa "DOP" para la capa de base de datos de Drupal. Dudo que él se dio cuenta de la cantidad de problemas que lo había causo en los próximos años.

Por supuesto, a mis padres, que me inculcaron no sólo el amor por el aprendizaje, sino un nivel de pedantería y la obstinación sin la cual yo nunca habría sido capaz de llegar tan lejos en Drupal, por no hablar de este libro.



**Ken Rickard** es un programador senior en Palantir.net, una firma con sede en Chicagoespecializada en el desarrollo de sitios web con Drupal. Él es un contribuidor frecuente al proyecto Drupal, y es el encargado de mantener el acceso de dominio, MySite y módulos API nodo del menú. En Palantir, que los arquitectos y construye sitios web de gran escala para una amplia gama de clientes, incluyendo la revista Foreign Affairs, NASCAR, y la Universidad de Chicago.

Desde 1998 hasta el 2008, Ken trabajó en la industria de la prensa, que comenzó su carrera administración de sitios web y más tarde convertirse en un investigador y consultor de Morris

DigitalWorks. En Morris, Ken ayudó a lanzar BlufftonToday.com, la primera página web del periódico lanzado en la plataforma Drupal. Más tarde dirigió el equipo de desarrollo de Drupal para SavannahNOW.com. Fue cofundador de la Prensa en grupo (Drupalhttp://groups.drupal.org/newspapers-on-drupal) Y es un asesor frecuente a las industrias de periódicos y editoriales.

En 2008, Ken ayudó a iniciar la Iniciativa Drupal Knight, un proceso de donación abierto para el desarrollo de Drupal, financiado por la John L. y James S. Fundación Knight. Él es también un miembro de la junta asesora de PBS Engage, un proyecto de la Fundación Knight para traer los medios sociales para el Public Broadcasting Service.

Antes de este libro, Ken era un revisor técnico para la editorial Packt Drupal 6 Sitio Planos de Timi Ogunjobi.



Debo agradecer a todo el personal de Palantir, la comunidad de Drupal, y, sobre todo, a mi esposa encantadora y paciente Amy, sin los cuales nada de esto sería posible.



**Juan Albin Wilkins** ha sido un desarrollador web durante mucho tiempo. En abril de 1993,él era uno de los pocos afortunados de utilizar el primer navegador gráfico, Mosaic

. 1.0, y que ha estado haciendo el desarrollo web profesional desde 1994 En 2005, John finalmente aprendió lo idiota que era para construir su propio framework de aplicaciones web, y descubrió el poder de Drupal; él nunca miró hacia atrás.

En la comunidad de Drupal, que es más conocido como JohnAlbin, uno de los 20 principales contribuyentes a Drupal 7 y el mantenedor del tema Zen, que es una temática altamente documentado rica en características "starter" con un marco CSS poderosamente flexible.

También ha escrito varios módulos de servicios públicos orientados al front-end, tales como el módulo de menú Bloquear. John trabaja actualmente con un grupo de desarrolladores muy cool Drupal, diseñadores y diseñadores de temas en Palantir.net.

Sus reflexiones ocasionales, vídeos y podcasts se pueden encontrar en http://john.albin.net.



Me gustaría dar las gracias a toda la comunidad de Drupal por su maravilloso apoyo, la amistad, el agravamiento, snark, y la inspiración; al igual que una familia. También me gustaría dar las gracias a mi verdadera familia, mi esposa y dos hijos, Jenny, Owen y Ella, por hacer que me dan ganas de ser una mejor persona. Los amo a todos.



Acerca de los revisores

**Davy Van Den Bremt** ha sido el desarrollo de sitios web con Drupal por cerca de cuatro años.Él vive en Gante, Bélgica, y trabaja como desarrollador de Drupal Senior en Krimson.

Estudió Ciencias de la Computación en la Universidad de Gante, sino enrollado en un desarrollador web como diseñador y el cliente. Se convirtió en un tiempo completo desarrollador Drupal, mientras trabajaba en la VRT, la compañía de radiodifusión pública Flemisch y se ha desarrollado desde los sitios web de las principales empresas de medios de comunicación belgas, agencias de publicidad, y las instituciones gubernamentales.

Él mantiene un blog en drupalcoder.com donde guarda las notas de todas las cosas de Drupal que descubre en su trabajo y quiere compartir con otros usuarios de Drupal.

Ha escrito algunos parches para Drupal 7 y mantiene unos módulos como el tema de Administración y Dirección de correo Marco Marketing.

**David Myburgh** ha estado involucrado con las computadoras, incluso antes de que existiera la web.Estudió para convertirse en un biólogo molecular, pero descubrió que le gustaba trabajar con los ordenadores más que las bacterias. Él tenía su propio negocio de computadoras en Sudáfrica, (donde creció) haciendo de soporte técnico y de ventas. Él incluso creó

algunas páginas web estáticas para los clientes durante ese tiempo.

Después de mudarse a Canadá, que fue aspirado en el mundo de Drupal hace unos años, cuando un amigo quería un sitio para una sociedad histórica local. Desde entonces, una vez más ha comenzado su propia compañía y ahora se basa exclusivamente en los sitios web de Drupal (que no "hace estático" ya no). No hay falta de trabajo en el mundo Drupal y ahora balancea su tiempo entre el trabajo y la familia. Se ha revisado varios Drupal

libros incluyendo Drupal 5 Temas y Drupal 6 Temas.



Me gustaría dar las gracias a mi familia por todo su apoyo a mí y lo que hago. Trabajar desde casa puede ser un arma de doble filo a veces, pero tener la oportunidad de ver crecer a mi hijo hace que todo valga la pena.



**Jojodae Ganesh Sivaji** ha estado involucrado con la comunidad Drupal para másde dos años. Sivaji es un miembro activo; que ha contribuido a la comunidad en términos de parches de escritura para núcleo y contrib módulos. Estuvo involucrado en el Google Summer of Code 2009. Allí trabajó para la organización de Drupal en concurso módulo de funciones de mejora y proyecto de corrección de errores con Matt Butcher y otros desarrolladores de Drupal. El proyecto se completó con éxito, bajo la guía de mentores, Matt Butcher y Shyamala.

Él ha desarrollado y mantiene un par de módulos y temas aportados endrupal.org. Página de la cuenta de usuario de Drupal Sivaji se puede encontrar en

http://drupal.org/user/328724.

En la actualidad es el desarrollador web y programador principal de SG E-ndicus InfoTech Pvt. Ltd, Chennai. En E-ndicus, él es responsable de análisis de requisitos, y llegar a ofrecer soluciones, construcción y mantenimiento de sitios web, principalmente en Drupal y Joomla.



Me gustaría expresar mi más sincero agradecimiento a mi mentor, Matt Butcher, por darme el tiempo y estímulo continuo para perseguir Drupal, incluyendo la revisión de este libro.

Además, me gustaría dar las gracias al Sr. Vikram Vijayaragavan,

Mrs. Shyamala, el Sr. Sri Ramadoss, ILUGC, y toda la comunidad de Drupal (especialmente la comunidad Drupal Chennai) por su apoyo con mi continuo evangelismo Drupal.



www.PacktPub.com

**Los archivos de soporte, libros electrónicos, ofertas de descuentos y más**

Es posible que desee visitar www.PacktPub.com para los archivos de soporte y descargas relacionadas con su libro.

¿Sabía usted que Packt ofrece versiones de libros electrónicos de cada libro publicado, con archivos PDF y ePub disponibles? Puede actualizar a la versión eBook en

www.PacktPub.com y como un cliente de impresión de libros, usted tiene derecho a un descuento en la copia de libros electrónicos. Póngase en contacto con nosotros en service@packtpub.com para más detalles.

En www.PacktPub.com, También se puede leer una colección de artículos técnicos gratuitos, registrarse para obtener una serie de boletines gratuitos y recibe descuentos exclusivos y ofertas en libros y libros electrónicos Packt.



http://PacktLib.PacktPub.com

¿Necesita soluciones inmediatas a sus preguntas de TI? PacktLib es la biblioteca de libros digitales en línea de Packt. Aquí, usted puede acceder, leer y buscar en toda la biblioteca de Packt de libros.

**¿Por qué suscribirse?**

* Todo tipo de búsquedas a través de cada libro publicado por Packt
* Copiar y pegar, imprimir y contenido favorito
* En la demanda y accesible a través del navegador web

**Acceso gratuito para los titulares de cuentas Packt**

Si usted tiene una cuenta con Packt en www.PacktPub.com, Puede utilizar esta opción para acceder PacktLib hoy y ver nueve libros totalmente gratis. Sólo tiene que utilizar sus credenciales de acceso para el acceso inmediato.

Tabla de contenidos

|  |  |
| --- | --- |
| **Prefacio** | **1** |
| **Capítulo 1: Desarrollo de Drupal 7** | **7** |
| **La introducción de Drupal (para desarrolladores)** | **7** |
| Tecnologías que impulsan Drupal | 8 |
| PHP | 9 |
| Bases de datos y MySQL | 9 |
| HTML, CSS y JavaScript | 10 |
| Otras tecnologías | 10 |
| El servidor web | 10 |
| El Sistema Operativo | 11 |
| **Arquitectura Drupal** | **11** |
| Bibliotecas del núcleo de Drupal | 13 |
| Hooks Drupal (hooks) | 13 |
| Módulos del núcleo de Drupal | 14 |
| La base de datos | 15 |
| El tema del sistema | 16 |
| **Subsistemas principales de Drupal** | **16** |
| Temas | 16 |
| Menús | 17 |
| Nodos | 17 |
| Archivos | 18 |
| Usuarios | 18 |
| Comentarios | 18 |
| Campos y entidades | 19 |
| Formularios API | 19 |
| Perfiles de montaje | 19 |
| Prueba simple | 20 |
| Bloques | 20 |
| Otros subsistemas | 20 |

*Tabla de contenidos*



|  |  |  |  |
| --- | --- | --- | --- |
| **Herramientas para el desarrollo de código de Drupal** | **20** | | |
| El control de versiones con Git y CVS | 21 | | |
| Libro de Codigos y Git | 21 | | |
| El sitio del API y estándares de codificación | 22 | | |
| Módulos orientados al desarrollador | 22 | | |
| El módulo del desarrollador | 22 | | |
| Drush (el shell Drupal) | 23 | | |
| Coder | 23 | | |
| **Resumen** | **23** | | |
| **Capítulo 2: Crear el primer módulo** | | | **25** | |
| **Nuestro objetivo: un módulo con un bloque** | | | **25** | |
| **Creación de un nuevo módulo** | | | **26** | |
| Los nombres del módulo | | | 27 | |
| ¿De dónde viene nuestro módulo? | | | 27 | |
| Crear el directorio de módulos | | | 29 | |
| Escribiendo el archivo de información. | | | 29 | |
| Creación de un archivo de módulo | | | 33 | |
| Las normas del código fuente | | | 35 | |
| Módulos Doc estilo Doxygen (Doxygen-style doc blocks) | | | 36 | |
| El hook de ayuda | | | 38 | |
| La función t() y las traducciones | | | 39 | |
| **Trabajar con la API Bloquear** | | | **42** | |
| El hook de bloque de información | | | 43 | |
| El hook de vista de bloque | | | 45 | |
| El primer módulo en acción | | | 48 | |
| **Escribir pruebas automatizadas** | | | **49** | |
| Creación de una prueba | | | 50 | |
| Comenzando | | | 50 | |
| Escribir un caso de prueba | | | 51 | |
| El patrón básico | | | 51 | |
| El método getInfo () | | | 52 | |
| Configuración del caso de prueba | | | 54 | |
| Escribir un método de prueba | | | 55 | |
| **Resumen** | | | **60** | |
| **Capítulo 3: Tema Capa de Drupal (Drupal's Theme Layer)** | | | | **61** |
| **La lógica de negocio frente a la lógica de presentación** | | | | **62** |
| **Granularidad de datos** | | | | **64** |
| **Motores temáticas** | | | | **66** |
| **Dos maneras de tema** | | | | **66** |
| Funciones temáticas | | | | 66 |
| Funciones de preproceso | | | | 68 |
| Anulaciones temáticas (Theme overrides) | | | | 69 |

**[ii ]**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Tabla de contenidos* | |
|  | Los archivos de plantilla | 70 |  |
|  | El zoológico preproceso | 72 |  |
|  | **Render elementos** | **77** |  |
|  | Render propiedades | 79 |  |
|  | hook\_element\_info | 80 |  |
|  | hook\_page\_alter () | 81 |  |
|  | **El poder de tema ()** | **82** |  |
|  | Sugerencias hook Tema | 83 |  |
|  | **Registro de Tema** | **85** |  |
|  | Valores predeterminados de variables | 85 |  |
|  | hook\_theme | 86 |  |
|  | hook\_theme\_registry\_alter | 88 |  |
|  | **¿Qué más?** | **89** |  |
|  | **Resumen** | **90** |  |
| **Capítulo 4: Tematización un Módulo** | | **91** |  |
|  | **La reutilización de una implementación tema por defecto** | **91** |  |
|  | Drupal bloques revisitados | 93 |  |
|  | Tematización un bloque Drupal | 98 |  |
|  | Renderizacion de elemento y una sugerencia hook tema | 99 |  |
|  | Creación de una función pre\_render | 100 |  |
|  | Colocación de CSS para hacer arreglos | 102 |  |
|  | Idiomas RTL | 103 |  |
|  | **Pasos para crear una aplicación tema por defecto** | **106** |  |
|  | hook\_theme () implementaciones | 107 |  |
|  | Variables frente a hacer elemento (Variables versus render element) | 108 |  |
|  | Funciones de preproceso | 109 |  |
|  | Los archivos de plantilla | 114 |  |
|  | **Resumen** | **118** |  |
| **Capítulo 5: Creación de una interfaz de administración** | | **119** |  |
|  | **El Usuario Warn modulo (The User Warn module)** | **119** |  |
|  | **Comenzando nuestro módulo** | **120** |  |
|  | **El sistema de menús Drupal** | **121** |  |
|  | Definición de una devolución de llamada de la página con hook\_menu | 121 |  |
|  | El uso de comodines (wildcards) en rutas de menús | 125 |  |
|  | **Form API** | **126** |  |
|  | Uso de drupal\_get\_form () | 127 |  |
|  | La construcción de una función de devolución de llamada de forma (form callback function) | 128 |  |
|  | La gestión de datos persistente | 133 |  |
|  | Proceso de envío de formulario | 136 |  |

**[iii ]**



*Tabla de contenidos*



|  |  |  |  |
| --- | --- | --- | --- |
| Un acceso directo para la configuración del sistema | | | 138 |
| Un acceso directo a los formularios de confirmación | | | 139 |
| **Envío de correo con drupal\_mail () y hook\_mail ()** | | | **141** |
| Drupal\_mail Calling () | | | 142 |
| Hook\_mail Implementación () | | | 144 |
| **El sistema de fichas (The token system)** | | | **146** |
| ¿Cuáles son los tokens? | | | 146 |
| La implementación de fichas en su texto | | | 147 |
| **Resumen** | | | **149** |
| **Capítulo 6: Trabajar con contenido** | | **151** | | |
| **¿Por qué crear sus propias entidades** | | **151** | | |
| **El objetivo** | | **152** | | |
| **Bundles** | | **152** | | |
| **La API de esquema (The Schema API)** | | **152** | | |
| **La declaración de nuestra entidad** | | **156** | | |
| La declaración de la entidad | | 156 | | |
| El controlador entidad | | 161 | | |
| **Dirección de la Entidad (Entity management)** | | **163** | | |
| Administración de tipos de obras de arte | | 163 | | |
| Adición de obras de arte | | 165 | | |
| Adición de nuevas obras de arte | | 167 | | |
| Devolución de llamada de validación | | 170 | | |
| Enviar callback | | 171 | | |
| Ahorro de su obra | | 172 | | |
| Manejo de las revisiones | | 175 | | |
| Visualización de obras de arte | | 176 | | |
| Edición de una obra de arte | | 177 | | |
| Eliminación de una obra de arte | | 178 | | |
| **Resumen** | | **182** | | |
| **Capítulo 7: Creación de nuevos campos** | | | **183** | |
| **Nuestro objetivo: un campo de "dimensiones"** | | | **183** | |
| **¿Cómo funciona la API de campo** | | | **184** | |
| **La creación de nuestro nuevo tipo de campo** | | | **185** | |
| Declarar el campo | | | 185 | |
| Definición de la estructura sobre el terreno | | | 186 | |
| Definición de vacío | | | 188 | |
| Ajustes de campo | | | 188 | |
| Validación de campo | | | 189 | |
| **La exposición de campos a la API formulario con widgets** | | | **191** | |
| Declarando un widget | | | 191 | |

**[iv ]**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Tabla de contenidos* | |
|  | Formularios simple widget (Simple widget forms) | 192 |  |
|  | Widgets complejos | 194 |  |
|  | **Usando formateadores para mostrar nuestro campo** | **199** |  |
|  | Declarar un formateador | 200 |  |
|  | Formateadores un solo valor (Single-value formatters) | 200 |  |
|  | Formateadores complejos | 201 |  |
|  | **La gestión de los campos no Field** | **205** |  |
|  | **Encontrar a entidades y campos** | **206** |  |
|  | **Resumen** | **210** |  |
| **Capítulo 8: Drupal permisos y seguridad** | | **211** |  |
|  | **Usando user\_access () para hacer valer los permisos** | **212** |  |
|  | **Comprobación de la cuenta de usuario correcta** | **213** |  |
|  | **Uso de hook\_permission ()** | **217** |  |
|  | Definición de permisos de su módulo | 218 |  |
|  | Hook\_permission guión () (Writing hook\_permission()) | 219 |  |
|  | **Declarar sus propias funciones de acceso** | **221** |  |
|  | **Responder cuando se niega el acceso** | **224** |  |
|  | **Activación de permisos mediante programación** | **227** |  |
|  | **Definir las funciones mediante programación** | **228** |  |
|  | **Asegurar forms en Drupal** | **229** |  |
|  | La API Form | 229 |  |
|  | Deshabilitar elementos de formulario | 230 |  |
|  | Pasar datos seguras a través de formularios | 231 |  |
|  | Ejecución de comprobaciones de acceso en los formularios | 233 |  |
|  | **Manejo de las devoluciones de llamada de AJAX de forma segura** | **235** |  |
|  | El uso de AJAX en formularios | 235 |  |
|  | El uso de AJAX en otros contextos | 236 |  |
|  | **Resumen** | **240** |  |
| **Capítulo 9: Nodo de Acceso** | | **241** |  |
|  | **Nodo de acceso en comparación con user\_access () y otra** |  |  |
|  | **comprobaciones de permisos** | **242** |  |
|  | Cómo Drupal concede permisos de nodos. | 242 |  |
|  | La función () node\_access | 244 |  |
|  | La lista blanca en el acceso | 246 |  |
|  | El almacenamiento en caché del resultado para el rendimiento | 246 |  |
|  | Invocando Hook\_node\_access () | 247 |  |
|  | El acceso a los propios nodos de un usuario | 248 |  |
|  | Invocando el acceso a la API de nodo | 248 |  |
|  | hook\_node\_access () en comparación con {node\_access} | 250 |  |

**[]**



*Tabla de contenidos*



|  |  |  |
| --- | --- | --- |
| **Uso de hook\_node\_access ()** | | **254** |
| Ejemplo del módulo de control de acceso | | 254 |
| Un segundo módulo de control de acceso | | 256 |
| Ver operaciones y los módulos de control de acceso | | 259 |
| **Al escribir un módulo de acceso del nodo** | | **260** |
| El {} node\_access mesa y su papel (The {node\_access} table and its role) | | 261 |
| Esquema {} node\_access tabla explica ({node\_access} table schema explained) | | 263 |
| Definir reglas de acceso de su módulo | | 264 |
| **Crear el rol de acceso al modulo** | | **266** |
| Uso de hook\_node\_access\_records () | | 266 |
| Uso de hook\_node\_grants () | | 269 |
| Las consideraciones de seguridad | | 271 |
| La reconstrucción de la tabla {} node\_access (Rebuilding the {node\_access} table) | | 273 |
| **Modificar el comportamiento de otros módulos** | | **274** |
| Uso de hook\_node\_grants\_alter () | | 275 |
| Uso de hook\_node\_access\_records\_alter () | | 279 |
| **Probar y depurar su módulo** | | **282** |
| Usando Desarrollo Nodo de Acceso (Using Devel Node Access) | | 282 |
| Uso de hook\_node\_access\_explain () | | 283 |
| (Using the Devel Node Access by user block) | | 284 |
| **Resumen** | | **285** |
| **Capítulo 10: JavaScript en Drupal** | | **287** | |
| **JavaScript dentro de Drupal** | | **287** | |
| **Adición de JavaScript** | | **288** | |
| Adición de archivos JavaScript y CSS a. Archivos info | | 289 | |
| Uso de drupal\_add\_js () | | 289 | |
| Adición de archivos JavaScript | | 289 | |
| Adición de archivos CSS | | 292 | |
| pasar las variables de PHP a JavaScript | | 293 | |
| Adición inline JavaScript | | 294 | |
| Adición inline de CSS | | 294 | |
| Uso de la API Library | | 295 | |
| Definición de una biblioteca con hook\_library | | 296 | |
| La alteración de la información en hook\_library | | 297 | |
| Utilización de matrices (arrays) renderizables | | 298 | |
| **Alteración de JavaScript** | | **299** | |
| Alteración de CSS | | 300 | |
| **Drupal specificos JavaScript** | | **301** | |
| Presentación temable | | 301 | |
| Cadenas traducibles | | 302 | |
| Comportamientos | | 303 | |

**[vi ]**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Tabla de contenidos* | |
|  | **Helpers de AJAX** | **305** |  |
|  | Agregar AJAX a los forms | 305 |  |
|  | AJAX automáticamente aplicado | 307 |  |
|  | Comandos AJAX | 309 |  |
|  | ajax\_command\_after | 309 |  |
|  | ajax\_command\_alert | 309 |  |
|  | ajax\_command\_append | 309 |  |
|  | ajax\_command\_before | 310 |  |
|  | ajax\_command\_changed | 310 |  |
|  | ajax\_command\_css | 310 |  |
|  | ajax\_command\_data | 310 |  |
|  | ajax\_command\_html | 310 |  |
|  | ajax\_command\_prepend | 311 |  |
|  | ajax\_command\_remove | 311 |  |
|  | ajax\_command\_replace | 311 |  |
|  | ajax\_command\_restripe | 311 |  |
|  | ajax\_command\_settings | 312 |  |
|  | **Resumen** | **312** |  |
| **Capítulo 11: Trabajar con archivos y Imágenes** | | **313** |  |
|  | **Los módulos de Twitpic y marcas de agua** | **313** |  |
|  | **Los archivos en Drupal** | **314** |  |
|  | **La API FILE** | **316** |  |
|  | **Envolturas Stream (Stream wrappers)** | **319** |  |
|  | Creación de una envoltura de secuencia (Creating a stream wrapper) | 320 |  |
|  | **Imágenes en Drupal** | **326** |  |
|  | API de imagen | 326 |  |
|  | Estilos de imagen | 331 |  |
|  | Creación de efectos de imagen | 334 |  |
|  | Creación de estilos de imagen para un módulo | 339 |  |
|  | **Resumen** | **342** |  |
| **Capítulo 12: Perfiles de instalación** | | **343** |  |
|  | **La introducción de perfiles de instalación** | **343** |  |
|  | **Distribuciones de Drupal** | **344** |  |
|  | La creación de una distribución | 344 |  |
|  | Perfiles estándar y mínimos | 344 |  |
|  | Creación de un directorio de perfil | 344 |  |
|  | Módulos de perfil y temas | 345 |  |
|  | **Creación de perfiles** | **345** |  |
|  | Activación de los módulos | 347 |  |
|  | La instalación del sistema de tareas | 348 |  |
|  | Elegir una tarea de instalación o usar hook\_install | 348 |  |
|  | Anatomía de una tarea de instalación | 348 |  |

**[vii ]**



*Tabla de contenidos*



|  |  |  |  |
| --- | --- | --- | --- |
| Creación de una tarea | | | 349 |
| La alteración de las tareas | | | 354 |
| Configuración de los bloques | | | 355 |
| Configuraciones variables | | | 357 |
| Filtros de texto | | | 358 |
| Colocación Código | | | 359 |
| **Ejecutar el instalador desde la línea de comandos** | | | **360** |
| **Resumen** | | | **362** |
| **Apéndice A: Database Access** | | | **363** | |
| **Consultas básicas** | | | **364** | |
| **Objetos de Resultados (Result objects)** | | | **365** | |
| **Consultas dinámicas** | | | **366** | |
| **Cosultas de Inserccion** | | | **368** | |
| **Consultas de actualización** | | | **370** | |
| **Consultas de Eliminacion** | | | **370** | |
| **Combinar consultas (Merge queries)** | | | **370** | |
| **Asignaturas avanzadas** | | | **372** | |
| Transacciones | | | 372 | |
| Slave Servers | | | 373 | |
| **Resumen** | | | **374** | |
| **Apéndice B: Seguridad** | | **375** | | |
| **Pensar de forma segura** | | **375** | | |
| **Filtrado contra escapar** | | **376** | | |
| Filtración | | 377 | | |
| Escapando HTML | | 377 | | |
| Inyección SQL | | 378 | | |
| **Control de acceso Nodo** | | **378** | | |
| **Manejo de código inseguro** | | **379** | | |
| **Mantenerse al día** | | **380** | | |
| **Resumen** | | **381** | | |
| **Índice** | | **383** | | |

**[viii ]**



Prefacio

Drupal es un sistema de gestión de contenido de código abierto galardonado. Es un sistema modular, con una elegante arquitectura basado en hooks, y un gran código. Los módulos son plugins para Drupal que amplían, construyen o mejoran la funcionalidad del núcleo de Drupal.

En Drupal 7 Desarrollo de modulos, seis desarrolladores profesionales de Drupal utilizan un enfoque práctico, basado en ejemplos para presentar a los desarrolladores de PHP, las nuevas y potentes herramientas de Drupal 7, las API y las estrategias para la escritura de código personalizado de Drupal.

Estas herramientas no sólo hacen que la gestión y el mantenimiento de sitios web mucho más fácil, pero también son muy divertidos para jugar y sorprendentemente fácil de usar.

**Lo que cubre este libro**

*Capítulo 1*, Introducción al Desarrollo de Módulos Drupal ofrece una introducción al ámbito de Drupal como gestor de contenidos basado en la web. Se detiene en aspectos básicos como las tecnologías que impulsan Drupal y el diseño arquitectónico de Drupal. Una breve idea de los componentes (subsistemas) de Drupal y las herramientas que se pueden utilizar para desarrollarlo, completa el cuadro básico de Drupal.

*Capítulo 2*, Un primer módulo, se pone las cosas en acción, mediante la descripción de cómo empezar a construir nuestro primer módulo en Drupal. Una vez hecho esto, nos va a decir cómo se puede utilizar la Block API para crear nuestro código personalizado para Drupal. Por último, hay una palabra o dos sobre cómo probar nuestro código escribiendo las pruebas automatizadas.

*Capítulo 3*, Drupal Themes, es todo acerca de la Capa de Theme en Drupal. Se inicia con maneras para temas, y ​​luego procede a aspectos relacionados con la tematización. Habla de "Render Elements and concludes by getting us familiar with 'Theme Registry'.

*Capítulo 4*, Tematización módulo que utiliza los conceptos que vimos en el capítulo anterior sobre los módulos temáticos en Drupal. Se nos familiariza con el concepto de la reutilización de una implementación tema por defecto, y nos enseña a construir una aplicación para el tema de las situaciones de la vida real.

*Prefacio*



*Capítulo 5*, Construcción de una interfaz de administración nos mostrará cómo ir sobre la construcción de un módulo con una interfaz de administración. Mientras se hace esto, los conceptos básicos de los módulos presentados en el Capítulo 2 serán útiles. A 'User Warn' module is developed as an illustration, in the chapter.

*Capítulo 6*, Trabajar con contenido hace hincapié en la gestión de contenidos. Creación de entidades, la clase del controlador, la integración de nuestra entidad con el API Field, y la visualización de formularios de confirmación son algunas de las cosas que nos encontramos en este capítulo.

*Capítulo 7*, Crear nuevos campos, se echa un vistazo en la creación de nuevos campos. Además, nos enseña cómo utilizar Widgets correspondientes para permitir a los usuarios editar los campos. Finalmente, para asegurar que los datos se muestren como se desea, el papel de Formateadores se discute en el capítulo.

*Capítulo 8*, Módulo de permisos y seguridad tiene que ver con el control de acceso y seguridad. Habla de permisos que ayudan a los usuarios para tener acceso (o ser negado) a características específicas. Además, el capítulo habla de cómo administrar funciones mediante programación.

Una de las áreas más importantes de la seguridad del sitio web, el manejo de formularios, se detalla aquí.

*Capítulo 9*, Nodo de acceso se refiere al acceso de nodo, que es una de las herramientas más poderosas de la API de Drupal. Arroja luz sobre cómo se determina el acceso a un nodo y en las principales operaciones controladas por la Node Access API, entre otras cosas.

*Capítulo 10*, JavaScript en Drupal proporciona los conocimientos fundamentales necesarios para trabajar con JavaScript dentro de Drupal. Esto ayuda a crear características de gran alcance, tales como la superposición, de autocompletado, arrastrar y soltar, y así sucesivamente.

*Capítulo 11*, Trabajar con archivos y las imágenes, habla de cómo la gestión y el mantenimiento se pueden hacer mucho más fácil mediante el uso de las API de archivos e imágenes en Drupal 7. Asimismo, el capítulo nos habla de las diversas técnicas de procesamiento de imágenes que implica el trabajo con imágenes, haciendo las cosas más coloridas y divertidas.

*Capítulo 12*, Perfiles de instalación se describe el proceso para trabajar con "Distribuciones" y "Perfiles de Instalación 'en Drupal. Ellos ayudan a hacer el trabajo del desarrollador más fácil.

*Apéndice A*, *Database Access*, ofrece ideas útiles sobre la capacidad de los desarrolladores para aprovechar la Capa de Base de datos de Drupal 7, con el fin de realizar consultas entre bases de datos de gran alcance.

*Apéndice B*, Seguridad, hace hincapié en la necesidad de desarrollar una práctica que asuma en mente el aspecto de la seguridad al escribir el código. Se trata de dos maneras de tratar con datos potencialmente inseguros, es decir, 'filtrado' y 'escaping'.

**[]**



*Prefacio*



**Para quién es este libro**

Si usted es un desarrollador de PHP o un usuario de Drupal que busca sumergirse en el desarrollo de Drupal, a continuación usted encontrará en este libro una excelente introducción a la codificación dentro de Drupal. Los que tienen un poco de experiencia en Drupal también encontrarán esto como una herramienta de valor incalculable para la actualización de sus conocimientos sobre las nuevas características de Drupal 7. Desarrolladores de temas que buscan ampliar sus capacidades, encuentran una introducción accesible a la codificación de PHP en el entorno Drupal.

Este libro supone que usted está familiarizado con la programación básica de PHP, junto con HTML y CSS. No se requiere experiencia en programación de Drupal, aunque también es una forma útil para los desarrolladores experimentados de Drupal para ponerse al día con Drupal 7.

**Convenciones**

En este libro, usted encontrará una serie de estilos de texto que distinguen entre diferentes tipos de información. Estos son algunos ejemplos de estos estilos, y una explicación de su significado.

Las palabras de código en texto se muestran de la siguiente manera:

"El tercer argumento especifica qué file\_save\_data () debe hacer cuando ya existe un archivo con el mismo nombre del archivo que estamos tratando de salvar ".

Un bloque de código se establece de la siguiente manera:

$contents = ";

$handle = fopen("/var/www/htdocs/images/xyzzy.jpg", "rb");

while (!feof($handle)) {

$contents .= fread($handle, 8192);

}

fclose($handle);

Cuando queremos llamar su atención sobre una parte específica de un bloque de código, las líneas o los artículos pertinentes se establecen en negrita:

**$items['user/%/warn'] = array(**

'title' => 'Warn',

'description' => 'Send e-mail to a user about improper site behavior.',

'page callback' => 'drupal\_get\_form',

**Nuevos términos** y palabras importantes aparecen en negrita.

**[]**



*Prefacio*



El sistema para el manejo de esto es colectivamente el colectivamente llamado **theme system**.

Palabras que se ven en la pantalla, en los menús, cuadros de diálogo, por ejemplo, aparecen en el texto de esta manera: "En la imagen anterior, se puede ver el paquete de agrupación Core en la esquina superior izquierda."



Advertencias o notas importantes aparecen en un cuadro como este.



Consejos y trucos aparecen así.

**Comentarios de los lectores**

Los comentarios de nuestros lectores, siempre son bienvenido. Háganos saber lo que piensas acerca de este libro-lo que le gustó o no le gusto. Los Comentarios de los lectores son importantes para nosotros desarrollar títulos que realmente obtengan el máximo provecho.

Para enviarnos comentarios generales, basta con enviar un e-mail afeedback@packtpub.com, Y mencione el título del libro a través del asunto de su mensaje.

Si hay un libro que usted necesita y desea verlo publicado, por favor envíenos una nota en la forma SUGIERE UN TÍTULO sobre www.packtpub.com o por e-mail

suggest@packtpub.com.

Si hay un tema que usted tiene experiencia en y usted está interesado en cualquiera de escritura o contribución en un libro, vea nuestra guía de autor www.packtpub.com / autores.

**Atención al cliente**

Ahora que usted es el orgulloso propietario de un libro de Packt, tenemos una serie de cosas que le ayudarán a obtener lo máximo de su compra.



**Descargar el código de ejemplo de este libro**

Puede descargar los archivos de código de ejemplo para todos los libros Packt que ha adquirido desde su cuenta en http://www.PacktPub.com. Si ha adquirido este libro en otros lugares, se puede visitar http://www. PacktPub.com / support y registrarse para tener los archivos enviados por correo-e directamente a usted.



**[]**



*Prefacio*



**Errata**

Aunque hemos tomado todas las precauciones para asegurar la exactitud de nuestro contenido, los errores ocurren. Si usted encuentra un error en uno de nuestros libros-tal vez un error en el texto o el código que agradecería que le informe a nosotros. De esta manera, usted puede ahorrar otros lectores de la frustración y ayudar a mejorar las versiones posteriores de este libro. Si encuentra alguna errata, por favor informe al visitarhttp://www.packtpub. com / support, La selección de su libro, hacer clic en el **presentación de erratas** **forma** vincular y entrar en los detalles de su fe de erratas. Una vez que su fe de erratas se verifica, su presentación será aceptada y la fe de erratas se cargará en nuestro sitio web, o añadir a cualquier lista de erratas existentes, en la sección de erratas de ese título. Cualquier errata existente puede ser visto por la selección de su título dehttp://www.packtpub.com/support.

**Piratería**

La piratería de material con copyright en Internet es un problema constante en todos los medios. En Packt, nos tomamos muy en serio la protección de nuestros derechos de autor y licencias. Si te encuentras con las copias ilegales de nuestras obras, en cualquier forma, en Internet, por favor indíquenos la dirección de ubicación o el nombre del sitio web inmediatamente para que podamos presentar un recurso.

Por favor, contacte con nosotros en copyright@packtpub.com con un enlace a la sospecha de material pirateado.

Apreciamos su ayuda en la protección de nuestros autores, y nuestra capacidad para lograr que el contenido valioso.

**Preguntas**

Puede ponerse en contacto con nosotros en questions@packtpub.com si usted está teniendo problemas con cualquier aspecto de la obra, y nosotros haremos nuestro mejor esfuerzo para resolver el problema.

**[]**



Desarrollando para Drupal 7

Drupal es un sistema basado en la web de gestión de contenidos (CMS) y la plataforma de medios sociales. Si bien es útil fuera de la caja, se ha diseñado pensando en los desarrolladores. El propósito de este libro es explicar cómo Drupal puede extenderse de muchas maneras y para muchos fines. Este capítulo presenta la terminología, las herramientas y procesos para el desarrollo de Drupal 7. Si bien los capítulos siguientes se centran en el código, en este capítulo se centra en los conceptos. Vamos a hablar de la arquitectura de Drupal, y cómo

se puede enganchar en Drupal en lugares estratégicos para extenderla para realizar nuevas tareas. Las siguientes son las principales cosas que estaremos cubriendo en este capítulo:

* Una introducción al desarrollo de Drupal
* La arquitectura de Drupal
* Los nodos, campos, usuarios y otros subsistemas principales
* Herramientas para el desarrollo en Drupal

Al final de este capítulo, se entiende los aspectos arquitectónicos de Drupal y estar listo para comenzar a escribir código.

**La introducción de Drupal (para desarrolladores)**

Fuera de la caja, Drupal 7 realiza todas las funciones estándar de un sistema de gestión de contenido basado en web. Los visitantes pueden ver la información publicada en el sitio, navegar por los menús y ver páginas individuales, completos con imágenes. Los usuarios pueden crear cuentas y dejar comentarios. Los administradores pueden gestionar la configuración del sitio y controlar los niveles de permisos de los usuarios. Los editores pueden crear contenidos, realizar una vista preliminar y luego publicarlo, una vez que esté listo. El contenido puede ser sindicado a RSS, donde los lectores de feeds pueden recoger nuevos artículos que se vayan publicando. Con varios temas incorporados, incluso la apariencia de la página se pueden cambiar fácilmente.

*Desarrollando para Drupal 7*



Tan fantástico ya que estas funciones son, seguro que no satisface las necesidades de todos los usuarios. Para ello, las capacidades de Drupal se pueden ampliar fácilmente con módulos, temas y perfiles de instalación. Echa un vistazo a la web principal de Drupal, http://drupal.org, Y usted encontrará miles de módulos que proporcionan nuevas características, y miles de temas que transforman la apariencia del sitio.

El hecho de que casi todos los aspectos del comportamiento de Drupal pueden ser interceptados y se transforman a través de los mecanismos de módulos y diseños ha llevado a muchos a afirmar que Drupal no es sólo un gestor de contenidos (CMS), sino un marco de gestión de contenidos (CMF) capaz de ser re-labrado a las necesidades específicas

y los requisitos funcionales.

Sea o no Drupal es llamado con razón un CMS o CMF está más allá de nuestros intereses presentes, pero lo cierto es que la mayoría enorme activo de Drupal es su extensibilidad. Se desea utilizar un servidor de directorio para la autenticación? Hay un módulo de Drupal para eso. ¿Quiere exportar datos a archivos CSV (Comma Versión Separado)? Hay varios módulos para que (dependiendo de los datos que desea exportar). Interesado en apoyo de Facebook, la integración con Twitter, o añadir un Compartir este botón? Sí, hay módulos para todos ellos también-todos los cuales están disponibles enhttp://drupal.org/.

Quiere integrar Drupal con esa herramienta personalizada que escribiste para resolver sus necesidades específicas de negocio? Puede que no haya un módulo para eso, pero con un poco de código, se puede escribir el suyo propio. De hecho, ese es el tema de este libro.

El propósito de este libro es conseguir que en rampa hacia arriba (lo antes posible) para el desarrollo de Drupal. A medida que avanzamos por capítulos a través de este libro, cubrimos las API y las herramientas que va a utilizar para crear sitios personalizados de Drupal, y no se adhieren a la teoría. Cada capítulo ofrece trabajo, el código de ejemplo de orientación práctica diseñada para mostrar cómo crear código. Seguimos convenciones de código de Drupal y utilizamos los patrones de diseño de Drupal en un esfuerzo por ilustrar la forma correcta de escribir código. Aunque ciertamente no podemos escribir el código exacto para satisfacer sus necesidades, nuestra esperanza es que el código indicado en este capítulo puede servir como una base para sus aplicaciones más grandes y mejores.

Así que vamos a empezar con algunas cuestiones preliminares.

**Tecnologías que impulsan Drupal**

Muchos libros de esta calaña se iniciará con un capítulo sobre la instalación de la plataforma. Hemos decidido no seguir este patrón por varias razones. En primer lugar, Drupal es increíblemente bien documentado en este sentido. La README archivo incluido con la descarga de Drupal debe satisfacer sus necesidades. En segundo lugar, nuestra experiencia ha sido que estos capítulos son innecesarios. Los desarrolladores de software necesitan raramente paso a paso las guías de instalación de un sistema tan simple como Drupal.

**[]**



*Capítulo 1*



Sin embargo, lo que queremos, para empezar, es una visión rápida de las tecnologías utilizadas en Drupal.

**PHP**

Drupal está escrito en el lenguaje de programación PHP (http://php.net). PHP es un multi-plataforma, lenguaje de scripting centrado en la web un amplio apoyo. Desde Drupal está escrito en PHP, este libro se centra en gran medida en el desarrollo de PHP.

Una parte específica de la información debe ser explícita: A partir de Drupal 7, la versión mínima de PHP es PHP 5.2 (a partir de este escrito, la versión actual de PHP es 5.3.3). Las versiones anteriores de Drupal incluyen soporte PHP 4.x, pero esto ya no es el caso.

Otra cosa a destacar es el estilo de codificación que utiliza PHP Drupal. Mientras que muchas de las aplicaciones PHP ahora se escriben utilizando programación orientada a objetos, Drupal no sigue su ejemplo. Por muchas razones, algunos históricos, algunos prácticos, Drupal está escrito en gran medida el uso de la programación procedimental. En lugar de depender en gran medida de las clases e interfaces, módulos de Drupal se componen de colecciones de funciones.

Antes de que nadie salta a conclusiones, sin embargo, nos gustaría hacer algunas salvedades por lo que acabamos de decir:

* Drupal utiliza con frecuencia objetos
* Drupal tiene ciertos subsistemas que son orientados a objetos
* Muchos módulos de Drupal están orientados a objetos sustancialmente
* Drupal hace uso frecuente de los patrones de diseño, ya que es sin duda el caso de que el código de procedimiento se puede utilizar patrones de diseño también

Mientras que la mayor parte de este manual se utilizan las estrategias de codificación de procedimiento, se encontrará con programación orientada a objetos aquí y allá. Si usted no está familiarizado con orientación a objetos convenciones y estilos de codificación, no se preocupe. Vamos a explicar estas piezas a medida que avanzamos.

**Bases de datos y MySQL**

En el pasado, Drupal ha apoyado dos bases de datos: MySQL y PostgreSQL. Drupal 7 se ha movido más allá de esto. Drupal ahora utiliza la poderosa librería PDO (PHP Data Objects) que es estándar en PHP 5. Esta biblioteca es una capa de abstracción que permite a los desarrolladores para apoyar a numerosas bases de datos incluyendo MySQL, PostgreSQL, SQLite,

MariaDB, y muchos, muchos otros. Mientras Drupal hace pruebas en sólo unas pocas bases de datos específicos (a saber, MySQL, PostgreSQL, SQLite y ahora), es posible ir más allá de estos para SQL Server, Oracle, DB2, y otros.

**[]**



*Desarrollando para Drupal 7*



Sin embargo, por el bien de tamaño y legibilidad, hemos centrado nuestros ejemplos en MySQL. Creemos que nuestro SQL debe ejecutarse en MariaDB, PostgreSQL, SQLite y sin modificaciones, pero no hemos hecho ningún intento de probar contra otras bases de datos. Si usted encuentra un error, le agradeceríamos oír hablar de ella. Packt Publishing seguimiento de erratas en su página web (http://packtpub.com), Y se puede presentar errores que se encuentran a través del formulario que encontrará allí.

Drupal ofrece una API de base de datos, junto con algunas convenciones de código SQL (como "no utilice LÍMITE en su SQL "). La intención de éstos es la combinación de código y la convención para que sea lo más fácil posible escribir código portable. Por lo tanto, no sólo

ilustrar la API a través de este libro, pero también nos centramos en escribir sentencias SQL que cumplen con las convenciones estándar de Drupal.

**HTML, CSS y JavaScript**

El formato de datos de facto web es el HTML (HyperText Markup Language) de estilo con CSS (Cascading Style Sheets). Componentes interactivos del lado del cliente son secuencias de comandos con JavaScript. Como desarrolladores de Drupal, nos encontraremos con los tres de estas tecnologías en este libro. Mientras que usted no necesita ser un ninja JavaScript para entender el código aquí, podrá disfrutar al máximo de este libro, si usted se siente cómodo con estas tres tecnologías.

**Otras tecnologías**

El Internet se nutre de cambio, lo que parece, y hay muchas otras tecnologías web que se han vuelto comunes. Aquí y allá, vamos a hablar de tecnologías como RSS (Really Simple Syndication), XML (eXtensible Markup Language), XML-RPC, y otros. Sin embargo, todos estos son de importancia secundaria para nosotros. Mientras Drupal ofrece soporte para muchas de estas cosas, su uso no es esencial para el desarrollo del módulo o tema.

**El servidor web**

Apache ha sido el servidor web predominante, pero no es de ninguna manera el único servidor. Mientras Drupal fue originalmente escrito con Apache en mente, muchos otros servidores web (incluyendo IIS, lighttpd y nginx) pueden ejecutar Drupal.

No cubrimos explícitamente la capa de servidor web en cualquier lugar aquí, principalmente porque el desarrollo rara vez se requiere trabajar en ese nivel bajo. Sin embargo, Drupal espera una buena cantidad de procesamiento de la capa de servidor web, incluyendo el manejo de la reescritura de URL.

**[10 ]**



*Capítulo 1*



**El Sistema Operativo**



Windows, Linux, Mac OS, BSD. Estos son términos que despiertan las guerras santas en línea modernos. Sin embargo, no nos importa para tomar parte en la discusión. Drupal se ejecutará en la mayoría (si no todos) los sistemas operativos de servidor más populares, incluyendo Windows y muchas UNIX / Linux variantes.



En aras de la divulgación completa, los autores de este libro trabajan principalmente en Linux y sistemas Mac OS X. Sin embargo, nuestro código debería funcionar en cualquier sistema de Drupal.

Una vez más, si usted encuentra ejemplos en los que nuestro código no se ejecuta debido a que el sistema operativo, o presentar una errata en la página web de Packt Publishing. Drupal se esfuerza por ser multiplataforma, y ​​nosotros también.

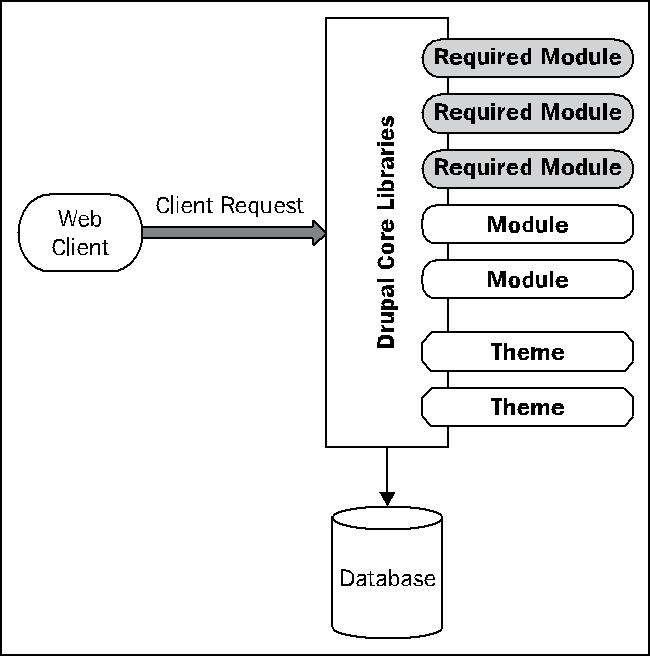
Con estos preliminares detrás de nosotros, vamos a pasar a la arquitectura de Drupal.

**Arquitectura Drupal**

En la sección anterior, hemos introducido las tecnologías que impulsan Drupal.

Sin embargo, ¿cómo encajan entre sí? En esta sección, ofrecemos una visión general de la arquitectura de Drupal.

Vamos a comenzar con una representación visual. El siguiente diagrama esboza los principales componentes de Drupal:



**[11 ]**



*Desarrollando para Drupal 7*



La figura anterior representa, en cierto modo menor consideración, ¿cómo se estructura Drupal. También indica cómo Drupal controla una solicitud. Vamos a hablar de los componentes en el resto de esta sección. Como una primera aproximación, sin embargo, vamos a caminar a través de un ejemplo simplificado de una solicitud típica, ya que se maneja en un sitio web Drupal:

* Un usuario introduce la URL http://example.com/node/123 en un navegador web y accesos Introduzca
* Los contactos del navegador al servidor web en example.com y solicita el recurso a / Node/123
* El servidor web reconoce que la petición debe ser manejada por PHP, y se pone en marcha (o contactos) un entorno de PHP para manejar la petición
* PHP se ejecuta Drupal index.php presentar, entregándoselo el camino / Node/123
* El núcleo de Drupal se somete a un proceso de arranque, inicialización de los recursos, y luego usa el sistema de menús para encontrar la manera de manejar / Node/123
* El sistema de nodos responde a la solicitud de / Node/123 cargando el nodo (pieza de contenido) con el ID 123. Estos datos son típicamente carga desde una base de datos
* El tema del sistema toma la información del nodo y se aplica el formato y el estilo de información a la misma, esencialmente la transformación de los datos en trozos de HTML con CSS asociada
* El núcleo de Drupal completa todo el procesamiento (dando muchos otros subsistemas la oportunidad de manejar varios bits de procesamiento de datos) y a continuación, devuelve los datos al cliente
* El navegador web cliente transforma el código HTML y CSS en una presentación visual para el usuario, que ejecute cualquier código JavaScript necesario en el camino
* El usuario ve el documento

Si bien esta ilustración no ha afectado a todos los detalles técnicos, proporciona una visión de la forma en funciones de Drupal en la web. En la siguiente sección, vamos a pasar algún tiempo viendo las grandes Drupal subsistemas-nodos, temas, menús, etc. Sin embargo, en la presente sección, estamos más preocupados con la forma general en que estos sistemas trabajan juntos.

Anteriormente, vimos que, en condiciones normales, PHP Drupal carga de index.php presentar para controlar una solicitud de Drupal. Este pequeño script se encarga de poner en marcha el entorno de Drupal, y es parte de lo que normalmente llamamos el núcleo de Drupal.

**[12 ]**



*Capítulo 1*



Núcleo de Drupal es la capa fundamental de Drupal que se suministra con un puñado de bibliotecas del núcleo junto con más de una docena de módulos básicos. El archivo de índice carga las bibliotecas y luego inicializa Drupal, un proceso llamado bootstrapping.

**Bibliotecas del núcleo de Drupal**

Las bibliotecas del núcleo proporcionan las funciones y servicios que se utilizan a lo largo de Drupal. Instalaciones para interactuar con la base de datos, la traducción entre lenguas, desinfección datos de usuario, formularios de construcción, y la codificación de datos se encuentran todos en bibliotecas del núcleo de Drupal. Estas herramientas son lo que podríamos llamar los servicios públicos: Facilitan el procesamiento de datos eficaz, pero no son responsables de manejar el ciclo de vida de una solicitud. El ciclo de vida es manejado por módulos.

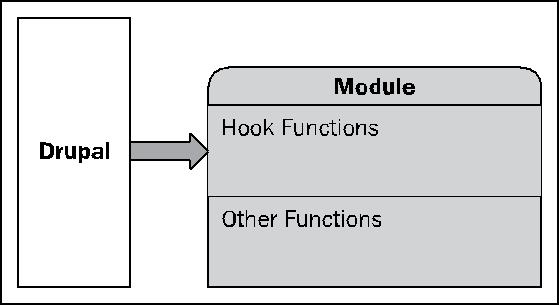
Una vez que Drupal ha cargado bibliotecas del núcleo y se inicializa la base de datos, carga los módulos y temas habilitados, y entonces comienza un proceso sistemático paso a paso para manejar la petición. Este proceso es lo que yo llamo el ciclo de vida de una solicitud. Funciona de la siguiente manera.

Drupal los pasos a través de un número de operaciones predeterminadas. Se comprueba si los módulos deben ser inicializado. Se ve hasta qué código es responsable de manejar la URL dada. Se comprueba si el usuario actual tiene acceso al recurso solicitado. Se comprueba si algunos datos se pueden recuperar de la memoria caché. Sigue paso a paso a través de este tipo de operaciones hasta que la solicitud está completa.

Sin embargo, lo más importante de este proceso paso a paso es la forma Drupal lo hace.

**Hooks Drupal**

El núcleo de Drupal no trata de hacer el procesamiento para cada uno de estos pasos. En lugar de ello, después de cada paso, que ofrece uno o más módulos de la oportunidad de manejar ese paso. Ponga en Drupal jerga, ofrece oportunidades para los módulos de hook en el ciclo de vida.



**[13 ]**



*Desarrollando para Drupal 7*



Por ejemplo, hemos observado que Drupal comprueba si cualquier módulo necesita ser inicializado. Lo que realmente hace, es mirar para ver si cualquiera de los módulos implementan un hook para la inicialización. ¿Cómo hacer esto? Analiza los módulos cargados para ver si alguno de ellos implementan la función hook\_init (). Para implementar un hook en Drupal es declarar una función que sigue el modelo de nomenclatura hook. Para un módulo de ficción llamado ¡hola para implementar hook\_init (), No haría más que declarar una función llamada hello\_init () (En sustitución de la palabra hook con el nombre del módulo).



Los desarrolladores con experiencia en programación orientada a objetos o con gran conocimiento de los patrones de diseño pueden reconocer esto como algo similar al paradigma de gestión de eventos, capturado en el patrón Passive Observador. Cuando se produce algún evento en particular, Drupal permite a los módulos de la oportunidad de responder a ese evento.



A través de este hook\_init () hook, Drupal ofrece módulos de la capacidad para inicializar ellos mismos o sus propios recursos a la derecha en el inicio de la solicitud. Una vez

todos estos módulos han sido inicializado, Drupal mueve a la siguiente etapa. A medida que avanza a través de la solicitud, llama hook tras hook, dando a los módulos la oportunidad de hacer lo suyo. Por último, se hace una última llamada para los módulos que implementanhook\_exit (), Y luego devuelve los datos restantes para el cliente y termina la petición.

Sistema de hooks de Drupal es quizás el aspecto más importante de la programación Drupal. La mayoría (si no todos) los módulos no son nada más que una colección de implementaciones de hook que Drupal llama automáticamente a medida que se abre camino a través de la solicitud de ciclo de vida. No es sólo el núcleo de Drupal que declara y pide hooks. Cualquier módulo puede declarar y llamar a su propio hook. Muchos módulos, de hecho, declaran sus propios hooks, ofreciendo otros módulos de la oportunidad de integrarse profundamente con sus servicios.

**Módulos del núcleo de Drupal**

Como se señaló anteriormente, Drupal ofrece varios módulos básicos. Estos módulos no se pueden desactivar, ya que sus capacidades son esenciales para el funcionamiento estándar de Drupal. Al igual que otros módulos (incluidos los que estaremos escribiendo), módulos básicos de función mediante la aplicación de los hooks de Drupal.

Como Drupal invoca estos hooks, cada módulo principal responderá cuando sea necesario, la realización de las funciones cruciales en momentos específicos durante el curso de una solicitud.

En la sección siguiente, discutiremos varios módulos básicos con más detalle, explicando qué propósitos sirven estos módulos.

**[14 ]**



*Capítulo 1*



**La base de datos**

Hemos dado un buen paseo a través de una petición típica Drupal, y hemos aprendido un poco más acerca de los módulos y las bibliotecas. Sin embargo, ¿qué pasa con la base de datos?

A diferencia de muchas arquitecturas, para Drupal la base de datos no está parado frente y al centro.

La capa de base de datos no es un módulo, ni qué módulos necesita declarar una relación específica con la base de datos. De hecho, muchos módulos de Drupal no siempre interactúan directamente con la base de datos.



A diferencia de (MVC) marcos Modelo-Vista-Controlador, Drupal no requiere que los módulos siguen un patrón MVC. Por esa razón, puede existir un módulo, sin embargo, no tiene ninguna estructura de base de datos (modelos), un controlador central, o incluso cualquier tematización centrado en el usuario (ver).



En lugar de tratar a la base de datos como un componente arquitectónico central, Drupal se limita a establecer un API para trabajar con la base de datos. En Drupal, la base de datos es sólo un lugar para almacenar los datos. Necesidad tablas personalizadas? Drupal proporciona un sistema para declararlas. Necesidad de obtener datos de la base de datos, o insertar o actualizar datos nuevos? Drupal ofrece funciones y una biblioteca OO para hacerlo. Sin embargo, si usted no necesita estas características para su código, no es necesario trabajar con la base de datos en absoluto. De hecho, en el próximo capítulo vamos a escribir nuestro primer módulo sin necesidad de escribir una sola consulta SQL.

Más adelante en este libro, veremos cómo interactuar con la base de datos utilizando herramientas de bases de datos robustas de Drupal.



**Más que los datos**



La base de datos de Drupal no almacena sólo los datos de la aplicación (o contenidos), sino también de configuración, cachés de metadatos (datos sobre los datos), la información estructural, ya veces incluso el código PHP.

Mientras Drupal puede no ser centrada en la base de datos, que sin duda requiere de una base de datos. Durante la inicialización, Drupal se conectará a una base de datos y recuperar ciertos datos de configuración. Más tarde, como muchos de la carga de módulos básicos, ellos también, en contacto con la base de datos para recuperar o actualizar la información. Para la mayoría de los módulos de Drupal, la base de datos es el lugar para el almacenamiento de datos.

**[15 ]**



*Desarrollando para Drupal 7*



**El tema del sistema**

El componente final de nuestro diagrama arquitectónico inicial es el tema del sistema.

Drupal separa los componentes look-and-feel del sistema del resto de Drupal, y proporciona una manera programática para los datos de tematización. El sistema para el manejo de lo que se llama colectivamente el sistema de tema.

Algunos de los tema del sistema reside en las bibliotecas del núcleo de Drupal. Esta parte es responsable de inicializar temas y localizar qué funciones de temas y plantillas se deben aplicar en determinadas circunstancias.

Sin embargo, la mayoría del código de tema reside en temas y módulos.

Un tema es un conjunto estructurado de código (como un módulo) que proporciona herramientas para la transformación de datos en bruto en la salida con formato. Sitios utilizan al menos un tema para aplicar un look-and-feel consistente y personalizada a todas las páginas del sitio.

Sin embargo, no todo el código tema reside dentro de un tema. Una de las ventajas que ofrece Drupal es la capacidad de definir la tematización por defecto dentro de los módulos, y luego proporcionar mecanismos por los que la capa tema puede reemplazar selectivamente esos temas. En otras palabras, un módulo podría declarar un esbozo del diseño para un componente, pero Drupal proporciona la estructura para un desarrollador es el tema de modificar más tarde el tema (no el módulo) para re-diseño de ese componente en una forma diferente.

Si todo esto suena conceptualmente difícil, no te preocupes. Capítulo 3 y el Capítulo 4 de este libro se dedican a trabajar con el sistema de tematización.

Ahora que hemos tenido una visión arquitectónica rápida, vamos a cambiar las perspectivas y rápidamente buscar por los principales subsistemas que ofrece Drupal.

**Subsistemas principales de Drupal**

En la sección anterior hicimos una vista de pájaro de la arquitectura de Drupal. Ahora vamos a refinar nuestra perspectiva un poco. Vamos a caminar a través de los principales subsistemas que Drupal 7 tiene para ofrecer.

**Temas**

El subsistema tema fue introducido anteriormente, y puesto que los capítulos 3 y 4 lo cubrirá, no vamos a pensar demasiado en ello aquí. Sin embargo, hay algunos detalles que deben ser mencionados al principio.

**[16 ]**



*Capítulo 1*



La responsabilidad de la tematización de un determinado elemento de datos se extiende a lo largo del núcleo de Drupal, los módulos, y el tema se aplica a sí mismo. Si bien no modificamos el código del núcleo de Drupal, es importante para que los desarrolladores sean capaces de entender que tanto el código del módulo y el código de tema pueden manipular la apariencia de los datos.

En este libro, nuestra atención se centrará en la perspectiva del módulo. Trabajamos principalmente con las funciones de la tematización y plantillas que están definidos dentro del módulo. Por lo general, es la mejor práctica para trabajar de esta forma primero-para asegurar que cada módulo tiene la capacidad de tema es de los propios datos.

**Menús**

Drupal no sólo mantiene el contenido, sino también detalles sobre cómo el sitio en sí está organizado. Es decir, las estructuras de cómo se relaciona el contenido.

La forma en principio de que hace esto es a través del subsistema de menú. Este sistema proporciona APIs para generar, recuperar y elementos que describen la estructura del sitio modificando. Ponga en el lenguaje común, que maneja los menús de navegación del sistema.



**Dos sistemas de menú?**

Una fuente de frustración para los desarrolladores nuevos a Drupal es el hecho de que el controlador frontal de la aplicación se llama el**enrutador menú**. Sin embargo, este sistema no es idéntico al subsistema de menú. Su responsabilidad consiste en asignar realidad las URLs a las funciones de devolución de llamada.



Volveremos al router menú en capítulos posteriores.

Los menús son jerárquicos, es decir, que tienen una estructura en forma de árbol. Un elemento de menú puede tener varios hijos, cada uno de los cuales pueden tener sus propios hijos, y así sucesivamente. De esta manera, podemos utilizar el sistema de menús para estructurar nuestro sitio en secciones y subsecciones.

**Nodos**

Tal vez el subsistema más importante a saber es el sistema de nodos. En la jerga de Drupal, un nodo es un pedazo de basado en texto, contenido publicable. Puede tener numerosos campos definidos, pero por lo general tiene un título, un cuerpo, y varias piezas de datos auxiliares, tales como marcas de tiempo, el estado de publicación, y la identificación del autor.



**Los nodos son el contenido**



En informática, el término "nodo" a menudo tiene un significado especial.

La propia definición de Drupal de nodo es distinto. No es un punto en un gráfico, sino más bien un fragmento de contenido. Uno puede preferir pensar en un nodo de Drupal como un documento estructurado.

**[17 ]**



*Desarrollando para Drupal 7*



El sistema de nodos se lleva a cabo principalmente en el módulo de nodo. Este módulo sofisticado proporciona docenas de hooks, aunque significa que muchos otros módulos pueden y deben interactuar con el módulo Node a través de implementaciones de hook.

Puesto que los nodos representan el contenido del sitio, la comprensión del sistema de nodos es un requisito indispensable para el desarrollador de Drupal. Por esa razón, se discuten los aspectos del sistema a lo largo del libro.

**Archivos**

En versiones anteriores de Drupal, generada externamente archivos (sobre todo las imágenes) no eran manejados directamente por Drupal. En su lugar, había un gran número de módulos disponibles para trabajar con archivos.

Esto ha cambiado en Drupal 7, que ahora cuenta con un subsistema de archivo centrada. Esto significa trabajar con imágenes, documentos, etc es ahora considerablemente más fácil.

Mientras Drupal ha tenido durante mucho tiempo un conjunto sofisticado de herramientas para tratar con el sistema de archivos (en el files.inc biblioteca central) ahora hay también una expediente módulo.

*Capítulo 11* analiza esta nueva API.

**Usuarios**

Drupal no está diseñado para ser más que un CMS, sino también una plataforma para los medios de comunicación social. Central a cualquier concepto de medios de comunicación social es un sistema de usuario robusto que puede apoyar no sólo a los usuarios administrativos, sino también los miembros del sitio. Drupal ofrece un poderoso subsistema de usuario que permite a los desarrolladores trabajar con casi todos los aspectos del ciclo de vida del usuario, a partir de lo que los campos aparecen en un perfil de usuario, con lo que los permisos (a un nivel de grano fino) los usuarios tienen, en qué esquema de cifrado en particular es utilizado para cifrar el

la contraseña del usuario.

Sistema de usuarios de Drupal incluso proporciona herramientas para la toma de autenticación y otros aspectos de acoplamiento activo de gestión de usuarios. Los módulos proporcionan, por ejemplo, la integración de LDAP o la autenticación a través de muchos de los servicios de autenticación de acceso público, como OpenID.

Se discute el sistema del usuario, en particular los aspectos de permisos, a lo largo de este libro.

**Comentarios**

Tal vez la herramienta de medios sociales más común es los comentarios. Drupal ofrece un subsistema que proporciona comentario funcionalidad para los nodos (y por extensión, a otros tipos de datos).

**[18 ]**



*Capítulo 1*



Mientras que uno podría imaginar que los comentarios no son más que un tipo de nodo (y, de hecho, hay módulos que hacen esto), los desarrolladores de Drupal han optado por implementar los comentarios como un tipo distinto. Lacomentario módulo contiene la mayoría del código de comentario. Sin embargo, de nuevo, como con el sistema de nodos, que proporciona numerosos hooks, y por lo tanto muchos otros módulos de interactuar con el sistema de comentarios.

**Campos y entidades**

En versiones anteriores de Drupal, el sistema de nodos fue realmente el único sistema para la creación de piezas estructuradas de contenido textual. (Los comentarios se centran demasiado generalmente útiles para la extensión.) Con el fin de ampliar el contenido de nodo más allá de los campos de título y corporales simples, era necesario tanto para escribir de encargo tipos de nodo o utilizar el kit de construcción de contenido (CCK) para construir los tipos de nodo.

Sin embargo, Drupal 7 introduce dos subsistemas importantes que cambian la siguiente:

* El sistema de campos aporta la mayor parte de la funcionalidad de la CCK en el núcleo
* El sistema de entidades hace posible definir otros tipos de datos estructurados que no son nodos

Ya estos nuevos sistemas están haciendo olas entre los desarrolladores de Drupal, con el

Módulo de Drupal Comercio liderando el camino en la definición de las entidades complejas que no son nodos.

Estos dos subsistemas son nuevas, importantes y también compleja. Así que vamos a cubrir en detalle en el Capítulo 6.

**Formularios API**

Otro subsistema de atención que se proporciona en el núcleo de Drupal es la API Forms

(FAPI). Este sistema proporciona una herramienta programática sólida para la definición, presentación, validación y envío de formularios. Se necesita mucho de la concurrida-de trabajo de desarrollo de las formularios, y también añade una capa de seguridad. FAPI es tan esencial para Drupal que usamos varias veces a lo largo del libro.

**Perfiles de montaje**

Más sofisticados casos de uso de Drupal se pueden beneficiar de la posibilidad de personalizar el proceso de instalación. Drupal ofrece un perfil de subsistema de la instalación que se puede aprovechar para crear un instalador personalizado.

**[19 ]**



*Desarrollando para Drupal 7*



Usando esto, los desarrolladores pueden crear temas y módulos personalizados, cambiar los parámetros de instalación, y en general más eficiente el proceso de instalación de sofisticados sitios Drupal.

**Prueba simple**

Probar mediante programación de código es una práctica bien establecida en la industria de desarrollo de software. En Drupal 7, que es una capacidad de la distribución del núcleo de Drupal. Utilizando el marco de prueba simple, los desarrolladores ahora pueden utilizar las pruebas de funcionamiento y de la unidad para validar su código.

Empleamos la prueba a lo largo de este libro. De hecho, vamos a escribir algunas de nuestras primeras pruebas en el Capítulo 2.

**Bloques**

Junto con el contenido principal, la mayoría de las páginas web también tienen contenido adicional que se muestra en la parte superior, inferior o laterales de la página. Subsistema de bloques de Drupal se encarga de la configuración y el despliegue de estas unidades de contenido.

La mayor parte de esta funcionalidad se concentra en la bloque módulo, y vamos a desarrollar nuestro primer bloque personalizado en el capítulo 2.

**Otros subsistemas**

En esta sección, hemos proporcionado cierta información básica sobre varios subsistemas de alto perfil. Sin embargo, esta lista no es exhaustiva. Hay muchos otros, e incluso algunos muy importantes (como Visto) que no están en el núcleo, pero siempre por módulos de terceros.

Algunos de estos otros subsistemas serán introducidos y discutidos en este libro. Sin embargo, Drupal es un sistema sofisticado, y ningún libro de una longitud manejable puede entrar en todos los detalles. Por esa razón, ofrecemos referencias a través del libro que apunta a los desarrolladores los recursos apropiados en

la web y en otros lugares.

**Herramientas para el desarrollo de código de Drupal**

Drupal es una plataforma sofisticada, y de la visión de arriba ya podemos ver que existen numerosos sistemas y estructuras para seguir la pista. En esta sección, tratamos de proporcionar herramientas que simplifican o agilizan el proceso de desarrollo.

**[20 ]**



*Capítulo 1*



Suponemos que usted tiene su propia pila de servidor web y sus propias herramientas de desarrollo de PHP. Los autores de este libro cada utilizan diferentes editores, sistemas operativos, y las pilas de servidor web, por lo que entendemos colectivamente que hay muchas buenas herramientas para el desarrollo de aplicaciones PHP. Y Drupal no requiere nada especial.

Si usted acaba de empezar, es posible que desee mirar a Acquia Drupal (http://acquia.com). Ofrecen pilas de aplicaciones completas para que pueda empezar en Windows, Linux o Mac OS X.

Mientras se ejecuta el depurador PHP desde luego no es necesario, es posible correr Xdebug o el Zend depurador para ser útil. (Uno de los autores de este libro primero aprendieron Drupal trabajó recorriendo paso a paso toda una carga de la página.)

**El control de versiones con Git y CVS**

La gestión de código fuente es una parte importante de cualquier ciclo de vida de desarrollo de software. En este sentido, Drupal 7 coincide con un período de transición importante para la comunidad de Drupal.

En años anteriores, el código fuente de Drupal se ha mantenido en el CVS venerable herramienta (Concurrent Versioning System). Sin embargo, Drupal ha crecido y las necesidades de la comunidad han cambiado. Drupal se está moviendo al sistema de control distribuido de versiones Git.

A medida que empezamos a trabajar con código de Drupal, que le ayudará a ser capaz de tener las herramientas necesarias para trabajar con Git. Desde programas de línea de comandos para aplicaciones de escritorio con todas las funciones, no hay escasez de herramientas para esto.

**El código del libro y Git**

Los autores de este libro han estado trabajando con Git desde hace algún tiempo (uno, de hecho, encabeza la conversión CVS-a-Git). Hemos hecho todo lo posible para asegurarse de que todas las contribuciones de código de este libro están disponibles en un repositorio Git.

Se puede acceder al código de este libro, verlo online en un navegador web, enviar parches, o incluso ramificarse su propia copia y construir su propia herramienta. Todo el código se encuentra en GitHub:

http://github.com/LearningDrupal7Development

Desde allí se podrá acceder al código-y en algunos casos, varias versiones de cada capítulo del mismo código.

**[21 ]**



*Desarrollando para Drupal 7*



**El sitio de API y estándares de codificación**

Se requiere una gran cantidad de conocimientos básicos para escribir un buen código de Drupal. Por supuesto, el objetivo de un libro como este es proporcionar que el conocimiento de fondo. Sin embargo, hay dos recursos de referencia que un desarrollador de Drupal creciente debería tener sobre-mano.

La primera es la documentación oficial de la API en línea. Casi todas las funciones de

Drupal se documenta el uso de la documentación del código en línea. El programa Doxygen se utiliza para extraer esa documentación y darle formato. Puede acceder a la documentación de la API completa en línea en http://api.drupal.org.

Junto con el uso de las API de Drupal, nos esforzamos por cumplir con las convenciones de código de Drupal. Las mejores prácticas en el desarrollo de software incluyen código de mantenimiento limpia, consistente y legible. Un aspecto de esto es la eliminación de matices en el formato de código siguiendo un patrón fijo.

Esto es particularmente importante en una plataforma como Drupal, donde miles de desarrolladores contribuyen al código. Sin estándares de codificación, el código se convertiría en una mezcolanza desordenada de estilos, y valioso tiempo de desarrollo se gastaría sólo descifrar el código en lugar de trabajar en él.

El sitio de Drupal tiene un manual sobre las mejores prácticas (http://drupal.org/node/360052) Que incluye una sección sobre los estándares de codificación (http://drupal.org/coding-standards). Todos los desarrolladores de Drupal cumplan con estas normas.

Aunque hemos tratado de seguir todas las instrucciones de codificación en este libro, no siempre explícitamente señalar lo que estas normas son. Así se anima a los nuevos desarrolladores para examinar las normas de codificación que figuran en la dirección de Internet mencionada anteriormente.

**Módulos orientado al desarrollador**

Hay unos pocos módulos de desarrollo y administrativas Drupal específicos que merecen una mención. Se trata de herramientas que se instalan en el servidor para ayudar a simplificar el desarrollo de Drupal.

**El módulo de desarrollador**

El módulo Developer proporciona varias herramientas sofisticadas diseñadas para ayudar a los desarrolladores a crear y depurar código de Drupal. Para esto, por favor consulte la siguiente

página: http://drupal.org/project/devel

**[22 ]**



*Capítulo 1*



Las siguientes son algunas de las características de este módulo:

* Funciones que se utilizan para volcar objetos y arrays en la salida con formato Drupal
* Herramientas para analizar el uso de la base de datos y el rendimiento
* Una herramienta tema que indica (gráficamente) que se temático elementos de una página por la que las funciones o plantillas
* Un generador de contenido para poblar rápidamente su sitio con contenido de pruebas

**Drush (el shell Drupal)**

A veces es mucho más fácil de ejecutar algunas tareas con un solo comando en una consola. Drush proporciona una interfaz de línea de comandos de Drupal. Puede ser utilizado para ejecutar tareas con sólo pulsar unas teclas en la consola:http://drupal.org/project/drush

En el desarrollo, a menudo tenemos que borrar caches, ejecutar tareas específicas o implementar los datos a un servidor remoto. Drush pueden ayudar a realizar tareas de este tipo.

**Coder**

El módulo Coder ofrece dos grandes características:

* Se puede examinar el código para el cumplimiento contra los estándares de codificación Drupal
* Puede convertir automáticamente los módulos de una versión de Drupal a otro: http://drupal.org/project/coder

Para los nuevos en Drupal, es bueno poder contar con un módulo de evaluar de forma automática si el nuevo código sigue los estándares existentes.

**Resumen**

Este capítulo ha sido una visión general de Drupal para los desarrolladores. Vimos lo tecnologías utiliza Drupal. Nos fijamos en la arquitectura de Drupal. Tomamos una mirada superficial a varios subsistemas prominentes de Drupal. También nos dieron una sensación de que las herramientas de desarrollo orientadas a se van a utilizar cuando se trabaja con Drupal.

Comenzando en el próximo capítulo, vamos a trabajar con el código. De hecho, cada uno de los capítulos siguientes se centrará en los aspectos prácticos de trabajar con Drupal. Viene a continuación es una introducción al sistema de bloques, donde vamos a escribir nuestro primer módulo.

**[23 ]**



Crear el primer módulo

El objetivo de este capítulo es la creación de módulos. En el último capítulo encuestamos a la arquitectura de Drupal avanzado. Aprendimos sobre las características básicas y subsistemas. También vimos algunas herramientas disponibles para el desarrollo. Ahora vamos a comenzar la codificación.

Éstos son algunos de los temas importantes que vamos a cubrir en este capítulo:

* Inicio de un nuevo módulo
* Creación . Info archivos para proporcionar Drupal con la información del módulo
* Creación . Módulo archivos para almacenar el código de Drupal
* Añadir nuevos bloques usando el Bloque Subsistema
* Uso de las funciones de Drupal comunes
* Formateo de código de acuerdo con las normas de codificación de Drupal
* Escribiendo una prueba automatizada para Drupal

Al final de este capítulo, usted debe tener los conocimientos básicos necesarios para la construcción de su propio módulo desde cero.

**Nuestro objetivo: un módulo con un bloque**

En este capítulo vamos a construir un módulo simple. El módulo se utilice el bloque de Subsistema para añadir un nuevo bloque personalizado. El bloque que añadimos simplemente mostrará una lista de todos los módulos actualmente habilitados en nuestra instalación de Drupal.



El subsistema de bloques fue presentado en el capítulo anterior, junto con otros subsistemas Drupal importantes.

*Crear el primer módulo*



Vamos a dividir esta tarea de construir un nuevo módulo en las tres partes:

* Crear un nuevo archivo de carpetas módulo y módulo
* Trabajar con el Bloque Subsistema
* Escribir pruebas automatizadas utilizando el marco SimpleTest incluido en Drupal

Vamos a proceder en este orden por el bien de la simplicidad. Se podría objetar que, a raíz de los procesos de desarrollo ágiles, debemos empezar por escribir nuestras pruebas. Este enfoque se denomina Test-Driven Development (TDD), y es una metodología de justicia popular.



Desarrollo ágil de software es una metodología particular, diseñado para ayudar a los equipos de desarrolladores de forma eficaz y eficiente construir software. Mientras



Drupal no se ha desarrollado mediante un proceso ágil, sí facilita muchas de las prácticas ágiles. Para aprender más sobre ágil, visita

http://agilemanifesto.org/.

Sin embargo, nuestro objetivo aquí no es para ejemplificar una metodología particular, sino para descubrir cómo escribir módulos. Es más fácil aprender el desarrollo del módulo por primera escritura del módulo, y luego aprender a escribir las pruebas unitarias. Es más fácil por dos razones:

* SimpleTest (a pesar de su nombre) es la parte menos simple de este capítulo. Tendrá el doble del peso de nuestro módulo de código real.
* Tendremos que llegar a conocimiento de las API que vamos a utilizar en el desarrollo antes de intentar escribir pruebas que suponen el conocimiento de las API.

En el desarrollo normal del módulo, sin embargo, que sin duda puede optar por seguir el

Enfoque TDD de escribir las pruebas primero, y luego escribir el módulo. Ahora vamos a pasar a la primera etapa de creación de un nuevo módulo.

**Creación de un nuevo módulo**

Creación de módulos de Drupal es fácil. ¿Qué tan fácil? Es bastante fácil que más de 5.000 módulos se han desarrollado, y muchos desarrolladores de Drupal son novatos incluso PHP! De hecho, el código de este capítulo es una ilustración de cómo puede ser fácil codificación módulo. Vamos a crear nuestro primer módulo con un solo directorio y dos archivos pequeños.

**[26 ]**



*Capítulo 2*



**Los nombres de módulo**

No hace falta decir que la construcción de un nuevo módulo requiere nombrar el módulo. Sin embargo, hay una ambigüedad de menor importancia que debe ser aclarado desde el principio, un módulo de Drupal tiene dos nombres:

* **Un nombre legible**: Este nombre está diseñado para ser leído por los seres humanos, y debe ser uno o un par de palabras de largo. Las palabras deben ser capitalizados y separados por espacios. Por ejemplo, uno de los módulos más populares de Drupal tiene el nombre legibles Vistas. A menos populares (pero quizás más creativa llamada) módulo de Drupal 6 tiene el nombre legible por humanos

**Eldorado Superfly**.

* **Un nombre legible por máquina**: Este nombre es utilizado internamente por Drupal. Puede estar compuesto de minúsculas y mayúsculas letras, dígitos y el guión bajo (utilizando letras mayúsculas en los nombres de la máquina está mal visto, sin embargo). No se permiten otros caracteres. Los nombres de los equipos de los dos módulos anteriores sonvistas y eldorado\_superfly, Respectivamente.

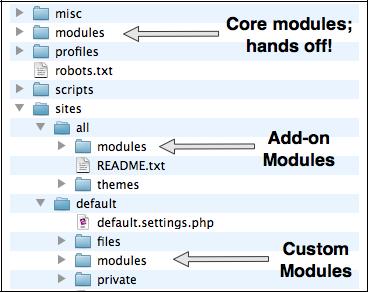
Por convención, los dos nombres debe ser lo más similar posible. Espacios deben ser reemplazados por guiones bajos. Letras mayúsculas generalmente deben cambiar a minúsculas.

Debido a la convención de asignación de nombres similares, los dos nombres por lo general se pueden utilizar indistintamente, y la mayoría de las veces no es necesario declarar específicamente cuál de los dos nombres que nos referimos. En los casos en que la diferencia que hay que conseguir (como en la siguiente sección), los autores deberán tener cuidado para hacerlo.

**¿De dónde viene nuestro módulo van?**

Uno de los aspectos menos intuitivos de desarrollo de Drupal es la disposición del sistema de ficheros. ¿Dónde ponemos un nuevo módulo? La respuesta obvia sería la de poner en el

/ modules directorio junto con todos los módulos principales.



**[27 ]**



*Crear el primer módulo*



Por obvio que parezca, la / modules carpeta no es el lugar adecuado para sus módulos. De hecho, nunca se debe cambiar nada en ese directorio. Está reservado para los módulos del núcleo de Drupal solamente, y será sobreescrito durante las actualizaciones.

El segundo lugar, mucho menos obvio para poner los módulos se encuentra en / sites / all / modules. Este es el lugar donde todos los no modificados módulos adicionales deben ir, y herramientas como Drush

(Una herramienta de línea de comandos Drupal) descargará los módulos de este directorio.

En cierto sentido, está bien para poner los módulos aquí. Ellos no serán reemplazados automáticamente durante las actualizaciones de la base.

Sin embargo, a partir de este escrito, / sites / all / modules no es el lugar recomendado para poner módulos personalizados menos que esté ejecutando una configuración multi-sitio y el módulo de encargo tiene que ser accesible a todos los sitios.

La recomendación actual es poner módulos personalizados en el / Sites / default / módulos directorio, que no existe de forma predeterminada. Esto tiene algunas ventajas. Uno es que los módulos adicionados estándar se almacenan en otro lugar, y esta separación hace que sea más fácil para nosotros encontrar nuestro propio código sin clasificar a través de obstáculos. Hay otros beneficios (por ejemplo, el orden de carga de los directorios de módulos), pero ninguno tendrá un impacto directo sobre nosotros.



A lo largo de este libro, siempre vamos a estar poniendo nuestros módulos personalizados en / sites / default / modules. Así se desprende de Drupal mejores prácticas, y también hace que sea fácil encontrar nuestros módulos en lugar de todos los otros módulos adicionales.



La única desventaja de almacenar todos los módulos personalizados en/ sites / default / modules aparece sólo en un conjunto específico de circunstancias. Si usted tiene Drupal configurado para servir a múltiples sitios fuera de una sola instancia, entonces el / Sites / default carpeta sólo se utiliza para el sitio predeterminado. Lo que significa esto, en la práctica, es que los módulos almacenados allí no se cargarán en absoluto para otros sitios.

En estos casos, generalmente se aconseja mover los módulos personalizados en

/ Sites / all / modules / custom.



**Otros directorios de módulos**



Drupal tiene un aspecto en algunos otros lugares para módulos. Sin embargo, esos lugares se reservan para propósitos especiales.

**[28 ]**



*Capítulo 2*



**Crear el directorio de módulos**

Ahora que sabemos que nuestros módulos deben ir en / sites / default / modules, Podemos crear un nuevo módulo de allí.

Los módulos pueden ser organizados en una variedad de maneras, pero la mejor práctica es crear un directorio de módulos en / sites / default / modules, Y luego colocar al menos dos archivos dentro del directorio: un . Info (Pronunciado "punto-info") de archivo y una . Módulo ("Dot-módulo") de archivos.

El directorio debe ser nombrado con el nombre legible por la máquina del módulo. Del mismo modo, tanto el . Info y . Módulo archivos deben utilizar el nombre legible por la máquina.

Vamos a nombrar a nuestro primer módulo con el nombre legible por máquina primero, Ya que es nuestro primer módulo. Por lo tanto, vamos a crear un nuevo directorio, / Sites / default / modules / primero, Y luego crear un first.info archivo y un first.module archivo:



Esos son los únicos archivos que necesitaremos para nuestro módulo.

Para los permisos, asegúrese de que su servidor web puede leer tanto la . Info y

. Módulo archivos. No debería ser capaz de escribir en cualquiera de los archivos, sin embargo.



En cierto sentido, el único archivo que sea absolutamente necesario para un módulo es la



. Info archivo situado en un lugar adecuado en el sistema. Sin embargo, desde la . Info fichero simplemente contiene información acerca del módulo, hay un módulo interesante puede ser construido con sólo este archivo.

A continuación, vamos a escribir el contenido de la . Info presentar.

**Escribiendo el archivo de información.**

El propósito de la . Info archivo es proporcionar Drupal con información sobre un módulo de información, como el nombre legible por humanos, ¿qué otros módulos requiere este módulo, y qué archivos Este módulo proporciona código.

La. Info archivo es un archivo de texto sin formato en un formato similar al archivo de configuración INI estándar. Una directiva de la . Info archivo se compone de un nombre y el signo igual y un valor:

name = valor

**[29 ]**



*Crear el primer módulo*



Por convenciones de código de Drupal, que siempre debe haber un espacio a cada lado del signo igual.

Algunas directrices utilizan una sintaxis de array como para declarar que un nombre tiene varios valores. El formato-array como se ve así:

Nombre [] = valor1 nombre [] = valor2

Tenga en cuenta que no hay ningún espacio en blanco entre los corchetes de apertura y el corchete de cierre.

Si un valor se extiende por más de una línea, debe ir entre comillas.

Cualquier línea que comienza con un ; (Punto y coma) se trata como un comentario y es ignorado por el intérprete Drupal INI.



Drupal no soporta encabezados de sección de estilo INI tales como las que se encuentran en el php.ini presentar.

Para empezar, vamos a echar un vistazo a una completa first.info presentar para nuestro primer módulo:

; $ Id $

name = Primera

Descripción = Un primer módulo. paquete = Drupal 7 core Desarrollo = 7.x

archivos [] = first.module

; Dependencias [] = carga automática; php = 5.2

Este archivo de diez líneas es casi tan complejo como un módulo de . Info archivo alguna vez llega.

La primera línea es un estándar. Cada . Info archivo debe comenzar con ; $ Id $. ¿Qué es esto? Es el marcador de posición para el sistema de control de versiones para almacenar información sobre el archivo. Cuando el archivo está desprotegido en el repositorio CVS de Drupal, la línea se ampliará automáticamente a algo como esto:

; $ Id: first.info, v 1.1 2009/03/18 20:27:12 mbutcher Exp $

Esta información indica que el archivo se ha pasado registró en CVS, y que lo haya facturado pulg

**[30 ]**



*Capítulo 2*



CVS va a desaparecer, por lo que es $ Id $. Mientras Drupal ha sido desarrollado en CVS desde los primeros días a través de Drupal 7, ahora se está migrando a un repositorio Git. Git no utiliza $ Id $, Por lo que es probable que entre la liberación de Drupal 7 y la liberación de Drupal 8, $ Id $ se eliminarán las etiquetas.



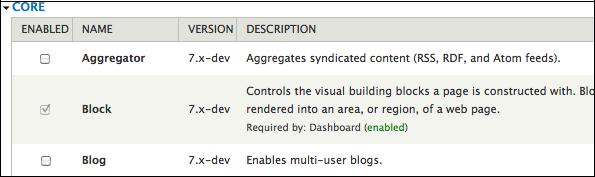
A lo largo de este libro, verá todas PHP y . Info los archivos que comienzan con el $ Id $ marcador. Una vez Drupal usa Git, esas etiquetas pueden desaparecer.

El siguiente par de líneas de interés en first.info son los siguientes:

name = Primera

Descripción = Un primer módulo. paquete = Drupal 7 Desarrollo

Los dos primeros son necesarios en todos los . Info presentar. La nombre Directiva se utiliza para declarar qué nombre legible por humanos del módulo es. La descripción proporciona una descripción de una o dos frases de lo que este módulo proporciona o se utiliza. Entre otros lugares, esta información se muestra en la sección de configuración del módulo de la interfaz de administración en los módulos.



En la captura de pantalla, los valores de la nombre y descripción campos se muestran en sus respectivas columnas.

El tercer elemento, paquete, Identifica qué familia (paquete) de los módulos de este módulo está relacionado. Los módulos del núcleo, por ejemplo, todos tienen el paquete Núcleo. En la imagen anterior, se puede ver el paquete de agrupación Core en la esquina superior izquierda.

Nuestro módulo se agrupan bajo el paquete Drupal 7 Desarrollo para representar su relación con este libro. Como usted puede notar, los nombres de paquetes se escriben como valores legibles.



Al elegir un nombre de módulo legible, recuerde que debe cumplir con las especificaciones mencionadas anteriormente en esta sección.

**[31 ]**



*Crear el primer módulo*



La siguiente directiva es la núcleo Directiva: core = 7.x. Esto simplemente declara que se requiere la versión de la línea principal de Drupal por el módulo. Todos Drupal 7 módulos tendrán la línea core = 7.x.

Junto con la versión de la base, una . Info archivo también puede especificar qué versión de PHP que requiere. Por defecto, Drupal 7 Drupal requiere 5.1 o más reciente. Sin embargo, si uno fuera a usar, por ejemplo, los cierres (una característica introducida en PHP 5.3), entonces sería necesario añadir la siguiente línea:

php = 5.3

A continuación, todos los . Info archivo debe declarar que los archivos del módulo contienen funciones de PHP, clases o interfaces. Esto se hace usando la archivos [] Directiva. Nuestro pequeño módulo inicial sólo tendrá un archivo, first.module. Así que tenemos un solo archivos [] Directiva.

archivos [] = first.module

Archivos más complejos a menudo tienen variosarchivos [] directivas, cada declaración de un archivo de código fuente PHP separado.



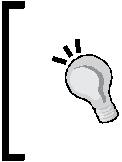
JavaScript, CSS, archivos de imagen, y PHP archivos (como plantillas) que no contienen funciones que el módulo necesita saber acerca de que no necesite ser incluida en archivos [] directivas. El punto de la directiva es simplemente para indicar a Drupal que estos archivos deben ser examinados por Drupal.



Una directiva que no vamos a utilizar para este módulo, sino que juega un papel muy importante es la dependencias [] Directiva. Esto se utiliza para enumerar los otros módulos que deben instalarse y activas para este módulo funcione correctamente. Drupal no permitirá

un módulo para ser activado a menos que sus dependencias han sido satisfechas.

Drupal no contiene una directiva para indicar que se recomienda otro módulo o es opcional. Es la tarea de los desarrolladores para documentar adecuadamente este hecho y darlo a conocer. Actualmente no existe una práctica recomendada para proporcionar dicha información.



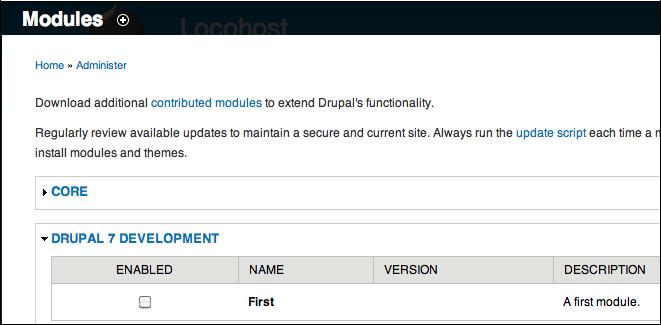
**[32 ]**



*Capítulo 2*



Ahora hemos creado nuestro first.info presentar. Tan pronto como Drupal lee este archivo, aparecerá en nuestra página Módulos módulo.



En la imagen, observe que el módulo aparece en el paquete de desarrollo de Drupal 7, y tiene el nombre y la descripción que le sean asignadas en el . Info presentar.

Con nuestro . Info archivo completado, ahora podemos seguir adelante y codificar nuestra . Módulo presentar.



Módulos controladas en el sistema de control de versiones de Drupal tendrán automáticamente unversión Directiva añade a la . Info presentar. Esto no debería normalmente ser alterado.



**Creación de un archivo de módulo**

La . Módulo archivo es un archivo PHP que contiene convencionalmente todas las principales implementaciones de hook para un módulo. Hablamos de hooks en un nivel alto en el primer capítulo. Ahora vamos a obtener algún conocimiento práctico de ellos.

Una implementación de hook es una función que sigue un cierto patrón de nomenclatura con el fin de indicar a Drupal que se debe utilizar como una devolución de llamada para un evento en particular en el sistema de Drupal. Para los programadores orientados a objetos, puede ser útil pensar en un hook tan similar al modelo de diseño de observador.

**[33 ]**



*Crear el primer módulo*



Cuando Drupal encuentra un evento para el cual no es un hook (y hay cientos de este tipo de eventos), Drupal se verá a través de todos los módulos para adaptar las implementaciones de hook. A continuación, ejecutar cada aplicación de hook, uno tras otro. Una vez que se han ejecutado todas las implementaciones de hook, Drupal continuará su procesamiento.

En el pasado, todas las implementaciones de hook Drupal tenían que residir en el . Módulo presentar. Requisitos de Drupal 7 son más indulgentes, pero en la mayoría de los módulos de tamaño moderado, sigue siendo preferible almacenar la mayoría de las implementaciones de hook en el . Módulo presentar.



Más adelante en este libro se encontrará con los casos en que las implementaciones de hook pertenecen a otros archivos. En tales casos, se explicarán las razones para organizar el módulo de tal manera.



Para empezar, vamos a crear un simple . Módulo archivo que contiene una sola aplicación de hook - uno que proporciona información de ayuda.

<? Php / / $ Id $

/ \*\*

* @ File
* Un módulo de Drupal que ejemplifica prácticas de codificación y APIs.
* Este módulo proporciona un bloque que muestra todos los
* módulos instalados. Ilustra las normas de codificación,
* prácticas, y el uso de la API de Drupal 7.

\* /

/ \*\*

\* Implementa hook\_help (). \* /

función first\_help ($ ruta, $ arg) {if ($ ruta == 'admin / help primera #') {

volver t ('Un módulo de demostración.');

}

}

Antes de entrar en el código en sí, vamos a hablar de algunos elementos estilísticos.

Para empezar, tenga en cuenta que este archivo, al igual que el . Info archivo, contiene un $ Id $ marcador que CVS reemplazará cuando el archivo se hizo el check in Todos los archivos PHP debe tener este marcador tras un comentario al estilo doble barra: / / $ Id $.

A continuación, el código anterior ilustra algunas de las normas de codificación importantes para Drupal.

**[34 ]**



*Capítulo 2*



**Las normas del código fuente**

Drupal tiene un conjunto exhaustivo y se aplican estrictamente las normas de codificación. Todo el código del núcleo se adhiere a estas normas. Más módulos add-on también lo hacen. (. Aquellos que generalmente no reciben informes de errores por no ajustarse) Antes de comenzar la codificación, que es una buena idea para familiarizarse con las normas tal como se documenta aquí:http://drupal.org/ codificación de las normas El módulo Coder mencionado en el capítulo anterior puede evaluar su código y que le avise a todas las infracciones a las normas de codificación.



A lo largo de este libro vamos a cumplir con las normas de codificación de Drupal. En muchos casos, vamos a explicar las normas a medida que avanzamos. Aún así, la fuente definitiva de las normas es el



URL en la lista anterior, no nuestro código aquí.

No vamos a reiterar las normas de codificación en este libro. Los detalles se pueden encontrar en línea. Sin embargo, varias normas importantes merecen mención inmediata. Me limitaré a mencionar aquí, y vamos a ver ejemplos en acción a medida que trabajamos a través del código.

* **Sangría**: Todos los archivos de PHP y JavaScript utilizan dos espacios para sangrar. Aquí nunca se usan para el formato de código.
* **La** <? Php?> **instrucción del procesador**: Los archivos que están completamente PHP debe empezar con<? Php, Sino que debe omitir el cierre ?>. Esto se realiza por varias razones, más notablemente para prevenir la inclusión de espacios en blanco de romper las cabeceras HTTP.
* **Comentarios**: Drupal usa al estilo Doxygen (/ \*\*\* /doc)-blocks para comentar funciones, clases, interfaces, constantes, archivos y variables globales. Todos los demás comentarios deben utilizar la doble barra inclinada (/ /) Comentario. El signo de libra (#) No debe ser utilizado para hacer comentarios.
* **Los espacios alrededor operadores**: La mayoría de los operadores deben tener un carácter de espacio en blanco en cada lado.
* **Separación de las estructuras de control**: estructuras de control deben tener espacios después del nombre y antes del corchete. Los cuerpos de todas las estructuras de control deben estar rodeadas por llaves, e incluso la desi declaraciones con cuerpos de una sola línea.
* **Funciones**: Funciones deben ser nombradas con letras minúsculas utilizando guiones para separar las palabras. Más adelante veremos cómo los nombres de método de clase difieren

de este.

* **Variables**: Los nombres de variables deben estar en minúsculas utilizando guiones para separar las palabras. Las variables de miembro de los objetos se denominan de manera diferente.

A medida que trabajamos a través de ejemplos, veremos estas y otras normas en la acción.

**[35 ]**



*Crear el primer módulo*



**Módulos Doc estilo Doxygen**

Drupal utiliza Doxygen para extraer documentación de la API de código fuente. Codificadores PHP experimentados pueden reconocer este concepto, ya que es similar a los comentarios phpDocumentor (o JavaDoc de Java). Sin embargo, Drupal tiene su idiosincrasia, y no seguir las mismas convenciones que estos sistemas.

Sólo vamos a ver los bloques de documentación que se aplican a nuestro ejemplo específico anterior. A medida que avanzamos a través del libro, veremos ejemplos más avanzados de las prácticas de documentación correctos.

Echemos un vistazo más de cerca a la primera docena de líneas de nuestro módulo:

<? Php / / $ Id $

/ \*\*

* @ File
* Un módulo de Drupal que ejemplifica prácticas de codificación y APIs.
* Este módulo proporciona un bloque que muestra todos los
* módulos instalados. Ilustra las normas de codificación,
* prácticas, y el uso de la API de Drupal 7.

\* /

Después de la instrucción del procesador de PHP y la $ Id $ línea, la parte del código es un gran comentario. El comentario comienza con una barra y dos asteriscos (/ \*\*) Y termina con un solo asterisco y una barra (\* /). Cada línea entre comienza con un asterisco. Este estilo de comentario se denomina bloque doc o bloque de documentación.

Un bloque doc es un comentario que contiene información de la API. Se puede extraer automáticamente por herramientas externas, que luego se puede formatear la información para su uso por los desarrolladores.



**Bloques Doc en acción: api.drupal.org**

Módulos Doc de Drupal se utilizan para generar la fuente definitiva de Drupal



Documentación de la API en http://api.drupal.org. Este sitio es una fantástica interfaz de búsqueda a todas y cada una de las funciones de Drupal, clases, interfaces y constantes. También contiene algunos consejos útiles de cómo-a la documentación.

Todos Drupal se documenta el uso de módulos Doc, y siempre se debe utilizar para documentar el código.

**[36 ]**



*Capítulo 2*



El bloque doc inicial en el fragmento de código anterior comienza con el @ File decorador.

Esto indica que el bloque doc describe el archivo como un todo, no una parte de él. Cada archivo debe comenzar con un bloque de doc a nivel de archivo.

A partir de ahí, el formato de este bloque doc es simple: Se inicia con una descripción de la frase única del archivo (que siempre debe estar en una línea), seguido de una línea en blanco, seguido de una o más descripciones de párrafo de lo que este archivo hace.

Los estándares de codificación de Drupal estipulan que módulos Doc siempre deben ser escritos usando y gramaticalmente correctas, frases interrumpidas completos.

Si miramos un poco más allá en nuestro archivo de módulo, podemos ver nuestra primera declaración de la función:

/ \*\*

\* Implementa hook\_help (). \* /

función first\_help ($ ruta, $ arg) {if ($ ruta == 'admin / help primera #') {

volver t ('Un módulo de demostración.');

}

}

Antes de pasar a la función, vamos a echar un vistazo a la cuadra doc aquí. Se trata de una sola frase: Implementa hook\_help (). Esta descripción de la frase única sigue un estándar de codificación de bloque doc Drupal, también. Cuando una función es una aplicación de hook, se debe indicar de manera exactamente en el formato utilizado anteriormente: Implementa NOMBRE DE HOOK. ¿Por qué la fórmula? Así que los desarrolladores pueden identificar muy rápidamente con el propósito general de la función, y también para que las herramientas automatizadas pueden encontrar implementaciones de hook.

Tenga en cuenta que no añadimos nada más de una descripción, ni nos documentamos los parámetros. Esto está bien cuando dos cosas son ciertas:

* La función implementa un hook
* La función es simple

En estos casos, la descripción de una sola línea va a hacer, ya que los programadores pueden simplemente referirse a la documentación de la API para el hook para aprender más.

Más adelante veremos cómo las funciones no de hook y hook implementaciones más complejas tienen una forma extendida de bloque doc comentario. Por ahora, sin embargo, nos hemos ocupado de los aspectos básicos de módulos Doc. Vamos a seguir adelante y mirar a la función de ayuda.

**[37 ]**



*Crear el primer módulo*



**El hook de ayuda**

Drupal define un hook de llamada hook\_help (). El hook de la ayuda se invoca (llama) cuando un usuario navega por el sistema de ayuda. Cada módulo puede tener una implementación de hook\_help (). Nuestro módulo proporciona breve texto de ayuda mediante la implementación del hook de ayuda.

función first\_help ($ ruta, $ arg) {if ($ ruta == 'admin / help primera #') {

volver t ('Un módulo de demostración.');

}

}

¿Cómo afecta esta función se convierta en una aplicación de hook? Estrictamente en virtud de su nombre: first\_help (). El nombre sigue el patrón de hook. Si el hook se llama hook\_help (), Y luego ponerlo en práctica, se sustituye la palabra hook con el nombre de el módulo. Por lo tanto, para poner en prácticahook\_help (), Simplemente declaramos una función en nuestra primero módulo llamado first\_help ().

Cada hook tiene sus propios parámetros y todos los hooks del núcleo de Drupal están documentados en http://api.drupal.org.

Lahook\_help () aplicación toma dos argumentos:

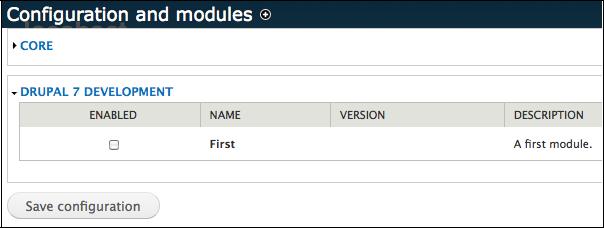
* $ Ruta: La ruta URI sistema de ayuda
* $ Arg: Los argumentos utilizados cuando se accede a esta URL

En nuestro caso, sólo estamos preocupados por el primero de estos dos. Básicamente, el sistema de ayuda hace coincidir los caminos de URI para ayudar texto. Nuestro módulo tiene que declarar lo que el texto de ayuda debe ser devuelta por los URIs específicas.

En concreto, el texto de ayuda en todo el módulo debe estar disponible en el URIadmin / ayuda # MODULE\_NAME, En donde MODULE\_NAME es el nombre legible por la máquina del módulo.

Nuestra función se activa marcando la $ Ruta. Si el $ Ruta está ajustado a admin / ayuda # primero, La pantalla de ayuda por defecto de un módulo, entonces se volverá un poco de texto simple ayuda.

Si tuviéramos que permitirá a nuestro nuevo módulo y luego ver página texto de ayuda de Drupal con nuestro nuevo módulo habilitado, veríamos esto:



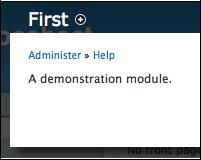
**[38 ]**



*Capítulo 2*



Observe que Ayuda ahora aparece bajo OPERACIONES. Si tuviéramos que hacer clic en el enlace Ayuda, veríamos nuestro texto de ayuda:



La clave para hacer que este sistema funcione es en el uso de la $ Ruta comprobación, que muestra la información de ayuda sólo cuando la ayuda sensible al contexto para este módulo está habilitado

vía hook\_help ().

if ($ ruta == 'admin / help # first') {return t ('Un módulo de demostración.');

}

Dado que este es nuestro primer módulo, vamos a detenernos en los detalles un poco más cuidado que lo haremos en los capítulos siguientes.

En primer lugar, el código anterior se ajusta a los estándares de codificación de Drupal, que hemos cubierto brevemente antes. El espacio en blanco separa la si y el paréntesis de apertura (, Y también hay un espacio entre el paréntesis de cierre ) y la llave de apertura ({). También hay espacios en ambos lados del operador de igualdad ==. Código se escribe con dos espacios por nivel, y nunca usar pestañas. En general, los codificadores de Drupal suelen utilizar comillas simples (') Para rodear cadenas a causa de la verdad es leve mejora de velocidad () ganó saltando por interpolación.

También es importante desde el punto de vista de las normas de codificación es el hecho de que adjuntamos el cuerpo de la si declaración rizado tirantes aunque el cuerpo es sólo una línea larga. Y lo dividimos en tres líneas, a pesar de que podría haber sido capaz de encajar en una.

Normas Drupal requieren que siempre lo hacemos.

Por último, en el ejemplo anterior, vemos una nueva función de Drupal: t ().

**La función y las traducciones () t**

Cada cadena de lenguaje natural que se puede mostrar a un usuario debe ser envuelto en el t () función. ¿Por qué? Debido a que el t () función es responsable de traducir las cadenas de un idioma a otro.

**[39 ]**



*Crear el primer módulo*



Drupal es compatible con docenas de idiomas. Este es uno de los rasgos más característicos de la internacionalización y localización esfuerzo de Drupal. El método por el cual Drupal soporta la traducción es en gran medida a través de lat () función.

Hay tres características de esta función que cada desarrollador debe entender:

* ¿Qué sucede cuando t () se llama
* Cómo Drupal construye la tabla de traducción
* Las características adicionales se obtiene mediante el uso de la t () función

En primer lugar, echemos un vistazo a lo que el t () función hace cuando se le llama. Si no hay soporte de idiomas está habilitado y hay un segundo argumento se pasa a t (), Simplemente se devuelve la cadena inalterada. Si más idiomas están habilitados y el idioma del usuario es algo que no sea Inglés, Drupal intentará reemplazar la cadena de idioma Inglés con una cadena en el idioma correspondiente.

La segunda cosa a tener en cuenta es la forma en Drupal se basa la información de la traducción. Hay dos aspectos en este: El aspecto humano y el técnico. Las traducciones son realizadas por los propios docenas y docenas de voluntarios que traducen no sólo el núcleo de Drupal, sino también muchos de los módulos adicionales. Sus traducciones se hacen entonces en paquetes de idioma descargable (. Po archivos) que se pueden instalar en

su sitio.

En el lado más técnico, este dedicado grupo de traductores no se limita a buscar el código fuente en busca de llamadas a la t () función. En lugar de ello, una herramienta automatizada entresaca el código e identifica todas las cadenas traducibles. Esta herramienta automatizada, sin embargo, sólo puede extraer los literales de cadena. En otras palabras, busca llamadas como esta:

t ('Este es mi cadena');

No se puede hacer nada con líneas como esta, sin embargo:

$ Variable = 'Esto es una cadena "; t ($ variable);

¿Por qué no funcionará el sistema de traducción en el caso anterior? Porque cuando el sistema de traducción automática funciona a través del código, no se ejecuta el código. Simplemente lee. Por esa razón, se convertiría en engorroso (y muchas veces imposibles) para determinar cuál es el valor correcto de una variable es.



La lugar módulo puede, bajo ciertas circunstancias, identificar otras cadenas que no se han pasado correctamente en el t () funcionar y ponerlos a disposición de los traductores. Esto, sin embargo, no se debe confiar.



**[40 ]**



*Capítulo 2*



Así que el t () función siempre debe darse una cadena literal para su primer argumento.

La tercera cosa a tener en cuenta sobre la t () función es que hace más que traducir las cadenas. Se ofrece un método de interpolación variable que es más seguro que el método habitual.

En muchas de las aplicaciones PHP, verá código como este:

print "Bienvenido, $ nombre de usuario.";

El código anterior reemplazará$ Usuario con el valor de la $ Usuario variable. Este código deja abierta la posibilidad de que el valor de $ Usuario contiene datos que romperá el código HTML de la salida - o peor aún, que abrirá una vía para que un usuario malicioso para inyectar JavaScript u otro código en la salida.

La t () función proporciona un suplente y más seguro, método para reemplazar los marcadores de posición de texto con un valor. La función toma un segundo argumento opcional, que es un array asociativo de artículos que pueden ser sustituidos. He aquí un ejemplo que reemplaza el código anterior:

valores $ = array ('@ usuario' => $ nombre de usuario); t print ('Bienvenido, @ usuario', $ valor);

En el caso anterior, declaramos un marcador de posición el nombre @ Usuario, El valor de los cuales es el valor de la $ Usuario variable. Cuando el t () función se ejecuta, las asignaciones en valores de $ se utilizan para sustituir los marcadores de posición con los datos correctos.

Pero hay un beneficio adicional: estas sustituciones se realizan de una manera segura.

Si el marcador de posición comienza con @, A continuación, antes de insertar el valor, Drupal desinfecta el valor con su interiorcheck\_plain () función (que nos vamos a encontrar muchas veces en los capítulos siguientes).

Si está seguro de que la cadena no contiene ninguna información peligrosa, puede utilizar un símbolo diferente para comenzar su marcador de posición: el signo de exclamación (!). Cuando eso se utiliza, Drupal simplemente insertar el valor como es. Esto puede ser muy útil cuando se necesita para insertar los datos que no se deben traducir:

valores $ = array ('! url' => 'http://example.com'); t print ('La página web se puede encontrar en url', los valores de $);

En este caso, la dirección se introducirá sin escapatoria. Podemos hacer esto de manera segura sólo porque ya sabemos el valor de la URL. No viene de un usuario desconfiado.

**[41 ]**



*Crear el primer módulo*



Por último, existe un tercer decorador de marcador de posición: el signo de porcentaje (%) Dice Drupal para escapar del código y marcarlo como subrayó.

valores $ = array ('% de color' => 'azul');

print t ('Mi color favorito es el% de color.', los valores de $);

Esto no sólo eliminar los caracteres peligrosos del valor, sino que también insertará marcado para tratar ese texto como texto subrayado. Por defecto, el código anterior daría lugar a la impresión de la cadena Mi color favorito es el azul <em> </ em>. Las etiquetas de énfasis fueron añadidas por una función de tema (theme\_placeholder ()) Convocada por la t () función.

Hay más cosas que se pueden hacer con t (),format\_plural (), Los contextos de traducción y otras características del sistema de traducción. Para obtener más información, es posible que desee comenzar con la documentación de la API parat () en http://api.drupal.org/api/

function/t/7.

Hemos tomado un desvío considerable para hablar sobre el sistema de traducción, pero con buena razón. Es una característica tremendamente poderosa de Drupal, y debe ser utilizado en todo el código. No sólo hace que los módulos traducible, pero añade una capa de seguridad.

Incluso se puede poner a algunas interesantes (si no ortodoxa) utiliza, como se ejemplifica por el

Módulo de Cuerda anulaciones en http://drupal.org/project/stringoverrides.

En este punto, hemos creado un módulo de trabajo, aunque lo único que hace es mostrar el texto de ayuda. Es hora de hacer este módulo un poco más interesante. En la siguiente sección vamos a utilizar el API de bloque para escribir código que genera un bloque de la lista de todos los módulos que se activa.

**Trabajar con la API Bloquear**

En el primer capítulo hemos hablado de los bloques, y en el uso de su paso de Drupal, que ya tiene, sin duda, la configuración y la gestión de bloques encontrado. En esta sección, vamos a aprender a crear bloques de código. La API de bloque proporciona las herramientas para conectar código personalizado en el subsistema de bloque.



El API Block ha cambiado sustancialmente desde Drupal 6. En Drupal 6, sólo había una función que se utiliza para todas las operaciones de bloque. Ahora hay una familia de funciones relacionadas.



Vamos a crear un bloque que muestra una lista con viñetas de todos los módulos activados actualmente en nuestro sitio.

**[42 ]**



*Capítulo 2*



Hay media docena de hooks en el API de bloque, proporcionando oportunidades para hacer de todo, desde la que se declara nuevos bloques para alterar el contenido y el comportamiento de los bloques existentes. Para nuestro módulo simple, vamos a utilizar dos hooks diferentes:

* hook\_block\_info (): Se utiliza para contar Drupal sobre el nuevo bloque o bloques que declararemos
* hook\_block\_view (): Esto le dice a Drupal qué hacer cuando se solicita un bloque para la visión

Una cosa a tener en cuenta, en el contexto de la API de bloque, así como otras APIs es que cada módulo sólo se puede implementar un enlace dado de una vez. Sólo puede haber unafirst\_block\_info () función.

Como los módulos deben ser capaces de crear múltiples bloques, que significa que el

Bloquear API debe hacer posible que una implementación de bloques para gestionar múltiples bloques. Por lo tanto, first\_block\_info () puede declarar cualquier número de bloques y first\_block\_view () puede devolver cualquier número de bloques.



El bloque de la totalidad de la API se documenta en el Drupal oficial 7



Documentación de la API, e incluso incluye un módulo de ejemplo: http://api.drupal.org/api/drupal/developer-- ejemplos - block\_example.module / 7.

Para mantener nuestro ejemplo simple, vamos a crear un solo bloque. Sin embargo, es bueno tener en cuenta que la API se ha diseñado de una manera que nos permitirá crear tantos bloques como queramos.

Vamos a empezar con una implementación de hook\_block\_info ().

**El hook de bloque de información**

Todas las funciones de nuestro módulo va a ir dentro de la first.module presentar-la ubicación por defecto para implementaciones de hook en Drupal. Antes, hemos creado first\_help (), Una implementación de hook\_help (). Ahora, vamos a aplicar el hook\_block\_info () hook.

El propósito de este libro es contar Drupal acerca de todos los bloques que el módulo proporciona. Tenga en cuenta que, como con cualquier hook, sólo tiene que ponerlo en práctica en los casos en que el módulo necesita para proporcionar esta funcionalidad. En otras palabras, si no se implementa el hook, Drupal simplemente asumir que este módulo no tiene bloques asociados.

**[43 ]**



*Crear el primer módulo*



Aquí está nuestra 'Información bloque' aplicación hook declarar un solo bloque:

/ \*\*

\* Implementa hook\_block\_info (). \* /

función first\_block\_info () {$ cuadras = array ();

$ bloques ['list\_modules'] = array (

'Info' => t ('Una lista de todos los módulos habilitados.'), 'Cache' => DRUPAL\_NO\_CACHE,

);

return $ bloques;

}

Una vez más, esta función está precedido por un bloque doc. Y ya que estamos escribiendo una aplicación trivial de hook\_block\_info (), No tenemos que añadir aparte de la documentación estándar nada.

Una implementación de hook\_block\_info () no tiene argumentos y se espera que regrese una matriz asociativa.



**Las matrices asociativas: estructura de datos de Drupal de elección**

Las matrices en PHP son muy rápidos. Ellos están bien soportados, y porque sirven una doble función como ambas matrices indexadas y matrices asociativas de diccionario, son flexibles. Por estas razones Drupal hace un uso intensivo de las matrices-a menudo en lugares donde uno esperaría objetos, listas enlazadas, mapas o árboles.



La matriz devuelta debe contener una entrada para cada bloque que este módulo se declara, y la entrada debe ser de la forma $ Nombre => array ($ property => $ valor).

Por lo tanto, la parte importante de nuestra función anterior es esta pieza:

$ bloques ['list\_modules'] = array (

'Info' => t ('Una lista de todos los módulos habilitados.'), 'Cache' => DRUPAL\_NO\_CACHE,

);

**[44 ]**



*Capítulo 2*



Esto define un bloque llamado list\_modules que tiene dos propiedades:

* info: Esto proporciona una descripción de una frase de lo que hace este bloque. La texto se utiliza en las pantallas de administración bloque.
* cache: Esto le dice a Drupal cómo almacenar en caché los datos de este bloque. Aquí, en el Código he puesto esto a DRUPAL\_NO\_CACHE, Que se limitará a renunciar por completo el almacenamiento en caché. Hay varios otros ajustes que proporcionan almacenamiento en caché global, el almacenamiento en caché de cada usuario, y así sucesivamente.

Hay un puñado de otras posibilidades que Drupal reconoce. Usted puede leer acerca de esto en la documentación de la API de Drupal en http://api.drupal.org/

api/function/hook\_block\_info/7.

Ahora hemos creado una función que le dice a Drupal sobre un bloque con nombre list\_modules. Con esta información, Drupal asumirá que, cuando pide que el bloque para ver, alguna función proporcionará el contenido del bloque. La siguiente función que implementamos se encargará de mostrar el bloque.

**El hook de vista de bloque**

En el apartado anterior hemos implementado el hook que le dice a Drupal sobre el nuevo bloque de nuestro módulo. Ahora tenemos que aplicar un segundo hook, un hook responsable de construir el contenido del bloque. Este hook se llamará cada vez que Drupal

trata de mostrar el bloque.

Una implementación de hook\_block\_view () se espera que tenga un argumento: el nombre del bloque para recuperar y devolver una matriz de datos para el nombre dado.

Nuestra aplicación va a proporcionar contenido para el bloque llamado list\_modules. Aquí está el código:

/ \*\*

\* Implementa hook\_block\_view (). \* /

función first\_block\_view ($ block\_name ='') {if ($ block\_name == 'list\_modules') {

$ List = module\_list ();

$ theme\_args = array ('artículos' => $ list, 'type' => 'ol'); $ Content = tema ('item\_list', $ theme\_args);

$ Block = array (

'Subject' => t ('módulos habilitados'), 'contenido' => $ contenido,

**[45 ]**



*Crear el primer módulo*



);

return $ block;

}

}

Por ahora, el bloque doc debe ser familiar. El estilo de codificación Drupal también debería resultar familiar. Una vez más, hemos puesto en marcha hook\_block\_view () simplemente siguiendo la convención de nomenclatura.

El argumento de que nuestra first\_block\_view () función toma, es el nombre del bloque. Al mirar a través de la documentación de Drupal es posible que vea este argumento llamado $ Which\_block o $ Delta-términos destinados a identificar el hecho de que el valor pasado en es el identificador para el que el bloque debe ser devuelto.



El término $ Delta se utiliza por razones históricas. No es una descripción particularmente apto para el papel de la variable, y más recientemente ha sido reemplazado por términos más descriptivos.



El único nombre del bloque que nuestra función debe manejar es el que nos declaramos enfirst\_block\_info (). Si el $ Block\_name es list\_modules, Tenemos que devolver contenido.

Echemos un vistazo a lo que sucede cuando llega una petición para la list\_modules bloque. Este es el contenido del si declaración anterior:

$ List = module\_list ();

$ theme\_args = array ('artículos' => $ list, 'type' => 'ol'); $ Content = tema ('item\_list', $ theme\_args);

$ Block = array (

'Subject' => t ('módulos habilitados'), 'contenido' => $ contenido,

);

return $ block;

En la primera línea, llamamos a la función Drupal module\_list (). Esta función simplemente devuelve una matriz de nombres de módulos. (De hecho, en realidad es una matriz asociativa de nombres de un módulo a otro nombre. Esta asignación duplicada se hace para acelerar las búsquedas.)

Ahora tenemos una gran variedad de datos en bruto. La siguiente cosa que necesitamos hacer es formatear que para su visualización. En Drupal formato casi siempre se hace por la capa de tematización. Aquí, queremos hacer pasar los datos a la capa tema y tener que encender nuestra lista de módulos en una lista ordenada HTML.

**[46 ]**



*Capítulo 2*



Los próximos capítulos analizarán detalladamente el sistema de tematización. Por ahora, sin embargo, nos limitaremos a conceder el hecho de que cuando usamos la función de tema en la forma en que hemos hecho anteriormente, devuelve HTML con formato.



La función principal para trabajar con el sistema de tematización es tema (). En Drupal 7, tema () toma uno o dos argumentos:

* El nombre de la operación tema
* Una matriz asociativa de variables para pasar a la operación tema



Las versiones anteriores de Drupal tomaron cualquier cantidad de argumentos, dependiendo de la operación es el tema que se realiza. Eso ya no es el caso en Drupal 7. Los detalles de este se tratan en los capítulos posteriores.



Para dar formato a una matriz de cadenas en una lista de HTML, se utiliza elitem\_list tema, y ​​se pasa de una matriz asociativa que contiene dos variables:

* los elementos que queremos que aparece
* el tipo de anuncio que queremos

Desde tema () obtenemos una cadena de HTML.

Ahora todo lo que tenemos que hacer es reunir los datos que nuestra opinión bloque debe volver. Una implementación de hook\_block\_view () se espera que regrese una matriz con dos elementos en el mismo:

* sujeto: El nombre o título de la manzana.
* contenido: El contenido del bloque, en forma de texto o en formato HTML.

Así que en primer lugar nos propusimos una cadena traducible no modificable. En el segundo, nos propusimos contenido al valor construido por tema ().

Una cosa que usted puede notar acerca de la $ Bloque matriz en el código anterior es su formato:

$ Block = array (

'Subject' => t ('módulos habilitados'), 'contenido' => $ contenido,

);

Así es como matrices más grandes deben tener el formato de acuerdo con las normas de codificación de Drupal. Y que coma final no es un error. Normas Drupal requieren que las matrices de varias líneas terminan cada línea-incluyendo el último elemento-con una coma. Esto es perfectamente legal en la sintaxis de PHP, y elimina los problemas de sintaxis de codificación simples que se producen cuando se agregan elementos o eliminan el código de serie.

**[47 ]**



*Crear el primer módulo*



**No en JavaScript!**

Programadores de Drupal cometen el error de utilizar una sintaxis similar en



Drupal JavaScript. Definiciones literales de objeto (el equivalente JavaScript de matrices asociativas) no permiten el último punto para terminar con una coma. Si lo hace, provoca errores en IE y otros navegadores.

Ahora hemos caminado a través del código de nuestro primer módulo. Para todos los propósitos prácticos, hemos escrito un módulo entero (aunque todavía tenemos algo de código de pruebas automatizadas para escribir). Vamos a ver cómo se ve en el navegador.

**El primer módulo de acción**

Nuestro módulo está escrito y listo para funcionar. Para probar esto, tenemos que activar primero el módulo, y luego ir a la página de administración de bloques.

El módulo se puede activar a través del menú de Módulos. Una vez activado, vaya a

**Estructura | Bloques**. Usted debe ser capaz de encontrar un bloque descrito como una lista de todos los módulos activados. (Este texto proviene de nuestrafirst\_block\_info ()declaración.)

Una vez que haya colocado este módulo en una de las regiones de bloque, usted debería ser capaz de ver algo como esto:



**[48 ]**



*Capítulo 2*



La salida de nuestro módulo es una sencilla lista ordenada de los módulos. Al igual que cualquier otro bloque, se puede colocar en cualquiera de las regiones de bloque en el sitio, y responde en todas las formularios en que se espera que un bloque de responder.

Ahora que tenemos un módulo de trabajo, vamos a escribir un par de pruebas automatizadas para ello.

**Escribir pruebas automatizadas**

La última cosa que vamos a hacer en este capítulo es escribir pruebas automatizadas para verificar que nuestro módulo funciona como se esperaba. Una vez más, algunas de las metodologías de desarrollo piden escribir las pruebas antes de escribir código. Dicha metodología es perfectamente aplicable con módulos de Drupal. Sin embargo, hemos retrasado pruebas de escritura hasta que tuvimos un poco de código de Drupal en nuestro haber. Ahora que hemos trabajado hasta un módulo completo, estamos listos para escribir algunas pruebas.

Drupal utiliza una herramienta de prueba automatizada llamado SimpleTest (o simplemente Testing). Se deriva en gran parte de la infraestructura de pruebas Open Source SimpleTest, aunque con muchas modificaciones. SimpleTest viene con Drupal 7.

En Drupal 6, SimpleTest era un módulo add-on y parches básicos requeridos. Esto ya no es el caso en Drupal 7.

Hay varios tipos de pruebas que se pueden construir en el código. Dos de los más populares son las pruebas unitarias y pruebas funcionales.

Una prueba unitaria se centra en las pruebas de piezas discretas de código. En el código orientado a objetos, el enfoque de las pruebas unitarias es a menudo el ejercicio de todos los métodos de un objeto (o clase). En el código de procedimiento, las pruebas unitarias se centran en funciones e incluso, en ocasiones, el mundial

variables. El objetivo es simplemente para asegurarse de que cada pieza (cada unidad) está haciendo su trabajo como se esperaba.

La mayor parte de las pruebas escritas para Drupal no son pruebas unitarias. En cambio, son las pruebas funcionales. Es decir, las pruebas están diseñadas para verificar que cuando se inserta una pieza dada de código en Drupal, funciona como se esperaba en el contexto de la aplicación. Se trata de una categoría más amplia de pruebas de que las pruebas unitarias. Se espera que los trozos más grandes de código (como, por ejemplo, Drupal como un todo) para que funcione correctamente ya antes de la prueba de funcionamiento se puede medir con precisión la corrección del código está probando. Y en lugar de llamar a la función-a-ser-probada directamente, muchas veces una prueba funcional se ejecutará toda la aplicación, en condiciones que hacen que sea fácil de comprobar, si el código está probando está trabajando. Por ejemplo, las pruebas de funcionamiento de Drupal a menudo

iniciar Drupal, agregue un usuario, activar algunos módulos, a continuación, recuperar direcciones URL a través de una conexión HTTP y finalmente probar la salida.

**[49 ]**



*Crear el primer módulo*



Hay muchas fuentes excelentes de información sobre estrategias de ensayo y sus fortalezas y debilidades. Vamos a saltar cualquier discusión de este y entrar directamente en el código. Hemos de tener en cuenta a medida que avanzamos que nuestro objetivo es verificar que nuestras funciones de bloque como se esperaba. Dado que las pruebas unitarias son más fáciles de construir, y desde nuestro módulo es extremadamente simple, vamos a construir una prueba unitaria para nuestro módulo.

Mientras que el módulo de prueba se incluye con Drupal 7, no está activado por defecto. Ir a la página de Módulos y habilitarlo. Una vez activado, usted debería ser capaz de ir a la pestaña Configuración y, en la sección de Desarrollo, busca la página de configuración de pruebas. Este es el punto de entrada en la interfaz de usuario de prueba.

**Creación de una prueba**

Las pruebas deben residir en su propio archivo. Así como código del módulo principal del módulo está en

Modulename / MODULENAME.module, Una prueba debe estar en Modulename / MODULENAME.test. El marco de pruebas será automáticamente recogerlo.

**Comenzando**

Al igual que con otros archivos en un módulo, el archivo que contiene las pruebas unitarias tiene que ser declarada en la década de los módulos . Info presentar. Todo lo que necesitamos hacer es agregar a la matriz de los archivos:

; $ Id $

name = Primera

Descripción = Un primer módulo. core = 7.x

paquete = Drupal 7 Archivos de desarrollo [] = first.module archivos [] = first.test

Todo lo que hemos hecho se añade first.test debajo first.module. Esto simplemente dice

Drupal para inspeccionar el contenido de este archivo durante la ejecución. Cuando se invoca el marco de pruebas, encontrará las pruebas de forma automática mediante la inspección de los contenidos de first.test.

Una vez instalado el módulo, Drupal almacena en caché el contenido de la . Info presentar. Después de la adición de un nuevo elemento en el archivo, debe volver a visitar la página de Módulos para forzar Drupal para volver a analizar el . Info presentar.

Ahora estamos listos para añadir algo de código para first.test.

**[50 ]**



*Capítulo 2*



**Escribir un caso de prueba**

Hay algunas áreas de Drupal que hacen uso de las características orientadas a objetos de PHP. Una de ellas es la API de base de datos que vamos a ver más adelante en el libro. Otro es el framework de pruebas. Utiliza la herencia de clases para declarar pruebas. Esto es sobre todo un vestigio de laSimpleTest API sobre la que las pruebas de Drupal se basa.

Dado que este es un libro sobre programación Drupal, PHP no, no vamos a pasar el tiempo la introducción de características orientadas a objetos de PHP. Si usted no está familiarizado con

Programación orientada a objetos (POO) en PHP, es posible que desee aprender los conceptos básicos antes de pasar a esta sección. Como la mayoría de las pruebas siguen un patrón de fórmulas, no hay necesidad de dominar programación orientada a objetos antes de escribir pruebas sencillas. Sin embargo, un poco de conocimiento de fondo facilitará la transición. Un buen punto de partida es manual OOP PHP.net 's

disponible en la URL http://www.php.net/manual/en/language.oop5.php.

**El patrón básico**

La mayoría de los casos de prueba siguen un patrón simple:

* Crear una nueva clase que se extiende DrupalWebTestCase
* Añadir una getInfo () función
* Realice cualquier configuración necesaria en el setup () método
* Escribe una o más métodos de ensayo, comenzando cada método con la palabra prueba
* En cada método de prueba, utilice una o más afirmaciones para poner a prueba los valores reales

A medida que avanzamos a través de nuestras propias pruebas, vamos a caminar a través de cada uno de estos pasos

En primer lugar, vamos a empezar por la adición de una clase de prueba dentro de nuestro first.test presentar. Debe ser algo como esto:

<? Php

/ \*\*

* @ File
* Las pruebas para el primer módulo

\* /

clase FirstTestCase extiende DrupalWebTestCase {/ / Métodos irán aquí.

}

Como de costumbre, empezamos el archivo de prueba con un bloque de doc. Después de eso, declaramos nuestro nuevo caso de prueba.

**[51 ]**



*Crear el primer módulo*



Los ejemplos que se ven en este capítulo se derivan en gran parte de la bloque. prueba presentar que se incluye con el núcleo de Drupal (modules / block / block.test). Si usted está ansioso de sumergirse en algunas pruebas unitarias detalladas, es decir un solo lugar



para comenzar.

Acabamos de crear una nueva testcaso clase, es decir, una clase que se encarga de prueba de un grupo relacionado particular de características. En nuestro caso, vamos a probar la implementación de bloques que escribimos en este capítulo. Usted puede, si lo desea, crear múltiples casos de prueba en el mismo . Prueba presentar. Para nuestro caso simple, no hay necesidad de hacer esto, sin embargo.

El caso de prueba se extiende una clase base llamada DrupalWebTestCase.DrupalWebTestCase proporciona muchas utilidades para la ejecución de pruebas, así como lógica de la base de pruebas que no está necesariamente expuesto a o utilizado por casos de prueba individuales. Por estas dos razones, todas las pruebas Drupal debe extenderse bien esta clase o de otra clase que ya

se extiende DrupalWebTestCase.

Una vez que tenemos la clase declarada, podemos crear nuestro primer método, getInfo ().



**Convenciones y clases de denominación**

Funciones de Drupal son nombrados en minúsculas, con palabras separadas por guión bajo (\_) Las clases y los métodos son diferentes. Las clases deben ser nombrados en notación mayúsculas "CamelCase", con la primera letra



capitalizado. Los métodos deben ser nombrados en "camelCase" con la primera letra en minúscula. Los guiones no se deben utilizar en los nombres de clase o de método.

**El método getInfo ()**

Ya hemos visto algunos casos en Drupal utiliza matrices asociativas anidados para pasar información. Nuestrofirst\_block\_info () función hizo justamente esto. La DrupalWebTestCase :: getInfo () método también devuelve una matriz de información. Esta vez, la información es acerca de la prueba.

El método se parece a esto (que se muestra en el contexto de toda la clase)

<? Php

/ \*\*

* @ File
* Las pruebas para el primer módulo

\* /

clase FirstTestCase extiende DrupalWebTestCase {

**[52 ]**



*Capítulo 2*



función pública getInfo () {return array (

'Name' => 'funcionalidad bloque Primer módulo', 'description' => 'bloques de prueba en el primer módulo.', 'Grupo' => 'Primero',

);

}

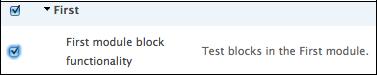
}

La getInfo () método devuelve una matriz con tres elementos:

* nombre: El nombre de la prueba.
* descripción: Una frase que describe lo que las pruebas lo hacen.
* grupo: El nombre del grupo al que pertenecen estas pruebas.

Los tres de ellos están destinados a ser legible. Los dos primeros se utilizan con fines puramente informativos. El tercero, grupo, También se utiliza para agrupar las pruebas similares juntos en la misma partida.

Cuando se ve desde Configuración | Pruebas, la información anterior se muestra como esto:



**Limpie el medio ambiente**



Si ya ha ejecutado las pruebas y su nueva prueba no aparece, es posible que tenga que pulsar el botón de Medio ambiente limpio para reiniciar el entorno de prueba.

Más arriba, Usted puede ver cómo el valor de grupo se convirtió en un campo de agrupación, y nombre y descripción se han utilizado para describir la prueba.

La getInfo () función podría parecer, a primera vista, que no tienen importancia, pero su prueba, absolutamente debe tenerlo. De lo contrario, no se pondrá a disposición para la ejecución del caso de prueba.

**[53 ]**



*Crear el primer módulo*



**Configuración del caso de prueba**

A menudo, un caso de prueba será necesario algún ajuste y configuración, donde se inicializan los valores compartidos y subsistemas disponibles.

Afortunadamente, Drupal se encarga de la mayor parte de los conceptos básicos. La capa de base de datos, sistema de módulos y la configuración inicial se puede hacer todo. Sin embargo, los casos de prueba a menudo tienen que manejar algunos de inicialización sí mismos. En los casos en que hay que hacer esto, no hay un método existente que se llamará antes de ejecutar las pruebas. Este es elsetup () método. Si bien no es necesario realizar ninguna configuración para nuestro módulo, voy a mostrar todos modos

por lo que podemos ver algunas cosas importantes.

<? Php

/ \*\*

* @ File
* Las pruebas para el primer módulo

\* /

clase FirstTestCase extiende DrupalWebTestCase {

public function setup () {parent :: setup ('first');

}

función pública getInfo () {return array (

'Name' => 'funcionalidad bloque Primer módulo', 'description' => 'bloques de prueba en el primer módulo.', 'Grupo' => 'Primero',

);

}

Una vez más, un método de instalación no es estrictamente necesario, pero cuando se utiliza uno debe tener por lo menos las líneas que se muestran en el ejemplo anterior.

De particular importancia es este bit:

parent :: setup ('first');

Esto le indica al método de configuración para llamar al setup () método que existe en el

DrupalWebTestCase clase. ¿Por qué haríamos eso?

**[54 ]**



*Capítulo 2*



DrupalWebTestCase :: setup () realiza algunas operaciones de configuración, cosas necesarias que deben hacerse antes de nuestras pruebas se ejecutará correctamente. Por esa razón, tenemos que asegurarnos de que cuando se sobrepase el método, de forma explícita llamamos. Transmitimos el conjunto de esta función, el nombre del módulo que estamos probando (primero), De manera que conozca para inicializar ese módulo para nosotros. Esto significa que no es necesario que preocuparse de instalar el módulo en nuestro código de prueba.

Al escribir sus propios casos, puede agregar más líneas de código de configuración bajo la parent :: setup () llamar. Más adelante en el libro, verá ejemplos más sólidos de los métodos de configuración.

Por ahora, sin embargo, vamos a pasar a la siguiente tipo de método. Vamos a escribir nuestra primera prueba.

**Escribir un método de prueba**

La mayoría de los métodos en un caso de prueba son los métodos de prueba; es decir, se ejecutan operaciones con el propósito de verificar que funcionen. Pero, como te darás cuenta, en ninguna parte de nuestro código Cómo llamamos explícitamente esos métodos de ensayo.

Entonces, ¿cómo SimpleTest saber para llamar a nuestros métodos? Al igual que con hooks de Drupal, la respuesta está en la convención de nomenclatura. Cualquier método que comienza con la palabra prueba se supone que es un caso de prueba, y se ejecuta automáticamente por el framework de pruebas.

Vamos a escribir dos métodos de prueba, una vez más se muestran en el contexto de toda la clase.

<? Php

/ \*\*

* @ File
* Las pruebas para el primer módulo

\* /

clase FirstTestCase extiende DrupalWebTestCase {

public function setup () {parent :: setup ();

}

función pública getInfo () {return array (

'Name' => 'funcionalidad bloque Primer módulo', 'description' => 'bloques de prueba en el primer módulo.', 'Grupo' => 'Primero',

**[55 ]**



*Crear el primer módulo*



);

}

**función pública testBlockInfo () {**

**$ Info = module\_invoke ('primero', 'block\_info');**

**$ This-> assertEqual (1, count ($ info), t ('Módulo define un bloque.'));**

**$ This-> assertTrue (isset ($ info ['list\_modules']), t ('existe la lista de módulos.'));**

**}**

**función pública testBlockView () {**

**$ Data = module\_invoke ('primero', 'block\_view', 'list\_modules');**

**$ This-> assertTrue (is\_array (datos $), t ('Block devuelve array renderizable.'));**

**$ This-> assertEqual (t ('módulos habilitados'), $ data ['subject'], t ('Asunto se establece'));**

**}**

}

El código anterior tiene dos métodos de ensayo:

* testBlockInfo ()
* testBlockView ()

Como su nombre indica, cada método es responsable de probar una de las dos funciones de bloque que escribimos antes.

Vamos a empezar por echar un vistazo de cerca a testBlockInfo ().

función pública testBlockInfo () {

$ Info = module\_invoke ('primero', 'block\_info');

$ This-> assertEqual (1, count ($ info), t ('Módulo define un bloque.'));

$ This-> assertTrue (isset ($ info ['list\_modules']), t ('existe la lista de módulos.'));

}

**[56 ]**



*Capítulo 2*



Esta función hace tres cosas.

En primer lugar, se ejecuta una función llamada module\_invoke (), El almacenamiento de sus resultados en $ Info. La module\_invoke () función llama un hook especial para un módulo en particular.



Esta función es la contraparte de uso poco frecuente de module\_invoke\_all (), Que ejecuta un hook en toda de los módulos en los que aparece ese hook.



La module\_invoke () método toma dos parámetros: el nombre del módulo y el nombre del hook para llamar. La convocatoria de este código, module\_invoke ('primero',

'Block\_info') es semánticamente equivalente a llamar first\_block\_info (). Nuestra única ventaja obtenida aquí es para asegurar que se puede llamar a través del sistema de hook.

Básicamente, entonces, hemos simulado las circunstancias en que nuestra información hook habría sido ejecutados por Drupal. Lo siguiente que debe hacer es asegurarse de que la información devuelta por nuestro hook es como se esperaba.

Hacemos esto al hacer un par de declaraciones a las afirmaciones, acerca de lo que esperamos. El marco de pruebas después valida estas excepciones. Si el código funciona como se espera, la prueba pasa. Si no, la prueba falla.

Aquí están las dos pruebas:

$ This-> assertEqual (1, count ($ info), t ('Módulo define un bloque.'));

$ This-> assertTrue (isset ($ info ['list\_modules']), t ('existe la lista de módulos.'));

(Tenga en cuenta que cada una de estas dos líneas se dividieron en una sola línea para el formato.)

Cada afirmación es típica de la forma $ This-> assertSOMETHING ($ condiciones, $ Mensaje), En donde ALGO es un tipo de afirmación, $ condiciones son las condiciones que deben ser satisfechas para que la prueba pase, y $ Mensaje es un mensaje que describe la prueba.

En nuestra primera prueba, la prueba afirma que 1 y count ($ info) debe ser igual. (El mensaje se usa simplemente por la interfaz de la prueba para demostrar lo que estaba poniendo a prueba.)

**[57 ]**



*Crear el primer módulo*



Usted puede notar que comenzó la función $ This-> assertEqual () el cual es un método miembro, pero que no hemos definido. ($ This, Para los nuevos en programación orientada a objetos de PHP, es una forma abreviada de referirse a la actual objeto.) La clase padre, DrupalWebTestCase, Proporciona una docena de métodos de aseveración que hacen que la escritura pruebas más fácil. Muchos de ellos se van a plantear en los siguientes capítulos, pero en nuestras pruebas utilizamos dos:

* $ This-> assertEqual (): Afirmar que el primero (conocido) es igual al valor segundos (probado) de valor.
* $ This-> assertTrue (): Afirman que el valor dado se evalúa como VERDADERO.

Mientras que la primera afirmación valida que hemos definido un bloque en nuestra aplicación hook bloque de información, la segunda afirmación comprueba que el nombre de este bloque es list\_modules. Por lo tanto, por el momento esta prueba se ha ejecutado, podemos estar seguros de que nuestra información hook regresa información acerca de nuestro bloque a bloque.

La siguiente prueba verifica que el first\_block\_view () la función devuelve la información correcta.

función pública testBlockView () {

$ Data = module\_invoke ('primero', 'block\_view', 'list\_modules');

$ This-> assertTrue (is\_array (datos $), t ('Block devuelve array renderizable.'));

$ This-> assertEqual (t ('módulos habilitados'), $ data ['subject'], t ('Asunto se establece'));

}

Una vez más, module\_invoke () se utiliza para ejecutar una vez en circuito esta aplicación hook vista de bloque. Y otra vez llevamos a cabo dos afirmaciones. En primer lugar, comprobamos que asegurarse de que una colección se devuelve desde first\_block\_view (). En segundo lugar, se verifica que el título está habilitado módulos, ya que esperamos.

Podríamos seguir y añadir otra afirmación, algo que se asegura de que el$ data ['content'] campo tiene los datos esperados en el mismo. Pero esa información es un poco volátil. No está confirmado, sobre el cual se habilitarán otros módulos, y las pruebas en contra de que se inyectan una dependencia externa en nuestra prueba, que se considera de mala educación.

En este punto, hemos definido un caso de prueba, FirstTestCase, Que define cuatro métodos. Dos de estos métodos son las pruebas, cada uno con dos afirmaciones. Así que cuando se corre la prueba, deberíamos ver un caso de prueba, dos pruebas, y dos afirmaciones para cada prueba.

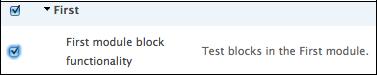
**[58 ]**



*Capítulo 2*

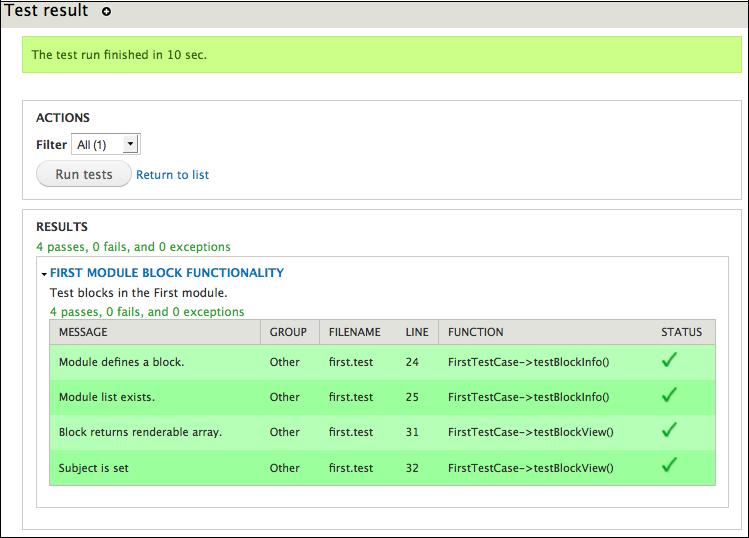


Para ejecutar la prueba, vaya a Configuración | Pruebas. Siempre y cuando su caso de prueba se lleva a cabo correctamente (incluyendo el getInfo () método), entonces debe aparecer en la lista.



Si seleccionamos nuestro grupo de pruebas y, a continuación, pulsamos el botón Ejecutar pruebas, se ejecutará nuestro caso de prueba. Los casos de prueba a menudo tardan mucho tiempo en ejecutarse. Detrás de las escenas, Drupal

en realidad construye una instalación especial de Drupal que se utilizará sólo para esta ronda de pruebas. Pero después de un minuto o dos, el marco de pruebas debe imprimir un informe que se ve algo como esto:



**[59 ]**



*Crear el primer módulo*



El informe anterior nos muestra que los cuatro de nuestras afirmaciones estaban a cargo (dos para cada prueba), y que todo el pasado.

En caso de que una prueba no pasa, éste se mostrará en rojo, con el indicador de estado para una X roja en lugar de una marca de verificación verde. Un mensaje de aviso puede aparecer, también (dependiendo del error o el fracaso).

**Resumen**

Hemos completado una caminata de extremo a extremo a través de la creación de un módulo. Comenzamos creando el directorio de módulos, seguido por el. Info presentar. A continuación, hemos añadido un

. Módulo presentar e implementado tres hooks, aprovechando varios núcleo de Drupal funciones en el proceso. Finalmente, escribimos nuestra primera prueba de este módulo, el aprendizaje sobre marco de pruebas OO de Drupal a medida que avanzábamos.

En el camino, nos enteramos de las directrices básicas de codificación, apoyo a la traducción, la mecánica de hooks, y el uso de la API de bloque.

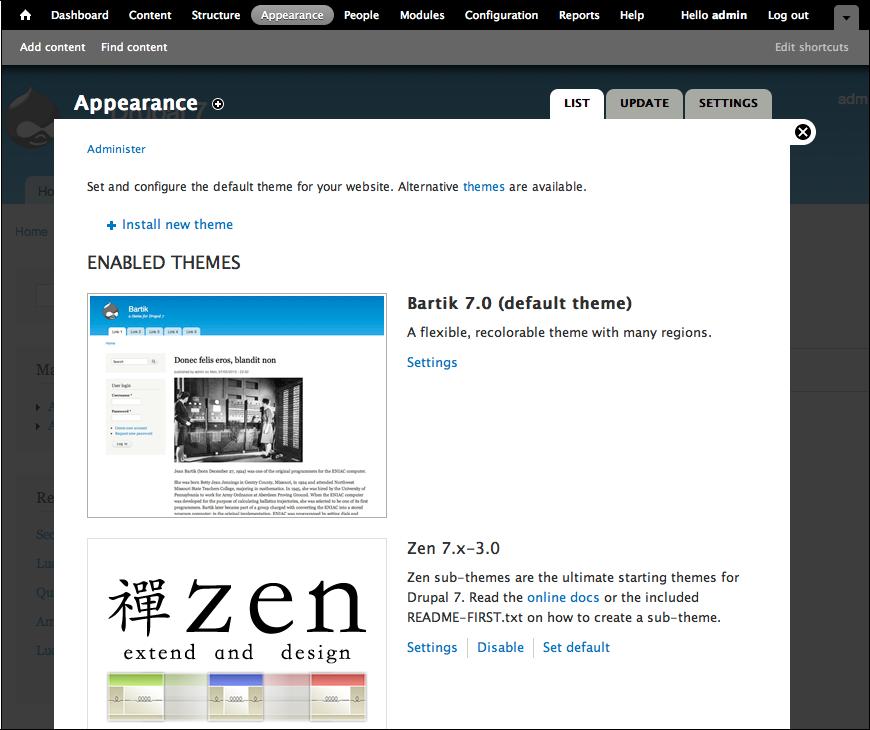
En los capítulos siguientes, vamos a aprovechar este conocimiento para crear módulos más potentes, haciendo uso de la capa de base de datos, el sistema de menús, linfáticos y otras herramientas. En los próximos capítulos, vamos a ver el tema del sistema, un mecanismo poderoso y extensible para la estructuración y la salida de formato.

**[60 ]**



Tema Capa de Drupal

La parte más obvia del sistema de tematización de Drupal es la página de Apariencia, que enumera todos los temas instalados en su sitio web. Cuando usted elige un tema de la página Apariencia admin, usted está solicitando una grafía particular a los datos y la funcionalidad de su sitio web. Sin embargo, el tema aplicado es en realidad sólo una pequeña parte de toda la capa de decoración temática.



*Tema Capa de Drupal*



Este libro se centra principalmente en la construcción de módulos que encapsulan trozos discretos de funcionalidad. Sin embargo, ya que estamos en última instancia, la creación de una aplicación web, tendrá todo emitida por su funcionalidad a ser marcado con HTML. Drupal requiere el proceso de envolver sus datos en HTML y CSS como tematización.

Durante los siguientes dos capítulos, vamos a discutir cómo el módulo debe integrarse con la capa de tema. Capítulo 3 hablará de la arquitectura del sistema, funciones de temas, plantillas, render elementos, y el registro temático. Capítulo 4 utilizará estos conceptos recién adquiridos para integrar un módulo de ejemplo con la capa de tematización.

**La lógica de negocio frente a la lógica de presentación**

Entonces, ¿cuál sería la mejor manera de obtener nuestros datos y la funcionalidad de marcado? ¿Estamos simplemente envuelva cada pieza de datos en HTML y volvemos el todo como una cadena gigante? Al igual que el siguiente ejemplo:

volver '<div class="wrapper">'. $ datos. '</ Div>';

Afortunadamente, no lo hacemos. Al igual que todas las demás aplicaciones bien diseñadas, Drupal separa la lógica de negocio de su lógica de presentación. Tradicionalmente, las motivaciones principales para esta separación de las preocupaciones son las siguientes:

1. Para hacer que el código sea más fácil de mantener.
2. Para que sea posible cambiar fácilmente la aplicación de una capa sin tener que volver a escribir las otras capas.

Como veremos, Drupal tiene el aspecto de "canje de capacidad" hasta el extremo.

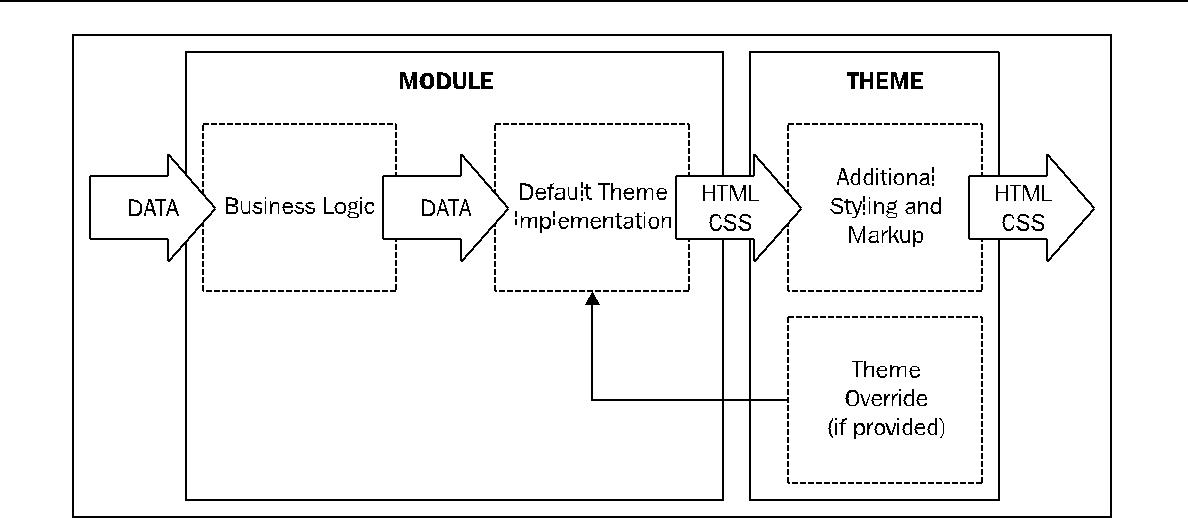
Como mencionamos en la introducción de este capítulo, el tema por defecto seleccionado en la página Apariencia es la parte más obvia de la capa de tema. Además, se podría pensar que el tema es responsable de aplicar el código HTML y CSS para el sitio web. Sin embargo, hay miles de módulos de terceros endrupal.org. En caso de que el tema sea responsable del marcado de seguridad de todos los datos de los módulos "? Obviamente no.

Desde un módulo es más íntimamente familiarizado con sus propios datos y la funcionalidad, es la responsabilidad del módulo para proporcionar la implementación tema por defecto. Siempre que el módulo utiliza el sistema el tema correctamente, un tema central será capaz de anular todo el código HTML y CSS por su propia aplicación para la implementación del módulo de intercambio de calor.

**[62 ]**



*Capítulo 3*



Después de los datos se ha recuperado y manipulado en el corazón de su módulo (la lógica de negocio), tendrá que proporcionar la implementación tema por defecto. A veces, un tema en particular tendrá que anular su aplicación a fin de que para lograr un objetivo de diseño específico; si el tema proporciona su propia implementación, Drupal utilizará la implementación tema en lugar de implementación por defecto del módulo.

Cuando la construcción de nuestro primer módulo en el capítulo 2, vimos un breve ejemplo de esto en la acción de la siguiente manera:

variables $ = array ('artículos' => $ list, 'type' => 'ol'); $ Content = tema ('item\_list', $ variables);

Al llamar a la tema () función, estamos delegando la responsabilidad de determinar y utilizar la aplicación tema adecuado. Estamos diciendo:

"Hey, tema ()! Quiero markup mis datos como un item\_list. ¿Puedes hacer eso por mí? No necesito saber los detalles. kthxbye ".

Nuestro módulo sólo tiene que decidir qué tema hook que quiere utilizar para el marcado de sus datos. Si los datos se mostrarán en una lista desordenada, una mesa o un wordle?

Enganche loco?



Además de API **hooks**, Drupal también tiene **hooks temáticos**. Un tema

hook es simplemente el *nombre* de una manera particular para el marcado de algunos datos. Por ejemplo, pasar datos a la item\_list hook tema dará lugar a diferentes marcas y luego pasar los datos a la Enlaces hook tema. Sin embargo, aunque normalmente función de enlace de cada módulo se llamará cuando Drupal invoca un hook de API, sólo una aplicación hook tema será invocado cuando Drupal invoca un hook tema.



**[63 ]**



*Tema Capa de Drupal*



En realidad, hay dos maneras diferentes que usted puede hacer una aplicación (que veremos más adelante), pero por ahora sólo vamos a hablar sobre el método más simple para las funciones de los desarrolladores temáticas de los módulos. Cuando se llama atema (), Se buscará una

función de tema predeterminado denominado theme\_HOOKNAME y una función de tema anulación opcional llamado THEMENAME\_HOOKNAME. Si se mira en partes internas de Drupal, encontrará una theme\_item\_list () dentro includes.inc o theme.inc. Esta es la implementación tema por defecto de Drupal para un item\_list. Si nuestro tema activo era Bartik, y si Bartik implementa una invalidación tema llamado bartik\_item\_list (), Luegotema ()utilizaría la implementación del tema Bartik en lugar de la predeterminada.

La figura anterior muestra una pieza de datos a medida que pasa a través de un módulo y un tema. Sin embargo, para que usted entienda todo el poder de capa tema de Drupal, también es necesario entender cómo se construye toda la página.

Sin embargo, ya que todas las modificaciones del tema que está activo se producen después de las modificaciones del módulo, desde la perspectiva de un desarrollador de módulos, toda esta herencia tema es transparente. Dado que los módulos no necesitan saber nada acerca de la estructura del tema y su ascendencia, vamos a simplemente hablar de "el tema" en este libro. Sólo ten en cuenta que el tema real puede ser más compleja.



**Base temas y subtemas**

Si usted ha leído algo sobre Drupal tematización previamente, usted probablemente ha oído hablar de **temas básicos** y **subtemas**. Cualquier tema puede

declarar un tema principal en su . Info presentar con la tecla de tema base y heredará todas las implementaciones de hook de su tema dominante (y el tema de los padres de su padre, y así sucesivamente).



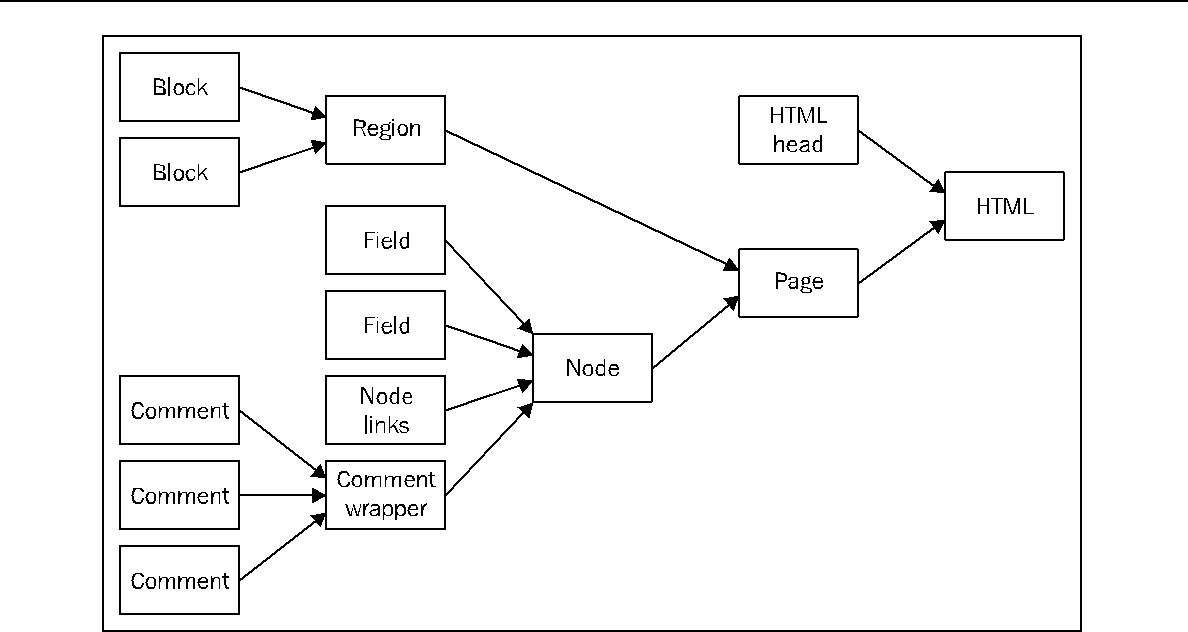
**Granularidad de datos**

Una de las cosas que hace que Drupal tematización tan poderosa es su granularidad. Cada pieza de contenido es manejado por separado como se pasa a través del sistema de tematización. Cada bit de datos está decorado con motivos y se combinan después en trozos cada vez más grandes. En cada paso en el proceso de agregación, dedicada al tema de nuevo. En la ilustración siguiente hará esto más claro:

**[64 ]**



*Capítulo 3*



Como se puede ver en la ilustración anterior, para un puesto típico blog, cada comentario se extrae de la base de datos y se envía a través del sistema el tema para obtener el marcado HTML agregado a él. A continuación, todos los comentarios son agregados juntos en un "comentario envoltorio", donde por lo general, se agrega un "comentario" forma marcado adicional y,. A continuación, el grupo único de los comentarios se pasa a la tematización nodo donde se combina con otros elementos de contenido del blog. Este proceso de bits de la tematización de contenidos, la agregación y la tematización de nuevo se repite hasta que hemos construido toda la página HTML listos para ser enviados a un navegador web.

Hay dos ventajas de este sistema granular. En primer lugar, ya que cada módulo es responsable de la tematización de sus propios datos, o bien puede crear un hook tema muy especializado para sus datos o se puede volver a utilizar un hook de tema existente. La reutilización de un hook tema asegura un conjunto consistente de marcado para las estructuras de datos similares, mientras que todavía permite que las clases CSS personalizados (La mayoría de los hooks temáticos permiten a las clases personalizadas que se pasan

como parámetros.) Por ejemplo, la lista de enlaces después de un nodo (leer más, añadir nuevo comentario, y así sucesivamente) reutiliza elEnlaceshook tema, y ​​los vínculos después de cada comentario utilizan el mismoEnlaces hook tema.

La segunda ventaja es para el desarrollador tema. Tener un sistema de tematización de grano fino significa que un tema, si se opta por, literalmente, puede volver a escribir todo el margen de beneficio para sus propios propósitos de diseño. Como desarrolladores de módulos que tenemos que ser muy conscientes del deseo de la themer tener anulaciones tematización granulares.

**[65 ]**



*Tema Capa de Drupal*



**Motores temáticas**

Algunos temas requieren motores temáticos alternativos. Motores temáticas pueden proporcionar sintaxis alternativa plantilla, las normas de denominación y funciones de ayuda. Varios motores temáticos están disponibles para su descarga enhttp://drupal.org/project/theme+engines. Sin embargo, no vamos a discutir cualquier tema, excepto motores para el motor de tema por defecto de Drupal, PHPTemplate. El motor del tema PHPTemplate ha sido el tema por defecto desde Drupal 4.7, se ha mejorado continuamente con cada versión, y ha demostrado su valía una y otra vez. Más del 99% de los temas disponibles para su descarga en drupal.org utilice el motor de temas PHPTemplate defecto. Todos los ejemplos de este libro asumen que está utilizando PHPTemplate. Así que, dicho lo suficiente.

**Dos maneras de tema**

Así que ahora que tenemos una buena comprensión de los conceptos de mayor nivel, vamos a llegar hasta el meollo de la cuestión de las implementaciones de tema. Como se mencionó anteriormente en este capítulo, en realidad hay dos formularios diferentes de implementar un hook tema:

* **Funciones temáticas**: Pasar datos a una función PHP para envolverlo en el marcado
* **Plantillas**: Pasar datos a una plantilla que es un archivo PHP mezclado con declaraciones de marcado y de impresión PHP

Echemos un vistazo a cada uno de ellos a su vez.

**Funciones temáticas**

Para un desarrollador de módulos, el tipo más fácil de entender es la implementación de una función de tema. Funciones temáticas sólo tiene que seguir algunas reglas simples a fin de que funcionen correctamente.

En primer lugar, el nombre de la función tema sigue el patrón:

theme\_ [nombre hook tema]



Ya que el nombre del hook tema se utiliza directamente en el nombre de la función de tema, los nombres de hook tema tienen las mismas restricciones en dar nombres como regulares de función PHP; los únicos caracteres válidos en los nombres de hook temáticos son caracteres alfanuméricos y guiones bajos. Así que si un módulo ha creado una example\_format hook tema, sería implementarlo con función de tema llamado theme\_example\_format ().



**[66 ]**



*Capítulo 3*



En segundo lugar, la función de tema sólo tendrá un único parámetro, como sigue:

función theme\_THEME\_HOOK\_NAME (variables $) {...}

Las variables de la función son tema de una matriz asociativa que contiene los fragmentos de datos que deseamos para el marcado y las opciones que queremos pasar a la función. Puede parecer muy extraño no utilizar múltiples parámetros y la capacidad de PHP para especificar valores predeterminados para cada parámetro. De hecho, las versiones anteriores de Drupal hicieron uso de múltiples parámetros. Ya veremos por qué Drupal ahora sólo utiliza un parámetro en un momento, cuando hablamos de las funciones de preproceso.

Para un ejemplo de un variables $ array, echemos un vistazo a cómo el DocBlock del theme\_item\_list () función define:

* **Artículos**: Un conjunto de elementos que se mostrarán en la lista. Si un elemento es una cadena, entonces se utiliza tal cual. Si un elemento es una matriz, entonces el elemento de "datos" de la matriz se utiliza como el contenido del elemento de la lista. Si un elemento es una matriz con un elemento de "niños", los niños se muestran en una lista anidada. Todos los demás elementos son tratados como atributos del elemento elemento de la lista.
* **Título**: El título de la lista.
* **Tipo**: El tipo de lista para volver (por ejemplo,ul,ol).
* **Atributos**: Los atributos aplicados al elemento de la lista.

La artículos y título claves contienen los datos reales, y la tipo y atributos claves son opciones que especifican cómo construir la lista de elementos.

En tercer lugar, la función de tema debería devolver una cadena que contiene la representación de los datos prestados. Esto suele ser una cadena de HTML, pero algunos hooks temáticos devuelve otros tipos de marcado temática. Por ejemplo,theme\_syslog\_format devuelve una cadena sencilla, con valores de datos de tubos separados para su uso en un \* NIX syslog registro de errores.

Eso es todo! Como puede ver, las funciones temáticos tienen requisitos muy simples y en todo lo demás son funciones estándar de PHP.

La principal diferencia entre la mayoría de funciones y funciones de temas es que nunca se debe llamar directamente a funciones temáticos. Puede ser tentador para tomar sus datos y call theme\_item\_list ($ vars)directamente, sino que en su lugar debe llamar tema ("item\_list", $ vars). Este método de llamar a funciones temáticos asegura indirectamente que los temas son capaz de anular la función de tema por defecto de cualquier módulo (o plantilla). También permite que el tema () funcionar para hacer magia adicional, incluyendo el permitir que otros módulos se alteran las variables de la función de tema antes de que sean utilizados.

**[67 ]**



*Tema Capa de Drupal*



**Funciones de preproceso**

Ahora estamos empezando a ver la verdadera flexibilidad del sistema el tema. Funciones de preproceso permiten un módulo para modificar las variables utilizadas por otro módulo cuando llama a un hook de tema. Así que si algún código pasa datos a tema () para un hook tema en particular, las funciones de preproceso serán llamados para alterar los datos antes de llamar a la implementación real de hook tema. Los siguientes pasos se llevan a cabo:

1. Llamadas Código tema ('hook\_name', las variables $).
2. tema () llamadas funciones de preproceso para hook\_name.
3. Funciones de preproceso modifican variables.
4. tema () pide la aplicación efectiva de hook\_name con variables modificadas.

Todas las funciones de preproceso adoptan la forma de:

[Módulo] \_preprocess\_ [nombre del tema hook] (y variables $)

Así que si el foo módulo quiere alterar las variables para el item\_list hook de tema, se podría definir la función como sigue:

foo\_preprocess\_item\_list función (y variables $) {/ / Añadir una clase a la lista de envoltura. $ variables ['atributos'] ['clase'] [] = 'foo-lista';

}

Tenga en cuenta que lavariables $ parámetro se define con un signo en frente de ella. Eso es PHP notación para pasar el parámetro por referencia. En lugar de obtener una copia de las variables, el foo\_preprocess\_item\_list () función tendrá acceso a la real variables $ que más tarde se pasa a la implementación de la función tema. Por lo que cualquier modificación que la función hace al preproceso variables $ parámetro se conserva cuando se pasan las variables de la función de tema. Esa es la razón de nuestro ejemplo foo\_preprocess\_item\_list () función no devuelve nada; su labor se realiza directamente en el original variables $.

Esto es extremadamente útil para desarrolladores de módulos, ya que permite todo tipo de integración con otros módulos. Dado que el parámetro de variables es una mezcla de datos y opciones, los módulos pueden alterar tanto los datos brutos y cambian la manera quedarán sin datos. Esto puede ser tan simple como un módulo que necesitan una clase especial para el uso en su código JavaScript y agregó que la clase al contenido temático de otro módulo añadiendo a la$ Var iables ['atributos'] ['clase'] matriz, o puede haber interacciones más complejas como la i18n módulo de traducir el lenguaje utilizado en bloques.

Imaginemos que hemos construido un módulo retro que integra GeoCities y queremos reemplazar todos los enlaces a la página de perfil de un usuario con un enlace a GeoCities página de inicio del usuario. Podemos hacerlo de manera relativamente fácil con una función de procesamiento previo.

**[68 ]**



*Capítulo 3*



Primero echemos un vistazo a la siguiente theme\_username la documentación de la función:

/ \*\*

* Dar formato a un nombre de usuario.
* @ param variables $
* Una matriz asociativa que contiene:
* - Cuenta: El objeto de usuario en formato.
* - Nombre: El nombre del usuario, desinfectados.
* - Extra: Texto adicional para anexar el nombre del usuario, desinfectados.
* **- Link\_path: La ruta o URL de la página del perfil del usuario, su domicilio**
* **página o otras páginas que desee para ligarse a para más información**
* **sobre el usuario.**
* - link\_options: una gama de opciones para pasar al l () la función de
* $ Parámetro opciones si vinculando el nombre del usuario para el usuario de
* página.
* - Attributes\_array: Un conjunto de atributos para pasar a la
* función drupal\_attributes () si no se vincula a la página del usuario.

\* /

Muy convenientemente, theme\_username () tiene un práctico $ Link\_path variable que queremos modificar para lograr nuestras risitas de la vieja escuela. Suponiendo que hemos utilizado alguna otra lógica de negocio con los hooks del módulo de usuario para cargar nuestra URL GeoCities en la cuenta del usuario (la parte "dura"), en sustitución del enlace a la página de perfil del usuario se puede lograr con la siguiente función de preproceso simple:

/ \*\*

\* Implementa genialidad con hook\_preprocess\_username (). \* /

función retro\_preprocess\_username (variables y $) {$ variables ['link\_path'] = $ variables ['cuenta'] -> geocities\_url;

}

Eso es todo! Nosotros no tenemos que anular la implementación tema del módulo de usuario; acabamos de modificar sus parámetros.

**Anulaciones temáticas**

Mientras que los desarrolladores de módulos por lo general no tienen que preocuparse de si un tema se impone a una función de tema en particular o no, sigue siendo importante para entender cómo funciona este mecanismo.

**[69 ]**



*Tema Capa de Drupal*



Un tema de Drupal se compone normalmente de estilo CSS, imágenes, JavaScript, archivos de plantilla

(Discutido en breve), una . Info archivo, y un template.php presentar. La template.php archivo es análogo a un módulo de . Módulo presentar. Contiene todas las funciones de PHP para el tema y se carga automáticamente cuando se inicializa el tema.

Si un tema quiere reemplazar una función tema en particular, tiene que copiar la función de tema de su ubicación original y pegarlo en sutemplate.php presentar. Entonces hay que cambiar el prefijo de la función de tema a su propio nombre y, finalmente, tiene que empezar a hacer los cambios que desee en la función.

Por ejemplo, si el tema Bartik quiere anular la theme\_menu\_local\_tasks () función con el fin de añadir un poco de margen de beneficio en torno a las pestañas de la página, sería copiar toda la función de includes / menu.inc, Péguelo en Bartik de template.php, Y cambiarle el nombre a bartik\_menu\_local\_tasks ().

Afortunadamente, cuando un tema se impone a una función de tema por defecto, las funciones de procesamiento previo de un módulo continúan trabajando de manera normal.

Los temas también tienen la capacidad de crear funciones de preproceso. Si el tema Bartik decide dar formato a nombre de un usuario en el "apellido, nombre" formato, se puede implementar un bartik\_preprocess\_username () función. Afortunadamente, las funciones de procesamiento previo de un tema no anulan las funciones de procesamiento previo de un módulo. Todas las funciones de procesamiento previo se ejecutan; primero las funciones de procesamiento previo de cualquier módulo y función de procesamiento previo del tema.

**Los archivos de plantilla**

Mientras que las funciones del tema podría ser el más fácil para los desarrolladores de módulos de entender, archivos de plantilla son los más fáciles de themers a captar. Cuando un hook tema se implementa con archivos de plantilla, que se utilizan en lugar de las funciones de tema. Sin embargo, desde el punto de vista de un desarrollador de módulos, en realidad hay una notable cantidad de similitud entre los archivos de plantillas y funciones temáticos. En primer lugar, echemos un vistazo a los archivos de plantilla.

Las plantillas son archivos que contienen principalmente HTML, pero con algunas declaraciones de PHP mezclados en el uso de las variables de la plantilla. En lugar de declarar un theme\_hook\_name () función, un módulo podría en lugar de crear un hook name.tpl.php presentar. Los siguientes son los contenidos de un archivo de plantilla típica, típica-hook.tpl.php:

<div class = "? <clases php print $;?>" <atributos php print $?; ? >>

<? Php if ($ title):>

<H2 <php print $ title\_attributes?; ? >> <Php print $ título?; ?>

</ H2>

<? Php endif;?>

**[70 ]**



*Capítulo 3*



<div class="submitted">

<? Php print t ('By! Autor @ hace tiempo', array ('time @' => $ time,

'! Autor' => $ autor)); ?>

</ Div>

<Div class = "contenido" <php print $ content\_attributes?; ? >> <? Php

/ / Nos escondemos los enlaces ahora para que podamos hacer más adelante. ocultar ($ contenido ['links']);

imprimir render ($ contenido);

?>

</ Div>

? <Php print render ($ contenido ['links']); > </ Div>

El ejemplo anterior muestra la gama completa de las cosas que usted está probablemente ve en un archivo de plantilla. Son los siguientes:

* Impresión de una variable que contiene una cadena
* Impresión de una cadena traducible usando t ()
* Condicional si/más/endif declaración
* El retraso de la prestación por parte de un hacer elemento con hide ()
* Impresión de una hacer elemento

Todo el PHP en una plantilla debe limitarse a imprimir variables. Esta cantidad limitada de PHP hace que sea mucho más fácil para los no programadores a aprender cómo utilizar los archivos de plantilla en comparación con las funciones del tema. Sin embargo, para los desarrolladores de módulos, la aplicación plantilla es todavía muy similar a la implementación de la función tema; el puñado de diferencias son relativamente menores.

Al igual que con las implementaciones de función del tema, nuestro módulo todavía tendría que invocar el hook tema usandotema ().

variables $ = array ('típica' => $ typical\_object); $ Salida = theme ('typical\_hook', las variables $);

**[71 ]**



*Tema Capa de Drupal*



La tema () función sería descubrir que el typical\_hook hook tema fue implementado como una plantilla y hacer el correspondiente típica-hook.tpl.php presentar.



Como hemos mencionado anteriormente en este capítulo, los únicos caracteres válidos en los nombres de hook temáticos son caracteres alfanuméricos y guiones bajos. Esto es cierto para todos los hooks del tema, independientemente de que se implementan como una función de tema o como un archivo de plantilla. Sin embargo, cuandotema () busca la plantilla



implementaciones, se convertirá automáticamente cualquier subrayado en el nombre del hook tema en guiones en la búsqueda del archivo de la plantilla. Por ejemplo, llamar tema ('user\_picture', variables $) se traducirá en el archivo de plantilla con nombre usuario-picture.tpl.php que se queden.

También, al igual que funciones de temas, otros módulos pueden modificar las variables que utilizan funciones de preproceso.

En los archivos de plantilla de la atención se centra en imprimir las variables en varios lugares en el marcado. Así que para los archivos de plantilla, la función de preproceso adquiere un papel más importante. La única diferencia entre las funciones de procesamiento previo de una función tema y un archivo de plantilla son el número y tipo de funciones de procesamiento previo.

**El zoológico preproceso**

Cuando se escribe una función de tema, su natural para pasar los datos en bruto como parámetros y generar ningún meta-datos relacionados con la pantalla dentro de la función. Con un archivo de plantilla, que no es realmente posible sin poner PHP compleja dentro de la plantilla. Sin embargo, como se indicó anteriormente, todo el PHP en un archivo de plantilla debe limitarse a sólo el mínimo necesario para imprimir una variable de PHP. Todo tratamiento que tenemos que hacer en los parámetros de datos en bruto para facilitar en las variables listos para imprimir se debe hacer en las funciones de preproceso.

**funciones de preproceso "TEMPLATE\_"**

Cuando un módulo define un hook tema mediante la creación de un archivo de plantilla, ese módulo también debe crear una función de procesamiento previo correspondiente para configurar y procesar las variables que son necesarios por el archivo de la plantilla, pero no se pasan como parámetros a tema (). Por convención, esa función puede procesar previamente debe ser de la siguiente forma:

template\_preprocess\_ [tema] (nombre del hook & $ variables)

La TEMPLATE\_ prefijo indica el tema del sistema de Drupal que esta función preproceso es el preprocesador primaria para las variables del hook tema y se debe ejecutar antes de la función de preproceso de cualquier otro módulo.

**[72 ]**



*Capítulo 3*



He aquí un ejemplo que debería hacer este concepto un poco más claro. Este es un fragmento de código real de la función de bloque de preprocesamiento de Drupal. En cada región la página, todos los bloques de la región obtener una variable cuyo valor se alterna entre "extraña" y "hasta". Estos valores se pueden utilizar para crear un estilo-rayas de cebra, es decir, un estilo alternativo de cada dos cuadras en una región.

función template\_preprocess\_block (& ​​$ variables) {

* Almacenamos todos los contadores del bloque utilizando drupal\_static (). $ Block\_counter = & drupal\_static (\_\_FUNCTION\_\_, array ());
* Todos los bloques de conseguir un contador independiente para cada región. if (isset ($ block\_counter! [$ variables ['bloque'] -> región])) {

$ Block\_counter [$ variables ['bloque'] -> región] = 1;

}

* Generar la variable rayas de cebra.

**$ variables ['block\_zebra'] = ($ block\_counter [$ variables ['bloque'] -> región]% 2)? 'Extraño': 'incluso';**

/ / Incrementa el contador de bloque de la región. $ Block\_counter [$ variables ['bloque'] -> región] + +;

}

La lógica de PHP en esta función está directamente relacionada con la pantalla del bloque y no a la lógica de negocio general de estos datos. Por lo tanto, no tiene sentido que el módulo de bloque calcularía que los metadatos antes de llamartema (); los metadatos pertenece claramente a la lógica de visualización, que es por eso que se coloca en función de preproceso del módulo de bloque.

**Funciones de preproceso multi-hook**

En algunas raras ocasiones, es posible que tenga que modificar o proporcionar algunas variables para todos los hooks temáticos. De hecho, el tema del sistema de Drupal sí proporciona algunas variables a todos

plantillas; la función de preproceso que proporciona estas variables es a la vez una función de preproceso "TEMPLATE\_" y una función de preproceso multi-hook. Funciones de preproceso múltiples hooks son simplemente funciones que no tienen un \_HOOK sufijo añadido a su nombre y se ejecutan para cada archivo de plantilla única. Su nombre es de la siguiente forma:

[Módulo] \_preprocess (& $ variables $ hook)

Obviamente, no puede haber un gran impacto en el rendimiento si un módulo innecesariamente implementa la función de preproceso multi-hook. Si estás pensando en escribir uno, si es posible, que no hacer varias funciones de preproceso que se dirigen a los hooks específicos que necesita en su lugar, en lugar de golpear a todos los hooks.

**[73 ]**



*Tema Capa de Drupal*



Ahora, si está pagando mucha atención a la forma del nombre, usted también notará que estas funciones realmente reciben dos parámetros, a saber, la variables $ matriz y un $ Hook parámetro. $ Hook, Como su nombre indica, contiene el nombre del hook tema real que se está ejecutando. Así, mientras que unfoo\_preprocess (variables y $, $ Hook) la función se ejecuta para cada archivo de plantilla, todavía será capaz de decir qué se solicita la plantilla actualmente. De hecho, $ Hook es el segundo parámetro para todas las funciones de preproceso, pero $ Hook sólo es útil para las funciones de preproceso multi-hook.

Para un buen ejemplo de una función de preproceso multi-hook, echemos un vistazo a la función que el sistema de tema utiliza para configurar varias variables comunes a todos los archivos de la plantilla-la template\_preprocess () función, que es la siguiente:

función template\_preprocess (variables y $, $ hook) {

* Recomendar a todas las plantillas donde se ubican. $ variables ['directorio'] = path\_to\_theme ();
* Inicialice atributo de clase HTML para el hook actual.

**variables $ ['classes\_array'] = array (drupal\_html\_class ($ hook));**

}

Como puede ver, esta función crea un preproceso $ Directorio variable que se puede utilizar para saber dónde se encuentra el archivo de plantilla en el servidor web. En el $ Classes\_array variable, también comienza a fraguar las clases CSS utilizados en la más exterior de envolver div de la plantilla.

**Funciones de proceso**

Obviamente, dentro de nuestro archivo de la plantilla, cuando imprimimos nuestra lista creada de forma dinámica de las clases, vamos a necesitar la variable a ser una cadena. ? <Php print $ classes\_array; ?> será, imprimir más inútilmente a cabo "array". En versiones anteriores de Drupal, clases eran

creado de forma dinámica, pero fueron inmediatamente concatenado en cadenas. Así temas verían una cadena larga con múltiples clases en el mismo,menu-block-wrapper menú-block-1 -gestión de menús nombre, Por ejemplo. Esto hizo que la eliminación o la alteración de las clases difíciles como themers tenían que dominar las funciones de manipulación de cadenas de PHP o incluso (¡oh!)

*expresiones regulares*.

En Drupal 7, este problema para los diseñadores de temas ha sido resuelto utilizando las nuevas funciones de proceso. Funciones de proceso son una fase adicional de funciones de procesamiento de variables que se ejecutan después de las funciones iniciales de preproceso. En todos los aspectos, las funciones del proceso son exactamente como las funciones de procesamiento previo; hay TEMPLATE\_ funciones con el prefijo de proceso, las funciones del proceso multi-hook, funciones de proceso de módulos proporcionados, y las funciones del proceso temáticos proporcionados. La única diferencia es que las funciones de proceso se ejecutan después de todas las funciones de procesamiento previo se han ejecutado.

**[74 ]**



*Capítulo 3*



Funciones de proceso son extremadamente útiles cuando se tiene metadatos que pueda ser manipulado por otros módulos o temas y desea retrasar la prestación de los metadatos hasta justo antes de que el archivo de plantilla se representa.

En el ejemplo de código anterior, el template\_preprocess () función crea un $ Classes\_array variable que contiene un conjunto de clases para ser utilizado en el envoltorio div en el archivo de plantilla. Los módulos y los temas pueden agregar fácilmente las clases con sólo añadir un elemento de matriz adicional desde el interior de su función de preproceso, de la siguiente manera:

$ variables ['classes\_array'] [] = 'extra-savoir-faire';

Los temas se pueden utilizar las funciones de manipulación de matrices más simples con el fin de eliminar o alterar las clases.

* Búsqueda de la clase falsa y devolver su matriz key
* ubicación. Si no lo encuentra, array\_search devuelve FALSO.
* Recuerde que 0 es una clave válida.

$ Key = array\_search ('falsos', $ variables ['classes\_array']); if ($ key! == false) {

/ / Modificar la clase. $ variables ['classes\_array'] [$ clave] = 'dude'.;

}

/ / O quite la clase sin sopa.

$ variables ['classes\_array'] = array\_diff ($ variables ['classes\_array'], array ('no-sopa'));

Además de la $ Classes\_array variable, el template\_preprocess () función crea también $ Attributes\_array,$ Title\_attributes\_array, Y $ Content\_attributes\_array las variables que se utilizan para HTML atributos de la envoltura más externa div, Etiqueta de encabezado del título, y la envoltura del contenido div, Respectivamente. Verá cada una de estas variables utilizadas en el típica-hook.tpl.php ejemplo, dada anteriormente en este capítulo.

Después de módulos y temas se les da la oportunidad de modificar estas variables, el sistema utiliza el tema template\_process () funcionar para hacer que esas matrices en una cadena sencilla, de la siguiente manera:

template\_process función (& $ variables $ hook) {/ / aplanan clases.

$ variables ['clases'] = implosión ('', $ variables ['classes\_array']);

/ / Aplanar atributos, title\_attributes y content\_attributes. $ variables ['atributos'] = drupal\_attributes (

$ variables ['attributes\_array']); $ variables ['title\_attributes'] = drupal\_attributes (

$ variables ['title\_attributes\_array']); $ variables ['content\_attributes'] = drupal\_attributes (

$ variables ['content\_attributes\_array']);

}

**[75 ]**



*Tema Capa de Drupal*



Un problema similar se turbó desarrolladores de módulos en Drupal 6. Era imposible para llamar drupal\_add\_css () o drupal\_add\_js () en un MODULE\_preprocess\_page () función debido a que las listas de archivos CSS y los archivos JavaScript que ya se generaron antes de cualquiera de las funciones de preproceso se corrieron. Una vez más, las funciones del proceso vienen al rescate. Drupal 7 retrasa la generación de estas listas hasta que el template\_process\_html () la función se ejecuta.

**Orden de ejecución de preproceso**

Ahora, con todos estos diferentes sabores de las funciones de procesamiento, que puede ser un poco confuso en cuanto a qué función se ejecuta en qué orden. Afortunadamente, sólo hay tres reglas simples que se utilizan para determinar el orden de procesamiento. Son los siguientes:

* Todas las funciones de procesamiento previo funcionar antes de todas las funciones de proceso
* TEMPLATE\_ prefijo funciones se ejecutan primero. [Módulo] \_ prefijo funciones se ejecutan siguiente. [Tema] \_ prefijo funciones corren última
* Funciones de múltiples funciones de hook se ejecutan antes de hook específico

Esto se traduce en el siguiente orden de ejecución de un hook tema en particular:

1. template\_preprocess ()
2. template\_preprocesss\_HOOK ()
3. MODULE\_preprocess ()
4. MODULE\_preprocess\_HOOK ()
5. THEME\_preprocess ()
6. THEME\_preprocess\_HOOK ()
7. template\_process ()
8. template\_processs\_HOOK ()
9. MODULE\_process ()
10. MODULE\_process\_HOOK ()
11. THEME\_process ()
12. THEME\_process\_HOOK ()

*¡Menos mal.*



Si el tema es en realidad una lista de la base hereditaria y los subtemas, cada

Theme\_Elemento prefijado anteriormente podría ser una lista de cada base y el tema de las funciones del sub-tema, lo que haría que la lista aún más larga. Vea la "Base temas y subtemas" punta cerca del comienzo de este capítulo si no lo ha leído ya.



**[76 ]**



*Capítulo 3*



Por cierto, ¿su cerebro duele todavía? Es posible que desee tomar un descanso ahora; salir y conseguir un poco de aire, o, al menos, tomar una copa fuerte práctico cuando usted comience a leer la siguiente sección.

**Render elementos**

Representan los elementos son nuevos a la capa tema de Drupal 7. Han existido desde Drupal 4.7 como parte de la API de forma, pero ahora han sido inyectadas en el corazón del sistema es el tema. Un elemento Render es una estructura de datos compleja que se pasa como parámetro único para tema (), Como una de sus variables. Render elementos son fundamentalmente las matrices que pueden incluir anidados:

* Los datos que se van a representar
* Otro render elementos que se consideran "hijos" del elemento
* Matriz de estructuras tales como archivos CSS y JavaScript, que deberá adjuntarse a la página cuando se representa
* Una lista de los hooks tema que se puede utilizar para el tema de los datos
* Una lista de las funciones de callback a ejecutarse en el elemento antes y después de su temática

En los archivos de plantilla, representan los elementos se manejan de forma ligeramente diferente a continuación, las variables normales, usando la sintaxis que vimos anteriormente en nuestro típica-hook.tpl.php ejemplo:

? <Php print render ($ element); ?>

En funciones de temas, representan los elementos se incluyen con su salida mediante el drupal\_render () función:

. $ Salida = drupal\_render ($ element);

Veamos un sencillo hacer elemento:

$ Element = array (

'# Prefijo' => 'class="plain"> <div',

**'# Markup' => '<p>'. t ('No hay cuchara.'). '</ P>',**

'# Sufijo' => '</ div>',

);

En el elemento anterior render nuestra principal propiedad es la # Markup propiedad, que utiliza una cadena que contiene el formato HTML como está para el elemento representado. Las otras propiedades hacen exactamente lo que insinúan, anteponiendo o de agregar el formato HTML para el elemento representado. Si drupal\_render ($ element) fue llamado, sería simplemente devuelva las tres cadenas concatenadas.

**[77 ]**



*Tema Capa de Drupal*



Ahora, eso fue un ejemplo muy simple, pero cuando empezamos a mirar más complejos se procesan elementos, veremos que cada clave de la matriz en un elemento de procesamiento puede ser una de las siguientes tres cosas:

1. Una representación propiedad del elemento. Estos tienen el prefijo #.
2. Un elemento secundario. Todas las claves de matriz no el prefijo # se considera que un niño elementos.
3. Una variable que se pasa a una función de tema. En el elemento de render nombres de estas variables tienen el prefijo # (Al igual que las propiedades son), pero tema () se tira de la # desde el nombre antes de enviarlo a la aplicación tema real.

Tomando estas reglas ligeramente los champiñones, podemos examinar el siguiente elemento de render:

$ Element = array (

'# Prefijo' => '<div class="less-simple">', '# sufijo' => '</ div>',

**'Gatito' => array (**

'Type #' => 'link', '# title' => t ('Mátame'),

'# Href' => 'admin / core / hackeo',

),

**'Separador' => array (**

'# Markup' => '<br />',

),

**'Domo' => array (**

'# Lema' => 'nombre de usuario', '# cuenta' => $ cuenta,

),

);

En primer lugar, debemos identificar a los niños, ya que son los más sencillos de detectar. gatito,separador, Y domo son los elementos secundarios de nuestro render elemento. La separador elemento secundario es otro ejemplo de un sencillo # Markup render elemento.

En cuanto a ladomo elemento, vemos que su # Tema propiedad se establece como nombre de usuario. drupal\_render () tendrá que elemento secundario y pasarlo a tema () con un tema

hook de nombre de usuario; lo que significa que tema ('username', $ element ['domo']) será llamado y tema () se tira de la # personajes de la parte delantera de todas las variables antes de pasar los datos a theme\_username ().

**[78 ]**



*Capítulo 3*



Por último, la gatito elemento de # Tipo propiedad se establece en enlace. La # Tipo propiedad indica drupal\_render () cómo hacer que ese elemento. Cuando aprendemos acerca hook\_element\_info (), Vamos a entender por qué, pero por ahora drupal\_render () pasará a la gatito elemento a la drupal\_pre\_render\_link () función que hará que el elemento utilizando L () y devolverá su salida.

**Render propiedades**

Render propiedades de los elementos se definen en dos lugares. El primer lugar donde se definen las propiedades es directamente en el interior drupal\_render () y sus funciones de ayuda. La siguiente es una lista completa de las propiedades utilizadas por drupal\_render ():

* # Access: Un valor booleano que indica si el usuario actual tiene acceso a la vista el elemento.
* # Caché: Matriz que indica si el elemento opcional recuperar de la caché o almacenada en caché después de la representación. Verdrupal\_render () para más información.
* # Markup: Una cadena que contiene marcado (como HTML). Si esta propiedad es establecido, la # Tipo propiedad no es necesario establecer, ya que se configurará automáticamente para el marcado.
* # Tipo: Cadena que indica que se está representando elemento. El valor por defecto propiedades de este tipo de elemento se extraen de los datos especificados con hook\_element\_info () y se fusionó con el elemento de render.
* # Defaults\_loaded: Un valor booleano que indica si el defecto del tipo de elemento propiedades ya han sido cargados. Si este es falso o no se establece, las propiedades predeterminadas de element\_info () se añaden antes drupal\_render () mira a cualquier otro render propiedades (excepto para # Access y # Caché).
* # Pre\_render: Una serie de devoluciones de llamada a aplicar al elemento antes de tematización.
* # Tema: Una cadena que especifica el hook es el tema de ser utilizado en el elemento.
* # theme\_wrappers: Una serie de hooks temáticos para ser utilizado en el elemento después tematización inicial y / o después de los elementos secundarios se han rendido. Funciones temáticas que son para ser utilizados como envoltorios necesitan ser especialmente escrito para buscar la Menores propiedad en el elemento de render pasó a ella desde el tema.
* # Post\_render: Una serie de devoluciones de llamada a aplicar al elemento después de la tematización.
* Menores: El elemento prestados y sus hijos. Normalmente se construye internamente por drupal\_render () ya que hace que los elementos, pero también se puede ajustar por una # Pre\_render devolución de llamada.
* Prefijo #: Una cadena que contiene el marcado que se antepone a la

Menores propiedad.

**[79 ]**



*Tema Capa de Drupal*



* # Sufijo: Una cadena que contiene el marcado que se adjunta a la

Menores propiedad.

* # Peso: Un número que se utiliza para ordenar los elementos secundarios.
* # Ordenados: Un valor booleano que indica si los elementos secundarios ya han sido ordenados. Dado que la clasificación de un render matriz es caro, no sé si los datos ya está ordenado (por ejemplo, los datos se solucionó cuando se recuperan de la base de datos), debe establecer esta propiedad VERDADERO.
* # estados: Información sobre el estado de JavaScript.
* # Adjunta: Una serie de CSS, JavaScript, bibliotecas u otros asociados anexos relacionados con el elemento. Ver drupal\_process\_attached () para más información.
* # Impresa: Un valor booleano que indica si el elemento ya se ha rendido.

**hook\_element\_info**

El segundo lugar se definen las propiedades que hay dentro hook\_element\_info (). Cada # Tipo de hacer elemento debe ser definido en una implementación de

hook\_element\_info (). system\_element\_info () define la mayor parte del núcleo de Drupal representan los elementos, que incluyen varios elementos útiles tales como lamarcado elemento, la enlace elemento, y todos los forma elementos. El siguiente es un breve fragmento de

system\_element\_info ():

/ \*\*

\* Implementa hook\_element\_info (). \* /

system\_element\_info function () {/ / código HTML. $ tipos ['marcado'] = array (

'# Markup' =>'',

'# Pre\_render' => array ('drupal\_pre\_render\_markup'),

);

/ / Un vínculo HTML. $ tipos ['enlace'] = array (

'# Pre\_render' => array ('drupal\_pre\_render\_link', 'drupal\_pre\_render\_markup'),

);

/ / Un elemento de formulario oculto. $ tipos ['ocultos'] = array (

'# Input' => TRUE,

'Proceso #' => array ('ajax\_process\_form'), '# lema' => 'oculto',

);

return $ tipos;

}

**[80 ]**



*Capítulo 3*



Como se puede ver, el enlace tipo especifica que el elemento de render se debe pasar a dos # Pre\_render funciones. Y es el drupal\_pre\_render\_link () función que busca los render propiedades especiales de los elementos en nuestro ejemplo de elemento de enlace, es decir, # Título,# Href, Y Opciones #.

Así que reiterar, hook\_element\_info () define las propiedades predeterminadas para sus render tipos de elementos, y también especifica hacen devoluciones de llamada que tienen su propia API interna, definiendo render propiedades de los elementos.

El uso de este marco, los módulos pueden crear su propio complejo de hacer elemento mediante la implementación dehook\_element\_info (), Utilizando las propiedades especificadas por drupal\_render (), Y mediante la creación de cualquier render devoluciones de llamada y las API asociadas.

**hook\_page\_alter ()**

Entonces, ¿cuál es el punto? Con la creación de estos complejos se procesan los elementos, nos retrasamos representación de los datos y permitir oportunidades para alterar los datos antes de representarlo en una cadena. Antes de rendir elementos fueron utilizados en el sistema el tema de Drupal, themers y desarrolladores de módulos a menudo tenían que completamente re-procesar los datos después de haber sido dictada la forma predeterminada. Esto era obviamente ineficiente. Ahora cada uno de ellos representan los elementos se pueden modificar en funciones de preproceso o incluso directamente en un archivo de plantilla con el show () y hide () funciones.

Ahora que hemos visto las entrañas de la API Render, se hace mucho más fácil entender cómo incrustado en plantilla del hide () La opción funciona. Si llama a un archivo de plantilla hide ($ element ['niño']); simplemente establece el # Impresa propiedad a VERDADERO, Así que cuando imprimir render ($ element); más tarde se llama, el elemento secundario no se imprime.

A continuación, puede llamar más tarde imprimir render ($ element ['niño']); y render () establecerá # Impresa a FALSO y pasar $ Element a drupal\_render ().

Implementaciones tema de Drupal utilizan representar elementos en varios lugares a lo largo de sus hooks temáticos. Pero los dos lugares primarios representan los elementos acostumbrarse están en los hooks de bloque y de la página del tema.

Cualquier hook\_block\_view () aplicación debe devolver un elemento renderizable, y cualquier devolución de llamada del menú que suministra contenido principal de una página también debe devolver un elemento de render.

Una vez que se recupera el contenido principal de la página, drupal\_render\_page () voluntad

decorar el elemento $ página utilizando hook\_page\_build (). Durante el bloqueo del módulo block\_page\_build (), Todas las regiones de la página se añaden a la $ Page como elemento elementos secundarios; y cada uno de los elementos secundarios de la región contienen elementos secundarios para cada uno de los bloques en esa región. drupal\_render\_page () luego permitir que los módulos para modificar el gigante $ Page render elemento utilizando hook\_page\_alter ().

**[81 ]**



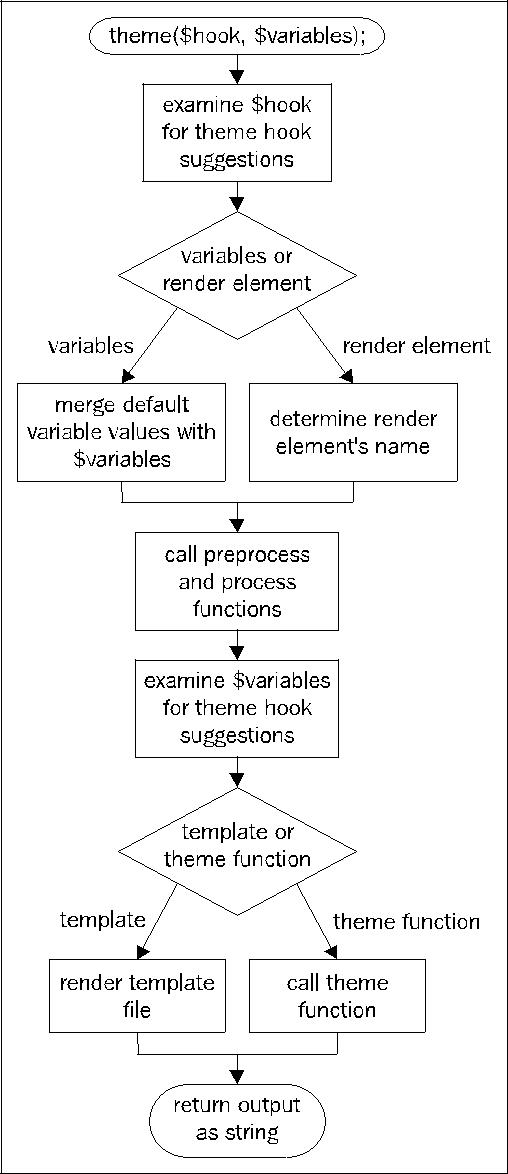
*Tema Capa de Drupal*



Dos casos de uso de gran alcance para hook\_page\_alter () sería la de permitir la inserción de un bloque dentro de la página principal de la página, o hacer lo contrario, mover un "campo" en un determinado lugar en una región página. Por supuesto, usted tendrá que leer el capítulo API campo (Capítulo 7, crear nuevos campos) primero!

**El poder de tema ()**

Resulta que la tema () función tiene que hacer un poco de trabajo una vez que se llama. El siguiente diagrama debe tomar sus responsabilidades y su orden de las operaciones más claras:



**[82 ]**



*Capítulo 3*



Hemos hecho ya discutimos la mayor parte del flujo de trabajo de tema (). Sólo hay un aspecto que aún no hemos visto. Hasta ahora, sólo hemos llamado tema () con una cadena simple pasado a su $ Hook parámetro. Sin embargo, en realidad podemos pasar los datos más complejos a la misma y hacer uso de las sugerencias de hook el tema del sistema de tema.

**Sugerencias hook Tema**

Así que volver a usar hooks temáticos en diversos lugares de nuestro código es una buena cosa, por supuesto. Sin embargo, un problema que vas a encontrar es que las funciones temáticos pierden el contexto cuando se reutiliza un hook tema. Por ejemplo, theme\_links () no tiene idea de si se trata de la tematización enlaces nodos o vínculos de comentarios, lo que hace difícil de peinar de forma diferente.

Afortunadamente, podemos proporcionar un contexto para el tema del sistema, proporcionando un patrón hook tema como primer parámetro:

[Hook de base] \_\_ [contexto]

Las partes del patrón se separan con un doble subrayado, ya que algunos hooks temáticos (como usuario y user\_profile) Podría ser confuso si tuviéramos que utilizar un solo guión para delimitar las partes del patrón. De hecho, podemos ofrecer contextos adicionales si es necesario, de la siguiente manera:

[Hook de base] \_\_ [contexto] \_\_ [incluso más contexto] \_\_ [No te vuelvas loco]

Entonces, ¿cómo funciona esto y cómo ayuda? En Drupal 7, nos temáticos enlaces del nodo llamando tema ('links\_\_node', $ vars). Así tema () utilizará un

theme\_links\_\_node () funcionar si se ha proporcionado. Sin embargo, si uno no lo hace existir, se utilizará theme\_links (). Esto nos permite conocer el contexto basado en la función de tema que estamos implementando. Un ejemplo más complejo es cuando Drupal 7 temas enlaces contextuales del nodo; que llama tema ('links\_\_contextual\_\_node', $ vars). Así, tema () buscará una theme\_links\_\_contextual\_\_node (), Luego para theme\_links\_\_contextual (), Y por último theme\_links (), Acortando el patrón en una unidad de contexto cada vez.

El patrón hook tema es un método fácil de usar para proveer contexto, pero algunos módulos contribuido necesita proporcionar las listas más complejas de contexto que el patrón de cadena sencilla puede proporcionar. Por esta razón, una gran variedad de posibles sugerencias de hook también se puede pasar a tema (). Por ejemplo, los dos puntos de vista y el módulo de bloque de menú utilizan este método. Mientras tematización árboles de menús, bloque Menu proporciona la siguiente matriz para el tema:

$ Hook = array ('menu\_tree\_\_menu\_block\_\_'. $ Delta,

'Menu\_tree\_\_menu\_block\_\_'. $ Nombre\_menú, 'menu\_tree\_\_menu\_block',

**[83 ]**



*Tema Capa de Drupal*



'Menu\_tree\_\_'. $ Nombre\_menú, 'menu\_tree',

);

tema ($ hook, $ árbol);

Esto permite que los diseñadores de temas para proporcionar una THEME\_menu\_tree\_\_menu\_block\_\_1 () función para anular el despliegue de sólo el primero configurado bloque de menú (con un ID delta 1) O para anular la visualización de todos los árboles que muestran el menú de gestión con un THEME\_menu\_tree\_\_menu\_block\_\_management (). Por supuesto, si el tema proporciona ninguna de esas funciones, tema () seguirá utilizando la implementación predeterminada, theme\_menu\_tree (). Para ser claro acerca de cómo funciona esto, tema () toma la serie de sugerencias y los cheques de izquierda a derecha hasta que encuentra una implementación real.

Si nos fijamos en la figura anterior de nuevo, te darás cuenta de que después de las funciones de preproceso y de procesos han llevado a cabo, tema () examina las variables $ para sugerencias hook tema adicionales. Por supuesto, sólo el código de llamada puede especificar sugerencias de hook en el primer parámetro a tema (). Con el fin de permitir que otros módulos y temas para agregar sus propias sugerencias, tema () examinará dos variables $ Theme\_hook\_suggestion y $ theme\_hook\_suggestions.tema () comprueba en primer lugar la forma singular de ese nombre de la variable. Si el $ Theme\_hook\_suggestion cadena no coincide con una aplicación real, tema () comprobará el $ theme\_hook\_suggestions matriz a partir de

de izquierda a derecha para encontrar una implementación. Si tema () no encuentra ninguna implementación de esas dos variables, se seguirá utilizando el hook tema que previamente había estado utilizando.

Dado que los temas pueden anular por completo el valor de $ Theme\_hook\_suggestion, Se recomienda que los módulos se adhieren a la modificación de la $ theme\_hook\_suggestions array. Así es como se hace:

* Añadir nuestras sugerencias al frente de la lista, ya que nuestro
* módulo es el más importante en el universo!

array\_unshift ($ variables ['theme\_hook\_suggestions'], 'links\_\_retro\_\_geocities', 'links\_\_retro');

Tenga en cuenta que $ theme\_hook\_suggestions y $ Theme\_hook\_suggestion no tome patrones. Si desea tema () buscar links\_\_retro\_\_geocities y links\_\_retro, Usted tendrá que proporcionar esas dos cadenas en el $ theme\_hook\_suggestions array.

Una última nota, todos los ejemplos anteriores asumió funciones de temas, pero lo mismo es cierto para las sugerencias de hook el tema de los archivos de plantilla. Si proporciona un node\_\_blog\_\_1 patrón a tema (), Se buscará un nodo - blog - 1.tpl.php presentar y luego para una node - blog.tpl.php presentar.

**[84 ]**



*Capítulo 3*



**Registro de Tema**

Así que todas las cosas maravillosas que tema () no, tomar una cantidad considerable de trabajo. Es extremadamente ineficiente para tema () para determinar toda esta información sobre hooks temáticos sobre la marcha cada vez que se llama. Así que para ayudar a su carga y mejorar el rendimiento, Drupal utiliza un registro temático.

El registro es el tema contiene una lista de todos los hooks temáticos conocidos por el sistema, y ​​para cada hook es el tema, también almacena la siguiente:

* Si se trata de una función de tema o una plantilla
* La lista de funciones de procesamiento previo para ejecutar
* La lista de funciones de proceso para ejecutar
* La ruta de acceso al archivo de plantilla (que incluye si el archivo de plantilla de módulo original se usa, o una versión del mismo tema.)
* El nombre de la función es el tema (que indica si se trata de la función de tema módulo original o una anulada por un tema.)
* La lista de variables que deben ser pasados ​​a tema () cuando este tema se llama hook y un valor predeterminado para cada variable
* Si, en lugar de una lista de variables, la función de un solo tema de espera rendir elemento como parámetro, y lo que hacen que el elemento debe ser nombrado

Si bien este registro el tema hace mejorar el rendimiento para el usuario del sitio web, sí causa cierta incomodidad para módulos y diseños desarrolladores. Desde Drupal está almacenando datos sobre los hooks del tema, si está escribiendo o alterar activamente un hook tema, tendrá que asegurarse de que usted vuelva a generar el registro tema antes de probar los cambios. Afortunadamente, esto puede fácilmente logrado haciendo clic en el botón Borrar todas las memorias caché en la página de Rendimiento que se encuentra en el administrador de configuración (admin / config / desarrollo / rendimiento). La desarrollo módulo también tiene un práctico **Reconstruir el** **Registro del tema en cada carga de página** opción en su configuración.

**Valores predeterminados de variables**

Antes, cuando hablábamos de las funciones temáticas y el parámetro de una sola variable $, te habrás dado cuenta de un corto venida de ese parámetro. Cuando usted tiene una lista normal de los parámetros de una definición de función, especificar los valores predeterminados para los parámetros que son opcionales. Esto permite que el código sólo para enumerar algunos parámetros cuando se llama a la función y tienen los parámetros adicionales acaba de obtener los valores por defecto razonables.

**[85 ]**



*Tema Capa de Drupal*



Si una función se define como sigue:

función foo\_spaghettify ($ code, $ type = 'fina', $ pegajosa = TRUE) {}

A continuación, el código que llama simplemente podría llamar foo\_spaghettify ($ code) y el $ Tipo y $ Pegajosa parámetros obtendrían los valores por defecto definidos en la definición de la función.

Con un solo variables $ parámetro que tenemos un problema. Si probamos el siguiente:

función foo\_spaghettify (variables $ = array ('código' => array (), 'tipo' => 'delgada', 'pegajosa' => TRUE) {}

Y luego llamó:

variables $ = array ('código' => $ code); $ Resultado = foo\_spaghettify ($ variables);

Entonces, descubriríamos que $ variables ['type'] y variables $ ['pegajosa'] no reciben el valor por defecto, y ellos también no están definidos. La razón es simple; valores de los parámetros por defecto sólo se acostumbran, si no se especifica el parámetro cuando se llama a la función, pero no especificaron el variables $ parámetro, por lo que no se utiliza el valor predeterminado.

Entonces, ¿cómo podemos resolver este problema en Drupal? Tenemos módulos que definen las variables predeterminadas en su hook\_theme funciones.

**hook\_theme**

De un módulo hook\_theme es el principal responsable de la especificación de un par de cosas, de la siguiente manera:

* El tema de hooks que el módulo es responsable de
* El tipo de ejecución tema (función de tema o plantilla)
* Valores de las variables predeterminadas Los hooks temáticos '
* Si el hook espera que un elemento de render en lugar de variables

Además de las principales responsabilidades, la hook\_theme Opcionalmente, puede especificar:

* ¿Qué archivo contiene la función de tema o funciones de preproceso (si no está en el archivo principal del módulo)
* Un modelo para su uso durante la búsqueda de descubrimiento automático de sugerencias hook tema anulados de un tema
* Algunas otras cosas esotéricas se pueden leer en su documentación: http://api.drupal.org/api/function/hook\_theme/7

**[86 ]**



*Capítulo 3*



De un módulo hook\_theme aplicación sólo tiene que devolver un conjunto de hooks tema que está creando. Por ejemplo:

/ \*\*

\* Implementa hook\_theme (). \* /

función retro\_theme () {return array (

**/ / La clave de la matriz es el nombre del hook tema.**

'Maravilla' => array (

* Utilice una plantilla y dar el nombre de la plantilla. 'Plantilla' => 'maravilla',
* Especifique los nombres de las variables y sus valores por defecto. "variables" => array (

'Twin' => 'zen', 'with\_monkey' => FALSE, 'activaciones' => array (),

),

* Añadir un patrón parcial para ayudar tema hook sugerencia
* partidos.

'Patrón' => 'wonder\_\_',

),

'Austinite' => array (

* Especifique el nombre del elemento de render. 'Render elemento' => 'my\_element',
* No utilizamos esta función el tema a menudo, así que vamos a poner
* en un archivo independiente que es cargado ligeramente, si es necesario. 'Archivo' => 'retro.texas.inc',

),

);

}

Vamos a examinar estos datos. Por defecto se asumirá un hook tema para ser implementado como una función de tema llamadotheme\_hook\_name (). Podemos alterar temporalmente el nombre del hook mediante un función clave / valor dentro de nuestra austinite matriz, pero por favor no confundir a nuestros themers pobres; dejar el nombre de la función por defecto solo. Si un hook tema en vez quiera utilizar la aplicación de plantilla, es necesario especificar el nombre del archivo de plantilla (sin el . Tpl.php extensión) en el plantilla clave / valor. Nuestro maravilla hook tema utilizará un wonder.tpl.php plantilla.

**[87 ]**



*Tema Capa de Drupal*



Nosotros también hemos especificado un patrón para ese hook. Tras el registro tema recupera la información de nuestro retro\_theme () función, que tratará de detectar automáticamente cualquier archivo de plantilla cuyos nombres comienzan con la patrón que especifiquemos. Así que el tema de registro buscará cualquier plantilla que comienza con wonder\_\_ y añadirá una entrada para ella en el registro temático. ¿Cuándo tema () se pasa sugerencias hook temáticos, que no busca en el directorio de archivos, busca el registro tema en su lugar, por lo que este patrón es esencial para hacer sugerencias de hook tema de trabajo.

Por último en nuestra maravilla definición, hemos especificado varias variables con el "variablesTecla '/ valor establecido, es decir, gemelo (Una cadena), with\_monkey (Un booleano), y activaciones (Una matriz), y les dio a cada uno un valor predeterminado.

En lugar de una lista de variables, la theme\_austinite () función espera un solo elemento de render como parámetro, por lo que es necesario especificar el nombre del elemento de render. Vamos a ser capaces de acceder al elemento de procesamiento de theme\_austinite () desde $ variables ['my\_element']. En el ejemplo anterior, hemos utilizado my\_element a hacen evidente que el valor especificado en render elemento es la clave que necesita ser utilizado en variables $ para tener acceso a ese elemento. Sin embargo, por convención, el núcleo de Drupal utiliza normalmente elemento o elementos para ese valor.

Para ser claros, un hook tema debe utilizar ya sea 'Render elemento'O "variables"En su conjunto de datos. Tenga en cuenta que incluso si se utiliza "variables", Una de las variables que todavía puede ser un elemento de representación.

Finalmente, se utilizó una expediente clave para especificar que estamos almacenando nuestra theme\_austinite () función en un retro.texas.inc presentar. tema () Cargará perezoso este archivo si tiene que utilizar este hook tema en una página determinada.

La documentación completa se puede encontrar en el DocBlock para la hook\_theme () API en el siguiente sitio:

http://api.drupal.org/api/function/hook\_theme/7

**hook\_theme\_registry\_alter**

Hay algunos casos extremos en que un módulo puede querer modificar la entrada del Registro para un hook tema en particular. Afortunadamente, hay un hook para eso. ™ El hook\_theme\_registry\_alter () hook puede ser implementado para alterar el tema Registro directamente después de que ha sido construido conhook\_theme () y mecanismos de detección automática del sistema tema. Supongamos un Espectro módulo quiere controlar si los mensajes de estado de Drupal se muestran al usuario. Si nos fijamos en template\_preprocess\_page (), Verás que la $ show\_messages variable controla si se muestran los mensajes de estado. Por desgracia, si lo intentamos

**[88 ]**



*Capítulo 3*



simplemente aplicar un spook\_preprocess\_page () función, se ejecuta después de template\_preprocess\_page ()y, por lo tanto, demasiado tarde. Así, en lugar necesitamos implementar el siguiente código:

/ \*\*

\* Implementa hook\_theme\_registry\_alter (). \* /

función spook\_theme\_registry\_alter (& $ theme\_registry) {

* Añadir nuestra función preproceso personalizado al inicio de
* la lista de funciones de preproceso.

array\_unshift ($ theme\_registry ['page'] ['funciones de preproceso'], 'spook\_control\_page\_messages');

}

/ \*\*

* Implementa una función de procesamiento previo encargo; uno que no se
* descubierto automático durante la construcción del registro temático.

\* /

spook\_control\_page\_messages de función (y variables) {

* Anulación # show\_messages antes template\_preprocess\_page ()
* acceda a ella.

$ variables ['page'] ['#'] = show\_messages spook\_get\_control ();

}

Se crea la orden de las funciones de preproceso descritos en el apartado anterior durante la generación del registro del sistema del tema y se basa en la convención de nombres. Sin embargo, hook\_theme\_registry\_alter () nos permite alterar ese orden para satisfacer nuestra propia propósitos nefastos.

**¿Qué más?**

Para la mayoría de los módulos que usted escribe, los contenidos de este capítulo cubren todo lo necesario para aprender sobre el tema del sistema. En realidad, este capítulo podría haber sido TMI. Sin embargo, el tema del sistema está lleno de deliciosos a Dios no cubierta aquí, incluyendo clases de accesibilidad,hook\_theme\_enable, Y hook\_theme\_disable por nombrar sólo algunos. Documentación en línea de Drupal incluye una guía de referencia tema que puede ser una herramienta muy útil para la comprensión de algunos de los temas más oscuros. Se puede encontrar en la siguiente dirección:

http://drupal.org/theme-guide

**[89 ]**



*Tema Capa de Drupal*



**Resumen**

Usted ha aprendido mucho en este capítulo. Dado que muchos de estos conceptos temáticos están relacionados entre sí, su reto de aprender de ellos (y explicarlas!), Por lo que todo el sistema complejo tiene sentido. Hemos tocado en casi todas las partes del sistema, incluyendo el tema:

* Funciones temáticas y archivos de plantilla
* Funciones de preproceso y de proceso
* Implementaciones tema por defecto y redefiniciones temáticos
* drupal\_render (), Hacen que los elementos y sus propiedades
* hook\_element\_info ()
* hook\_page\_alter ()
* tema (), Hooks temáticos y sugerencias hook tema
* Registro temático, hook\_theme (), Y hook\_theme\_registry\_alter ()

Una vez creado un gráfico que mostraba todas las partes del sistema de tema en un gráfico, pero era más aterrador entonces útil. O por supuesto, si te gusta el horror, diríjase a:

http://www.slideshare.net/JohnAlbin/default-theme-implementations

En el próximo capítulo, vamos a tomar algunos de los temas más importantes en este capítulo y las usamos para construir un ejemplo del mundo real.

**[90 ]**



Tematización un Módulo

Ahora que usted ha aprendido la arquitectura de la capa de tema, vamos a poner ese conocimiento en práctica. En este capítulo, vamos a escribir algo de código real que tanto utiliza las implementaciones temáticos existentes y construye nuevas implementaciones temáticos.

Si bien el capítulo anterior fue un viaje relámpago de la capa de tema, este capítulo será una exploración más reflexivo del sistema y cómo utilizarlo mejor. Vamos a tocar en las siguientes ideas:

* Las ventajas de ser perezoso mediante la reutilización de código
* Encontrar y reutilización de un hook tema
* Colocación de CSS para representar los elementos
* Crear hojas de estilo para idiomas RTL
* La construcción de un hook tema

Para ayudarnos a conocer estos puntos, vamos a construir un módulo simple en nuestros ejemplos. Drupal viene con el módulo de blog, que crea blogs multi-usuario, una para cada usuario y un blog agregado. Sin embargo, muchos sitios web sólo es necesario un único blog. Vamos a volver a crear algunas de las funciones del módulo de blog, y es re-propósito

para un único blog que utiliza el tipo de contenido del artículo por defecto de Drupal. Si intenta salir de código de este capítulo, debe deshabilitar el módulo blog de Drupal 7 primero.

**La reutilización de una implementación tema por defecto**

Ahora, la primera pregunta que hay que preguntarse es "¿Debo volver a usar un hook tema existente o construir mi propio?" La reutilización de código es una de las virtudes de un programador perezoso. Así que la respuesta a esa pregunta es "sí, ser perezoso!"

*Tematización un Módulo*



Después de todo, si usted elige un hook tema existente, es suficiente para construir los datos. Usted no tiene que preocuparse por la construcción de la aplicación, registrarse conhook\_theme, La construcción de cualquier helper pre-render funciones, decidir sobre el mejor HTML de usar, y la construcción de cualquier CSS apoyar y jQuery. Lo que no me gusta de saltarse un trabajo extra?

Sin embargo, la pereza no es la única razón por la que usted desea volver a utilizar hooks temáticos existentes. No sólo es el camino más fácil, que también descubrirá integraciones interesantes con otros módulos. Como aprendimos en el Capítulo 3, Tema Capa de Drupal, módulos puede alterar la forma en que una aplicación tema funciona mediante el uso de funciones de preproceso / proceso. Esas alteraciones no están vinculadas a la utilización de sólo Drupal núcleo del hook tema; se ejecutan incluso cuando el módulo utiliza el hook tema.

Por ejemplo, al escribir el menu\_block módulo, que podría haber usado mis propios hooks tema a tema de las diferentes partes de los árboles de menús que muestra. Sin embargo, me decidí a volver a utilizar el menú hooks temáticos relacionados con los árboles del núcleo y sólo bombear los datos con gran cantidad de metadatos adicionales que el uso del núcleo carecía. Más tarde descubrí que otro módulo se diseñó para modificar árboles de menús del núcleo y hacer que se expanden y colapsan de forma dinámica. Ese módulo hizo esto mediante la alteración de hooks temáticos principales. Desde mi menu\_block módulo utiliza los mismos hooks, los dos módulos fueron inmediatamente interoperable. Ni desarrollador del módulo tenía que escribir ningún código de integración. ¡Cuenta!

La parte más difícil de reutilizar hooks temáticos existentes es simplemente encontrar el adecuado para usted. Usted puede buscar los de las implementaciones de tema por defecto de Drupal core en:

http://api.drupal.org/api/group/themeable/7

Cualquier hook implementado como una función de tema aparece en primer lugar. Los archivos de plantilla se enumeran a continuación, en la sección Archivos.

En Drupal 7, hay 184 hooks tema que usted podría utilizar en su propio código.

Muchos de esos hooks temáticos son específicos para el uso de un módulo de la base de, como tematización una forma de administración, pero sigue siendo útil para ir a través de toda la lista para encontrar los hooks se podía reutilizar. Para hacerlo aún más fácil, hemos incluido una lista de los hooks reutilizado más comúnmente temáticos:



**Hooks tema común**



file\_link html\_tag

Devuelve HTML de un enlace a un archivo.

Devuelve HTML de una etiqueta HTML genérico con atributos. A menudo, esto puede ser demasiado genérico un hook tema de usar, pero es muy útil para agregar una etiqueta a la <head> de un documento o de la tematización una etiqueta dentro de un elemento de representación.

|  |  |
| --- | --- |
| imagen | Devuelve HTML para una imagen. |
| image\_style | Devuelve HTML para una imagen con un estilo de imagen específica. |
| item\_list | Devuelve HTML para una lista de artículos que opcionalmente pueden anidarse. |

**[92 ]**



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | *Capítulo 4* | |
|  |  | |  |
|  | **Hooks tema común** | |  |
|  | Enlaces | Devuelve HTML para una lista de enlaces (no se pueden anidar). | |
|  | more\_link | Devuelve HTML para un enlace más, que se utiliza a menudo en los bloques. | |
|  | buscapersonas | Devuelve HTML para un elemento de consulta pager, una lista de páginas para los conjuntos de resultados | |
|  |  | demasiado largo para una sola página. | |
|  | progress\_bar | Devuelve HTML para un indicador que muestra el progreso de una tarea. | |
|  | tabla | Devuelve HTML para una tabla. | |
|  | nombre de usuario | Devuelve HTML para un nombre de usuario. | |
|  | lista\_usuarios | Devuelve HTML para una lista de usuarios. | |
|  | user\_picture | Devuelve HTML para una imagen configurada para la cuenta del usuario. |  |

Los hooks temáticos mencionados anteriormente son los hooks más utilizados en el núcleo de Drupal y también en Drupal contrib módulos.

**Drupal bloques revisitados**

Así que vamos a empezar a construir nuestra single\_blog módulo. Vamos a empezar con la . Info presentar, por supuesto. Todas las líneas en este . Info archivo debe ser familiar para usted:

; $ Id $

name = Blog Individual

Descripción = Permite que un solo blog durante los usuarios individuales o múltiples.

core = 7.x

paquete = Drupal 7 Desarrollo

archivos [] = single\_blog.module

Una de las cosas que el módulo de blog proporciona es un bloque de la lista entradas de blog recientes. Vamos a utilizar la API de bloque que se introdujo en el Capítulo 2, Crear el primer módulo, a medida que construimos un par de hooks básicos y una función de API para nuestro nuevo módulo:

<? Php / / $ Id $

/ \*\*

\* @ File

**[93 ]**



*Tematización un Módulo*



\* Permite a un blog de un solo usuario. \* /

* Después de aprender API valor en el Capítulo 5, usted será capaz de
* realizar estos ajustes configurables.

define ('SINGLE\_BLOG\_NODE\_TYPE', 'artículo'); define ('SINGLE\_BLOG\_LIST\_COUNT', 3);

/ \*\*

* Devuelve una lista de entradas de blog.
* @ Param $ número
* El número de entradas de blog para volver.
* @ Return
* Un objeto de conjunto de resultados que contiene la lista de entradas de blog.

\* /

función single\_blog\_list ($ numero) {

* Utilice la API de base de datos para recuperar los datos.
* @ See http://drupal.org/node/310069

$ Query = db\_select ('nodo', 'n')

-> Campos ('n', array ('nid', 'title', 'creado', 'el uid')) -> condición («tipo», SINGLE\_BLOG\_NODE\_TYPE) -> condición ("status", 1)

-> OrderBy ('creado', 'DESC') -> rango (0, $ numero) -> addtag ​​('node\_access') -> execute ();

return $ consulta;

}

/ \*\*

\* Implementa hook\_block\_info (). \* /

función single\_blog\_block\_info () {$ cuadras = array ();

* La clave de la matriz define el parámetro $ delta utilizado en todos los
* otros hooks de bloque.

$ bloques ['reciente'] = array (

/ / El nombre del bloque en la página de administración de bloques. 'Info' => t ('Entradas de Blog recientes'),

);

return $ bloques;

}

**[94 ]**



*Capítulo 4*



En el capítulo 5, La construcción de una interfaz de administración, usted aprenderá cómo construir formularios de administración que podrían hacer nuestra single\_blog módulo configurable, pero en este capítulo simplemente creamos un par de constantes PHP que definen el tipo de nodo que se utilizará para las entradas de blog y la cuenta de la lista para el bloque.

El siguiente, le ofrecemos una API simple para nuestro módulo que permite a cualquier código para recuperar una lista de nodos en el blog; hemos llamado a la función single\_blog\_list (). Estamos utilizando la API de base de datos de Drupal para consultar los datos. Usted puede aprender más sobre él a partir de la documentación en línea de Drupal en http://drupal.org/node/310069. Por ahora, usted tiene que confiar en los nombres de métodos relativamente auto-documentado de DB API. Elegimos el único ID de nodo, título, fecha de creación y autor (UID es sinónimo de identificación del usuario) Campos de nodos que son de la SINGLE\_BLOG\_NODE\_TYPE tipo de contenido y que se publican (estado = 1). Nosotros sólo seleccionamos la $ Número últimas nodos que tienen

ha creado.

¿Por qué creamos un single\_blog\_list () función en lugar de limitarse a poner ese código de consulta de base de datos dentro hook\_block\_view ()? Algunos desarrolladores de Drupal están tan atrapados en enganchar en las API de Drupal que se olvidan de escribir APIs abstraídas de la lógica de negocio de su propio módulo. No cometa el mismo error. Podríamos poner la consulta de base de datos dentro de un hook de Drupal, pero que reduce la posibilidad de que otros módulos se pueden integrar con su módulo de manera que nunca podría anticipar. Recuerde, ser perezoso. Si la API de su módulo es lo suficientemente bueno, algún otro desarrollador escribir el código para integrar el módulo con el de ellos.

Ahora que tenemos nuestra función API escrito, vamos a usar para construir el bloque usando Bloquear API de Drupal:

/ \*\*

\* Implementa hook\_block\_info (). \* /

función single\_blog\_block\_info () {$ cuadras = array ();

* La clave de la matriz define el parámetro $ delta utilizado en todos los
* otros hooks de bloque.

$ bloques ['reciente'] = array (

/ / El nombre del bloque en la página de administración de bloques. 'Info' => t ('Entradas de Blog recientes'),

);

return $ bloques;

}

/ \*\*

\* Implementa hook\_block\_view ().

\*

**[95 ]**



*Tematización un Módulo*



* Primer proyecto!
* @ Pararm $ delta
* El nombre del bloque solicitado.

\* /

función single\_blog\_block\_view ($ delta ='') {

* Crear un bloque vacío. $ Block = array (

'Subject' =>'', 'contenido' =>'',

);

* Compruebe que el bloque se está solicitando. if ($ delta == 'reciente') {

/ / Establecer el título del bloque.

$ Bloque ['subject'] = t ('Entradas de Blog recientes');

/ / Comprobar si el usuario puede acceder a los contenidos. if (user\_access ('content access')) {

/ / Recuperar los nodos más recientes.

$ Resultado = single\_blog\_list (SINGLE\_BLOG\_LIST\_COUNT);

/ / Crear enlaces para cada entrada del blog. $ items = array ();

foreach ($ resultado as $ node) {$ items [] = array (

=> L 'data' ($ node-> title, 'nodo /'. $ Node-> nid), 'class' => array ('nodo'. $ Node-> nid),

);

}

if (! vacías ($ items)) {

/ / Tema de la lista de entradas de blog. $ Bloque ['content'] = theme ('item\_list', array (

'items' => $ items));

}

}

}

return $ block;

}

**[96 ]**



*Capítulo 4*



Nuestro single\_blog\_block\_info () función es un simple hook\_block\_info () aplicación, como se describe en el capítulo 2. Volvemos una matriz de información que describe los bloques que nuestro módulo proporciona. Cada clave de la matriz es el "delta" para ese bloque. El delta, cuando se combina con el nombre del módulo, crea el ID de bloque único para un bloque que se almacena en la block.tpl.php el expediente de $ Block\_html\_id. Por ejemplo, hemos definido nuestro bloque "reciente del blog posts" como tener un delta de reciente, Por lo que su ID de bloque completo es de un solo blog-reciente. El identificador del bloqueo es utilizado por Drupal para asignar bloques de regiones. La "info"Devuelto por nuestra single\_blog\_block\_info () función define el "nombre" del bloque que se ve

en la página de administración de bloques.

La single\_blog\_block\_view () implementos de función hook\_block\_view (). Cuando Drupal quiere rendir un bloque en particular, llama al hook\_block\_view () implementación del módulo responsable de la manzana, que pasa en el bloque de $ Delta como un parámetro. Nuestra función comprueba primero la $ Delta dada a la misma y luego devuelve los datos del bloque solicitado como una matriz que contiene el título del bloque en el sujeto clave y su contenido en el contenido clave. En primer lugar, vamos a definir el bloque título utilizando la t () función. Desde el Capítulo 2, usted debe recordar que t () traduce cadenas de idioma Inglés a otros idiomas. Entonces nuestra función comprueba que el usuario que está viendo la página se puede acceder a los contenidos del sitio web a través user\_access ("acceso

contenido "); esta llamada comprueba si el usuario actual tiene la acceder al contenido permiso. Si el usuario actual no tiene los permisos adecuados, te darás cuenta de que nuestra función devuelve un vacío contenido clave; esto señala a Drupal que el bloque solicitado no se debe representar.

Lo último que vamos a hacer antes de que finalmente empezamos tematización es llamar a nuestra función de la API, single\_blog\_list (), Con el fin de obtener los datos en bruto de la base de datos. En realidad, single\_blog\_list () devuelve un objeto conjunto de resultados, pero no hemos aprendido las funciones de la API de bases de datos especiales para recuperar cada fila del objeto conjunto de resultados. Si nos limitamos a iterar sobre este objeto utilizando foreach, Vamos a tener una serie de objetos que contienen los datos de cada fila.

foreach ($ resultado as $ node) {}

Específicamente, cada vez a través de la foreach bucle, $ Node será un objeto con las propiedades de cada campo de base de datos que habíamos pedido, nid,título,creado, Y UID. Cuando el $ Resultado objeto ha procesado todas las filas, el foreach bucle finalizará automáticamente.

**[97 ]**



*Tematización un Módulo*



**Tematización un bloque Drupal**

Así que, ahora que sabemos cómo conseguir todos nuestros datos, tenemos que decidir cómo vamos a tema de él. Estamos realmente va a construir tres versiones de nuestro

single\_blog\_block\_view (). Esto nos permitirá probar varios métodos diferentes de tematización y aplicar todos los aspectos de la capa tema que aprendimos en el capítulo 3.

Mirando hacia atrás en la tabla anterior, hooks tema común, nos ayudarán a decidir que los hooks temáticos existentes serían buenos candidatos para nuestros datos. Inicialmente, estamos creando una lista de enlaces a nuestras publicaciones en el blog, así que theme\_links () en realidad sería una combinación perfecta. Sin embargo, en la siguiente iteración de nuestra hook\_block\_view () función que vamos a crear más de una simple lista de enlaces, por lo que vamos a ver de nuevo. La item\_list hook nos permitirá crear una lista que contiene los datos arbitrarios.

/ / Tema de la lista de entradas de blog.

$ Bloque ['content'] = theme ('item\_list', array ('artículos' => $ items));

Al mirar la documentación para theme\_item\_list () (Disponible en

http://api.drupal.org/api/function/theme\_item\_list/7) Podemos ver que espera como un parámetro artículos array. Cada elemento de la lista puede ser una simple cadena o la cadena puede ser colocado en el datos clave de una matriz en la que las otras teclas se tratan como atributos para el elemento de la lista:

$ items [] = array (

=> L 'data' ($ node-> title, 'nodo /'. $ Node-> nid), 'class' => array ('nodo'. $ Node-> nid),

);

En la superficie, esta construcción es similar a la API de Render, pero esto es simplemente la convención utilizada por esta función el tema; que no utiliza drupal\_render () para convertir la matriz en una cadena.

El camino interior a un nodo es siempre "node / [ID de nodo]". Afortunadamente, incluso si el autor da a la entrada del blog un alias de URL, no tenemos que averiguar el alias que el L () función reescribirá automáticamente para utilizar la dirección URL correcta. Así, con la L () función, nos estamos tomando el título de cada nodo y el ID de nodo y la construcción de un simple enlace al nodo y colocándolo en el elemento de la lista datos elemento. theme\_item\_list () trata la clase elemento como un atributo para la envoltura <li> elemento.

Okay. Hemos terminado nuestro primer borrador de nuestro módulo. Buscando en la siguiente captura de pantalla, se puede ver que nuestros bloque muestra como una simple lista desordenada de títulos de nodo. Si esto fuera un tema de aplicación Drupal 6, tendríamos que hacer! Sin embargo, en Drupal 7, todohook\_block\_view () y todas las devoluciones de llamada de páginas (las funciones que devuelven el contenido principal de una página) debe devolver una matriz renderizable lugar de una cadena. Así, mientras que nuestro código funciona (desde Drupal 7 considera una cadena simple para ser una matriz renderizable degenerado), tendremos que arreglar ese pequeño defecto en nuestro segundo proyecto.

**[98 ]**



*Capítulo 4*



La siguiente es una captura de pantalla de la versión de uno de los bloques de nuestro módulo:



**Render elemento y una sugerencia hook tema**

Ok, primero vamos a arreglar el fallo que nos fuimos en nuestra primera versión de hook\_block\_view () mediante la conversión de que correctos tema () llamar en una matriz de render. Esto es realmente muy fácil, simplemente hay que editar nuestro single\_blog\_block\_view () y reemplazar:

$ Bloque ['content'] = theme ('item\_list', array ('artículos' => $ items)); con el siguiente código:

$ Bloque ['content'] = array (

**'# Lema' => 'item\_list\_\_single\_blog',**

'# artículos' => $ items,

);

Como se puede ver en el nuevo código, sólo tenemos que convertir todas las claves del array pasado en el variables $ parámetro a tema () en # prefijo nombres clave. artículos se convierte en # artículos, Y así sucesivamente. También vamos a aprovechar esta oportunidad para añadir un tema hook sugerencia como se describe en el Capítulo 3. tema () comprobará para una item\_list\_\_single\_ Blog aplicación antes de utilizar la opción predeterminada item\_list hook tema.

**[99 ]**



*Tematización un Módulo*



La conversión de una llamada a un elemento de tema de hacer fue bastante indoloro, no? Vamos a añadir algo más contenido a nuestro bloque. Módulo de blog de Drupal core proporciona una página (y el menú de devolución de llamada) que muestra teasers de las publicaciones en blog recientes en la ruta / blog. Dado que las devoluciones de llamada de menú no son parte del sistema de tema, vamos a salir de esa página sin aplicarse, pero vamos a suponer que lo hicimos en práctica. En ese caso, nuestro bloque debería proporcionar un eslabón más que va a la página del blog.

Adición de contenido adicional a un elemento de render es fácil. Desde nuestra $ Bloque ['content'] es un solo elemento de render, primero tenemos que hacer para mover el elemento existente como un elemento secundario, lo que podemos hacer por moverlo a $ Bloque ['content'] ['list']. La lista nombre es una etiqueta arbitraria que estamos dando a un elemento secundario; con tal de que no se inicia con una #, No importa mucho lo que nosotros llamamos. Vamos a añadir nuestro nuevo eslabón más como

un hermano a la lista niño:

* Tema de la lista de entradas de blog. $ Bloque ['content'] ['list'] = array (

'# Lema' => 'item\_list\_\_single\_blog', '# items' => $ items,

);

* Añadir un enlace a la lista completa de entradas de blog. $ Bloque ['content'] ['más'] = array (

'# Lema' => 'more\_link', '# url' => 'blogs',

'# Title' => t ('Lea las últimas entradas de blog.'),

);

Al mirar hacia atrás en nuestra lista de hooks tema común, usted debe haber notado la more\_link hook tema que usamos para nuestro bloque de **más** enlace. Una vez más sólo tenemos examinar la documentación para theme\_more\_link para determinar la forma de estructurar el elemento secundario. Ver http://api.drupal.org/api/function/theme\_more\_link/7.

**Creación de una función pre\_render**

Mira ahora en nuestro contenido, probablemente notará que hay una pieza de contenido que todavía está encerrado en una cadena en lugar de ser modificados en un elemento de render: la datos elemento de cada uno de los $ items pasado en nuestro item\_list render elemento. La conversión de la L () llamar a un elemento de render es más sencilla; basta con cambiar la siguiente línea:

=> L 'data' ($ node-> title, 'nodo /'. $ Node-> nid),

**[100 ]**



*Capítulo 4*



en estas líneas:

=> Array 'data' ('type #' => 'link',

'# Title' => $ node-> title, '# href' => 'nodo /'. $ Nid nodo->,

),

system\_element\_info () define el enlace elemento y podemos ver a partir de su declaración y examinar su # Pre\_render devoluciones de llamada que hacen que los elementos con un # Tipo de enlace se representará en un vínculo mediante L ().

Desafortunadamente, la theme\_item\_list () función no espera que un elemento de render en su llave de datos, por lo que se ahogue en estos contenidos. Teniendo en cuenta el empuje para hacer que todo sea alterable con render arrays, esto parece un descuido. Si encuentras un error o inconsistencia en Drupal (y usted!), Puede contribuir con sólo hacer una nota de la misma y la creación de un problema en la cola de la edición de Drupal. De hecho, acabo de crear un tema nuevo para este problema en http://drupal.org/node/891112.

Mientras tanto, vamos a tener que trabajar alrededor de este problema en lugar de cortar el núcleo de Drupal. Afortunadamente, una inspección detallada de las propiedades de hacer elemento a nuestra disposición (de nuevo, véase el capítulo 3) indican que se puede utilizar una# Pre\_render callback para alterar nuestro render elemento antes de que se pasa a la función de la temática del elemento. Vamos a añadir una

en nuestro $ Bloque ['content'] ['list'] elemento secundario:

$ Bloque ['content'] ['list'] = array ('# temáticos' => 'item\_list\_\_single\_blog', '# items' => $ items,

'# Pre\_render' => array ('single\_blog\_item\_list\_child\_render'),

);

Desde drupal\_render () pasará todo el elemento secundario a nuestra # Pre\_render devolución de llamada, sólo tenemos que asegurarnos de que nuestro callback modifica la datos elemento de todos nuestros artículos.

/ \*\*

* Prestar los elementos secundarios de theme\_item\_list () antes de su
* datos es temática.

\* /

función single\_blog\_item\_list\_child\_render ($ elementos) {foreach (array\_keys ($ elementos ['records']) as $ clave) {

* Tome la matriz renderizable que nos propusimos en
* single\_blog\_block\_view () y lo hacen en la cadena
* que theme\_item\_list () espera.

**[101 ]**



*Tematización un Módulo*



if (($ is\_array elementos ['records'] [$ clave] ['datos'])) {$ elementos ['records'] [$ clave] ['data'] =

drupal\_render ($ elementos ['records'] [$ key] ['data']);

}

}

return $ elementos;

}

En nuestra single\_blog\_item\_list\_child\_render () función, simplemente recorrer todos los # artículos, Determinar si tienen una matriz en su datos elemento y llamada drupal\_render () en su contenido.

**Colocación de CSS para hacer arreglos**

Si nos fijamos en la captura de pantalla de la primera versión, se puede ver que el estilo por defecto de nuestro bloque es menos inspiradora, así que vamos a tweak que al dar nuestro contenido algún estilo predeterminado sensata mediante la adición de una hoja de estilos CSS.

Desde la versión 5, Drupal ha tenido undrupal\_add\_css () funcionar para agregar las hojas de estilo CSS a las páginas. ¿Qué hay de nuevo en Drupal 7 es que, debido al bloqueo y páginas caché de Drupal y las capacidades de hook\_page\_alter (), Ahora tenemos que fijar nuestra hoja de estilos directamente en el elemento de render que estamos creando. Si tuviéramos que utilizardrupal\_add\_css (), La hoja de estilos no se almacena en caché con su bloque y también sería considerablemente más difícil de modificar la hoja de estilo si un hook\_page\_alter () implementación deseada

a (Por ejemplo si se retira el bloque y quería quitar el CSS también.)

Así que en lugar de llamar a drupal\_add\_css () desde dentro de nuestra single\_blog\_block\_view () función, lo añadimos a la matriz devuelta render:

/ / Añadir un archivo CSS con el estilo de la manzana. $ Bloque ['content'] ['# adjunto'] ['css'] [] = drupal\_get\_path ("módulo",

'Single\_blog'). '/ Solo blog.css';

Utilizamos drupal\_get\_path () para encontrar el camino a nuestro módulo relativo a la raíz del sitio Web. La # Adjunta matriz puede contener una lista de los archivos CSS y JS de adjuntar a nuestra render elemento. Para los archivos de JavaScript, simplemente añada a la js matriz a través de

['# Adjunta'] ['js'] [].

Y aquí están los contenidos de nuestra hoja de estilo de un solo blog.css:

/ \* $ Id $ \* /

.. Bloque de un solo-blog de contenido ul {padding-left: 0; / \* LTR \* /

}

**[102 ]**



*Capítulo 4*



.. Bloque de un solo-blog de contenido li ul {margin-bottom: 10px; de tipo list-style: ninguno;

}

**Idiomas RTL**

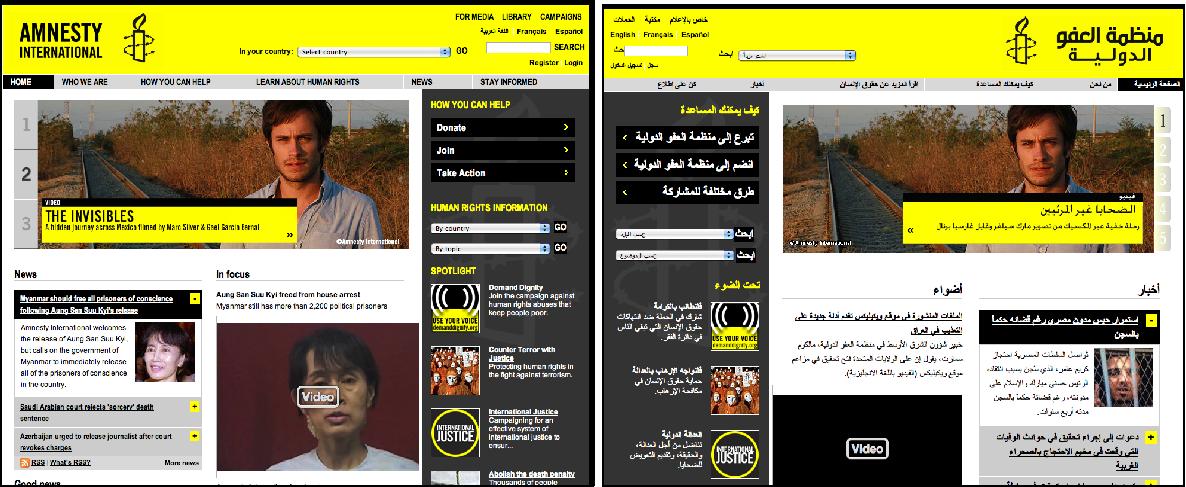
Una cosa que usted tendrá que tener en cuenta al escribir hojas de estilo es el apoyo de Drupal para idiomas RTL, los idiomas que se leen de derecha a izquierda, por ejemplo, el árabe o el

Hebreo. Los usuarios de los sitios web de RTL esperan que todo lo relacionado con ese sitio web fluya de derecha a izquierda en vez de la normal de Inglés de izquierda a derecha. La convención utilizada por los sitios web que soportan ambos idiomas RTL y LTR es para voltear la disposición del diseño horizontal dependiendo de la direccionalidad de la lengua.

Todo un ejemplo vivo de cómo se da la vuelta diseños de página web de derecha a izquierda es Amnistía

El sitio web de Internacional; comparar la versión del idioma árabe en http://www.

amnesty.org / ar la versión en idioma Inglés en http://www.amnesty.org/en. Observe cómo la barra lateral cambia de bando según el idioma:



Desde un punto de vista CSS, esto significa elementos HTML cuyo estilo de la izquierda se diferencia de su necesidad estilo del lado derecho a que su estilo alterado cuando el idioma actual es RTL. Si se está visualizando un lenguaje RTL, Drupal, por cada hoja de estilo, busca una hoja de estilo suplementario RTL cargar. Así que, si el hebreo es el idioma activo, Drupal buscará de un solo blog-rtl.css cargar, además de (y sólo después) de la solicitada solo blog.css presentar. Desde nuestra -Rtl hoja de estilos se carga, además de la hoja de estilo estándar, simplemente tenemos que incluir las reglas y propiedades necesarias para sustituir la versión del LTR de nuestros estilos. Para que sea más fácil hacer un seguimiento de esas propiedades, módulos de Drupal deben colocar una / \* LTR \* / comentar junto a cada propiedad que necesita ser reemplazado.

**[103 ]**



*Tematización un Módulo*



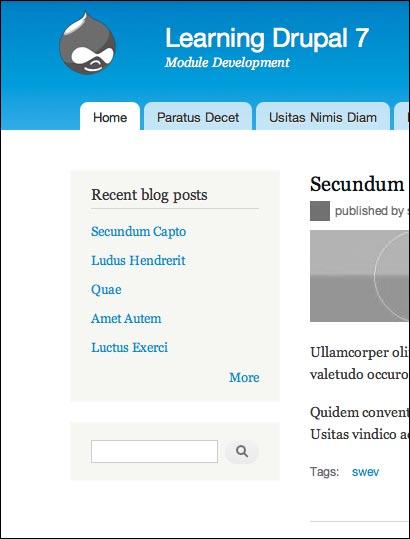
Tenga en cuenta que la . Bloque de un solo-blog. Contenido ul gobernar en el solo blog.css hoja de estilo especifica un margen izquierdo. Ya que es la única propiedad que es direccional, es el único que tenemos que reemplazar en el de un solo blog-rtl.css presentar.

/ \* $ Id $ \* /

.. Bloque de un solo-blog de contenido ul {padding-right: 0;

}

Tenga en cuenta que si nuestro margen izquierdo original 10px, Habríamos necesitado para anular que en nuestra hoja de estilo RTL estableciendo padding-left a 0 , y luego poner padding-right a 10px. La siguiente es una captura de pantalla de la versión dos de nuestro bloque de módulo:



Si nos fijamos en la pantalla, se puede ver el nuevo vínculo Más y cómo la pantalla de nuestro bloque ha mejorado.

**[104 ]**



*Capítulo 4*



Después de todas estas modificaciones, el segundo borrador de nuestrasingle\_blog\_block\_view () la función se ha completado y debe tener este aspecto:

/ \*\*

* Implementa hook\_block\_view ().
* Segundo proyecto!

\*

* @ Pararm $ delta
* El nombre del bloque solicitado.

\* /

función single\_blog\_block\_view ($ delta ='') {

* Crear un bloque vacío. $ Block = array (

'Subject' =>'', 'contenido' =>'',

);

* Compruebe que el bloque se está solicitando. if ($ delta == 'reciente') {
  + Establecer el título del bloque.

$ Bloque ['subject'] = t ('Entradas de Blog recientes');

/ / Comprobar si el usuario puede acceder a los contenidos. if (user\_access ('content access')) {

/ / Recuperar los nodos más recientes.

$ Resultado = single\_blog\_list (SINGLE\_BLOG\_LIST\_COUNT);

/ / Crear enlaces para cada entrada del blog. $ items = array ();

foreach ($ resultado as $ node) {$ items [] = array (

=> Array 'data' ('type #' => 'link',

'# Title' => $ node-> title, '# href' => 'nodo /'. $ Nid nodo->,

),

'Class' => array ('nodo'. $ Node-> nid),

);

}

if (! vacías ($ items)) {

/ / Tema de la lista de entradas de blog.

**[105 ]**



*Tematización un Módulo*



$ Bloque ['content'] ['list'] = array ('# temáticos' => 'item\_list\_\_single\_blog', '# items' => $ items,

'# Pre\_render' => array ('single\_blog\_item\_list\_child\_render'),

);

* Añadir un enlace a la lista completa de entradas de blog. $ Bloque ['content'] ['más'] = array (

'# Lema' => 'more\_link', '# url' => 'blogs',

'# Title' => t ('Lea las últimas entradas de blog.'),

);

* Añadir un archivo CSS con el estilo de la manzana.

$ Bloque ['content'] ['# adjunto'] ['css'] [] = drupal\_get\_path ('módulo', 'single\_blog'). '/ Solo blog.css';

}

}

}

return $ block;

}

**Pasos para crear una aplicación tema por defecto**

Bien, ahora es el momento de exorcizar nuestra perezosa-desarrollador de la costumbre y la práctica de la construcción de nuestro propio hook tema. Desde el capítulo 3, se debe recordar que tendremos que hacer las siguientes cosas:

1. Registre el hook tema y definir las variables predeterminadas.
2. Construir la implementación predeterminada de nuestro hook tema.
3. Re-crear el registro temático.
4. Construir un elemento de rendir a utilizar el hook tema.

Nuestra implementación actual de las entradas de blog recientes bloquean simplemente muestra una lista de títulos de los blogs. Sin embargo, sería bueno incluir la fecha de cada puesto, así como el autor

(Si tenemos varias personas la creación de puestos). Así que en esta tercera y última versión de nuestro módulo, vamos a crear un de un solo blog-block-item.tpl.php para representar el contenido de cada elemento en nuestra lista de entradas de blog. Por convención en Drupal, cualquier CSS,

Archivos JavaScript, o plantilla que necesita un módulo deben usar guiones en lugar de guiones en sus nombres de archivo.

**[106 ]**



*Capítulo 4*



Antes de comenzar la construcción de la necesaria single\_blog\_block\_item hook tema, primero vamos a añadir todos los datos que necesitamos para la tercera versión de nuestro módulo. Mirando hacia atrás en la forma en que generamos los artículos a la lista, podemos ver que todos los datos que queremos es que en el $ Node variable.

/ / Crear enlaces para cada entrada del blog. $ items = array ();

foreach ($ resultado as $ node) {$ items [] = array (

=> Array 'data' ('type #' => 'link',

'# Title' => $ node-> title, '# href' => 'nodo /'. $ Nid nodo->,

),

'Class' => array ('nodo'. $ Node-> nid),

);

}

Así, en lugar de crear un sencillo hacer elemento utilizando los bits de la $ Node variables, vamos a pasar toda esa variable en nuestro nuevo hook tema:

/ / Crear enlaces para cada entrada del blog. $ items = array ();

foreach ($ resultado as $ node) {$ items [] = array (

=> Array 'data' (

**'# Lema' => 'single\_blog\_block\_item', '# nodo' => $ node,**

),

'Class' => array ('nodo'. $ Node-> nid),

);

}

**hook\_theme () implementaciones**

Vamos a tener que crear un single\_blog\_theme () aplicación de hook\_theme () y registrar una single\_blog\_block\_item hook tema.

/ \*\*

\* Implementa hook\_theme (). \* /

función single\_blog\_theme ($ existente, escriba $, $ tema, $ ruta) {return array (

'Single\_blog\_block\_item' => array (

**[107 ]**



*Tematización un Módulo*



"variables" => array ('nodo' => NULL,

),

'Plantilla' => 'single-blog-bloque-item',

),

);

}

Voy a explicar la las variables matriz en la siguiente sección, pero vamos a ir rápidamente sobre la otra clave ahora.

Dado que este es un hook tema, y ​​se implementará utilizando una plantilla en lugar de una función de tema, tendremos que incluir la plantilla clave y especificar el nombre base del archivo de plantilla, de un solo blog-bloque-item. Drupal añadirá automáticamente la

. Tpl.php al final del nombre de la base cuando se busca el archivo, por lo que no debe incluirlo.

**Variables frente a hacer elemento**

En el capítulo 3, aprendimos sobre las diferencias entre el uso de la las variables clave y el uso de la render elemento clave en su hook\_theme (). Una y sólo una de esas claves deben estar presentes en cada declaración de hook tema. Sin embargo, todavía puede ser un poco confuso en cuanto a lo que para usar cuando usted está construyendo su implementación tema.

Sólo hay una situación en la que usted podría utilizar el render elemento clave: si sus datos pueden ser representados por un solo elemento o representar por una sola matriz renderizable anidadas render elementos. Si ese no es el caso, entonces usted debe especificar el las variables clave y especificar las variables que se pasará a tema () y su valores por defecto.

Lo mismo ocurre con nuestras características coincidan con el elemento requerimiento representaría por encima? Nuestro $ Node variable es sólo un objeto de nodo parcial y no un elemento de procesamiento, por lo que debe utilizar el las variables clave y especificar los valores predeterminados para todas nuestras variables.

Como nota al margen, si nos fijamos en su lugar la forma en que hemos construido el elemento de datos en la segunda versión de nuestro módulo (a enlace # Tiporender elemento), podemos ver que podríamos seguir adelante y utilizaciónrender elemento como la clave si nuestra segunda versión del módulo tenía una hook\_theme () aplicación.

Dado que la variable de nodo es un objeto, se establece el valor por defecto a ser simplemente el NULL valor.

**[108 ]**



*Capítulo 4*



**Funciones de preproceso**

Nuestro hook tema ahora se le da un $ Node objeto, pero los archivos de plantilla esperan las variables que contienen cadenas o hacer elementos. Así que vamos a tener que transformar el $ Node datos de objetos en una serie de variables. Técnicamente, podríamos haber realizado esta lógica empresarial directamente dentro de nuestra single\_blog\_block\_view () función, pero en lugar de eso vamos a hacer esta transformación en una función de procesamiento previo.

Eso es en realidad el propósito de la función de preproceso: para transformar los datos en bruto en variables necesarias para la plantilla de un hook de tema o función de tema. (Además, recuerdan que las funciones de preproceso nunca deben consultar los datos en bruto, los datos brutos deben ser pasados ​​como variables.)

Dado que somos dueños de este hook tema, tendremos que definir nuestra función de preproceso con un TEMPLATE\_ prefijo.

/ \*\*

\* Preprocesa las variables individuales de elementos de bloque blog. \* /

función template\_preprocess\_single\_blog\_block\_item (variables y $) {$ node = $ variables ['nodo'];

Para que sea más fácil acceder a todas las propiedades del objeto de nuestro nodo variable que vamos a crear en primer lugar una $ Node variable local que vamos a utilizar dentro de la función de preproceso:

/ / Crear una matriz renderizable por el título. $ variables ['title'] = array (

'Type #' => 'link', '# title' => $ node-> title,

'# Href' => 'Nodo /'. $ Nid nodo->,

);

A continuación vamos a crear el $ Title variable como un elemento de procesamiento; es idéntica a lo que vimos en la segunda versión de nuestro módulo:

/ / Formato de la fecha de creación del nodo. $ variables ['creados'] = $ node-> creado;

$ variables ['fecha'] = format\_date ($ node-> creado, 'custom', 'F d, Y');

Fecha de marcas de tiempo no hacen muy buena render elementos, por lo que sólo tendremos que crear dos variables, una con el, valor de la fecha sin formato en bruto y uno con fecha con formato:

* Cargue el objeto de cuenta con el creador y la tienda del nodo
* en una variable para themer conveniencia.

$ variables ['user'] = user\_load ($ node-> uid);

**[109 ]**



*Tematización un Módulo*



/ / Tema del nombre de usuario.

$ variables ['name'] = theme ('username', array (

'Cuenta' => $ variables ['user']));

}

Y, por último, vamos a pasar la $ Usuario objeto de la autora y el tema del nombre de usuario.

Todo lo que queda es para ordenar las variables de la forma en que deseamos en nuestro archivo de la plantilla!

Sin embargo, ya hemos hecho el último cambio en nuestro . Módulo archivo, vamos a ver el producto terminado:

<? Php / / $ Id $

/ \*\*

* @ File
* Permite que un solo blog durante los usuarios individuales o múltiples.

\* /

* Después de aprender API valor en el Capítulo 5, usted será capaz de
* realizar estos ajustes configurables.

define ('SINGLE\_BLOG\_NODE\_TYPE', 'artículo'); define ('SINGLE\_BLOG\_LIST\_COUNT', 5); define ('SINGLE\_BLOG\_DATE\_FORMAT', 'F d, Y');

/ \*\*

* Devuelve una lista de entradas de blog.
* @ Param $ número
* El número de entradas de blog para volver.
* @ Return
* Un objeto de conjunto de resultados que contiene la lista de entradas de blog.

\* /

función single\_blog\_list ($ numero) {

* Utilice la API de base de datos para recuperar los datos.
* @ See http://drupal.org/node/310069

$ Query = db\_select ('nodo', 'n')

-> Campos ('n', array ('nid', 'title', 'creado', 'el uid')) -> condición («tipo», SINGLE\_BLOG\_NODE\_TYPE) -> condición ("status", 1)

-> OrderBy ('creado', 'DESC') -> rango (0, $ numero) -> addtag ​​('node\_access') -> execute ();

**[110 ]**



*Capítulo 4*



return $ consulta;

}

/ \*\*

\* Implementa hook\_block\_info (). \* /

función single\_blog\_block\_info () {$ cuadras = array ();

* La clave de la matriz define el parámetro $ delta utilizado en todos los
* otros hooks de bloque.

$ bloques ['reciente'] = array (

/ / El nombre del bloque en la página de administración de bloques. 'Info' => t ('Entradas de Blog recientes'),

);

return $ bloques;

}

/ \*\*

* Implementa hook\_block\_view ().
* Tercer proyecto!

\*

* @ Pararm $ delta
* El nombre del bloque solicitado.

\* /

función single\_blog\_block\_view ($ delta ='') {

* Crear un bloque vacío. $ Block = array (

'Subject' =>'', 'contenido' =>'',

);

* Compruebe que el bloque se está solicitando. if ($ delta == 'reciente') {

/ / Establecer el título del bloque.

$ Bloque ['subject'] = t ('Entradas de Blog recientes');

/ / Comprobar si el usuario puede acceder a los contenidos. if (user\_access ('content access')) {

/ / Recuperar los nodos más recientes.

$ Resultado = single\_blog\_list (SINGLE\_BLOG\_LIST\_COUNT);

**[111 ]**



*Tematización un Módulo*



/ / Crear enlaces para cada entrada del blog. $ items = array ();

foreach ($ resultado as $ node) {$ items [] = array (

=> Array 'data' (

'# Lema' => 'single\_blog\_block\_item', '# nodo' => $ node,

),

'Class' => array ('nodo'. $ Node-> nid),

);

}

if (! vacías ($ items)) {

* Tema de la lista de entradas de blog. $ Bloque ['content'] ['list'] = array (

'# Lema' => 'item\_list\_\_single\_blog', '# items' => $ items,

'# Pre\_render' => array ('single\_blog\_item\_list\_child\_render'),

);

* Añadir un enlace a la lista completa de entradas de blog.

$ Bloque ['content'] ['más'] = array ('# lema' => 'more\_link',

'# Url' => 'blogs',

'# Title' => t ('Lea las últimas entradas de blog.'),

);

/ / Añadir un archivo CSS con el estilo de la manzana. $ Bloquear ['content'] ['# adjunto'] ['css'] [] =

drupal\_get\_path ('módulo', 'single\_blog'). '/ Solo blog.css';

}

}

}

return $ block;

}

/ \*\*

\* Prestar los elementos secundarios de theme\_item\_list () antes de sus datos tiene como tema.

\* /

función single\_blog\_item\_list\_child\_render ($ elementos) {foreach (array\_keys ($ elementos ['records']) as $ clave) {

/ / Toma el array renderizable que nos propusimos en

**[112 ]**



*Capítulo 4*



* single\_blog\_block\_view () y lo hacen en la cadena
* que theme\_item\_list () espera.

if (($ is\_array elementos ['records'] [$ clave] ['datos'])) {$ elementos ['records'] [$ clave] ['data'] =

drupal\_render ($ elementos ['records'] [$ key] ['data']);

}

}

return $ elementos;

}

/ \*\*

\* Implementa hook\_theme (). \* /

función single\_blog\_theme ($ existente, escriba $, $ tema, $ ruta) {return array (

'Single\_blog\_block\_item' => array ('Variables' => array (

'Nodo' => NULL,

),

'Plantilla' => 'single-blog-bloque-item',

),

);

}

/ \*\*

\* Preprocesa las variables individuales de elementos de bloque blog. \* /

función template\_preprocess\_single\_blog\_block\_item (variables y $) {$ node = $ variables ['nodo'];

* Crear una matriz renderizable por el título. $ variables ['title'] = array (

'Type #' => 'link', '# title' => $ node-> title,

'# Href' => 'nodo /'. $ Nid nodo->,

);

* Dar formato a la fecha de creación del nodo.

$ variables ['creados'] = $ node-> creado;

$ variables ['fecha'] = format\_date ($ node-> creado, "costumbre", SINGLE\_BLOG\_DATE\_FORMAT);

**[113 ]**



*Tematización un Módulo*



* Cargue el objeto de cuenta con el creador y la tienda del nodo
* en una variable para themer conveniencia.

$ variables ['user'] = user\_load ($ node-> uid); / / Tema del nombre de usuario.

$ variables ['name'] = theme ('username', array (

'Cuenta' => $ variables ['user']));

}

**Los archivos de plantilla**

Con el fin de hacer que los archivos de plantilla tan fácil de entender para los no programadores, la plantilla debe limitarse a las siguientes declaraciones de PHP:

* ? <Php print $ variables; ?>
* <? Php if ([condición]):>
* <? Php elseif ([condición]):>
* <? Php más:>
* ? <Php endif; ?>
* ? <T php print ('cadena'); ?>
* ? <Php hide ($ element ['pieza']); ?>
* ? <Php espectáculo ($ element ['pieza']); ?>
* ? <Php print render ($ element ['pieza']); ?>
* ? <Php print render ($ element); ?>

La impresión y si/elseif/más Snippets PHP deben explicarse por sí mismo. Sin embargo, es importante reiterar que, con el fin de hacer que Drupal multilingüe, nunca debemos incluir una palabra Inglés al descubierto en nuestras plantillas; lugar debemos usar t () en un

? <T php print ('cadena'); ?> fragmento.

Por último, la show (),hide (), Y render () funciones son funciones especiales themer-de conveniencia que sólo deben utilizarse en archivos de plantilla; nunca deben usarse en las funciones de procesamiento previo, funciones temáticos o cualquier otro lugar. render () es básicamente el mismo que el drupal\_render () función que ya hemos aprendido sobre. La hide () función se puede utilizar en un elemento secundario de un arsenal hacer antes en la plantilla antes de que el hacer llamadas de matriz render (); esto evitará que el elemento secundario se incluya con el resto de la matriz hacen cuando se representa. Por ejemplo (de Bartik denode.tpl.php):

<? Php

/ / Nos escondemos los enlaces ahora para que podamos hacer más adelante. ocultar ($ contenido ['links']);

imprimir render ($ contenido);

**[114 ]**



*Capítulo 4*



/ / Sólo mostrar el div contenedor si hay enlaces. $ links = render ($ contenido ['links']);

if ($ links):

?>

<div class="link-wrapper"> <php print $ links?; ?>

</ Div>

Como puede ver, estas funciones de conveniencia que sea más fácil para destrozar, envuelva o quitar piezas de render arrays.

Por lo tanto, vamos a crear nuestra de un solo blog-block-item.tpl.php plantilla:

<? Php / / $ Id $

/ \*\*

* @ File
* HTML para un elemento en la lista de bloques del mismo blog.
* Las variables disponibles:
* - $ clases: Cuerda de clases que se pueden utilizar para
* estilo contextualmente a través de CSS. Puede ser manipulado
* a través de la variable a partir de $ classes\_array funciones de preproceso.
* Los valores por defecto pueden ser uno o más de los siguientes:
* --Blog-bloque de un solo punto: el tipo de plantilla actual,

\* es decir, "hook tematización".

* - $ Fecha: fecha de creación del formato. Funciones de preproceso puede
* reformatearla llamando format\_date () con el deseado
* parámetros en la variable $ creado.
* - $ Title: Una matriz que renderizable que proporciona un título y

enlazar con el nodo.

* - $ Name: nombre de usuario temática de la producción autor nodo desde
* theme\_username ().
* - $ Classes\_array: Array de valores de atributos de clase HTML.

\* Se aplana en una cadena dentro de la

\* clases $ variables.

\*

* Otras variables:
* Se proporcionan las siguientes variables para la información contextual.
* - $ Node: nodo objeto parcial. Contiene datos que pueden no ser seguros.
* - $ Creado: Tiempo del nodo se publicó en formato Unix
* marca de tiempo.

**[115 ]**



*Tematización un Módulo*



* - $ User: El objeto de usuario del autor del nodo.
* @ See template\_preprocess\_single\_blog\_block\_item ()

\* /

?>

<Div class = "<php ​​print $ clases;?>">

? <div class="date"> <php print $ fecha; >: </ Div>

<H4 <php print $ title\_attributes?; ?? >> <Php print render ($ title); > </ H4>

<div class="name">

<? Php print t ('por el nombre de usuario', array ('username' => $ nombre)); > </ Div>

</ Div>

La primera parte de cualquier archivo de plantilla debe ser un gran bloque de documentación que explica todas las variables disponibles para los diseñadores de temas, incluyendo las variables de conveniencia, no sólo las impresas en el interior de nuestra plantilla.

La única variable que no hemos creado explícitamente en nuestra función de preproceso era el $ clases variable. Esta es una cadena que contiene clases CSS útiles que se deben colocar en la parte más externa elemento HTML envoltura en nuestro archivo de plantilla. La $ clases variable se creó por template\_processs () y su correspondiente $ Classes\_array variable se creó por template\_preprocess (). Si queremos añadir clases adicionales a la $ clases cadena, debemos añadir un elemento de matriz a la $ Classes\_array variable durante nuestra función de preproceso y automáticamente se añade a la $ clases cadena antes de llegar al archivo de plantilla.

La cadena que se pasa a la t () función, por! usuario incluye la ! Nombre de usuario testigo para dar contexto a los traductores cuando se trata de traducir "por"; ver el t () Documentación de la API para obtener más información.

La última cosa que debemos hacer, ya que hemos actualizado el marcado HTML devuelto por nuestro bloque, es actualizar también la hoja de estilo:

/ \* $ Id $ \* /

.. Bloque de un solo-blog de contenido ul {padding-left: 0; / \* LTR \* /

}

**[116 ]**



*Capítulo 4*



.. Bloque de un solo-blog de contenido li ul {margin-bottom: 10px; de tipo list-style: ninguno;

}

.. Bloque único blog de la fecha {font-weight: bold;

}

. Bloque de un solo-blog h4 {margin: 0;

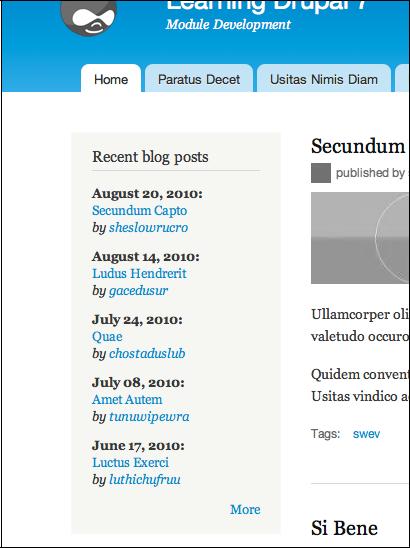
}

.. Bloque de un solo blog de nombre {font-style: italic;

}

¡Felicitaciones! Ya hemos terminado!

Echa un vistazo a nuestra realización se muestra en la siguiente captura de pantalla:



**[117 ]**



*Tematización un Módulo*



**Resumen**

En este capítulo hemos utilizado nuestros conceptos anteriores y construimos una implementación tema de ejemplo con situaciones del mundo real. Además de la revisión de los conceptos de tematización, que debería haber recogido en algunas de las estrategias utilizadas por los desarrolladores de Drupal y aprendido un poco acerca de contribuir con sus experiencias y conocimientos a la comunidad de Drupal.

En el próximo capítulo, usted aprenderá acerca de la construcción de interfaces de administración para su módulo.

Así, mientras que el módulo de Blog Individual utiliza constantes ligeramente cobardes que definieron algunos parámetros no modificables, el módulo en el próximo capítulo tendrá una rica interfaz de administración para permitir que los administradores del sitio para configurar sus ajustes.

**[118 ]**



La construcción de una interfaz de administración

En este capítulo vamos a crear un módulo con una interfaz de administración. . Este módulo se basará en muchos de los conceptos de creación de módulos que se introdujeron en el Capítulo 2 Algunos de los conceptos que cubriremos en este capítulo son:

* Mapeo funciones de Drupal a elementos de menú utilizando hook\_menu ()
* Creación de formularios básicas con el API Form
* Administrar la configuración de Drupal usando variable\_set () y variable\_get ()
* El envío de correo utilizando drupal\_mail () y hook\_mail ()
* Usando el nuevo sistema de fichas de Drupal 7

Una vez finalizado este capítulo, usted debe tener un buen control sobre muchos conceptos que están en la base de casi todos los módulos que va a escribir en el futuro.

**El Usuario Warn módulo**

En este capítulo vamos a crear el usuario Warn módulo. Este módulo permite a los administradores enviar a los usuarios una advertencia por correo electrónico cuando el usuario viola los términos de un sitio de servicio o de otra manera se comporta de una manera que no es apropiada. El Usuario Warn módulo implementará las siguientes características:

* El módulo se exponen los valores de configuración para los administradores del sitio, incluyendo el texto de correo predeterminado
* Este e-mail incluirá fichas de Drupal, que permiten al administrador para reemplazar y / o añadir variables específicas a la dirección de correo
* Los administradores del sitio podrán enviar un correo de usuario a través de una nueva pestaña en su página de perfil de usuario
* Advertencia correos electrónicos se enviarán mediante la aplicación de correo predeterminada de Drupal

*La construcción de una interfaz de administración*



**A partir de nuestro módulo**

Vamos a comenzar como lo hicimos en el capítulo 2, mediante la creación de una nueva carpeta para nuestro módulo llamado user\_warn en el sites / default / modules directorio de nuestra instalación de Drupal. Nosotros a continuación, puede crear un user\_warn.info presentar como se muestra en la siguiente:

; $ Id $

name = Usuario Warn

Descripción = Expone una interfaz de administración para enviar correos electrónicos de advertencia a los usuarios de la conducta.

core = 7.x

paquete = Drupal 7 Archivos de desarrollo [] = user\_warn.module

Usted debe estar bastante familiarizado con esto ahora. También vamos a crear nuestro user\_warn. módulo presentar y agregar una implementación de hook\_help () dejar que los administradores del sitio sabe lo que hace nuestro módulo.

<? Php / / $ Id $

/ \*\*

* @ File
* Usuario Warn archivo de módulo
* Este módulo permite a los administradores del sitio para enviar una advertencia de stock
* e-mail a un usuario o usuarios especificados a través de la interfaz de administración.
* Administradores
* puede configurar la dirección de correo predeterminado, ya la sustitución de tokens.

\* /

/ \*\*

\* Implementar hook\_help (). \* /

función user\_warn\_help ($ ruta, $ arg) {if ($ ruta == 'admin / help # user\_warn') {

volver t ('Warn usuario permite adminitrators para ver el estándar de correo electrónico a los usuarios del sitio para notificarles de la conducta impropia.');

}

}

Esto tampoco es nada nuevo, así que vamos a pasar a las cosas buenas.

**[120 ]**



*Capítulo 5*



**El sistema de menús Drupal**

Sistema de menús de Drupal es engañosamente nombrado. El nombre implica que es responsable de la navegación de su sitio, y si bien esto es cierto que hace mucho más. En su esencia, el sistema de menús es responsable de asignar rutas de Drupal a las funciones que generan los contenidos de la página solicitada. El sistema de menús también se encarga de controlar el acceso a páginas de Drupal, que actúa como uno de los porteros centrales de seguridad de Drupal.

Los desarrolladores de módulos de Drupal pueden asignar rutas de acceso a las funciones de Drupal mediante la implementación dehook\_menu (), Que agrega rutas al sistema de menús, les asigna reglas de acceso, y, opcionalmente, crea los elementos de navegación para ellos.

**Definición de una devolución de llamada de la página con hook\_menu**

Para nuestro módulo tendremos que implementar dos nuevas páginas a una página de configuración para el usuario Advierta módulo, y una ficha en el área de perfil del usuario, donde los administradores pueden seguir para enviar los mensajes de correo electrónico reales a un usuario específico. Estos requerirán cada uno su propiohook\_menu () aplicación tal como se define en el siguiente ejemplo.



Este ejemplo sólo roza la superficie de las opciones disponibles en el sistema de menús. Para más detalles, los desarrolladores deben revisar la documentación de la API en:



http://api.drupal.org/api/function/hook\_menu/7

El ejemplo es el siguiente:

/ \*\*

\* Implementar hook\_menu (). \* /

función user\_warn\_menu () {$ items = array ();

$ items ['admin / config / personas / user\_warn'] = array ('title' => 'El usuario Warn',

'Description' => 'Configuración para el usuario Warn módulo.', 'Página callback' => 'drupal\_get\_form',

La página de argumentos '=> array (' user\_warn\_form '), los argumentos de acceso' => array ('administrar los usuarios'), 'tipo' => MENU\_NORMAL\_ITEM,

);

**[121 ]**



*La construcción de una interfaz de administración*



$ items ['user /% / warn'] = array ('title' => 'Warn', 'description' =>

'Enviar e-mail a un usuario sobre el comportamiento inadecuado del local.', 'Página callback' => 'drupal\_get\_form',

La página de argumentos '=> array (' user\_warn\_confirm\_form ', 1), los argumentos de acceso' => array ('administrar los usuarios'),

'Type' => MENU\_LOCAL\_TASK,

);

return $ artículos;

}

Al igual que muchas implementaciones de hook de Drupal, hook\_menu () devuelve un array asociativo estructurado con información acerca de los elementos del menú se está definiendo. El primer elemento de nuestro ejemplo define la página de configuración del módulo, y el segundo define la ficha de usuario donde los administradores pueden seguir para enviar el correo electrónico real. Veamos el primer tema con más detalle.

Los elementos del menú tienen la forma adecuada de su camino. Se trata de una ruta de Drupal interna sin principio o al final barras. Este camino no sólo define la ubicación de una página, sino también su lugar en la jerarquía del menú, con cada parte de la dirección de ser un niño del pasado. En este ejemplo,personas es un hijo de config que es en sí mismo un niño de administración.

Si una ruta solicitada no existe, Drupal se abrirá camino en la jerarquía hasta que encuentra una página que no existe. Usted puede ver esto en acción mediante la solicitud de

admin / config / personas / xyzzy que muestra la página en admin / config / personas.

Si va a crear un elemento de menú para la administración del sitio debe comenzar conadministración. Esto la coloca en la interfaz administrativa de Drupal y aplica el tema de administración definido por la configuración del sitio.



Ajustes específicos del módulo debe estar siempre presente bajo admin / config. Drupal 7 ofrece varias categorías que los desarrolladores de módulos deben utilizar para organizar mejor sus ajustes de acuerdo a los grupos funcionales de Drupal como People y licencias o creación de contenido.



El valor asociado a esta clave es en sí mismo una matriz asociativa con varias claves que definen la acción que se debe tomar cuando se solicita este URL. Ahora podemos mirar a los de detalle. El primer punto define el título de la página:

'Title' => 'El usuario Warn',

**[122 ]**



*Capítulo 5*



Esto se utiliza en una variedad de contextos de visualización-como encabezamiento de la página, en el código HTML <title> etiqueta y como subtítulo en la interfaz de administración en combinación con la descripción (si está definiendo un elemento de menú de administración).

'Description' => 'Configuración para el usuario Warn módulo.',

La descripción es sólo eso: una descripción de texto de la página ya que este elemento de menú define. Esto debería proporcionar al usuario una información más detallada acerca de las acciones que pueden tomar en esta página. Esta descripción también se utiliza como la etiqueta del título cuando se pasa sobre un vínculo.



Títulos y descripciones de los artículos del menú se pasan a través t () internamente por Drupal, por lo que este es un caso en el que no hay que preocuparse por hacer que nosotros mismos.



Para una página de administración, estos dos elementos definen la forma en su página aparece en

Área de administración de Drupal como se muestra en la siguiente:



Los siguientes dos artículos definen lo que sucederá cuando se solicita su página:

La página de callback '=>' drupal\_get\_form ',

La página de argumentos '=> array (' user\_warn\_form '),

La página de callback ' define la función que se llamará (sin los paréntesis) y La página de argumentos " contiene una serie de argumentos que se pasan a

esta función.

A menudo, se creará una función personalizada que procesa, formatos, y devuelve los datos específicos. Sin embargo, en nuestro caso estamos llamando la función interna de Drupal drupal\_get\_form () que devuelve una matriz según lo definido por API Formulario de Drupal. Como argumento que estamos pasando el ID de forma de la forma que queremos mostrar. Nos sumergiremos en drupal\_get\_form () y la API de la Forma en más detalle más adelante en el capítulo.

El quinto elemento controla quién puede acceder a su página.

los argumentos de acceso '=> array (' administrar los usuarios '),

**[123 ]**

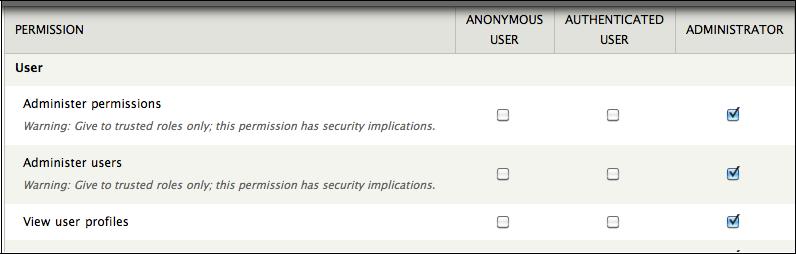


*La construcción de una interfaz de administración*



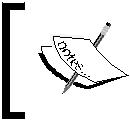
los argumentos de acceso de los toma una matriz que contiene una cadenas de permisos. Cualquier usuario que se le ha asignado uno de estos permisos será capaz de acceder a esta página. ¿A alguien más se le dará una página de acceso denegado. Los permisos se definen por los módulos que utilizan hook\_permission (). Puede ver una lista completa de los permisos definidos en la actualidad en

admin / personas / permisos como se muestra:



Usted puede ver el 'Administrar los usuarios' permiso en la parte inferior de la lista. En el ejemplo anterior, sólo el rol de administrador tiene este permiso, y como resultado sólo aquellos usuarios asignados a esta función podrá acceder a nuestra página.

Tenga en cuenta que los títulos de los permisos aquí no necesariamente coinciden con lo que se necesita para entrar en la matriz de argumentos de acceso. Por desgracia, la única buena manera de encontrar esta información es mediante la comprobación de lahook\_perm () implementación del módulo en cuestión.



Los derechos de acceso, los permisos y la seguridad serán cubiertos con más detalle en*Capítulo 8*,*Drupal permisos y seguridad*.

El punto final define qué tipo de elemento de menú que estamos creando:

'Type' => MENU\_NORMAL\_ITEM,

La 'Tipo' es una máscara de bits de indicadores que describen qué características queremos que nuestra opción de menú para tener (por ejemplo, si es visible en la ruta de navegación). Drupal define más de 20 constantes para los elementos de menú que debe cubrir cualquier desarrollador situación se encuentran in El tipo predeterminado es MENU\_NORMAL\_ITEM, lo que indica que este artículo será visible en el árbol de menús, así como la ruta de navegación.

**[124 ]**



*Capítulo 5*



Esta es toda la información que se necesita para registrar nuestra ruta. Ahora, cuando Drupal recibe una solicitud de este URL, devolverá los resultados de

drupal\_get\_form (user\_warn\_form).



Drupal almacena en caché todo el menú, por lo que los nuevos elementos del menú / actualizados no se reflejará inmediatamente. Para borrar manualmente la caché, visita



**Administración** | Configuración | Desarrollo | Rendimiento y haga clic en Eliminar todos los cachés.

**El uso de comodines en rutas de menús**

Hemos creado un elemento de menú simple, pero a veces simple no vamos a hacer el trabajo. En el módulo de usuario Advierta que queremos tener un elemento de menú que está ligado a la página de perfil de cada usuario individual. Las páginas de perfil en Drupal viven en el caminousuario / <id\_usuario>, Así que ¿cómo

Cómo podemos crear un elemento de menú diferente para cada usuario? Afortunadamente, el sistema de menú nos permite usar comodines cuando definimos nuestras rutas de menús.

Si nos fijamos en el segundo elemento de menú definido en el ejemplo anterior, verá que su definición difiere un poco de nuestro primer ejemplo.

**$ items ['user /% / warn'] = array (**

'Title' => 'Warn',

'Description' => 'Enviar e-mail a un usuario sobre el comportamiento inadecuado del local.',

La página de callback '=>' drupal\_get\_form ',

**La página de argumentos '=> array (' user\_warn\_confirm\_form ', 1),**

los argumentos de acceso '=> array (' administrar los usuarios '),

**'Type' => MENU\_LOCAL\_TASK,**

);

La primera diferencia es que el trayecto se define con % como una de las entradas de ruta. Esto indica un comodín; nada se puede entrar aquí y se mantendrá la jerarquía de la opción del menú. En Drupal, que siempre habrá Identificación de un usuario. Sin embargo, no hay nada que cualquier usuario entre en una URL como usuario / xyzzy / warn o alguna otra cosa potencialmente más dañino. Su código siempre debe ser escrito de una manera tal como para manejar estas eventualidades, por ejemplo, mediante la verificación de que el argumento en realidad se asigna a un usuario de Drupal. Esta sería una buena mejora.

La otra diferencia en este ejemplo es que hemos añadido 1 como un argumento adicional que se pasa a nuestra página de devolución de llamada.

**[125 ]**



*La construcción de una interfaz de administración*

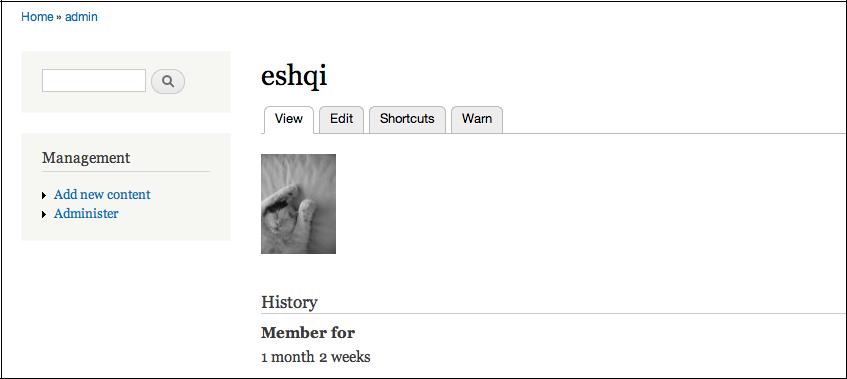


Cada argumento en el camino de un elemento de menú se puede acceder como un argumento que está disponible para ser pasado a nuestra página de devolución de llamada, comenzando con 0 para el argumento de la raíz. Así que aquí la cadena usuario es el elemento 0 y el ID del usuario es el elemento 1. Para utilizar el ID de usuario como argumento de la página de devolución de llamada, que hacen referencia a él por su número. El resultado en este caso es que la identificación del usuario se pasa como un argumento adicional para drupal\_get\_form ().

Tenemos otra diferencia en este segundo elemento del menú:

'Type' => MENU\_LOCAL\_TASK,

Hemos definido nuestro tipo como MENU\_LOCAL\_TASK. Esto le dice a Drupal que nuestra opción de menú se describen las acciones que se pueden realizar con el elemento padre. En este ejemplo,Advertir es una acción que se puede realizar en un usuario. Estos son por lo general representan como un ficha adicional en la página en cuestión, como se puede ver en el ejemplo siguiente pantalla de perfil de usuario:



Una vez definidas las rutas de acceso para nuestras páginas a través de hook\_menu (), Ahora tenemos que construir nuestras formularios.

**API Form**

En el desarrollo web estándar, una de las tareas más tediosas y poco gratificantes es la definición de los formularios HTML y el manejo de sus presentaciones. Diseñe el formulario, crear etiquetas, escribir la función de presentación, averiguar el manejo de errores, y lo peor de todo es que de un sitio a gran parte de este código es repetitivo-que es fundamentalmente el mismo, que sólo difieren en la presentación. API Formulario de Drupal es una poderosa herramienta que permite a los desarrolladores crear formularios y manejar envíos de formularios de forma rápida y sencilla. Esto se hace mediante la definición de conjuntos de elementos de formulario y la creación de la validación y enviar las devoluciones de llamada para el formulario.

**[126 ]**



*Capítulo 5*



En versiones anteriores de Drupal, API formulario se conoce comúnmente como

FAPI. Sin embargo, Drupal 7 tiene ahora tres API que podrían encajar esta API sigla-Form, API campo (que se verá en *Capítulo 6* y *Capítulo 7*) Y API de archivos (lo que usted aprenderá acerca de *Capítulo 11*). Nosotrosse evite el uso de la FAPI sigla completamente, para evitar confusiones, pero todavía se encontrará ampliamente en referencias en línea.



API La forma es también un elemento crucial en la seguridad de Drupal. Proporciona tokens forma única que se verifican en el envío de formularios, evitando ataques de cross-site request forgery, y automáticamente la validación de campos obligatorios, longitudes de campo, y una variedad de otras propiedades de los elementos de formulario.

Mientras API Form es una de las herramientas más útiles y potentes en la caja de herramientas del desarrollador de módulos, también puede ser uno de los más complicados. Una información más detallada más allá de este simple ejemplo se puede encontrar en las siguientes URL:

* API Formulario guía de inicio rápido:

http://api.drupal.org/api/drupal/developer--topics--forms\_api. html / 7

* API Formulario de Referencia Completo:

http://api.drupal.org/api/drupal/developer--topics--forms\_api\_ reference.html / 7

**Uso de drupal\_get\_form ()**

En nuestra primera implementación menú visto anteriormente, definimos la página de devolución de llamada como drupal\_get\_form (). Esta función API Formulario devuelve una matriz estructurada que representa un formulario HTML. Esto se representa por Drupal y presentado como un formulario HTML para el usuario. drupal\_get\_form () toma un ID de formulario como parámetro. Este ID de forma puede ser lo que quieras, pero debe ser único dentro de Drupal. Típicamente será

<module\_name> \_ \_form <description>.

El ID de la forma es también el nombre de la función de devolución de llamada drupal\_get\_form () llamará a construir tu formulario. La función especificada debe devolver un array con el formato correcto que contiene todos los elementos de su formulario necesita.



Dado que el ID formulario también sirve como función de devolución de llamada del formulario, que debe ser un nombre de variable PHP válida. Los espacios y los guiones no están permitidos. Todas las ID de formulario deben ser precedidas por el nombre del módulo seguido de un guión, con el fin de evitar un conflicto de nombres.



**[127 ]**



*La construcción de una interfaz de administración*



Otros parámetros se pueden pasar a drupal\_get\_form () Además de la ID de formulario. Estos parámetros adicionales consiguen simplemente pasan a través de la función de devolución de llamada para su propio uso. Vamos a ver cómo funciona esto más adelante en el capítulo.



En Drupal 6, drupal\_get\_form () devuelto un formulario HTML totalmente rendido. Esto se ha cambiado en Drupal 7 con el fin de permitir una mayor flexibilidad en la tematización y más fácil manipulación formulario. drupal\_get\_form () ahora vuelve una forma de matriz unrendered que habrá de superarse para drupal\_ render () para la salida final. En el ejemplo anterior el sistema de menús maneja el cambio de forma transparente, pero otro código convertido de Drupal 6 puede necesitar ser cambiado.



**La construcción de una función de devolución de llamada de forma**

Para el usuario Warn módulo que necesitamos una forma que permite que el administrador del sitio para acceder a los siguientes elementos:

* Una línea de asunto para nuestro correo electrónico saliente
* El texto de nuestro correo electrónico saliente
* Una casilla de verificación que indica si el administrador debe enviar una copia oculta en los mensajes salientes
* Un botón de envío

Nuestra definición de menú especificado user\_warn\_form como la argumentos página, Por lo que necesitamos para crear esa función y definir nuestra forma dentro de ella.



Esta función toma dos parámetros:$ Form y $ Form\_state. Nosotros no vamos a usar estos parámetros en el contexto de simplemente mostrar un formulario. Pero, para más información sobre su uso consulte la Guía de inicio rápido API Form.



/ \*\*

* Constructor de la forma; Crear y mostrar la configuración del usuario Warn
* forman configuraciones.

\* /

user\_warn\_form función ($ form, & $ form\_state) {/ / Campo de texto para el asunto del correo electrónico. $ Form ['user\_warn\_e-mail\_subject'] = array (

'Type #' => 'campo de texto',

'# Title' => t ('Advertencia asunto del correo electrónico'),

'# Description' => t ('El asunto del e-mail que se envía a los usuarios.'),

**[128 ]**



*Capítulo 5*



'# Size' => 40, '# maxlength' => 120, '# required' => TRUE,

);

/ / Área de texto para el cuerpo del e-mail. $ Form ['user\_warn\_e-mail\_text'] = array (

'# Type' => 'textarea', '# filas' => 10, '# columnas' => 40,

'# Title' => t ('Advertencia texto del correo electrónico'), '# required' => TRUE,

'# Description' => t ('El texto del correo electrónico que se envía a los usuarios.'),

);

/ / Casilla de verificación para indicar si administrador debe enviar una copia oculta en mensajes de correo electrónico. $ Form ['user\_warn\_bcc'] = array (

'Type #' => 'checkbox',

'# Title' => t ('BCC admin en todos los e-mails'),

'# Description' => t ("Indica si el usuario administrador (tal como se establece en la configuración del sitio) debe ser enviado en todos los e-mails de advertencia."),

);

/ / Enviar el botón $ form ['submit'] = array (

'Type #' => 'enviar',

"Valor # '=> t (' Save settings '),

);

devolver $ form;

}

Las propiedades de un forma elemento siempre empieza con un # firmar con el fin de distinguirlos de los campos de formulario anidadas. Para obtener más información, visite el

**API Formulario Guía Acelerada**.

Esto es muy similar a lo que hicimos antes, mientras que la aplicación de hook\_menu (). Creamos un array asociativo con un formato especial y volver a la función de llamada. En este caso, cada elemento de la matriz corresponde a un elemento en el formulario.

Deja mirada en el campo del asunto por primera vez como un ejemplo.

$ Form ['user\_warn\_e-mail\_subject'] = array (

**[129 ]**



*La construcción de una interfaz de administración*



Cada elemento está codificado por una cadena única, que se convertirá en el atributo name del elemento cuando se representa la forma. Este elemento se le asigna un conjunto

de atributos.



Para una matriz completa de todos los elementos de formulario definidos por Drupal, así como las propiedades de cada uno implementos, visite:



http://api.drupal.org/api/drupal/developer--topics-- forms\_api\_reference.html / 7

'Type #' => 'campo de texto',

El primer atributo es 'Type #' que define lo que se representará elemento de formulario. Todos los elementos de formulario HTML estándar tienen tipos, así como algunos elementos de Drupal-específicas definidas. En este caso vamos a crear un campo de texto básico.

'# Title' => t ('Advertencia asunto del correo electrónico'), '# description' => t (

'El asunto del e-mail que se envía a los usuarios.'),

Las siguientes dos atributos, '# Título' y '# Description' definir la etiqueta del elemento y una descripción opcional.

Cualquier atributo que un elemento HTML estándar tiene disponibles como propiedades de la API para formularios o atributos también. Por ejemplo, consulte los siguientes dos líneas de código.

'# Size' => 40, '# maxlength' => 120,

Como era de esperar, estos definen los atributos de tamaño y maxlength de nuestro campo de texto.

Una de las cosas buenas de la API Form es que validará automáticamente muchos de los atributos del elemento cuando se envía el formulario. En este caso Drupal generará un error si el texto se presenta con una longitud superior a maxlength del elemento. Todo esto sucede de forma transparente con ningún código extra del desarrollador.

API Form también añade algunas propiedades de conveniencia para propósitos de validación, como '# Required'.

'# Required' => TRUE,

¿Cuándo '# Required' está ajustado a VERDADERO, Drupal generará un error si el formulario se envía sin un valor en ese elemento. Los campos necesarios están también marcados con un asterisco en sus etiquetas. Una vez más, esto sucede de forma transparente sin ningún código adicional. Drupal incluso resaltar el campo cuando un error se refiere a ella! Esta validación manejo de errores y la forma en la marcha es una de las razones Formulario API es una bendición como para los desarrolladores. Realmente reduce la cantidad de trabajo pesado involucrado en la creación y manejo de formularios HTML.

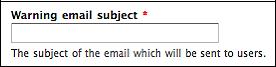
**[130 ]**



*Capítulo 5*



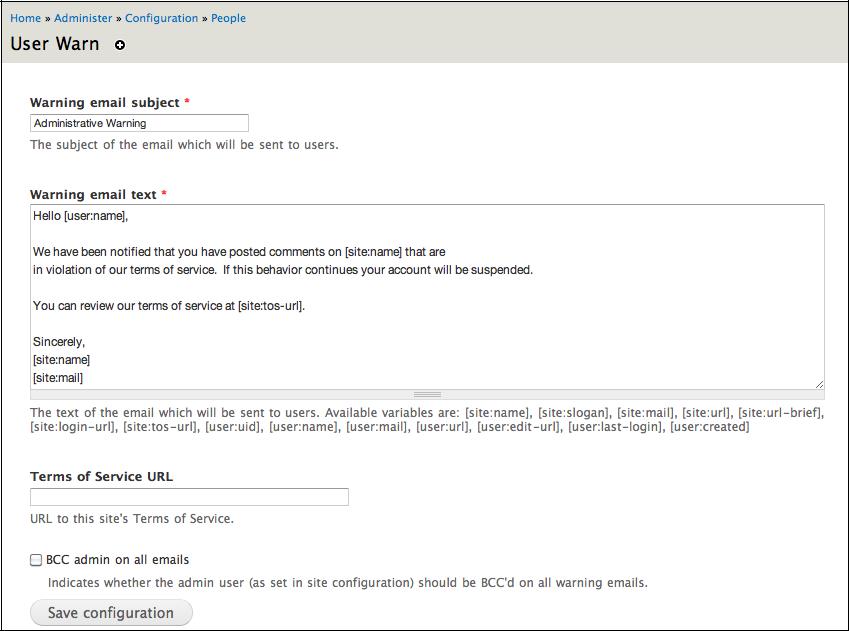
Lo que sigue es cómo aparecerá este campo de texto cuando se representa por Drupal en el tema por defecto admin:



Pasando a los siguientes elementos, se puede ver este patrón se repita. Por ejemplo, el campo cuerpo del correo electrónico de tipo '# Textarea' implementa el '# filas' y

'# columnas' propiedades, al igual que el código HTML correspondiente atributos para un textarea. La elemento casilla de verificación (que indica si el administrador debe enviar un BCC en los correos electrónicos salientes) y botón de envío son igualmente fáciles de operar.

Cuando visitamos la URL que se registró anteriormente (admin / config / personas / user\_warn), Obtenemos el formulario representado como se ve en la siguiente captura de pantalla:



**[131 ]**



*La construcción de una interfaz de administración*



Drupal también ofrece varios elementos de forma personalizada, además de los campos HTML estándar. Se puede ver un ejemplo de uno de estos en el drupal\_get\_form () callback para nuestro segundo elemento del menú:

/ \*\*

\* Formulario constructor; mostrar el formulario de correo electrónico de confirmación. \* /

**función user\_warn\_confirm\_form ($ form, & $ form\_state, $ uid) {$ form ['cuenta'] = array (**

**'Type #' => 'valor', '# value' => user\_load ($ uid),**

**);**

volver confirm\_form ($ form,

t ('¿Está seguro que desea enviar un aviso por e-mail a este usuario?'),

'User /'. $ Uid

t ('Esta acción no se puede deshacer. "), t (' Enviar e-mail '),

t ("Cancelar")

);

}

Nos gustaría volver a esta función con más detalle más adelante en el capítulo, pero por ahora nos centraremos en el área resaltada. Como usted recordará desde antes en este capítulo, se utilizó un comodín en la ruta de menú para tomar el ID del usuario y la contraseña en la página de devolución de llamada. Como se puede ver ahora, esto está siendo pasado como tercer parámetro a nuestra función de devolución de llamada (después de la requerida$ Form y $ Form\_state parámetros). Ahora podemos utilizar este ID para recuperar datos sobre el usuario para su uso futuro.

También, se puede ver que estamos definiendo un nuevo elemento de formulario de tipo "Valor". El elemento de valor es similar a los campos ocultos HTML con dos ventajas distintas.

En primer lugar, los elementos de valor pueden contener los datos que desee en lugar de sólo cadenas. Matrices de objetos, o cualquier otra estructura de datos compleja se pueden almacenar y se pasan en un elemento de valor.

La segunda ventaja es que los elementos de valor no se imprimen al navegador en el código HTML. Esto puede mejorar la seguridad de sus datos ya que evita que los usuarios vean y / o modificarlo en una instancia local.

En este ejemplo de código estamos asignando el elemento de valor"Cuenta" con el valor de un objeto de usuario de Drupal. Este objeto se transmite cuando se envía el formulario, y la función de recepción podrá utilizarlo cuando sea necesario. Elementos de valor son muy útiles y los desarrolladores siempre deben considerar el uso de ellos en lugares donde de otro modo utilizar campos ocultos.

**[132 ]**



*Capítulo 5*



Drupal ofrece también un elemento API Formulario de tipo 'Oculto', Deben desarrolladores prefieren usarlo.

API Form hace que la construcción de formularios increíblemente simple, pero ahora mismo este formulario tiene dos problemas. En primer lugar, el módulo debe proporcionar algunos ajustes predeterminados razonables para los administradores de sistemas. En segundo lugar, al momento de enviar el formulario, ninguno de los datos enviados en realidad maneja de alguna manera. Echemos un breve desvío de manejo de formularios y ver cómo gestiona Drupal los datos del sistema persistentes.

**La gestión de datos persistente**

Drupal proporciona un mecanismo mediante el cual los datos, que debe persistir semipermanente (como la configuración del sistema), se pueden guardar y recuperar. Estos artículos se refieren tanto confusa como "variables" (nos referiremos a ellos específicamente como variables persistentes a partir de ahora para evitar confusiones). Las variables persistentes se almacenan en una tabla de base de datos, introducido por un nombre único proporcionado por el módulo que las implementa.

Las variables persistentes se guardan con variable\_set (), Y recuperados utilizando variable\_get (). Estas variables pueden ser cualquier tipo de datos que un desarrollador necesita, ser es una simple cadena, una matriz asociativa, o un objeto PHP. La API de Drupal para ajustar / conseguir que se encarga de toda la serialización / unserialization que es necesario detrás de las escenas.

variable\_get ()También puede proporcionar un valor por defecto, que es útil para situaciones donde se necesita una variable que aún no se ha establecido, por ejemplo, después de un módulo se instala por primera vez. Podemos usar esto a nuestro favor en nuestro formulario de configuración, como se muestra en el siguiente fragmento:

$ Form ['user\_warn\_e-mail\_subject'] = array ('# type' => 'campo de texto',

**'# Default\_value' => variable\_get ('user\_warn\_e-mail\_subject', 'Advertencia Administrativa'),**

'# Title' => t ('Advertencia asunto del correo electrónico'), '# size' => 40,

'# Maxlength' => 120, '# required' => true, '# description' => t (

'El asunto del e-mail que se envía a los usuarios.'),

);

**[133 ]**



*La construcción de una interfaz de administración*



Este es el mismo elemento API Forma hemos creado anteriormente, pero con una nueva línea añadida. Esta línea añade el '# Default\_value' propiedad del elemento de formulario. Esta propiedad indica la forma qué datos debe contener el elemento cuando el formulario se carga por primera vez.

Estamos asignando esta propiedad de los resultados de una llamada avariable\_get () usando dos parámetros. El primer parámetro es la clave única asociada con estos datos. Es una práctica común para dar una variable persistente el mismo nombre que el elemento de formulario que está asociado, y lo hemos hecho aquí.



Al igual que los elementos de menú, las variables persistentes se almacenan en caché por Drupal, por lo que a menudo tendrá que borrar las cachés después de modificarlos.

El segundo parámetro especifica los datos que deben ser devuelto si esta variable no se ha definido de forma explícita. En este ejemplo hemos establecido que para ser la cadena 'Atención Administrativa'. Si esta variable se ha establecido explícitamente en algún momento previamente, luego de que los datos serán devueltas porvariable\_get () en su lugar. De lo contrario, se devolverá el valor por defecto.

Ahora, la primera vez que se carga el formulario, cualquier dato que esté en la variable persistente

'User\_warn\_e-mail\_subject' se establece como el valor de la forma de asunto del correo electrónico elemento. También podemos hacer esto a nuestros otros elementos de formulario si lo deseas. Al final de nuestra función será la siguiente. Tenga en cuenta que también hemos añadido una constante que contiene el texto por defecto de nuestro e-mail. La eliminación de este gran bloque de texto de nuestra definición de la matriz hace que nuestro código sea más legible y fácil de mantener en el futuro.

Constantes de Drupal se definen normalmente en la parte superior de un . Módulo archivo, pero en aras de la claridad este ejemplo incluye la definición constante con la función:

define ('USER\_WARN\_MAIL\_TEXT',

**'Hola,**

**Se nos ha notificado que usted ha publicado comentarios en nuestro sitio que están en violación de nuestros términos de servicio. Si este comportamiento continúa se suspenderá su cuenta.**

**Atentamente,**

**El personal administrativo ');**

user\_warn\_form función ($ form, & $ form\_state) {$ form = array ();

/ / Campo de texto para el asunto del correo electrónico. $ Form ['user\_warn\_e-mail\_subject'] = array (

'Type #' => 'campo de texto',

**[134 ]**



*Capítulo 5*



**'# Default\_value' => variable\_get ('user\_warn\_e-mail\_subject', 'Advertencia Administrativa'),**

'# Title' => t ('Advertencia asunto del correo electrónico'), '# size' => 40,

'# Maxlength' => 120, '# required' => true, '# description' => t (

'El asunto del e-mail que se envía a los usuarios.'),

);

/ / Área de texto para el cuerpo del e-mail. $ Form ['user\_warn\_e-mail\_text'] = array (

'# Type' => 'textarea', '# filas' => 10, '# columnas' => 40,

**'# Default\_value' => variable\_get ('user\_warn\_e-mail\_text', USER\_WARN\_MAIL\_TEXT),**

'# Title' => t ('Advertencia texto del correo electrónico'), '# required' => TRUE,

'# Description' => t (

'El texto del correo electrónico que se envía a los usuarios. '),

);

* Casilla de verificación para indicar si administrador debe enviar una copia oculta
* en los e-mails.

$ Form ['user\_warn\_bcc'] = array ('# type' => 'checkbox',

**'# Default\_value' => variable\_get ('user\_warn\_bcc', false),**

'# Title' => t ('BCC admin en todos los e-mails'),

'# Description' => t ('Indica si el usuario administrador (tal como se establece en la configuración del sitio) debe ser BCC \' d en todos los e-mails de advertencia. '),

);

/ / Enviar el botón $ form ['submit'] = array (

'Type #' => 'enviar',

"Valor # '=> t (' Save settings '),

);

devolver $ form;

}

Las variables persistentes son una excelente manera de almacenar la configuración del módulo y otra información configurable por el usuario. Sin embargo, de haber dado a nuestros valores de configuración por defecto razonables, todavía nos queda la cuestión de cómo guardar los cambios en estos valores predeterminados.

**[135 ]**



*La construcción de una interfaz de administración*



**Proceso de envío de formulario**

Cuando se envía un formulario HTML construido con API Form, Drupal busca dos funciones: una función de validación expresamente mencionados y una función de enviar. Estas funciones se nombran mediante la adopción de la forma y el ID añadiendo ya sea\_Validate () o \_submit () dependiendo de la función que se está escribiendo.

La función de validación hace una validación adicional más allá de lo que ofrece Drupal. Por ejemplo, si usted quiere comprobar si un código postal válido. Incluso si la validación falla en un elemento, todas las funciones de validación se sigue llamando, así que Drupal puede devolver varios errores en un solo formulario. Sin embargo, si cualquier función de validación falla, la ejecución no procede a la función de enviar.



Validar funciones son opcionales. Si usted no necesita ninguna validación adicional, entonces usted no tiene que escribir uno. En este caso, Drupal se acaba de hacer su validación por defecto. Para obtener más información sobre cómo



escribir validar las funciones, consulte la documentación de la API de formulario en los enlaces indicados anteriormente en este capítulo.

Una vez que la forma supera la validación, la función de presentar se llama. Esta es la configuración, el envío de correo electrónico, y la creación de contenido, entre otras cosas que el trabajo real se realiza-ahorro. La forma de presentar la función es uno de los principales caballos de batalla de módulos de Drupal.

Como escritor módulo va a gastar una enorme cantidad de tiempo escribiendo presentar funciones y código de soporte para presentar funciones. Esto es bueno, porque significa que usted está pasando tiempo en el código que es único a su proyecto, y no volver a crear la rueda cada vez que quiera convertir un campo rojo porque un campo requerido no se encuentra.

Así que vamos a aplicar esto al usuario Advierta formulario de configuración. El ID de formulario para el formulario de configuración es user\_warn\_form, Por lo que nuestra función presentará será nombrado

user\_warn\_form\_submit ().

Formulario de presentar funciones toman dos argumentos.$ Form es la matriz API formulario original para el formulario enviado, y $ Form\_state es una matriz que contiene información asociativa específica a esta presentación. En particular, $ Form\_state ['valores'] contiene todos los valores de los formularios presentados con llave en sus propiedades del nombre. En general, $ Form\_state ['valores'] es lo único que tendrá que preocuparse de en validate y presentar funciones.

/ \*\*

\* Guardar los ajustes de configuración de usuario Advierten módulo. \* /

user\_warn\_form\_submit función ($ form, & $ form\_state) {

**[136 ]**



*Capítulo 5*



variable\_set ('user\_warn\_e-mail\_subject', $ form\_state ['valores'] ['user\_warn\_e-mail\_subject']);

variable\_set ('user\_warn\_e-mail\_text', $ form\_state ['valores'] ['user\_warn\_e-mail\_text']);

variable\_set ('user\_warn\_bcc', $ form\_state ['valores'] ['user\_warn\_bcc']);

drupal\_set\_message (t ('Los ajustes se han guardado'));

}

Después de todo eso, nuestra función de presentar es bastante simple. Estamos salvando nuestras datos presentados utilizando variable\_set (), A continuación, establecer un simple mensaje que indica que los valores se han guardado correctamente. Nuestras necesidades de validación son manejados por Drupal incorporado en la validación de formularios, por lo que ni siquiera necesitamos una función de validación para estos datos.



La función drupal\_set\_message () establece un mensaje simple que se muestra en un área específica en la parte superior de una página de Drupal. Para obtener más



detalles ver http://api.drupal.org/api/function/drupal\_ set\_message / 7.

Después se envía un formulario, se volverá a cargar la página de envío de formulario. Puesto que hemos salvado los nuevos datos en nuestras variables persistentes a través de la función de enviar y, dado que la forma está precargado con datos por defecto sobre la base de los datos de las variables, ahora deberíamos ser capaces de presentar nuevos datos de estos elementos y verlos reflejados después de que la solicitud ha sido enviada.



Si quieres Drupal para redirigir a una página diferente después de la presentación del formulario, puede establecer $ Form\_state ['redirección'] a la trayectoria deseada en su función de enviar. Si esto no funciona, compruebe si se ha especificado para $ Form\_state que se pase por referencia, añadiendo un símbolo de unión a la misma en su firma de la función.



API formulario nos proporciona una gran cantidad de poder con una muy pequeña cantidad de trabajo, pero Drupal ofrece algunos atajos que hacen que las formularios comunes aún más fácil.

**[137 ]**



*La construcción de una interfaz de administración*



**Un acceso directo para la configuración del sistema**

La necesidad de guardar los ajustes de configuración en las variables persistentes a través de un formulario estándar es bastante común. Afortunadamente, una vez más, Drupal nos ha dado un poco de magia para simplificar esta tarea. Esa magia es la system\_settings\_form () función.

Cuando se pasa una forma array API formulario estándar a través de esta función y devolver los resultados, se obtiene varios beneficios. Echa un vistazo a la siguiente versión modificada

de user\_warn\_form ():

/ \*\*

\* Formulario constructor; Construir el usuario Warn forma ajustes. \* /

user\_warn\_form función ($ form, & $ form\_state) {

/ / Campo de texto para el asunto del correo electrónico. $ Form ['user\_warn\_e-mail\_subject'] = array (

'Type #' => 'campo de texto',

**'# Default\_value' => 'Atención Administrativa',**

'# Title' => t ('Advertencia asunto del correo electrónico'), '# size' => 40,

'# Maxlength' => 120, '# required' => true, '# description' => t (

'El asunto del e-mail que se envía a los usuarios.'),

);

* Área de texto para el cuerpo del e-mail. $ Form ['user\_warn\_e-mail\_text'] = array (

'# Type' => 'textarea', '# filas' => 10, '# columnas' => 40,

**'# Default\_value' => USER\_WARN\_MAIL\_TEXT,**

'# Title' => t ('Advertencia texto del correo electrónico'), '# required' => TRUE,

'# Description' => t (

'El texto del correo electrónico que se envía a los usuarios.'),

);

* Casilla de verificación para indicar si administrador debe enviar una copia oculta
* en los e-mails.

$ Form ['user\_warn\_bcc'] = array ('# type' => 'checkbox',

**'# Default\_value' => FALSE,**

'# Title' => t ('BCC admin en todos los e-mails'),

**[138 ]**



*Capítulo 5*



'# Description' => t ('Indica si el usuario administrador (tal como se establece en la configuración del sitio) debe ser BCC \' d en todos los e-mails de advertencia. '),

);

**volver system\_settings\_form (forma $);**

}

La primera cosa que usted notará es que ya no tenemos un elemento de botón de enviar. Esto es así porque system\_settings\_form () añade una en por nosotros de forma automática. Se pone el

etiqueta 'Guardar opciones'.

Además, system\_settings\_form () Contiene su propio manejador de envío personalizada system\_settings\_form\_submit (), Que guarda automáticamente todos los elementos de formulario en variables persistentes con el mismo nombre. Usted no tiene que escribir una función presente en todos, Drupal se encarga de todo detrás de las escenas.

Puede parecer una tontería usar una función de la API para algo tan simple como añadir un botón de enviar y automatizar el manejo de las variables persistentes. Sin embargo, menos código que tiene que escribir los menos errores usted introduce. Con tan sólo alrededor de 30 líneas de código, ahora tenemos una manera plenamente funcional con una amplia validación, ajustes predeterminados personalizables, y la capacidad para que los usuarios cambien la configuración predeterminada como lo deseen.

Habiendo establecido formulario de configuración de nuestro módulo, ahora tenemos que añadir una función que permite a los administradores enviar realmente este e-mail a los usuarios.

**Un acceso directo a los formularios de confirmación**

Anteriormente en este capítulo, hemos añadido una pestaña 'Warn' para las páginas de perfil de usuario. Los administradores de sistemas deben ser capaces de hacer clic en esta ficha para enviar el correo electrónico de advertencia a los usuarios.

Sin embargo, sería bueno si pudiéramos añadir un paso de confirmación aquí, para evitar que los correos electrónicos que se envíen inadvertidamente.

Esta es otra situación en la que Drupal ofrece una función de acceso directo conveniente. Vamos a revisar la función de devolución de llamada que vimos anteriormente al discutir"Valor" formar elementos.

/ \*\*

\* Formulario constructor; mostrar el formulario de correo electrónico de confirmación. \* /

función user\_warn\_confirm\_form ($ form, & $ form\_state, $ uid)

{

$ Form ['cuenta'] = array ('# type' => 'valor', '# value' => user\_load ($ uid),

);

**[139 ]**



*La construcción de una interfaz de administración*



**volver confirm\_form ($ form,**

**t (**

**'¿Está seguro que desea enviar un aviso por e-mail a este usuario?'), 'User /'. $ Uid**

**t ('Esta acción no se puede deshacer. "), t (' Enviar e-mail '),**

**t ("Cancelar")**

**);**

}

La confirm\_form () función permite a los desarrolladores crear fácilmente formularios de confirmación asociados con acciones específicas. Lleva siete argumentos, que parece intimidante, pero en realidad son bastante intuitivo.

El primer argumento contiene elementos de formulario que queremos tener fusionaron en el formulario de confirmación resultante. Como vimos anteriormente, hemos creado un elemento de valor que contiene un objeto de cuenta de usuario. Necesitamos que esto se transmite a presentar la función de la forma, por lo que nos propusimos que se agregue con todos los demás elementos queconfirm\_form () crea por sí solo.

El segundo argumento especifica la pregunta que quieras hacer cuando el usuario se presenta con la opción de confirmación. Esto es bastante sencillo y tenemos un mensaje apropiado allí.

El tercer argumento indica qué URL el usuario debe enviarse a si el usuario cancela el formulario. Por lo general, esta será una ruta interna Drupal sin principio o al final barras. Por lo general los administradores del sitio podrán acceder a esta página desde una página de perfil de usuario, por lo que es conveniente que cuando se cancela esta forma los administradores se devuelven a esta página de perfil.

Los últimos tres argumentos especifican distintos epígrafes utilizados en el formulario. Son la descripción de texto adicional que se muestra arriba del botón de confirmación, el texto del botón de confirmación, y el texto del enlace cancelar. Todos estos mensajes son opcionales, y Drupal utilizarán los parámetros por defecto, si no se cambia aquí de forma explícita.

El código anterior muestra lo siguiente:



**[140 ]**



*Capítulo 5*



Formularios generados por confirm\_form () sólo llamar a sus funciones presentar si la forma se confirma en realidad, lo que los desarrolladores no tienen que comprobar por sí mismos. Si se cancela el formulario, el usuario simplemente se redirige a la dirección URL especificada en la llamada de función.

Ahora hemos conseguido una introducción bastante completa para API Formulario de Drupal. Podemos crear formularios a partir de cero, escribir, validar y presentar los manipuladores, y utilizar algunas de las funciones internas de Drupal para crear tipos de formularios comunes. Estamos a dos tercios del camino en este capítulo y todavía no hemos tocado el propósito de enviar central del módulo de un e-mail a un usuario!

**Envío de correo con drupal\_mail () y hook\_mail ()**

Drupal implementa un sistema de plantillas de correo electrónico personalizado. Inicialmente puede parecer que este sistema es demasiado complicado, pero permite una enorme cantidad de flexibilidad para los desarrolladores de módulos.

Envío de e-mail en Drupal es un proceso de varios pasos:

1. drupal\_mail ()se llama, especificando qué es el correo se está enviando y qué opciones son exclusivas de este mensaje específico (dirección de correo electrónico del destinatario, el idioma que el correo debe ser enviado en, y así sucesivamente).
2. Drupal entonces construye un mensaje de correo electrónico con encabezamientos estándar combinadas con la información remitida adrupal\_mail ().
3. La hook\_mail () aplicación estipulado en el drupal\_mail () se llama. Aquí es donde se añaden el asunto y el cuerpo del correo.
4. El conjunto electrónico completo compuesto se pasa a hook\_mail\_alter (), Lo que permite que otros módulos lo modifican (por ejemplo, para agregar una firma común a todos los correos electrónicos salientes.).
5. El correo se pasa a drupal\_send\_mail () para la entrega.

Es un proceso bastante largo sólo para el envío de un simple e-mail! Sin embargo, en la mayoría de los casos los desarrolladores sólo tendrán que preocuparse de dos de los anteriores pasos insultosdrupal\_mail () y aplicar hook\_mail ().

**[141 ]**



*La construcción de una interfaz de administración*



**Configuración de correo PHP**

Para que Drupal para enviar un correo electrónico, el servidor debe estar configurado para que PHP de mail () función se activará. Normalmente, esto implica la instalación y configuración de un servidor de correo local. Los proveedores de alojamiento más comunes ya están configurados correctamente para hacer esto, pero si usted está en un VPS u otro servidor a la medida puede que tenga que manejar por su cuenta. Este proceso varía enormemente dependiendo de su sistema operativo y una variedad de otros factores. Búsqueda de la configuración del correo php en Google lo más probable es que comience en la dirección correcta.



**Drupal\_mail Calling ()**

La siguiente es la función que es llamada cuando se envía el formulario de confirmación (que indica que debemos de hecho enviar una advertencia por correo electrónico al usuario en cuestión).

/ \*\*

\* Enviar un correo electrónico de advertencia al usuario especificado. \* /

user\_warn\_confirm\_form\_submit función ($ form, & $ form\_state) {$ representan = $ form\_state ['valores'] ['cuenta'];

**drupal\_mail ('user\_warn', 'avisar', cuenta de correo-> $,**

**user\_preferred\_language ($ cuenta), $ form\_state ['valores'], variable\_get ('site\_mail', NULL), TRUE**

**);**

}

Como se puede ver, drupal\_mail () requiere que pasamos un poco de información. Echemos un vistazo a cada uno de estos argumentos en detalle:

* El primer argumento indica qué módulo debe invocar hook\_mail () para enviar este mensaje. Estamos estableciendo este a 'User\_warn' ya que estamos haciendo nuestra propia hook\_mail () aplicación. Sin embargo, puede enviar un correo electrónico implementado por otro módulo, si es necesario. Vamos a mirar el user\_warn\_mail () aplicación en un momento.

**[142 ]**



*Capítulo 5*



* El segundo argumento, advertir, Es una clave que se pasa a hook\_mail (). Cualquier hook\_mail () implementación puede definir varios mensajes de correo electrónico, únicamente identificado por una clave de texto. (Módulo de usuario de Drupal implementa dieciocho (!) Para cosas como la confirmación de cuenta y contraseñas olvidadas). Se especifica que el correo específico que queremos enviar con este parámetro.
* El tercer argumento contiene la dirección del destinatario. Sacamos esto de objeto de usuario para el usuario cuyo perfil que visitamos, tal como fue aprobado por el formulario de confirmación de arriba.
* El cuarto argumento especifica el idioma del correo debe ser enviado pulg Esto es importante porque los usuarios individuales pueden especificar una preferencia de idioma que es diferente del idioma por defecto del sitio. Debemos honrar esta elección

si es posible, al enviar nuestro e-mail a este usuario. La user\_preferred\_ idioma () función facilita esta tarea mediante la adopción de un objeto de usuario y volviendo la elección del idioma del usuario.

* El quinto argumento es un array asociativo de parámetros que se pasan al hook\_mail (). Cualquier información personalizada necesaria para construir el e-mail debe ser poner aquí. En nuestro caso, toda la información personalizada que necesitamos para construir el e-mail ya está en los datos presentados en el formulario de confirmación, por lo que se acaba de utilizar$ Form\_state ['valores'] aquí.
* El sexto argumento contiene la dirección de correo electrónico de quien se debe enviar este correo. La primera vez que instaló Drupal había que especificar una dirección de correo electrónico administrativa. Esta dirección ya se está usando como fuente para otros e-mails del sistema (como verificación de la cuenta) así que tiene sentido para usarlo como nuestro remitente del correo electrónico también. El correo electrónico se almacena como una variable persistente con

la tecla 'Site\_mail', Por lo que podemos fácilmente agarrarla utilizando variable\_get () como se explicó anteriormente en el capítulo.

* Finalmente, la última variable indica si el correo en realidad se debe enviar. Esto no es ninguna sorpresa saber que un mensaje de correo en Drupal se basa en una matriz asociativa especialmente estructurado. Al final del proceso de construcción electrónico, esta matriz está típicamente pasa a la funcióndrupal\_mail\_send () que se encarga de la entrega real del correo. Sin embargo, mediante el establecimiento de este parámetro para FALSO,drupal\_mail () pasará por alto la etapa de suministro, lo que le permite tomar la matriz estructurada vuelve y manejar la entrega de sí mismo.

**[143 ]**



*La construcción de una interfaz de administración*



**Hook\_mail Implementación ()**

En la última sección, cuando llamamos drupal\_mail (), Indicamos que debería invocar hook\_mail () en el user\_warn módulo. Esto significa que Drupal se busca una función llamada user\_warn\_mail (). Esta función es la siguiente:

/ \*\*

\* Implementar hook\_mail (). \* /

función user\_warn\_mail ($ key, & $ mensaje, $ params) {switch ($ key) {

caso "advertir":

$ Cuenta = $ params ['cuenta'];

$ Subject = variable\_get ('user\_warn\_e-mail\_subject', 'Advertencia Administrativa');

$ Body = variable\_get ('user\_warn\_e-mail\_text', 'Usted \' ve sido advertido '!);

if (variable\_get ('user\_warn\_bcc', false)) {$ ADMIN\_MAIL = variable\_get ('site\_mail', NULL); $ Mensaje ['headers'] ['CCO'] = $ ADMIN\_MAIL;

}

$ Mensaje ['a'] = $ mail account->; $ Mensaje ['subject'] = $ asunto; $ Mensaje ['cuerpo'] [] = $ cuerpo; romper;

}

}

Como se puede ver la función anterior recibe tres argumentos:

* La clave que aprobamos en el parámetro de dos de nuestro llamado a drupal\_mail (), Lo que indica cuál es el mensaje que debe enviarse.
* El conjunto estructurado que Drupal crea para representar un mensaje de correo electrónico. En este punto, esta matriz ya ha sido rellenado con la información de encabezado predeterminado de correo.
* Los datos pasábamos de drupal\_mail () en el $ params argumento (en este caso, objeto de cuenta de un usuario.)

**[144 ]**



*Capítulo 5*



Como se discutió anteriormente, es posible que los hook\_mail () para manejar múltiples diferentes mensajes de correo electrónico como se indica por la tecla pasado desde drupal\_mail (). A pesar de que sólo estamos enviando un e-mail con una clave de 'Warn', Todavía nos ponemos en una estructura de switch / case para hacer más fácil el manejo mas mails más tarde si es necesario.

Ahora podemos seguir adelante con el verdadero propósito de nuestras hook detalles de implementación-añadiendo a la $ Mensaje matriz que son exclusivos de nuestro correo. Normalmente, este es el asunto, el cuerpo, y cualquiera de las cabeceras adicionales que pudiéramos necesitar.

Asunto y el texto de Nuestro correo electrónico se han establecido a través de la página de configuración del módulo, por lo que les recuperar a través de variable\_get () y los puso a la $ Mensaje ['subject'] y $ Mensaje ['cuerpo] propiedades aquí.



Tenga en cuenta que no aprobamos el sujeto y las cuerdas a través del cuerpo t () como lo hemos hecho en otros contextos. Estas cadenas son suministrados por el administrador del sitio a través del formulario de configuración de usuario Warn del módulo, y como tales no son traducibles. Textos de sistema Sólo Hardcoded necesitan ser pasado a través de t ().



La otra cosa que necesitamos hacer es CCO el administrador del sitio si ese valor de configuración se ha establecido.

if (variable\_get ('user\_warn\_bcc', false)) {$ ADMIN\_MAIL = variable\_get ('site\_mail', NULL); $ Mensaje ['headers'] ['CCO'] = $ ADMIN\_MAIL;

}

Al igual que con los demás ajustes de configuración, podremos recuperarla usando variable\_get (). Si es

VERDADERO, Entonces tenemos que configurar el administrador del sitio para ser CCO se remitan. A diferencia del destinatario de correo electrónico, Cc y CCO se establecen mediante la adición de cabeceras a la $ Mensaje array. Los encabezados son en sí mismos una matriz asociativa celebrada bajo la 'headers' clave, y tenemos que añadir una nueva cabecera con la tecla 'CCO'. Asignamos este al correo electrónico del sitio de administración de la misma manera como lo hicimos endrupal\_mail () mientras que la creación del remitente del correo.

Esto es todo lo que tenemos que hacer! $ Mensaje se pasa por referencia, por lo que ni siquiera hay que devolverlo. Drupal se acaba de proceder desde aquí. Después de otros módulos tienen la oportunidad de alterar el correo a través de hook\_mail\_alter (), La $ Mensaje matriz se pasa al drupal\_mail\_system () donde tendrá el formato del mensaje de correo final y entrega (si se ha especificado esta opción cuando llamaste drupal\_mail ()).

**[145 ]**



*La construcción de una interfaz de administración*



**La depuración de problemas con el correo**

Hay una variedad de razones por las que un e-mail no fuera entregado. Si la dirección del destinatario no existe o hay otro problema en el extremo receptor, el correo se recuperó para la dirección de correo electrónico especificada en el sexto argumento de drupal\_mail () (El administrador del sitio en este ejemplo.). En el caso de un local de mal configurado

sistema, usted puede ser capaz de encontrar más información en los registros de errores de PHP. La



Módulo Redireccionamiento de correo puede ser útil si usted está teniendo problemas para enviar el correo en el servidor de desarrollo:

http://drupal.org/project/reroute\_e-mail

Todo esto es bueno, y de hecho tenemos un módulo totalmente funcional ahora. Sin embargo, hay una cuestión más que debemos mirar a abordar.

**El sistema de fichas**

Sería bueno si pudiéramos incluir alguna información personalizada en el texto electrónico y sin tener que codificar en forma de configuración del módulo. Por ejemplo, deberíamos ser capaces de incluir el inicio de sesión del usuario que se está advertido, o el nombre del administrador del sitio. Esto nos lleva a nuestro último tema, el uso de sistema de fichas de Drupal.

**¿Cuáles son los tokens?**

Un símbolo es un pequeño fragmento de texto que se puede colocar en una pieza de texto mediante el uso de un marcador de posición. Cuando el texto se pasa a través de la función token\_replace (), A continuación, las fichas son reemplazadas con la información apropiada. Tokens permiten a los usuarios incluir datos que podrían cambiar en los bloques de texto, sin tener que volver atrás y cambiarlo por todas partes que están referenciados.



En versiones anteriores de Drupal, tokens se implementan utilizando el módulo contribuido con nombre, como era de esperar, Token. Esta funcionalidad resultó ser tan popular y ampliamente utilizado que estaba incluido en el núcleo de Drupal 7.



Un símbolo de la muestra es [Site: name]. Cuando el texto que contiene esta muestra se pasa a través token\_replace (), Es reemplazado con el nombre de su sitio como se define en **Casa** |

**Administrar** | Configuración | Información del sitio. Si cambia el nombre de su sitio, a continuación, en el futuro todo el texto que contiene esta muestra reflejará este cambio. Drupal expone una serie de fichas que contienen información sobre los usuarios, nodos, la configuración de todo el sitio, y mucho más.

**[146 ]**



*Capítulo 5*



Tokens también pueden ser 'chained'-un testigo puede referirse a otra ficha que se puede referir a otro más. Como un ejemplo, el token [Nodo: autor] contiene el nombre del autor de un nodo, y el token [Usuario: e-mail] contiene la dirección de correo electrónico de un

dado de usuario. Para recuperar la dirección de correo electrónico del autor de un nodo, puede encadenar los dos juntos con el token [Nodo: autor: e-mail].



Los desarrolladores de módulos también pueden exponer sus propios símbolos para otros desarrolladores de módulos para aprovechar. Para obtener más información sobre cómo exponer las fichas en su módulo, consulte los siguientes sitios:



http://api.drupal.org/api/function/hook\_token\_info/7 http://api.drupal.org/api/function/hook\_tokens/7

Sistema de fichas de Drupal es muy flexible y evita que los constructores de sitio y los desarrolladores de tener que reemplazar la información en el texto del sitio cada vez que cambie. Así que vamos a ver cómo podemos usar tokens en nuestro módulo.



**¿Cómo sabemos qué fichas están disponibles?**

Drupal 7 no incluye una interfaz de usuario para navegar por las fichas disponibles, sin embargo, el módulo Token contribuido implementa una muy buena forma de árbol JavaScript del navegador para ellos. Puede descargarlo e instalarlo desde el siguiente sitio:



http://drupal.org/project/token

Además los desarrolladores de módulos pueden utilizar la función token\_info () para obtener una matriz estructurada que contiene todas las fichas en el sistema. Esto se puede analizar y / o se muestra si lo deseas.

**La implementación de fichas en su texto**

El lugar obvio donde Warn usuario podría usar tokens está en el texto de los mensajes de correo electrónico salientes. Vamos a ampliar el texto muy simple defecto incluimos más arriba, y también lo puso en una constante, para facilitar la legibilidad y facilidad de mantenimiento del módulo. Para ello será necesario la actualización de una parte del código anterior, pero en el futuro, sólo tendrá que cambiar esta información en un solo lugar.

define ('USER\_WARN\_MAIL\_TEXT', 'Hola [usuario: nombre],

Se nos ha notificado que usted ha publicado comentarios en [site: name] que están en violación de nuestros términos de servicio. Si este comportamiento continúa se suspenderá su cuenta.

Atentamente,

[Site: name] ');

**[147 ]**



*La construcción de una interfaz de administración*



Este texto contiene tres fichas:

* [Site: name]: El nombre del sitio como se ha descrito anteriormente
* [Site: mail]: La dirección de correo electrónico administrativa (este es el mismo e-mail dirección devuelta por variable\_get ('El sitio electrónico')
* [Usuario: Nombre]: El nombre de inicio de sesión de un usuario especificado

Con el fin de hacer que esto funcione, tenemos que poner en práctica token\_replace () en nuestra hook\_mail () aplicación como se destaca a continuación:

/ \*\*

\* Implementar hook\_mail (). \* /

función user\_warn\_mail ($ key, & $ mensaje, $ params) {switch ($ key) {

caso "advertir":

$ Cuenta = $ params ['cuenta'];

$ Subject = variable\_get ('user\_warn\_e Email\_ tema', 'Advertencia Administrativa');

**$ Body = variable\_get ('user\_warn\_e-mail\_text', USER\_WARN\_MAIL\_TEXT);**

if (variable\_get ('user\_warn\_bcc', false)) {$ ADMIN\_MAIL = variable\_get ('site\_mail', NULL); $ Mensaje ['headers'] ['CCO'] = $ ADMIN\_MAIL;

}

$ Mensaje ['a'] = $ mail account->; $ Mensaje ['subject'] = $ asunto; $ Mensaje ['cuerpo'] [] = token\_replace ($ cuerpo,

**array (=> $ cuenta 'user'));**

romper;

}

}

Como puede ver, ahora estamos estableciendo el cuerpo del correo electrónico con el valor de retorno de token\_replace (). Esta función es muy simple, sólo toma dos argumentos:

* El texto con fichas en su lugar.
* Un conjunto de objetos con clave que se utilizarán en el proceso de sustitución de tokens. En este caso, el objeto de usuario para que el destinatario de este e-mail a medida que pasaba en el $ params argumento de la drupal\_mail (). Si usted necesita otros repuestos (como para un nodo) deberá añadir objetos adicionales en el array.

**[148 ]**



*Capítulo 5*



Eso es todo! El texto regresó de token\_replace () será ahora algo parecido a esto:

Hola eshqi,

Se nos ha notificado que ha publicado los comentarios en el sitio más fresco en el mundo que se encuentran en violación de nuestros términos de servicio. Si este comportamiento continúa se suspenderá su cuenta.

Atentamente,

El sitio más fresco en el mundo

Este e-mail es mucho mejor y personalizada tanto para el remitente y el destinatario.

**Resumen**

En realidad, el usuario Warn módulo es, probablemente, de utilidad limitada, pero sí ayuda a introducir muchos de los conceptos básicos que los desarrolladores de Drupal utilizarán sobre una base del día a día. Ahora son capaces de crear páginas con una URL específica utilizandohook\_menu (), E implementar formularios en las páginas utilizando el API de formularios. Los valores presentados de esta forma se pueden guardar con funciones como system\_settings\_form (),confirm\_form (), O en su propio manejador de envío personalizada. También puede enviar los resultados de un envío de formulario como un correo electrónico personalizado mediante tokens dinámicos para reemplazo de texto.

En el capítulo 7, crear nuevos campos, vamos a empezar a examinar la nueva API de Campo de Drupal 7, la implementación núcleo de lo que era antes el módulo CCK.

**[149 ]**



Trabajar con contenido

Drupal 7 introduce cambios importantes en la forma en que trata el contenido de Drupal. En versiones anteriores, casi todo el contenido fue considerado un "nodo". Al hacer que el contenido de un objeto de serie con un API común, cualquier módulo podría agregar datos y manipular ese objeto para crear modelos de datos complejos y flujos de trabajo.

Eso trabajado muy bien, con la excepción de que Drupal tenía varios otros tipos de objetos, como usuarios o comentarios, que no eran realmente "contenido" de por sí, pero aún podrían haberse beneficiado de la misma rica API. Para Drupal 7, por lo tanto, la mayoría de los tipos de objetos separados se fusionaron en una sola super-sistema conocido como "entidades". Nodos, los usuarios, comentarios, y varios otros tipos de objetos de datos son ahora los casos particulares de la entidad de un objeto de datos genéricos. Esto permite que todos los tipos de datos que tengan el mismo, o al menos muy similar, principios activos y de flujo de trabajo, evitar código duplicado y reduciendo el número de piezas móviles desarrolladores necesitan para seguir la pista. Lo más importante es que nos permite adjuntar Fields, piezas estructuradas discretos de información, a cualquier tipo de entidad, y no sólo a los nodos.

En este capítulo, veremos cómo definir nuevos tipos de entidad. Hay un montón de piezas móviles, y mientras el sistema entidad automatiza gran parte del proceso que para nosotros no automatiza todo. En el camino vamos a tocar en varias nuevas piezas de Drupal y reiteramos lo que hemos cubierto en los capítulos anteriores sobre las devoluciones de llamada de página y manejo de formularios.

**¿Por qué crear sus propias entidades**

Por lo general, no es necesario crear un nuevo tipo de entidad. Los nodos son todavía extremadamente flexible, y más a menudo que no puede manejar lo que sea de casos de uso que necesitan. Sin embargo, hay casos en los que es necesaria la creación de entidades separadas en lugar de tipos de nodos separados, como por ejemplo:

* Es posible que tengamos entidades que tienen totalmente diferente manejo permiso o flujo de trabajo de los nodos, como los productos en un sistema de comercio electrónico.

*Trabajar con contenido*



* Podemos estar accediendo a las entidades que no se almacenan en bases de datos locales de Drupal, como un almacén de datos heredados.
* Es posible que tengamos que tener variantes internas, como los tipos de nodos, pero los nodos no apoyar "tipos de tipo sub".

Para simplificar, no vamos a hacer nada demasiado exótico por ahora. En su lugar, vamos a ver una relativamente simple de casos de uso y el espejo nodo manejo bastante de cerca.

**El objetivo**

Para nuestro ejemplo, vamos a crear una nueva entidad llamada "obra de arte". Esta entidad representará una obra de arte en manos de un museo y gestionado a través de Drupal. Al igual que los nodos, obras de arte tendrán subtipos como "pintura" y "escultura". Estaremos dispuestos a permitir a los usuarios crear, editar y eliminar obras de arte, así como configurar lo que se dispone de campos en cada tipo de obra.

En la práctica la mayoría de los museos reales habrían su colección almacenada en un sistema de gestión de la colección dedicada y que tendrían que acaba de proporcionar una envoltura que lee datos de una manera Drupal ambiente. Para nuestros propósitos, sin embargo vamos a suponer un pequeño museo que quiere usar Drupal como un sistema de gestión de cobro simple, que implica la plena crear, leer, actualizar y eliminar capacidades.

**Bundles**

En versiones anteriores de Drupal nodos sólo tenían la posibilidad de tener subtipos. En Drupal 7, todas las entidades que tienen la capacidad de apoyar a los sub-tipos. En la jerga de Drupal, estos subtipos se llaman "paquetes".



La**liar** es un subtipo de una entidad que se puede configurar por separado.

Los tipos de nodo son un ejemplo de un paquete. No todos los tipos de entidades tienen bultos. Los usuarios, por ejemplo, no tienen haces separados.



Por ahora, vamos a duro-código de dos haces, la pintura y la escultura. En un caso de uso real, estaríamos propensos a incluir también un sistema de administración para crear y gestionar paquetes.

**La API de esquema**

Vamos a necesitar un lugar para almacenar los datos de obras de arte, así que tenemos que crear algunas nuevas tablas de la base de datos. En lugar de crear directamente, sin embargo, vamos a dejar que Drupal hacer eso por nosotros mediante una parte de la capa de base de datos llamada a la API de esquema.

**[152 ]**



*Capítulo 6*



La API de esquema permite la definición y manipulación de las tablas en la base de datos de Drupal SQL-base de datos agnóstico.

En primer lugar, vamos a crear un nuevo módulo llamado "obra de arte". Comience con la artwork.info y artwork.module archivos, como hemos visto en capítulos anteriores. Sin embargo, nosotros también agregar otro archivo, artwork.install. Este archivo contiene hooks que Drupal solamente siempre utiliza cuando se instala el módulo, se quita o se actualiza de modo que sólo se carga en esos momentos, el ahorro considerable de código en la mayoría de las cargas de la página.

El hook más importante en el artwork.install archivo es hook\_schema (), Que define las tablas de base de este módulo proporciona. Vamos a empezar con la definición de la tabla siguiente, en estrecha colaboración basada en la tabla de nodos:

artwork\_schema function () {$ schema ['obra de arte'] = array (

'Description' => 'La tabla base para las ilustraciones.' => Array 'campos' (

'Ayuda' => array (

'Description' => 'El identificador principal de una obra.', 'Type' => 'serial',

'Sin signo' => TRUE, 'not null' => TRUE,

),

'Vid' => array ('description' =>

"El identificador de versión actual {artwork\_revision}. Vid. ',' Tipo '=>' int ',

'Sin signo' => TRUE, 'not null' => TRUE, 'default' => 0,

),

'Type' => array (

'Description' => 'El artwork\_type {} de esta obra de arte.', 'Type' => 'varchar',

'Length' => 32, 'no nulo' => TRUE, 'default' =>'',

),

'Title' => array (

'Description' => 'El título de esta obra de arte.', 'Type' => 'varchar',

'Length' => 255, "no nulo '=> TRUE,

**[153 ]**



*Trabajar con contenido*



'Default' =>'',

),

'Creado' => array ('description' =>

'La marca de tiempo Unix cuando se creó la obra.', 'Tipo' => 'int',

'No nulo' => TRUE, 'default' => 0,

),

'Cambiado' => array ('description' =>

'La marca de tiempo Unix, cuando la obra se salvó recientemente.', 'Tipo' => 'int',

'No nulo' => TRUE, 'default' => 0,

),

),

'claves únicas' => array (

|  |  |  |
| --- | --- | --- |
| 'Aid\_vid' => | | array ('ayuda', 'vid'), |
| "Ayuda" | => | array ("ayuda") |
| ), |  |  |

'Clave principal' => array ("ayuda"),

);

return $ esquema;

}

Eso se parece a un montón de código, pero no deja de ser otra gran variedad de Drupal. Las claves de la $ Schema array son los nombres de las tablas que se creen. Cada tabla se define entonces como otra matriz anidada que define los campos, índices y otros datos sobre la mesa.

La mayoría son fáciles de entender.



Ver http://drupal.org/node/146843 para más información sobre la API de esquema.

Tenga en cuenta que estamos usando un campo entero llamado ayuda para nuestra clave principal. También guardamos el paquete que pertenece a una obra de arte en una columna llamada tipo (Al igual que los nodos), y tenemos un campo "identificador de versión" llamado vid. Todas las entidades pueden apoyar el control de versiones de un modo similar a los nodos, así que vamos a construir que desde el primer momento.

Para almacenar las revisiones viejas, necesitaremos otra mesa también. Llamaremos a esa mesa artwork\_revision:

$ Schema ['artwork\_revision'] = array ('description' =>

'Almacena información acerca de cada versión guardada de una obra de arte {}.',

**[154 ]**



*Capítulo 6*



'Campos' => array ('ayuda' => array (

'Description' => 'La obra de arte {} Esta versión pertenece.', 'Type' => 'int',

'Sin signo' => TRUE, 'not null' => TRUE, 'default' => 0,

),

'Vid' => array (

'Description' => 'El identificador primario para esta versión.', 'Type' => 'serial',

'Sin signo' => TRUE, 'not null' => TRUE,

),

'Title' => array (

'Description' => 'El título de esta versión.', 'Type' => 'varchar',

'Length' => 255 'no nulo' => TRUE, 'default' =>'',,

),

'Creado' => array ('description' =>

'La marca de tiempo Unix cuando se creó la obra.', 'Tipo' => 'int',

'No nulo' => TRUE, 'default' => 0,

),

),

'índices' => array ('ayuda' => array ("ayuda"),

),

'Clave principal' => array ('vid'), 'las claves externas' => array (

'Obra de arte' => array ('table' => 'obra de arte', 'columnas' => array (

'Ayuda' => 'ayuda',

),

),

),

);

**[155 ]**



*Trabajar con contenido*



Nota aquí, así que estamos declarando explícitamente la ayuda campo de la tabla de revisión para ser una clave externa a la ayuda campo de la tabla obras de arte. Aunque Drupal no aprovecha misma información clave externa, otros módulos pueden hacerlo. Por convención, las tablas de Drupal deben ser sustantivos en singular.

Con estas dos tablas definidas enartwork\_schema (), Drupal creará automáticamente las tablas correspondientes para nosotros en la base de datos cuando el módulo se activa primero. Si nuestro módulo se desinstala por completo, sino que también se hará cargo de la eliminación de ellos para nosotros.

**La declaración de nuestra entidad**

Hay dos partes a contar Drupal acerca de nuestra nueva entidad. La primera es otro hook definición llamada hook\_entity\_info (). Este hook dice Drupal sobre la entidad o entidades que estamos ofreciendo, y también proporciona el sistema de interfaz de usuario de campo con la información necesaria que nos permita atribuimos campos para entidades-más sobre esto más adelante. La segunda parte es una "clase controlador", que es una clase PHP que será responsable de la carga y, en nuestro caso, crear, guardar y eliminar nuestras obras de arte.

Drupal incluye una clase controlador llamadaDrupalDefaultEntityController que maneja el caso más común, que vamos a estar emulando. Es extremadamente básico, sin embargo, y sólo se ocupa de la carga de objetos. Afortunadamente, es muy fácil de subclase el controlador por defecto y añadir nuestra propia funcionalidad de modo que es precisamente lo que vamos a hacer.



Un controlador es un objeto cargador para una entidad. Todos los tipos de entidad debe tener un controlador, pero muchos pueden utilizar el valor predeterminado. Diferentes controladores pueden requerir llaves adicionales en una definición de la entidad.



**La declaración de la entidad**

En primer lugar vamos a decirle a Drupal acerca de nuestro tipo de entidad utilizando hook\_entity\_info ():

/ \*\*

\* Implementa hook\_entity\_info (). \* /

artwork\_entity\_info function () {$ return ['obra de arte'] = array ('label' => t ('Obra de Arte'),

'Class controller' => 'ArtworkController', 'tabla base' => 'obra de arte',

'Tabla de revisiones' => 'artwork\_revision', 'callback uri' => 'artwork\_uri',

**[156 ]**



*Capítulo 6*



'Fieldable' => TRUE, 'claves de entidad' => array (

'Id' => 'ayuda', 'revisión' => 'vid', 'haz' => 'Tipo', 'label' => 'Título',),

'claves de paquetes' => array (=> 'Tipo' 'paquete',

),

'Cache estática' => TRUE, "paquetes" => array (), "los modos de vista '=> array (

=> Array 'full' (

'Label' => t ('contenido completo'), 'configuración personalizada' => FALSE,

),

'Teaser' => array ('label' => t ('Teaser'),

'configuración personalizada' => FALSE,

),

),

);

foreach (artwork\_types () como $ type => $ info) {$ retorno ['obras de arte'] ['bundles'] [$ Tipo] = array (

'Label' => $ info-> nombre, 'admin' => array (

'Path' => 'admin / estructura / obras de arte / manage /% artwork\_type', 'camino real' => 'admin / estructura / obras de arte / gestionar /'.

str\_replace ('\_', '-', $ type), el argumento paquete '=> 4,

los argumentos de acceso '=> array (' administrar obras de arte '),

),

);

}

return $ de retorno;

}

**[157 ]**



*Trabajar con contenido*



Una vez más, nuestro principal medio de comunicación con Drupal es a través de grandes conjuntos estructurados que definen toda la información que necesitamos. En este caso, nuestra $ Return matriz tiene una sola entrada, obras de arte. La cadena obras de arte, Como la clave de nivel superior, servirá como el "nombre de equipo" de esta entidad, que es la forma en que se hace referencia en el código. Laetiqueta clave especifica en qué nombre se deben mostrar al usuario. La tabla base, revisión entradas claves de mesa y objetos indican al sistema entidad acerca de cómo nuestras obras de arte va a ser almacenado, y son utilizados por el controlador por defecto:

* La mesa principal, donde se guardan obras de arte se llama obra de arte, que tiene un campo de clave principal (el campo "id") de ayuda.
* Las revisiones se almacenan en una tabla llamada artwork\_revision, y de la revisión de la identificación única se llamavid.
* Ya que estamos soportando múltiples paquetes, también indican al sistema qué campo se indicará a qué paquete pertenece una obra determinada. En este caso, se utiliza el tipo campo.
* El nombre legible por humanos de una obra determinada se almacena en eltítulo campo.

La modos de vista clave define las diferentes formularios en que nuestra entidad se puede ver. En este caso estamos definiendo una versión "completa" y una versión de "hook", al igual que los nodos utilizan, pero se podría definir lo que sea vista modos que queríamos. Otros módulos son libres para inyectar modos de visualización adicionales a través hook\_entity\_info\_alter () así. Como siempre, la clave de la matriz de modos de visualización es el nombre de la máquina del modo de visualización y la etiqueta propiedad es el nombre de persona amigable. La ajustes personalizados bandera indica si el

El campo de IU debería permitir formateadores de campo para configurar por separado para que el modo de vista por defecto. Se cambia fácilmente a través de la interfaz de usuario.

También definimos un uri\_callback función, a saber, artwork\_uri (). Eso nos permite abstraer la definición de la ruta dentro de Drupal donde se accederá a esta obra de arte. En lugar de codificar una ruta de acceso, como ilustraciones / $ ayudas, Llamamos a una devolución de llamada

funcionar para generar por nosotros. Eso es lo más importante cuando se enumeran las entidades de diferentes tipos, como podemos simplemente llamar a una sola función,entity\_uri ($ tipo, $ entidad), Y volver la información correcta para pasar a la url () o L () funciones. Nuestra sencilla de devolución de llamada se ve así:

función artwork\_uri ($ obra) {return array (

'Path' => 'artwork /'. $ Obra de arte-> auxilios,

);

}

**[158 ]**



*Capítulo 6*



El valor de retorno de la devolución de llamada es una matriz con dos llaves: camino, Que es el camino de Drupal donde la entidad vive, y Opciones, Que define otros parámetros para la url () y L () funciones para las cosas tales como las páginas anclas o OBTENER valores de consulta. Es seguro omitir la Opciones clave si no es necesario. Aunque nuestra aplicación es trivial, implementaciones alternativas podrían, por ejemplo, poner a todas las entidades de un determinado tipo en una sola página y tener un ancla para cada uno.



Al crear un vínculo con una entidad, siempre use entity\_uri (tipo $, $ Entidad) para generar los parámetros para pasar a la ya sea url () o L () funciones.



Tenga en cuenta que estas claves de matriz asumen que estamos usando el controlador por defecto para nuestra entidad. Un "controlador" es un objeto que se encarga de la carga del objeto de entidad para nosotros. El controlador se define como una clase PHP, y se puede escribir para cargar nuestra entidad desde cualquier lugar, no sólo la base de datos local. En nuestro caso, estamos definiendo un controlador personalizado

llamado ArtworkController que se extenderá desde DrupalDefaultEntityController, Por lo que utiliza las mismas claves. DrupalDefaultEntityController es un controlador genérico para las entidades que se almacenan en la base de datos local y se comportan, más o menos, como nodos. Si estábamos sacando datos de un sistema totalmente diferente tendríamos implementar nuestro propio controlador de cero que implementa la DrupalEntityControllerInterface interfaz, y entonces podríamos necesitar diferentes claves definidas en el hook entidad.

Otros dos claves importantes son las banderas cacheables y fieldable:

* cache estática indica que el controlador debe guardar una copia de una entidad en memoria después de que ha sido solicitada de manera que si tratamos de cargar una segunda vez en la misma solicitud de página sólo podemos usar esa copia en caché.
* fieldable indica a la API de campo que podemos adjuntar campos de esta entidad, de la misma manera como nodos. Eso es muy importante, ya que es una de las principales razones para definir un nuevo tipo de entidad en el primer lugar.

La segunda parte del hook es un poco más complicado. Es sobre todo allí para apoyar a la API de Campo, que se tratarán en el próximo capítulo. Ya que tenemos varios paquetes, tenemos que decirle a la API de campo lo haces de lo que tenemos y lo que los caminos para poner las interfaces adicionales de gestión de campo para nuestra entidad. Para ello, se define, para cada uno de nuestros paquetes, una etiqueta que se muestra al usuario y la información del menú que necesitará la API de campos para agregar a sí mismo en el menú. Las llaves aquí, bajoadministración, Son bastante explica por sí mismo. camino define la ruta que se debe utilizar en hook\_menu () para las páginas de la interfaz de usuario de campo, mientras camino real es la ruta exacta que se debe utilizar al generar enlaces dentro de la interfaz de administración.

**[159 ]**



*Trabajar con contenido*



Debido a que nuestra ruta contiene un marcador de posición de menú, también tenemos que especificar qué índice se encuentra en el argumento paquete. Recuerde que los argumentos del menú son 0-basan, por lo que el índice

4 es la quinta parte de la ruta, que en este caso es % Artwork\_type. También podemos controlar los permisos de un usuario necesita para acceder a las páginas de configuración de campo de esta entidad con elacceso de devolución de llamada y argumentos de acceso llaves, que trabajan de la misma manera como en un elemento de menú normales. Si no se especifica una devolución de llamada de acceso, entonces el user\_access devolución de llamada, que comprueba contra los permisos de usuario, es el valor predeterminado.

Por supuesto, ya hemos definido un nuevo marcador de posición del menú que necesitamos una devolución de llamada para ello.

También existe la artwork\_types () función, que aún no existe. Vamos a crear aquellos momentos. Son realmente muy simple, pero son una parte estándar de cualquier entidad.

artwork\_types función () {

$ tipo = & drupal\_static (\_\_FUNCTION\_\_);

if (empty ($ tipo)) {$ tipos ['pintura'] = (objeto) array (

'Type' => 'pintura', 'name' => t ('Pintura'),

'Description' => t ('Un cuadro hecho con pintura.')

);

$ tipos ['escultura'] = (objeto) array ('type' => 'escultura',

'Name' => t ('Escultura'),

'Description' => t ('Una talla hecha de piedra o de madera.')

);

}

return $ tipos;

}

función artwork\_type\_load ($ tipo) {

$ tipo = artwork\_types ();

$ Type = str\_replace ("-", "\_", $ type);

volver isset ($ tipos [$ tipo])? $ tipos [$ Tipo]: FALSO;

}

La artwork\_types () función devuelve una lista de objetos de tipo de obras de arte. Cada objeto de tipo de obra de arte no es más que una stdClass Objeto PHP que contiene toda la información pertinente que hay sobre cada paquete. Hay dos atributos importantes

al objeto de tipo obra de arte:

* Debe ser un objeto
* Tiene que haber una propiedad de ese objeto que coincide con el claves de paquetes definición hook\_entity\_info ()

**[160 ]**



*Capítulo 6*



En nuestro caso, hemos definido el paquete de utilizar el tipo propiedad para nuestros tipos de obras de arte tienen una propiedad llamada tipo que contiene el nombre de la máquina del haz. Una propiedad llamada nombre para el nombre de persona amistosa del paquete es una convención estándar, pero no es estrictamente necesario, ya que es un ser humano-legible descripción propiedad.

La artwork\_type\_load () función es necesario que el marcador de posición de menú para trabajar, pero también es una función de confort muy bueno tener disponible también. En general es una buena práctica para proporcionar, APIs flexibles limpias para cualquier sistema que desarrollamos, aunque no esperamos que para usarlos. Lo más probable es que vamos bien encontramos un uso para ellos fuera de plazo o alguien más lo hará pensar en un uso que no esperábamos.

Tenga en cuenta que estamos sustituyendo los guiones en un nombre de tipo de subrayado. Eso es porque, por convención, todas las rutas de Drupal utilizan guiones en lugar de guiones, pero nombres de lotes necesitan usar guiones bajos, no guiones. Cuando usamos un nombre de tipo en un URL siempre vamos a utilizar un guión, y lo que aquí primero doblar el tablero de nuevo a un guión para asegurarse de que nos encontramos con el tipo de obra correcta.

Hay otro detalle importante aquí, y que es la drupal\_static () función. Esta función actúa como un colector central para las variables de PHP estático, es decir, aquellos que no son técnicamente global, pero si persisten entre llamadas a una función. Se utilizan muy comúnmente como una caché de peso ligero para evitar la re-elaboración o re-cargar los mismos datos dentro de la misma solicitud de página, pero que en algunos casos pueden dar lugar a efectos secundarios extraños cuando los datos que se almacenan en caché los cambios de mitad de solicitud, como cuando se escribe

pruebas unitarias.

La drupal\_static () función actúa como un colector central para este tipo de variables estáticas. Al poner todas estas variables estáticas en un lugar y dar a la variable un nombre que coincide con nuestra función (que es lo que el \_\_FUNCTION\_\_ Medio de constantes PHP), permitimos que los sistemas que necesitan para restablecer la fuerza cachés estáticas sin tener una separada $ Reinicio parámetro para cada parte del sistema.



Utilice el drupal\_static () función para almacenar en caché barato estructuras de datos para una solicitud de página. ¿Todavía no los datos de caché que es demasiado grande y demasiado barato para regenerar, sin embargo, ya que utiliza la memoria.



**El controlador entidad**

En la información hook entidad declaramos que íbamos a usar nuestra propia clase controlador. Eso significa que tenemos que proporcionar una. Sin embargo, una clase de controlador no siempre puede ser pequeña, y si es sólo rara vez se utiliza no queremos analizar ese código en todas las páginas. Afortunadamente PHP proporciona una manera de cargar las definiciones de clase en la demanda, y

Drupal hace que sea muy fácil para exponer las clases de cargador automático de PHP.

**[161 ]**



*Trabajar con contenido*



Como regla general, las grandes clases, rara vez se utilizan deben ser colocados en un archivo separado, mientras que las clases más pequeñas o muy de uso frecuente se deben dejar en el archivo de módulo. Para evitar la sobrecarga de encontrar la clase cuando sea necesario. Además, las clases que se utilizan normalmente en conjunto pueden ser colocados en un solo archivo de manera que tendremos que cargar un solo archivo.



Vamos a crear un nuevo archivo en nuestro módulo llamado artwork.controller.inc. A continuación, añadir que

presentar a la archivos [] matriz en la artwork.info presentar. En artwork.controller.inc, Vamos a empezar con sólo el siguiente código:

clase ArtworkController extiende DrupalDefaultEntityController {

}

Ahora, cuando nuestro módulo está activado, Drupal escanea todos los archivos en el archivos [] matriz en artwork.info, Encontrar el ArtworkController clase, y la memoria caché de su ubicación. Más tarde, cuando algún código intenta crear una nueva instancia de ArtworkController será flojo a cargar el artwork.controller.inc de archivos, logrando la clase disponible para ser utilizado.

Naturalmente, la ArtworkController clase tiene que hacer realmente algo. Vamos a añadir métodos adicionales para que a medida que avanzamos. Ya existe un load () método, heredado de

DrupalDefaultEntityController, Así como varios otros.

La mayoría del código de Drupal prefiere trabajar procesalmente, sin embargo, incluso si el motor bajo el capó es orientado a objetos. Por lo tanto, al igual que el módulo de nodo vamos a ofrecer un conjunto de funciones de la API de utilidad para nosotros y otros desarrolladores de módulos para su uso.

función artwork\_load ($ aid = NULL, $ vid = NULL, $ restablecer = false) {$ ayudas = (isset ($ auxilios) array ($ auxilios): array ());

$ condiciones = (? isset ($ vid) array ('vid' => $ vid): array ()); $ Obra = artwork\_load\_multiple (ayudas $, $ condiciones, $ restablecimiento); return $ obra? reset ($ obra): FALSO;

}

función artwork\_load\_multiple (ayudas $ = array (), $ condiciones = array (),

$ Reset = false) {

volver entity\_load ('obra de arte', $ ayudas, $ condiciones, $ reset);

}

Tenga en cuenta que en realidad estamos pasando toda la lógica de carga de nuevo a la entity\_load () función, que a su vez va a crear una nueva instancia de la ArtworkController según sea necesario y llamar a la load () método en él. La load () método asume que todas las operaciones son multi-carga. De hecho, el sistema entidad asume las operaciones de múltiples objetos siempre que sea posible. Eso hace que la carga de varios objetos a la vez mucho más barato como podemos cargar todos a la vez, durante la carga de un único objeto es la misma operación exacta. "Uno" es un caso especial de "muchos". Ahora podemos cargar uno o una docena de obras de los objetos con la misma facilidad.

**[162 ]**



*Capítulo 6*



La mayoría de las operaciones de la entidad en Drupal asumen varios objetos.

Ahora que hemos definido nuestra entidad y podemos cargarlo, ¿qué podemos hacer con él? En estos momentos no es mucho, ya que no tenemos manera de crear o editar entidades. Todavía tenemos que definir el resto del ciclo de vida de entidad y exponer una interfaz de usuario para ello.

**Dirección de la Entidad**

En la práctica, los diferentes tipos de entidades frecuentemente necesitarán un flujo de trabajo distinto para crear, editar y eliminar entidades. Algunos incluso pueden no ser editable a través de la interfaz de usuario. Para simplificar las cosas, por ahora, sin embargo, que me quedo con el flujo de trabajo de nodo familiar y replicamos que, más o menos.

Debido a que hay varias partes, vamos a hacer frente a uno a la vez:

1. Crear un flujo de trabajo administrativo para la gestión de los tipos de obras de arte.
2. Crear páginas para agregar y editar obras de arte.
3. Crear una página para ver las obras de arte.
4. Crear un formulario para la supresión de obras de arte.

**Administración de tipos de obras de arte**

Vamos a empezar con un talón de un administración sección de la gestión de los tipos de obras de arte. Aunque no vamos a permitir que los usuarios creen nuevos tipos de obras de arte que todavía tiene que tener por lo menos una sección administrativa muy mínimo en el que el sistema de campo puede colgar su interfaz. Vamos a empezar con una maniobra sencilla artwork\_menu () hook:

función artwork\_menu () {$ items ['admin / estructura / obras de arte'] = array (

'Title' => 'Administrar obras de arte', 'description' => 'Administrar obras de arte.',

La página de callback '=>' artwork\_overview\_types ',' argumentos de acceso '=> array (' administrar obras de arte '),

);

$ items ['admin / estructura / obras de arte / manage /% artwork\_type'] = array ('title' => 'Ver tipo obra de arte',

'Title callback' => 'artwork\_type\_page\_title', 'argumentos title' => array (4),

La página de callback '=>' artwork\_information ',' Página de argumentos '=> array (4),

**[163 ]**



*Trabajar con contenido*



los argumentos de acceso '=> array (' administrar obras de arte '),

); $ items ['admin / estructura / obras de arte / manage /% artwork\_type / ver'] =

array (

'Title' => 'Ver',

'Type' => MENU\_DEFAULT\_LOCAL\_TASK,

);

return $ artículos;

}

Una vez más, mientras que las grandes matrices de Drupal parecen una gran cantidad de código que es todo bastante sencillo. El primer punto establece nuestra página de aterrizaje tipo obra de arte en admin / estructura / Obras (Que es paralelo deliberadamente el admin / estructura / contenido ruta de acceso para la gestión de los nodos). La segunda establece una devolución de llamada para la visualización y edición de cada tipo de obra (bundle), aunque no vamos a hacer nada con ella, excepto mostrar un poco de información estática. El tercer elemento de menú crea una pestaña del menú (o en Drupal jerga, "tarea local") llamaVista, Que es el valor predeterminado y se de paso a través de la página original de tipo obra de arte de administrar.

La definición pestaña es importante. Recuerde que en la última sección que nuestro artwork\_entity\_info () función dijo que el sistema de interfaz de usuario de campo que debería sumar su páginas administrativas propias de admin / estructura / obras de arte / manage /% artwork\_type. Cuando el módulo de interfaz de usuario El campo está activado, por lo tanto, que se sumará páginas adicionales

pestañas al lado de nuestra pestaña Vista. Si nosotros no declaramos nuestra página como una pestaña, que se perderá cuando alguien vea las páginas de la UI de campo.

Hay algunas nuevas funciones que necesitamos aquí para apoyar a estas páginas, todas ellas muy corto.

artwork\_overview\_types función () {

foreach (artwork\_types () como $ type => $ info) {$ type\_url\_str = str\_replace ('\_', '-', $ tipo);

$ Label = t ('View @ type', array ('type @' => $ info-> nombre)); $ items [] = l ($ etiqueta, 'admin / estructura / obras de arte / manage /'.

$ Type\_url\_str);

}

devolver el tema ('item\_list', array ('items' => $ items));

}

**[164 ]**



*Capítulo 6*



Esta devolución de llamada, simplemente nos da una lista no ordenada de todos los tipos de obras de arte que tenemos disponibles. Cada uno se presenta como un enlace a la página de gestión de ese tipo. Tenga en cuenta que nosotros transformamos este tipo de obras de arte para usar guiones en lugar de guiones en el camino. Es un estándar de Drupal que vamos a utilizar también.

función artwork\_type\_page\_title ($ tipo) {

volver t ('Administrar tipo @', array ('type @' => $ tipo-> nombre));

}

Queremos que el título de cada página sea una cadena basada en el tipo que se pasa, así que tenemos que especificar una devolución de llamada título. La devolución de llamada título no podía ser más simple:

artwork\_information función ($ artwork\_type) {

return $ nombre artwork\_type->. ':'. $ Artwork\_type-> descripción;

}

La página de resumen para cada tipo de obra de arte es trivial, así, al menos en nuestro caso.

Un tipo de entidad más compleja tendría varias opciones de configuración aquí en una forma, tanto la forma en que los nodos hacen. En nuestro caso, sin embargo, nos limitaremos a mostrar el título y la descripción del tipo de obra de arte.

Lo creas o no, eso es todo. Porque no estamos permitiendo que cualquier otra configuración de usuario de los tipos de obras de arte a nosotros mismos, la creación de ese esqueleto mínimo es todo lo que necesitamos hacer. Sin embargo, si el módulo de interfaz de usuario El campo está habilitado debemos vea Administrar campos y administrar las pestañas de visualización durante la visualización de cada uno de esos tipos de obras de arte.

Ir a la página de la pintura, a continuación, seleccione Administrar campos. Añadir un nuevo campo de texto de una línea llamada Artista, tal como lo haría para un nodo. Ahora ve a la escultura y agregue el campo Artista ahora existente para ese tipo obra de arte.

Esa es la mayor ventaja de hacer un nuevo objeto de datos de una entidad. Todo el poder de campos obtenemos gratuitamente de esa manera. Tome en cuenta que incluso los campos que normalmente se conectan a los tipos de nodos están disponibles para nosotros. Ahora podemos construir el modelo de datos de nuestras obras de arte sin tener que escribir ningún código adicional, el uso de todos los tipos de campo disponibles a nosotros, ya sea en el interior o en módulos de terceros.

**Adición de obras de arte**

Ahora que hemos nuestros tipos artwork conectado, tenemos que ser capaces de crearlos. Una vez más el modelado de la forma en que los nodos de trabajo vamos a añadir las siguientes nuevas entradas a nuestro hook de menú:

función artwork\_menu () {$ items ['artwork / agregar'] = array (

'Title' => 'Añadir nueva obra de arte',

La página de callback '=>' artwork\_add\_page ',

**[165 ]**



*Trabajar con contenido*



los argumentos de acceso '=> array (' crear obras de arte '),' peso '=> 1,

=> 'Gestión' 'nombre\_menú', 'file' => 'artwork.pages.inc',

);

foreach (artwork\_types () como $ tipo) {

$ Type\_url\_str = str\_replace ('\_', '-', $ type-> type); $ items ['artwork / add /'. $ Type\_url\_str] = array (

'Title' => $ name tipo->,

'Title callback' => 'check\_plain', 'Página callback' => 'artwork\_add', los argumentos de las páginas '=> array (2),

los argumentos de acceso '=> array (' crear obras de arte '),' description '=> $ Descripción tipo->,

);

}

/ / ...

return $ artículos;

}

La primera opción de menú crea una página de índice de tipos de obras de arte para que seamos capaces de crear tipos de obras de arte. A continuación, lazo sobre todos los tipos de obras de arte disponibles y crear un elemento de menú para cada uno, convirtiendo guiones en cada tipo de ilustraciones en guiones. El título de cada elemento del menú es el nombre del tipo de obra de arte, pero para estar seguros que lo lleva a través de la check\_plain () función. check\_plain () tendrá todo lo que se especifica en el título y escapar de todas y cada una de HTML en el mismo, por lo que es seguro de usar como el título de la página. Eso es siempre necesaria cuando se muestra una cadena generado por el usuario como un título de página para evitar tanto los riesgos de seguridad y las páginas potencialmente rotos.

El listado de la página es bastante simple, pero sí muestra una característica interesante del sistema de menús. El siguiente es un ejemplo:

función artwork\_add\_page () {$ item = menu\_get\_item ();

$ links = system\_admin\_menu\_block ($ item);

foreach ($ vínculos como $ enlace) {

$ items [] = l ($ enlace ['title'], $ enlace ['href'], $ item ['localized\_options'])

. ':'. filter\_xss\_admin ($ enlace ['description']);

}

devolver el tema ('item\_list', array ('items' => $ items));

}

**[166 ]**



*Capítulo 6*



Esta página simplemente generar una lista con viñetas de los elementos del menú que son elementos secundarios inmediatos de la página actual se está viendo. menu\_get\_item () devuelve el elemento de menú actual mientras system\_admin\_menu\_block (), A pesar de su nombre, es una función de propósito general útil para sacar a los niños de un elemento de menú arbitraria.

Hay otros dos puntos a tener en cuenta aquí:

* En primer lugar, el enlace lleva a la localized\_options clave de la opción de menú como su propia gama de opciones. De esta manera permitimos que el sistema de menú para manejar lo atribuye el enlace debe tener ningún tipo de interacción con el sistema de traducción, y así sucesivamente.
* En segundo lugar, nos encontramos la descripción a través de filter\_xss\_admin (). La función simplemente despojar a las etiquetas HTML que no son seguros. La filter\_xss () función nos permite definir una lista arbitraria de etiquetas permitidas, mientras que filter\_xss\_admin () es simplemente un contenedor que incluye un conjunto bastante permisiva de etiquetas permitidas.

**Adición de nuevas obras de arte**

Ahora tenemos que proporcionar una forma para agregar nuestras nuevas obras de arte. Hemos definido una devolución de llamada de menú denominado artwork\_add anterior, por lo que vamos a definir esa devolución de llamada.

función artwork\_add ($ tipo) {global $ user;

$ tipo = artwork\_types ();

$ Type = isset ($ tipo)? str\_replace ('-', '\_', $ type): NULL; if (empty ($ tipos [$ type])) {

volver MENU\_NOT\_FOUND;

}

$ Obra = entity\_get\_controller ('obra de arte') -> create ($ tipo);

drupal\_set\_title (t ('Create @ name', array ('@ name' => $ tipos [$ tipo] -> nombre)), PASS\_THROUGH);

volver drupal\_get\_form (tipo $ '\_artwork\_form', $ obra.);

}

Al igual que antes, limpiamos el nombre del tipo de obras de arte (paquete) y luego, si no existe tal tipo de obra volvemos una página de error 404 no encontrado. A continuación, creamos un nuevo objeto obra de arte vacío, establezca un título de la página y, a continuación, mostrar un formulario.

**[167 ]**



*Trabajar con contenido*



Hay un par de piezas en movimiento aquí. El primero es el método create del controlador. La entity\_get\_controller () función crea una nueva instancia de la clase del controlador para las entidades de las ilustraciones, que es el ArtworkController clase que definimos anteriormente. El método de crear no es parte de la interfaz de control normal, pero es realmente un buen complemento. Es muy sencillo:

public function create ($ type ='') {return (objeto) array (

'Ayuda' =>'', 'type' => $ tipo, 'title' =>'',

);

}

Es más fácil definir las grandes estructuras de datos en PHP como matrices, por lo que vamos a hacer eso y lo echó de nuevo a un objeto para devolverlo. Ahora podemos crear un nuevo objeto de obras de arte desde cualquier lugar.

Siguiente es el PASS\_THROUGH constante que estamos pasando como segundo parámetro a drupal\_set\_title (). Normalmente drupal\_set\_title se tira hacia fuera código HTML de la título para evitar problemas de seguridad. Pasando PASS\_THROUGH como el segundo parámetro indica a la función de permitir HTML en la cadena que estamos dando, porque ya hemos comprobado para asegurarse de que es seguro.



Utilice el PASS\_THROUGH marca con drupal\_set\_title () para permitir HTML en el título de la página. Recuerde que significa que es a nosotros a prueba de cordura y filtrar el título para evitar los ataques de cross-site-scripting.



Por último está el formulario. Estamos dando la forma de un nombre dinámico para que form\_alter implementaciones se pueden orientar de forma única un tipo de obras de arte específica. ¿Cómo entonces corresponderá al nombre del formulario con la función de constructor de la forma que lo define? Drupal ofrece otro hook opcional para nosotros llamada hook\_forms () que nos permite definir dinámicamente formularios y cómo debemos comportarnos.

artwork\_forms function () {$ formularios = array ();

if ($ tipo = artwork\_types ()) {

foreach (array\_keys ($ tipo) como $ tipo) {

$ formularios [escriba $. '\_artwork\_form'] ['Callback'] = 'artwork\_form';

}

}

return $ formularios;

}

**[168 ]**



*Capítulo 6*



Este hook define una nueva forma al vuelo para cada tipo obra de arte que tenemos, y declara que todos ellos deben utilizar la misma función de devolución de llamada, artwork\_form (). Ahora todo lo que necesitamos es la artwork\_form () función y todas las formularios de las ilustraciones tendrán el mismo procesamiento.

artwork\_form función ($ form, & $ form\_state, $ obra) {

* Establezca el id para identificar esto como una forma de obra de arte de edición. $ Form ['# id'] = "forma-obra de arte ';
* Guardar su trabajo para más tarde, en caso de que lo necesitemos.

$ Form ['# artwork'] = $ obra de arte; $ Form\_state ['obra de arte'] = $ obra de arte;

/ / campos comunes. No tenemos muchos. $ ['Title'] form = array (

'Type #' => 'campo de texto', '# title' => t ('Título'),

'# Default\_value' => $ obra de arte-> title, '# peso' => -5,

'# Required' => TRUE,

);

$ [«Revisión»] form = array (

'# Access' => user\_access ('administrar obras de arte'), '# type' => 'checkbox',

'# Title' => t ('Crear nueva revisión'), '# default\_value' => 0,

);

/ / Añadir los botones. $ Form ['botones'] = array (); $ Form ['botones'] ['# peso'] = 100; $ Form ['botones'] ['submit'] = array (

'Type #' => 'enviar', '# value' => t ('Save'), '# peso' => 5,

'# Submit' => array ('artwork\_form\_submit'),

);

if (empty ($ obra de arte-> ayuda)!) {$ form ['botones'] ['borrar'] = array (

'# Access' => user\_access ('delete obras de arte'), 'type #' => 'Enviar',

"Valor # '=> t (' Eliminar '),' # peso '=> 15,

**[169 ]**



*Trabajar con contenido*



'# Submit' => array ('artwork\_form\_delete\_submit'),

);

}

['# Validate'] [] = 'artwork\_form\_validate' $ form; field\_attach\_form ('obra de arte', $ obra de arte, $ form, $ form\_state);

devolver $ form;

}

Una vez más, un buen número de líneas, pero todos bastante simple. Primero establecimos una propiedad falsa en el formulario para que form\_alter implementaciones pueden detectar todas las formularios de las ilustraciones de cualquier tipo y pegar el objeto en sí mismo obra en la forma en caso de que la necesite más adelante en validar o sometemos rutinas. A continuación, añadir un formacampo para el título y una casilla de verificación para indicar si se debe guardar una nueva revisión o sobrescribir la revisión existente. A continuación, añadir no uno, sino dos botones, uno que siempre está disponible para la operación de almacenamiento y otra para borrar una obra si estamos editando una obra existente. Más sobre esto más adelante.

La última línea es muy importante. La field\_attach\_form () llamada es lo que pasa a nuestro formulario de apagado al sistema de campo a fin de que todos los campos que hemos añadido a este tipo de obras de arte se pueden agregar al formulario. Piense en ello como un campo específico form\_alter operación. Las propiedades del formulario que el sistema de campo añade son bastante complejos, pero no tiene que preocuparse de ellos. Sólo trabajan.

**Devolución de llamada de validación**

Similar a la forma, la devolución de llamada de validación pasa la mayor parte de su lógica fuera del sistema de campos.

artwork\_form\_validate función ($ form, & $ form\_state) {$ obra = $ form\_state ['obra de arte'];

Validación / / Campo.

field\_attach\_form\_validate ('obra de arte', $ obra de arte, $ form, $ form\_state);

}

Si tuviéramos ninguna validación de nuestra propia incluiríamos aquí, pero en este caso ningún tipo de validación que nos gustaría hacer (comprobación de campos obligatorios) de la API de forma ya lo hace por nosotros.

Todo lo que necesitamos hacer es decirle al sistema de campos para hacer su propia validación.

**[170 ]**



*Capítulo 6*



**Enviar callback**

La devolución de llamada de presentar es un poco más interesante, pero no por mucho:

artwork\_form\_submit función ($ form, & $ form\_state) {global $ user;

$ Obra = & $ form\_state ['obra de arte'];

/ / Establecer el UID de la obra si se está creando en este momento. if (empty ($ obra de arte-> uid)) {

$ Obra de arte-> uid = $ usuario-> UID;

}

$ Obra de arte-> title = $ form\_state ['valores'] ['title']; $ Obra de arte-> revision = $ form\_state ['valores'] ['revisión'];

/ / Notificar a los widgets de campo.

field\_attach\_submit ('obra de arte', $ obra de arte, $ form, $ form\_state);

* Guarde la ilustración. artwork\_save ($ obra de arte);
* Notificar al usuario.

drupal\_set\_message (t ('Obra guardado.'));

$ Form\_state ['redirect'] = 'artwork /'. $ Obra de arte-> ayuda;

}

Primero sacamos el objeto ilustraciones viejas de la $ Form\_state variable. Siguiente nos pusimos a su usuario, si se trata de una nueva obra de arte, y rellenarla con los pocos valores que tenemos de la forma. Ya hemos confirmado que esos valores son válidos en el hook de validación. field\_attach\_submit (), Al igual que sus funciones de hermanos, deja el sistema de Campo responder al hecho de que una forma se acaba de presentar y hacer lo que quiera

hacer. Entonces simplemente nos ahorramos la obra de arte.

Es importante señalar aquí que la devolución de llamada presentar está haciendo muy poco. Es sólo código de unión entre la forma y elartwork\_save () funcionar, en realidad. Eso es una buena cosa. Esto significa que las operaciones de obras de arte, tales como el ahorro, no dependen en el sistema de forma.



Nunca hacer el proceso grave en un formulario de envío de devolución de llamada. El trabajo real de ahorro de una entidad y sus datos debe suceder en guardar las rutinas de la entidad. La forma de presentar de devolución de llamada es sólo un código de adhesión.



**[171 ]**



*Trabajar con contenido*



**Ahorro de su obra**

¿Qué artwork\_save () parecer, sin embargo? Como artwork\_load () no deja de ser un contenedor delgado:

función artwork\_save ($ obra) {

volver entity\_get\_controller ('obra de arte') -> save ($ obra);

}

Una vez más, dejamos que la clase del controlador haga el trabajo duro. La save () método en sí es en realidad divide en dos partes, una para el ahorro normal y una para revisiones.

public function save ($ obra) {$ transaction = db\_transaction ();

try {

el usuario $ a nivel mundial;

* Determine si vamos a insertar una nueva obra de arte. $ Obra de arte-> is\_new = empty ($ obra de arte-> auxilios);
* Establezca los campos de marca de tiempo.

if (empty ($ obra de arte-> creado)) {$ obra de arte-> created = REQUEST\_TIME;

}

$ Obra de arte-> changed = REQUEST\_TIME;

$ Obra de arte-> revision\_timestamp = REQUEST\_TIME;

$ Update\_artwork = TRUE;

* Dale módulos la oportunidad de preparar los datos de campo para
* ahorro.

field\_attach\_presave ('obra de arte', $ obra de arte);

if (! $ obra de arte-> is\_new &&! empty ($ obra de arte-> revisión) && $ obra de arte-> vid) {

$-> Obra old\_vid = $ obra de arte-> vid; unset ($ obra de arte-> vid);

}

/ / Si se trata de una nueva obra de arte ...

if ($ obra de arte-> is\_new) {/ / Guardar la nueva obra de arte.

drupal\_write\_record ('obra de arte', $ obra de arte);

**[172 ]**



*Capítulo 6*



/ / Guardar la revisión inicial. $ This-> saveRevision ($ obra de arte, $ user-> uid);

$ Op = 'insert';

}

else {

/ / Guardar la obra actualizada. drupal\_write\_record ('obra de arte', $ obra de arte, «ayuda»);

if (empty ($ obra de arte-> revisión)!) {$ this-> saveRevision ($ obra de arte, $ user-> uid);

}

else {

$ This-> saveRevision ($ obra de arte, $ user-> uid TRUE); $ Update\_artwork = FALSE;

}

$ op = 'update';

}

* Si el ID de revisión es nuevo o actualizado, guardarlo en la obra de arte. if ($ update\_artwork) {

db\_update ('obra de arte') -> campos (array ('vid' => $ obra de arte-> vid)) -> de condición ("ayuda", $ obra-> ayuda) -> execute ();

}

* Ahorra campos.

Función $ = 'field\_attach\_'. $ Op; Función $ ('obra de arte', $ obra de arte);

module\_invoke\_all ('entity\_' $ op, $ obra, 'obra de arte'.);

* Propiedades internas claras. unset ($ obra de arte-> is\_new);
* Ignorar servidor esclavo temporal para dar tiempo a los salvados
* Para propagarse al esclavo.

db\_ignore\_slave ();

return $ obra;

}

**[173 ]**



*Trabajar con contenido*



catch (Exception $ e) {$ transaction-> rollback ();

watchdog\_exception ('obra de arte', $ e, NULL, WATCHDOG\_ERROR); return false;

}

}

La mayor parte de la rutina de ahorro se explica por sí, y mucha de ella es específica para las propiedades que hemos declarado en nuestra entidad. Algunas piezas tienen mayor discusión, sin embargo.

Estamos terminando todo el proceso en un bloque try-catch PHP. Eso es porque el proceso de guardar puede ser arbitrariamente complejo, que implica una serie de consultas, cualquiera de los cuales podría, potencialmente, romper. Por lo tanto, comenzamos una operación de base de datos con la db\_transaction ()función. Las transacciones permiten a varias consultas de base de datos, ya sea a todos tienen éxito o fracasan todos juntos. Las consultas que se ejecutan, desde cualquier lugar, entre el db\_transaction () llamar y cuando el $ Transaction la variable sale de ámbito al Al final de la función será parte de la transacción.

Si cualquier consulta, en cualquier lugar, fracasan, la base de datos generará un Excepción. (Hay otras razones un Excepción podría ser lanzado, también, pero esa es la más común.) En ese caso, rodamos toda la transacción con el rollback () método. Cualquier

consultas que ya habían corren llegar sin hacer, y todo el proceso de guardado de manera efectiva no sucede. Que nos impide terminar con una obra a medio-salvo, lo que requiere la edición manual de la base de datos para limpiar. También registramos la excepción utilizando una función de utilidad especial que decodifica la información útil de la excepción para nosotros.



Tenga en cuenta que las transacciones no están disponibles si el uso de tablas de bases de datos de MySQL MyISAM. En caso de que el código seguirá funcionando pero cualquier intento de deshacer una transacción será ignorado y la base de datos seguirá siendo dejado en un estado potencialmente inestable. Por esa razón, cuando se ejecuta en MySQL por defecto de Drupal para tablas InnoDB, que soportan transacciones.



Una vez que hemos terminado de guardar la obra de arte, también llamamos db\_ignore\_slave (). Si nos estamos quedando un sitio de alto tráfico, entonces podemos tener tanto una base de datos principal y una o varias bases de datos de esclavos configurados, algo Drupal soporta de forma nativa. Sin embargo, puede haber una latencia antes de nuestra obra de arte se propaga desde el servidor principal al servidor esclavo. Por lo tanto, decimos a Drupal que, sólo para el usuario actual, se debe omitir el servidor esclavo durante unos minutos para que el usuario que presentó la obra de arte lo verá de inmediato, incluso si hay un breve retraso para otros usuarios.

**[174 ]**



*Capítulo 6*



Configuraciones de bases de datos maestro / esclavo son un tema avanzado que no vamos a cubrir en detalle. Sin embargo, Drupal transparente cae de nuevo a la conducta de fuente única, si no se define un esclavo así que siempre es bueno para planificar el futuro para soportar una configuración maestro / esclavo.



Una vez más, hay varios puntos en los que nosotros llamamos el sistema de campos de dejar que haga su parte con nuestra entidad, que a su vez debe explicarse por sí mismo. También hacemos un llamamiento

module\_invoke\_all () con el fin de permitir que otros módulos para interactuar con cualquier entidad cuando se crean o actualizan.

**Manejo de las revisiones**

La única otra parte compleja del proceso consiste en la manipulación de revisión, que puede ser un poco difícil para envolver la cabeza alrededor. En cuanto a la saveRevision () método puede ayudar a:

función saveRevision ($ obra, $ uid $ update = false) {

* Aferrarse a creator\_uid original de la obra de arte, pero de intercambio
* en creator\_uid de la revisión de la escritura momentánea. $ Temp\_uid = $ obra-> UID;

$ Obra de arte-> uid = $ UID;

if ($ actualización) {

drupal\_write\_record ('artwork\_revision', $ obra, 'vid');

}

else {

drupal\_write\_record ('artwork\_revision', $ obra de arte);

}

/ / Reset creator\_uid de la orden con el valor original. $ Obra de arte-> uid = $ temp\_uid;

}

Después de que hemos salvado la obra de arte a la mesa de obra, también ahorramos a la mesa artwork\_revision. Si estamos creando una nueva revisión, vamos a guardar un nuevo registro a la tabla y drupal\_write\_record () se rellenará el vid propiedad para la nueva versión de identificación. Luego, de vuelta en save (), Que se actualice el registro de la tabla obras de arte para que apunte al nuevo registro de revisiones. Si nos limitamos a sobrescribió el viejo expediente de revisión luego de que paso no es necesario.

Eso es todo! Ahora podemos crear nuevos objetos de las ilustraciones directamente desde la interfaz de usuario con un sencillo formulario, incluyendo cualquier Campos hemos decidido unir nuestras obras de arte. No sólo eso, pero ya que nos ponemos la lógica de ahorro en el controlador en lugar de en el formulario de envío de devolución de llamada, podemos crear fácilmente nuevas obras de arte mediante programación; sólo tenemos que crear un nuevo objeto de obras de arte con las propiedades que queremos y llamadaartwork\_save ().

**[175 ]**



*Trabajar con contenido*



**Visualización de obras de arte**

La última línea de nuestro formulario de presentar controlador redirige al usuario a la ilustraciones / $ ayudas página, probablemente para ver la obra que acabamos de crear. Eso sería grandioso si hubiera una página allí para mostrar la obra de arte. Añadamos ahora.

En primer lugar tenemos un elemento de menú correspondiente:

función artwork\_menu () {$ items ['artwork /% artwork'] = array (

'Title callback' => 'artwork\_page\_title', 'argumentos title' => array (1),

La página de callback '=>' artwork\_page\_view ',' Página de argumentos '=> array (1),

los argumentos de acceso '=> array (' view 'obras de arte), "tipo" => MENU\_CALLBACK,

);

$ items ['artwork /% ilustraciones / view'] = array ('title' => 'Ver',

'Type' => MENU\_DEFAULT\_LOCAL\_TASK, 'peso' => -10,

); / / ...

}

Estamos planeando con anticipación para añadir más pestañas ("tareas locales") más adelante, pero definitivamente queremos la ficha Ver sea la predeterminada. El elemento de menú define un Título de devolución de llamada y Página de devolución de llamada, Ambos de los cuales son bastante sencillo.

función artwork\_page\_title ($ obra) {return $ obra-> title;

}

función artwork\_page\_view ($ obra, $ view\_mode = 'full') {/ / Eliminar el contenido previamente construido, si existe. $ Obra de arte-> content = array ();

if ($ view\_mode == 'Teaser') {$ obra de arte-> content ['title'] = array (

'# Markup' => filter\_xss ($ obra de arte-> title), '# peso' => -5,

);

}

/ / Construir contenidos campos. field\_attach\_prepare\_view ('obra de arte',

**[176 ]**



*Capítulo 6*



array ($ obra de arte-> aid => $ obra), $ view\_mode);

entity\_prepare\_view ('obra de arte', array ($ obra de arte-> aid => $ obra de arte)); $ Obra de arte-> content + = field\_attach\_view ('obra de arte', $ obra de arte,

$ View\_mode);

return $ obra-> content;

}

La Título de devolución de llamada no necesita ninguna explicación. La artwork\_page\_view () función de nuevo no hace mucho más allá de los datos de pastor fuera a la API de Campo. En particular, tenga en cuenta que lo hace tomar un $ View\_mode parámetro, que debe ser uno de los modos de vista que hemos definido de nuevo en el artwork\_entity\_info (). Incluso si no queremos aprovechar nosotros mismos el sistema de campo lo va a requerir.

Si estamos en el modo de "hook", queremos inyectar el título de la obra en el resultado visible. Si no es así, no tenemos datos para mostrar que no sean los campos adjuntos como el título ya se está mostrando como el título de la página. El código de campo es bastante repetitivo tres líneas de código que producen una matriz renderizable, que simplemente retornamos.

No ha sido tan difícil, ¿verdad?

**Edición de una obra de arte**

Ahora que tenemos las obras de arte que podemos ver, queremos ser capaces de volver y editarlos. Afortunadamente, la mayoría de los trabajos que ya se hicieron como parte del proceso de creación. De hecho, vamos a volver a utilizar la misma forma y el mismo save () método del controlador. Todo lo que realmente necesitamos es añadir la página de devolución de llamada para usarlo.

función artwork\_menu () {$ items ['artwork /% ilustraciones / edit'] = array (

'Title' => 'Editar',

La página de callback '=>' artwork\_page\_edit ',' Página de argumentos '=> array (1),

=> Array 'de acceso argumentos' ('obras de arte Actualizar'), 'peso' => 0,

'Type' => MENU\_LOCAL\_TASK,

'Contexto' => MENU\_CONTEXT\_PAGE | MENU\_CONTEXT\_INLINE,

);

/ / ...

}

función artwork\_page\_edit ($ obra) {$ tipo = artwork\_types ();

**[177 ]**



*Trabajar con contenido*



drupal\_set\_title (t ('Editar <em> @ type </ em> @ title', array ('tipo @' => $ tipos [$ obra de arte-> type] -> nombre, '@ title' => $ obra de arte-> título)), PASS\_THROUGH);

volver drupal\_get\_form ($ obra-> type '\_artwork\_form', $ obra.);

}

La página de edición de devolución de llamada es bastante simple que casi podríamos conseguir lejos con no tenerlo en absoluto y sólo llamando al formulario directamente desde el sistema de menús. Sin embargo, como el nombre del formulario es dinámico basado en el tipo de obra que tiene que tener un evento pequeño. Nótese también que estamos estableciendo un título dinámico y otra vez lo que permite a través de HTML.

Eso es todo! La forma que hemos construido a principios maneja tanto nuevos como obras de arte existentes de la misma, por lo que la edición de una obra de arte, incluyendo el ahorro de nuevas revisiones, se hace.

**Eliminación de una obra de arte**

La pieza que falta final es la capacidad de eliminar una obra de arte. Naturalmente no queremos dejar que los usuarios lo hacen por capricho, así que les daremos un formulario de confirmación.

Recordemos antes que en nuestro formulario de obras de arte de edición que tenía un botón de eliminación con su propio manejador de enviar. Primero tenemos que definir ese manejador presentar:

artwork\_form\_delete\_submit función ($ form, & $ form\_state) {$ destino = array ();

if (isset ($ \_GET ['destino'])) {$ destino = drupal\_get\_destination (); unset ($ \_GET ['destino']);

}

$ Obra = $ form ['# artwork'];

$ Form\_state ['redirect'] = array ('obra de arte /' $ obra de arte-> ayuda "/ delete ', array (' consulta '=> $ destino)..);

}

Todo esto presente manejador no se redirigir al usuario a ilustraciones / $ auxilios / eliminar. El resto del código no es simplemente para manejar sistema de redireccionamiento de Drupal, que en gran medida puede copiar y pegar. En ese camino vamos a poner un formulario de confirmación de modo que los usuarios tienen que confirmar que realmente quieren decir para borrar una obra de arte:

función artwork\_menu () {$ items ['obra de arte de las ilustraciones /% / delete'] = array (

'Title' => 'Eliminar',

La página de callback '=>' drupal\_get\_form ',

La página de argumentos '=> array (' artwork\_delete\_confirm ', 1),

**[178 ]**



*Capítulo 6*



los argumentos de acceso '=> array (' delete obras de arte '),' peso '=> 1,

'Type' => MENU\_LOCAL\_TASK, 'contexto' => MENU\_CONTEXT\_INLINE,

);

/ / ...

}

Llamadas Esta opción del menú drupal\_get\_form directamente, cargando específicamente la artwork\_ delete\_confirm forma creada por la función del mismo nombre:

artwork\_delete\_confirm función ($ form, & $ form\_state, $ obra) {$ form ['# artwork'] = $ obra de arte;

/ / Siempre proporcione Identificación entidad en la misma tonalidad forma como en el formulario de edición entidad.

$ Form ['ayuda'] = array ('# type' => 'valor', '# value' => $ obra de arte-> auxilios); volver confirm\_form ($ form,

t ('¿Está seguro que desea eliminar% title?', array ('% title' => $ obra de arte-> title)),

'Artwork /'. $ Obra de arte-> auxilios,

t ('Esta acción no se puede deshacer. "), t (' Eliminar '),

t ("Cancelar")

);

}

En lugar de construir una forma completa, simplemente vamos a pasar los datos a una función de utilidad de la API de formulario llamado confirm\_form ().confirm\_form () tiene una serie de parámetros: una forma array, una pregunta que hacer, un camino para redirigir a si el usuario cambia de opinión, una etiqueta para el botón Sí lo digo en serio, y una etiqueta para el No, he cambiado mi enlace mente.

La única información del formulario que necesitamos es la obra de arte que desea eliminar. Tampoco se mostrarán, pero tenemos que pasar a lo largo de la devolución de llamada presentar. El resto de los parámetros son simplemente texto que se mostrará al usuario.

Si el usuario envía el formulario de confirmación, entonces sabemos que realmente en serio. La devolución de llamada presentar maneja la supresión de la obra:

artwork\_delete\_confirm\_submit función ($ form, & $ form\_state) {if ($ form\_state ['valores'] ['confirmar']) {

$ Obra = artwork\_load ($ form\_state ['valores'] ['ayuda']); artwork\_delete ($ form\_state ['valores'] ['ayuda']);

**[179 ]**



*Trabajar con contenido*



watchdog ('obra de arte', '@ type:. título eliminado%', array ('type @' => $ obra de arte-> tipo, '% title' => $ obra de arte-> title));

$ tipo = artwork\_types ();

drupal\_set\_message (t ('. @ type% title ha sido borrado', array ('tipo @' => $ tipos [$ obra de arte-> type] -> nombre, '% title' => $ obra de arte-> title))) ;

}

$ Form\_state ['redirect'] = '<front>';

}

Sólo para estar seguro, lo primero que confirmemos que el $ Form\_state ['valores'] ['Confirmar'] propiedad se establece en VERDADERO. Si es así, lo primero que tenemos que hacer es cargar la obra de arte. Eso no es realmente necesario para la operación de eliminación pero necesitamos el objeto obra de arte para mostrar mensajes al usuario y registrar los mensajes de registro. Entonces todo lo que hacemos es llamar

artwork\_delete ().

No debe ser ninguna sorpresa que artwork\_delete () es otra sencilla envoltura:

artwork\_delete función ($ auxilios) {

artwork\_delete\_multiple (array ($ auxilios)) return;

}

función artwork\_delete\_multiple ($ ayudas) {

volver entity\_get\_controller ('obra de arte') -> delete ($ SIDA);

}

Al igual que la carga, la eliminación es una operación de valores múltiples. Podemos eliminar un número arbitrario de obras de arte en una sola operación, y uno es sólo un caso especial de muchos. El trabajo pesado es de nuevo entregado a otro controlador:

public function delete ($ ayudas) {if (! empty ($ ayudas)) {

$ Obras = $ this-> load ($ ayudas, array ()); $ Transaction = db\_transaction ();

try {db\_delete ('obra de arte')

-> Condición ("ayuda", $ ayudas, 'IN') -> execute ();

db\_delete ('artwork\_revision') -> condición ("ayuda", $ ayudas, 'IN') -> execute ();

**[180 ]**



*Capítulo 6*



foreach ($ obras de arte como $ artwork\_id => $ obra) {field\_attach\_delete ('obra de arte', $ obra de arte);

}

db\_ignore\_slave ();

}

catch (Exception $ e) {$ transaction-> rollback ();

watchdog\_exception ('obra de arte', $ e, NULL, WATCHDOG\_ERROR); return false;

}

module\_invoke\_all ('entity\_delete', $ obra, 'obra de arte');

/ / Borrar la página y bloque y cachés de las ilustraciones. cache\_clear\_all ();

$ This-> ResetCache ();

}

return true;

}

Una vez más utilizamos una transacción para asegurarse de que todo el proceso de eliminación, ya sea éxito o fracasa. Primero cargamos los objetos de las ilustraciones que estamos a punto de eliminar, como otras operaciones requerirán ellos. Entonces manejamos nuestra propia lógica eliminación:

La eliminación de los registros apropiados de las tablas obras de arte y artwork\_revision. Después llamamos field\_attach\_delete () para permitir que el sistema de Campo eliminar cualquier dato de campo que se asocian con dichas obras de arte, y, finalmente, desactiva el servidor esclavo que antes.

Hay otra tarea de cuidar; tenemos que vaciar la caché. Por defecto Drupal almacena en caché la mayoría de las solicitudes de página para el rendimiento. Si has eliminado una obra de arte, sin embargo, tenemos que eliminar esa página de la caché. Puede haber muchas otras páginas afectadas por esta eliminación, sin embargo, así que simplemente borrar toda la caché de páginas concache\_clear\_all (). Se obtendrá reconstruido según sea necesario. También borrar la caché estática

de obras de arte cargada de objetos que el controlador mantiene de modo que si más adelante en esta misma petición a alguien intenta cargar la obra que acaba de eliminar consiguen un error en lugar de un objeto fantasma.

Y ya está!

**[181 ]**



*Trabajar con contenido*



**Resumen**

Esto ha sido todo un capítulo. No sólo hemos aprendido cómo crear nuestras propias entidades en Drupal, hemos ampliado nuestro conocimiento de muchos de Drupal

sistemas clave.

* Hemos visto cómo definir nuevas entidades y encapsular sus rutinas de carga y de ahorro en una clase de controlador.
* Hemos visto cómo integrar nuestra entidad con la API de campo, proporcionando a los administradores del sitio con una enorme flexibilidad y poder para construir modelos de datos personalizados.
* Hemos aprendido cómo definir nuevas tablas en la base de datos de Drupal en forma de base de datos-agnóstico.
* Hemos aprendido acerca de las transacciones de bases de datos, una forma sencilla de asegurarse de que una operación de base de datos compleja éxito total o falla completamente para evitar la corrupción de datos.
* Hemos aprendido sobre cómo evitar problemas de datos obsoletos cuando se utiliza la replicación maestro / esclavo.
* Hemos visto cómo se muestran las formularios de confirmación a los usuarios para asegurarse de que no hacen clic en un botón por accidente.
* Hemos aprendido cómo han llamado dinámicamente formularios que se construyen utilizando el mismo proceso.
* Hemos visto cómo aprovechar la capacidad de carga automática de Drupal para las clases para ayudar a mantener el tamaño del código hacia abajo.

Como se señaló al comienzo del capítulo, en muchos casos, una entidad personalizada no será necesario el ciclo de vida completo de la creación, edición y eliminación. Algunos ni siquiera pueden interactuar con bases de datos locales de Drupal en absoluto, confiando en un load () método que llama a un servicio REST o servidor SOAP. Es posible que tengamos que definir un complejo conjunto de controles de permisos para nuestra entidad personalizada, o en algunos casos puede incluso no ser una manera para que los usuarios vean una entidad directamente.

Al observar un ejemplo completo, sin embargo, tenemos una idea de la mayor parte de las piezas que podríamos implementar si necesitábamos. Desde nuestro ejemplo obras de arte se asemeja mucho al tipo de entidad nodo, aunque en una forma simplificada, que ayuda a proporcionar un contexto para el funcionamiento del sistema del nodo de suma importancia en Drupal.

**[182 ]**



Creación de nuevos campos

En el capítulo anterior vimos cómo definir nuevas entidades. La principal ventaja de definir nuestro contenido como una entidad, por supuesto, fue para que sea accesible a la API de Campo. En este capítulo, vamos a ver cómo funcionan los campos y cómo definir nuevos campos para insertarse en cualquiera de nuestras entidades o para las ya definidas, tales como nodos.

En el camino vamos a cubrir:

* Lo API Field es y para qué sirve
* Cómo utilizar las devoluciones de llamada de magia
* El uso de campos para almacenar datos
* El uso de widgets para conectar los campos en el formulario de la API
* Uso de formateadores para controlar la visualización de campos

Cuando hayamos terminado, usted debe tener una comprensión de cómo todas las piezas clave de

Sistema de manejo de datos principal de Drupal encajan entre sí y cómo aprovecharlas mejor.

**Nuestro objetivo: un campo de "dimensiones"**

En el último capítulo, hemos creado un nuevo tipo de entidad llamada ilustraciones. Como entidad fieldable, podemos adjuntar cualquier campo que queremos a ella. A veces sin embargo, todavía tendremos que crear nuestros propios tipos de campo. Eso puede suceder si queremos tratar una determinada pieza de información de forma atómica en lugar de construir hacia arriba a partir de piezas más pequeñas. Hay muchas razones por las que podría ser el caso. Son los siguientes:

1. Queremos tratar conceptualmente ese pedazo de datos como un único trozo con un significado propio y no como una serie de fragmentos que juntos sabemos tiene un significado.
2. Tenemos datos complejos, pero quiero tener varias instancias de los datos en una sola entidad.

*Creación de nuevos campos*



1. Queremos presentar una interfaz personalizada unificada a los usuarios durante la edición de los datos, sobre todo si es de varios valores.
2. Queremos mostrar los datos al usuario en un formato personalizado.

Todas estas son razones por las que puede escribir nuestro propio código de campo.

En nuestro caso, se trata de obras de arte. Sus obras tienen dimensiones, ya sea la altura y el ancho o el alto, ancho y profundidad. Aunque sin duda podríamos simplemente añadir tres campos numéricos para nuestros paquetes de obras de arte y lo llaman un día, eso no es muy atractivo tanto para el editor de contenidos o por visitantes de ese sitio. Se hace aún más feo si queremos permitir que los campos de valores múltiples; decir si una obra determinada es una colección de pequeñas estatuas o una serie de cuadros similares.

Por lo tanto, vamos a definir un nuevo tipo de campo para almacenar dimensiones, ya sea la altura y el ancho o la altura, anchura y profundidad. Aunque en nuestro caso estamos hablando de obras de arte, el campo mismo se aplicaría del mismo modo que a los coches, edificios, animales, o cualquier otro contenido que representa un objeto que ocupa espacio. Un buen tipo de campo es lo suficientemente genérico como para adaptarse a muchas situaciones diferentes.

**¿Cómo funciona la API de campo**

Como se indicó anteriormente, hay varias partes complementarias diferentes a la definición de un campo:

* **Tipo de campo**: Esto es estrictamente hablando, sólo la definición de contenido. Se define el nombre del campo y lo que su estructura de datos interna es, pero no cómo guardarlo o cómo mostrarlo.
* **Campo**: Esta es una configuración particular de un tipo de campo.
* **Instancia de campo**: Esta es la combinación de un campo en particular con un haz o subclase de un tipo de entidad
* **Widget**: Se trata de un elemento de formulario que expone el campo a un editor de contenidos. Podría utilizar los campos de texto simples o ser algo tan complejo como un sistema interactivo

-Basado en flash herramienta.

* **Formatter**: Se trata de una pieza de código que da formato a un campo para la visualización en pantalla.

Normalmente sólo envuelve sistema el tema de Drupal para hacerlo.

Tenga en cuenta que en ninguna parte de cualquiera de las partes mencionadas Cómo definimos como o donde los datos se almacenan. Eso es manejado por un motor de almacenamiento de campo, que se puede configurar de forma independiente por campo. Por defecto todos los campos utilizan un motor de almacenamiento común que salva campos a la base de datos de Drupal. Eso es suficiente para nuestras necesidades, así que no voy a entrar en los motores de almacenamiento de campo en profundidad.

**[184 ]**



*Capítulo 7*



Aunque un tema avanzado, **almacenamiento de campo conectables** es una de las principales novedades de la API de campo y es otra opción para el manejo de las fuentes de datos remotas en Drupal.



**La creación de nuestro nuevo tipo de campo**

Los tipos de campo son definidos por los módulos, así que vamos a empezar por la creación de un nuevo módulo llamado dimfield.module. Su archivo de información es la siguiente:

name = campo Dimensiones

descripcion = altura oferta Field, anchura y profundidad paquete = Drupal 7 Desarrollo

core = 7.x

archivos [] = dimfield.module

**Declarar el campo**

Ahora en dimfield.module, Tenemos que poner en práctica hook\_field\_info (), Que es como le decimos a Drupal acerca de nuestro nuevo tipo de campo.

función dimfield\_field\_info () {return array (

"dimensiones" => array ('label' => t ('Dimensiones'), 'description' => t (

'Este campo almacena una altura y anchura, y profundidad.'), 'Settings' => array ('num\_dimensions' => 2), 'instance\_settings' => array (

'Max\_height' => 0, 'max\_width' => 0, 'max\_depth' => 0,

),

'Default\_widget' => 'dimfield\_combined', 'default\_formatter' => 'dimfield\_default',

),

);

}

**[185 ]**



*Creación de nuevos campos*



Al igual que la mayoría de los "info hooks", esta función devuelve una gran variedad de definiciones, definiendo uno o más campos. Además, como es de esperar, hay una correspondiente hook\_field\_info\_ alterar () hook. En nuestro caso, sólo tenemos el llamado dimensiones. Echemos un vistazo a cada uno propiedad a su vez:

* etiqueta y descripción especificar el nombre legible y explicación de este campo.
* ajustes define un conjunto de opciones de configuración para el campo y su valores por defecto. Estos ajustes son fijos y después de crear una instancia de un campo no se pueden cambiar, así que utilice con precaución. Generalmente solo deseas ajustes de campo si se cambia el ajuste afectaría a cómo se salva de datos.
* instance\_settings es la misma que la matriz de configuración, excepto que puede ser cambiado después de un campo se ha creado. Eso hace que en general se prefiere sobre los ajustes a nivel de campo.
* default\_widget y default\_formatter especificar qué widget y formateador Drupal debe utilizar para un campo determinado antes de que el usuario especifica una. Al igual que los campos, widgets y formateadores tienen nombres de cadena única. Vamos a hablar de cómo escribir aquellos más adelante en este capítulo.

El código anterior le dice a Drupal que hay un nuevo tipo de campo denominado dimensiones definidas por nuestrodimfield módulo, y le da un poco de metadatos al respecto. Sin embargo, Drupal todavía necesita saber que el campo está organizado. Para ello, ponemos en práctica un par de otros hooks.

**Definición de la estructura sobre el terreno**

En realidad, no, no lo hacemos. Aunque llama hooks en la documentación de Drupal, estas funciones son pseudo-hooks: devoluciones de llamada de módulo por arte de magia especiales que se llaman individualmente por Drupal y no junto con el hook y usadas por todos los módulos. Desde nuestro módulo se denominadimfield, El código de soporte para todos los tipos de campo se define en la dimfield módulo vivir juntos en la misma callback magia. Por esa razón, es generalmente una buena idea para no definir demasiados tipos de campo en un solo módulo, el código puede obtener difícil de manejar. También utilizamos un nombre diferente para el módulo y para el tipo de campo para ayudar a mantener un registro de cuándo tenemos que utilizar lo que.



Una devolución de llamada módulo de magia, o pseudo-hook, se ve como un hook, pero se llama individualmente en lugar de junto a las implementaciones de todos los demás módulos activos.



**[186 ]**



*Capítulo 7*



La devolución de llamada magia más importante para un tipo de campo es la devolución de llamada de esquema, su definición se puede ver en el siguiente ejemplo:

dimfield\_field\_schema función ($ field) {if ($ campo ['type'] == "dimensiones") {$ schema ['columnas'] ['height'] = array (

'Type' => 'int', 'not null' => FALSE,

);

$ Schema ['columnas'] ['width'] = array ('type' => 'int',

'No nulo' => FALSE,

);

$ Schema ['índices'] = array ('height' => array ('height'), 'width' => array ('width'),

);

if ($ campo ['settings'] ['num\_dimensions'] == 3) {$ schema ['columnas'] ['profundidad'] = array (

'Type' => 'int', 'not null' => FALSE,

);

$ Schema ['índices'] ['profundidad'] = array ('profundidad');

}

$ Schema ['columnas'] ['unidades'] = array ('type' => 'varchar',

'Length' => 10, 'no nulo' => FALSE,

);

return $ esquema;

}

}

Como era de esperar de un nombre como hook\_field\_schema (), Su valor de retorno es un

Variedad de esquemas de Drupal. Aunque los campos no siempre se guardarán en una base de datos SQL, por lo general son, y es una sintaxis conveniente para reutilizar. Tenga en cuenta que, en este caso, definimos dos columnas de base de datos, para la altura y la anchura, y posiblemente una tercera para la profundidad si nuestro campo está configurado para tener tres dimensiones. (Nos saltamos el apoyo a cuatro o cinco dimensiones, por ahora, ya que es un caso extremo.) La diferencia en la estructura de datos es la razón del número de dimensiones es un campo en lugar de establecer un entorno de instancia del campo.

**[187 ]**



*Creación de nuevos campos*



Dado que las mediciones de la longitud realmente no tienen sentido sin una unidad, también vamos a grabar lo que la unidad de las dimensiones se encuentran, por ejemplo, centímetros o metros. Para simplificar las cosas sólo nos salvaremos enteros, aunque en la práctica nos gustaría apoyar a valores de coma flotante. También tenga en cuenta que toda la función se envuelve en unSi () declaración para comprobar el tipo de campo. Si se tratase de múltiples tipos de campo en este módulo, que definirían su esquema utilizando la misma función y que tendríamos que diferenciar entre ellos sobre la base del valor de $ ['Type'] campo.

**Definición de vacío**

La segunda devolución de llamada magia que necesitamos es determinar si un campo dado tiene un valor vacío. Mientras que puede parecer una pregunta simple, es en realidad depende de nuestra aplicación en particular.

Considere esto: ¿Es un campo de dimensión vacía si no tiene la altura, pero sólo tiene una anchura, o sólo si ambos valores están vacíos? Drupal no sabe lo que queremos decir, así que tenemos que contarla.

función dimfield\_field\_is\_empty ($ item, $ field) {if ($ campo ['type'] == "dimensiones") {

if (empty ($ item ['altura']) && empty ($ item ['width']) && ($ campo ['settings'] ['num\_dimensions'] == 2 | | empty ($ item ['profundidad' ])))

{

return true;

}

}

return false;

}

En el fragmento anterior, definimos vacío en el sentido de que todas las dimensiones en uso son un valor vacío, el cual PHP se define para incluir una cadena vacía o 0. Una vez más observamos que estamos comprobando contra el tipo de campo específico ya que podríamos añadir otro tipo de campo a este módulo posterior.

**Ajustes de campo**

Aunque no es absolutamente necesario, también necesitamos un formulario de configuración para los ajustes de campo. La mayoría de los campos se configuran a través de la interfaz web de Drupal, por lo que necesitamos una forma de permitir a los usuarios configurar las opciones disponibles. Esa es otra de devolución de llamada magia. Veamos un ejemplo:

dimfield\_field\_settings\_form función ($ campo, $ ejemplo, $ has\_data) {if ($ campo ['type'] == "dimensiones") {

$ settings = $ campo ['settings']; $ Form ['num\_dimensions'] = array (

**[188 ]**



*Capítulo 7*



'Type #' => 'select',

'# Title' => t ('¿Cuántas dimensiones'), 'Opciones de #' => array (

2 => t ('Alto y ancho'),

3 => t ('altura, ancho y profundidad),

),

'# Default\_value' => $ ajustes ['num\_dimensions'], '# required' => FALSE,

'# Description' => t (

"¿Esto es para un objeto de 2 dimensiones o 3 dimensiones? '),

);

devolver $ form;

}

}

Sólo tenemos un solo elemento de formulario aquí, es decir, un cuadro de selección que permite al usuario seleccionar si estamos tratando con un objeto de 2 dimensiones o 3 dimensiones. Es este valor que va a determinar la estructura del campo en sí, tal como se define en el esquema de devolución de llamada.

**Validación de campo**

Aunque hay un par de otras devoluciones de llamada que podríamos poner en práctica, sólo hay una que cubriremos por ahora, ya que es bastante importante, a saber, la validación.

dimfield\_field\_validate función ($ obj\_type, $ object, $ field, $ ejemplo, $ langcode, y $ items, y $ errors) {

if ($ campo ['type'] == "dimensiones")'' {$ columnas = array (

'Height' => 'max\_height', 'width' => 'max\_width',);

if ($ campo ['ajustes'] ['num\_dimensions'] == 3) {$ columnas ['profundidad'] = 'max\_depth';

}

foreach ($ artículos como $ delta => $ item) {

foreach ($ columnas como $ columna => $ max\_key) {if ($ instance ['settings'] [$ max\_key] &&

! Vacía ($ item [$ columna]) &&

$ Item [$ columna]> $ instance ['settings'] [$ max\_key]) {$ errores [$ campo ['field\_name']] [$ delta] [] = array (

'Error' => 'dimfield\_'. $ Max\_key, 'message' => t (

'% Name:. La columna% no puede ser mayor que% max', array ('columna%' => $ columna,

'% Name' => $ instance ['label'],

**[189 ]**



*Creación de nuevos campos*



'% Max' => $ instance ['settings'] [$ max\_key]'', x)

),

);

}

}

}

}

}

Al igual que todos los campos se pueden validar de forma individual, por lo que puedo todos los elementos de formulario. Sin embargo, recuerdan que los campos se pueden guardar en cualquier parte del código. Puede que no estemos utilizando una forma en absoluto. Por tanto, debemos validar los propios datos de campo, antes de guardarlo en la

base de datos. En este caso, estamos comprobando para asegurarse de que si la dimensión tiene un valor, y si se establece un máximo para ello, es dentro de ese límite máximo. Si no es así, a continuación,

establecimos un error en el $ errors matriz, que se pasa en por referencia. Ese error consiste, naturalmente, una gran variedad de posibles errores. Corresponde al código de llamada a decidir cómo manejar esa condición de error. Se podría mostrar un mensaje en la pantalla si el error se produce a partir de un formulario de usuario, o puede enviar un objeto de mensaje no válido de nuevo a través de una

Conexión SOAP si el campo (y la entidad está al lado de) está siendo salvados por código desencadenado por un servidor remoto.



Para más amplia información sobre cada uno de los parámetros a las funciones de devolución de llamada API de campo, consulte los ejemplos en el field.api. php presentar en el módulo de campo.



Otro punto importante a señalar aquí es que el campo se pasa una matriz de elementos, no un elemento individual. Desde la perspectiva del código, los campos de Drupal son siempre de valores múltiples. Incluso si sólo hay un valor, incluso si el campo está configurado para permitir solamente un valor, que es todavía de varios valores como la medida de lo nuestro código se refiere. "Uno" es simplemente un caso especial de

"Muchos". Que en realidad se simplifica en gran medida la mayor parte de nuestra lógica, ya que no es necesario para manejar dos posibles casos diferentes. Simplemente podemos repetir con un foreach () lazo sobre nuestros datos y nosotros nos encargaremos de uno o cien valores igual de bien.



Recuerde que los campos en Drupal son siempre una matriz multi-valor en el código. Esa matriz puede tener una sola entrada, pero todavía puede ser tratada como un número arbitrariamente grande de valores.



Una vez más, tenga en cuenta que en ninguna parte en la definición de tipo de campo o apoyar código hacemos realmente para guardar los datos. De hecho, no hay ni una sola consulta SQL. Simplemente estamos describiendo los datos. Aseguramiento de los datos en sí, y decidir dónde guardarlo, es la responsabilidad del sistema de la base. Eso permite una gran flexibilidad, ya que nuestro campo de dimensión ahora se puede utilizar para almacenar los datos en una base de datos SQL local o un servidor SOAP remoto sin ningún cambio de código de nuestra parte.

**[190 ]**



*Capítulo 7*



**La exposición de campos a la API formulario con widgets**

Aunque los campos se pueden almacenar en cualquier lugar (o al menos en cualquier lugar para que escribir un motor de almacenamiento) y se accede en una variedad de formularios, con mucho, el flujo de trabajo de usuario más común es la de crear y editar una entidad que contiene campos usando una forma incrustado en una web página. En Drupal, todas las formularios que se muestran al usuario se controlan mediante el API Form, presentado en el Capítulo 5. La forma en que el sistema de campo se expone a la API de forma es a través de widgets.

Los widgets son simplemente formar fragmentos de API que pueden quedar integradas en una forma más grande de Drupal. Pueden ser muy simples o muy complejas, dependiendo de cómo queremos dar a conocer la información al usuario. De hecho, algunos de los más grandes poderes de widgets

viene del hecho de que los elementos del formulario expone el widget de no tienen que asignar al almacenamiento de la misma tipo de campo en absoluto. Imagine, por ejemplo, un campo que almacena los puntos geográficos. Aunque podríamos simplemente ofrecer al usuario una serie de campos de texto para ingresar

Valores X e Y, que sería mucho más agradable si pudiéramos ofrecerles un mapa interactivo que hacer clic en. Los datos de las coordenadas serían luego son asignadas de nuevo en los valores de X e Y antes de que sea almacenada, sin el campo en sí es toda el más sabio. Con los widgets, que podemos hacer exactamente eso.

**Declarando un widget**

Al igual que con los tipos de campo, los widgets se inician con un hook info:

función dimfield\_field\_widget\_info () {return array (

'Dimfield\_simple' => array (

'Label' => t ('campos de texto separados'), 'description' => t (

'Permitir al usuario introducir cada dimensión por separado.'), 'Los tipos de campo' => array ("dimensiones"),

'comportamientos' => array (

"múltiples valores '=> FIELD\_BEHAVIOR\_DEFAULT,' valor predeterminado '=> FIELD\_BEHAVIOR\_DEFAULT,

),

),

=> Array 'dimfield\_combined' (

'Label' => t ('campo de texto combinado'), 'description' => t (

'Permitir al usuario introducir todas las dimensiones juntos.'), 'Los tipos de campo' => array ("dimensiones"),

'settings' => array ('size' => 10),

**[191 ]**



*Creación de nuevos campos*



'comportamientos' => array (

"múltiples valores '=> FIELD\_BEHAVIOR\_DEFAULT,' valor predeterminado '=> FIELD\_BEHAVIOR\_DEFAULT,

),

),

);

}

En el fragmento anterior, estamos definiendo dos widgets en lugar de sólo uno. El primero es un widget sencillo, que consiste en los campos de texto simples, una para cada dimensión. En el segundo, ofrecemos un único campo de texto en el que el usuario introducirá los dos o tres dimensiones en H x W x D Formato.

Ambos Widgets especificar explícitamente los tipos de campos que van a trabajar. Aunque estamos definiendo estos widgets en el mismo módulo que el tipo de campo, eso no implica necesariamente una relación entre ellos. De hecho, cualquier módulo puede definir

widgets que funcionan con cualquier tipo de campo. Los widget sólo tiene que saber cómo ese tipo de campo quiere que sus datos. La segunda aplicación también incluye una matriz ajustes, lo que nos permite configurar el widget por instancia.

También tenga en cuenta la propiedad comportamientos. De forma predeterminada, los widgets se encargará de un único valor del campo y Drupal ofrecen una forma dinámica para agregar valores adicionales dentro del formulario. Sin embargo, también podemos decirle a Drupal dejar nuestro widget de manejar campos de valores múltiples en caso, por ejemplo, queremos ofrecer un mapa interactivo de valor múltiple coordina hemos comentado anteriormente.

**Formularios simple widget**

Echemos un vistazo a lo simple widget de primero, y luego volver y mirar a la más compleja. El único callback debemos definir un widget es su forma de devolución de llamada, que define los campos de formulario que componen el widget. Veamos un ejemplo:

función dimfield\_field\_widget\_form (forma y $, & $ form\_state, $ field, $ ejemplo, $ langcode, $ items, $ delta, $ element) {

$ Base = $ element;

if ($ instance ['Widget'] ['type'] == 'dimfield\_simple') {$ element ['height'] = array (

'Type #' => 'campo de texto', '# title' => t ('Altura'),

'# Default\_value' => isset ($ items [$ delta] ['altura'])? $ items [$ delta] ['height']: NULL,

) + $ Base;

**[192 ]**



*Capítulo 7*



$ Element ['width'] = array ('# type' => 'campo de texto', '# title' => t ('Ancho'),

'# Default\_value' => isset ($ items [$ delta] ['width'])? $ items [$ delta] ['width']: NULL,

) + $ Base;

if ($ campo ['settings'] ['num\_dimensions'] == 3) {$ element ['profundidad'] = array (

'Type #' => 'campo de texto', '# title' => t ('Profundidad'),

'# Default\_value' => isset ($ items [$ delta] ['profundidad'])? $ items [$ delta] ['profundidad']: NULL,

) + $ Base;

}

$ Element ['unidades'] = array ('# type' => 'select', '# title' => t ('Unidades'),

'# Default\_value' => isset ($ items [$ delta] ['unidades'])? $ items [$ delta] ['unidades']: null,

'Opciones de #' => dimfield\_units (),);

}

return $ elemento;

}

Una vez más, observamos que estamos comprobando que en tu cesta estamos utilizando en esta devolución de llamada, ya que tanto los widgets se utilice el mismo callback. Nuestros parámetros incluyen la forma que se añadirá este widget a y su $ Form\_state. A pesar de que se pasan por referencia, nosotros en realidad no modificaremos directamente (la mayoría de las veces). En su lugar, vamos a devolver un $ Element Fragmento API formulario que Drupal

insertar en el formulario en el lugar correcto. La $ Element que se pasa contiene información básica sobre el widget en sí, que almacenaremos en nuestra propia variable para pasar adelante. El API Formulario ignorará propiedades que no reconoce, pero que los datos estarán disponibles para nosotros más tarde.

**[193 ]**



*Creación de nuevos campos*



En este caso simple, todo lo que estamos haciendo es crear dos o tres elementos de formulario para las dimensiones, uno para cada dimensión, y un cuadro de selección para establecer las unidades. Las unidades disponibles son proporcionados por una función de utilidad simple que también escribimos:

dimfield\_units función ($ unidad = null) {$ estática unidades;

if (empty ($ unidades)) {$ unidades = array (

'pulgadas' => t ('pulgadas'), 'pies' => t ('Pies'), 'm' => t ('Metros'),

);

}

if ($ unidad) {

volver isset ($ unidades [$]) de la unidad? $ unidades [$ unidad]:'';

}

return $ unidades;

}

Esa función de utilidad poco nos permite obtener una lista consistente de unidades apoyamos en cualquier lugar que lo necesitamos, además de que proporciona un mapeo fácil desde el "nombre interno" de una unidad a un nombre de persona amigable traducido.

Es importante señalar que los elementos de forma que estamos creando se denominan exactamente la misma que las columnas de la campo de dimensiones. Drupal necesita la "forma procesada" valor a tener exactamente los mismos nombres "elemento de formulario", como las columnas de campo para que pueda guardarlos correctamente. Lo que hace este un widget simple es que la forma de los mapas de uno-a-uno para la definición del campo, por lo que no tiene que hacer ningún procesamiento adicional. En este punto, estamos en esencia hacer. Los usuarios serán capaces de seleccionar nuestro widget, Drupal se encargará de la lógica multi-valor para nosotros, y guardar los datos en el campo, todo ello sin mayor interacción de nosotros.

**Widgets de complejos**

Veamos ahora en la más compleja de widgets. En este caso, vamos a mostrar todas las dimensiones en un solo campo de texto para que el usuario sólo tiene que rellenar un único campo.

En primer lugar, porque nuestro widget más complejo tiene ajustes que tenemos que poner en práctica, se utiliza elwidget\_settings\_form devolución de llamada, de la forma siguiente:

dimfield\_field\_widget\_settings\_form función ($ campo, $ instance) {$ form = array ();

$ = $ Widget de ejemplo ['Widget']; $ settings = $ widgets ['settings'];

**[194 ]**



*Capítulo 7*



if ($ Widget ['type'] == 'dimfield\_combined') {$ form ['size'] = array (

'Type #' => 'campo de texto',

'# Title' => t ('Tamaño de campo de texto'), '# default\_value' => $ ajustes ['size'], '# required' => true, '# element\_validate' =>

array ('\_element\_validate\_integer\_positive'),

);

}

devolver $ form;

}

Al igual que con todos nuestros otros servicios repetidos, comprobamos el tipo de widget y luego regresar un fragmento forma para el campo que queremos. Tenga en cuenta que el campo de texto tiene el mismo nombre que la propiedad que establezca que definimos en dimfield\_field\_widget\_info (), Que es como Drupal sabe qué configuración es cuál. También estamos aprovechando la capacidad del formulario de API para proporcionar validadores de elementos específicos. En este caso, estamos utilizando una devolución de llamada de validación que ofrece Drupal. Se lanzará un error de validación si el usuario especifica que no sea un número entero positivo nada. (Un widget de tamaño de -7.4 no tendría mucho sentido, ¿verdad?)

Ahora, podemos expandir nuestra field\_widget\_form callback para incluir nuestro nuevo widget.

función dimfield\_field\_widget\_form (forma y $, & $ form\_state, $ field, $ ejemplo, $ langcode, $ items, $ delta, $ element) {

$ Base = $ element;

if ($ instance ['Widget'] ['type'] == 'dimfield\_simple') {/ / ...

}

elseif ($ instance ['Widget'] ['type'] == 'dimfield\_combined') {$ element ['# element\_validate'] = array (

'\_dimfield\_combined\_validate');

$ Default = NULL;

if (isset ($ items [$ delta])) {$ item = $ items [$ delta];

if (isset ($ item ['height'], $ item ['width'])) {$ default = $ item ['height']. 'X'. $ Item ['width']; if ($ campo ['settings'] ['num\_dimensions'] == 3) {

$ Default. = 'X'. $ Item ['profundidad'];

}

**[195 ]**



*Creación de nuevos campos*



}

}

$ Element ['dimfield\_combined\_wrapper'] ['# theme'] = 'dimfield\_combined\_wrapper';

$ Element ['dimfield\_combined\_wrapper'] ['height\_width\_depth'] = array ('# type' => 'campo de texto',

'# Default\_value' => $ valor predeterminado,

'# Tamaño' => $ instance ['Widget'] ['settings'] ['size'],) + $ base;

$ Element ['dimfield\_combined\_wrapper'] ['unidades'] = array ('# type' => 'select',

'# Title' => t ('Unidades'),

'# Default\_value' => isset ($ items [$ delta] ['unidades'])? $ items [$ delta] ['unidades']: null,

'# options' => dimfield\_units (),

);

}

return $ elemento;

}

En el primer bloque de código, nos montamos nuestro valor predeterminado para el elemento de la forma de los valores disponibles en el campo. Desde nuestro widget sólo se encarga de una sola instancia, tenemos que comprobar si hay solo este delta para ver si tenemos un valor definido. De ser así, se concatena la altura, el ancho y la profundidad potencial junto con una × entre ellos.

Entonces establecimos nuestros dos elementos de formulario. Uno de ellos es nuestra altura combinada, el ancho y la profundidad de campo de texto y el otro es las unidades, como hemos visto antes. La parte más importante, sin embargo, es que la primera línea:

$ Element ['# element\_validate'] = array ('\_dimfield\_combined\_validate');

Así como hemos especificado una devolución de llamada de validación existente para un campo de texto hace un momento, esta vez vamos a especificar una devolución de llamada de validación personalizada. Sin embargo, no vamos a usar sólo para la validación. Más bien, vamos a utilizar para modificar los valores de los formularios presentados. Vamos a echar un vistazo a esa función dada aquí:

\_dimfield\_combined\_validate función ($ element, & $ form\_state) {

* Esta función también se conoce como el momento de presentar el campo
* formulario de configuración. Si es así, omita la validación, ya que
* no funcionará de todos modos.

if ($ form\_state ['forma completa'] ['# form\_id'] ==

**[196 ]**



*Capítulo 7*



'Field\_ui\_field\_edit\_form') {

volver;

}

$ values ​​= $ form\_state ['valores']; $ Language = $ valores ['language']; $ Field\_name = $ element ['# field\_name'];

$ num\_dimensions = 2;

if (array\_search ('profundidad', $ element ['# columnas'])) {$ num\_dimensions = 3;

}

foreach ($ valores [$ field\_name] [$] lingüísticas como $ delta => $ item) {

if (substr\_count ($ item ['dimfield\_combined\_wrapper'] ['profundidad height\_width\_'], 'x') == $ num\_dimensions - 1) {

if ($ num\_dimensions == 2) {

list ($ altura, $ width) = explode ('x', $ item ['dimfield\_combined\_ envoltura'] ['height\_width\_depth']);

$ New\_values ​​= array ('height' => trim ($ height), 'width' => trim ($ width),

'Unidades' => $ item ['dimfield\_combined\_wrapper'] ['unidades'],

);

}

elseif ($ num\_dimensions == 3) {

list ($ altura, $ anchura, $ profundidad) = explode ('x', $ item ['dimfield\_combined\_wrapper'] ['height\_width\_depth']); $ New\_values ​​= array (

'Height' => trim ($ height), 'width' => trim ($ width), 'profundidad' => trim ($ profundidad),

'Unidades' => $ item ['dimfield\_combined\_wrapper'] ['unidades'],

);

}

form\_set\_value ($ element, $ new\_values, $ form\_state);

}

else {

form\_set\_error ($ field\_name, t ('Debe especificar todas las dimensiones, separadas por un \' x \) '.');

}

}

}

**[197 ]**



*Creación de nuevos campos*



Durante la fase de validación del envío del formulario, esta función será llamada con el elemento que se une a (el height\_width\_depth elemento) y el $ Form\_state variable, que se pasa por referencia, para que podamos modificarlo. La Lo primero que comprobamos es que no vamos a mostrar este widget en la página de configuración de campo. Si es así, no nos preocupamos validarlo porque nada se guardará de todas formularios.

Luego, comprobamos para ver cuántas dimensiones que estamos tratando ya que la lógica será un poco diferente. A continuación, iterar sobre cada valor presentado y, en el supuesto de que tiene el requisito× carácter en el mismo, romper la cadena presentada en tres enteros. La explode () función en PHP tomará una cadena, que se dividió en una matriz usando

el primer parámetro como un delimitador, mientras que el list () operador asignará esa matriz a dos o tres variables separadas para nosotros. Entonces tomamos esos valores y establecer activamente la altura, anchura, unidades y valores de profundidad potenciales dentro del estado de formulario utilizando

form\_set\_value ().

Aunque parece extraño para utilizar el paso de validación de manipular los datos del formulario, que es el único lugar en el que la forma de la API nos permite hacerlo. El resultado neto es que creamos nuevos valores en el $ Form\_state colección que coincide con las columnas en nuestro campo.

Cuando Drupal presenta el widget, que se verá a través de la$ Form\_state para las variables que responden a los nombres de las columnas en el campo. No le importa que pongamos esos valores que nosotros mismos, al igual que existen es lo que importa. La cadena original todavía existe en la height\_width\_depth variable, pero Drupal simplemente lo ignorará.

También vamos a hacer un poco de tematización costumbre de nuestro widget combinado. Tenga en cuenta las siguientes líneas:

$ Element ['dimfield\_combined\_wrapper'] ['# theme'] = 'dimfield\_combined\_ envoltorio';

$ Element ['dimfield\_combined\_wrapper'] ['# adjunto'] ['css'] [] = drupal\_ get\_path ('módulo', 'dimfield'). '/ Dimfield-admin.css';

La primera línea le dice al sistema de representación de utilizar un hook tema llamado dimfield\_ combined\_wrapper para hacer todo lo que aparece debajo de $ elemento ['dimfield\_ combined\_wrapper']. El segundo le dice al sistema que también cargar un archivo CSS particular cuando se muestra este elemento de formulario. En nuestro caso vamos a hacer algo sencillo y sólo se adhieren los dos forman elementos height\_width\_depth y unidades en un conjunto empaquetado de divs:

función dimfield\_theme () {return array (

'Dimfield\_combined\_wrapper' => array ('render elemento' => 'elemento',

),

);

**[198 ]**



*Capítulo 7*



}

función theme\_dimfield\_combined\_wrapper (variables $) {$ element = $ variables ['elemento'];

$ Hwd = drupal\_render ($ element ['height\_width\_depth']); $ unidades = drupal\_render ($ elemento ['unidades']);

volver <<< END

<div class="clearfix">

<div class="dimfield-combined"> {$ hwd} </ div> <div class="dimfield-units"> {$ unidades} </ div> </ div>

END;

}

Todos los conjuntos de elementos de formulario tienen el mismo aspecto que una función de tema: que se pasan como una sola matriz llamada $ Element. Entonces tomamos los dos componentes que conocemos constituyen la matriz completa, rinde por separado, y pegarlas en un conjunto de divs. El archivo CSS adjuntamos antes hará que los divs aparecen al lado del otro, la creación de una interfaz de usuario mucho más atractivo. El archivo CSS es muy simple:

. Dimfield combinado-{float: left; margin: 0 30px 0 0;

}

Al tomar ventaja de la forma en Drupal busca y guarda los datos del formulario, somos capaces de desarrollar cualquier tipo de artilugio arbitrariamente compleja que queremos. Incluso podríamos tener un widget

que muestra nada en la pantalla en todo, pero asigna un valor durante su fase de validación en base a algunos datos de terceros, cualquier otro campo de la misma forma, la información de la URL, o incluso la hora del día. Drupal diligentemente guardar los datos, sin importarle cómo llegó allí, siempre y cuando nuestro widget le dio el nombre de Drupal esperaba.

**Usando formateadores para mostrar nuestro campo**

Ahora que hemos definido nuestro tipo de campo, y hemos creado un widget para que sea editable desde un formulario, la única pieza que queda es decidir cómo mostrar en la salida de usuario. (Salida de usuario por lo general significa que la pantalla del ordenador, pero también podría significar una fuente RSS, página impresa, o varios otros tipos de salida). Drupal nos permite controlar que la pantalla utilizando los formateadores.

**[199 ]**



*Creación de nuevos campos*



Formateadores siguen un patrón muy similar a los tipos de campo y widgets. Hay un hook de información para definir lo que se dispone de formateadores, y luego hay una serie de devoluciones de llamada para todos los formateadores nuestro módulo define. En la mayoría de los casos, sin embargo, sólo hay una devolución de llamada que necesitamos preocuparse.

**Declarar un formateador**

En primer lugar, echemos un vistazo a el hook información dada aquí:

función dimfield\_field\_formatter\_info () {return array (

'Dimfield\_default' => array ('label' => t ('Default'),

'tipos de campo' => array ("dimensiones"),

),

'Dimfield\_table' => array ('label' => t ('Mostrar como table'),

'tipos de campo' => array ('dimensiones'), 'Ajustes' => array ('units\_as' => 'columna'),

),

);

}

En el fragmento anterior, definimos dos formateadores, y no hay mucho de definir. Cada formateador tiene un nombre interno definido por la clave de la matriz, una etiqueta legible para el usuario, y una lista de los tipos de campo que se aplica. Al igual que con los widgets, podemos definir un formateador en cualquier módulo que funciona con cualquier tipo de campo que queramos, siempre y cuando sabemos cómo manejar los datos que nos da.

**Formateadores un solo valor**

Formateadores sólo tienen dos devoluciones de llamada, y la mayoría de los formateadores sólo utilizará uno. Una vez más, echemos un vistazo a la sencilla aplicación primero.

dimfield\_field\_formatter\_view función ($ obj\_type, $ object, $ field, $ ejemplo, $ langcode, $ items, $ display) {

$ Element = array ();

$ settings = $ display ['settings'];

switch ($ display ['type']) {case 'dimfield\_default':

foreach ($ artículos como $ delta => $ item) {

if ($ campo ['settings'] ['num\_dimensions'] == 2) {= $ salida t ('@ @ la altura de la unidad por @ @ ancho de unidad', array (

**[200 ]**



*Capítulo 7*



'@ Height' => $ item ['height'], '@ width' => $ item ['width'],

'Unitarios @' => dimfield\_units ($ item ['unidades']),));

}

elseif ($ campo ['settings'] ['num\_dimensions'] == 3) {= $ salida (t

'@ @ Unidad altura por anchura @ @ @ por unidad de profundidad @ unidad', array ('@ height' => $ item ['height'],

'@ Width' => $ item ['width'], '@ profundidad' => $ item ['profundidad'],

'Unitarios @' => dimfield\_units ($ item ['unidades']),));

}

$ Element [$ delta] = array ('# markup' => $ salida);

}

romper;

}

return $ elemento;

}

La formatter\_view Se espera callback para devolver un array renderizable que el tema del sistema puede entender. En este caso, simplemente queremos una cadena con formato que describe los datos almacenados en el campo. Al igual que antes, tenemos varios valores de campo que se pasan a la devolución de llamada a la vez en una matriz. Así que simplemente iterar sobre ellos uno por uno y los asignamos a la $ Element variable. La # Markup tipo de elemento dice Drupal "Aquí hay algo de HTML. Ya he formateado, simplemente lo utilizan". Cuando este elemento se vuelve más tarde, en la página, las cuerdas se generaron utilizando la t () función simplemente conseguir que se muestra con todas las indicaciones en ellos.

**Formateadores complejos**

Hay, por supuesto, no hay nada que nos impida la prestación de todos los valores juntos si queremos. De hecho, nuestro segundo formateador hará exactamente eso. En lugar de una serie de valores, uno tras otro, nos rendimos todos los valores disponibles en una sola tabla.

Entonces surge la pregunta, ¿cómo nos mostramos unidades? A medida que su propia columna? En línea en cada celda? Sólo en la cabecera de cada columna de dimensión? En casos como este, la mejor opción es dejar que el usuario decida utilizar las capacidades de configuración de formateadores.

**[201 ]**



*Creación de nuevos campos*



Recordemos que hace un momento que el dimfield\_table formateador declarada ajustes clave, que era una matriz. Esa matriz define todos los parámetros posibles ajustes para que formateador y sus valores predeterminados. Con el fin de hacer uso de la configuración del formateador también hay otros dos hooks que necesitamos para poner en práctica: hook\_field\_formatter\_

settings\_summary ()y hook\_field\_formatter\_settings\_form ().

dimfield\_field\_formatter\_settings\_form función ($ campo, $ ejemplo, $ view\_mode, $ form, & $ form\_state) {

$ Display = $ instance ['display'] [$ view\_mode]; $ settings = $ display ['settings'];

$ Form = array ();

if ($ display ['type'] == 'dimfield\_table') {$ form ['units\_as'] = array (

'# Title' => t ('Mostrar unidades'), 'type #' => 'select', 'Opciones de #' => array (

'Columna' => t ('A medida que su propia columna'), 'célula' => t ('En cada celda'),

'None' => t ('No mostrar unidades),

),

'# Default\_value' => $ ajustes ['units\_as'], '# required' => TRUE,

);

}

devolver $ form;

}

dimfield\_field\_formatter\_settings\_summary función ($ campo, $ ejemplo, $ view\_mode) {

$ Display = $ instance ['display'] [$ view\_mode]; $ settings = $ display ['settings'];

$ Summary ='';

if ($ display ['type'] == 'dimfield\_table') {if ($ ajustes ['units\_as'] == 'columnas') {

$ Summary = t ('Mostrar unidades como su propia columna');

}

else if ($ ajustes ['units\_as'] == 'celulares') {$ summary = t ('Mostrar las unidades en cada celda');

}

**[202 ]**



*Capítulo 7*



else if ($ ajustes ['units\_as'] == 'none') {$ summary = t ('No mostrar unidades);

}

}

return $ sumaria;

}

El hook de la forma es una forma muy simple que ofrece al usuario un cuadro de selección para elegir lo que el units\_as configuración debería ser: columna, celda, o ninguno. Al igual que con otras formularios de ajustes, el nombre del elemento del formulario coincide con el nombre de la variable de configuración de lo que se guarda automáticamente. El hook de resumen, entonces, simplemente toma esa configuración y devuelve una cadena que Drupal puede mostrar al usuario para que sepa lo que es el ajuste actual.

Ahora vamos a echar un vistazo al código de enlace vista para el formateador tabla:

dimfield\_field\_formatter\_view función ($ obj\_type, $ object, $ field, $ ejemplo, $ langcode, $ items, $ display) {

$ Element = array ();

$ settings = $ display ['settings'];

switch ($ display ['type']) {/ / ...

case 'dimfield\_table': $ filas = array ();

foreach ($ artículos como $ delta => $ item) {$ row = array ();

if ($ ajustes ['units\_as'] == 'celulares') {$ row [] = t ('@ valor (% de unidades)', array (

'@ Valor' => $ item ['height'],

'% de unidades' => dimfield\_units ($ item ['unidades']),));

$ Fila [] = t ('@ valor (% de unidades)', array ('@ valor' => $ item ['width'],

'% de unidades' => dimfield\_units ($ item ['unidades']),));

}

else {

$ Row [] = $ item ['height']; $ Row [] = $ item ['width'];

}

if ($ campo ['settings'] ['num\_dimensions'] == 3) {if ($ ajustes ['units\_as'] == 'celulares') {

**[203 ]**



*Creación de nuevos campos*



$ Row [] = t ('@ valor (% de unidades)', array ('@ valor' => $ item ['profundidad'],

'% de unidades' => dimfield\_units ($ item ['unidades']),));

}

else {

$ Row [] = $ item ['profundidad'];

}

}

if ($ ajustes ['units\_as'] == 'columnas') {$ row [] = dimfield\_units ($ item ['unidades']);

}

$ filas [] = $ fila;

}

$ Header = array (t ('Altura'), t ('Ancho'));

if ($ campo ['ajustes'] ['num\_dimensions'] == 3) {$ header [] = t ('Depth');

}

if ($ ajustes ['units\_as'] == 'columnas') {$ header [] = t ('Unidades');

}

$ Element = array ('# lema' => 'table', '# filas' => $ filas => $ header '# header',

);

romper;

}

return $ elemento;

}

En este formateador, construimos una serie de filas de datos que constan de altura, anchura y profundidad en su caso. Cada fila es una matriz, y luego tenemos una matriz de matrices para darnos una estructura de tabla. Esa matriz que asignamos a un elemento que logrará rendido como tabla, Tal como se define por el # Tema clave. Debido a Drupal define un theme\_table () función, que los datos se van pasando a la función cuando se procesa el elemento y devuelve como una tabla HTML.

Tenga en cuenta también que estamos cambiando la forma en la mesa se construye sobre la base de los ajustes configuramos hace un momento. Puede o no puede haber una columna dedicada a las unidades, y que pueden o no pueden mostrar las unidades como parte de cada célula. Debido al hecho de que un ajustes valor es exclusivo, no es necesario que preocuparse de esos dos chocar como tanto nunca puede ser verdad.

**[204 ]**



*Capítulo 7*



Podemos, por supuesto, tienen mucho más formateadores complejos. Para utilizar nuestro mapa de ejemplo que hemos visto anteriormente, un formateador podría tomar una serie de datos de coordenadas y salida de ellos, sobre un mapa mediante un servicio de mapas. Mientras volvemos una matriz renderizable

a Drupal, podemos hacer lo que queramos.

**La gestión de los campos no Field**

Una de las ventajas de utilizar el sistema de campo es que nos da una interfaz de usuario consistente y de gran alcance para todos los datos de campo que añadimos a una entidad. Sin embargo, una entidad puede tener datos que no son de campo sobre ella también. Un nodo o título obra de arte, por ejemplo, no es un campo. Laencuesta módulo en el núcleo no utiliza los campos para la configuración de sondeo. Una entidad que es ser tomado de un tercer sistema de partidos puede tener todo tipo de datos asociados a ella, que no viven en un módulo de campo.

Afortunadamente, Drupal ofrece una manera para que integremos estos datos en la interfaz de usuario de Campo. Volvamos al módulo de obras de arte a partir del capítulo 6 Trabajar con contenido e integrar el título en el sistema de campos. Se necesita más que un solo hook:

artwork\_field\_extra\_fields function () {$ adicional = array ();

foreach (artwork\_types () como $ tipo) {$ adicional ['obras de arte'] [$ tipo-> type] = array (

"Forma" => array ('title' => array (

'Label' => t ('Título'),

'Description' => t ('El nombre de la obra'), 'peso' => -5,

),

),

'Display' => array ('title' => array (

'Label' => t ('Título'),

'Description' => t ('El nombre de la obra'), 'peso' => -5,

),

),

);

}

return $ adicional;

}

**[205 ]**



*Creación de nuevos campos*



hook\_field\_extra\_fields () nos permite definir, según el tipo de entidad y por paquete, lo que "campos adicionales" el sistema de campo debe tener en cuenta. En nuestro caso, lazo sobre todos los tipos de obras de arte y declarar que todos ellos tienen un pseudo-campo de título. Esa pseudo-campo está presente tanto cuando se muestra el formulario de edición para las entidades de las ilustraciones (la primera tecla) y al mostrarlos (la segunda tecla). En ambos casos, la definición es la misma, pero que fácilmente podría hacerlos diferentes si queremos.

Sólo hay tres claves para cada definición.etiqueta y descripción debe explicarse por sí mismo. La peso clave define lo que el "peso" por defecto de la pseudo-campo del título es en relación con otros campos (o pseudo-campos) en una entidad. Al establecer el peso a -5, nos aseguramos de que de forma predeterminada el título será mostrado por primera vez.

No existe ningún requisito de que permanezcan allí, sin embargo. Ahora que el sistema de campo sabe sobre el campo del título, el usuario puede arrastrar y soltar fácilmente el título que aparecerá en medio de dos campos diferentes, en la parte inferior de la página, o incluso nada en absoluto.

**Encontrar a entidades y campos**

Creación, modificación y supresión de los datos está bien y es bueno, y es una parte fundamental de cualquier sistema de gestión de contenidos. Sin embargo, hay también otra característica importante que hace que una pena-Buscando CMS.

En un caso muy simple, la búsqueda de datos es fácil. Tenemos una base de datos, sabemos SQL, así que vamos a rock and roll. Sin embargo, Drupal no restringe los datos a vivir en un

Base de datos SQL. De hecho, con los motores de almacenamiento de campo una sola entidad podía vivir concebible en una variedad de diferentes lugares. Consideremos el caso de una obra que tiene imágenes asociadas a él a través de un campo que atrae imágenes de Flickr, así como información general adicional de un tercer sistema de gestión de colección de fiesta. Basta recurrir a SQL para encontrar obras de arte que coincidan algunos criterios dados no es factible, ya que dos de los tres almacenes de datos no es una base de datos SQL. Por otra parte, no puede

ser capaz de predecir la estructura de la tabla de la que es desde Drupal crea su base de datos de forma dinámica.

Afortunadamente, Drupal proporciona un sistema unificado de consultas para buscar entidades y campos.

A pesar de que no permite la búsqueda a través de los diferentes almacenes de datos, proporciona un almacén de datos forma agnóstica de la búsqueda de entidades. (Buscando a través de diferentes almacenes de datos es un problema muy complejo, por lo general mejor resuelto por la indexación de todos los datos de varios almacenes de datos en un único índice de búsqueda, tales como Apache Solr.)

**[206 ]**



*Capítulo 7*



Para demostrar cómo funcionan las consultas de entidad y de campo, vamos a empezar por la creación de un nuevo elemento de menú para listar las obras de arte de reciente creación. Lo pondremos en el módulo de obra por ahora.

función artwork\_menu () {/ / ...

$ items ['artwork / reciente'] = array ('title' => 'obras de arte' Recientemente añadido,

La página de callback '=>' artwork\_page\_list\_recent ',' argumentos de acceso '=> array (' ver obras de arte '),' file '=>' artwork.pages.inc ',

);

return $ artículos;

}

Así como Drupal proporciona un generador de consultas de bases de datos SQL que abstrae la lógica específica de la base de datos, sino que también proporciona un generador de consultas de entidades y campos.

La API para ello es una sola clase llamada, aburridamente suficiente, EntityFieldQuery.

Antes de echar un vistazo a EntityFieldQuery directamente, vamos a echar un paso atrás y considerar qué tipo de cosas que podemos buscar en. A nivel conceptual, hay tres "niveles" de los datos por los cuales podemos filtrar:

* **Datos de nivel de las Entidades**: Se trata de datos que son comunes a todas las entidades de todo tipo. Esto incluye el propio tipo de entidad, el nombre del paquete, el identificador de la entidad, y el ID de revisión (si procede). Todas las entidades de cualquier tipo tendrán estos elementos.
* **Propiedades**: Son los elementos de datos que son comunes a todos los objetos de un tipo de entidad dada, pero no a todas las entidades. Es decir, se trata de piezas de datos

común a todos los nodos, o para todos los usuarios, o para todas las obras de arte. Los ejemplos incluyen el "título del nodo" y "el uid creador de nodos, el usuario 'nombre de usuario' para las entidades usuarias, y el 'title artwork" y "fecha de creación" de obras de arte.

* **Los campos**: Ellos son, por supuesto, específica a una definición de paquete dado (pintura o escultura). Sin embargo, también pueden ser compartidos por las entidades de distinto tipo.

Durante la búsqueda de entidades, podemos filtrar por u orden de los datos en cada uno de esos niveles.

No todas las combinaciones tienen sentido, sin embargo, y la consulta rechazarán combinaciones absurdas iniciando una excepción.

Dado que no podemos saber lo que el motor de almacenamiento o motores son que la entidad utiliza, eso también limita la complejidad de las búsquedas que podemos hacer. Por ejemplo, sólo somos capaces de hacer "y" búsquedas, no en condiciones complejas con OR. Tampoco podemos buscar a través de diferentes almacenes de datos. Sin embargo, eso todavía deja una enorme variedad de casos de uso que pueden ser resueltos fácilmente.

**[207 ]**



*Creación de nuevos campos*



Utilice siempre EntityFieldQuery en la búsqueda selectiva de entidades. Nunca trate de consultar la base de datos directamente, ya que no hay garantía de que no existe una base de datos relacional implicado.



Vamos a empezar con simplemente mostrando las cinco obras de arte de más reciente creación:

función artwork\_page\_list\_recent () {$ contenido = array ();

$ Consulta = new EntityFieldQuery (); $ Consulta

-> EntityCondition ('ENTITY\_TYPE', 'obra de arte') -> propertyOrderBy ('creado', 'DESC') -> rango (0, 5);

$ Resultado = $ query-> execute ();

$ obras = artwork\_load\_multiple (array\_keys ($ resultado ['obra de arte'])); foreach ($ obras de arte como $ obra) {

$ Contenido [] = artwork\_page\_view ($ obra, 'Sueños de juventud');

}

return $ contenido;

}

Comenzamos creando un nuevo objeto de la consulta. Luego llamamos a los métodos en él, para filtrar por "tipo de entidad es 'obra de arte'", y luego "orden directamente en el hotel 'creado', el más reciente en primer lugar"

(Es decir, el orden descendente). La range () método, al igual que DB queries, toma la posición inicial como primer parámetro y el conde como su segundo.

Si asumimos una base de datos SQL, SQL resultante tendrá un aspecto similar al siguiente:

SELECT artwork.aid AS entity\_id, artwork.vid AS revision\_id, obras de arte. escriba los paquetes AS, 'obra de arte' AS ENTITY\_TYPE

DESDE

obras de arte obras de arte

ORDER BY DESC LIMIT artwork.created 5 OFFSET 0

La ventaja aquí es que, si el autor de las ilustraciones del módulo de cambiar la estructura de la tabla de nosotros, o si las obras de arte fueron almacenadas en un almacén de datos no es de SQL, la consulta SQL anterior se rompería horriblemente pero el EntityFieldQuery seguirá trabajando porque Drupal construye que dinámicamente en función de lo que estamos buscando.

**[208 ]**



*Capítulo 7*



Ahora mira a esa pregunta otra vez. Está regresando tres propiedades, el identificador de entidad, id de revisión, y el nombre de paquete. Todas las consultas de entidad devuelven esos mismos datos: el tipo de entidad, id entidad, id de revisión y el paquete. Cada registro está codificado por su identificador de entidad, que a su vez forma impide por el tipo de entidad (ya que es posible volver a múltiples tipos de entidades

en una sola consulta).

Porque sabemos exactamente lo que es la estructura, podemos aprovechar fácilmente. En este caso queremos cargar completamente todas las obras que encontramos, por lo que se corre array\_keys () en la $ Resultado ['obra de arte'] matriz, para obtener un conjunto de sólo esos identificadores y luego cargarlos. Tenga en cuenta que estamos usando artwork\_load\_multiple (). En lugar de cargar cada obra por separado, y corriendo lo son necesarias consultas a hacerlo, cargamos todos a la vez. Eso significa que uno o cinco (o cincuenta si nos permitimos que muchos) Resultados Todas tienen aproximadamente la misma cantidad de tiempo para cargar.

Una vez que tenemos nuestros objetos de las ilustraciones, simplemente obtener la matriz hacen en representación de cada uno usando la funcionalidad que construimos en el capítulo 6 y unirlos, a continuación, devolver la matriz resultante. Cuando se dictó veremos todas nuestras obras de arte, uno tras otro.

Ahora vamos a hacer un pequeño cambio en nuestra consulta de campo:

$ Consulta = new EntityFieldQuery (); $ Consulta

-> EntityCondition ('ENTITY\_TYPE', 'obra de arte') -> propertyOrderBy ('creado', 'DESC') -> fieldCondition ('field\_artist', 'valor', 'Da Vinci') -> rango (0, 5);

$ Resultado = $ query-> execute ();

Aquí, también filtramos por sólo aquellas obras de arte que tienen un campo denominado field\_artist, La valor la columna de la que es exactamente igual a la cadena Da Vinci. El resto del código es el mismo, pero ahora vamos a conseguir menos resultados. Si asumimos que todo se almacena en la base de datos SQL local, entonces la consulta podría ser algo como esto:

SELECT field\_data\_field\_artist0.entity\_id AS entity\_id, field\_artist0.revision\_id field\_data\_ AS revision\_id, field\_data\_field\_artist0. agrupar los paquetes AS, AS fcet.type ENTITY\_TYPE

DESDE

field\_data\_field\_artist0 field\_data\_field\_artist

INNER JOIN field\_config\_entity\_type fcet EN fcet.etid = field\_data\_ field\_artist0.etid

INNER JOIN obras de arte obras de arte en artwork.aid = field\_data\_field\_artist0. entity\_id

WHERE (field\_data\_field\_artist0.field\_artist\_value = 'Da Vinci') AND (field\_data\_field\_artist0.deleted = 0) Y (fcet.type = 'obra de arte') ORDER BY DESC artwork.created

LÍMITE 5 OFFSET 0

**[209 ]**



*Creación de nuevos campos*



Una vez más, no hay ninguna garantía de que incluso estamos tratando con una base de datos SQL. ¿No estás contento estamos dejando figura Drupal todo eso para nosotros?

También es posible hacer consultas más complejas en los campos, por ejemplo, la siguiente:

$ Consulta = new EntityFieldQuery (); $ Consulta

-> EntityCondition ('ENTITY\_TYPE', 'obra de arte') -> propertyOrderBy ('creado', 'DESC') -> fieldCondition ('field\_artist', 'valor', 'Da Vinci', 'CONTIENE',

0)

-> Rango (0, 5);

$ Resultado = $ query-> execute ();

En este caso, en lugar de utilizar un simple "es exactamente igual a la" comparación, estamos pidiendo una entidad cuya field\_artist campo tiene la cadena "Da Vinci" en cualquier lugar. Eso nos permite mantener a la vez "Da Vinci" y "Leonardo Da Vinci". El quinto parámetro nos permite restringir los resultados a sólo aquellos campos que tienen "Da Vinci" como su primer caso (recuerda, basado en 0) si son de valor múltiple.

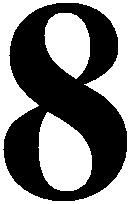
Hay, por supuesto, que correspondeentityOrderBy (),PropertyCondition (), Y fieldOrderBy () métodos para la creación de consultas de campo más interesantes. Consulte la documentación en línea en includes / entity.inc para toda la gama de opciones.

**Resumen**

Hemos completado un módulo que aborda las tres partes más importantes de la API de Campo. Hemos creado un nuevo tipo de campo para almacenar los datos, los widgets correspondientes para permitir a los usuarios editar, y formateadores para controlar cómo se muestran los datos cuando la entidad del campo está fijado a es visto.

Aunque nuestros casos de uso eran razonablemente simple, Drupal permite Widgets y Formateadores arbitrariedad compleja. Algunos widgets pueden interactuar con el URL o fuentes de datos de terceros para manejar los valores por defecto, o tal vez mostrar un conjunto completamente diferente de los campos del formulario, bajo ciertas condiciones. Formateadores pueden utilizar el tema del sistema para mostrar los datos propios o de apalancamiento bibliotecas de JavaScript para crear visualizaciones interactivas de los datos almacenados en un campo. Con los motores de almacenamiento intercambiables, los casos de uso avanzados incluso pueden cargar datos de otra base de datos o el servidor completo, incluyendo uno que no implique SQL.

**[210 ]**



Drupal Permisos

y Seguridad

Permisos encuentran en el centro del paradigma de seguridad de Drupal. En pocas palabras, los permisos determinan quién puede realizar la acción que en una página web. Por lo general, los permisos permiten a los usuarios acceder (o ser negado el acceso) a las características específicas, como el acceso al formulario de contacto en todo el sitio o la capacidad de cambiar el autor de una pieza de contenido.

Estos permisos no se asignan a usuarios individuales, pero en su lugar a las clases de usuarios, definidas como funciones. Un rol es un conjunto de permisos. Los usuarios individuales pueden entonces ser asignados a una o más funciones, como corresponde a las reglas de negocio de su proyecto.



Nota: Al asignar permisos a los roles, el valor predeterminado "autenticado el usuario" papel es un caso especial. Cualquier permiso concedido a este rol también se otorgará a cualquier otra función, excepto "usuario anónimo". ¿Por qué? Esto es debido a que el acto de inicio de sesión en una cuenta de usuario define un usuario como "autenticado".

Las funciones personalizadas creadas para un sitio heredan los permisos básicos asignados a los usuarios a seguir "usuario autenticado".



La comprensión de este comportamiento es fundamental para los constructores de sitio, por lo que es crucial para los desarrolladores de módulos. Puede que tenga que crear permisos muy específicos con el fin de satisfacer la lógica de negocio que su módulo requiere. En nuestro análisis, vamos a explorar los problemas comunes que pueden ocurrir cuando los permisos son demasiado amplias o demasiado estrecha.

Cualquier módulo puede establecer nuevos permisos. En este capítulo, vamos a discutir las mejores prácticas para la seguridad y la facilidad de uso cuando se definen opciones de permiso de su módulo.

*Drupal permisos y seguridad*



En este capítulo, vamos a cubrir lo siguiente:

* Roles y permisos conceptos de Drupal
* Uso user\_access () hacer valer permisos
* Uso hook\_permission ()
* Control de acceso con hook\_menu ()
* Errores comunes en la definición de permisos
* Declarar sus propias funciones de acceso
* Asegurar acciones sensibles con permisos
* Responder cuando se niega el acceso
* Activación de permisos mediante programación
* Formularios de permisos, seguridad, y Drupal
* Consideraciones de seguridad para el procesamiento de AJAX

**Usando user\_access () para hacer valer los permisos**

La user\_access () función es el elemento de seguridad primario en la API de Drupal. La mayoría de las solicitudes de páginas pasan a través de la función, al igual que muchas de las funciones administrativas y la presentación de determinados elementos de la página. Páginas, bloques, campos y elementos de formulario son algunos de los elementos que pueden mostrarse u ocultarse envolviendo su exhibición en un user\_access () llamar.

La función es muy elemental, teniendo sólo dos argumentos:

user\_access ($ string, $ cuenta = NULL)

Aquí, $ String es el nombre legible por la máquina de la autorización, y $ Cuenta es una copia opcional de un $ Usuario oponerse, como el devuelto por la función user\_load ().

La siguiente es una comprobación de acceso típico, tomada desde el módulo de Menú:

$ ['Menú'] form = array ('# type' => 'fieldset',

'# Title' => t ('Configuración' del menú),

'# Access' => user\_access ('administrar menú'), '# plegable' => TRUE,

'# Derrumbó' =>! $ Enlace ['link\_title'],

);

**[212 ]**



*Capítulo 8*



El código anterior comprueba si el usuario edita una página puede añadir un enlace a esa página en el menú de navegación del sitio. El permiso menú de administrar indica que la función del usuario es de confianza suficiente para hacer cambios estructurales en el sitio (por ejemplo, como la adición de un enlace a este contenido en el menú principal, que aparece en cada página). La user\_access () función devuelve un valor booleano, es decir, si VERDADERO, El usuario puede realizar la acción solicitada; si FALSO, El usuario no puede. En el caso de este código de formulario, sólo se mostrará el elemento del formulario, si vuelve la comprobación de acceso VERDADERO. De lo contrario, el valor por defecto del formulario será retenido.

Tenga en cuenta que el ejemplo anterior no pasa una$ Cuenta objeto. Como resultado, la user\_access () incumplimientos de función a la utilización de la corriente $ Usuario objeto, que es, el usuario Actualmente realiza la solicitud de página. La $ Usuario objeto se almacena en una variable global, y por lo que se puede acceder en cualquier momento una específica $ Cuenta no se especifica.

Usted no está obligado a especificar un $ Cuenta al llamar user\_access (), Y en la mayoría de los casos esto está muy bien, pero hay casos de uso donde usted puede ser que desee comprobar el permiso en contra de un usuario que no inició la sesión actual en $ Usuario.

**Comprobación de la cuenta de usuario correcta**

En la mayoría de los casos, las comprobaciones de permisos se hacen contra el usuario actual, que se define en el $ Usuario objeto. Autores de módulo deben prestar especial atención al contexto de su comprobaciones de permisos, especialmente cuando se muestra información sobre los usuarios específicos.

Por ejemplo, es posible que desee agregar una sección a la página de la cuenta de usuario que un administrador del sitio puede comprobar los papeles que un usuario individual tiene. Para ello queremos poner en práctica hook\_user\_view () y probar el mundial $ Usuario objeto de asegurar que se trata de un administrador de confianza, ¿quién puede ver esta información.

En primer lugar, establecer un chequeo sencillo para el usuario actual: ¿Tiene él / ella tiene permiso para ver esta información?

example\_user\_view función ($ cuenta $ view\_mode) {if (! user\_access ('papeles vista de los usuarios')) {

volver;

}

}

Usted verá este patrón frecuente en el código de Drupal. A falta de la comprobación de acceso conduce a un retorno de la función y hace que el código sea más fácil de seguir. Ya que sólo estamos agregando la información a una página existente, devolviendo ningún dato está muy bien. (Más adelante en el capítulo, vamos a ver otras formularios de lidiar con permisos denegados.)

**[213 ]**



*Drupal permisos y seguridad*



Si el usuario actual pasa esta comprobación de acceso, entonces tenemos que ir a buscar la información que queremos. Esta información no es sobre el $ Usuario sino de la $ Cuenta que se está viendo. Así que agregamos la lógica:

/ \*\*

\* Implementar hook\_user\_view (). \* /

example\_user\_view función ($ cuenta $ build\_mode) {if (! user\_access ('papeles vista de los usuarios')) {

volver;

}

/ / Obtiene las funciones del usuario. $ List = $ cuenta-> papeles; if (! empty ($ lista)) {

* Preparar la información para la tematización. variables $ = array (

'items' => $ de lista,

);

$ Content = tema ('item\_list', $ variables);

* Coloque el contenido en la página de usuario de acuerdo con la API. $ Cuenta-> content ['summary'] ['salida'] = array (

'# Type' => 'user\_profile\_item', '# title' => t ('Los roles de usuario'), '# markup' => $ contenido,

'# Atribuye' => array ('class' => array ('de creación de contenidos')),

);

}

}

Cuando se implementa, nuestro código produce el siguiente resultado en una página de usuario:



**[214 ]**



*Capítulo 8*



Si hubiéramos ejecutar accidentalmente la comprobación de permisos en el $ Cuenta objeto, entonces podríamos devolver los permisos incorrectos. Para mayor claridad, vamos a echar un vistazo a un ejemplo más complejo. En el siguiente fragmento, queremos mostrar una lista de todos los tipos de contenido que puede crear un usuario. Nuestra función comenzará al igual que la última aplicación, y luego se vuelven más complejas.

/ \*\*

\* Implementar hook\_user\_view (). \* /

example\_user\_view función ($ cuenta $ build\_mode) {

si {return (user\_access ('permisos de creación de vista de contenido')!);

}

/ / Obtener los tipos de nodos definidos.

$ Node\_types = node\_permissions\_get\_configured\_types (); if (empty ($ node\_types)) {

volver;

}

/ / Hacer un arreglo para la salida de lista. $ List = array ();

foreach ($ node\_types como $ tipo) {

if (user\_access ('crear'. $ tipo. 'contenido', $ cuenta)) {/ / Obtener el nombre legible por humanos del tipo de contenido. $ List [] = check\_plain (node\_type\_get\_name ($ type));

}

}

El fragmento de código anterior define una función que tira de los permisos para la cuenta que se está viendo por el usuario actual. Nuestros dos conjuntos de comprobaciones de permisos operan en diferentes cuentas de usuario.

La pieza importante en este caso es la user\_access () comprobamos que corremos para cada tipo de nodo. Si tuviéramos que dejar fuera de la $ Cuenta, Entonces este control sería asumir que lo que queríamos saber qué tipos de contenido que el usuario actual podría crear. Hacerlo significaría los mismos resultados aparecerían sin importar qué cuenta de usuario página que vimos.



Nota: El uso de la $ Cuenta objeto en lugar del $ Usuario objeto es una práctica estándar de Drupal, y una buena práctica de codificación. En Drupal, el $ Usuario objeto es un valor global, y sería un error pasar (a veces por referencia!) cuando nos referimos únicamente para extraer información

de ella. En su lugar, funciones de búsqueda como hook\_user\_view () actuar siempre en una copia llamada $ Cuenta. Este patrón se repite con frecuencia en el núcleo de Drupal, y usted debe seguir esta práctica.



**[215 ]**



*Drupal permisos y seguridad*



Para terminar este ejemplo, vamos a añadir nuestra función de tema para producir la salida adecuada.

if (! empty ($ lista)) {

* Preparar la información para la tematización. variables $ = array (

'items' => $ de lista,

);

$ Content = tema ('item\_list', $ variables);

* Coloque el contenido en la página de usuario de acuerdo con la API. if (! isset ($ cuenta-> content ['ejemplo'])) {

$ Cuenta-> content ['ejemplo'] = array ();

}

$ Cuenta-> content ['ejemplo'] + = array ('# type' => 'user\_profile\_category',

'# Atribuye' => array ('class' => array ('user-miembro')), '# peso' => 5,

'# Title' => t ('Contenido'),

);

$ Cuenta-> content ['ejemplo'] ['salida'] = array ('# type' => 'user\_profile\_item',

'# Title' => t ('permisos de creación de Content'), '# markup' => $ contenido,

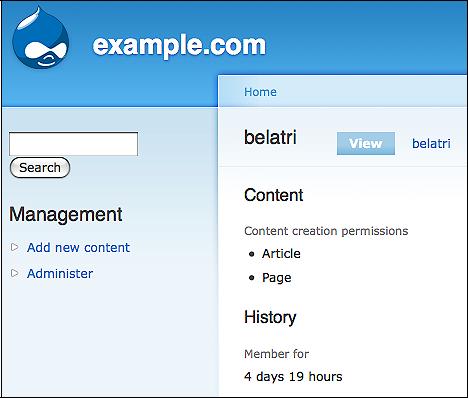
'# Atribuye' => array ('class' => array ('de creación de contenidos')),

);

}

}

Con esta función el tema en su lugar, podemos ver el resultado siguiente:



**[216 ]**



*Capítulo 8*



La user\_access () función es muy eficaz, y que se puede utilizar en casi todos los casos. Ya que sólo toma dos parámetros, puede no ser apropiado para todos los controles de acceso, y nunca se puede comprobar varios permisos a la vez. Más tarde, vamos a ver en los casos de uso donde se necesita una función más compleja para comprobar los permisos.

**Uso de hook\_permission ()**

Ahora que usted entiende los fundamentos de un sistema de acceso de usuario de Drupal, podemos explorar cómo los módulos pueden extender ese sistema. Primero, una pequeña lección de historia.

Hasta Drupal 7, hook\_permission () era conocido como hook\_perm (). El cambio se hizo para una mayor claridad en el código, como parte de una limpieza semántica general del núcleo de Drupal. (Escribí el parche, de hecho.) hook\_permission () También incluye una serie de mejoras de usabilidad, que alteró el formato del valor de retorno de la función. Estos cambios son lo suficientemente sustancial como para incluso los desarrolladores de Drupal experimentados para explorar cada elemento del nuevo hook.

El propósito de hook\_permission () es definir y devolver una matriz que contiene toda la información necesaria acerca de los permisos de su módulo. Esto incluye las cuerdas simples que se pueden pasar a user\_access (), Además de un nombre legible para el permiso y una descripción opcional. Antes de Drupal 7, fue devuelto sólo la cadena simple.

El siguiente es un ejemplo, tomado del módulo principal Buscar:

función search\_permission () {return array (

'Administrar búsqueda' => array ('title' => t ('Administrar búsqueda'),

),

'Buscar contenido' => array ('title' => t ('Usar búsqueda'),

),

'Utilizar la búsqueda avanzada' => array ('title' => t ('Use la búsqueda avanzada'),

),

);

}

El módulo declara tres permisos independientes de una manera típica para módulos de Drupal. El primero es el permiso para la configuración administrativa del módulo.

Este tipo de permisos rara vez se dan a la función "usuario autenticado" y nunca se debe dar a los usuarios anónimos.

**[217 ]**



*Drupal permisos y seguridad*



El segundo permiso concede la capacidad de buscar el contenido del sitio a través del formulario de búsqueda predeterminado. El tercer permiso se extiende el segundo para incluir un formulario adicional para búsquedas avanzadas.



La presencia de estos permisos muy específicos puede parecer extraño, dado que hay una acceder al contenido permiso en el módulo de nodo que otorga a los usuarios la posibilidad de ver el contenido del sitio. Sin embargo, la búsqueda puede ser considerado como un caso especial por algunos sitios. La separación de la contenido de la búsqueda el permiso de la acceder al contenido permiso incluye una capa de flexibilidad que permite la personalización de proyectos.



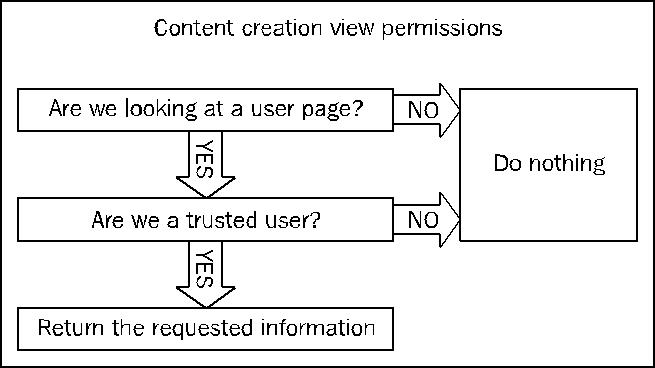
Las personas que se quejan de que Drupal es demasiado complejo deberían considerar este caso por un momento. Desde una perspectiva de constructor de sitios, que tienen tres permisos adicionales significa más trabajo de configuración. Sin embargo, imaginar lo frustrante que sería si usted necesita para desactivar la búsqueda para algunos usuarios, pero no puede (o sólo puede hacerlo mediante programación).

En casos como estos, Drupal casi siempre adopta la flexibilidad. La presencia de varios permisos en un módulo de la base significa que alguien ha necesitado que la separación por una buena razón. (En el próximo capítulo, de hecho, vamos a discutir la razón de laAcceso nodo de derivación permiso, lo cual es una característica nueva en Drupal 7.)

**Definición de permisos de su módulo**

Antes de escribir cualquier código de hook\_permission (), Es la mejor práctica para sacar un lápiz y papel (o un buen programa de diagramación), y hacer un gráfico de las acciones que se pueden esperar los usuarios tomar. De hecho, muchos desarrolladores experimentados escribir este libro al final del desarrollo, después de desconcierto a través de todos los casos de uso en el código.

Vamos a considerar el módulo de ejemplo anterior. Es muy directo. Queremos mostrar un poco de información acerca de los usuarios a los administradores de sitios de confianza. Nuestro caso de uso se ve algo como lo siguiente:



**[218 ]**



*Capítulo 8*



En el primer caso, ¿Estamos viendo una página de usuario? es una pregunta nuestro módulo no necesita preguntar. Sabemos a través de la API que el uso de hook\_user\_view () sólo dispara cuando alguien está buscando a la información de un usuario. Así que no necesitamos nuestra propia comprobación de permisos para esa acción.

Lo que nos importa es la pregunta ¿Estamos un usuario de confianza? Aquí tendremos que crear una comprobación de permisos.

Podríamos simplemente utilizar el existente administrar usuarios permiso proporcionado por el núcleo, pero es que la mejor solución? En muchos casos de uso, es probable que sea. Sin embargo, supongamos que usted está construyendo un sitio web del periódico. En este escenario, tenemos que "editores del sitio", cuyo trabajo consiste en supervisar el trabajo de otros. Para ello, estos editores tienen que ser capaces de revisar los tipos de contenido que puede crear un usuario. Sin embargo, administrar usuarios es un permiso mucho más potente, que permite la edición de las cuentas de usuario y la asignación de roles de usuario. Es posible que no quiera dar ese gran autoridad y responsabilidad a nuestros editores del sitio.

En este caso, de hecho, en la mayoría de los casos-la creación de un permiso discreta para cada acción es la mejor opción. Dado que no se puede predecir de forma fiable todos los escenarios de uso para su código, pilotes demasiadas acciones en un solo permiso puede limitar cómo la gente puede utilizar su módulo. Cada vez que se ve a sí mismo escribiendo unadministrar X permiso en el código, usted debe preguntarse si ese permiso se puede hacer más granular.

**Hook\_permission guión ()**

Para nuestro módulo de ejemplo, entonces, tenemos que definir un permiso que otorga la posibilidad de ver la información acerca de los usuarios. Nuestra gama permiso es bastante directo, y tiene tres partes posibles:

1. El nombre legible por la máquina del permiso. Este elemento se utiliza por el módulo de código para comprobaruser\_access (). Para nuestro ejemplo, utilizamos los "permisos de creación de vista de contenido" de cadena. Por convención, esta cadena debe estar en Inglés y en minúsculas. No tiene que ser una oración completa.
2. La etiqueta legible para el permiso. Este puede ser el mismo que la cadena legible por la máquina, sino que debe ser formateado con un capital inicial. Usted puede capitalizar palabras según sea necesario. A diferencia del nombre legible por la máquina, esta cadena debe ser envuelto en t () de modo que la salida puede ser traducido. Usaremos "Ver las opciones de creación de contenido para usuarios registrados" ya que los usuarios anónimos no tienen una página de la cuenta que desea ver.

**[219 ]**



*Drupal permisos y seguridad*



1. Una descripción opcional del permiso. Esta cadena debe ser una frase completa, y envuelta en t (). Directrices de la interfaz de usuario de Drupal animamos a utilizar este elemento si el permiso necesita aclaración especial, sobre todo si el permiso se le está dando a los usuarios no confiables que puedan constituir un riesgo para la seguridad. Para nuestro ejemplo, nos indicará claramente: "Muestra los tipos de contenido que un usuario puede crear en la página del perfil del usuario."

Cuando ponemos estas tres piezas juntas, nuestro hook parece el siguiente código:

/ \*\*

\* Implementar hook\_permission (). \* /

función example\_permission () {return array (

'permisos de creación de vista de contenido' => array (

'Title' => t ('Ver las opciones de creación de contenidos para los usuarios registrados'),

'Description' => t ('Muestra los tipos de contenido que un usuario puede crear en el usuario \' s página de perfil. '),

),

);

}

Uso de la descripción elemento es tentador, por la totalidad, pero las pruebas de interfaz de usuario de Drupal descubrió que sirve sobre todo un desorden visual para el usuario final.



La siguiente captura de pantalla muestra ejemplos de los tipos de permisos que se benefician de una descripción completa. Esta página se puede encontrar en la siguiente ruta:

admin / personas / permisos



**[220 ]**



*Capítulo 8*



**Declarar sus propias funciones de acceso**

La user\_access () función es una función de gran utilidad, pero no puede cubrir todos los casos de uso lógicas. No puede, por ejemplo, comprobar dos permisos de acceso al mismo tiempo. Usted puede, por supuesto, escribir una sentencia como:

if (user\_access ('permiso único') && user\_access ('permiso de dos')) {/ / realizar alguna acción ...

}

Al asegurar las acciones dentro de su código, este enfoque está perfectamente bien. Sin embargo, los controles de acceso basados ​​en menús no pueden ser objeto de estas normas. En estos casos, tenemos que declarar nuestro propio acceso de devolución de llamada dentro de la menú artículo.

Por ejemplo, vamos a tomar nuestro último ejemplo de módulo. Supongamos que en lugar de mostrar esta información en la página del perfil del usuario, hemos querido hacer una página independiente para obtener esta información. Podríamos hacer una pestaña en la página de usuario, utilizando una devolución de llamada del menú como el siguiente:

/ \*\*

\* Implementar hook\_menu (). \* /

función example\_menu () {$ items ['user /% de usuarios / content'] = array (

'Title' => 'La creación de contenidos',

La página de callback '=>' example\_user\_page ',' Página de argumentos '=> array (1),

los argumentos de acceso '=> array (' permisos de creación de vista de contenido '),' type '=> MENU\_LOCAL\_TASK,

);

return $ artículos;

}

Esto funciona muy bien, a menos que teníamos que dejar un permiso o una condición adicional durante la comprobación de acceso. Supongamos que nuestras reglas de negocio dicen que sólo los editores que también puede administrar usuarios puede ver esta página.

En ese caso, tenemos un problema, ya que el user\_access () función no puede aceptar varios permisos en una sola llamada. En este caso, tenemos que utilizar la acceso devolución de llamada parámetros del menú $ Item:

/ \*\*

\* Implementar hook\_menu (). \* /

función example\_menu () example\_menu () {

**[221 ]**



*Drupal permisos y seguridad*



$ items ['user /% de usuarios / content'] = array ('title' => 'La creación de contenidos',

La página de callback '=>' example\_user\_page ',' Página de argumentos '=> array (1),

"Acceso callback '=>' example\_user\_access ',

los argumentos de acceso '=> array (' permisos de creación de vista de contenido ',' administrar los usuarios '),

'Type' => MENU\_LOCAL\_TASK,

);

return $ artículos;

}

Para que este código tenga éxito, también debemos proporcionar la nueva función example\_user\_access ():

/ \*\*

* Devolución de llamada de acceso que comprueba varios permisos.
* Toma una lista de permisos y requiere que todos regresan
* TRUE.

\* /

función example\_user\_access () {

foreach (func\_get\_args () como $ permiso) {if (! user\_access ($ permiso)) {

return false;

}

}

return true;

}



Nota: En el desarrollo de su módulo, debe reconstruir la caché de menú con el fin de ver los cambios de permisos. Usted puede hacer esto mediante el vaciado de la cache\_menu mesa, o el uso de las herramientas proporcionadas por el módulo Devel o Drush.



Incluso puede dictar una lógica más compleja dentro de una función de control de acceso. Supongamos también queremos evitar que los usuarios puedan ver las páginas que no sean los suyos. Nos gustaría editar las funciones de la siguiente manera:

/ \*\*

\* Implementar hook\_menu (). \* /

función example\_menu () {$ items ['user /% de usuarios / content'] = array (

'Title' => 'La creación de contenidos',

**[222 ]**



*Capítulo 8*



La página de callback '=>' example\_user\_page ',' Página de argumentos '=> array (1),

"Acceso callback '=>' example\_user\_access ',

**los argumentos de acceso '=> array (1,**

**'permisos de creación de vista de contenido', 'administrar los usuarios'),**

'Type' => MENU\_LOCAL\_TASK,

);

return $ artículos;

}

/ \*\*

\* Acceso de devolución de llamada que comprueba varios permisos. \* /

example\_user\_access function () {global $ user;

$ arguments = func\_get\_args (); $ Cuenta = array\_shift ($ argumentos); if ($ cuenta-> UID! = $ user-> uid) {

return false;

}

foreach ($ argumentos como $ permiso) {if (! user\_access ($ permiso)) {

return false;

}

}

return true;

}

En la forma anterior, las reglas de acceso complejas pueden ser aplicadas en las devoluciones de llamada del menú.



Al escribir un cheque de acceso fuera del sistema de menús, es tentador para encadenar una serie de SI comprueba para producir la lógica necesaria.

Considere la posibilidad de mover estas declaraciones en una función de acceso claramente definidos.



Estos pueden mejorar la legibilidad y portabilidad del código. Si quieres puntos de estilo extra, considere agregar un drupal\_alter () función antes de volver VERDADERO o FALSO para permitir a otros módulos para reescribir la lógica estándar.

**[223 ]**



*Drupal permisos y seguridad*



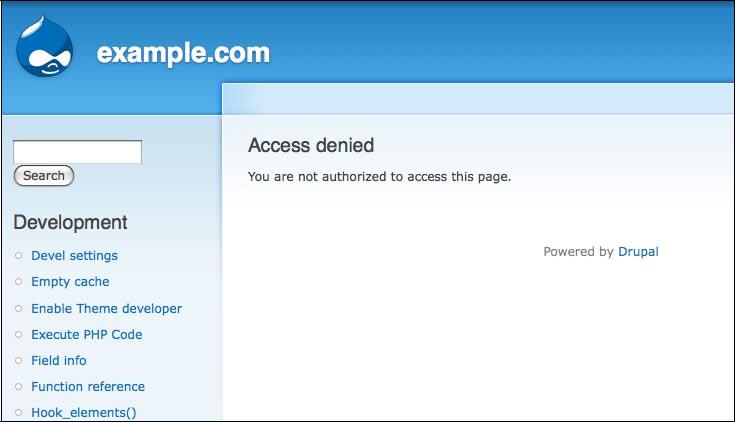
**Responder cuando se niega el acceso**

En un cambio significativo respecto a principios de Drupal, el drupal\_access\_denied () función ya no debería ser llamada cuando se devuelva un contexto normal de la página.

* Funciones de devolución de llamada Página quieran informar de un mensaje de acceso denegado volviesen MENU\_ACCESS\_DENIED en vez de llamar drupal\_access\_denied ()
* Sin embargo, las funciones que se invocan en contextos en los que retorno valor puede no suben a menu\_execute\_active\_handler () deben llamar drupal\_access\_denied (). Para más detalles ver: http://api.drupal.org/api/function/drupal\_access\_denied/7

Sin embargo, ¿qué significa esto en la práctica?

Una ventaja de utilizar una devolución de llamada de menú es que si se deniega el acceso para la solicitud de la página, Drupal maneja automáticamente la respuesta mediante la ejecución de la tradicional drupal\_access\_denied () función.



Sin embargo, el motor de renderizado de Drupal 7 respeta más contextos que en la página web tradicional. Su función de devolución de llamada puede devolver un JSON oponerse, un archivo, o estar respondiendo como una parte de una página más grande (como un formulario). Por ejemplo, considere el siguiente fragmento de código desde contact\_site\_form ():

/ / Comprobar si el control de inundaciones se ha activado para el envío de e-mails. $ Limit = variable\_get ('contact\_threshold\_limit', 5);

$ Window = variable\_get ('contact\_threshold\_window', 3600);

**[224 ]**



*Capítulo 8*



if (! flood\_is\_allowed ('contacto', $ limit, $ window) &&! user\_access ('administrar formularios de contacto')) {

drupal\_set\_message (t ("No se pueden enviar más de% de mensajes de carrera en @ intervalo. Por favor, inténtelo de nuevo más tarde.", array ('% limit' => $ limit, '@ intervalo' => format\_interval ($ window))), ' error ');

drupal\_access\_denied (); drupal\_exit ();

}

El código anterior se detiene todo el procesamiento de la página del formulario de contacto si el usuario es sospechoso de spam de la forma. Si el usuario viola el umbral permitido de mensajes por hora, un mensaje de advertencia se entrega y se les niega el acceso a la página se representa.

Observe el uso dedrupal\_exit () aquí para detener el resto de la ejecución de la página. Dado que se realiza este mensaje de acceso denegado durante la definición de la forma, drupal\_exit () debe ser invocada para dejar el resto del proceso de renderizado.



Nota: No llame a la normal de PHP exit () funcionar desde dentro del código de Drupal. Si lo hace, podrá suspender la ejecución de las funciones internas (como el manejo de sesión) o API hooks. La drupal\_exit () función se proporciona para detener la ejecución de una solicitud de Drupal de forma segura.



Dentro de un contexto de página normal, sin embargo, debemos volver la constante

MENU\_ACCESS\_DENIED en lugar de drupal\_exit (). Podríamos hacer esto en su lugar de la utilización de una devolución de llamada de menú personalizado. Volviendo a nuestro ejemplo anterior:

/ \*\*

\* Implementar hook\_menu (). \* /

función example\_menu () {$ items ['user /% de usuarios / content'] = array (

'Title' => 'La creación de contenidos',

La página de callback '=>' example\_user\_page ',' Página de argumentos '=> array (1),

los argumentos de acceso '=> array (' permisos de creación de vista de contenido '),' type '=> MENU\_LOCAL\_TASK,

);

return $ artículos;

}

/ \*\*

\* Página personalizada de devolución de llamada para una ficha de usuario. \* /

**[225 ]**



*Drupal permisos y seguridad*



función example\_user\_page ($ cuenta) {global $ user;

if (! $ user-> uid = $ cuenta-> uid) {return MENU\_ACCESS\_DENIED;

}

/ / ...

Hay una sutil pero importante diferencia entre los dos enfoques. Si utilizamos una devolución de llamada del menú para afirmar el control de acceso, el vínculo de la ficha sólo se brindará si el

pasa la comprobación de acceso de usuario. Si utilizamos una comprobación de acceso dentro de la página de devolución de llamada, siempre se representará en la ficha. Es mala usabilidad de presentar una ficha que sólo imprime un "acceso denegado" mensaje al usuario. Por esta razón, las comprobaciones de acceso de nivel de página casi siempre debe ser manejado por hook\_menu ().

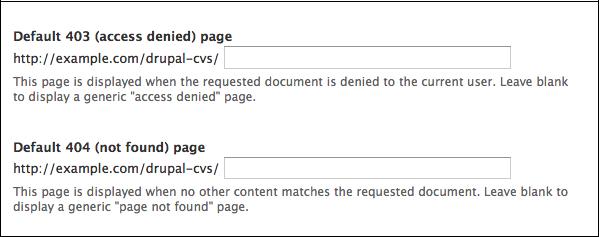


**¿Debo usar drupal\_access\_denied () o una página personalizada?**

drupal\_access\_denied () devuelve una versión de la tradicional Apache **403 acceso denegado** página, servido por Drupal. Buena usabilidad sugiere que la provisión de una página de mensajes de error más amigable ayuda a los usuarios navegar por su sitio con facilidad. Si usted apoya esta idea, no dude en crear una costumbre 403 página. Drupal permite asignar cualquier página de contenido que la página 403 mensajes.



La drupal\_access\_denied () la función devuelve el resultado de esa página, así que no hay necesidad de codificar un 403 mensaje personalizado en su módulo ya que se puede crear y editar a través de la interfaz normal de contenidos Drupal.



La configuración de su página 403 y 404 se encuentran en la configuración de información del sitio.

**[226 ]**



*Capítulo 8*



**Activación de permisos mediante programación**

Funciones de usuario y permisos de Drupal son manejados a través de configuraciones de la interfaz de usuario. Sin embargo, puede haber casos de uso donde el módulo necesita para establecer o modificar los permisos. Hay incluso un módulo llamado Permisos seguros

(http://drupal.org/project/secure\_permissions) Que deshabilita la interfaz de usuario para las funciones de edición y permisos y obliga a todos los ajustes que se definirán en código.

Si su módulo necesita para definir los permisos en el código, Drupal 7 ofrece algunos nuevos hooks para hacer la tarea más fácil. Tomemos un ejemplo común. Su módulo crea una devolución de llamada página que debe ser visible por los usuarios "anónimos" autenticado "pero no. Para activar esta función cuando el módulo está habilitado, puede utilizarhook\_enable () de la siguiente manera:

función example\_enable () {

$ permissions = array ('ejemplo de la página vista'); user\_role\_change\_permissions (DRUPAL\_AUTHENTICATED\_USER,

$ permisos);

}

Esta función entra en sus del módulo . Instalar presentar. Cuando el módulo está activado, Drupal añadir el ver ejemplo página permiso para la función de usuario autenticado.

Usted puede (y normalmente debe) hacerlo de la otra cuando el módulo está desactivado:

función example\_disable () {

$ permissions = array ('ejemplo de la página vista'); papeles $ = user\_roles ();

* Dado que se pueden establecer permisos por el papel, retire nuestro permiso de
* cada rol.

foreach ($ papeles como $ deshacerse => $ nombre) {$ (user\_role\_revoke\_permissions deshacerse, $ permisos);

}

}

También es posible añadir / eliminar varios permisos a la vez. Para ello, tenemos que construir un conjunto de permisos que se pasa al user\_role\_change\_ permisos (). Supongamos que nuestro módulo quiere eliminar el valor por defecto acceder al contenido el permiso de la función de usuario anónimo, mientras que la adición de nuestro nuevo ver ejemplo página permiso. Para ello, construimos una matriz en el formato 'Nombre de permiso' => TRUE o FALSO, Para cada rol.

función example\_enable () {$ permissions = array (

'Acceder al contenido' => ', ejemplo de la página vista' FALSE => TRUE,

);

user\_role\_change\_permissions (DRUPAL\_ANONYMOUS\_USER, $ permisos);

}

**[227 ]**



*Drupal permisos y seguridad*



Cuando nuestro módulo está habilitado, los ajustes para estos dos permisos se pueden cambiar para el usuario anónimo.



La user\_role\_change\_permissions () función se utiliza realmente en el formulario de presentar controlador para el formulario de permisos. Al abstraer esta lógica a una función, Drupal ofrece una decisión fácil API para otros módulos. Cuando la construcción de los módulos, usted debe buscar oportunidades similares para que otros desarrolladores pueden construir fuera de su código en lugar de volver a implementar una lógica similar.



**Definir las funciones mediante programación**

Al igual que con los permisos, Drupal 7 permite a los roles que se establezcan a través de una simple llamada de función. El nuevo user\_role\_save () y user\_role\_delete () funciones proporcionan las herramientas a sus necesidades de módulos.

La user\_role\_save () función solamente entraña una nueva función llamada a la {papeles} mesa y le asigna un identificador de papel apropiado ($ Deshacerse). La user\_role\_delete () función elimina ese papel de la {papeles} mesa, y también realiza la limpieza de los permisos asociados almacenados en la {Role\_permission} de mesa y cualquier usuario asignaciones de funciones almacenados en

la {Users\_roles} tabla.

Digamos que su módulo permite a los usuarios a moderada otras cuentas de usuario. Se trata de una potente capacidad en un sitio, por lo que su módulo crea automáticamente un nuevo papel que contiene los permisos adecuados.

Como en nuestro ejemplo anterior, vamos a utilizar hook\_enable () para crear el nuevo rol.

/ \*\*

\* Crear una función para la gestión de cuentas de usuario. \* /

función account\_moderator\_enable () {

/ / Crear el papel 'cuenta moderador. user\_role\_save ("cuenta de moderador ');

}

Después de crear la función, también podemos asignar automáticamente una serie de permisos:

/ \*\*

\* Crear una función para la gestión de cuentas de usuario. \* /

función account\_moderator\_enable () {

/ / Crear el papel 'cuenta moderador. user\_role\_save ("cuenta de moderador ');

**[228 ]**



*Capítulo 8*



$ permissions = array ('perfiles de acceso de usuario', 'administrar los usuarios',

);

$ Role = user\_role\_load\_by\_name ('moderador acount'); user\_role\_grant\_permissions ($ role-> deshacen, $ permisos);

}

Cuando se desinstala el módulo, hay que eliminar el papel también.

función account\_moderator\_uninstall () {user\_role\_delete ("cuenta de moderador ');

}

**Asegurar formularios en Drupal**

Manejo de formularios es una de las áreas más importantes de seguridad del sitio web. El manejo inadecuado de los datos del formulario puede conducir a múltiples fallos de seguridad que incluyen la inyección de SQL y cross-site solicitud falsificación (CSRF). Si bien no podemos cubrir todos los aspectos de la seguridad en un breve capítulo, es importante señalar algunas pautas claras para los desarrolladores de módulos de Drupal.



Ver http://en.wikipedia.org/wiki/CSRF para obtener información sobre RFCE, y para cross-site scripting (XSS) ver http://en.wikipedia.

org / wiki / XSS.



**La API de las Formularios**

En primer lugar, usted debe utilizar siempre el Drupal Forms API en la creación y procesamiento de formularios en Drupal. Por un lado, el hacerlo le hace la vida más fácil porque el

Formularios API contiene normas para la definición del formulario, AJAX manipulación, elementos requeridos, el manejo de validación, y presentar manipulación. (Vea más sobre las formularios en el Capítulo 5.)

Desde un punto de vista de la seguridad, la API de formularios es fundamental, ya que contiene mecanismos integrados para la prevención de las solicitudes CSRF.

Siempre que Drupal se crea un formulario a través de la API, la forma está etiquetado con un token único llamado el form\_build\_id. La form\_build\_id es un hash MD5 aleatoria usada para identificar el formulario durante el procesamiento. Este token se añade por el drupal\_build\_form () rutina:

$ Form\_build\_id = 'forma'. drupal\_hash\_base64 (uniqid (mt\_rand (), TRUE)

. mt\_rand ());

$ ['# Build\_id'] = $ form\_build\_id forma;

**[229 ]**



*Drupal permisos y seguridad*



La forma está etiquetado adicionalmente con un $ Form ['# simbólico'] elemento durante drupal\_process\_form (). La # Simbólico se utiliza para garantizar que se incluye una solicitud de formulario a partir de una petición conocida (es decir, una petición HTTP que se ha emitido una sesión válida para el sitio). La # Simbólico valor se establece con drupal\_get\_token ():

función drupal\_get\_token ($ value ='') {

volver drupal\_hmac\_base64 (.. $ value, session\_id () drupal\_get\_private\_key () drupal\_get\_hash\_salt ());

}

Cuando Drupal procesa una forma, tanto el $ Form\_build\_id y $ Form ['# simbólico'] valores se validan para asegurar que la solicitud forma se originó en el sitio de Drupal.



También debemos señalar que Drupal forma predeterminada a usar el POSTE método. Si bien es posible que presenten sus formularios de Drupal a través de OBTENER, Los desarrolladores siempre les anima a utilizar POSTE, Que es más seguro. Veremos asegurar OBTENER peticiones cuando discutimos AJAX manejar un poco más adelante en este capítulo.



**Deshabilitar elementos de formulario**

Además de la seguridad global de una forma específica, es posible que también desee activar o desactivar determinadas partes de un formulario, ya sea en forma de su propio módulo o el proporcionado por Drupal core (u otro módulo contribuido). En el primer ejemplo de este capítulo, hemos visto cómo esto se puede hacer mediante eluser\_access () función (o una función de control de acceso similar) para marcar un elemento de forma individual o sección entera de una forma

como inaccesible.

$ ['Menú'] form = array ('# type' => 'fieldset',

'# Title' => t ('Configuración' del menú),

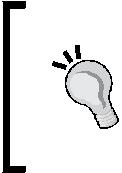
'# Access' => user\_access ('administrar menú'), '# plegable' => TRUE,

'# Derrumbó' =>! $ Enlace ['link\_title'],

);

Cuando se representa la forma de edición de contenido, los usuarios sin el menú de administrar permiso no podrá ver este elemento del formulario.

Tenga en cuenta que '# Access' => FALSE no es el mismo que '# Disabled' => FALSE Formularios API de Drupal. Uso # Disabled => FALSE hará que el elemento de formulario y desactivar la entrada de datos a ese elemento, mientras '# Access' => FALSE quita el elemento completo de la salida.



**[230 ]**



*Capítulo 8*



Este enfoque es la manera adecuada de eliminar elementos de formulario de Drupal. Usted puede encontrar que la tentación de unset () ciertos elementos de formulario, pero ya que las formularios de Drupal se pasan por referencia a través de una serie de drupal\_alter () hooks, la unset () no puede considerarse fiable. Uso unset () También elimina contexto valioso que otros módulos pueden confiando en la tramitación del $ Form.

**Pasar datos seguras a través de formularios**

Como regla general, las formularios de Drupal no utilizan el tradicional oculto elemento form de HTML. Desde oculto elementos de formulario se prestan en el navegador, los usuarios curiosos

(Y los 'malos') pueden ver los elementos de un formulario, la comprobación de los tokens y otros dispositivos de seguridad.

Desde Drupal es una aplicación PHP, puede utilizar los procesos del lado del servidor para manejar elementos de forma secreta, en lugar de confiar en la información que se pasa como campos ocultos del navegador.

Para pasar esos datos, un elemento de formulario puede definirse como '# Type' => 'valor'. El uso de este elemento Formularios API impide que los datos se representa en el explorador. Como ventaja adicional, también permite el paso de datos complejos, tales como una matriz-durante una petición de formulario. Esta técnica se utiliza comúnmente para elementos de forma que el usuario no debe ver como el id de un elemento para borrar en un paso de confirmación. Considere el siguiente código deaggregator.module:

función aggregator\_admin\_remove\_feed ($ form, $ form\_state, $ feed) {return confirm\_form (

array (

'Feed' => array ('type #' => 'valor', '# value' => $ de alimentación,

),

),

t ('¿Está seguro que desea eliminar todos los elementos del feed feed%?', array ('% feed' => $ feed-> title)),

'Admin / config / services / agregador', t ('Esta acción no se puede deshacer. "), T (' Eliminar elementos '),

t ("Cancelar")

);

}

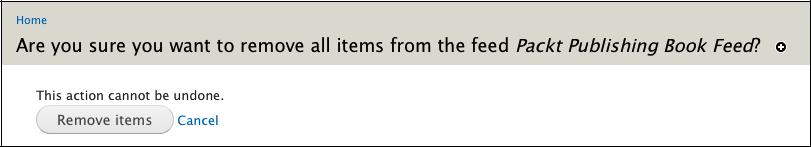
**[231 ]**



*Drupal permisos y seguridad*



El formato que se presenta al usuario final no contiene ninguna información sobre el elemento que desea eliminar. Esos datos se pasa detrás de las escenas.



La forma, como se muestra en el navegador, sólo contiene los datos que Drupal tiene que validar el formulario y extraer los datos de su memoria caché:

<Form action = "/ drupal-cvs/admin/config/services/aggregator/remove/1" accept-charset = método "UTF-8" = "post" id = "agregador-admin-remove-feed" class = " confirmación ">

<div>

Esta acción no se puede deshacer.

<input type="hidden" name="confirm" id="edit-confirm" value="1" /> <div class="container-inline">

<input type="submit" name="op" id="edit-submit" value="Remove items" class="form-submit" />

<a href="/drupal-cvs/admin/config/services/aggregator"> Cancelar </ a>

</ Div>

<input type="hidden" name="form\_build\_id" id="form-049070cff46eabd3b069f980066b7ad4" value="form-049070cff46eabd3b069f980066b7ad4" />

<input type="hidden" name="form\_token" id="edit-aggregator-admin-remove-feed-form-token" value="48b0294050ef62b7d55778cf1992f326" />

<input type="hidden" name="form\_id" id="edit-aggregator-admin-remove-feed" value="aggregator\_admin\_remove\_feed" />

</ Div>

</ Form>

El manejador de presentar el formulario recoge el valor de los datos para su procesamiento:

/ \*\*

* Retire todos los elementos de una fuente y redirigir a la página de información general.
* @ Param $ feed
* Una matriz asociativa que describe el alimento que desea borrar.

\* /

función aggregator\_admin\_remove\_feed\_submit ($ form, & $ form\_state) {aggregator\_remove ($ form\_state ['valores'] ['feed']); $ Form\_state ['redirect'] = 'admin / config / services / agregador';

}

**[232 ]**



*Capítulo 8*



**Ejecución de comprobaciones de acceso en los formularios**

Si bien es perfectamente posible ejecutar comprobaciones de acceso en la construcción de una forma, los desarrolladores, que normalmente no se ejecutan las comprobaciones de acceso en el tratamiento de una forma de\_Validate () o \_submit () devoluciones de llamada. Si lo hace, interfiere con la lógica de la hook\_form\_alter (). Por ejemplo, si su módulo desea alterar la forma elemento de menú de arriba, de modo que otros usuarios pueden añadir elementos de contenido al menú sin poder editar el menú completo, puede hacerlo fácilmente:

example\_form\_alter función (& $ form, $ form\_state, $ form\_id) {

if (empty ($ form! ['# node\_edit\_form']) && isset ($ form ['menú'])) {$ form ['menú'] ['# access'] = example\_user\_access (

'Asignar contenido a la oferta');

}

}

Este código cambia la devolución de llamada de acceso de la menú formar parte de nuestra propia función. Desde hook\_form\_alter () carreras después de un formulario está construido en un principio, podemos modificar cualquier elemento de formulario de esta manera.

Sin embargo, formar \_Validate () y \_submit () devoluciones de llamada no se ejecutan a través de cualquiera de las funciones alternas. Esto significa que siempre se imponen ningún control de acceso que se ejecutan durante esas devoluciones de llamada. Tomemos, por ejemplo, el siguiente ejemplo del núcleo de Drupal node.module, Que hace que sea imposible para los usuarios normales para cambiar el autor de una nodo o el momento de su presentación:

/ \*\*

\* Llevar a cabo comprobaciones de validación en el nodo dado. \* /

función node\_validate ($ node, $ form = array ()) {$ tipo = node\_type\_get\_type ($ node);

if (isset ($ node-> nid) && (node\_last\_changed ($ node-> nid)> $ node-> cambiar)) {

form\_set\_error ('cambiado', t ('El contenido de esta página o bien ha sido modificado por otro usuario, o si ya se han presentado modificaciones mediante este formulario Como resultado, los cambios no se pueden guardar..'));

}

**if (user\_access ('administrar nodos')) {/ / Validar el "escrito por" el campo.**

**if (! empty ($ node-> nombre) &&! ($ cuenta = user\_load\_by\_name ($ node-> nombre))) {**

**/ / El uso de empty () es obligatoria en el contexto de los nombres de usuario**

**[233 ]**



*Drupal permisos y seguridad*



* **como la cadena vacía denota el usuario anónimo. En caso de que**
* **se trata de un usuario anónimo nos fijamos el ID de usuario en 0. form\_set\_error ('nombre', t ('El nombre de usuario% no existe.',**

**array ('name%' => $ node-> nombre)));**

**}**

**/ / Validar el "autor de" campo.**

**if (empty ($ node-> fecha) && strtotime ($ fecha nodo->) === FALSO!) {form\_set\_error ('date', t ('Debe especificar una fecha válida.'));**

**}**

**}**

/ / Hacer controles de validación específicos del tipo de nodo. node\_invoke ($ node, 'validate', $ form); module\_invoke\_all ('node\_validate', $ node, $ form);

}

La inclusión de este control de acceso puede añadir un nivel de prevención en el error de que los usuarios que no pueden "administrar nodos 'no pueden alterar el autor sin permisos especiales, pero no se hace Drupal más seguro. Esto se debe a la seguridad para este elemento de formulario ya está establecido en el $ Form definición, por lo que su uso aquí es redundante:

Información sobre el autor / / Nodo para administradores form ['autor'] = array $ (

'Type #' => 'fieldset',

**'# Access' => user\_access ('administrar nodos'),**

'# Title' => t ('Autoría información'), '# plegable' => TRUE,

'# Derrumbó' => TRUE,

'# Grupo' => 'additional\_settings', '# adjunto' => array (

'Js' => array (drupal\_get\_path ("módulo", "nodo"). '/ Node.js'),

),

'# Peso' => 90,

);

En cambio, la colocación de una comprobación de acceso en el controlador de validar fuerza un módulo de autor para evitar el código mediante la sustitución del núcleo node\_validate () y node\_submit () devoluciones de llamada, que pueden introducir errores adicionales o agujeros de seguridad en el código.

Por esta razón, los autores del módulo se recomienda en absoluto la ejecución de los controles de acceso durante el procesamiento de formularios.

**[234 ]**



*Capítulo 8*



**Manejo de las devoluciones de llamada de AJAX de forma segura**

Drupal 7 viene con un marco de AJAX mejorada que hace que sea fácil de construir los elementos de visualización interactivas para páginas y formularios. El problema de seguridad para Drupal es que las devoluciones de llamada de AJAX toman la forma de devoluciones de llamada del menú, que a diferencia de la mayoría de las formularios de Drupal, son esencialmenteOBTENER peticiones al navegador. Este hecho significa que cualquier solicitud de una devolución de llamada AJAX debe ser tratada como malicioso y que todas las solicitudes deben hacerse la prueba de validez antes de que se envíe una respuesta AJAX.

**El uso de AJAX en formularios**

Cuando se utiliza el # Ajax elemento con el API Forms, Drupal asegura automáticamente la devolución de llamada AJAX mediante la comprobación de la validez de la solicitud de formulario. Esta acción sólo funciona, por supuesto, si usted sigue el FormsAPI correctamente. Uso de la # Ajax elemento de forma activa la ajax\_get\_form () función, que utiliza form\_build\_id para la prueba de validez:

función ajax\_get\_form () {

$ Form\_state = form\_state\_defaults (); $ Form\_build\_id = $ \_POST ['form\_build\_id'];

/ / Obtener el formulario de la caché.

$ Form = form\_get\_cache ($ form\_build\_id, $ form\_state); if (! $ form) {

* Si $ form no se puede cargar desde la memoria caché, el form\_build\_id
* en $ \_POST debe ser válido, lo que significa que alguien
* realizado una petición POST a sistema / ajax sin llegar
* viendo la forma de que se trate en el navegador.
* Esto es probablemente un intento de hacking que nunca sucede bajo
* circunstancias normales, así que simplemente no hacen nada. watchdog ('ajax', 'los datos de forma no válida POST.', array (),

WATCHDOG\_WARNING); drupal\_exit ();

}

/ / ...

Como vimos en la sección anterior que form\_build\_id aseguró que la solicitud de formulario emitido por el sitio de Drupal y era válido.

**[235 ]**



*Drupal permisos y seguridad*



**El uso de AJAX en otros contextos**

Si bien el manejo forma de AJAX proporciona un API ordenado y un control de seguridad, no somos tan afortunados cuando se utilizan otras devoluciones de llamada de AJAX. Para citar a Greg Knaddison, miembro del equipo de seguridad de Drupal y autor de Cracking Drupal, la obra definitiva sobre la seguridad de Drupal:

*[I] t es tentador cuando se construye una amplia gama de características de AJAX para volver a sumergirse en la creación de una vulnerabilidad CSRF mediante peticiones GET .... Sin embargo, debido a esta práctica de la adopción de medidas en respuesta a las peticiones GET no es tan común o estándar que el sistema de la forma, no hay manera de proporcionar esta protección de forma automática o fácilmente.*

*Cracking Drupal, pg 18.*

Para entender el punto, vamos a ver un caso típico menú AJAX utilización de devolución de llamada. Supongamos que queremos un módulo que permite a los usuarios añadir o eliminar elementos de una lista a través de una devolución de llamada AJAX dinámico. El módulo puede configurar algo como lo siguiente:

función example\_menu () {$ items = array ();

$ items ['elemento de ejemplo-ajax /% / add'] = array ('title' => 'Ejemplo AJAX añadir a la lista', 'Página callback' => 'example\_ajax\_add', los argumentos de las páginas '=> array (1) ,

los argumentos de acceso '=> array (' añadir a mi lista '),' type '=> MENU\_CALLBACK,

);

return $ artículos;

}

example\_ajax\_add función ($ item) {/ / hacer algo.

}

Mirando el código anterior, hay varios aspectos que serán evidentes de inmediato:

* La devolución de llamada de acceso predeterminado user\_access () es probablemente insuficiente, ya que estamos manejando una lista de cada usuario
* El permisoañadir a mi lista proporciona ningún medio para comprobar si el usuario es el propietario de la lista que se está editando
* Simplemente tratando de ocultar el elemento de menú de la navegación del sitio (a través del uso de la MENU\_CALLBACK propiedad) no impedirá que otros usuarios (o incluso los rastreadores de motores de búsqueda) de la larga búsqueda de la página

**[236 ]**



*Capítulo 8*



Como resultado de ello, no podemos confiar en la devolución de llamada del menú para despedir a cualquier acción en example\_ajax\_add ()sin la adición de algunos controles de seguridad adicionales.

En primer lugar, sabemos que tenemos que comprobar que el usuario que realiza la acción. De nuestra discusión anterior, este es el más entregado a través de una devolución de llamada de acceso, por lo que editamos nuestra declaración:

"Acceso callback '=>' example\_access\_ajax\_add ',

Para ejecutar esta comprobación con éxito, también es necesario conocer la $ Usuario cuya lista se está actualizando:

$ items ['elemento de ejemplo-ajax /% / add /% user'] = array (

También tenemos que pasar la $ Usuario en nuestro callback acceso:

los argumentos de acceso '=> array (3),

Así que nuestro hook reescrito parece el siguiente código:

función example\_menu () {$ items = array ();

$ items ['elemento de ejemplo-ajax /% / add /% user'] = array ('title' => 'Ejemplo AJAX añadir a la lista',

La página de callback '=>' example\_ajax\_add ',' Página de argumentos '=> array (1, 3),

'Callback acceso' => 'example\_access\_ajax\_add', 'argumentos de acceso' => array (3),

'Type' => MENU\_CALLBACK,

);

return $ artículos;

}

En nuestra devolución de llamada de acceso, ahora podemos comprobar que el enlace hace referencia al usuario actual. Así que nuestro código HTML se verá algo como lo siguiente:

<a href="/example-ajax/3/add/10"> Añadir a mi lista </ ​​a>

El código para generar el enlace correría a través de Drupal L () función:

if ($ usuario-> UID> 0) {

$ Salida = l (... T ('Agregar a mi lista ")," ejemplo-ajax /' $ item-> id '/ add /' $ user-> uid);

return $ resultados;

}

**[237 ]**



*Drupal permisos y seguridad*



En nuestra devolución de llamada, el sistema de menús se transforma 10 en una norma $ Usuario objeto, que comprobamos para la validez de dos maneras:

función example\_access\_ajax\_add ($ cuenta) {global $ user;

if ($ cuenta-> UID | | $ cuenta-> uid = $ usuario-> UID!) {return false;

}

return true;

}

En primer lugar, si user\_load () retornos FALSO, Entonces el argumento de la página no es válida. En segundo lugar, si el vuelto $ Cuenta no coincide con el usuario que realiza la petición, la solicitud no es válida.

Esto es bastante bueno. Permite que nuestro código para comprobar que el usuario que realiza la petición AJAX es el usuario actualmente conectado. Sin embargo, ¿cómo sabemos que esta petición vino de nuestro servidor y no es un ataque CSRF?

Bueno, sinceramente, no lo sabemos y no podemos saber. Sin embargo, podemos ser un poco paranoico y añadir otra capa de seguridad.

Knaddson nos da la clave para Cracking Drupal, cuando dice:

*El equipo de seguridad está trabajando en una API para hacer [asegurar AJAX callback] mucho más fácil, pero eso API todavía no está disponible ... El sistema está basado en el mismo sistema de fichas se utiliza para proteger las formularios de Drupal.*

*Cracking Drupal, página 18*

Para implementar esta estructura, hay que añadir un argumento adicional a nuestra página de devolución de llamada:

función example\_menu () {$ items = array ();

$ items ['elemento de ejemplo-ajax /% / add /% de usuarios /%'] = array ('title' => 'Ejemplo AJAX añadir a la lista',

La página de callback '=>' example\_ajax\_add ',' Página de argumentos '=> array (1, 3),

"Acceso callback '=>' example\_access\_ajax\_add ',

**los argumentos de acceso '=> array (3, 4),**

'Type' => MENU\_CALLBACK,

);

return $ artículos;

}

**[238 ]**



*Capítulo 8*



Esto nos permite pasar un token de autenticación Drupal en nuestro devolución de llamada gratuita. Para que esto funcione, modificamos nuestro código de creación de enlace para incluir un token:

if ($ usuario-> UID> 0) {

= $ Salida l (t ('Agregar a mi lista'), 'ejemplo ajax /'. $ Item-> id. '/ Añadir /'. $ User-> uid). '/'. drupal\_get\_token ($ user-> uid);

return $ resultados;

}

Esto generará un enlace similar a:

<a href="/example-ajax/3/add/10/c4d312412df415ca0"> Añadir a mi lista </ ​​a>

Luego, en nuestra devolución de llamada de acceso, vamos a consultar la ficha en adición al usuario:

función example\_access\_ajax\_add ($ cuenta $ token = NULL) {global $ user;

/ / Comprobar la validez de la cuenta de usuario.

if ($ cuenta-> el uid == 0 | | $ cuenta-> uid = $ usuario-> UID!) {return false;

}

/ / Comprobar la validez del token de devolución de llamada.

if (! empty ($ token) | | drupal\_valid\_token ($ modo, $ cuenta-> uid)) {return false;

}

return true;

}

Símbolo API manejo de Drupal realiza la validación para nosotros, y se garantiza la misma protección que se da a las formularios regulares de Drupal.



Tenga en cuenta que este método sólo funciona correctamente para los usuarios registrados en quienes se atiende páginas no almacenados en caché. La salida de enlace que para acceder a esta devolución de llamada no puede ser almacenado en caché, ya que el almacenamiento en caché devuelve la misma salida HTML a todos los usuarios.



Como regla general, usted sólo tendrá que preocuparse acerca de la manipulación de token para devoluciones de llamada de AJAX que realizan acciones creativas o destructivas, tales como la edición de una lista de favoritos del usuario. Esto se debe a este tipo de acciones por lo general escriben a la base de datos, y pueden cambiar ciertos ajustes para los usuarios de Drupal. Devoluciones de llamada de AJAX simples que sólo leen y devuelven datos no necesariamente tienen que ser asegurado de esta manera a menos que los datos son específicos del usuario.

**[239 ]**



*Drupal permisos y seguridad*



**Resumen**

Nuestra cobertura del sistema de permiso de Drupal debe darle toda la información que necesita para configurar correctamente las reglas de acceso para su módulo.

En este capítulo, hemos aprendido los conceptos básicos de la autorización y el sistema de papel en Drupal. También hemos visto cómo utilizar user\_access () hacer valer los permisos. Hemos hablado de cómo hook\_menu () maneja el control de acceso, y también cómo utilizar

hook\_permission ().

Hemos visto la importancia de las definiciones de permisos granulares, y cuándo utilizar una función que no seauser\_access () para afirmar el control de permisos. Hablamos de cómo escribir una función de control de acceso personalizado, y cómo responder cuando se le niega el acceso. También vimos cómo asignar y quitar permisos mediante hook\_enable () y

hook\_disable ().

Aprendimos a manejar papeles programación junto con los fundamentos de asegurar formularios de Drupal. Por último nos fijamos en cómo manejar con seguridad las devoluciones de llamada de AJAX.

**[240 ]**



Nodo de Acceso

Out-of-the-box, Drupal es un gran sistema para la creación y gestión de contenidos. Los usuarios pueden acceder y crear contenidos. El uso adecuado de los roles y permisos permite a los editores del sitio para revisar una parte o todo el contenido de un sitio. Los visitantes del sitio pueden leer los mensajes publicados.

¿Pero qué sucede si desea que todos los visitantes del sitio para ver algunos contenidos, pero sólo a usuarios registrados para ver una lista de selección de contenido restringido? Si, por ejemplo, el sitio requiere pagó la inscripción para acceder a artículos de fondo sobre la forma de construir sitios web de Drupal, los permisos básicos provistos por Drupal no son suficientes.

Hay casos en que es necesario establecer normas más avanzadas en relación con cuál de los usuarios (o grupos de usuarios) puede crear, ver, editar y eliminar contenido. Para habilitar estas reglas, Drupal proporciona un sistema de nodo de acceso. Nodo de Acceso proporciona una API para la determinación de las subvenciones o los permisos que un usuario tiene para cada nodo. Al entender cómo funcionan estas subvenciones, un desarrollador de módulos puede crear y hacer cumplir las reglas de acceso complejas.

En Drupal 7, cualquier módulo puede declarar las reglas de acceso de nodo. Se trata de un cambio de las versiones anteriores, y proporciona algunas de las herramientas más poderosas para el desarrollo de Drupal.

En este capítulo, vamos a cubrir:

* Nodo de acceso en comparación con user\_access () y otras comprobaciones de permisos
* Cómo Drupal concede permisos de nodos
* La node\_access () función
* hook\_node\_access () en comparación con {Node\_access}
* El control de permisos para crear contenido
* Uso hook\_node\_access ()
* Al escribir un módulo de nodo de acceso

*Nodo de Acceso*



* La {Node\_access} mesa y su papel
* Definir reglas de acceso de su módulo
* Uso hook\_node\_access\_records ()
* Uso hook\_node\_grants ()
* La reconstrucción de la {Node\_access} tabla
* Modificar el comportamiento de otros módulos
* Uso hook\_node\_access\_records\_alter ()
* Uso hook\_node\_grants\_alter ()
* Probar y depurar su módulo
* Usando Desarrollo Nodo de Acceso

**Nodo de acceso en comparación con user\_access () y otras comprobaciones de permisos**

Desemejante user\_access (), Mediante el sistema de acceso de nodo no es un simple caso de la implementación de un control de permiso específico antes de ejecutar el código.

Como vimos en el capítulo anterior, user\_access () determina qué código puede ser ejecutado por un usuario específico y bajo una serie de condiciones dadas. Sistema de nodo de acceso de Drupal es similar, pero debe tener en cuenta las condiciones variables dentro de un determinado conjunto de nodos. Por ejemplo, ciertos usuarios se les puede permitir corregir cualquier contenido básico página pero no se les permite corregir cualquier contenido del artículo. Esta condición significa que Drupal debe ser capaz de distinguir entre las diferentes propiedades de cada nodo.

La API de nodo de acceso es una serie de funciones relacionadas entre sí que proporcionan una interfaz de programación coherente para hacer este tipo de comprobaciones de acceso. Debido a la naturaleza flexible de Drupal sin embargo, hay varias maneras de definir y poner en práctica mecanismos de control de acceso del nodo.

**Cómo Drupal concede permisos de nodos**

Como mencionamos en la introducción, hay cuatro operaciones fundamentales que afectan a los nodos: Crear, Ver, Actualizar y Eliminar. Colectivamente, éstos se refieren como CRUD (Ver donde se sustituye por Lee). Cuando se trata de los nodos, es vital saber que se está realizando la operación.

**[242 ]**



*Capítulo 9*



La API de nodo de acceso permite a los módulos para alterar el comportamiento del flujo de trabajo por defecto de Drupal CRUD. Normalmente, los nodos de Drupal son creados por un solo usuario. Ese usuario

"Dueño" del nodo y, en la mayoría de los casos, puede editar o borrar el nodo a voluntad. Algunos usuarios, como el usuario administrativo 1, puede editar cualquier nodo. Pero por defecto, Drupal no tiene concepto de la propiedad del grupo de nodos. Ciertas funciones se pueden dar permiso para editar todos los nodos de un tipo (como se muestra por el núcleocorregir cualquier contenido del artículo permiso, por ejemplo), pero fuera de la caja no hay ninguna disposición para restringir el acceso para ver este contenido.

La API de nodo de acceso se desarrolló fuera de la necesidad de definir un conjunto flexible y extensible de reglas de acceso. Mucho ha mejorado en Drupal 7, los desarrolladores tan experimentados querrán revisar este material con cuidado.

Permisos nodo de acceso se verifican en dos casos:

* Cuando se hacen peticiones para actuar sobre un nodo individual.
* Cuando las consultas de bases de datos devuelven listas de nodos que coinciden con las condiciones dadas.

Con el fin de manejar el acceso del nodo de forma segura, los desarrolladores de módulos deben ser conscientes de los dos casos.

El primer caso es bastante simple, y es manejado generalmente por una devolución de llamada del menú y el node\_access () función. A menos que su módulo tiene la intención de interferir con la normal manejo de node\_menu (), Usted puede ser capaz de saltarse el resto de este capítulo.

Sin embargo, todos los desarrolladores de módulos necesitan entender el impacto de los dos casos. Destaquemos aquí.



Cualquier consulta de base de datos que implica la {Node} tabla debe ser construido de forma dinámica y se marcará como una consulta de nodo de acceso. De no hacerlo, puede introducir vulnerabilidades de seguridad en los sitios que ejecutan su código.



Para entender esta regla, echemos un vistazo a un ejemplo sencillo del núcleo de Drupal. La siguiente consulta se encuentra en node\_page\_default (), La función que proporciona la página básica anuncio nodo:

$ Select = db\_select ('nodo', 'n') -> campos ('n', array ('nid')) -> condición ("promover", 1) -> condición ("status", 1) -> orderBy ('pegajosa', 'DESC') -> orderBy ('creado', 'DESC') -> extend ('PagerDefault')

**[243 ]**



*Nodo de Acceso*



-> Límite (variable\_get ('default\_nodes\_main', 10)) -> addtag ​​('node\_access');

$ Nids = $ select-> execute () -> fetchCol ();

Esta instrucción de selección utiliza generador de consultas de Drupal 7 para ir a buscar una lista de nodos publicados que han sido promovidos a la primera página, ordenados por "adherencia" y la edad. Nótese, sin embargo, el elemento final de la consulta: -> Addtag ​​('node\_access').

Esta directiva invoca elnode\_query\_node\_access\_alter () función que permite a las reglas de acceso de nodo a ser aplicados antes de que la consulta se envía a la base de datos. Si no se utiliza el generador de consultas dinámicas y la node\_access etiqueta significará que su sentencia select pasará por alto las características de seguridad integradas de Drupal. Si lo hace, podrá conceder el acceso no deseado para ver, editar o eliminar el contenido al ignorar los permisos definidos para el sitio.

No vamos a entrar en el funcionamiento interno de node\_query\_node\_access\_alter () todavía. En pocas palabras, se asegura de que cualquier consulta a la {Node} tabla exige correctamente las reglas de acceso de nodos definidos para el sitio.

Debido a cómo se maneja esta ejecución, sin embargo, los desarrolladores de módulos tienen una capacidad casi infinita para modificar la forma Drupal maneja el acceso a los nodos. El propósito del resto de este capítulo es explicar cómo este sistema ha sido diseñado y las mejores maneras para que usted pueda aprovechar la API de nodo de acceso para satisfacer sus necesidades específicas.

**La función () node\_access**

node\_access () es la devolución de llamada de acceso principal para las operaciones de nodo. Se define en node\_menu () como la devolución de llamada de acceso para cualquier intento de crear, ver, editar o borrar un nodo. La función en sí es uno de los más complejos en el núcleo de Drupal en virtud de los ocho separada retorno declaraciones a futuro dentro de la función. La comprensión de la lógica detrás de estos retornos es la clave para usar correctamente el Nodo de Acceso.

Para empezar, vamos a examinar la documentación y las líneas iniciales de la node\_access () función:

/ \*\*

* Determinar si el usuario actual puede realizar la operación dada
* en el nodo especificado.

\*

* @ Param $ op
* La operación a realizar en el nodo. Los valores posibles son:
* - "Vista"
* - "Actualizar"

**[244 ]**



*Capítulo 9*



* - "Eliminar"
* - "Crear"
* @ Param $ node
* El objeto nodo en el que la operación se va a realizar, o
* tipo de nodo (por ejemplo, 'foro') para "crear" la operación.
* @ Param $ cuenta
* Opcional, un objeto de usuario que representa al usuario para quien la
* la operación se va a realizar.
* Determina el acceso de un usuario que no sea el usuario actual.
* @ Return
* TRUE si la operación puede llevarse a cabo, de lo contrario.

\* /

función node\_access ($ op, $ node, $ cuenta = NULL) {global $ user;

$ = & derechos drupal\_static (\_\_FUNCTION\_\_, array ());

De la lectura sobre el código, podemos utilizar nuestro conocimiento de Drupal para inferir algunos puntos clave:

* La $ Op parámetro indica que se solicita la operación de nodo.
* Creación de nodos es un caso especial, incluso cambiando el parámetro $ nodo enviado a la función.
* Nodo de Acceso es una acción impulsada por los usuarios. Eso significa que importe que está tratando de realizar la operación.
* Nodo de Acceso en Drupal 7 es estáticamente almacena en caché por usuario para la duración de la solicitud de página. Eso significa que una vez que se establece, no se puede cambiar hasta que se envíe una nueva solicitud o drupal\_static\_reset ('node\_access') se llama.



Recordemos nuestra discusión de $ Usuario y $ Cuenta en el capítulo anterior. La node\_access () función acepta una $ Cuenta oponerse, pero se cambia por el uso de lo global $ Usuario objetar si no se suministra. Esta característica permite que los controles de acceso que se realizan por otras que el usuario actual usuarios.



Un solo nodo puede devolver diferentes respuestas a una solicitud de acceso dependiendo de quién está haciendo la solicitud y qué petición se está hecho.

**[245 ]**



*Nodo de Acceso*



**La lista blanca en el acceso**

La primera comprobación que node\_access () hace es ver si la devolución de llamada se invoca correctamente:

if ($ node | |! in\_array ($ op, array ('view', 'update', 'Eliminar', 'crear'), TRUE)) {

* Si no había ningún nodo para comprobar en contra, o los $ op no era uno
* apoyados de los volvemos acceso denegado.

return false;

}

Este código muestra un poco de paranoia no se encuentra en la mayor parte de la API de Drupal. Comprobación de la validez de los parámetros de entrada asegura que el acceso no se concede por accidente. Cuando se trata de control de acceso, el impago a FALSE (que significa "negar el acceso") es el comportamiento apropiado.

**El almacenamiento en caché el resultado para el rendimiento**

La siguiente sección de código realiza tres chequeos simples, además de una optimización de la memoria caché estática:

* Si no se proporciona un objeto de usuario, el control de acceso es por la
* usuario actual.

if (empty ($ cuenta)) {$ cuenta = $ usuario;

}

* $ Nodo puede ser un objeto o un tipo de nodo. Dado que los tipos de nodos
* no puede ser un número entero, utilice uno nid o tipo que la estática
* identificador de cache.

$ Cid = is\_object ($ node)? $-> Nodo nid: $ nodo;

* Si ya hemos comprobado el acceso a este nodo, el usuario y op,
* devolver desde la caché.

if (isset ($ derechos [$ cuenta-> UID] [$ cid] [$ op])) {return $ derechos [$ cuenta-> UID] [$ cid] [$ op];

}

if (user\_access ('Acceso nodo de derivación', $ cuenta)) {$ derechos [$ cuenta-> UID] [$ cid] [$ op] = true;

return true;

}

**[246 ]**



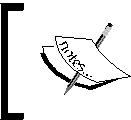
*Capítulo 9*



(! user\_access ('acceder al contenido', $ cuenta)) {$ si los derechos [$ cuenta-> UID] [$ cid] [$ op] = false; return false;

}

La primera cláusula if asegura que tenemos una adecuada $ Cuenta para el registro.



Recuerde que incluso los usuarios anónimos generan una válida $ Cuenta objeto y puede haber asignado los permisos.

La segunda cláusula refuerza la memoria caché estática. Se trata de una optimización del rendimiento de nuevo en Drupal 7.

El tercero es un user\_access () verifique nuevo en Drupal 7 y permite super-usuarios pasar todos los controles de acceso del nodo y realizar todas las operaciones en todos los nodos. Este permiso se separó de la administrar nodos permiso de versiones anteriores con el fin de indicar con mayor claridad cómo las funciones de acceso a nodo. Tiene la ventaja añadida de permitir permisos más granulares.

El último es otro user\_access () comprobar. Simplemente comprueba que un usuario puede acceder al contenido en el sitio. Si no, entonces el usuario siempre se le niega el acceso a todas las operaciones de nodo.

**Hook\_node\_access Invocando ()**

Para este punto, el código es bastante obvio y las intenciones son claras: Drupal se está ejecutando controles de seguridad básicas contra los valores conocidos. En este punto, el módulo de nodo central comienza la consulta de otros módulos sobre el estado de acceso del nodo. La siguiente pieza invocahook\_node\_access () para comprobar si las reglas de acceso:

* Otorgamos acceso al nodo si ambas de las siguientes condiciones
* se cumplen:
* - No hay módulos dicen para denegar el acceso.
* - Al menos un módulo dice a permitir el acceso.
* Si no hay ningún módulo especificado ya sea permitir o denegar, caemos de nuevo a la
* mesa node\_access.

$ Acceso = module\_invoke\_all ('node\_access', $ node, $ op, $ cuenta); if (in\_array (NODE\_ACCESS\_DENY, $ Acceso, TRUE)) {

$ derechos [$ cuenta-> UID] [$ cid] [$ op] = false; return false;

}

elseif (in\_array (NODE\_ACCESS\_ALLOW, $ de acceso, TRUE)) {$ derechos [$ cuenta-> UID] [$ cid] [$ op] = true;

return true;

}

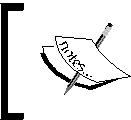
**[247 ]**



*Nodo de Acceso*



Aquí vemos una clara diferencia entre Drupal 7 y Drupal 6 (y anterior): cualquier módulo puede responder a este control de acceso. Antes de Drupal 7, sólo los módulos que definen un tipo de nodo podría responder, utilizando el viejo hook\_access () función. Esta limitación hace que sea difícil para los desarrolladores de módulos para modificar la lógica de negocio para node\_access (). Este es un cambio importante en la API de Drupal, y uno que se explorar con cierta profundidad.



Las constantes NODE\_ACCESS\_DENY y NODE\_ACCESS\_ALLOW son fijados por node.module. Veremos esto más adelante en el capítulo.

Nótese también la nota en los comentarios: Si no hay ningún módulo especificado ya sea permitir o negar, caemos de nuevo a la mesa node\_access. La orden de ejecución de Nodo Acceso engancha asuntos. Si tenemos en cuenta la lógica de nuestras reglas de negocio, hay que recordar que otros módulos también pueden tener una participación en los derechos de acceso a un nodo.

Hasta el momento, estamos hasta cinco sentencias return en el código.

**El acceso a los propios nodos de un usuario**

La cláusula siguiente es una excepción para el manejo de los nodos creados por el usuario actual:

/ / Comprobar si los autores pueden ver sus propios nodos no publicados. if ($ op == 'view' &&! $ node-> Estado && user\_access ("visión propia

contenido inédito ', $ cuenta) && $ cuenta-> el uid == $ node-> UID && $ cuenta-> UID! = 0) {

$ derechos [$ cuenta-> UID] [$ cid] [$ op] = true; return true;

}

Drupal asume que el contenido publicado no debe ser visible para los usuarios. Sin embargo, la ver propio contenido inédito existe el permiso para que los usuarios autenticados para ver su contenido, incluso si no se ha publicado. A menos que un módulo de terceros interviene, sólo los usuarios con este permiso, Acceso nodo de derivación o 01 de mayo de usuario ver contenido inédito.

**Invocando el acceso a la API de nodo**

Ahora que Drupal ha sido responsable de este caso especial, el código se cae a través de la {Node\_access} tabla para comprobar los permisos.

* Si el módulo no se anulan los derechos de acceso, utilice los establecidos
* en la tabla node\_access.

if ($ op! = 'crear' && $ node-> nid) {

**[248 ]**



*Capítulo 9*



if (module\_implements ('node\_grants')) {$ query = db\_select ('node\_access'); $ Query-> addExpression ('1 '); $ Condición de consulta-> ('grant\_' $ op, 1, '> ='.);

$ Nids = db\_or () -> condición ('nid', $ node-> nid); if ($ estado del nodo->) {

$ nids-> condición ('nid', 0);

}

$ Query-> condición ($ nids); $ Gama consulta-> (0, 1);

$ subvenciones = db\_or ();

foreach (node\_access\_grants ($ op, $ cuenta) como $ realm => $ gids) {foreach ($ gids como $ gid) {

$ subvenciones-> condición (db\_and () -> condición ('gid', $ gid) -> condición ('reino', $ dominio)

);

}

}

if (count ($ subvenciones)> 0) {$ query-> condición ($ subvenciones);

}

$ Resultado = (int) $ consulta -> execute () -> FetchField ();

$ derechos [$ cuenta-> UID] [$ cid] [$ op] = $ result; return $ resultado;

}

elseif (is\_object ($ node) && $ op == 'view' && estado del nodo-> $) {

* Si no hay módulos implementan hook\_node\_grants (), el valor predeterminado
* comportamiento es permitir que todos los usuarios ver los nodos publicados,
* así reflejar que aquí.

$ derechos [$ cuenta-> UID] [$ cid] [$ op] = true; return true;

}

}

Aquí llegamos al corazón de la API de nodo de acceso. La clave está en la función node\_access\_grants (), Que define los permisos para el usuario actual para el operación actual. Módulos responden a esta función utilizando hook\_node\_grants (), Que vamos a examinar en detalle un poco más adelante.

**[249 ]**



*Nodo de Acceso*



Esta cláusula es principalmente una función de generador de consultas, diseñado para crear el buen combinado del {Node} tabla a la {Node\_access} mesa y luego devolver el resultado de esa consulta. La comprensión de cómo se construye esa consulta - y cómo el {Node\_access} tabla se llena - es la clave para entender el Nodo de Acceso.



**Tratar con contenido inédito**

Para los autores del módulo con experiencia, una gran diferencia entre Drupal 7 y las versiones anteriores es que hook\_node\_grants () ahora se invoca para contenido inédito. Gran cuidado se debe tener cuidado para evitar la exposición de contenidos inéditos para los usuarios.



Si la API Nodo de Acceso no afirma reglas, esta cláusula será por defecto que permite el acceso al contenido. Pero note que la función no termina aquí.

elseif (is\_object ($ node) && $ op == 'view' && estado del nodo-> $) {

* Si no hay módulos implementan hook\_node\_grants (), el valor predeterminado
* comportamiento es permitir que todos los usuarios ver los nodos publicados,
* así reflejar que aquí.

$ derechos [$ cuenta-> UID] [$ cid] [$ op] = true; return true;

}

}

return false;

}

Al final de la función, observe la final devolverá FALSE; comunicado. Desde un punto de vista de la seguridad, este cajón de sastre asegura que nunca le concedemos el acceso de accidente. Siempre que escribir un cheque de acceso,FALSO debe ser su retorno por defecto.

**hook\_node\_access () en comparación con {node\_access}**

Para los desarrolladores de módulos, los dos puntos clave para la node\_access () función son los comportamientos de los dos invocaciones de hook: hook\_node\_access () y hook\_node\_grants (). Debido a la diferencia fundamental entre la forma

los dos hooks se implementan, para el resto del capítulo, nos referiremos a un módulo de acceso nodo como uno que implementa hook\_node\_grants ()y escribe en la{Node\_access} tabla. Un módulo que implementa hook\_node\_access () será referido como un módulo de control de acceso.

**[250 ]**



*Capítulo 9*



hook\_node\_access () es el más simple de los dos sistemas. Es un hook de auto-contenido que permite a los módulos de control de acceso individual para pasar juicio sobre un nodo. Tenga en cuenta, sin embargo, que en el núcleo de Drupal su uso está limitado a sólo tres de las cuatro operaciones de nodo: Crear, Actualizar y Eliminar. Esto lo podemos ver claramente en la aplicación node.module 's:

/ \*\*

\* Implementa hook\_node\_access (). \* /

node\_node\_access función ($ node, $ op, $ cuenta) {$ tipo = is\_string ($ node)? $ Node: $ node-> type;

if (in\_array ($ tipo, node\_permissions\_get\_configured\_types ())) {

if ($ op == 'crear' && user\_access ('crear'. $ tipo. 'contenido', $ cuenta)) {

volver NODE\_ACCESS\_ALLOW;

}

if ($ op == 'update') {

if (user\_access ('editar cualquier' tipo $ 'contenido', $ cuenta) |.. |.. (user\_access ('editar propia' $ tipo "contenido", $ cuenta) && ($ cuenta-> el uid == $ node -> uid))) {

volver NODE\_ACCESS\_ALLOW;

}

}

if ($ op == 'delete') {

if (user\_access ('borrar cualquier' tipo $ 'contenido', $ cuenta) |.. |.. (user\_access ('delete propia' $ tipo "contenido", $ cuenta) && ($ cuenta-> el uid == $ node -> uid))) {

volver NODE\_ACCESS\_ALLOW;

}

}

}

volver NODE\_ACCESS\_IGNORE;

}

Porque hook\_node\_access () incendios antes de comprobar el {Node\_access} mesa, que se utiliza para definir el comportamiento por defecto para los permisos de nodo. Este comportamiento es muy útil para cosas como la creación y edición de contenido según el tipo de nodo, pero puede

ser muy limitante en la definición de las reglas para la visualización de un nodo. Por esa razón, el núcleo de Drupal nunca afirma un valor a la operación Vista para un nodo.

**[251 ]**



*Nodo de Acceso*



En cambio, el núcleo devuelve la constante NODE\_ACCESS\_IGNORE, lo que indica que el acceso a la operación de Vista debe ser manejado por el {Node\_access} tabla.

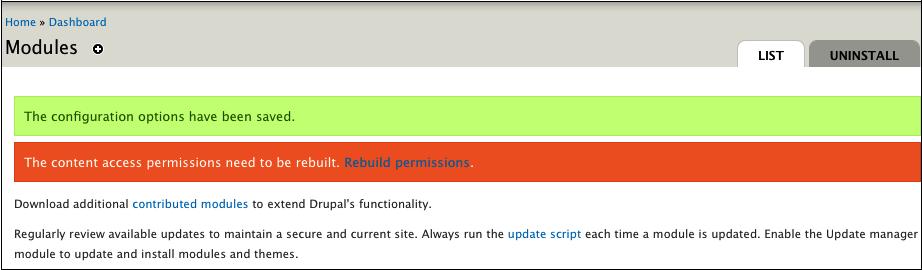
Por defecto, el {Node\_access} tabla contiene un solo registro:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **nid** | **gid** | **reino** | **grant\_view** | **grant\_update** | **grant\_delete** |
|  |  |  |  |  |  |
| 0 | 0 | todo | 1 | 0 | 0 |

Esta fila permite que el sistema de acceso de nodo para generar una combinación predeterminada de la {Node} tabla a la {Node\_access} tabla que siempre devolverá true.

Sin embargo, si su sitio no está en funcionamiento todos los módulos de acceso de nodo (es decir, no hay módulos que implementan hook\_node\_grants ()), Entonces no hay ÚNETE será requerida. En este caso, se devolverán todos los nodos.

Si instala un módulo de acceso del nodo, sin embargo, que de inmediato se le pida para reconstruir los permisos de acceso de contenido para el sitio.

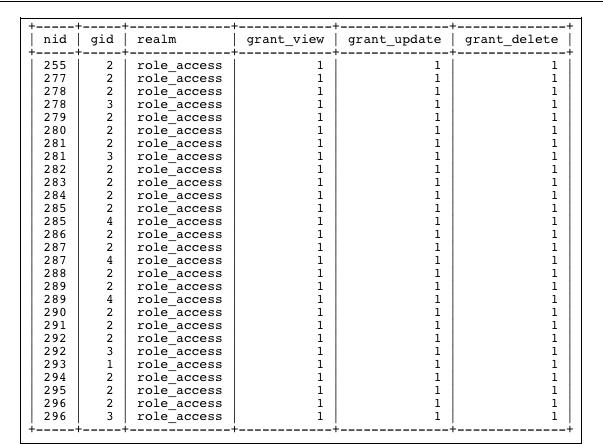


La reconstrucción de los permisos se actualizará el {Node\_access} mesa con las reglas definidas por el módulo (s) que haya instalado. Cuando activamos el módulo ejemplo estaremos construyendo en este capítulo, aquí está el resultado:

**[252 ]**



*Capítulo 9*



Tenga en cuenta que el {Node\_access} tabla utiliza el ID de nodo (nid) como su clave primaria. Como resultado, sólo se puede utilizar para comprobar el acceso para el contenido que ya se ha creado. Esto significa que los módulos de acceso de nodo no se pueden valer los permisos para la operación de creación. Para ello, el módulo debe implementar hook\_node\_access () y responder a la operación de "crear".

La reconstrucción de los permisos de acceso también se disparará node\_query\_node\_access\_ alterar () para comenzar la reescritura de consultas de nodo para la vista, actualizar y eliminar acciones.

Sin embargo, los módulos que sólo implementar hook\_node\_access () no escribir registros a la {Node\_access} mesa, ni imponen la lógica alteración de la consulta. Como resultado de ello, podríamos considerar hook\_node\_access () ser un enfoque más sencillo (y a menudo más eficiente) a resolver el problema del acceso de nodo.



Cuando se utiliza un módulo de acceso del nodo, Drupal añadirá una instrucción JOIN para cada consulta de nodo. Esto puede afectar al rendimiento de menor importancia para su sitio si usted tiene una gran cantidad de contenido.



**[253 ]**



*Nodo de Acceso*



**Uso de hook\_node\_access ()**

Drupal 7 sustituye a la antigua hook\_access () función con hook\_node\_access (). Este cambio fue una de las muchas mejoras que provienen de una sesión de trabajo en DrupalCon Szeged en 2008, y el crédito va a Larry Garfield (Crell) para implementar estos cambios.

hook\_node\_access () permite que cualquier módulo de tener voz y voto en la forma de acceso de nodo es manejado. No requiere de almacenamiento de base de datos, o el uso de el resto de la API de nodo de acceso. Además, se puede aplicar a cualquiera de las cuatro operaciones básicas CRUD.

El hook pasa tres parámetros:

* $ Node, El objeto de nodo completo se actúe sobre ellos, o una cadena de tipo de nodo en el caso de la operación de creación.
* $ Op, Que se realizó la operación.
* $ Cuenta, El usuario que solicita acceso al nodo para esta acción.

Los tres de estos parámetros siempre se rellenará, y su hook debe responder de acuerdo con la operación específica que se realiza. Hay tres posibles valores de retorno para cada solicitud:

* NODE\_ACCESS\_ALLOW

Indica que la operación se debe permitir. Se trata de una explícita permitir, y al menos un módulo debe devolver este valor para que la operación será validada por hook\_node\_access ().

* NODE\_ACCESS\_DENY

Indica que la operación no debería permitirse. Se trata de una explícita negar, y si ha sido emitido, se cancelará cualquierNODE\_ACCESS\_ALLOW directivas emitidas por otros módulos.

* NODE\_ACCESS\_IGNORE

La declaración de retorno por defecto. Este valor indica que el módulo no tiene ningún interés en la posibilidad de permitir o denegar la operación que se solicita.

**Un módulo de control de acceso de la muestra**

Por ejemplo, supongamos que tenemos la regla editorial siguiente en nuestro sitio: Cualquier usuario autenticado puede enviar los artículos siempre que él o ella ha sido un miembro del sitio por más de dos días. (Esta regla impide a las personas crear una cuenta sólo para enviar correo no deseado al sitio, sino que también podría darse el lujo de un "enfriamiento" en período anterior gente responde a otros puestos.)

**[254 ]**



*Capítulo 9*



El permiso básico "crear contenido del artículo 'no permite para este caso de uso, sino un módulo de rápida usando hook\_node\_access () puede. Digamos que es el módulo de retardo.

Suponiendo que hemos escrito nuestros del módulo delay.info archivo, podemos crear el siguiente archivo:

<? Php

/ \*\*

* Implementar hook\_node\_access ().
* Sólo permiten que los mensajes de los usuarios con cuentas de más de dos días de edad.

\* /

función delay\_node\_access ($ node, $ op, $ cuenta) {

}

Ahora podemos implementar nuestros controles de lógica personalizada. En primer lugar, hacer caso omiso de todas las operaciones que no sean crear.

función delay\_node\_access ($ node, $ op, $ cuenta) {if ($ op! = 'crear') {

volver NODE\_ACCESS\_IGNORE;

}

}



La respuesta correcta cuando el módulo no desea hacer valer el permiso de un nodo es NODE\_ACCESS\_IGNORE. Si no devuelve este valor es un error de la API.



En segundo lugar, hacemos caso omiso de todos los tipos de contenido que no sean «artículo»:

if ($ op = 'crear' | | $ node = 'artículo'!) {NODE\_ACCESS\_IGNORE retorno;

}

A continuación, tenemos que comprobar la información sobre el usuario que realiza la acción. Datos de tiempo de registro se almacena en $ Cuenta-> creado como una marca de tiempo Unix, por lo que puede compararse con la hora actual fácilmente.

/ / Drupal almacena la solicitud de la página en el REQUEST\_TIME constante. if (empty ($ cuenta-> creado) | | $ cuenta-> creado> (REQUEST\_TIME

- (48 \* 3.600))) {

volver NODE\_ACCESS\_DENY;

}

**[255 ]**



*Nodo de Acceso*



Y hemos terminado, ¿no? Bueno, hemos aplicado correctamente las reglas, que es una declaración de negación para los usuarios que no pasan el umbral de tiempo. Sin embargo, no hemos completado nuestra función, ya que la API requiere una declaración de retorno de todos los módulos, por lo que hay que añadir una nota final que indica nuestra indiferencia a otras reglas.

volver NODE\_ACCESS\_IGNORE;

Ahora hemos terminado. En un módulo sencillo, en tan sólo dieciséis líneas de código (seis de los cuales son comentarios), hemos cambiado las reglas para la creación de contenido en nuestro sitio. Aquí es todo el módulo (excepto para el archivo delay.info):

<? Php

/ \*\*

* Implementar hook\_node\_access ().
* Sólo permiten que los mensajes de los usuarios con cuentas de más de dos días de edad.

\* /

función delay\_node\_access ($ node, $ op, $ cuenta) {if ($ op = 'crear' | |! $ node = 'artículo') {

volver NODE\_ACCESS\_IGNORE;

}

/ / Drupal almacena la solicitud de la página en el REQUEST\_TIME constante. if (empty ($ cuenta-> creado) | | $ cuenta-> creado> (REQUEST\_TIME

- (48 \* 3.600))) {

volver NODE\_ACCESS\_DENY;

}

volver NODE\_ACCESS\_IGNORE;

}

Así de simple y de gran alcance hook\_node\_access () puede ser.

**Un segundo módulo de control de acceso**

Ahora, echemos un vistazo a un ejemplo un poco más complejo. Supongamos que queremos que los usuarios puedan crear contenido con normalidad, pero en vez de la edición de contenido por tipo, queremos que ellos sean capaces de editar todo el contenido creado por los usuarios que tienen los mismos roles que hacen.

Antes de escribir ese código, vamos a esbozar nuestra lógica un poco. Para la operación de la "actualización", siga estas reglas:

1. Obtenga el papel (s) del usuario que intenta editar el contenido.
2. Obtener el papel (s) del usuario que es el autor de contenido.
3. Si las funciones son iguales, que el usuario pueda editar el contenido.
4. Si el papel no coinciden, negar acceso de edición.

**[256 ]**



*Capítulo 9*



Es una buena idea escribir tales reglas y revisarlas con el patrocinador del proyecto, ya que al hacerlo puede revelar las implicaciones de la aplicación de la lógica que haya definido.

En este caso, la regla # 4 pasa a ser clave, porque vamos a negar explícitamente el acceso para modificar el contenido, incluso para los usuarios que podrían tener la corregir cualquier contenido del artículo permiso. Si lo hace, podría dar lugar a confusión para algunos usuarios, por lo que debe asegurarse de que esta es la regla que queremos.



Debemos señalar, desde una perspectiva de seguridad, que estas reglas colocarán todos los usuarios autenticados "en el mismo grupo. Este comportamiento puede ser deseable, o puede abrir un riesgo de que todos los usuarios del sitio registrados pueden editar todos los contenidos. Para el propósito de nuestro módulo de demostración, supongamos que se desea este comportamiento, y que los usuarios que necesitan su contenido protegido se les dará un papel adicional.



Dado que este es el comportamiento que queremos, la funcionalidad es bastante fácil de hacer cumplir. Vamos a crear unrole\_edit.module presentar para este comportamiento. Nuestro módulo comienza al igual que el módulo de retardo.

<? Php

/ \*\*

* Implementar hook\_node\_access ().
* Permitir a los usuarios editar los nodos creados por sus compañeros.

\* /

función role\_edit\_node\_access ($ node, $ op, $ cuenta) {if ($ op! 'update' =) {

volver NODE\_ACCESS\_IGNORE;

}

/ / Devolver la respuesta predeterminada. volver NODE\_ACCESS\_IGNORE;

}

Ya que sólo se preocupan por la operación de actualización, ignoramos todos los demás casos.

Los papeles para nuestro usuario activo se encuentran en el $ Cuenta-> papeles propiedad y de fácil acceso. Para conocer las funciones de propietario del nodo, sin embargo, debemos cargar el usuario basado en el $ node-> UID.

$ Owner = user\_load ($ node-> uid);

Para papel exacto que empareja, ahora podemos hacer unaarray\_diff () en los roles de los dos de los usuarios.

$ Diff = array\_diff ($ cuenta-> papeles, $ owner-> roles);

**[257 ]**



*Nodo de Acceso*



Si el array $ diff está vacío, entonces los dos usuarios tienen las mismas funciones y pueden ser considerados iguales.

if (empty ($ diff)) {

volver NODE\_ACCESS\_ALLOW;

}

Por último, queremos cambiar nuestro valor de retorno por defecto, ya que nuestra regla # 4 indica que queremos control de acceso estricto cumplimiento. Así que si el diff devuelve un valor, le negamos el acceso:

volver NODE\_ACCESS\_DENY;

Y eso es todo el módulo. 17 líneas de código:

<? Php

/ \*\*

* Implementar hook\_node\_access ().
* Permitir a los usuarios editar los nodos creados por sus compañeros.

\* /

función role\_edit\_node\_access ($ node, $ op, $ cuenta) {if ($ op! 'update' =) {

volver NODE\_ACCESS\_IGNORE;

}

$ Owner = user\_load ($ node-> uid);

$ Diff = array\_diff ($ cuenta-> papeles, $ owner-> roles); if (empty ($ diff)) {

volver NODE\_ACCESS\_ALLOW;

}

volver NODE\_ACCESS\_DENY;

}

Si queremos optimizar el código, podríamos añadir en dos comprobaciones adicionales. El primero simplemente comprueba para un partido de id de usuario y elimina la necesidad de que el array\_diff ():

if ($ cuenta-> el uid == $ node-> uid) {return NODE\_ACCESS\_ALLOW;

}

$ Owner = user\_load ($ node-> uid);

La segunda sería utilizar drupal\_static () para evitar correr user\_load () múltiples veces por el mismo ID de usuario.

$ propietarios = & drupal\_static (\_\_FUNCTION\_\_); if (! isset ($ propietarios [$ node-> uid])) {

$ propietarios [$ node-> el uid] = user\_load ($ node-> uid);

}

**[258 ]**



*Capítulo 9*



En las páginas de nodo listado, este hook a menudo se llama varias veces. Usted puede ser capaz de optimizar el código mediante el uso de drupal\_static () para eliminar consultas repetidas a la base de datos. Con el módulo de Desarrollo puede ayudarle a encontrar las consultas redundantes que se beneficiarían con el almacenamiento en caché estática:



http://drupal.org/project/devel

Ahora nuestro módulo es de 23 líneas de código optimizado:

<? Php

/ \*\*

* Implementar hook\_node\_access ().
* Permitir a los usuarios editar los nodos creados por sus compañeros.

\* /

función role\_edit\_node\_access ($ node, $ op, $ cuenta) {if ($ op! 'update' =) {

volver NODE\_ACCESS\_IGNORE;

}

if ($ cuenta-> el uid == $ node-> uid) {return NODE\_ACCESS\_ALLOW;

}

$ propietarios = & drupal\_static (\_\_FUNCTION\_\_); if (! isset ($ propietarios [$ node-> uid])) {

$ propietarios [$ node-> el uid] = user\_load ($ node-> uid);

}

$ Diff = array\_diff ($ cuenta-> papeles, $ propietarios [$ node-> UID] -> papeles); if (empty ($ diff)) {

volver NODE\_ACCESS\_ALLOW;

}

volver NODE\_ACCESS\_DENY;

}

Posiblemente Podríamos optimizar este aún más por el almacenamiento de la estáticamente $ Diff resultados, pero eso es probablemente innecesario y añade poco a nuestro ejemplo.

Los mismos tipos de lógica se podrían aplicar a la operación de borrado para los nodos también.

**Ver operaciones y los módulos de control de acceso**

Mientras hook\_node\_access () se puede aplicar a todas las operaciones CRUD, que muy rara vez se aplica a la operación de Vista y el núcleo de Drupal nunca lo hace. Autores del módulo se recomienda encarecidamente a volver siempre NODE\_ACCESS\_IGNORE para la operación Ver de manera que la API de nodo de acceso puede funcionar como se espera.

**[259 ]**



*Nodo de Acceso*



El razonamiento detrás de esta regla es que las reglas de acceso avanzada son más propensos a ser aplicado a la operación View; muchos sitios sólo requieren restricciones de acceso sobre quién puede ver el contenido, ya que sólo los editores y administradores pueden crear contenido.



Si implementa hook\_node\_access () sobre el funcionamiento View, que es probable que altere el comportamiento esperado de acceso del nodo de manera que tengan la depuración difícil para el dueño del sitio y otros desarrolladores de módulos.



Por lo tanto, los módulos de acceso único nodo deben valer las reglas sobre el funcionamiento View; módulos de control de acceso deben abstenerse de hacerlo.

Si usted piensa que debe hacer cumplir las normas en Vista hook\_node\_access (), Por favor documente claramente que lo ha hecho en la página del proyecto de su módulo y en un acompañamiento README.txt presentar.

**Al escribir un módulo de acceso del nodo**

Claramente, hook\_node\_access () es una función muy potente. Como hemos visto, muchos casos de uso comunes se pueden resolver en tan sólo unas pocas líneas de código. Por lo que es importante para los desarrolladores de módulos a considerar al utilizar hook\_node\_access () para implementar el control de acceso en lugar de un módulo de acceso de nodo completa.

Como los módulos de control de acceso no deben responder a la operación View, módulos de acceso de nodo se hacen necesarios cada vez que necesite usar las reglas de acceso para restringir el acceso a los nodos que un usuario puede ver. La razón de esto tiene que ver con la forma en Drupal construye sus controles de acceso al enumerar los nodos.hook\_node\_access () sólo es eficaz

para ejecutar comprobaciones de acceso con los nodos individuales - un hecho puesto de manifiesto por el hecho de que el$ Node objeto se pasa como parámetro.

Sin embargo, hook\_node\_access () De nada sirve a nosotros cuando la generación de una lista de nodos, por tres razones:

* Proporciona medios para alterar un nodo listado consulta para generar una lista adecuada de estaciones accesibles.
* Ejecución de una función de búsqueda individual para cada nodo en una página puede ser intensivo en recursos.
* Una implementación de hook\_node\_access () puede anular el comportamiento definido por otros módulos, que puede no ser deseable.

Aunque es tentador para resolver todas sus necesidades de acceso con hook\_node\_access (), Haciendo así que limita la flexibilidad de su módulo. hook\_node\_access () se recomienda para los módulos de casos de uso individuales, especialmente las escritas para un sitio o proyecto específico.

**[260 ]**



*Capítulo 9*



Escribir un módulo de acceso de nodo completo, por el contrario, es una tarea más compleja, y que vamos a examinar en detalle en el resto de este capítulo. Las ventajas de escribir un módulo de acceso de nodo son:

* Módulos de acceso Nodo pueden trabajar en conjunto con otros módulos para proporcionar reglas de control de acceso flexibles.
* Reglas de acceso Nodo establecidos por un módulo pueden ser modificadas por cualquier otro módulo, mientras que hook\_node\_access () implementaciones no pueden.
* Módulos de acceso Nodo suelen abordar los escenarios de uso comunes y por lo tanto reutilizables por más personas. Si bien esto no es una ventaja técnica, es una gran razón para contribuir código al proyecto Drupal.

**El {} node\_access mesa y su papel**

La principal diferencia entre los módulos de control de acceso y módulos de acceso de nodo es el uso de la {Node\_access} tabla. Cuando se instala un módulo de acceso de nodo, consultas de bases de datos para la {Node} tabla puede ser ÚNETEcado a la {Node\_access}

mesa con el fin de delimitar la lista de valores devueltos.

Por defecto, el {Node\_access} tabla contiene un registro único que permite que todo el contenido sea visto por todos los usuarios (excepto los que no pueden acceder al contenido, Como vimos cuando se mira en el node\_access () función).

Cuando un módulo de acceso de nodo se habilita, sin embargo, se elimina este registro predeterminado, y consultas de nodo será modificado apropiadamente. Veamos de nuevo en una función de consulta constructor muestra que devuelve una simple lista de nodos publicados:

$ Select = db\_select ('nodo', 'n') -> campos ('n', array ('nid')) -> condición ("promover", 1)

-> Condición ("status", 1) -> orderBy ('pegajosa', 'DESC') -> orderBy ('creado', 'DESC') -> extend ('PagerDefault') -> límite (10) -> addtag ​​('node\_access');

$ Nids = $ select-> execute () -> fetchCol ();

**[261 ]**



*Nodo de Acceso*



Cuando se ejecuta, la siguiente consulta se pasa a la base de datos:

SELECT n.nid DESDE DONDE n.promote nodo n = 1 Y n.status = 1 ORDER BY n.sticky DESC DESC n.created LÍMITE 0, 10;

La consulta anterior significa:

*Seleccione los diez primeros identificadores de nodos distintos de los nodos publicados y promovidos, ordenados por*

*"Pegajoso" y la edad.*

Si tenemos un módulo de acceso de nodo se activa, sin embargo, hook\_query\_alter () se disparará el node\_query\_node\_access\_alter () función, que volverá a escribir la consulta agregando un condicional ÚNETE a la mesa de acceso a nodo.



Utiliza la API de base de datos de Drupal 7 del add\_tag ​​() método para registrar las funciones de enlace. El valor pasado al método - aquí node\_access

- Informa el nombre del hook alter. Altera consulta toman el formatohook\_query\_TAG\_NAME\_alter ().



Nuestra consulta nodo de acceso de ejemplo se parece a lo que después de haber sido procesado por \_alter node\_query\_node\_access ():

SELECT DISTINCT (n.nid) desde el nodo n INNER JOIN node\_access na EN n.nid = na.nid WHERE (n.promote = 1) Y (n.status = 1) Y (((na.gid =

0) Y (na.realm = 'all')) O (((na.gid = 2) Y (na.realm = 'Acceso role\_'))) Y na.grant\_view> = 1) ORDER BY DESC n.sticky , n.created DESC LIMIT 0, 10;

En la llanura Inglés, esto es lo que significa:

*Seleccione los diez primeros identificadores de nodos distintos de los nodos publicados y promovidos, ordenados por*

*"Pegajoso" y la edad, con la condición de que una de las condiciones siguientes: 1) El valor por defecto 'todos los usuarios pueden ver todos los nodos "regla sigue vigente; o, 2) Al menos una de las becas de acceso del usuario actual permite el acceso a ver el nodo.*

Eso es mucho para tomar, así que vamos a descomprimir un poco más.

¿Cuándo node\_query\_node\_access\_alter () ejecuta, hace algunas preguntas básicas antes de modificar la consulta. Estos son:

* ¿Puede este usuario acceder a todos los contenidos?

Esta solicitud se lleva a cabo por el node\_access\_view\_all\_nodes () función, que comprueba si los módulos de nodo de acceso están habilitadas, lo que los permisos del usuario son, y si el acceso se concede por el default 'ver todo el contenido' de registros. Si es TRUE, entonces la consulta alter sale sin cambiar la consulta.

**[262 ]**



*Capítulo 9*



* ¿Puede este usuario Acceso nodo de derivación?

Nuevo en Drupal 7, el Acceso nodo de derivación el permiso se ha dividido de la administrar nodos permiso. Este permiso permite super-usuarios ignoran todas las reglas de acceso de nodo (incluidas las aplicadas por hook\_node\_access ()).

Normalmente, este permiso sólo es retenido por el usuario 1, pero en algunos sitios, seleccione papeles siempre se debe permitir para ver todos los nodos.

* ¿Qué permisos de acceso del nodo tiene este usuario?

Aquí, la función llama node\_access\_grants () para determinar qué nodo de permisos de acceso de la cuenta de usuario ha sido concedido. Estas subvenciones son devueltos por los del módulo hook\_node\_grants () función. Cada subvención devuelto se utiliza para crear una nueva cláusula OR en la consulta de acceso del nodo final.



Tenga en cuenta que dicho *Cláusula OR*. Esto es crucial. Acceso nodo en Drupal es todavía un sistema permisivo. Si está utilizando varios módulos de acceso de nodo y un acceso subvenciones, que el acceso no puede ser quitado por las subvenciones de otro módulo de acceso de nodo. Hay, sin embargo, maneras de hacer cumplir "negar" las subvenciones a través de cualquierahook\_node\_access () o hook\_node\_ grants\_alter (), Del que hablaremos más adelante.



* ¿Qué acciones estamos realizando?

La última pregunta verifica la operación que se lleva a cabo (Ver, actualización o eliminación) para añadir la cláusula de acceso adecuado en contra de la grant\_view,grant\_update, O grant\_delete columnas de la {Node\_access} tabla.

Desemejante hook\_node\_access (), La consulta se ejecuta en contra de la {Node\_access} mesa y no le importa que nodo individual que estamos tratando de ver, actualizar o eliminar. En su lugar, el sistema de acceso de nodo aquí proporciona una capa de abstracción para que las consultas del listado se pueden filtrar adecuadamente de acuerdo con las reglas de acceso de dicho lugar.

**Esquema {} node\_access tabla explica**

La {Node\_access} mesa funciona almacenando la información necesaria para el funcionamiento de un JOIN para la adecuada {Node} tabla basada en las condiciones descritas anteriormente. Para entender completamente cómo funciona el ejemplo consulta anterior (y para planificar su módulo de acceso del nodo), vamos a examinar la estructura de la propia tabla.

**[263 ]**



*Nodo de Acceso*



La {Node\_access} tabla contiene seis columnas, cada una con una función específica en el API.

* nid es un número entero y es la clave externa a la {node} tabla, utilizado como UNIRSE campo para las instrucciones SELECT.
* gid es un número entero y representa el id subvención declarada por un nodo de acceso módulo. Todo gids se emparejan con una apropiada reino, Tal como se define por el módulo.
* reino es una cadena (un máximo de 255 caracteres) que indica el nombre de la regla de acceso asignada por un módulo. Los módulos pueden definir varios reinos, y cada reino pueden tener múltiplesgids. Tomados en conjunto, un par reino / gid crea una subvención de acceso específico para la Identificación del nodo dado.
* grant\_view es un número entero que proporciona un valor booleano que indica que la concesión de esta fila permite que el nodo sea vista por los usuarios con los permisos adecuados.
* grant\_update es un número entero que proporciona un valor booleano que indica que la concesión de esta fila permite que el nodo a ser editado por los usuarios con los permisos adecuados.
* grant\_delete es un número entero que proporciona un valor booleano que indica que la concesión de esta fila permite que el nodo que sean borradas por los usuarios con los permisos adecuados.

Tomados en conjunto, cada fila de la {Node\_access} tabla define un conjunto de reglas para un nodo. Un nodo puede tener más de un conjunto de reglas. Tenga en cuenta que el {Node\_access} tabla en sí no se preocupa por el estado de la publicación del nodo. Al definir ámbitos de sus módulos, usted debe tomar en cuenta cómo manejar el acceso a contenidos inéditos.



Debido a que pueden escribir varios registros a la{Node\_access} mesa y complemento ÚNETEs para la mayoría de las consultas de los nodos, módulos de acceso de nodo pueden requerir muchos recursos. A fin de mantener un alto rendimiento, los sitios que planean utilizar los módulos de acceso de nodo general deberían permitir 5-10% de la potencia de procesamiento adicional para el servidor de base de datos.



**Definir reglas de acceso de su módulo**

Ahora que entendemos cómo los límites de Drupal acceso a los nodos, estamos listos para explorar la API para los desarrolladores de módulos. Antes de escribir ningún código, sin embargo, hay que escribir las reglas de acceso que queremos aplicar y revisar su precisión y problemas potenciales.

**[264 ]**



*Capítulo 9*

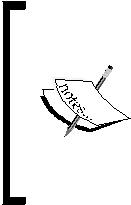


Para nuestro módulo de prueba, queremos aprovechar el papel del sistema de Drupal para crear niveles de usuarios. Cada miembro de una función sólo será capaz de ver, editar o eliminar los contenidos creados por otros miembros de ese papel. Algunos usuarios, sin embargo, puede que no se les permite editar o borrar el contenido, por lo que nuestro módulo tiene que separar los tres permisos correctamente.

Escrito, aquí es lo que nuestro conjunto de reglas se ve así:

1. Todos los usuarios serán asignados a uno o más roles.
2. Todos los nodos se asignarán subvenciones basadas en el papel (s) del autor del nodo.
3. Los usuarios pueden asignar subvenciones individuales para la vista, actualizar y eliminar acciones en función de sus roles y permisos asignados.
4. Los usuarios con los permisos adecuados podrán ver, actualizar y eliminar un nodo, independientemente de su estado de publicación.
5. Si el autor de un nodo de cambios, las subvenciones para ese nodo va a cambiar.
6. Si el autor de un nodo se le asigna nuevos roles, las subvenciones para los nodos del autor no se verán alterados a menos que se actualizan los nodos.

Estas reglas todos parecen bastante sencillo, a excepción de la regla # 6. Para ser sincero, nos ponemos en la regla # 6 que nos impida tener que escribir código adicional para manejar este caso, lo que no haría más que complicar nuestro ejemplo. Sin embargo, cuando



escribir un módulo de acceso del nodo, es necesario tener en cuenta todas las variables que pueden afectar a cómo se aplican sus conjuntos de reglas.

Recuerde:

* Las reglas de acceso de nodo se recalculan y se almacenan cada vez que se guarda un nodo
* Concesiones de acceso del usuario se calculan para cada petición
* Reglas de acceso Nodo deben reconstruirse para todo el sitio cada vez que un módulo de acceso de nodo se añade o elimina
* Garantiza el acceso de nodo pueden ser diferentes para cada una de las tres operaciones - Ver, Actualizar y Eliminar - incluso para el mismo usuario

El módulo tiene que ser consciente del contexto mayor en el que operan sus reglas.

Así que escribir el comportamiento esperado del módulo - y la publicación de esa definición en un archivo README y como en línea y la documentación de ayuda del módulo - es una parte clave para escribir un buen módulo de acceso a nodo.

Sobre la base de las reglas y pautas anteriores, tenemos una buena idea de cómo escribir nuestro módulo. Llamémoslo rol Acceso y manos a la obra.

**[265 ]**



*Nodo de Acceso*



**Crear el módulo de acceso de papel**

Comenzamos con el módulo estándar . Info archivo y un talón . Módulo presentar, como se muestra a continuación:

; $ Id $

name = Función de acceso de

Descripción = Límite de acceso al contenido por rol de usuario. core = 7.x

archivos [] = role\_access.module

Guardar como role\_access.info dentro de un / Sites / default / modules / role\_access / directorio.

A continuación, cree un talón role\_access.module archivo:

<? Php

/ / $ Id $

/ \*\*

* @ File
* Archivo de módulo de acceso de funciones.
* Proporciona un esquema de acceso al contenido simple basado en la
* El papel del autor.

\* /

Ahora estamos listos para comenzar a construir nuestro módulo. El código en las siguientes secciones será todo añadir a este archivo base.

**Uso de hook\_node\_access\_records ()**

Hacer cumplir las reglas con un módulo de acceso de nodo tienen dos partes. El primero es escribir las reglas de su módulo a la {Node\_access} *tabla.* Esta acción se lleva a cabo con lahook\_node\_access\_records () función. Esta función se llama cada vez que un nodo se crea o actualiza. Es la responsabilidad de su módulo para responder adecuadamente a este hook.



Al crear o actualizar los nodos, los módulos no deben realizar consultas de base de datos directos para {Node\_access}. Si lo hace, rompe el API porque otros módulos ya no pueden confiar en el comportamiento que se espera del sistema de acceso de nodo.



**[266 ]**



*Capítulo 9*



¿Cuándo hook\_node\_access\_records () los incendios, que pasa a un solo parámetro, la $ Node objeto que se actúe en consecuencia. Nuestro módulo debe responder sobre la base de la información en el $ Node oponerse o ser capaz de derivar sus reglas de esa información.



Esta última afirmación puede parecer obvio, pero vale la pena repetirlo. Si sus reglas de negocio se basan en la información especial que no se encuentra en el valor por defecto$ Node objeto, es su responsabilidad para añadir que los datos utilizando hook\_node\_load (). Veremos esto con más detalle más adelante en este capítulo.



Para Papel Access, es necesario conocer los roles asignados al usuario que fue autor del nodo.

/ \*\*

* Implementar hook\_node\_access\_records ().
* Queremos almacenar una fila para cada función asignada
* al autor de los contenidos.

\*

\* /

role\_access\_node\_access\_records función ($ nodo) {

* En primer lugar obtener el registro de usuario. Tenga en cuenta que evitamos el uso de $ usuario aquí,
* ya que es el objeto global $ user.

$ Cuenta = user\_load ($ node-> uid);

/ / Ahora, la matriz de funciones desde el objeto $ cuenta. papeles $ = array\_keys ($ cuenta-> roles);

Aquí se utiliza la API de Drupal para agarrar los papeles asignados al autor nodo. El uso dearray\_keys () en la última línea significa que se nos dará una sencilla tabla de id de rol. Estos identificadores de roles serán utilizados como los identificadores de subvención que almacenamos en el {Node\_access} tabla. Un típico papeles $ resultado se verá así si var\_dump () su valor:

array (2) {[0] => int (2) [1] => int (4)

}

**[267 ]**



*Nodo de Acceso*



A partir de aquí, tenemos la obligación de construir un bien formado $ subvenciones matriz que define las reglas de nuestro módulo. Esta matriz coincide con el esquema de la {Node\_access} mesa y añade un 'prioridadTecla '. Para nuestro módulo, volvemos un elemento de matriz para cada rol:

* Iniciar una array $ subvenciones. $ subvenciones = array ();
* Iterar a través de los papeles $ y obtener nuestras subvenciones.
* Utilizamos el id papel como el id de subvención, así que vamos a nombrar esa manera para mayor claridad.

foreach ($ papeles como $ grant\_id) {$ subvenciones [] = array (

'Reino' => 'role\_access', / / ​​El nombre de nuestro módulo. 'Gid' => $ grant\_id,

'Grant\_view' => 1, 'grant\_update' => 1, 'grant\_delete' => 1,

'Prioridad' => 0, / / ​​Si no es cero, se ignoran otras subvenciones.

);

}

/ / Devolver nuestras subvenciones. return $ subvenciones;

}

Inspección de la salida de este código nos muestra:

$ subvenciones [0] = array ('reino' => 'role\_access', 'gid' => 2,

'Grant\_view' => 1, 'grant\_update' => 1, 'grant\_delete' => 1, 'prioridad' => 0,

),

$ subvenciones [1] = array ('reino' => 'role\_access', 'gid' => 4,

'Grant\_view' => 1, 'grant\_update' => 1, 'grant\_delete' => 1, 'prioridad' => 0,

);

Tenga en cuenta que no es necesario para identificar el nodo en sí. El API se encarga de eso por nosotros.

**[268 ]**



*Capítulo 9*



Un par de cosas a tener en cuenta al volver a sus subvenciones nodo.

* Su módulo puede valer uno o varios "reinos", según corresponda a su lógica de negocio.
* El 'reino' debe ser una cadena única de su módulo. Namespace su subvención con el nombre del módulo. Si sólo se almacena una única subvención, utilice el nombre del módulo como el reino.
* Las tres becas son cada comprobarse por separado para determinar los permisos. Esto significa que usted puede definir las tres subvenciones (ver, actualizar y eliminar) en una sola sentencia.
* El elemento de "prioridad" es obsoleto para Drupal 7. Puede utilizarse para descartar los subsidios establecidos por otros módulos. Sin embargo, esto se realiza mejor a través de la nuevahook\_node\_access\_records\_alter (). (Ver http://drupal.org/ node/686858 para más detalles.)
* Sus donaciones declaraciones deben ser enteros (0 o 1) Y no booleanoVERDADERO o FALSO. Capa de base de datos de Drupal 7 utiliza estricta tipificación variable que Drupal 6 y por debajo, así que asegúrese de actualizar el código heredado.
* La {Node\_access} mesa no distingue entre nodos publicados y no publicados. Sólo los usuarios de confianza se debe dar permiso para acceder a los contenidos inéditos.

Ahora hemos establecido nuestras normas en la base de datos. Vamos a examinar la segunda parte del sistema de acceso a nodo.



Ahora mismo, si ahorramos un nodo con nuestro módulo de esta forma, no pasa nada. ¿Por qué? Debido a que la API de nodo Access supone que el módulo también se implementa hook\_node\_grants (). Sin ese hook, no se almacenarán sus archivos. Drupal hace esto para ahorrar en gastos base de datos asociada con el almacenamiento de los registros no utilizados.



**Uso de hook\_node\_grants ()**

Por cada solicitud de página que implica nodos, Drupal consulta los módulos activados para el nodo de acceso de subvenciones que están en vigor para el usuario actual. Módulos de responder a esta consulta mediante hook\_node\_grants ().

Desemejante hook\_node\_access\_records (), Que es el nodo-céntrica, hook\_node\_grants () está centrada en el usuario. El hook pasa dos parámetros:

* $ Cuenta - El objeto de usuario de la persona que ve la página.
* $ Op - La operación que se lleva a cabo (ver, actualizar o eliminar).

**[269 ]**



*Nodo de Acceso*



Tenga en cuenta que nosotros no tenemos acceso a la $ Node objetar aquí. Esto es porque el API de acceso a nodo se utiliza para proporcionar el filtrado avanzado de consultas, tanto para los nodos individuales y para los grupos de nodos. Este nivel de abstracción es lo que hace funcionar el acceso de nodo.

Así que nuestro módulo Papel acceso debe determinar qué permisos tiene nuestro usuario actual. Esta es una operación bastante simple, ya que las funciones de usuario están asociadas a la $ Cuenta objetar:

/ \*\*

\* Implementar hook\_node\_grants (). \* /

role\_access\_node\_grants función ($ cuenta, $ op) {/ / Obtener los roles de usuario.

papeles $ = array\_keys ($ cuenta-> roles);

* La comprobación de errores. if (empty ($ roles)) {

volver array ();

}

* Iniciar una array $ subvenciones. $ subvenciones = array ();
* Iterar a través de los roles. foreach ($ papeles como $ grant\_id) {

$ subvenciones ['role\_access'] [] = $ grant\_id;

}

* Devolver las subvenciones.

return $ subvenciones;

}

Una vez más, se espera que devuelva un $ subvenciones array. Esta gama está codificado por el ámbito (s) de nuestro módulo. Cada reino puede entonces declarar una tabla de id de la subvención.

Estos valores se concatenan para todos los módulos en el sitio, y una final $ subvenciones matriz se construye. Esta matriz se utiliza para modificar las consultas a la {Node} mesa con el fin de hacer cumplir nuestras normas de acceso a los nodos.

Estas subvenciones deben coincidir con los previstos en el hook\_node\_access\_records (), De lo contrario, las subvenciones se producirá un error y la operación será rechazada.

**[270 ]**



*Capítulo 9*



**Las consideraciones de seguridad**

El código anterior funciona bien. Pero hay un defecto potencialmente peligroso en su lógica: no tenemos en cuenta las variaciones de las diferentes operaciones. Como está escrito, el módulo da View, Actualizar y Eliminar el acceso a todos los nodos basados ​​en rol de usuario. Esto podría ser un gran problema si no queremos que algunos roles para eliminar contenido.

Una forma de corregir este problema es aprovechar el sistema de permiso básico para establecer normas adicionales que nuestro módulo implementa. Podemos asignar permisos específicos para permitir el acceso a cada función a las distintas operaciones.

Si usted recuerda el Capítulo 8, Drupal permisos y seguridad, la aplicación de hook\_permission () nos da una manera fácil de hacer esto.

/ \*\*

* Implementar hook\_permission ().
* Definir los permisos de los módulos de la siguiente manera:
* - Vista de papel acceder al contenido
* - Actualización de contenidos Acceso papel
* - Eliminar el contenido de acceso de función
* Nombrar éstos adecuadamente ayuda a evitar conflictos con otros módulos.
* Tenga en cuenta que nombramos éstos en función del valor $ op aprobada por
* hook\_node\_grants (). Esto nos permite utilizar la concatenación de cadenas
* al realizar el proceso de registro de acceso.

\* /

función role\_access\_permission () {return array (

'Ver el contenido de acceso función' => array (

'Title' => t ('View papel con restricción de contenido'), 'description' => t ('Permitir a los usuarios ver el contenido asignado por

papel. '),

),

'Actualización de contenido de acceso función' => array ('title' => t ('Editar papel con restricción de contenido "),

'Description' => t ('Permitir a los usuarios editar el contenido asignado por papel.'),

),

'Acceder a contenidos función delete' => array (

'Title' => t ('Borrar el contenido de rol restringida'), 'description' => t ('Permitir a los usuarios eliminar el contenido asignado por

papel. '),

),

);

}

**[271 ]**



*Nodo de Acceso*



Una vez que tenemos estos permisos en su lugar, simplemente podemos hacerlas cumplir dentro hook\_node\_grants (). Hay que añadir la lógica permiso para nuestra foreach bucle.

|  |  |
| --- | --- |
| función | role\_access\_node\_grants ($ cuenta $ op) { |
| / / Get | los roles de usuario. |
| papeles $ | = Array\_keys ($ cuenta-> roles); |

* La comprobación de errores. if (empty ($ roles)) {

volver array ();

}

* Iniciar una array $ subvenciones. $ subvenciones = array ();
* Iterar a través de los roles. foreach ($ papeles como $ grant\_id) {
  + **Compruebe el permiso de devolución de llamada!**

**if (user\_access ($ op 'acceder al contenido papel').) {$ subvenciones ['role\_access'] [] = $ grant\_id;**

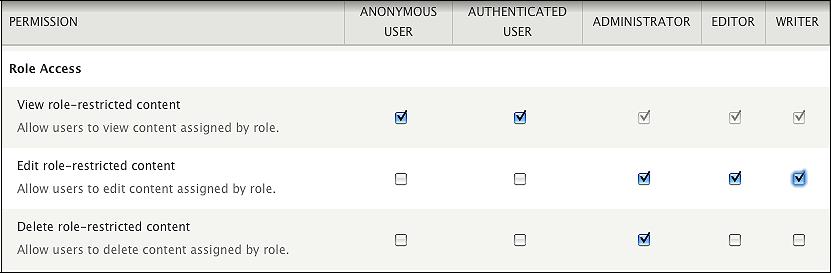
**}**

}

/ / Devolver las subvenciones. return $ subvenciones;

}

Ahora tenemos un mecanismo para restringir las reglas de acceso en función de la operación que se realiza. Nuestra configuración recomendada es la siguiente:



**[272 ]**

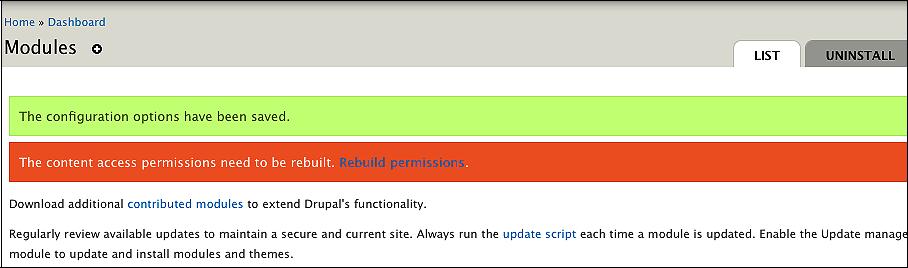


*Capítulo 9*



**La reconstrucción de la {} node\_access mesa**

Una de las partes más difíciles del sistema de acceso de nodo es la reconstrucción de la {Node\_access} tabla. La primera vez que instala un módulo de acceso de nodo, te darás cuenta de una advertencia en la parte superior de la página de configuración, que le pide que reconstruir los permisos.



Como administrador del sitio, siempre hay que reconstruir los permisos cuando se le pida hacerlo. Como desarrollador de módulo, usted es responsable de asegurar que esos permisos se reconstruyen correctamente.

En nuestro ejemplo de código del módulo, evitamos este problema basándose en los datos que está siempre presente en el$ Node oponerse, la identidad del usuario, a partir del cual podemos derivar las funciones del usuario. Sin embargo, si su módulo se basa en datos que no están almacenados en el núcleo de Drupal o módulos de terceros - ambos de los cuales debe figurar como dependencias [] en su module.info archivo - entonces es su responsabilidad para almacenar los datos necesarios para reconstruir la {Node\_access} tabla correctamente.

Por ejemplo, echemos un vistazo rápido en el módulo de acceso de dominio. Este módulo almacena información acerca de sus subvenciones en el {Domain\_access} tabla, que refleja gran parte de los datos en {Node\_access}. El esquema de la tabla es el siguiente:

$ Schema ['domain\_access'] = array (=> array 'campos' (

'Nid' => array ('type' => 'int', 'sin signo' => TRUE, 'not null' => TRUE, 'default' => 0),

'Gid' => array ('type' => 'int', 'sin signo' => TRUE, 'not null' => TRUE, 'default' => 0),

'Reino' => array ('type' => 'varchar', 'length' => '255 ',' not null '=> TRUE,' default '=>'')),

'Clave principal' => array ('nid', 'gid', 'reino'), => array 'índices' (

'Nid' => array ('nid')),

);

**[273 ]**



*Nodo de Acceso*



Acceso al dominio sigue la pista de dos reinos separados, sino que establece las tres operaciones de subvención a VERDADERO para cada nodo. Así que esta tabla almacena sólo los datos necesarios para reconstruir la tabla de acceso de nodo.

Para asegurarse de que sus datos están presentes durante la reconstrucción, el módulo debe aplicar hook\_node\_load (). Esto asegurará que los datos requeridos por la aplicación de hook\_node\_access\_records () está disponible para usted.



Es importante para cargar estos datos en hook\_node\_load () en lugar de en el interior hook\_node\_access\_records () por la siguiente razón. Otros módulos pueden querer actuar sobre la base de los datos - en particular los módulos



que implementan hook\_node\_access\_records\_alter (). Mientras hook\_node\_load () permite la $ Node oponerse a ser alterado y ampliado, hook\_node\_access\_records () no lo hace. Por lo que es su módulo de responsabilidad de garantizar que los datos utilizados por la lógica de acceso a nodo se cargan en la$ Node oponerse correctamente.

Puesto que el módulo Papel acceso siempre se puede acceder a $ Node-> UID para obtener sus datos, no vamos a preocuparse por el almacenamiento de datos para nuestro módulo.

**Modificar el comportamiento de otros módulos**

Nuestra elección de Papel Access como un módulo de la muestra fue intencional, por dos razones: en primer lugar, podemos ignorar el problema de almacenamiento de datos se discutió anteriormente; en segundo lugar, el sistema de papel nos da una buena oportunidad de ver cómo otros módulos pueden modificar el comportamiento de los módulos de acceso a nodo.

Si ha guardado e instalado el código de Papel El acceso a este punto, se verá que funciona muy bien, pero con dos problemas potenciales.

* Para la mayoría de los sitios, los usuarios anónimos no se les permite crear contenido, sino que se les permite ver el contenido. Desde Papel acceso restringe la operación de Ver para usuarios con el mismo papel, esto significaría que los usuarios anónimos no puedan ver cualquier contenido.
* Todas las funciones personalizadas también etiquetados como usuarios autenticados (id papel 2). Esto significa que cualquier contenido creado por alguien en un papel 'administrador' también se ha marcado para los usuarios autenticados. Esto parece demasiado permisiva.

Podríamos escribir algo de lógica en el módulo Papel Acceso para manejar estos casos de uso, pero también puede darse el caso de que la funcionalidad por defecto es correcta. Así que en lugar de editar el módulo o crear algunos ajustes del módulo especiales y el manejo de excepciones, en Drupal 7 se puede escribir un módulo de extensión simple que modificar el comportamiento del módulo principal.

**[274 ]**



*Capítulo 9*



Usando el nuevo hook\_node\_access\_records\_alter () y hook\_node\_grants\_ alterar (), Que básicamente podemos alterar el comportamiento de cualquier otro módulo de acceso a nodo. Para ello, vamos a crear el acceso Papel Extender módulo para implementar nuestros comportamientos opcionales.

**Uso de hook\_node\_grants\_alter ()**

Cuando se utiliza el acceso de nodo alteran hooks, debemos decidir: ¿Hay que modificar lo que se guarda en la base de datos {Node\_access} o debemos alterar la forma en que el usuario interactúa con los datos almacenados? Dado que lo que se quiere convertir este módulo off, la mejor solución es dejar

{Node\_access}solo y en vez alterar las subvenciones al cargar la página. Esto lo hacemos con la nuevo Drupal 7 hook\_node\_grants\_alter ().

hook\_node\_grants\_alter () es un hook muy potente. Después de Drupal ha reunido toda los permisos de acceso del conjunto de nodos por los módulos de su sitio, este hook se dispara y permite a un módulo para modificar la acumulada$ subvenciones matriz por referencia. En la estructura, el hook se parece mucho a hook\_node\_grants (). Pasa Y $ subvenciones, Además del usuario que solicita $ Cuenta objeto y la solicitada $ Op.

Para hacer nuestra primera obra regla, tenemos que controlar la operación Ver y decidir qué roles de usuarios pueden ver el contenido como si estuvieran en la función de usuario autenticado. En primer lugar, creamos nuestrarole\_access\_extend.info archivo, y entonces se crea una

role\_access\_extend.module.

<? Php

/ / $ Id $

/ \*\*

* @ File
* Función de acceso de Extend
* Opciones de configuración adicionales para Papel de Acceso.

\* /

Sabemos que necesitamos una opción de configuración para permitir a los usuarios anónimos para ver el contenido que los usuarios autenticados. En lugar de crear una nueva configuración, podemos volver a aprovechar el sistema de permisos:

/ \*\*

\* Implementar hook\_permission (). \* /

función role\_access\_extend\_permission () {return array (

'Acceder a la vista papel como usuario autenticado' => array (

**[275 ]**



*Nodo de Acceso*



'Title' => t ('Ver el contenido de rol limitado como usuario autenticado'),

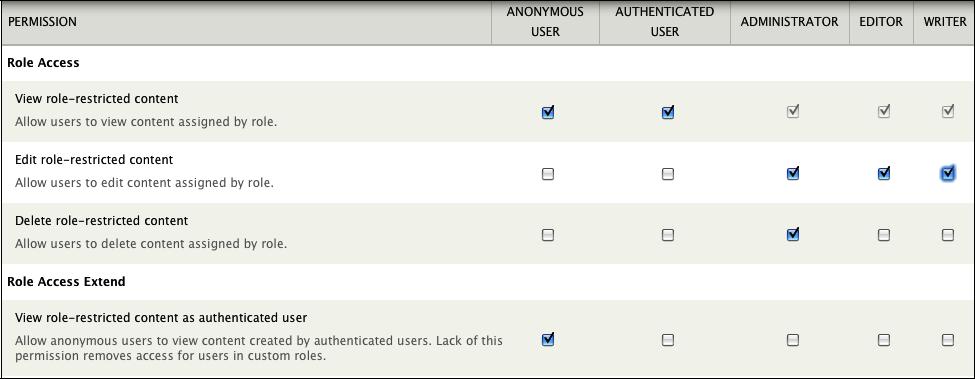
'Description' => t ('Permitir a los usuarios anónimos para ver contenido creado por usuarios autenticados. Falta de este permiso elimina el acceso para los usuarios en roles personalizados.'),

),

);

}

Así que ahora tenemos una nueva configuración de permisos:



A continuación, podemos utilizar hook\_node\_grants\_alter () modificar los permisos que el anonimato (y otros usuarios) tienen. Para ello, tenemos que entender el formato del array $ subvenciones que se pasa a nuestro hook.

Drupal reúne estas subvenciones con el node\_access\_grants () función, que combina todas las subvenciones de módulos en una sola matriz asociativa de matrices. La $ subvenciones claves del array son los reinos de control de acceso; y la matriz asociada a estas teclas indican los identificadores de subvenciones que están activas para ese reino. La var\_dump () de una típica $ subvencionesarray es así:

array (1) {

["Role\_access"] => array (1) {

1. => Int (2),
2. => Int (4),

}

}

**[276 ]**



*Capítulo 9*



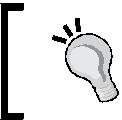
Podemos modificar cualquier elemento de esta matriz, agregando o quitando elementos que se adapten a nuestras reglas de negocio. Recuerde, sin embargo, que la matriz resultante se utilizará para escribir una ÚNETE consultar a la {Node\_access} tabla. Puede ayudar a leer la matriz anteriormente en que contexto. Una consulta de nodo estándar podría ejecutar una sencillaSELECT:

SELECT titulo, nid DESDE DONDE nodo de estado> 0 ORDER BY pegajosa, creado LÍMITE 10;

Cuando se aplican los ganglios concesiones de acceso, la consulta se ejecuta como:

N.title SELECT, n.nid DE nodo n INNER JOIN node\_access na EN n.nid = na.nid WHERE (na.realm = 'role\_access' Y na.gid = 2) Y n.status> 0 ORDER BY pegajosa, creada LÍMITE DE 10;

Como autor del módulo, es su responsabilidad de entender cómo se pueden reescribir estas consultas para que su código puede producir los resultados deseados.



Recuerde que los identificadores de subvención son la matriz *valores*, No la matriz *teclas* para el acceso del nodo reino!

Ahora que sabemos cómo se verá afectada la consulta, se puede escribir el código para agregar la subvención necesaria para que los usuarios anónimos actúan como usuarios autenticados.

/ \*\*

\* Implementar hook\_node\_grants\_alter (). \* /

role\_access\_extend\_node\_grants\_alter función (& $ subvenciones, $ cuenta $ op)

{

* Nosotros sólo actuamos en la operación 'view'.
* Si nuestra concesión no está presente, no hacer nada.

if ($ op = 'vista' | | isset ($ subvenciones ['role\_access'])!) {return;

}

* Obtiene el identificador rol definido por "usuario autenticado". $ Deshacerse = DRUPAL\_AUTHENTICATED\_RID;
* Compruebe el permiso y fijar la subvención.

if (user\_access ('Acceso vista rol como usuario autenticado')) {$ subvenciones ['role\_access'] [] = $ deshace;

}

}

**[277 ]**



*Nodo de Acceso*



Este código será otorgar a los usuarios anónimos con el permiso de acceso adecuado para ver el contenido como si fueran autentificados.



**Advertencia de seguridad!**

Tenga mucho cuidado con cualquier código que proporciona este tipo de escalada de privilegios. Por ejemplo, si no logramos comprobar que $ Op == 'vista' estaríamos dando a los usuarios anónimos permiso para ver, actualizar y borrar todo el contenido en el sitio!



El ejemplo de arriba es grande, pero lo que si usted quiere restringir las funciones personalizadas para sólo ver el contenido creado por la gente de esos roles? Es decir, que tenga que quitar la posibilidad de ver el contenido como un usuario autenticado. Con una ligera modificación en el código, podemos hacerlo:

/ \*\*

\* Implementar hook\_node\_grants\_alter (). \* /

role\_access\_extend\_node\_grants\_alter función (& $ subvenciones, $ cuenta $ op)

{

* Nosotros sólo actuamos en la operación 'view'.
* Si nuestras subvenciones no está, no haga nada.

if ($ op = 'vista' | | isset ($ subvenciones ['role\_access'])!) {return;

}

/ / Comprobar el permiso.

$ Acceso = user\_access ('acceder a la vista papel como usuario autenticado');

* Obtiene el identificador rol definido por "usuario autenticado". $ Deshacerse = DRUPAL\_AUTHENTICATED\_RID;
* Revise los usuarios autenticados.

**if ($ cuenta-> UID> 0) {**

* **Los usuarios con más de una función deben haber "autenticado usuarios '**
* **eliminado.**

**if (count ($ cuenta-> papeles)> 1 && in\_array ($ deshacerse, $ subvenciones ['acceso role\_']) &&! $ Acceso) {**

* **La matriz de las subvenciones es del orden $ subvenciones [$ reino] [$ clave] =>**
* **$ Valor, por lo que darle la vuelta, sin establecer, y voltear hacia atrás.**

**$ subvenciones ['role\_access'] = array\_flip ($ subvenciones ['role\_access']); unset ($ subvenciones ['role\_access'] [$ deshacerse]);**

**$ subvenciones ['role\_access'] = array\_flip ($ subvenciones ['role\_access']);**

**}**

**}**

**[278 ]**



*Capítulo 9*



/ / Fecha de usuarios anónimos. else if ($ acceso) {

$ subvenciones ['role\_access'] [] = $ deshacerse;

}

}

Con este código en su sitio, podemos asignar fácilmente los permisos adecuados para permitir que los roles ver contenido como si fueran usuarios autenticados.

**Uso de hook\_node\_access\_records\_alter ()**

Esto es genial! Uso muy poco código, hemos hecho un cambio importante en la lógica de negocio del módulo Papel acceso, algo que era casi imposible antes

a Drupal 7.

Todavía tenemos un problema, sin embargo. Dado que todos los roles personalizados también se les da el papel de "usuario autenticado ', estamos almacenando subvenciones en el {Node\_access} tabla que puede ser demasiado permisiva. Puede ser que no queremos para almacenar los registros en absoluto. Así que tenemos otro hook que podemos usar, junto con un nuevo permiso.

En primer lugar, editamos role\_access\_extend\_permission ():

/ \*\*

\* Implementar hook\_permission (). \* /

función role\_access\_extend\_permission () {return array (

'Acceder a la vista papel como usuario autenticado' => array (

'Title' => t ('Ver el contenido de rol limitado como usuario autenticado'),

'Description' => t ('Permitir a los usuarios anónimos para ver contenido creado por usuarios autenticados. Falta de este permiso elimina el acceso para los usuarios en roles personalizados.'),

),

**'Asignar acceso rol de usuario autenticado' => array (**

**'Title' => t ('Save contenido de rol limitado como usuario autenticado'),**

**'Description' => t ('Save contenido nuevo y actualizado para que los usuarios autenticados tienen permisos. <em> Normalmente esto está desactivado. </ Em>'),**

**),**

);

}

**[279 ]**



*Nodo de Acceso*



Este permiso será informar a nuestro uso de hook\_node\_access\_records\_alter ().

hook\_node\_access\_records\_alter () es casi idéntico al registros hook\_node\_access\_ (). La función pasa a la Y $ subvenciones matriz por referencia, además de la $ Node se actúe en consecuencia.

/ \*\*

* Implementar hook\_node\_access\_records\_alter ().
* Si un usuario guarda el contenido, asegúrese de que un registro de acceso para la
* Papel 'usuario autenticado "en realidad se debe almacenar.

\* /

función role\_access\_extend\_node\_access\_records\_alter (& $ subvenciones, $ node)

{

Si ejecutamos un var\_dump () en el típico $ subvenciones se pasa a esta función, vemos una matriz que debe parecer familiar:

array (2) {[0] => array (6) {

["Campo"] => string (11) "role\_access" ["gid"] => int (2)

["Grant\_view"] => int (1) ["grant\_update"] => int (1) ["grant\_delete"] => int (1) ["prioridad"] => int (0)

}

[1] => array (6) {

["Campo"] => string (11) "role\_access" ["gid"] => int (5)

["Grant\_view"] => int (1) ["grant\_update"] => int (1) ["grant\_delete"] => int (1) ["prioridad"] => int (0)

}

}

Lo que tenemos que hacer es asegurarse de que 'role\_access' el reino sólo devuelve Identificación subvención 2 si el rol del usuario lo permite. Así que corremos un cheque por los permisos del usuario y modificar la$ subvenciones matriz como sea necesario.

/ / No podemos usar el global $ user aquí; queremos que el creador / editor del contenido.

$ Cuenta = user\_load ($ node-> uid);

**[280 ]**



*Capítulo 9*



/ / Comprobar el permiso.

$ Acceso = user\_access ('asignar acceso rol como usuario autenticado', $ cuenta);

* Obtiene el identificador rol definido por "usuario autenticado". $ Deshacerse = DRUPAL\_AUTHENTICATED\_RID;
* Ahora agregue el papel.

if ($ acceso) {$ subvenciones ['role\_access'] [] = array (

'Reino' => 'role\_access', 'gid' => $ deshacerse, 'grant\_view' => 1, 'grant\_update' => 1, 'grant\_delete' => 1, 'prioridad' => 0,

);

}

/ / O se lo quite. else {

foreach ($ subvenciones como $ key => $ {subvención)

if ($ subvención ['reino'] = 'role\_access'!) {continue;

}

if ($ subvención ['gid'] == $ deshace) {unset ($ subvenciones [$ key]);

}

}

}

}

Cuando se ejecuta este código, nuestro $ subvenciones se modificará según sea necesario, y los registros enviados a la {Node\_access} tabla reflejará los nuevos permisos. Otro var\_dump () se ve así:

array (1) {[1] => array (6) {

["Campo"] => string (11) "role\_access" ["gid"] => int (5)

["Grant\_view"] => int (1) ["grant\_update"] => int (1) ["grant\_delete"] => int (1) ["prioridad"] => int (0)

}

}

**[281 ]**



*Nodo de Acceso*



Ahora tenemos un conjunto de reglas de avanzada que nos da una gran flexibilidad, y tiene dos nuevos hooks en su caja de herramientas de Drupal.

**Probar y depurar su módulo**

Pruebas y módulos de acceso de nodo de depuración presenta un desafío particular en Drupal, en gran parte porque la mayoría de las reglas de acceso son basado en el usuario. Ese hecho, combinado con la capacidad del usuario 1 para anular todas las comprobaciones de acceso, significa que no se puede probar módulo de acceso de nodo a través del navegador mientras está conectado como usuario 1. Tampoco se puede probar habiendo iniciado sesión como un usuario que tenga elAcceso nodo de derivación permiso, ya que el permiso hace que todo el sistema de acceso de nodo a ser ignorado, otorgando al usuario Ver, Actualizar y Eliminar el permiso de todos los nodos.

Si bien no tenemos espacio para escribir una suite SimpleTest para nuestro módulo aquí, hay algunos trucos simples que usted puede recordar para que su desarrollo (y el apoyo!) La vida más fácil.

* No pruebe nunca como usuario 1 o usuario que pueda Acceso nodo de derivación.
* Usted puede usar hook\_node\_load () y hook\_node\_view () para añadir la regla de las unidades dentro del objeto de nodo para la exhibición. Si lo hace, será sólo asegúrese de mostrar esta información a usuarios de confianza.
* Recuerde que debe examinar el contenido de {Node\_access} después de guardar un nodo.

Asegúrese de que las reglas de la tabla reflejan la lógica del código.

* Asegúrese de que los datos que usted necesita para guardar sus reglas se carga en el $ Node se oponga para que pueda reconstruir de forma segura {Node\_access} cuando se necesita.
* Asegúrese de probar el acceso tanto a los contenidos publicados y no publicados.

Estas directrices le ayudarán, pero hay una mejor, más rápido y más fácil de depurar el código de trabajo.

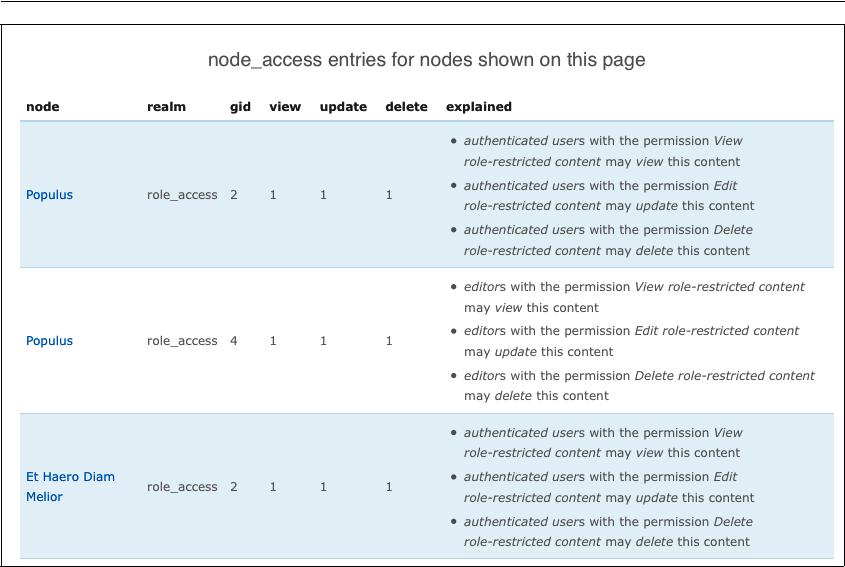
**Usando Desarrollo Nodo de Acceso**

El módulo Devel nodo de acceso es parte de la suite módulo Devel (http://drupal. org / proyecto / desarrollo). El módulo se mantiene por salvis (http://drupal.org/ user/82964) Y le da una vista basada en navegador en cómo las reglas de acceso de nodo son están aplicando en su sitio.

**[282 ]**



*Capítulo 9*



La clave para el Desa nodo de acceso es su propio sistema de hook interno, que permite a los módulos de acceso de nodo para declarar un resumen legible de sus reglas. Escribir este libro es una buena práctica, ya que ayuda a articular lo que hace su módulo.

**Uso de hook\_node\_access\_explain ()**

hook\_node\_access\_explain () es una función que deben ser respondidos sólo por el módulo que establece las subvenciones devueltos por hook\_node\_access\_records (). Así que para nuestro ejemplo, vamos a ponerlo en práctica en el módulo de acceso de base de funciones.

El hook pasa un argumento $ Row, Que es un objeto que representa una fila de la {Node\_access} tabla. Su módulo debe inspeccionar los datos y responder con información de si el $ Row pertenece a la misma.

/ \*\*

\* Implementar hook\_node\_access\_explain (). \* /

role\_access\_node\_access\_explain función ($ fila) {/ / Si no es nuestra subvención, ignorar.

if ($ row-> realm = 'role\_access'!) {return NULL;

}

**[283 ]**



*Nodo de Acceso*



* Obtenga una lista de roles de usuario. papeles $ = user\_roles ();
* Recibe nuestras definiciones de permisos.

$ permissions = role\_access\_permission ();

* Inicie una variable de resultados, para la tematización de la salida. $ resultados = array ();
* Compruebe cada regla de acceso.

foreach (array ('view', 'update', 'Eliminar') as $ op) {if (in\_array ($ row-> gid, array\_keys ($ roles))) {

$ resultados [] = t ('% roles con el permiso% perm podrá% op este contenido', array ('función%' => $ papeles [$ row-> gid], '% perm' =>

$ permisos [$ op. 'Acceder al contenido rol'] ['title'], '% op' => $ op));

}

}

* Prepararse para la tematización. Los $ resultados ya están seguros, de haber corrido
* a través de t ().

variables $ = array ('items' => $ resultados, 'title' =>'', 'type' => 'ul',

);

/ / Devuelve una lista con temas muy bien. devolver el tema ('item\_list', $ variables);

}

Al proporcionar este hook, tanto el desarrollador y los usuarios pueden habilitar módulo Devel Nodo de Acceso para ver cómo se están aplicando las reglas de acceso de nodo.

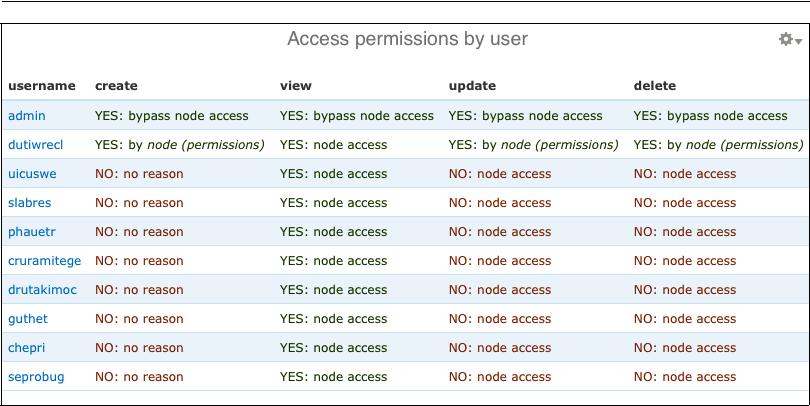
**Usando el Desarrollo Nodo de Acceso a bloque usuario**

El módulo Devel Nodo de Acceso también proporciona un bloque que muestra los resultados de la node\_access () función. Este bloque puede ayudar a clasificar a través de la razón (s) por qué se ha concedido o denegado el acceso. Se presenta una tabla que muestra los diez más recientes visitantes del sitio y su acceso a un nodo específico.

**[284 ]**



*Capítulo 9*



En el caso anterior, la dutiwrecl usuario ha sido concedido permisos de edición por el módulo de nodo. Otros usuarios pueden ver el contenido debido a un módulo de acceso de nodo (en este caso de acceso de dominio, http://drupal.org/project/domain) Ha concedido el acceso.

Si revisa primero nuestra discusión de cómo la node\_access () función opera, puede ver rápidamente lo útil puede ser la utilidad de este desarrollador.

**Resumen**

Este ha sido un largo capítulo, y esperamos que lo encontró gratificante. Comprender y utilizar el acceso de nodo es una de las herramientas más poderosas en el API de Drupal. Hemos cubierto una amplia gama de temas, pero los puntos clave a tener en cuenta son:

* ¿Cómo se determina el acceso a un nodo
* Para utilizar siempre la sintaxis de consulta dinámica de listas de nodos y marcar los nodos consultas con add\_tag ​​('node\_access')
* Las diferencias entre los hook\_node\_access () y escribir un módulo de acceso del nodo
* Cómo alterar crear, ver, actualizar y eliminar el acceso a los nodos individuales
* Los tres grandes operaciones controladas por el API Nodo de Acceso
* Cómo filtrar el listado de consultas mediante la API de nodo de acceso

**[285 ]**



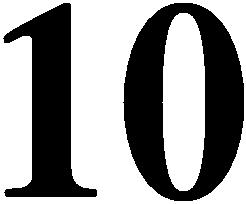
*Nodo de Acceso*



* Cómo escribir su propio módulo de nodo de acceso
* Cómo modificar módulos existentes de acceso a los ganglios
* La importancia del almacenamiento de los datos requeridos por la norma establecida y lo carga con hook\_node\_load ()

Con una comprensión de estos elementos y las herramientas de depuración proporcionadas por Devel nodo de acceso, usted debería ser capaz de aplicar las reglas de acceso necesarios para cualquier proyecto de Drupal.

**[286 ]**



JavaScript en Drupal

JavaScript se utiliza en Drupal como en la mayoría de otras aplicaciones web. Se utiliza para las funciones de energía, como la superposición, autocompletado, arrastrar y soltar, y así sucesivamente. Este capítulo se centrará en la integración de JavaScript en Drupal y cómo utilizar los helpers de JavaScript.

Algunos de los temas importantes en este capítulo se cubren son:

* Cómo agregar JavaScript a las páginas a través de Drupal
* Alterar JavaScript agregar a una página con Drupal core y otros módulos
* Uso de las funciones de ayuda integradas en Drupal
* Trabajar con la tematización y traducciones en JavaScript
* Trabajar con AJAX y Drupal

Al final de este capítulo usted debe tener el conocimiento de base para trabajar con JavaScript dentro de Drupal.

**JavaScript dentro de Drupal**

JavaScript es una parte integral de Drupal proporciona características dinámicas, una experiencia de administración única, y una biblioteca de JavaScript para desarrolladores de módulos para su uso. El framework jQuery JavaScript viene con Drupal junto con varios plugins jQuery. JavaScript se proporciona el uso de la API de la Biblioteca, que está disponible para los módulos para aprovechar.

Junto con jQuery 1.4.4 los siguientes plugins son proporcionados por Drupal:

* jQuery UI 1.8.6
* jQuery galleta, un plugin utilidad simple y ligero para leer, escribir y borrar las cookies

*JavaScript en Drupal*



* jQuery Form, un plugin para actualizar fácilmente y discretamente formularios HTML para usar AJAX
* IQUERY vez, que filtra todos los elementos que había el mismo filtro que se les aplica previamente
* jQuery BBQ, un botón de retroceso y biblioteca de consultas
* Farbtastic, una rueda de color

**Adición de JavaScript**

La mayor parte del JavaScript escrito para Drupal utiliza jQuery, pero no es un requisito. Cuando se añade JavaScript a la página, hay algunas cosas a tener en cuenta.

Drupal establece jQuery para usar su modo sin conflicto. Esto significa que el $ variable se abandonó tan Drupal puede trabajar con otras bibliotecas de JavaScript que pueden utilizar la $ variable. Más detalles están disponibles en http://api.jquery.com/jQuery.noConflict.

A falta de$ hay dos métodos para escribir JavaScript que utiliza jQuery. La primera es utilizar jQuery en cualquier lugar donde se pudo haber utilizado la $. Por ejemplo:

jQuery (). ready (function () {

...

});

La otra forma es envolver el código en una función anónima y elegir un alias. Por ejemplo:

(Function ($) {$ (). Ready (function () {

...

});

}) (JQuery);

En este caso jQuery se pasa en con el alias de $. Técnicamente, $ puede ser reemplazado con un alias válido diferente de su elección. Esto funciona para JavaScript dentro de un archivo o cuando se coloca en línea en la página.

Drupal tiene la capacidad de procesamiento previo archivos de JavaScript que varios archivos se convierten en menos archivos. Preprocesamiento ofrece una mejora de rendimiento para los usos finales, ya que tienen un menor número de archivos para descargar. Para el preprocesador para crear válida

JavaScript se recomienda que en los lugares donde se permite y comas de JavaScript opcionales, deben ser utilizados. En el ejemplo anterior, el punto y coma después de la clausura (JQuery) es un ejemplo de dónde usar los puntos y comas opcionales.

**[288 ]**



*Capítulo 10*



**Adición de archivos JavaScript y CSS a. Archivos info**

El método más simple para añadir JavaScript y CSS a una página es mediante su inclusión en el

. Información de archivo para un módulo. Cuando JavaScript y CSS se especifican en un archivo de información. Éstos se añaden a todas las páginas y configurados para utilizar procesamiento previo (más sobre esto más adelante). Un ejemplo que agrega un archivo de script y CSS se ve así:

Scripts [] = foo.js hojas de estilo [Pantalla] [] = bar.css

Cada uno de ellos es un archivo y la ruta de acceso es relativa a la raíz del módulo. La guiones la propiedad es un conjunto de archivos de script. La las hojas de estilo propiedad es una matriz de tipos de medios y de cada tipo de medio es un array de archivos CSS.

**Uso de drupal\_add\_js ()**

El método más común de la adición de JavaScript para una página es mediante el uso de la función drupal\_add\_js (). Esta función de utilidad brinda la posibilidad de agregar los archivos (tanto externa a Drupal y dentro del sistema de archivos), añadir inline JavaScript y pasar variables entre PHP y JavaScript.

Normalmente en cascada Hojas de estilo (CSS) se utilizan junto JavaScript. Drupal proporciona una función para añadir CSS a una página que funciona de manera similar adrupal\_add\_js (). Se llama drupal\_add\_css (). Las API entre las dos funciones son casi idénticos. El conjunto de características se diferencia en que CSS no tiene variables para pasar de PHP y las hojas de estilo que los medios de comunicación y opciones de Internet Explorer.

A través de los ejemplos de este capítulo vamos a crear un módulo Hello World que muestra Hello World en diferentes formularios utilizando JavaScript. La definición de la función de drupal\_add\_js () ofrece dos argumentos con valores que varían en función de lo que está haciendo con él. La definición es la siguiente:

drupal\_add\_js función ($ data = NULL, $ options = NULL)

A medida que trabajamos a través del módulo Hello World vamos a examinar las diferentes variaciones y posibles valores que se pueden pasar a drupal\_add\_js ().

**Adición de archivos JavaScript**

Adición de un archivo es el comportamiento predeterminado de drupal\_add\_js () y drupal\_add\_css (). Adición de un archivo de JavaScript y un archivo CSS a una página parecida a lo siguiente:

drupal\_add\_js ('ruta / al / hello\_world.js'); drupal\_add\_css ('ruta / al / hello\_world.css');

**[289 ]**



*JavaScript en Drupal*



Esto proporciona la adición de un archivo en la forma más simple. En el caso de añadir un archivo, el primer argumento es siempre la ruta al archivo. Rutas a los archivos dentro de la instalación de Drupal son en relación con la ruta de la base de la página. Cuando estos archivos se muestran en el navegador se agregará la ruta de la base a la ruta del archivo dentro de Drupal.

Como los módulos pueden vivir en más de un lugar en el sistema de archivos, la función drupal\_get\_path () puede ser utilizado para obtener la ruta del módulo que proporciona el archivo. La reescritura de los ejemplos anteriores para que apunte a la ubicación del módulo en el sistema de archivos de forma dinámica, escribiríamos:

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); drupal\_add\_js ($ path '/ hello\_world.js'.); drupal\_add\_css (. $ path '/ hello\_world.css');



Para obtener más información sobre el uso drupal\_get\_path () con módulos, temas y otros sistemas dentro de Drupal, consulte la documentación de la API en

http://api.drupal.org/api/function/drupal\_get\_path/7.



En este sencillo ejemplo, sólo estamos de paso en el $ Data argumento, en este caso el nombre del archivo, ya que por defecto de Drupal a la adición de archivos. El segundo argumento se llama $ options y puede aceptar una cadena con el tipo de JavaScript o CSS que se añade, o un conjunto de opciones. La adición de esta JavaScript en más detalle podría volver a escribir como:

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); drupal\_add\_js ($ path '/ hello\_world.js', 'archivo'.); drupal\_add\_css ($ path '/ hello\_world.css', 'archivo'.);

Al hacer frente a los archivos, el segundo argumento se puede configurar para 'Archivo' para los archivos internos para el sistema de archivos de Drupal o en una URL ruta relativa, y "Externo" para los archivos fuera de la instalación de Drupal.

La $ options argumento se puede utilizar para establecer otras opciones para cada archivo, incluyendo peso,grupo,every\_page,alcance,aplazar,preproceso, Y el almacenamiento en caché.

Archivos JavaScript se prestan sobre la base de grupo, ya sean en cada página, y después en peso. Los grupos de JavaScript son JS\_LIBRARY,JS\_DEFAULT, Y JS\_THEME.

Dentro de cada grupo de archivos son sub-agrupados en función de si every\_page se establece en true. Scripts en cada página aparecen antes que los archivos en algunas páginas. Finalmente, dentro de cada archivo subgrupo están clasificadas por peso.

**[290 ]**



*Capítulo 10*



Cuando se utiliza una biblioteca JavaScript o plugin, hay que añadir a un grupo de JS\_LIBRARY, De modo que se agrega a la página antes de que el JavaScript que utiliza la biblioteca o plugin. Cuando se añaden dos bibliotecas de la página que son dependientes entre sí, pueden ser añadidos con el mismo grupo con una seguida de la otra en el orden en que deben ser incluidos o con dos pesos diferentes. Para ilustrar esto, el siguiente código añademylibrary.js a la página como una biblioteca antes hello\_world.js.

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); $ Options = array (

'Grupo' => JS\_LIBRARY,

);

drupal\_add\_js ($ path '/ mylibrary.js', $ options.); drupal\_add\_js ($ path '/ hello\_world.js'.);

En nuestro ejemplo, usted se dará cuenta de que podemos dejar de lado la configuración predeterminada. En este caso 'Archivo' es por defecto por lo que no necesita ser añadido a una de las llamadas.

Por JavaScript incluido en la página cada vez que hay una opción de every\_page que se debe establecer en true. Cuando se establece en true para una secuencia de comandos que impacta preprocesamiento (más sobre esto más adelante) y el orden de la secuencia de comandos se incluye. A los archivos de grupo de bandera como en todas las páginas se incluyen antes que los archivos que no están incluidos en cada página.

JavaScript dentro de un grupo y en el subgrupo de ser o no ser incluido en todas las páginas están clasificadas por peso. El peso por defecto es 0. Archivos con un peso inferior se listan antes que los archivos con un peso superior.



Bibliotecas que se basan en drupal.js tienen que ser conscientes del conjunto de peso para drupal.js. Definir la ponderación para jquery.js es - 20, Y el peso de drupal.js es - 1.



La alcance que un archivo se puede añadir a cualquiera de los dos es el 'Cabecera' o 'Pie de página'. El valor predeterminado es 'Cabecera' y coloca el JavaScript en la cabecera de la página. Los lugares más comunes para incluir el código JavaScript se encuentran en el encabezado o pie de página. Ámbitos personalizados para JavaScript esté colocado se pueden definir en un tema o módulo. En los casos en que han sido definidos ámbitos personalizados, los alcances previstos en el tema o el módulo puede ser utilizado, además de 'Cabecera' y 'Pie de página'.

Aplazar es una opción para la etiqueta script en HTML, con el apoyo de Internet Explorer. Cuenta el navegador que el guión puede aplazar la ejecución hasta después de que la página ha sido prestado. Esto es útil para los scripts que no necesitan para ejecutar o estar disponible cuando se procesa la página. En Drupal se establece en VERDADERO o FALSO.

**[291 ]**



*JavaScript en Drupal*



Las propiedades de pre-procesamiento y almacenamiento en caché van de la mano. Preprocesamiento es la característica Drupal ofrece para agregar los archivos agregados a la página en menos archivos. Preprocesamiento de archivos se basa en el grupo y si una página se incluye en cada página.

Por ejemplo, los archivos agrupados comoJS\_LIBRARY en todas las páginas se agrupan en un solo archivo preprocesado. Archivos agrupados como JS\_LIBRARY que no son en todas las páginas se agrupan en otro archivo preprocesado. Cada grupo y subgrupo de every\_page es un archivo preprocesado diferente. Esto se hace para minimizar la cantidad de JavaScript enviado al usuario y para tomar ventaja de navegador basado en el almacenamiento en caché. ¿Cuándo cache está ajustado a

FALSO archivos no se preprocesadas ya que los archivos preprocesados ​​se almacenan en caché.

Poner esto en conjunto, un archivo JavaScript establece aplazar con el almacenamiento en caché y su procesamiento previo de discapacitados, con un grupo creado para ser añadido después drupal.js se vería como la siguiente:

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); $ Options = array (

'Grupo' => JS\_LIBRARY, 'cache' => FALSE, 'preproceso' => FALSE, «retrasar '=> TRUE,

);

drupal\_add\_js ($ path '/ mylibrary.js', $ options.);

**Adición de archivos CSS**

Archivos CSS se añaden en una forma similar a los archivos JavaScript. La API para drupal\_add\_css () sólo difiere en las opciones que se pueden pasar en medio de la segunda argumento. Las opciones para los archivos CSS son peso,grupo,every\_page,medios de comunicación,

basename, navegadores, Y preproceso. Al igual que drupal\_add\_js (), 'Archivo' se utiliza para los archivos internos de Drupal o utilizando una ruta relativa a Drupal y"Externo" se utiliza para los archivos CSS externos a Drupal y tener una URL completa.

Hay tres grupos previstos como constantes con Drupal:

* CSS\_SYSTEM es para los archivos del sistema y las bibliotecas
* CSS\_DEFAULT es el módulo de archivos CSS debe utilizar
* CSS\_THEME viene después de las otras opciones y se utiliza para el tema de CSS

CSS tiene diferentes medios de comunicación que ha aplicado. Por ejemplo, las hojas de estilo con un medio de comunicación de la pantalla sólo se aplican cuando la página se representa para las pantallas. Otros casos, como la impresión, ignoran la hoja de estilo. El valor predeterminado es"Todos".

**[292 ]**



*Capítulo 10*



El código siguiente agrega un archivo CSS sistema que no está pre-procesado y sólo se utiliza para la pantalla:

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); $ Options = array (

'Grupo' => CSS\_SYSTEM, 'medios' => 'pantalla', 'preproceso' => FALSE,

);

drupal\_add\_css (. $ path '/ hello\_world.css', $ opciones);

**Al pasar las variables de PHP a JavaScript**

Drupal proporciona un medio para pasar variables de PHP a JavaScript usando drupal\_add\_js (). Muchas aplicaciones quieren transmitir la información de configuración tener JavaScript que se ejecuta en una página. Esta función es el medio en Drupal para pasar esa información fácilmente.

Drupal llama a las variables pasadas de PHP para la configuración de JavaScript. Un ejemplo simple de un entorno que pasa el texto "Hello World!" desde PHP a JavaScript haría

aspecto:

drupal\_add\_js (array ('helloWorld' => "Hello World!"), 'configuración');



Los nombres de variables en la parte de PHP de Drupal son en minúsculas con un guión que separa las palabras. En JavaScript deben estar en lowerCamelCase. Para obtener más información, consulte el estándar de codificación en



http://drupal.org/coding-standards.

JavaScript puede acceder a este aDrupal.settings.helloWorld. Por ejemplo, el siguiente JavaScript mostraría el "Hola Mundo" como un pop-up:

alerta (Drupal.settings.helloWorld);

Los ajustes son a diferencia de otras aplicaciones de drupal\_add\_js (). Se agregan a la página en la cabecera con un peso de JS\_LIBRARY. No hay otras opciones además de especificar que se trata de un ajuste, como se muestra en el siguiente ejemplo:

drupal\_add\_js (array ('hello\_world' => "Hello World!"), 'configuración');

**[293 ]**



*JavaScript en Drupal*



Ajustes deben añadirse en una manera que respete la namespacing de otros ajustes agregados a la página. Pasando múltiples ajustes deben hacerse en una matriz anidada. Por ejemplo:

$ settings = array ('helloWorld' => array (

'Display' =>, 'message' 'alerta' => 'Hello World!',

),

);

drupal\_add\_js (ajustes $, 'Marco');

En este ejemplo, el mensaje está disponible en Drupal.settings.helloWorld.message. Mantener todos los ajustes en Drupal.settings.helloWorld, Mantiene la configuración de este módulo separado de los ajustes añadidos por otros módulos.

**Adición inline JavaScript**

JavaScript se pueden agregar en línea en la página utilizando elinline opción. Un ejemplo de que las alertas de "Hello World!" se vería así:

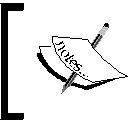
drupal\_add\_js ('alert ("Hello World!");', 'inline');

Las opciones accesibles a en línea con JavaScript son aplazar,grupo,every\_page,peso, Y alcance. Inline JavaScript no se almacena en caché por el navegador y no puede ser preprocesado. JavaScript Inline que se agrega, que aplaza hasta después del navegador ha cargado con un peso de JS\_THEME se vería así:

$ options = array ('type' => 'inline', 'grupo' => JS\_THEME, «retrasar '=> TRUE,

);

drupal\_add\_js ("alert ('Hello World!')", $ opciones);



Documentación de la API para drupal\_add\_js () está disponible en

http://api.drupal.org/api/function/drupal\_add\_js/7

**Adición de CSS en línea**

CSS también puede ser añadido en línea y la API es similar a la de JavaScript. Para añadir CSS en línea que se vería así:

drupal\_add\_css ("body {color: # ffffff;}", 'inline');

**[294 ]**



*Capítulo 10*



El segundo argumento puede contener una gran variedad de opciones para su uso incluyendo el grupo, el alcance y preproceso. Una vez que se añade CSS, con un peso de CSS\_THEME establece en no preproceso, que se verá como:

$ Options = array ('type' => 'inline',

'Grupo' => CSS\_THEME, 'preproceso' => FALSE,

);

drupal\_add\_css ("body {color: # ffffff;}", $ opciones);



Documentación de la API para drupal\_add\_css () está disponible en

http://api.drupal.org/api/function/drupal\_add\_css/7

**Uso de la API Library**

Drupal 7 proporciona una API Biblioteca donde las bibliotecas y los plugins de JavaScript y CSS pueden definirse como programáticamente añadir más tarde. jQuery y el resto de las bibliotecas proporcionadas con Drupal son definidos mediante la aplicación de los módulos del sistema hook\_library (). Podemos utilizar farbtastic, un selector de color de jQuery incluye con Drupal, como un ejemplo. Se incluyen en farbtastic es un archivo de JavaScript y un archivo CSS. Para agregar farbtastic a una página que se vería de la siguiente manera:

drupal\_add\_library ('sistema', 'farbtastic');

Esto no sólo aumentará JavaScript y CSS, pero también añadirá las bibliotecas dependientes. Un ejemplo de esto puede verse en el módulo de superposición dondedrupal\_add\_library () añade la superposición. Se hace un llamado para agregar la superposición de los padres de JavaScript que se ve

de esta manera:

drupal\_add\_library ('overlay', 'padre');

En la función overlay\_library (), padre va a tener jQuery barbacoa y jQuery UI núcleo como dependencias a los padres de superposición. Esto significa esos dos bibliotecas

se añadirá a la página antes de la superposición. Drupal sabe la cadena de las dependencias de la biblioteca, por lo que no tiene que hacerlo.



Documentación de la API sobre drupal\_add\_library () está disponible en http://api.drupal.org/api/function/drupal\_add\_library/7.

**[295 ]**



*JavaScript en Drupal*



**Definición de una biblioteca con hook\_library**

Cuando un módulo tiene una biblioteca o un plug-in que quiere utilizar o poner a disposición de otras bibliotecas, debe definirlo como una biblioteca utilizando hook\_library (). Ya que tenemos un archivo JavaScript y archivo CSS en nuestro módulo Hello World, podemos agregarlo como una biblioteca de la siguiente manera:

/ \*\*

\* Implementa hook\_library (). \* /

función hello\_world\_library () {

$ Path = drupal\_get\_path ('módulo', 'hello\_world'); $ bibliotecas = array (); $ bibliotecas ['hello\_world\_library'] = array (

'Title' => 'Hello World', 'website' => 'http://example.com', 'versión' 0.0 => '1 ',

'Js' => array (

$ Ruta. '/ hello\_world.js' => array (),

),

'Css' => array (

$ Ruta. '/ Hello\_world.css' => array (),

),

'dependencias' => array (

array ('sistema', 'ui.dialog'),

),

);

return $ bibliotecas;

}

La título,sitio web, Y versión propiedades se utilizan para definir los metadatos acerca de las bibliotecas. Esto es importante en la búsqueda de información, documentación y comprobación de actualizaciones a una biblioteca.

La js,css, Y dependencias hacer todo el trabajo. Si se definen las dependencias, se añaden antes de que el JavaScript y CSS definidos aquí. A continuación, el JavaScript y CSS se añaden con la clave para cada línea de ser el primer argumento a favor de cualquiera de los dos

drupal\_add\_js () o drupal\_add\_css () y el valor siendo el Opciones argumento para cada una de las funciones correspondientes.

Drupal tiene tres dependencias especiales que se agregan, que no necesita ser definida. Ellos son jquery.js,jquery.once.js, Y drupal.js. Estos se añaden a la página cuando la primera llamada a drupal\_add\_js () se hace o cuando drupal\_add\_library () primero se llama.

**[296 ]**



*Capítulo 10*



De aquí en adelante, cuando queremos utilizar la biblioteca hola mundo, nos gustaría añadir que con la siguiente llamada:

drupal\_add\_library ('hello\_world', 'hello\_world\_library');

El primer argumento es el módulo que define la biblioteca y el segundo argumento es la clave para la biblioteca definida.



Documentación de la API sobre hook\_library () está disponible en

http://api.drupal.org/api/function/hook\_library/7.

**La alteración de la información en hook\_library**

Drupal proporciona un hook\_library\_alter () función, donde los módulos pueden interceptar las bibliotecas definidas por hook\_library () y, o bien actuar o realizar cambios en ellos. Un ejemplo sencillo podría ser otro módulo que proporciona una versión más reciente

de los hello\_world.js guión. Para el ejemplo se llama el módulo Hola

Mundial actualización. En el archivo hello\_world\_update.module tenemos la siguiente aplicación de hook\_library\_alter ():

/ \*\*

\* Implementa hook\_library\_alter (). \* /

función hello\_world\_library\_alter (& $ bibliotecas, $ module) {if ($ module == 'hello\_world' &&

isset ($ bibliotecas ['hello\_world\_library'])) {

* Verifique la versión actual es más antigua que la que estamos
* actualizar a.

if (($ version\_compare bibliotecas ['hello\_world\_library'] ['versión'], '2 .0 ',' <')) {

/ / Actualizar el Hello World existente a la versión 2.0. $ bibliotecas ['hello\_world\_library'] ['versión'] = 0,0 '2 '; $ bibliotecas ['hello\_world\_library'] ['js'] = array (

drupal\_get\_path ('hello\_world\_update' "módulo"). '/ world\_2.0.js Hello\_' => array (),

);

}

}

}

Los dos argumentos pasados ​​son las bibliotecas definidas por un módulo y el nombre del módulo. En este caso, comprobamos para ver si la versión ya definida es más antigua que la versión proporcionada por este módulo. De ser así, se sustituye la llamada de JavaScript con una diferente.

**[297 ]**



*JavaScript en Drupal*



Documentación de la API para hook\_library\_alter () está disponible en http://api.drupal.org/api/function/hook\_library\_ alter / 7



**Utilización de matrices renderizable**

Una matriz Drupal renderizable es la manera gran parte de la salida de una página de Drupal es representado antes de que se procesa en HTML. Funciones temáticas, se discutió en el Capítulo 3, Tema Capa de Drupal, pueden devolver un array Drupal renderizable o HTML para la producción. Si se devuelve un array renderizable será generado en HTML drupal\_render (). Si está familiarizado con el Formulario de API de versiones anteriores de Drupal que has visto la renderizable Arrays. Ellos son las matrices de la Forma API extrapolados a otros usos. La matriz renderizable para un elemento de formulario podría tener este aspecto:

$ Form ['opciones'] = array ('# type' => 'campo de texto',

'# Title' => t ('Nombre del autor'), '# maxlength' => 25, '# adjunto' => array (

'Css' => array (

drupal\_get\_path ('hello\_world' "módulo"). '/ Example.css',

),

'Js' => array (

"Alert ('Hello World!')" => Array ('type' => 'inline'),

),

);

La # Adjunta propiedad en una matriz renderizable o formulario se puede utilizar para añadir JavaScript, CSS, y las bibliotecas. Las teclas son las siguientes:

* js es JavaScript
* css se utiliza para la CSS
* biblioteca se usa para añadir bibliotecas

Para cada elemento de una matriz de secciones, como js, La clave está en los datos y las opciones son el valor. Si el valor se omite, se supone que las opciones por defecto.

Utilización de matrices renderizable y adjuntando el JavaScript, CSS, y las bibliotecas es importante en varios lugares a través de Drupal donde el almacenamiento en caché de los elementos individuales suceder. Un ejemplo es con bloques que pueden ser almacenados en la caché. Un caso de uso común es cuando se añade a un bloque de JavaScript. Si el código JavaScript se añade dentro del contenido del bloque usingdrupal\_add\_js () que se añadirá cuando el bloque no se almacena en caché. Bloques en caché no reconstruir el contenido para drupal\_add\_js () no se llamará.

**[298 ]**



*Capítulo 10*



Si una matriz de renderizable se utiliza en lugar con JavaScript unido, la matriz renderizable es almacenada en memoria caché que contiene las llamadas a la JavaScript. Cuando el contenido en caché, está renderizado y JavaScript, CSS, entonces las bibliotecas incluidas en la matriz renderizable se añadirá a la página.

Un ejemplo que se suma JavaScript y CSS a la salida de un bloque se vería como la siguiente:

$ Salida ['content'] = array (

"Valor # '=>' El contenido del bloque. ',' # Adjunto '=> array (

'Css' => array (

drupal\_get\_path ('hello\_world' "módulo"). '/ Example.css',

),

'Js' => array (

"Alert ('Hello World!')" => Array ('type' => 'inline'),

),

),

);

**La alteración de JavaScript**

JavaScript agregado a una página tiene una última oportunidad de ser alterada antes de ser prestado a la página de salida. Justo antes de que el JavaScript se representa en HTML que se pasa a través hook\_js\_alter (). Un módulo que implementa hook\_js\_alter () tiene una última oportunidad para actuar en o cambiar el JavaScript.

Un ejemplo de esto sería si un módulo quiere intercambiar la versión comprimida de jQuery con una versión sin comprimir. Esto sería útil para propósitos de depuración. En el ejemplo, vamos a llamar el módulo de jQuery comprimir. En el archivo jquery\_uncompressed.modulehook\_js\_alter () se vería como la siguiente:

/ \*\*

\* Implementa hook\_js\_alter (). \* /

jquery\_uncompressed\_js\_alter función (& $ javascript) {$ path = drupal\_get\_path ('módulo', 'jquery\_uncompressed') $ javascript ['otros servicios / jquery.js'] ['datos'] = $ PATH. '/ JQuery.

uncompressed.js ';

}

La $ Javascript matriz pasada contiene todo el código JavaScript que se añade a la página actual. Este hook se llama justo antes de JavaScript se vuelve a añadir a la página. Este es el último punto de alterarlo.

**[299 ]**



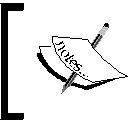
*JavaScript en Drupal*



Las claves de la matriz, para encontrar lo que estás buscando, dependen del tipo de

JavaScript que se añadió.

* Archivo y JavaScript externo se identifica por la ruta del archivo que se pasó por su ubicación.
* Ajustes están disponibles en $ Javascript ['settings'].
* Inline JavaScript no es fácilmente identificable. Cada vez que un script de línea se agrega a una página que se agrega con un valor numérico a partir de 0. Si se agregaron dos scripts de línea de una página, que se añadirían a las$ Javascript [0] y $ Javascript [1]. Las cifras se basan en el orden en que fueron añadido a una página, que es poco fiable.



Documentación de la API para hook\_js\_alter () está disponible en

http://api.drupal.org/api/function/hook\_js\_alter/7

**La alteración de CSS**

CSS puede ser alterada de una manera similar a JavaScript utilizando hook\_css\_alter (). Una matriz que contiene toda la CSS a punto de ser introducido en la página se pasó a través de hook\_css\_alter () como una última oportunidad para hacer cambios antes de que el CSS se hace en HTML y se agrega a la página. Un ejemplo de uso de lahook\_css\_alter () sería para eliminar la system.css archivo proporcionado por el núcleo, como se ve aquí:

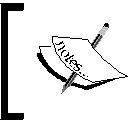
/ \*\*

\* Implementa hook\_css\_alter (). \* /

función example\_css\_alter (& $ css) {unset ($ css [drupal\_get\_path (, "sistema" "módulo") '/ system.css'.]);

}

Identificar los elementos de la matriz pasada en los archivos CSS es similar a los archivos JavaScript. La clave para los archivos internos y externos es la ruta de acceso al archivo que se utiliza cuando se agregó originalmente. Al igual que en línea con JavaScript, se añade CSS en línea utilizando las teclas numéricas en el orden se añadió el CSS.



Documentación de la API para hook\_css\_alter () está disponible en

http://api.drupal.org/api/function/hook\_css\_alter/7

**[300 ]**



*Capítulo 10*



**Específica Drupal JavaScript**

Dentro de Drupal existen numerosas funciones de ayuda. Estos van desde sistemas como la tematización y la traducción de las funciones de utilidad que analizan JSON.

**Presentación temable**

Toda la presentación dentro de Drupal es personalizable a través del sistema de tema y JavaScript no es diferente. Drupal proporciona un sistema para la tematización de la presentación generada por el código JavaScript que se puede reemplazar dentro de JavaScript en un tema. Podemos empezar con un ejemplo,hello\_world.js, Que se suma por un módulo que tiene este aspecto:

(Function ($) {$ (). Ready (function () {

. $ ('# Hola mundo') html ('<h2> Hello World </ h2>'); });

}) (JQuery);

Cuando se carga la página, el HTML dentro del div con el ID de hola-mundo será reemplazado con <h2> Hola mundo! </ h2>. ¿Qué pasa si un tema quiere que el envoltorio sea un h3 etiqueta en lugar de una h2 etiquetar? Aquí es donde la tematización Drupal JavaScript entra en juego

Un módulo debe proporcionar una función de tematización dentro de la Drupal.theme.prototype espacio de nombres. Entonces, el módulo llamaría Drupal.theme () para acceder a la función de tema. Este sistema permite que un tema para proporcionar una función de anulación dentro del Drupal.theme espacio de nombres. El siguiente ejemplo ilustra esto.

(Function ($) {

Drupal.theme.prototype.hello = function (texto) {return '<h2>' + texto + '</ h2>';

}

.. $ () Ready (function () {$ ('# hola mundo') html (Drupal.theme ('hola',)) 'Hello World!';

});

}) (JQuery);

**[301 ]**



*JavaScript en Drupal*



Drupal.theme () llamará al Drupal.theme.prototype.hello () función y pase en todos los argumentos, sino la primera. Esto es útil porque el tema puede proporcionar una función de anulación. Si un módulo proporciona un hello\_world.js presentar la declaración como la de arriba, y un tema proporciona un archivo JavaScript con la siguiente función, se cambiará la salida.

(Function ($) {

Drupal.theme.hello = function (texto) {return '<h3>' + texto + "</ h3> ';

}

}) (JQuery);

Drupal.theme () llamará Drupal.theme.hello () en lugar de Drupal.theme. prototype.hello (). El tema puede anular la presentación en JavaScript de esta manera.

**Cadenas traducibles**

Todo el texto dentro de la interfaz de Drupal es traducible. El texto dentro de JavaScript se puede traducir también. Dentro del código PHP para Drupal lat () función se utiliza para las traducciones. Dentro de JavaScript del Drupal.t () función se utiliza para manejar las traducciones.

Continuando con el módulo Hello World podemos extenderlo a manejar, decir "hola" a diferentes ciudades, donde el "hola" parte es traducible.

(Function ($) {

Drupal.theme.prototype.hello = function (texto) {

return '<h2>' + Drupal.t ('Hola @ de la ciudad', {'@ ciudad: text}) +' </ h2> ';

}

.. $ () Ready (function () {$ ('# hola mundo') html (Drupal.theme ('hola', 'Chicago'));

});

}) (JQuery);

La parte que nos interesa es donde dice:

Drupal.t ('Hola @ de la ciudad', {'@ ciudad: text})

Aquí la cadena 'Hola @ ciudad' es una cadena traducible. La @ Ciudad parte es dinámico y se rellenará por el valor de texto. Esta configuración permite traducciones

modificar la cadena y todavía pasar las partes dinámicas a través.

**[302 ]**



*Capítulo 10*



Hay tres formularios de variables se pueden pasar a través de Drupal.t () dependiendo el primer carácter de la variable:

* Las variables a partir de una ! se insertan como es, sin ninguna modificación.
* Cuando una variable comienza con una @ símbolo del valor se pasa a través Drupal.checkPlain (), Una función que convierte la cadena en texto plano. Marcado HTML se convierte en texto que se puede mostrar.
* %en el comienzo de una variable hará que la variable que se va a pasar a través de

Drupal.checkPlain () y Drupal.theme ('marcador').

**Comportamientos**

Si está familiarizado con la escritura de jQuery, ya sabes que el código que desee ejecutar lo antes posible en una carga de la página está envuelto en algo como lo siguiente:

$ (Document). Ready (function () {

...

});

Cuando el documento está listo se ejecutará el código envuelto dentro de esta función. Este es un patrón común de JavaScript basado en jQuery. Drupal ofrece un sistema que se envuelve y se extiende este concepto denominado comportamientos. Comportamientos están unidos y separados de una sección de contenido. El uso más común es para adjuntar un comportamiento a una página entera.

Reescribiendo el ejemplo Hello World visto previamente utilizar comportamientos se vería de la siguiente manera:

(Function ($) {

Drupal.theme.prototype.hello = function (texto) {

return '<h2>' + Drupal.t ('Hola @ de la ciudad', {'@ ciudad: text}) +' </ h2> ';

}

Drupal.behaviors.helloWorld = {adjuntar: función (contexto, ajustes) {

. $ ('# Hola-mundo', contexto) html (Drupal.theme ('hola', 'Chicago'));

}

}

}) (JQuery);

**[303 ]**

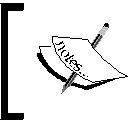


*JavaScript en Drupal*



El comportamiento es en Drupal.behaviors.helloWorld y tiene una adjuntar funcionar para fijar el comportamiento de la página. Cuando el documento esté listo, Drupal llama todo el adjuntar funciones para todos los comportamientos. Pasa en el contexto, el documento, y el ajustes, que en la página de carga es el valor de Drupal.settings.

En la superficie esto parece un método poco intuitivo en comparación con el patrón más común usado en jQuery. El patrón de comportamientos ofrece mucho más. Por ejemplo, cuando se añade contenido recuperado a través de AJAX a la página, se pasa a través de laadjuntar funciones para todos los comportamientos. La contexto se agrega el contenido que se obtuvo a partir de la solicitud y el AJAX ajustes es o bien la configuración que se devolvieron a la petición AJAX o los ajustes en Drupal.settings.



Para más información general sobre el contexto ver

http://api.jquery.com/jQuery/.

Junto con la capacidad de adjuntar comportamientos es la capacidad de separar un comportamiento. Una estructura típica para un comportamiento es:

(Function ($) {Drupal.behaviors.example = {

adjuntar: función (contexto, ajustes) {

...

}

separar: function (contexto, la configuración, el gatillo) {

...

}

}

}) (JQuery);

Por ejemplo, cuando un JavaScript basado en AJAX elimina el contenido de una sección de la página antes de agregar nuevo contenido a la sección, el despegar funciones para los comportamientos se llaman en el contenido que se está eliminado. Esto proporciona una oportunidad para que los comportamientos que actúan sobre este contenido por última vez.

Un ejemplo de esto se inicia por el sistema de Drupal AJAX. Antes de enviar un formulario, a través de AJAX a Drupal, despegar se pidió a todos los comportamientos. Esto proporciona una oportunidad para que los comportamientos que hacen que los cambios en la forma, que no deben ser enviados a Drupal,

que ser eliminado. Cuando el nuevo contenido se pone en la página de los comportamientos se unen de nuevo.

**[304 ]**



*Capítulo 10*



**Helpers de AJAX**

Incluido en Drupal es una biblioteca de AJAX que integra Drupal, jQuery y AJAX. Este sistema proporciona un conjunto de propiedades de formularios y funciones, para usar en las devoluciones de llamada de Drupal, que rápidamente y fácilmente construir AJAX en módulos de Drupal. Uso de las funciones de ayuda y propiedades AJAX se pueden incorporar a las páginas de Drupal más que atornilla en las páginas.

**Agregar AJAX a las formularios**

Un uso común de AJAX está actualizando dinámicamente formularios, a partir de la entrada, a otras partes del formulario. Cuando se actualiza un elemento en forma, otros elementos cambian o se rellenan con la información sobre la base de un cambio. Por ejemplo, podemos mirar a un elemento de formulario lista de opciones que se actualiza un elemento de marcado.

Comenzamos con una implementación de hook\_menu () para definir el formulario, como se ve aquí:

/ \*\*

\* Implementa hook\_menu (). \* /

función hello\_world\_menu () {$ items = array ();

$ items ['hello\_world / simple\_form\_example'] = array ('title' => 'Hello World: Simple AJAX Ejemplo', 'Página callback' => 'drupal\_get\_form',

La página de argumentos '=> array (' hello\_world\_simple\_form\_example '),' callback acceso '=> TRUE,

);

return $ artículos;

}

Seguimos esto mediante la creación de una devolución de llamada formulario llamado hello\_world\_simple\_form\_ ejemplo (). Esto crea el formulario para insertar en la página.

hello\_world\_simple\_form\_example función ($ form, & $ form\_state) {$ form = array ();

$ Form ['hello\_city'] = array ('# title' => t ("Elige una ciudad"), "tipo # '=>' select ',' # options '=> array (

t ('Mundo') => t ('Mundo'), t ('Chicago') => t ('Chicago'), t ('Nueva York') => t ('Nueva York'),

**[305 ]**



*JavaScript en Drupal*



t ('Los Angelas') => t ('Los Angelas'),

),

'# Ajax' => array (

'Callback' => 'hello\_world\_simple\_form\_callback', 'envoltorio' => 'ajax\_markup\_div',

),

);

$ Form ['ajax\_markup'] = array (

'# Prefijo' => '<div id="ajax\_markup\_div">', '# sufijo' => '</ div>',

'# Markup' => t ('Hola Mundo'),

);

if (empty ($ form\_state ['valores'] ['hello\_city'])!) {$ form ['ajax\_markup'] ['# markup'] = t ("¡Hola! la ciudad", array ('! ciudad'

=> $ Form\_state ['valores'] ['hello\_city']));

}

devolver $ form;

}

Aquí tenemos dos elementos de formulario de hello\_city y ajax\_markup. La hello\_city elemento tiene un # Ajax propiedad, que es nuevo en Drupal 7. Define una devolución de llamada, que es una función de devolución de llamada interna Drupal, y una envoltura. Este es el envoltorio en la página que se actualizará por la respuesta a la petición AJAX.

El segundo elemento, llamadoajax\_markup, Es un elemento de formulario marcado. Este elemento contiene código HTML. Inicialmente se poblamos el marcado con 'Hello World'. Para este elemento se actualice tenemos que añadir un envoltorio a la misma. En este caso, la envoltura es una div con el mismo ID que nos propusimos como el envoltorio en el # Ajax parte de la hello\_city elemento.

Después de que el formulario se establece allí es un si declaración para el caso en que un valor se encuentra disponible en $ Form\_state para la hello\_city elemento. Cuando se crea inicialmente la forma, no habrá un valor. Cuando se hace la petición AJAX, se pasa a través de este formulario con los valores del formulario. Cuando esto ocurre, el si declaración se ejecuta haciendo que el ajax\_markup elemento que se debe actualizar.

Esto será seguido por la ejecución de la devolución de llamada definido por el # Ajax propiedad en el hello\_world elemento. Esa función se parece a esto:

función hello\_world\_simple\_form\_callback ($ form, $ form\_state) {return $ formulario ['ajax\_markup'];

}

**[306 ]**



*Capítulo 10*



La función de devolución de llamada hello\_world\_simple\_form\_callback () pasa el $ Form y $ Form\_state las variables después de que han pasado por simple\_form\_example hello\_world\_ (). En este caso nos vamos a volver al elemento de formulario que es siendo reemplazado.

Drupal sabe que esta es una matriz renderizable y lo hace en el valor adecuado. Drupal envía el código HTML actualizada en la página donde los manipuladores de Drupal AJAX recuperar los cambios y vuelva a colocar la envoltura.

**AJAX aplica automáticamente**

AJAX se puede aplicar automáticamente a los elementos de una página. Esto se hace mediante la aplicación de lautilizar ajax- clase a un elemento de una página. Un uso típico sería aplicar la utilizar ajax- clase a un enlace dentro de la página para activar una acción AJAX. Los enlaces son de uso común porque la página de puntos del enlace a podría ser para los casos en los que JavaScript está deshabilitado por si falla.

En el siguiente ejemplo vamos a proporcionar un enlace que, al hacer clic, se sumará "Hola Mundo" a un div dentro de la página. Para empezar, tenemos dos devoluciones de llamada de menú que nos agregamos a hello\_world\_menu (). Un artículo de la página que estamos generando y la otra es la dirección URL de devolución de llamada utilizado para AJAX o cuando javascript está deshabilitado.

$ items ['hello\_world / link'] = array ('title' => 'Hello World: Link',

La página de callback '=>' hello\_world\_link ',' acceso callback '=>' user\_access ',

los argumentos de acceso '=> array (' content access '),

);

$ items ['hello\_world\_link\_callback'] = array ('página callback' => 'hello\_world\_link\_response', 'callback acceso' => 'user\_access',

los argumentos de acceso '=> array (' content access '),

);

El primer elemento del menú es a nuestra página donde se encuentra el enlace para que el AJAX agregará contenido. El segundo elemento es el de devolución de llamada que se encargará de la solicitud AJAX o la solicitud de página, cuando JavaScript no está disponible.

función hello\_world\_link () {drupal\_add\_js ("otros servicios / ajax.js ');

$ Link = l (t ('Say Hello'), 'hello\_world\_link\_callback / Nojs /', array ('atributos' => array ('class' => array ('uso-ajax'))));

return '<div>'. $ Enlace. '</ Div> <div id="saying-hello"> </ div>';

}

**[307 ]**



*JavaScript en Drupal*



La devolución de llamada página en el enlace vive, se inicia mediante el uso de drupal\_add\_js () añadir otros servicios / ajax.js. Esta JavaScript hace el trabajo para hacer automáticamente el AJAX trabajo. Esto es seguido por un enlace con una devolución de llamada a hello\_world\_link\_callback / Nojs /. La primera parte de este enlace es la devolución de llamada, donde se maneja la petición AJAX.

La / Nojs / al final es especial. Cuando JavaScript no está disponible, se pasa a la función de respuesta por lo que sabe que fue llamado a ser una carga de página completa. Cuando se dispone de JavaScript se reemplaza con / Ajax /. Esto se pasa a la función de devolución de llamada para que sepa que fue llamado a través de una petición AJAX.

Lo que hace que este vínculo convertido en un eslabón de AJAX es la clase que se añade con el nombre utilizar ajax-. El archivo JavaScript que añadir al principio, ajax.js, Busca enlaces con esta clase y las convierte en AJAX.

función hello\_world\_link\_response ($ type = 'ajax') {if ($ tipo == 'ajax') {

$ = salida t ("Hello World!"); comandos $ = array ();

comandos $ [] = ajax\_command\_append ('# diciendo-hola ", $ salida); $ Page = array ('tipo #' => 'ajax\_commands', '# ajax\_commands' =>

comandos $); ajax\_deliver ($ page);

}

else {

volver t ("Hola Mundo en una nueva página.");

}

}

Cuando la devolución de llamada para atender la solicitud se le llama, el tipo de acción se pasa pulg Cuando no se dispone de JavaScript, Nojs se pasa pulg Cuando la solicitud es AJAX, ajax se pasa pulg Esto se puede utilizar en la lógica funciones de devolución de llamada para responder adecuadamente a cada caso.

En este caso utilizamos un si comunicado. Cuando la solicitud es AJAX, respondemos de una manera y cuando JavaScript no está disponible respondemos de manera diferente.

Cuando la solicitud es una devolución de llamada AJAX empezamos creando el texto de la respuesta de "Hello World!". Esto es seguido por la creación de una matriz para que contenga los comandos que queremos Drupal

para ejecutar y agregar un comando a él.

Un comando es una acción que queremos que el JavaScript para realizar cuando se recibe la respuesta AJAX. Los comandos proporcionan formularios en que jQuery se puede manipular el contenido de la página con los datos respondido. En este caso el comando utilizado es el ajax\_command\_append (). Este comando acepta un selector de jQuery y contenido. El contenido se agrega al selector. Esta función utiliza la Drupal jQuery.append () función.

**[308 ]**



*Capítulo 10*



Una vez que la respuesta se configura que se inserta en una matriz de renderizable. El tipo es ajax\_commands, Que saben cómo hacer el comando AJAX que se creó en la devolución de llamada. Para enviar la respuesta AJAX correctamente ajax\_deliver () se utiliza. Esta función formatea correctamente la respuesta para el JavaScript en el lado de recepción.

Además, Drupal seguimiento de los archivos JavaScript y CSS dentro de una página. Si un nuevo archivo se agrega en una petición AJAX que aún no está cargado en la página del nuevo archivo se envía como parte de la respuesta y se agrega a la página junto con el resto de

la respuesta.

Para casos en los que JavaScript no está disponible en la vista inicial de página y el enlace es seguido, se trata como una solicitud de página completa y se envía al usuario a una nueva página. Esta página vive en la misma callback que construyó la respuesta AJAX. La diferencia es Nojs se pasa a la devolución de llamada para que sepa la respuesta no está en AJAX. En este caso, la más se ejecuta la generación de un mensaje diferente para la nueva página.

**Comandos AJAX**

Drupal ofrece varios comandos de AJAX que se pueden añadir o modificar el contenido de una página utilizando métodos de jQuery. En el apartado anterior hemos cubierto ajax\_command\_append (). Aquí están todos los posibles comandos que se pueden utilizar.

**ajax\_command\_after**

¿Cuándo ajax\_command\_after () se utiliza, jQuery.after () es el método utilizado para añadir contenido a la página. Los argumentos son $ Selector,$ Content, Y $ ajustes.$ Selector es el selector de jQuery en la página y $ Content es el contenido de agregar después de que el selector. $ ajustes, El tercer argumento, es un conjunto de configuraciones utilizadas por los comportamientos para este único comando.

**ajax\_command\_alert**

El comando de alerta es un comando del núcleo de JavaScript. ajax\_command\_alert ($ texto) se utiliza para alertar texto devuelto. El texto se presenta en un cuadro de alerta.

**ajax\_command\_append**

Este comando es similar al ajax\_command\_after. En lugar de añadir el contenido después de que el selector de que lo anexa al final del selector. La interfaz para la función es la misma que ajax\_command\_after () con el $ Selector,$ Content, Y$ ajustes argumentos. $ Selector es el selector de jQuery en la página, $ Content es el contenido que se anexará al selector, y $ ajustes es utilizado por el comportamiento de sólo este comando.

**[309 ]**



*JavaScript en Drupal*



**ajax\_command\_before**

Para agregar contenido antes de un uso elemento ajax\_command\_before (). Utiliza jQuery.before () agregar contenido antes de un selector. Una vez más la $ Selector, $ Content, y $ ajustes se utilizan argumentos.

**ajax\_command\_changed**

Tomar nota de que algo dentro de una página ha cambiado ajax\_command\_ cambiado ($ selector, $ asterisco) puede ser utilizado. Elementos que se encuentran en la página con la selector jQuery determinado tendrá la cambiado-ajax clase se les aplica. $ Asterisco es un selector CSS opcional, que reside en el interior $ Selector. Esto se utiliza para añadir opcionalmente un asterisco a ese elemento.

**ajax\_command\_css**

ajax\_command\_css () utiliza el jQuery.css () comando para actualizar la CSS dentro la página. Este comando toma en $ Selector y $ Argumento argumentos. Por ejemplo, cambiar el color de fondo de una página se vería de la siguiente manera:

comandos $ [] = ajax\_command\_css ('cuerpo', array ('background-color' => '# FFFFFF'));

**ajax\_command\_data**

jQuery proporciona un comando 'Datos' para almacenar datos en una página fuera de atributos de los elementos. La ajax\_command\_data () función permite que Drupal AJAX para agregar y datos dentro de la memoria caché de datos jQuerys actualización. Los tres argumentos son:

* $ Selector, El selector de elementos jQuery
* $ Name, El nombre que se accede al elemento de datos
* $ Value, El valor del elemento

**ajax\_command\_html**

ajax\_command\_html () utiliza jQuery.html () para actualizar el HTML para un selector dado. Los argumentos son $ Selector (El selector de jQuery), $ Html (El código HTML para actualizar el selector de usar), y $ ajustes (configuración opcional para este comando a utilizar).

**[310 ]**



*Capítulo 10*



**ajax\_command\_prepend**

Para agregar contenido al principio de un elemento de uso de la prepend comando. Este utiliza jQuery.prepend () para agregar el contenido. Los argumentos a favor

ajax\_command\_prepend () son $ Selector, $ Content, Y $ ajustes.

**ajax\_command\_remove**

Este comando remueve los elementos de una página. El único argumento es el selector para ser eliminado. jQuery.remove () se utiliza para eliminar los elementos de la página.

**ajax\_command\_replace**

El ajax\_command\_html reemplaza el contenido HTML en un elemento. Para casos en los que todo el elemento deba ser sustituido ajax\_command\_replace () debe ser utilizado. Se aprovecha de jQuery.replaceWith () reemplazar todo el elemento. Los tres argumentos son $ Selector,$ Html, Y $ ajustes.$ Html es el HTML completo el selector será reemplazado por. Por ejemplo, tomemos el html:

<div class="container">

<div class="inner"> Hola mundo! </ div> </ div>

Un ajax\_command\_replace () se parece a lo siguiente:

comandos $ [] = ajax\_command\_replace ('<h2> Adiós mundo </ h2>!' 'interior.';

Esto actualizará el código HTML para parecerse a la siguiente:

<div class="container"> <h2> Adiós mundo! </ h2>

</ Div>

**ajax\_command\_restripe**

El comando redefinir dice Drupal para redefinir una mesa. Esto es útil cuando el contenido dentro de la tabla ha sido alterado. El único argumento es el selector de jQuery para la tabla.

**[311 ]**

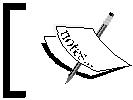


*JavaScript en Drupal*



**ajax\_command\_settings**

ajax\_command\_settings () se usa para añadir ajustes a la respuesta. La primera argumento es la configuración que se envían con la respuesta. Si sólo se da el primer argumento, o el segundo argumento es FALSO, Entonces el ajuste sólo será utilizado para la respuesta. Si el segundo argumento, el$ Merge argumento, se establece en VERDADERO los ajustes se fusionarán en Drupal.settings.



Para obtener más información sobre la visita jQuery API

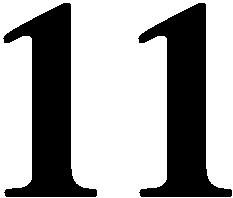
http://api.jquery.com.

**Resumen**

Hemos cubierto los principios y los comandos básicos del uso de JavaScript y jQuery en el contexto de Drupal. Empezamos añadiendo JavaScript a una página como un archivo, en línea, y como escenario. Continuamos observando añadiendo bibliotecas completas con dependencias y alterando la derecha JavaScript antes de que se represente la página.

Drupal proporciona funciones auxiliares y bibliotecas que son útiles en la creación de módulos de Drupal. Cubrimos cómo estas bibliotecas y algunos de los elementos más comúnmente usados ​​funcionan.

**[312 ]**



Trabajar con archivos e imágenes

Drupal 7 introduce una nueva API para los archivos y las imágenes, con lo que la funcionalidad de los módulos aportados populares como Imagecache y ImageField en el núcleo por primera vez. En este capítulo, vamos a construir dos módulos que se aprovechan de esta nueva funcionalidad. Algunos de los conceptos que cubriremos son:

* La comprensión de Drupal público y privado sistemas de archivos
* Asociar archivos e imágenes con contenido
* La implementación de una envoltura de flujo para el manejo de archivos personalizados
* Imágenes mediante programación utilizando la manipulación de estilos de imagen y efectos
* Comprender la nueva funcionalidad Image Styles de Drupal
* La implementación de sus propios efectos de imagen para su uso con Image Styles

En el momento en que haya terminado con este capítulo, usted debe tener una buena comprensión de la forma de manipular los archivos y las imágenes usando Drupal, incluyendo algunos métodos para recuperar archivos remotos e imágenes.

**Los módulos de Twitpic y marcas de agua**

En este capítulo, estaremos construyendo laTwitpic módulo. Este módulo le permitirá interactuar con las imágenes almacenadas en el sitio web de Twitpic (http://twitpic. com) E integrarlos con Drupal en una variedad de maneras. Se proporciona una corriente envoltorio que permite a los desarrolladores para tirar imágenes de Twitpic, refiriéndose a ellos con un URI personalizado, y ofrece una demostración de cómo de Drupal API imagen puede ser usado para manipular estas imágenes.

*Trabajar con archivos e imágenes*

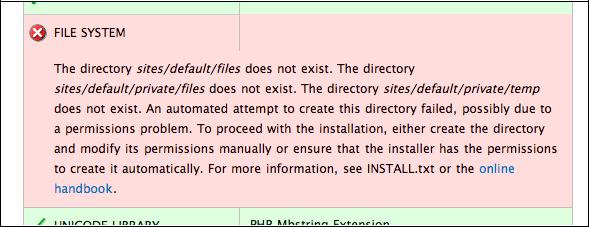


También vamos a crear un segundo módulo, que permite a los usuarios añadir una sencilla marca de agua de texto a las imágenes. Los usuarios podrán configurar la marca de agua para que muestre el texto de encargo en un color especificado. Este efecto estará disponible para su uso en

**Estilos de imagen**, La aplicación de Drupal 7 de laImagecachemódulo.

**Los archivos en Drupal**

Al instalar Drupal, por primera vez, usted probablemente tiene el siguiente error y se ha preguntado por qué era necesario crear tres directorios de los ficheros:



Drupal define tres tipos de almacenamiento de archivos, es decir, público, privado y temporal. Archivos públicos están disponibles para el mundo en general para ver o descargar. Aquí es donde se almacenan cosas como el contenido de la imagen, logos, y archivos descargables. Su directorio de archivos públicos debe existir en algún lugar debajo de la raíz de Drupal, y debe ser de lectura y escritura por cualquier "usuario" de su servidor web se ejecuta bajo. Archivos públicos no tienen restricciones de acceso. Cualquier persona, en cualquier momento, puede navegar directamente a un archivo público y ver o descargar.

Los archivos privados no están disponibles en el mundo para su descarga general. Directorio Los archivos privados debe residir fuera del directorio raíz de Drupal. Sin embargo, seguirá siendo de escritura por el usuario del servidor web. Aislar a los archivos privados de esta manera permite a los desarrolladores controlar quién puede y no puede acceder a ellos cuando lo deseen. Por ejemplo, podría escribir un módulo que sólo permite a los usuarios que tienen un rol específico, para acceder a los archivos PDF en el sistema de ficheros privados.

**[314 ]**



*Capítulo 11*



Es muy importante que los archivos privados que viven fuera de la raíz de la web de Drupal, a pesar del hecho de que por defecto no lo hacen. Para que los archivos privados que sean útiles, deben ser legibles para el usuario ejecuta el servidor web como.

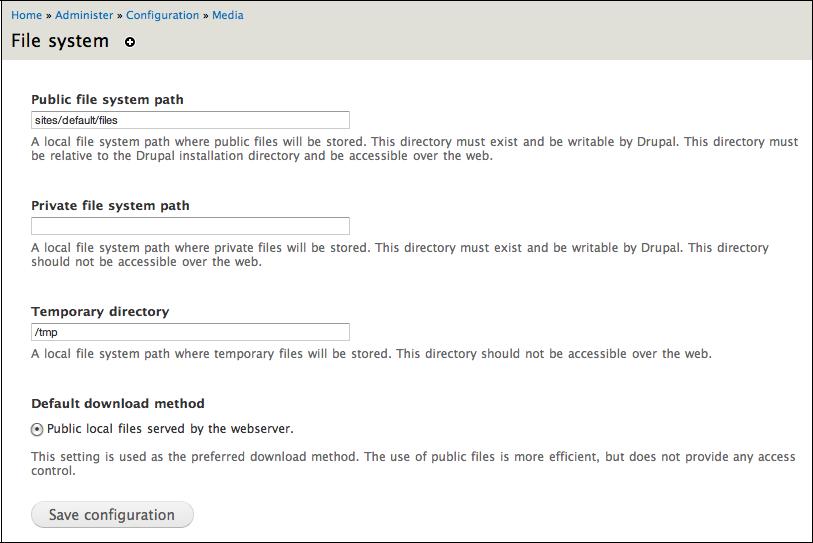


Sin embargo, si estos archivos son entonces bajo la raíz web de Drupal, serán legibles a nadie. Las pruebas adecuadas es muy importante para asegurar correctamente los archivos privados. Para obtener más información sobre cómo proteger adecuadamente su sistema de archivos privados, consulte el siguiente sitio:

http://drupal.org/node/344806

Almacenamiento de archivos temporales es típicamente utilizado por Drupal para operaciones internas. Cuando los archivos se guardan primero elaborado con Drupal, que se escriben primero en el área de archivo temporal para que puedan ser verificados por los problemas de seguridad. Después de que han sido considerados seguros, que están escritas en su ubicación final.

Cada uno de los directorios en el mensaje de error anterior refleja la ubicación por defecto para cada tipo de archivo. Puede cambiar estas ubicaciones predeterminadas después de la instalación está completa al iniciar sesión como administrador y visitaradmin / config / media / sistema de archivos como se ve en la siguiente imagen:



También puede indicar si el método de descarga por defecto debe ser pública o privada. (Después de la instalación es público.)

**[315 ]**



*Trabajar con archivos e imágenes*



**API archivo**

En Drupal 6, se prestó mayor manejo de archivos funcionalidad a través de una API de núcleo áspera combinado con módulos de terceros, tales como FileField. Drupal 7 proporciona una API más robusto y consistente que permite a los desarrolladores interactuar con los archivos en un conjunto estándar de funciones que realizan tareas como crear, abrir, mover y eliminar archivos.

Para que los archivos que se van asociadas a los nodos y otros contenidos Drupal, deben tener un registro en Drupal expediente tabla. Cada registro identifica un archivo con un identificador único, así como los metadatos asociados, como el tamaño del archivo y tipo MIME.

Muchas funciones de la API de archivos, tales como file\_copy () y file\_move (), Tome un objeto de archivo como uno de sus argumentos. El objeto de archivo es un estándar de la clase PHP que contiene los metadatos de la archivos mesa, y estas funciones de API gestionar la actualización de la información en el archivos mesa cuando los archivos se mueven o eliminan. Esta es una razón que es tan importante la utilización de estas funciones de la API de archivos asociados con el contenido-si no lo hace, el archivos tabla será inconsistente y sus archivos no puede mostrar correctamente.

Si tiene que trabajar con archivos fuera del contexto de contenidos Drupal, hay un conjunto separado de funciones de la API de archivos con no administrado en su nombre. Por ejemplo, cuando file\_copy () se actualizará la tabla de archivos y copiar su archivo, file\_unmanaged\_copy () se acaba de copiar el archivo.

Para obtener una lista completa de las funciones disponibles en la API de archivos, consulte la documentación de la API en:

http://api.drupal.org/api/group/file/7



Aquí está un ejemplo sencillo de cómo funciona esto. Una tarea común, mientras que la construcción de un sitio de Drupal está importando datos de otra fuente y convertirlo en nodos. Esto no sólo incluirá información textual, como el título y el cuerpo, sino también imágenes. Estas imágenes pueden vivir en un directorio local, o pueden vivir en un sitio web que está importando desde.

Echemos un vistazo a cómo podemos tomar un archivo desde un sitio externo, guardarlo en el sistema de archivos por defecto, y adjuntarlo a un nodo que creamos. Para este ejemplo, se va a trabajar con el campo imagen en el tipo de contenido del artículo.

Primero tenemos que conseguir un archivo y guardarlo:

$ Image = file\_get\_contents ('http://drupal.org/files/issues/druplicon\_ 2.png');

$ File = file\_save\_data ($ imagen, "público :/ / druplicon.png ', FILE\_EXISTS\_ REPLACE);

**[316 ]**



*Capítulo 11*



Con el fin de abrir los archivos desde ubicaciones remotas, PHP debe tener la allow\_ url\_fopen ajuste habilitadas en su php.ini. Para obtener más información, consulte:



http://us2.php.net/manual/en/filesystem.configuration. php # ini.allow-url-fopen

Esto es bastante sencillo. Utilizando la función de PHP file\_get\_contents (), Cogemos una imagen de la mascota de Drupal, el Druplicon, y guardarlo en la variable $ Imagen. Luego guardamos localmente usando la función API de Drupal file\_save\_data (), que devuelve un objeto de archivo file\_save\_data (), Y toma tres argumentos. El primer argumento es el contenido del archivo, como una cadena. file\_get\_contents () devuelve una cadena, así que esto funciona bien.

El segundo argumento especifica la ubicación donde se guardará el archivo. Este destino debe ser representado como un URI, utilizando una de las envolturas de corriente registrados en el sistema. Vamos a discutir las envolturas de flujo en más detalle más adelante en el capítulo, pero por ahora, sólo sé que usted puede referirse a cualquiera de los tipos de sistemas de archivos de Drupal usando un esquema URI personalizado, es decir,:/ público /,privada :/ /, O temperatura :/ /. Esto leer o escribir el archivo en el sistema de archivo adecuado sin que el desarrollador necesidad de conocer los detalles de dónde se encuentren físicamente los archivos. Aquí estamos salvando nuestro archivo al sistema de archivos públicos.

El tercer argumento especifica qué file\_save\_data () debe hacer cuando ya existe un archivo con el mismo nombre que el archivo que estamos tratando de salvar. Hay tres constantes definidas para indicar las posibles acciones que Drupal puede tomar:

* FILE\_EXISTS\_REPLACE: El nuevo archivo se debe sobrescribir el archivo existente.
* FILE\_EXISTS\_RENAME: Cambie el nombre del nuevo archivo anexando una de incremento número al nombre del nuevo archivo hasta que se produzca ninguna colisión. Por ejemplo, si druplicon.pngy druplicon\_1.png ya existía, entonces el nuevo archivo sería druplicon\_2.png.
* FILE\_EXISTS\_ERROR: No hacer nada y justo retorno FALSO.

La opción por defecto es FILE\_EXISTS\_RENAME pero hemos determinado que el archivo debe ser reemplazado si es que existe.

Una vez guardado el archivo, se devuelve un objeto de archivo. Este objeto contiene el fid o ID de archivo, así como los metadatos asociados. Ahora que hemos guardado la imagen, podemos crear un nodo y adjuntar la imagen a la misma:

$ Node = new stdClass; $ Node-> type = 'artículo'; node\_object\_prepare ($ node);

**[317 ]**



*Trabajar con archivos e imágenes*



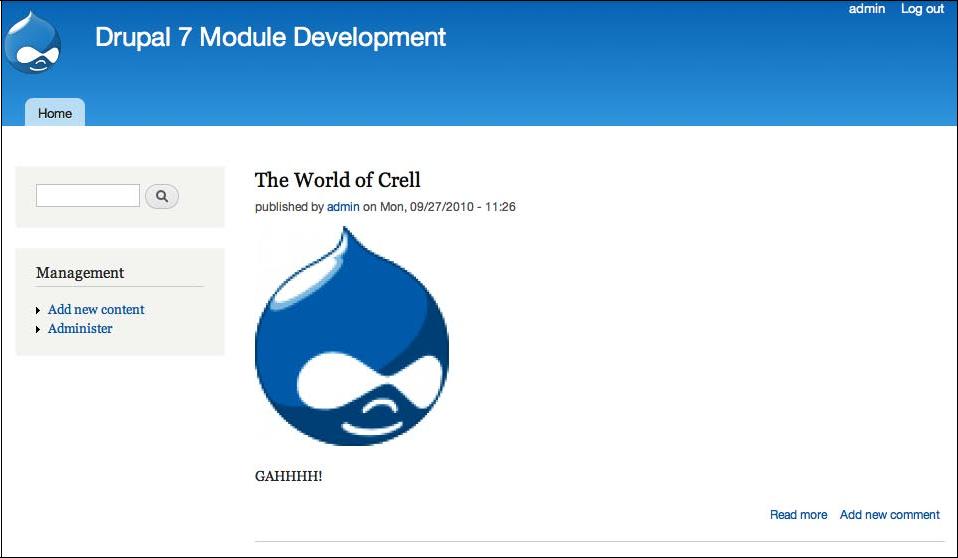
$ Node-> title = "El Mundo de Crell '; $ Node-> language = LANGUAGE\_NONE;

$ Node-> cuerpo [LANGUAGE\_NONE] ['0 '] [' valor '] =' GAHHHH '; $ Node-> field\_image [LANGUAGE\_NONE] ['0 '] [' fid '] = $ file-> fid; node\_save ($ node);

Como se discutió en el Capítulo 6, Trabajar con contenido, un nodo es un objeto y campos son propiedades del objeto, indexada por el lenguaje. En términos de este ejemplo, el

línea resaltada es la más importante. Todo lo que necesitamos hacer, es asociar nuestro archivo con el campo de la imagen, se añade el fid de nuestro objeto de archivo devuelto a un fid propiedad de la instancia del campo. Cuando se guarda el nodo, Drupal extraerá toda la información adecuada de la archivos mesa y agregarlo al campo de la imagen.

Eso es todo! Después de ejecutar este código, se puede visitar la página principal del sitio y usted debería ver algo como lo siguiente:



Este sencillo ejemplo muestra lo fácil que es manejar archivos en Drupal, y debe proporcionar un buen punto de partida para una mayor exploración.

Como se mencionó anteriormente en este capítulo, la API de archivos de Drupal 7 utiliza contenedores de flujo de PHP. También se introduce la posibilidad de que los desarrolladores crear sus propios envoltorios corriente PHP y los integran con el manejo de archivos de Drupal. Echemos un vistazo a lo que la corriente envolturas son y cómo los desarrolladores pueden usarlos.

**[318 ]**



*Capítulo 11*



**Envolturas Stream**

Si usted ha estado escribiendo PHP por mucho tiempo, lo más probable que haya necesarios para trabajar con archivos locales o remotos en algún momento. El siguiente código PHP es una manera común de leer un archivo en una variable que se puede hacer algo con:

$ contenido = ";

$ Gestor = fopen ("/ var / www / htdocs / images / xyzzy.jpg", "rb"); while (! feof ($ gestor)) {

$ contenido = fread ($ gestor, 8192).;

}

fclose ($ gestor);

Esto es bastante sencillo. Usted obtiene un identificador para un archivo local utilizando fopen () y leer 8 KB trozos del archivo utilizando fread () hasta feof () indica que se ha llegado al final del archivo, momento en el que usted fclose () el mango. Los contenidos del archivo son ahora en la variable $ contents. Además de los archivos locales, también se puede acceder a los archivos remotos a través de fopen () de esta manera:

$ Gestor = fopen ("http://drupal.org/files/issues/druplicon\_2.png", "rb");

Datos que se puede acceder de esta manera es en flujo continuo, lo que significa que puede abrirlo, cerrarlo, o buscar a un lugar arbitrario en el mismo. Envolturas Stream son una capa de abstracción encima de los arroyos que cuentan PHP cómo manejar tipos específicos de datos. Cuando se utiliza una envoltura de flujo, se hace referencia al archivo como un tradicional URL-esquema :/ / destino. A menudo

el objetivo será la ruta y el nombre de un archivo ya sea ubicado localmente o de forma remota, pero como veremos en nuestro código de ejemplo, puede ser cualquier dato que identifica de manera única los datos que está intentando acceder.

Los ejemplos anteriores utilizan dos de PHP incorporado en las envolturas de los arroyos. El segundo utiliza el http:// envoltura para acceder a sitios web utilizando el http protocolo y el primero utiliza el file :/ / envoltura para acceder a los archivos en el almacenamiento local. file :/ / es el esquema por defecto cuando no se especifica, por lo que en este caso simplemente pasando la ruta del archivo funciona bien.

PHP también permite a los desarrolladores definir sus propios envoltorios para los esquemas que PHP no maneja fuera de la caja, y la API de archivos de Drupal ha sido construido para tomar ventaja de esto. Por ejemplo, Drupal define elprivado esquema para permitir a los desarrolladores de Drupal para interactuar con los archivos del sistema de archivos privados de Drupal. Echemos un vistazo a cómo funciona esto mediante la creación de un plan para recuperar imágenes de un sitio web remoto.

**[319 ]**



*Trabajar con archivos e imágenes*



**Creación de una envoltura de secuencia**

En este ejemplo vamos a crear una envoltura de secuencia para recuperar fotos de Twitpic, un servicio de alojamiento de imágenes para los usuarios de Twitter. Twitpic define una API REST para recuperar fotos de la URLhttp://twitpic.com/show/ <tamaño> / <image-id> donde el tamaño es ya sea Mini o pulgar, Y la imagen-id es un identificador único que puede recuperar de la dirección URL de una foto, como se ve en la siguiente captura de pantalla:



La API completa Twitpic se define en http://twitpic.com/api.do

Así que usted puede recuperar la imagen de esta foto de la URL http://twitpic. com/show/thumb/7nyr2. Esto hace que sea muy fácil para referirse a las fotos de Twitpic en el código. Sin embargo, si este formato URL debe cambiar, entonces usted podría terminar con un montón de código para limpiar. Podemos mitigar este escribiendo una envoltura de secuencia que encapsula esta lógica en un solo lugar. Esta envoltura de flujo utilizará el formato

Twitpic :/ / <image-id> / <tamaño>.

Hay dos cosas que tenemos que hacer para crear un contenedor de secuencia personalizada en Drupal. En primer lugar, tenemos que crear una clase personalizada que implementa la funcionalidad de nuestro, entonces tenemos que registrar la clase utilizandohook\_stream\_wrappers ().

PHP define un conjunto de funciones que fluyen envoltorios puede poner en práctica, como se indica en

http://www.php.net/manual/en/class.streamwrapper.php. Drupal se expandió en esa lista y crearon una interfaz llamadaDrupalStreamWrapperInterface. Cualquier clase de contenedor de corriente utilizada en Drupal debe implementar esta interfaz o de lo contrario no se registrará correctamente.

**[320 ]**



*Capítulo 11*



En algunos casos, puede que no necesite algunas de esta funcionalidad proporcionada por la interfaz. Por ejemplo, en nuestro ejemplo, sólo estamos leyendo fotos de Twitpic, sin ofrecer la posibilidad de escribir datos en cualquier lugar, por lo que funciona como stream\_write ()

y stream\_mkdir () no se aplican. En estos casos, simplemente retornamos FALSO.



Para los detalles de implementación completos de

DrupalStreamWrapperInterface, Consulte http://api.drupal.

org/api/drupal/includes--stream\_wrappers.inc/7. Usted puede también quiera consultar prototipo de la clase envoltura de secuencia de PHP enhttp://



php.net / manual / es / streamwrapper.stream-open.php.

DrupalStreamWrapperInterface es bastante extensa, con más de 20 funciones para ser definido por las clases de aplicación. Cada envoltura de flujo debe cumplir cada una de las funciones de manejo de archivos de PHP. Como se mencionó anteriormente, muchos de estos simplemente retorno FALSO. Otros simplemente pasan a través de igualar la funcionalidad de PHP. (stream\_eof () simplemente pide y devuelve los resultados de feof ($ gestor).)

Puesto que la clase es tan grande, es posible que desee ponerlo en un archivo separado para mejorar la legibilidad y mantenibilidad del código. Usted puede hacer esto mediante la creación de un nuevo archivo en el directorio de su módulo, y agregarlo a laarchivos [] matriz en su module.info presentar como se muestra:

archivos [] = twitpicstreamwrapper.inc

Con el fin de mantener las cosas simples, discutiremos solamente las partes más notables de la clase que se muestran en el siguiente código. El listado de código completo se puede descargar desde el sitio web de Packt.

/ \*\*

* Twitpic Corriente Envoltura
* Esta clase proporciona una implementación completa envoltura de secuencia.

\* /

clase TwitPicStreamWrapper implementa DrupalStreamWrapperInterface {

/ \*\*

\* Instancia URI como esquema :/ / destino. \* /

protected $ uri;

/ \*\*

\* Un asa recurso genérico. \* /

public $ gestor = NULL;

**[321 ]**



*Trabajar con archivos e imágenes*



/ \*\*

* Invalida getExternalUrl ().
* Devuelva la URL HTML de una imagen de Twitpic.

\* /

función getExternalUrl () {

* Obtenga image\_id y tamaño de la URI en una matriz. $ Target = file\_uri\_target ($ this-> uri);

$ options = array\_combine (array ('image\_id', 'size'), explode ('/', $ destino)

);

* Si no se especifica ningún tamaño, por defecto en el pulgar.

if (empty ($ options ['size'])) {$ options ['size'] = 'pulgar';

}

/ / Crear el URL

$ Url = 'http://twitpic.com/show/'. $ options ['size']. '/'. $ options ['image\_id'];

return $ url;

}

/ \*\*

\* Soporte para fopen (), file\_get\_contents (), etc \* /

función pública stream\_open ($ uri, $ mode, $ options, & $ ruta\_abierta) {$ allowed\_modes = array ('r', 'rb');

if (! in\_array (modo $, $ allowed\_modes)) {return false;

}

$ This-> uri uri = $;

$ Url = $ this-> getExternalUrl ();

$ This-> handle = ($ opciones y STREAM\_REPORT\_ERRORS)? fopen ($ url, $ mode): @ fopen ($ url, $ mode);

return (bool) $ this-> mango;

}

}

**[322 ]**



*Capítulo 11*



A sólo lectura envoltura de secuencia como la nuestra tiene que realizar dos funciones principales. En primer lugar, es necesario traducir un URI como Twitpic :/ / y6vvv/thumb a una dirección URL o la ruta que se puede abrir y leer. En segundo lugar, tiene que ser capaz de abrir un identificador de archivo a este recurso para que los desarrolladores pueden obtener los datos necesarios.

Para administrar el primer requisito, se ha implementado getExternalURL (). Cualquier clase que implementa DrupalStreamWrapperInterface que se requiere para anular esta función con su propia implementación. Este código es bastante sencillo; acabamos Parse URI del objeto, establecimos algunos valores por defecto adecuados, y devolver una URL API Twitpic apropiadamente estructurado:

función getExternalUrl () {

/ / Obtener image\_id y el tamaño de la URI en una matriz. $ Target = file\_uri\_target ($ this-> uri);

$ options = array\_combine (array ('image\_id', 'size'), explode ('/', $ destino)

);

Observe el uso de file\_uri\_target () para recuperar la información de destino de la URI. Esta es una función de ayuda proporcionada por la API de archivos de Drupal para hacerlo más fácil para analizar los URI envoltura de secuencia. También puede llamar al file\_uri\_scheme () para recuperar el esquema de un URI.

La stream\_open () función es igualmente sencilla. Esta se llamará cuando un desarrollador intenta abrir un recurso manejado por nuestra envoltura de flujo utilizando funciones PHP como fopen () o file\_get\_contents (). Esta función toma cuatro argumentos, y necesita volver FALSO o un identificador para nuestro recurso.

El primer argumento es el URI de nuestra envoltura. El segundo argumento, $ Mode, Indica si la secuencia debe ser abierta para la lectura y / o escritura, así como de otras banderas. Cualquier modo puede tenerb anexa a la misma, para indicar que el archivo debe abrirse en modo binario. (Entonces, ¿dónde r indica de sólo lectura, rb indica sólo lectura en modo binario.)

**$ allowed\_modes = array ('r', 'rb');**

if (! in\_array (modo $, $ allowed\_modes)) {return false;

}

Estamos implementando un esquema de sólo lectura, por lo que si tenemos cualquier modo que no r o rb nosotros devolverá FALSE.

**[323 ]**



*Trabajar con archivos e imágenes*



El tercer argumento es una máscara de bits de las opciones definidas por PHP. La que estamos tratando aquí es STREAM\_REPORT\_ERRORS, Que indica si o no

Errores de PHP deben ser suprimidas (por ejemplo, si no se encuentra un archivo). El segundo es

STREAM\_USE\_PATH, Que indica si PHP de ruta de inclusión debe ser revisado si no se encuentra un archivo. Esto no es relevante para nosotros, así que lo ignoran. Si un archivo se encuentra en la ruta de inclusión, a continuación, el cuarto argumento $ Opened\_url se debe establecer con la ruta real del archivo.

Mirando el resto del código de stream\_open (), Podemos ver cómo viene esto juntos:

$ This-> uri uri = $;

$ Url = $ this-> getExternalUrl ();

Guardamos nuestra URI en una propiedad protegida, y luego llamamos getExternalURL () traducirlo a una dirección URL real Twitpic que podemos tomar una foto de. Podemos entonces fopen () esta URL y poner nuestra mano interior.

if ($ options && STREAM\_REPORT\_ERRORS) {$ this-> handle = fopen ($ url, $ mode);

}

else {

$ This-> mango = @ fopen ($ url, $ mode);

}

Si STREAM\_REPORT\_ERRORS no se establece, suprimimos fopen () errores anteponiendo @, Lo que indica a PHP que los errores no se deben anotar. Siempre es una buena práctica de codificación para manejar adecuadamente las opciones disponibles para sus funciones de flujo si son aplicables a su caso el usuario.

Además de crear una implementación de DrupalStreamWrapperInterface, Los módulos que definen sus propias envolturas de flujo deben registrarlos en el registro envoltura de secuencia de Drupal mediante la implementación dehook\_stream\_wrappers (). Este hook devuelve una matriz asociativa que define algo de información acerca de nuestra envoltura de flujo, como se muestra en el siguiente código:

/ \*\*

\* Implementar hook\_stream\_wrappers (). \* /

twitpic\_stream\_wrappers función () {{return array (

'Twitpic' => array (

'Name' => 'Twitpic fotos', 'class' => 'TwitpicStreamWrapper',

'Description' => t ('Fotos de la servicio de alojamiento Twitpic.')

**[324 ]**



*Capítulo 11*



'Type' => STREAM\_WRAPPERS\_READ\_VISIBLE,

),

);

}

La matriz que se teclea en el esquema de nuestro envoltorio, en este caso Twitpic. Cada esquema tiene a su vez definir otro array asociativo con las siguientes claves:

* nombre: Un nombre descriptivo corto para nuestra envoltura.
* clase: El nombre de la clase PHP que implementa corriente de Drupal interfaz de contenedor.
* descripción: Una o dos frases describiendo lo que hace este envoltorio.
* tipo: Constante que indica qué tipo de envoltura de secuencia es legible y / o grabable, local o remoto, entre otras cosas. Estos se definen en includes / stream\_wrappers.inc y puede ser revisado en: http://api. drupal.org/api/drupal/includes--stream\_wrappers.inc/7

Tenga en cuenta que en nuestro ejemplo hemos definido nuestra envoltura como STREAM\_WRAPPERS\_ READ\_VISIBLE. Esto significa que sólo se lee, pero visible en la interfaz de usuario de Drupal. Un ejemplo de un contenedor que no es visible en la interfaz de usuario es Drupal temperatura :/ / esquema, que es sólo para uso interno (que se establece en STREAM\_WRAPPER\_HIDDEN).

Esto es todo lo que se necesita para implementar su propia envoltura de secuencia personalizada. Puede parecer mucho, pero una vez que entienda lo que debe ponerse en práctica, es realmente muy simple.

Ahora que su envoltura de secuencia ha terminado, usted será capaz de acceder a las fotos de

Twitpic misma facilidad que cualquier otra fuente remota utilizando la API de archivos de Drupal. Ahora que podemos hacer esto, echemos un vistazo a algunas de las formularios en que de Drupal API imagen se puede utilizar para modificar y administrar imágenes.



En este ejemplo se ha centrado sobre todo en la parte Drupal-específica de las envolturas de flujo de escritura. Para obtener más documentación general sobre



envolturas de flujo ver http://us2.php.net/manual/en/intro. stream.php

**[325 ]**



*Trabajar con archivos e imágenes*



**Imágenes en Drupal**

Así como el contribuido FileField módulo de archivo ampliamente manejado en el manejo de Drupal

6, dos módulosImageField y Imagecache, En gran parte se maneja el tratamiento de imágenes. ImageField fue utilizado para adjuntar imágenes a los nodos, y Imagecache se utilizó para crear derivaciones de esas imágenes por el cambio de tamaño o recortar ellos. Esto era muy

popular para cosas como la creación de miniaturas cuadrados en una cuadrícula de galerías de imágenes. La funcionalidad de ambos módulos se ha puesto en núcleo para Drupal 7, junto con una mejora de la API para la gestión de esta funcionalidad de código.

**API de imagen**

El Drupal 7 API de imagen proporciona una variedad de funciones para manipular imágenes. Por defecto, Drupal utiliza la biblioteca de gestión de imágenes GD que se incluye con PHP. Sin embargo Drupal también ofrece la posibilidad de cambiar a una biblioteca diferente si es necesario.

Por ejemplo, un módulo contribuido podría implementar el ImageMagick biblioteca para desarrolladores que necesitan apoyo para este tipo de imagen adicionales, tales como TIFF, que GD no soporta.

Trabajar con imágenes es similar a trabajar con archivos. Usted obtiene un objeto de imagen al abrir una imagen local utilizandoimage\_load (), Y luego pasar este objeto a una de las funciones de manipulación de imágenes proporcionadas por Drupal. Una vez que haya realizado las modificaciones deseadas en la imagen, guardarla usando image\_save ().



Funciones de la API de imagen sólo pueden acceder a los archivos en su sistema de archivos local.



Puede seguir utilizando los sistemas de envoltura de secuencia como :/ público / y privada :/ / para referirse a los archivos, pero los sistemas de archivos remotos no funcione correctamente.

Las siguientes funciones de Drupal están disponibles para la manipulación de imágenes:

* image\_crop (): Recortar una imagen con las dimensiones especificadas.
* image\_desaturate (): Convertir una imagen a escala de grises.
* image\_resize (): Cambiar el tamaño de una imagen con las dimensiones especificadas. Esto puede afectar relación de aspecto de la imagen.
* image\_rotate (): Girar una imagen para un número específico de grados.
* image\_scale (): Cambiar el tamaño de una imagen con las dimensiones especificadas sin afectar relación de aspecto de la imagen.
* image\_scale\_and\_crop (): Combine escala y los cultivos en una sola operación.

**[326 ]**



*Capítulo 11*



Para más detalles sobre estas y otras funciones disponibles en la API de imagen hacen referencia a http://api.drupal.org/api/group/image/7

Echemos un vistazo a cómo podemos integrar algunas de estas funciones con nuestro módulo de Twitpic. En primer lugar, vamos a hacer crear unhook\_menu () aplicación que podemos utilizar para activar nuestro código.

/ \*\*

\* Implementar hook\_menu (). \* /

función twitpic\_menu () {$ items = array ();

$ items ['twitpic / munge /%'] = array ('title' => 'Munge una imagen Twitpic',

'Description' => 'Muestra una imagen de Twitpic, munged de diversas maneras (recortar, rotar, etc.)',

La página de callback '=>' twitpic\_image\_munge ',' argumentos de acceso '=> array (' access\_content '),' type '=> MENU\_CALLBACK,

);

return $ artículos;

}

Cuando la dirección URL Twitpic / munge se solicita, este hook llamará a la función twitpic\_image\_munge (), Que toma dos argumentos. El primero es el ID de la imagen Twitpic queremos manipular, y la segunda es una operación que debe realizarse. Las operaciones permitidas son girar,escala, Y desaturar. Estos argumentos serán pasados ​​automáticamente al twitpic\_image\_munge () cuando anexa a la dirección URL, como se discutió en el Capítulo 5, La construcción de una interfaz de administración.

Echemos un vistazo a la función de devolución de llamada ahora:

/ \*\*

* Munge una imagen de Twitpic, aplicando un cambio de tamaño, recorte o
* girar a la misma.

\*

* @ Param $ id
* ID de la imagen, como se extrae de su URL original.
* @ Param $ operation
* Una operación a realizar en la imagen. Puede ser "rota",
* "Escala", o "desaturate '.

**[327 ]**



*Trabajar con archivos e imágenes*



\* /

twitpic\_image\_munge función ($ id, $ operation = 'girar') {/ / Si obtenemos una operación no permitida, simplemente devuelva. $ operaciones = array ('rotar', 'desaturar "," escala ");

if (in\_array ($ operación, $ operaciones)!) {return;

}

$ Twitpic\_uri = 'twitpic :/ /'. $ Id. '/ Thumb';

$ Local\_uri = 'public :/ /'. $ Id. ». Jpg ';

$ Twitpic\_image = file\_get\_contents ($ twitpic\_uri);

$ Local\_path = file\_unmanaged\_save\_data ($ twitpic\_image, $ Local\_

uri, FILE\_EXISTS\_REPLACE);

$ Local\_image = image\_load ($ local\_path);

switch ($ operation) {

caso "escala":

image\_scale ($ local\_image, NULL, 50, FALSE);

romper;

case 'desaturate':

image\_desaturate ($ local\_image);

romper;

caso "rota":

image\_rotate ($ local\_image, 45, 0x7D26CD);

romper;

}

$ Local\_uri = drupal\_tempnam ('public :/ / ", $ id);

image\_save ($ local\_image, $ local\_uri);

devolver el tema ('imagen', array ('path' => $ local\_uri));

}

Lo primero que hacemos es definir nuestras operaciones permitidas, y regresar si la operación aprobada en no coincide con uno de ellos. Esto permite que fracasemos con gracia en esa situación. Luego armamos el Twitpic URI y hacerse con una copia local de la imagen Twitpic utilizando file\_get\_contents () como lo hicimos antes en este capítulo. A los efectos de este ejemplo, acabamos de codificar el tamaño de pulgar, Pero se puede agregar fácilmente que como un argumento adicional, si desea hacerlo.

**[328 ]**



*Capítulo 11*



Después de esta preparación, empezamos de entrar en el meollo de la función.

$ Local\_path = file\_unmanaged\_save\_data ($ twitpic\_image, $ local\_uri, FILE\_EXISTS\_REPLACE);

$ Local\_image = image\_load ($ local\_path);

Como no estamos asociando este archivo con cualquier otro tipo de contenido en Drupal, no tenemos necesidad de que se introducirá en elarchivos tabla. Por lo tanto, usamos file\_unmanaged\_save\_data () en contraposición a file\_save\_data (), Con el fin para evitar registros innecesarios se escriban.

Una vez guardado el archivo, que llamamos image\_load () para volver $ Local\_image, Una imagen del objeto Drupal que podemos transmitir a las funciones de manipulación de imágenes. Como el objeto de archivo, un objeto de imagen contiene una variedad de información sobre la imagen que se ha cargado, incluyendo su altura y anchura, tipo MIME, y un mango a la imagen.

Ahora que tenemos un objeto de imagen, que puede meterse con él utilizando funciones de la API de Drupal. A los efectos de este experimento, que acabamos de codificar algunas manipulaciones de la muestra adentro, para ver el tipo de cosas que puede hacer.

switch ($ operation) {case 'desaturate':

**image\_desaturate ($ local\_image);**romper;

caso "escala":

**image\_scale ($ local\_image, NULL, 50, FALSE);** romper;

caso "rota":

**image\_rotate ($ local\_image, 45, 0x7D26CD);** romper;

}

Como puedes ver, tenemos tres manipulaciones posibles dependiendo de la operación pasa a través de la URL. Todas estas funciones trabajan directamente en nuestro objeto de imagen, $ Local\_image que se pasa por referencia, por lo que no tiene que preocuparse acerca de los valores de retorno.

El primer ejemplo, image\_desaturate (), Es el más simple. Simplemente convierte la imagen a escala de grises, sin argumentos de configuración.

**[329 ]**



*Trabajar con archivos e imágenes*



El caso escala utiliza image\_scale (), Que toma cuatro argumentos. Los dos primeros son la altura del objetivo y la anchura de la nueva imagen. Desde image\_scale () conserva la relación de aspecto de una imagen, que en realidad sólo tendrá que proporcionar una de estas, y puede dejar el otro NULL, como lo hemos hecho en el ejemplo. Si usted proporciona dos valores, uno de ellos puede acabar menor a la indicada, debido a los cálculos necesarios para preservar la relación de aspecto de la imagen. El último argumento especifica si o no image\_scale () debe upscale una imagen cuando se proporciona una altura y / o anchura que es más grande que el original. Normalmente, esto se traduce en una imagen de calidad inferior pixelada, por lo que hemos especificado FALSO.

Por último, la girar caso utiliza image\_rotate (), Que toma dos argumentos. El primero es el número de grados la imagen debe ser girado a, y el segundo especifica un color de fondo opcional que se debe utilizar para llenar cualquier espacio dejado por la rotación. Este color se debe especificar utilizando el valor hexadecimal del color. En este ejemplo, rotamos 45 grados y llenar el fondo con el color morado.

Por último tenemos que guardar y mostrar la imagen resultante:

$ Local\_uri = drupal\_tempnam (':/ pública /'); image\_save ($ local\_image, $ local\_uri);

devolver el tema ('imagen', array ('path' => $ local\_uri));

}

Creamos un nombre de archivo local utilizando drupal\_tempnam (), Que sólo genera un nombre de archivo aleatorio que podemos utilizar cuando se le da un camino. Puede parecer tentador utilizar la información como del archivo de Twitpic ID o la operación a realizar en el nombre del archivo. Sin embargo, en este ejemplo, estos elementos se pasan a través de la URL, y esto no se puede confiar. Es posible crear un problema de seguridad si no se es muy cuidadoso a partir de datos enviados por los usuarios en los nombres de archivo, por lo que lo mejor es evitar que, si es posible.

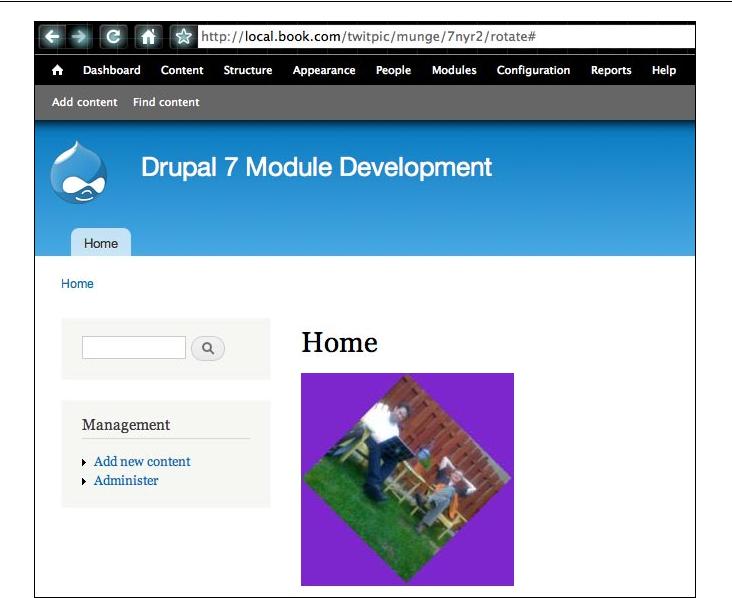
Ahora podemos pasar este camino, junto con nuestro objeto de imagen modificado, a image\_save (), Para salvar nuestra imagen para el sistema de ficheros públicos. Una vez guardado, se trata de una simple cuestión de llamar a la función de tema para las imágenes, para generar el código HTML necesario para nuestra imagen para

se mostrará. El siguiente es un ejemplo de lo que se ve cuando se solicitetwitpic/munge/7nyr2/rotate:

**[330 ]**



*Capítulo 11*



Todo esto está muy bien, pero a veces puede ser bueno para crear previamente algunas manipulaciones para uso repetido, o incluso un conjunto de manipulaciones que se ejecutan en orden. Esto es lo que los estilos de imagen son para, y vamos a ver ahora.

**Estilos de imagen**

Estilos de imagen es el reemplazo de la base de Drupal 7 para el popular Imagecache módulo, que ha estado presente desde Drupal 5. Estilos de imagen es un conjunto de manipulaciones que se pueden guardar como un grupo que se va a aplicar a las imágenes. Una vez guardados, estos estilos se pueden utilizar fácilmente como formateadores para los campos de la imagen, o se pueden utilizar en código llamando a una función de tema como lo hicimos anteriormente con nuestras imágenes artesanales.

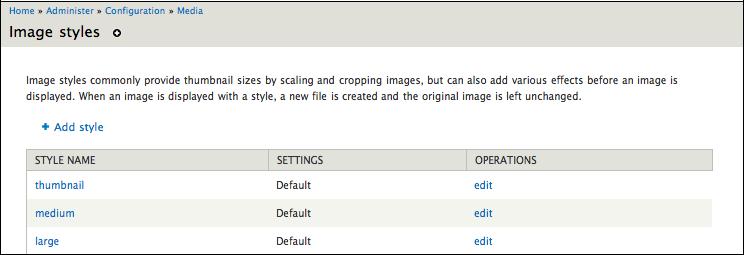
**[331 ]**



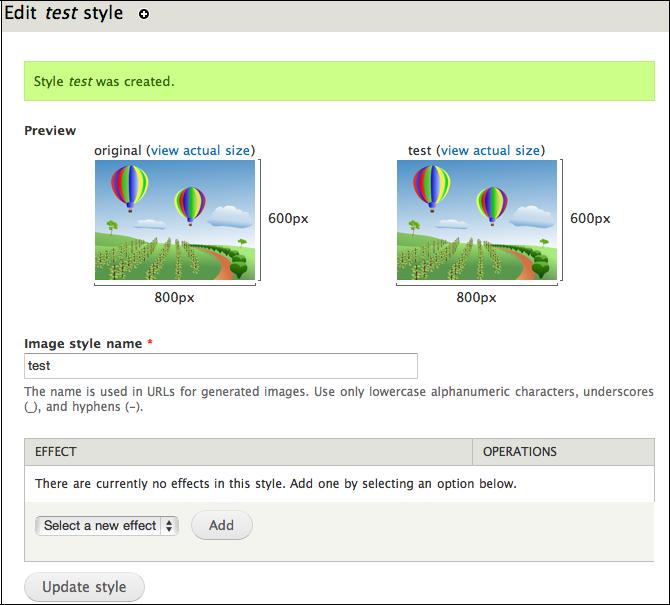
*Trabajar con archivos e imágenes*



Se puede crear un estilo de imagen, visitando admin / config / medios de comunicación / imagineros estilos donde podrás ver la siguiente pantalla:



Barcos Drupal con tres estilos de imagen de uso común en el lugar, es decir, en miniatura, medianas y grandes. Cada una de estas escalas de una imagen a un tamaño específico. Si desea crear un nuevo estilo, puede hacer clic en Añadir estilo, y después de introducir un nombre exclusivo para su estilo, verá la siguiente pantalla:



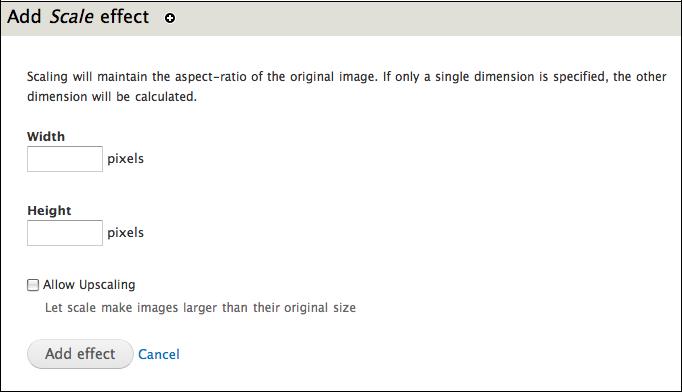
**[332 ]**



*Capítulo 11*



Aquí es donde usted puede comenzar a agregar nuevas manipulaciones (o, en la jerga de Drupal, "efectos") a la imagen. Como se puede ver en el menú desplegable mostrado, todos los efectos estándar de Drupal están disponibles para ser utilizados. Cuando seleccione un efecto y haga clic en el botón Agregar, puede obtener un formulario de configuración que le permite especificar opciones para tal efecto. Normalmente estas opciones de configuración que coincida con los argumentos de las funciones asociadas. Así que el efecto Desaturar no tiene un formulario de configuración, pero el efecto de escala es así, como se muestra:



Una vez que las opciones adecuadas se introducen y se guardan, la pantalla de estilos se actualizará con una vista previa en vivo de lo que el nuevo efecto se verá así. A medida que agrega más y más efectos, esta actualización en vivo seguirá reflejando ellos. También puede ajustar el orden en que se aplican los efectos, ya que esto puede afectar en gran medida la imagen resultante. La rotación antes de que un resultados de los cultivos en una imagen que es diferente de uno que se recorta antes de girar.

Esto puede parecer como un montón de información de configuración para un libro centrado en la codificación, pero proporciona el contexto para lo que vamos a hacer. Como puede ver, todas Drupal de efectos incorporados están disponibles para su uso en un estilo, pero lo que si usted necesita un efecto que no está en la lista? Por ejemplo, un efecto común es añadir una marca de agua de texto a las imágenes. Echemos un vistazo a cómo podemos hacer eso.

**[333 ]**



*Trabajar con archivos e imágenes*



**Creación de efectos de imagen**



Tenga en cuenta que este ejemplo no es compatible con bibliotecas de imágenes enchufables, y sólo funcionará con GD. Para obtener más información sobre cómo hacer



sus funciones de imagen compatibles con múltiples bibliotecas, consulte la documentación de la API de image\_toolkit\_invoke () en:

http://api.drupal.org/api/function/image\_toolkit\_ invocan / 7

Vamos a empezar con la suposición de que se ha creado un módulo estándar . Info archivo con toda la información necesaria para un módulo llamado filigrana. El primer paso en la creación de una nueva forma de imagen está implementando hook\_image\_effect\_info ().

/ \*\*

\* Implementa hook\_image\_effect\_info (). \* /

función watermark\_image\_effect\_info () {efectos $ = array ();

$ efectos ['filigrana'] = array ('label' => t ('Watermark'),

'Help' => t ('Añadir una marca de agua a una imagen. ")," Efecto callback' => 'watermark\_effect', 'forma callback' => 'watermark\_form', 'Resumen lema' => 'watermark\_summary',

);

return $ efectos;

}

Este tipo de hook debe mirar bastante familiarizado por ahora. Es muy similar a hook\_menu (), El cual fue discutido en *Capítulo 5*, *La construcción de una interfaz de administración*. Nosotros crear una matriz asociativa de los efectos que estamos definiendo, y devolverlo a la persona que llama. La matriz está enchavetado por un identificador único, en nuestro caso, filigrana. Esta matriz contiene la siguiente información acerca de nuestro efecto de la imagen:

* etiqueta: La etiqueta que se utilizará en la efectos desplegable de los estilos página. Este texto también se utilizará después de haber agregado su efecto, si no se ha definido un tema resumen a continuación.
* ayudar: Una descripción de texto de lo que hace su efecto.
* efecto de devolución de llamada: Una función que será llamada para ejecutar realmente su efectuar en una imagen. Esta función indica el lugar en el código de manipulación de imagen posible.

**[334 ]**



*Capítulo 11*



* formulario de devolución de llamada: Si su efecto permite a los usuarios introducir datos de configuración (Tales como los campos de anchura y altura en el ejemplo anterior escala), entonces esto indica la función que define el formulario de configuración. Este valor es opcional. Si su efecto no requiere ninguna información de configuración, a continuación, esta clave no es necesario.
* Resumen tema: Después de que su efecto se añade a un estilo, puede crear un tema devolución de llamada que devuelve el texto personalizado que se mostrará en la pantalla de estilos de administración. Por ejemplo, el efecto de giro se muestra el número de grados que el usuario ha introducido. Este valor también es opcional. Si no se proporciona, su efecto se mostrará el valor proporcionado en eletiqueta clave.

Como puede ver, hemos definido tres devoluciones de llamada que tenemos que crear. En primer lugar, vamos a ver en nuestro formulario de configuración. Sería bueno que los usuarios sean capaces de especificar qué texto les gustaría que en las imágenes con marcas de agua, por lo que ofrecerá la opción de introducir un texto personalizado, así como darles una opción para indicar qué color debe ser.

/ \*\*

* Estructura formulario para el formulario de configuración de marca de agua.
* @ Param $ data
* La configuración actual de este efecto marca de agua.

\* /

watermark\_form función ($ data) {$ form ['text\_color'] = array ('# type' => 'campo de texto',

'# Default\_value' => (isset ($ datos ['text\_color']))? $ data ['text\_ de color']: '# FFFFFF ",

'# Title' => t ('Color del texto'),

'# Description' => t ('El color del texto que se utilizará para esta marca de agua. Utilice colores hexagonales de estilo web (# FFFFFF para el blanco, # 000000 para el negro).'),

'# Tamaño' => 7, '# maxlength' => 7,

'# Element\_validate' => array ('image\_effect\_color\_validate'),

);

$ Form ['text'] = array ('# type' => 'campo de texto',

'# Default\_value' => (isset ($ datos ['texto']))? $ data ['text']: 'Drupal ama gatitos!',

'# Title' => t ('texto de marca de agua "),

'# Description' => t ('el texto que ha escrito sobre la imagen.'), '# Size' => 30,

'# Maxlength' => 60,

**[335 ]**



*Trabajar con archivos e imágenes*



);

devolver $ form;

}

Efecto formulario callbacks regresan estructuras API Formulario parciales. Usted sólo tiene que completar los campos que desea que los usuarios introduzcan, y Drupal se hará cargo de los detalles de la presentación. Hemos definido dos elementos de forma de mantener nuestras opciones de configuración, es decir,text\_color y texto. En general se trata de matrices API Formulario bastante estándar, pero hay un par de cosas que señalan.

Su solicitud de devolución de llamada también toma un solo argumento-$ Data-Que es una matriz asociativa que contiene los valores actuales de nuestras opciones de forma, si los hubiere. Usted debe utilizar esta información para llenar suforma elementos con las opciones actuales del usuario o, si están vacíos, para llenar en configuraciones normales. Hemos puesto el color por defecto para ser blanco, y el texto predeterminado para Drupal ama gatitos!

Una última cosa a tener en cuenta es que estamos utilizando la # Element\_validate propiedad para especificar un conjunto de devoluciones de llamada, que se utiliza para validar este elemento específico.

Esta es una propiedad de la API formulario que no hemos visto antes, pero usted puede utilizarlo en cualquier definición de elemento de formulario. Drupal contiene la funciónimage\_effect\_color\_validate () para verificar que los colores hexadecimales están correctamente introducidos, y tomar ventaja de esta función especificando aquí.

Ahora vamos a hacer un mensaje personalizado en base a las opciones del usuario, que se mostrará en efecto el listado del estilo de la imagen. Esto sólo nos permite dar un poco más de detalle para el usuario, y lo hace más evidente lo que nuestro efecto lo está haciendo.

/ \*\*

* Callback Tema para la salida de la imagen de resumen efecto marca de agua.
* @ param variables $
* Una matriz asociativa que contiene los datos de configuración.

\* /

función theme\_watermark\_summary (variables $) {$ data = $ variables ['datos'];

volver t ('con el color @ textcolor', array ('@ textcolor' => $ data ['text\_ de color']));

}

Como se puede ver simplemente devolvemos el texto que estamos utilizando junto con el color que se ha configurado. Esto se añade a la etiqueta que especificamos en nuestra declaración de efecto inicial.

**[336 ]**



*Capítulo 11*



Finalmente podemos hacer la modificación en nuestra imagen:

/ \*\*

* Imagen efecto de devolución de llamada; añadir una marca de agua de texto a una imagen.
* @ Param $ imagen
* Un objeto de imagen devuelto por image\_load ().
* @ Param $ data
* Una serie de atributos a utilizar al realizar la
* marca de agua efecto.
* @ Return
* TRUE en éxito. En caso de fallo.

\* /

función watermark\_effect (& $ imagen, $ data) {

$ data ['text\_color'] = str\_replace ('#','', $ data ['text\_color']); $ Roja = hexdec (substr ($ datos ['text\_color'], 0, 2));

$ Verde = hexdec (substr ($ datos ['text\_color'], 2, 2)); $ Azul = hexdec (substr ($ datos ['text\_color'], 4));

$ Color = imagecolorallocate ($ imagen-> recursos, $ rojo, verde $, $ azul); imagestring ($ imagen-> recursos, 5, 5, 5, $ data ['texto'], $ color);

}

Desde una perspectiva de Drupal, lo importante es que nuestro efecto de devolución de llamada, watermark\_effect (), Toma dos argumentos. El primero es un objeto de imagen como nos hemos encontrado antes, que contiene un identificador de recursos que podemos pasar a las funciones de GD para su modificación. La segunda es la matriz de datos como la aprobada en nuestro formulario de configuración de devolución de llamada, que contiene una matriz asociativa de la información de configuración enviado por el usuario. Esta matriz se teclea en los nombres de los campos como se especifica en el formulario de devolución de llamada anteriormente. Utilizamos la información en$ Data para formatear correctamente nuestro texto.

Más allá de que esta funcionalidad es bastante sencillo. La mayoría del código se refiere a formatear adecuadamente la información de color. En las primeras cuatro líneas convertimos el color hexadecimal enviada por el usuario (que se encuentra en$ data ['text\_color']) A los valores decimales RGB que vamos a necesitar más adelante. Hacemos esto mediante el análisis cada valor hexadecimal individual de la cadena y el uso de hexdec () para convertirlos a su equivalente decimal.

El resto del código se refiere a la funcionalidad de creación de imagen de la biblioteca de imágenes GD. Cubriremos esto brevemente, pero para más información, consulte la documentación GD en:

http://in2.php.net/manual/en/book.image.php

**[337 ]**



*Trabajar con archivos e imágenes*



Una vez que tenemos los valores de color, utilizamos imagecolorallocate () para crear un identificador de color que representa este color. Finalmente podemos llamar imagestring (), Una función de GD que traza el texto en una cadena. Sin entrar en demasiados detalles, los siguientes son los argumentos de esta función se lleva a:

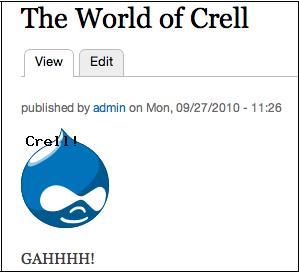
* Un mango de la imagen (en este ejemplo, la propiedad de los recursos del objeto de imagen)
* Un tamaño de la fuente 1-5 (5 es el más grande)
* Las coordenadas X e Y de los que el texto debe ser trazado con (0, 0) es la esquina superior izquierda de la imagen
* El texto que se representa, en este ejemplo, la $ data ['text'] como enviada por el usuario a través de nuestro formulario de configuración
* El identificador de color nos creó con la elección del color enviado por los usuarios

Ahora debería ser capaz de instalar este módulo y tener la opción de marca de agua a una imagen en la pantalla Image Styles admin, como se muestra en la siguiente captura de pantalla:



Ahora puede agregar este efecto, solo o en combinación con otros efectos a un nuevo estilo de la imagen. Este estilo de imagen se puede aplicar a las imágenes en los nodos mediante el ajuste de la configuración de visualización en las pantallas de administración para el tipo de contenido del nodo.

La siguiente imagen muestra nuestro nodo ejemplo anterior con una marca de agua configurada por el usuario aplicado a la imagen:



**[338 ]**



*Capítulo 11*



Derecho Pretty cool? Sin embargo, a veces sería agradable ser capaz de enviar algunos estilos de imagen preconfigurados con su módulo. Por ejemplo, un Drupal populares contribuyó módulo es el módulo de pirata, que se traduce todo su sitio web para piratear-hablan en charla como un día del pirata. Puede ser que sea fresco para el módulo de pirata para poder enviar un estilo de imagen totalmente preconfigurado que se suma el YAR! una marca de agua a las imágenes. En lugar de tener que crear este estilo de imagen a sí mismos, los administradores podrían simplemente instalar el módulo y que sería todo listo para ellos. No sólo eso, sino porque este estilo es en el código, es más fácil de implementar entre los entornos de desarrollo. Afortunadamente, esto es bastante fácil de hacer y le proporcionará nuestra demostración final de la API de imagen de Drupal 7.

**Creación de estilos de imagen de un módulo**

En este punto de su experiencia de Drupal, que es fácil adivinar lo que está involucrado en la creación de un estilo de imagen en el módulo de un hook y una matriz asociativa. El hook es hook\_image\_default\_styles () y la matriz es una definición simple de los componentes de su estilo.



Para aquellos familiarizados con vistas en Drupal 6, esto es muy similar a la exportación vistas al código utilizando hook\_views\_default\_view ().

/ \*\*

\* Implementa hook\_image\_default\_styles (). \* /

watermark\_image\_default\_styles function () {$ styles = array ();

$ estilos ['yar'] = array ('efectos' => array (

array (

'Name' => 'marca de agua', => array 'data' (

'Text\_color' => '# 000000', 'text' => 'YAR!'

),

'Peso' => 0,

),

),

);

return $ estilos;

}

**[339 ]**



*Trabajar con archivos e imágenes*



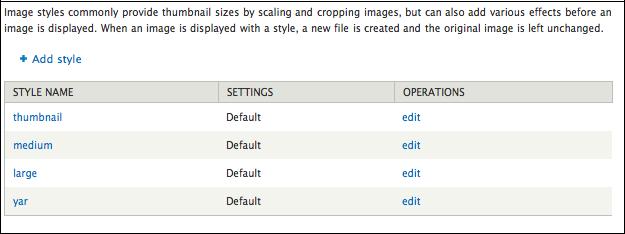
Esto es tan simple como hooks Drupal conseguir, especialmente teniendo en cuenta todo lo que ya se ha cubierto en este capítulo. Nuestra implementación de hook\_image\_default\_styles () retornos $ estilos, Una matriz asociativa de la imagen definiciones de estilo, con llave en una cadena única. Esta cadena actuará como el nombre del estilo y se mostrará en la página de estilos de imagen de administrador.

Cada definición de estilo es un conjunto de efectos de imagen, junto con sus ajustes de efectos específicos, si los hay. Las teclas de esta matriz puede contener son como sigue:

* nombre: El nombre único del efecto estamos implementando.
* datos: Matriz opcional de valores de configuración. Estos deben ser codificados en la misma cadena de texto que se utilizan en la forma de devolución de llamada definido por el efecto. Tenga en cuenta que los valores por defecto definidos en la forma de devolución de llamada no será respetada si no los incluyen. Usted tendrá que introducir a mano aquí.
* peso: Un peso opcional, que se utiliza para controlar el efecto de pedido cuando múltiples efectos se incluyen en un solo estilo. Como es norma Drupal, un menor peso

'Flotarán en la superficie "y" hundirse hasta el fondo' pesos más altos.

Eso es realmente todo lo que hay que hacer. Si carga los estilos de imagen lista, ahora verá su estilo en la lista (puede que tenga que borrar la caché de Drupal primero).

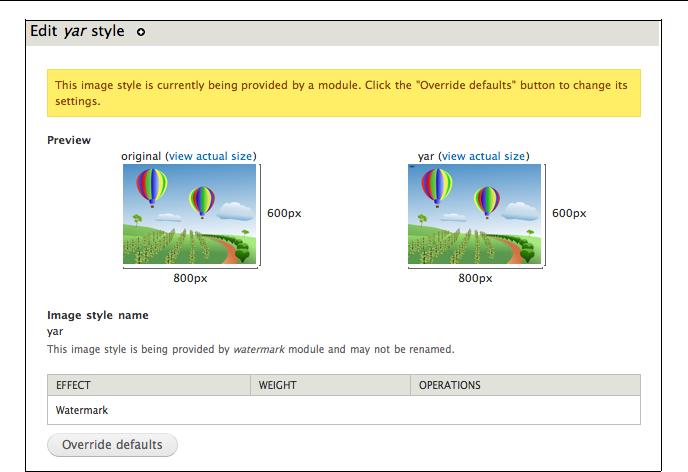


Un aspecto bueno de estilos de imagen definidos por un módulo es que están protegidos de ser cambiado o modificado por los usuarios finales (o roto accidentalmente por los propios desarrolladores). Esto se lleva a cabo de dos maneras. En primer lugar, con el fin de modificar la configuración de un estilo implementado a través de hook\_image\_default\_styles (), Debe tomar específicamente el paso extra de decir que desea anularlo. Echa un vistazo a la siguiente pantalla:

**[340 ]**



*Capítulo 11*



Como puede ver, hay una gran advertencia que indica que con el fin de cambiar la configuración de este módulo, debe hacer clic en el botón Reemplazar valores predeterminados. Esto puede ayudar a evitar que los usuarios cambien inadvertidamente los ajustes de imagen, lanzando así fuera el contenido y el tema de su sitio. Una vez que los impagos se han anulado, el usuario podrá modificar la configuración que deseen.

Esto nos lleva al segundo aspecto útil de la aplicación de sus estilos en el código, no importa cuánto se modifica un estilo; siempre se puede revertir a su estado original con el clic del ratón. Observe el cambio en la lista de estilo después de un usuario ha anulado un estilo predeterminado:



Como puede ver, este estilo se ha marcado como Reemplazado y ha ganado un enlace revert! Al hacer clic en el enlace revert te lleva a una pantalla de confirmación, y si usted se compromete a revertir, la configuración del estilo se restablecerá a su estado original, tal como se definen en el módulo.

**[341 ]**



*Trabajar con archivos e imágenes*



Estilos predeterminados son muy simples de implementar, pero como usted puede ver que traen con ellos una enorme cantidad de funcionalidad que puede mejorar drásticamente su flujo de trabajo. Si usted pasa mucho tiempo en la configuración de los ajustes de imagen o configuración entre los sitios en movimiento, no hay realmente ninguna razón para no aplicar sus estilos en un módulo usandohook\_image\_default\_styles ().

**Resumen**

Las nuevas API de archivos y de la imagen en Drupal 7 no sólo hacen que la gestión y el mantenimiento de estos activos mucho más fácil, sino que también son muy divertidos para jugar con y sorprendentemente fácil de usar. Ahora usted debe sentirse cómodo con los conceptos centrales de la gestión de archivos en Drupal. Usted debe saber la diferencia entre los sistemas de archivos públicos y privados de Drupal, y entender cómo Drupal asocia metadatos del archivo con los archivos que representan en elarchivos tabla. Usted también debe tener una buena comprensión de las envolturas de la secuencia y la forma en que está implementado en Drupal 7. Finalmente usted debe saber cómo utilizar la API de imagen para recortar o modificar imágenes de otra forma, y ​​la forma de guardar estas modificaciones Estilos de imagen para su uso en su contenido .

Muchos de los ejemplos de este capítulo son sólo cáscaras, y podría ser fácilmente mejorado o ampliado. Por ejemplo, sería razonablemente fácil de expandir el módulo de marca de agua de manera que se puede posicionar la marca de agua sobre la base de la elección del usuario, o contorno que para facilitar la lectura más fácil con distintos fondos. Experimentación

se anima!

Antes hemos hablado de cómo los ajustes de estilo de imagen predefinidos pueden mejorar el flujo de trabajo y facilidad de mantenimiento. En el siguiente capítulo nos pateamos ese concepto a un nivel superior al aprender cómo crear completamente preconfigurados instalaciones de Drupal usando instalar perfiles.

**[342 ]**



Perfiles de montaje

Este capítulo se centrará en la creación de perfiles de instalación. En capítulos anteriores se cubrieron los módulos y características del edificio en Drupal. Este capítulo cubre la agrupación módulos y temas con un instalador que configura Drupal para un propósito específico.

Éstos son algunos de los temas importantes que vamos a cubrir en este capítulo:

* Inicio de un nuevo perfil de instalación
* Agrupación de módulos y temas en una distribución
* Creación de un . Perfil presentar para almacenar las tareas de instalación
* Activación de los módulos en el proceso de instalación
* Configuración de los bloques por defecto
* Al ejecutar el instalador sin utilizar el asistente interactivo

Al final del capítulo, usted debe tener los conocimientos básicos para crear su propio perfil de instalación y distribución de Drupal.

**La introducción de perfiles de instalación**

Perfiles de instalación son la manera de tapar en y personalizar el proceso de instalación de Drupal. Con los perfiles de instalación, formularios adicionales y los pasos se pueden añadir al programa de instalación, los módulos se pueden habilitar, los bloques se pueden configurar, configuración por defecto se pueden configurar, y el sitio de Drupal se pueden personalizar.

Un perfil de instalación es útil cuando usted repite las mismas instalar tareas regularmente.

Estas tareas pueden ser automatizadas, lo cual es perfecto para la creación de una distribución de Drupal o para situaciones en las que usted construye el mismo sitio de base en varias ocasiones.

*Perfiles de montaje*



**Distribuciones de Drupal**

Distribuciones proporcionan funcionalidad out-of-the-box y sin la necesidad de descargar e instalar nada, pero la propia distribución. Sin una distribución, normalmente se necesita descargar Drupal y módulos adicionales y temas que usted necesita para que el sitio funcione. Luego hay que pasar el tiempo que discurre por un Drupal normales

proceso de instalación, seguidos de tiempo dedicado a la configuración del sitio. Con una «distribución», que descargarlo, ejecutarlo a través del instalador, y usted tiene un funcionamiento y un sitio configurado. Las distribuciones se basan en los perfiles de instalación, proporcionando todo lo necesario en un solo lugar.

**La creación de una distribución**

Distribuciones consisten en el núcleo de Drupal, módulos y temas adicionales, y un perfil de instalación para configurar el sitio. Al pasar por la creación de un perfil de instalación, también vamos a caminar a través de los pasos necesarios para crear una distribución en toda regla.

**Perfiles estándar y mínimos**

Drupal viene pre-empaquetado con los perfiles de instalación estándar y mínimos. El perfil de instalación estándar instala un conjunto común de módulos básicos. Esto es similar al perfil de instalación que viene con Drupal 6. El perfil mínimo de instalación sólo instala los módulos necesarios, junto con el bloque ydblog módulos.

**Creación de un directorio de perfil**

Los perfiles se encuentran en el / perfiles directorio, frente a la base del sitio Drupal. Para crear un nuevo perfil, crear un directorio junto con los perfiles estándar y mínimas previstas por el núcleo. El nombre debe ser el nombre legible por la máquina del perfil de la instalación que está creando. A lo largo de este capítulo vamos a construir un perfil de instalación con el nombre de la tienda.



Módulos, temas y perfiles no pueden compartir el mismo nombre legible por la máquina. Este nombre debe ser un espacio de nombres único para la instalación de Drupal.



**[344 ]**

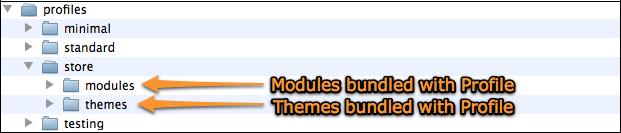


*Capítulo 12*



**Módulos de perfil y temas**

Mientras que los módulos y perfiles pueden ser almacenados en la / sites / all carpeta, tal como se describe en el capítulo 2, los módulos y los temas incluidos con un perfil tienen un lugar especial donde puedan residir. En el perfil que pueden tener sus propios módulos y directorios temas:



El almacenamiento de los módulos y temas dentro del perfil permite distribuciones a embalar juntos en una parte de Drupal, y permite que el sistema de módulos y tema principal estándar para trabajar.



Como los módulos y los temas pueden ser almacenados en más de un lugar hay un orden a la forma en Drupal elige qué fuente a utilizar para un módulo. Módulos y temas en la / sites / all carpeta se eligen a través de los de la carpeta del perfil, y los módulos y temas en una carpeta específica del sitio se eligen sobre las de las / Site / todos carpeta.



**Drupal.org liados Distribuciones**



La drupal.org sistema de envasado es capaz de tomar un perfil de instalación con una configuración adicional y la creación de una distribución descargable. Los detalles están disponibles en



http://drupal.org/node/642116.

**Creación de perfiles**

Perfiles consisten en una . Info archivo y un . Perfil (Pronunciado "punto-perfil") de archivos. Esto es similar a cómo se estructuran los módulos, como se describe en el capítulo 2.

La . Info archivo para un perfil se parece a una . Info presentar para un módulo y contiene la misma información.

name = Ejemplo Tienda

description = "Un perfil de instalación ejemplo tienda." version = VERSION

core = 7.x

archivos [] = archivos store.profile [] = archivos store.install.inc [] = store.install

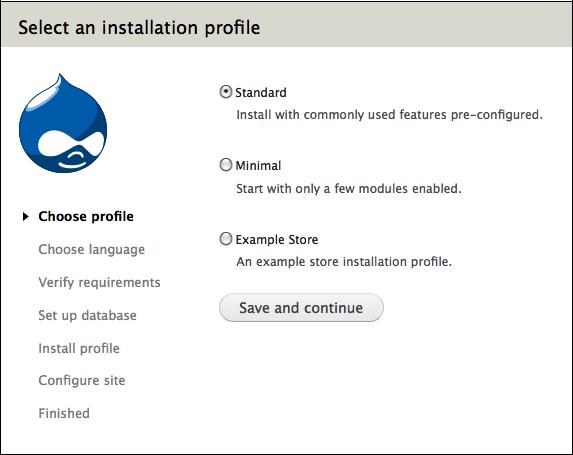
**[345 ]**



*Perfiles de montaje*



La nombre y descripción campos de la store.info archivo se utiliza en la primera etapa de la instalación, como se ve en la siguiente imagen:



Si sólo hay un perfil presente en la / perfiles directorio, Drupal seleccionará automáticamente ese perfil y pasar a la siguiente etapa.

La . Perfil archivo tiene dos propósitos. Primero, permite personalizaciones para el proceso de instalación. Estos cambios incluyen la adición de tareas en el instalador, como agregar un formulario personalizado, o alterar las tareas de instalación por defecto.

El segundo propósito de la . Perfil archivo es actuar como un módulo personalizado para el sitio. Después de la instalación se haya completado, el perfil funciona como cualquier otro módulo en el sitio, salvo que no se puede desactivar. Puede funcionar hooks Drupal, alterar formularios, y hacer todo lo demás un módulo normal puede hacer.

**[346 ]**



*Capítulo 12*



**Activación de los módulos**

Al igual que los módulos pueden ser dependencias de otros módulos, los módulos se pueden configurar como dependencias de un perfil. Todos los módulos dependientes se activarán como parte del proceso de instalación. La diferencia entre la dependencia de módulos y dependencias perfil es que las dependencias del perfil pueden desactivarse sin desactivar el perfil.

Para agregar una dependencia a un perfil, utilice el campo de las dependencias de la . Info presentar. Aquí hay un ejemplo de nuestra store.info archivo de los módulos que queremos activar:

dependencias [] = dependencias bloque [] = comentario dependencias [] = dependencias dependencias de contacto [] = dependencias de contexto [] = dependencias salpicadero [] = ayuda dependencias [] = dependencias imagen [] = menú [] = ruta dependencias [] = taxonomía dependencias dependencias [] = DBLog [] = dependencias de búsqueda [] = dependencias de acceso directo [] = dependencias barra de herramientas [] = dependencias de superposición [] = dependencias field\_ui [] = dependencias de los archivos [] = rdf



Por defecto, el campo, el almacenamiento de SQL campo, filtrar la lista, nodo, número, opciones del sistema, texto y módulos de usuario están habilitadas. Cualesquiera otros módulos que desee necesidad permitido que se enumeran en el. Info presentar para el perfil.



**Garantizar dependencias perfil**

Desde las dependencias de los perfiles se pueden desactivar, perfiles que se basan en las dependencias deben asegurarse de que el módulo esté habilitado antes de actuar sobre ella. La funciónmodule\_exists () puede ser utilizado para probar si un módulo está instalado y activado tanto. Para más detalles sobre module\_exists () ver

http://api.drupal.org/api/function/module\_exists/7.



**[347 ]**



*Perfiles de montaje*

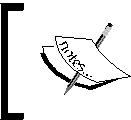


**El sistema de tareas de instalación**

El programa de instalación de Drupal es tarea basada. Las tareas consisten en medidas para recoger información y actuar en consecuencia. Por ejemplo, tres de las principales tareas son para recoger los detalles de configuración de base de datos, escriba la configuración de base de datos para la settings.php presentar, e instalar los módulos. Núcleo de Drupal proporciona un conjunto de tareas predeterminadas para el instalador para ejecutar ese

un perfil de instalación puede añadir o modificar. Los dos hooks que permiten acceder a las tareas de instalación son:

* hook\_install\_tasks (): Esto permite que el perfil para agregar tareas al final de la tareas predeterminadas
* hook\_install\_tasks\_alter (): Esta permite que el perfil de alterar todas las tareas incluidos los proporcionados como valores predeterminados por el instalador



Para el primer paso de la instalación, se selecciona un perfil de instalación. Este paso no puede ser alterada por un perfil de instalación.

**Elegir una tarea de instalación o el uso de hook\_install**

Dado que los perfiles operan como un módulo, que pueden tener un . Instalar archivo que contiene una hook\_install () que se llamará cuando se instala el perfil. Esto deja dos opciones de dónde realizar tareas de configuración.

Hay dos diferencias principales entre estos dos tipos de tareas:

* El perfil hook\_install () se ejecuta antes de instalar encargo tareas y la forma de configuración del sitio.
* Tareas en el instalador pueden tener formularios, utilizar la API de lote, y más. hook\_install () sólo puede funcionar como una devolución de llamada en una página sola carga.



Para obtener la documentación completa sobre hook\_install () visite la documentación de la API en http://api.drupal.org/api/function/hook\_ install / 7.



**Anatomía de una tarea de instalación**

Hay cinco propiedades para describir cada paso de trabajo para el instalador. hook\_install\_tasks () describe cada paso para el instalador como una matriz por clave. La tecla es un nombre único a una función de devolución de llamada que se ejecuta la tarea. Las propiedades de cada elemento que describen la tarea son los siguientes:

**[348 ]**



*Capítulo 12*



* display\_name: Un nombre legible por humanos para mostrar para esta tarea. Esto se utiliza en la interfaz de usuario para informar al usuario sobre el grado en que están.
* visualización: Indica al instalador si se muestra o no la tarea. Esto se utiliza para proporcionar un control preciso sobre la pantalla de la tarea en el instalador. Es útil para tareas que pueden aparecer sólo en determinadas condiciones. El valor es un valor booleano y el valor predeterminado es si no se establece la clave "display\_name".
* tipo: Especifica el tipo de tarea que se ejecute. Existen tres tipos de tareas de la instalador puede ejecutar. La normal tipo de tarea se ejecutará una función de devolución de llamada y opcionalmente devolver algo de HTML. La forma tipo lleva a un formulario utilizando el API de formularios. La lote Tipo devolverá una definición API lotes.
* ejecutar: Indica al instalador cuando para ejecutar la tarea. El ajuste predeterminado es

INSTALL\_TASK\_RUN\_IF\_NOT\_COMPLETED, Que cuenta la instalación para ejecutarlo la tarea una vez en el proceso de instalación cuando se había llegado en la lista de tareas de instalación. Alternativamente, INSTALL\_TASK\_RUN\_IF\_REACHED indica al instalador que ejecute

la tarea en cada carga de página si la tarea se alcanza en la lista de tareas y INSTALL\_TASK\_SKIP indica al instalador que no tome la tarea.

* función: Parámetro opcional para definir una función de devolución de llamada que no sea el nombre de clave. Esto es útil si desea llamar a la misma función de devolución de llamada en más de un paso.

**Creación de una tarea**

Vamos a crear dos tareas para el tienda perfil. Las tareas van a crear dos tipos de contenido y complete la configuración predeterminada de nuestro formulario de contacto en todo el sitio.

Comenzamos definiendo la tarea de crear los dos tipos de contenido. Esto se hará en store\_install\_tasks (), La puesta en práctica de los perfiles hook\_install\_tasks () que va en store.profile:

/ \*\*

\* Implementa hook\_install\_tasks (). \* /

store\_install\_tasks function () {$ tasks = array ();

$ tareas ['store\_create\_content\_types'] = array ('type' => 'normal'

);

return $ tareas;

}

**[349 ]**



*Perfiles de montaje*



En este caso hemos creado la clave de matriz para store\_create\_content\_types, Que es la función de devolución de llamada para esta tarea. La tipo está ajustado a normal, Es decir, esta es una tarea que se ejecuta, y puede devolver HTML.

Luego, en nuestro perfil, creamos la función store\_create\_content\_types () de la siguiente manera:

store\_create\_content\_types función (& $ install\_state) {$ tipo = array (

array (

'Type' => 'Página', 'name' => st ('Página'),

'Base' => 'node\_content',

'Description' => st ("Usar <em> páginas </ em> para el contenido estático, como una página" Acerca de nosotros ".")

'Custom' => 1, 'modificado' => 1, 'bloqueado' => 0,

),

array (

'Type' => 'producto', 'name' => er (el "producto"), 'base' => 'node\_content',

'Description' => st ('Use <em> productos </ em> para artículos de la tienda.'),

'Custom' => 1, 'modificado' => 1, 'bloqueado' => 0,

),

);

foreach ($ tipos como $ tipo) {

$ Type = node\_type\_set\_defaults ($ tipo); node\_type\_save ($ tipo); node\_add\_body\_field ($ tipo);

}

}

En este caso la función ha pasado una variable, que sea $ Install\_state. Esta es una variable pasada a través de todo el proceso de instalación y de cada una de las tareas. Contiene el estado y toda la información relevante que debe ser pasado alrededor del instalador.

El contenido de esta función define dos tipos de contenido. También se asegura de que toda la información por defecto se rellena, y guarda los tipos, creando dos nuevos tipos de contenido.

**[350 ]**



*Capítulo 12*



A continuación, vamos a crear un formulario para introducir la información de contacto predeterminada para el formulario de contacto en todo el sitio. En este caso ampliamos la store\_install\_tasks () funcionar para agregar una tarea para el formulario.

/ \*\*

\* Implementa hook\_install\_tasks (). \* /

store\_install\_tasks function () {$ tasks = array ();

$ tareas ['store\_create\_content\_types'] = array ('type' => 'normal'

);

**$ tareas ['store\_configure\_contact\_form'] = array ('display\_name' => t ('información de contacto del sitio Default'), 'type' => 'form',**

**);**

return $ tareas;

}



En este capítulo se dará cuenta de la utilización de la t () y st () funciones para traducciones. La t () función es la función estándar a utilizar para el texto traducible. La st () función debe utilizarse cuando algunos de los sistemas no están disponibles. Por ejemplo, en las tareas del programa de instalación antes de instalar los módulos. Si hay funciones que necesita para ser llamado en ambos lugares y que necesita para descubrir qué función usar, get\_t () devuelve el nombre de la función de traducción apropiado utilizar.



Después de la tarea store\_create\_content\_types, La tarea store\_configure\_contact\_ forma se añade. Ofrecemos esta tarea con un display\_name por lo que se mostrará como un paso en el instalador. La tipo se establece en un formulario. Al igual que cualquier otra forma de Drupal, tendrá acceso a FORM\_ID\_validate y FORM\_ID\_submit devoluciones de llamada.

Una vez que un perfil se ha seleccionado en el primer paso de la instalación, cualquier tarea adicional se visualizará usando su display\_name en los pasos del instalador.

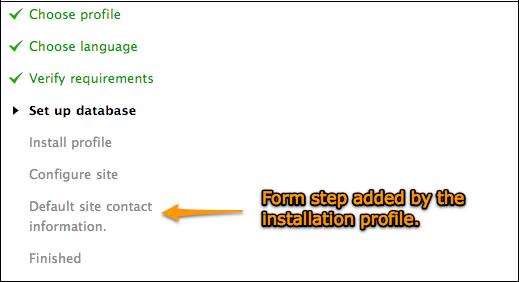
**[351 ]**



*Perfiles de montaje*



La función forma se verá un poco diferente a una función normal formulario:



store\_configure\_contact\_form función ($ form, & $ form\_state, & $ install\_ estado) {

drupal\_set\_title (t ('información de contacto sitio Default'));

$ Form ['destinatarios'] = array ('# type' => 'textarea', '# title' => t ('destinatarios'),

'# Default\_value' =>'',

'# Description' => t ("Ejemplo:. '. Webmaster@example.com' o 'sales@example.com, support@example.com' Para especificar varios destinatarios, separe cada dirección de correo electrónico con una coma") ,

'# Required' => TRUE,

);

$ = Array ('# type' => 'textarea', '# title' => t ('Auto-respuesta'), '# default\_value' =>'' form ['respuesta'],

'# Description' => t ('respuesta automática opcional. Deje en blanco si no desea enviar al usuario un mensaje de respuesta automática.'),

);

$ Form ['submit'] = array ('# type' => 'enviar', 'valor #' => t ('Save'),

);

devolver $ form;

}

**[352 ]**



*Capítulo 12*



Tenga en cuenta que $ Install\_state se pasa como una tercera variable en la función de la forma. Esto es diferente de las funciones de forma típica en la API de formularios. Proporciona acceso a la información estatal y en el instalador.



**Ajuste del título de la página**

Ajuste de la display\_name para una tarea sólo establece el título en la lista de tareas del instalador. Para definir el título de la página, utilice drupal\_set\_title (). Para más detalles sobre el uso de drupal\_set\_title () consulte la documentación de la API en

http://api.drupal.org/api/function/drupal\_set\_title/7.



Al igual que otras funciones de la API de forma que tenemos acceso a la store\_configure\_

contact\_form\_validate () y store\_configure\_contact\_form\_submit () funciones, como se ve a continuación:

store\_configure\_contact\_form\_validate función ($ form, & $ form\_state) {/ / Validar y cada destinatario de correo electrónico.

$ destinatarios = explode (',', $ form\_state ['valores'] ['destinatarios']); foreach ($ destinatarios como & $ recipient) {

$ Recipient = trim ($ recipient);

if (! valid\_email\_address ($ recipient)) {form\_set\_error ('beneficiarios', t ('% destinatario es un inválido e-mail

. dirección ', array ('% destinatario '=> $ recipient)));

}

}

form\_set\_value ($ form ['destinatarios'], implosión (',', $ destinatarios), $ form\_state);

}

store\_configure\_contact\_form\_submit función ($ form, & $ form\_state) {valor $ = $ form\_state ['valores'];

$ valores + = array ('cid' => 1,

);

drupal\_write\_record ('contacto', los valores $, array ('cid')); watchdog ('contacto', 'La categoría por defecto ha sido actualizado.',

array (), WATCHDOG\_NOTICE, l (t ('Editar'), 'admin / estructura / contacto / editar /'

. valores de $ ['cid']));

}

**[353 ]**



*Perfiles de montaje*



Hay algunas tareas que se deben ejecutar después de un cierto punto en el proceso en cada carga de página. Un ejemplo de esto se encuentra en la instalación por defecto tareas. El instalador se inicializa con un bajo nivel de arranque Drupal permite la funcionalidad de nivel muy básico.

Después de que los módulos se han instalado, pero antes de la parte de configuración del sitio del proceso, hay un punto en el que el instalador puede hacer un arranque completo Drupal con toda la funcionalidad de todos los módulos activados. Una vez que el instalador llega a este punto se ejecuta una tarea de todas las cargas de páginas posteriores para realizar un arranque completo Drupal. La definición de esta tarea tiene el siguiente aspecto:

tareas $ = array (

...

'Install\_bootstrap\_full' => array ('run' => INSTALL\_TASK\_RUN\_IF\_REACHED,

),

...

);

Al establecer ejecutar a INSTALL\_TASK\_RUN\_IF\_REACHED, La función install\_bootstrap\_ completo () se ejecuta en cada carga de página después de que se encontró en las tareas de instalación.

**La alteración de las tareas**

El programa de instalación proporciona un método para alterar las tareas en el instalador. Donde hook\_install\_tasks () proporciona un método para agregar tareas al final de la instalación proceso, hook\_install\_tasks\_alter () proporciona acceso a todas las funciones del programa de instalación (incluyendo las tareas predeterminadas proporcionadas por el instalador). Esto permite un perfil para insertar tareas más temprano en el proceso de instalación o alterar las tareas predeterminadas.



Las tareas realizadas antes de la install\_bootstrap\_full () tarea, que es antes de instalar los módulos, sólo se tiene acceso a un nivel muy básico de

Configuración de Drupal, con acceso al módulo del sistema, módulo de usuario, el



Configuración de PHP, y un sitio de settings.php presentar.

Veamos un ejemplo de la alteración de las tareas antes de instalar los módulos; sólo el sistema de nivel de base está disponible, lo que significa que su perfil de instalación aún no es capaz de llamar a hook\_form\_alter o hook\_form\_FORM\_ID\_alter. En este caso queremos alterar el paso en el que la base de datos está configurado para agregar algunas instrucciones adicionales.

/ \*\*

\* Implementa hook\_install\_tasks\_alter (). \* /

store\_install\_tasks\_alter función (& $ tareas, & $ install\_state) {$ tareas ['install\_settings\_form'] ['función'] store\_database\_ = '

settings\_form ';

}

**[354 ]**



*Capítulo 12*



En este caso, la tarea que pone en marcha la base de datos se altera, diciéndole que se llame a una función diferente, utilizando la función clave para la tarea.

store\_database\_settings\_form función ($ form, & $ form\_state, & $ install\_ estado) {

/ / Recuperar el formulario predeterminado.

$ Default\_form = install\_settings\_form ($ form, & $ form\_state, & $ install\_state);

$ Default\_form ['basic\_options'] ['base de datos'] ['# description'] = st ('Si elige usar SQLite proporcione una ruta absoluta fuera del directorio web raíz.').;

return $ default\_form;

}

En esta función de devolución de llamada de trabajo, recuperamos el formulario predeterminado proporcionado por Drupal, alterarlo, y usar ese formulario. Ya que estamos usando una forma de respuesta diferente, tenemos que proporcionar \_Validate y \_submit manejadores para procesar la información presentada en el formulario. Las dos funciones siguientes envuelven la \_Validate y \_submit funciones

utilizado para install\_settings\_form ():

/ \*\*

\* API Validar formulario de forma store\_database\_settings\_form. \* /

función store\_database\_settings\_form\_validate ($ form, & $ form\_state) {install\_settings\_form\_validate ($ form, & $ form\_state);

}

/ \*\*

\* API de envío de formularios de forma store\_database\_settings\_form. \* /

función store\_database\_settings\_form\_submit ($ form, & $ form\_state) {install\_settings\_form\_submit ($ form, & $ form\_state);

}

**Configuración de los bloques**

Fuera de la caja, Drupal no tiene ningún bloques configurados para mostrarse. Un perfil de la instalación tendrá que habilitar y configurar los bloques que quiere mostradas. En este caso vamos a utilizar hook\_install () en un store.install presentar como parte de nuestra instalación. Cuando se instalan los módulos para el perfil, la función store\_install () se ejecutará, la configuración de los bloques. La siguiente es una básica store\_install () función que permite tres bloques en el tema Bartik:

función store\_install () {

/ / Habilitar algunos bloques estándar. valores $ = array (

array (

**[355 ]**



*Perfiles de montaje*



=> 'Sistema' 'módulo', 'delta' => 'tema' 'main' => 'Bartik', 'Estado' => 1, 'peso' => 0, 'región' => 'contenido', 'páginas' =>'',

'Cache' => DRUPAL\_NO\_CACHE,

),

array (

=> 'Usuario' 'módulo', 'delta' => 'login', 'lema' => 'Bartik', 'Estado' => 1, 'peso' => 0,

'Región' => 'sidebar\_first', 'páginas' =>'',

'Cache' => DRUPAL\_NO\_CACHE,

),

array (

=> 'Sistema' 'módulo', 'delta' => 'gestión', 'lema' => 'Bartik', 'Estado' => 1,

'Peso' => 1,

'Región' => 'sidebar\_first', 'páginas' =>'',

'Cache' => DRUPAL\_NO\_CACHE,

),

);

$ Query = db\_insert ('block') -> campos (array ('module', 'delta', 'tema', 'Estado', 'peso', 'región', 'páginas', 'cache'));

foreach ($ valores como $ record) {$ query-> valores ($ registro);

}

$ Query-> execute ();

}

No existe una función de la API para permitir a los bloques. Aquí tenemos que añadir filas a la tabla de bloques para cada bloque que queremos configurar. En este caso, hacemos posible que el contenido principal, el formulario de acceso, y el bloque de administración del sitio.

**[356 ]**



*Capítulo 12*



Cada fila que se agrega a la tabla de bloque contiene la siguiente información sobre el bloque:

* módulo: El módulo que posee el bloque.
* delta: El nombre del bloque dentro del módulo.
* tema: El tema del bloque mostrará dentro. Cada tema tiene una configuración del bloque individual.
* estado: Cuando se establece en 1, se muestra el módulo.
* peso: Se utiliza para ordenar el bloque dentro de una región. Pesos más grandes son se muestra a continuación pesos más pequeños.
* región: La región de la página que el bloque se mostrará dentro.
* páginas: Cualquier regla sobre las páginas que el bloque se mostrará sucesivamente.
* cache: ¿Cómo se almacenan en caché el bloque.



En Drupal 7, las barras laterales se cambió el nombre para un mejor soporte multi-idioma. Lo que solía ser la barra lateral izquierda es ahora sidebar\_first y lo que solía ser la barra lateral derecha es ahora sidebar\_second.



**Configuraciones variables**

Perfiles de instalación a menudo quieren establecer su propia configuración de variables, que son fuera de la caja, aparte de las previstas por el núcleo de Drupal. Drupal proporciona lavariable\_set () función que se puede utilizar en una tarea de instalación o en hook\_install () para establecer una variable. Por ejemplo, añadir las siguientes líneas astore\_install () establecerá el admin\_theme con el tema Seven:

función store\_install () {

...

/ / Establecer el tema de administración a siete. variable\_set ('admin\_theme', 'siete');

}

**[357 ]**



*Perfiles de montaje*



**Filtros de texto**

Filtros de texto, conocidos como formatos de entrada en las versiones anteriores de Drupal, se crean y configuran el perfil de instalación. Para que un sitio para tener filtros de texto que los usuarios pueden elegir, el perfil de instalación debe configurar el filtro primero. El método más simple para la que fueron creados en elhook\_install () función dentro del perfil de . Instalar presentar. Siguiendo con este ejemplo, el siguiente código en la parte superior de store\_install () añadiría Filtered HTML y Full HTML filtros de texto, con la filtra un conjunto como el valor por defecto:

/ / Añadir los formatos de texto. $ Filtered\_html\_format = array (

'Formato' => 'filtered\_html', 'name' => 'Filtered HTML', 'peso' => 0,

'filtros' => array (filtro / / URL.

'Filter\_url' => array ('peso' => 0, 'status' => 1,

),

* Filtro de HTML. 'Filter\_html' => array (

'Peso' => 1, 'status' => 1,

),

* Filtro de salto de línea. 'Filter\_autop' => array (

'Peso' => 2, 'status' => 1,

),

* Filtro corrector de código HTML. 'Filter\_htmlcorrector' => array (

'Peso' => 10, 'status' => 1,

),

),

);

$ Filtered\_html\_format = (objeto) $ filtered\_html\_format; filter\_format\_save ($ filtered\_html\_format);

$ Full\_html\_format = array ('format' => 'full\_html', 'name' => 'Full HTML', 'peso' => 1,

'filtros' => array (

**[358 ]**



*Capítulo 12*



/ / Filtro de URL. 'Filter\_url' => array (

'Peso' => 0, 'status' => 1,

),

* Filtro de salto de línea. 'Filter\_autop' => array (

'Peso' => 1, 'status' => 1,

),

* Filtro corrector de código HTML. 'Filter\_htmlcorrector' => array (

'Peso' => 10, 'status' => 1,

),

),

);

$ Full\_html\_format = (objeto) $ full\_html\_format; filter\_format\_save ($ full\_html\_format);

**Colocación Código**

Código colocado en el . Perfil archivo se cargará en cada carga de página. Esto significa que después de la instalación de Drupal, las tareas de instalación contenidas en la . Perfil archivo para la instalación se ha cargado todo el tiempo. Dado que las tareas de instalación sólo se utilizarán durante el instalador, que no necesitan ser cargados después de eso.

El primer paso es mover las tareas a un archivo diferente que se puede cargar sólo cuando sea necesario. En este caso vamos a crear un archivo llamadostore.install.inc en el perfil de instalación y seguir las siguientes funciones de instalación de tareas en él:

store\_configure\_contact\_form () store\_configure\_contact\_form\_validate () store\_configure\_contact\_form\_submit () store\_database\_settings\_form () store\_database\_settings\_form\_validate () store\_database\_settings\_form\_submit () store\_create\_content\_types ()

**[359 ]**



*Perfiles de montaje*



El perfil tendrá que cargar este archivo durante el proceso de instalación para tener acceso a las tareas y formularios. Para esta actualización store\_install\_tasks\_alter () necesidades para añadir una llamada a incluir este archivo como se muestra aquí:

/ \*\*

\* Implementación de hook\_install\_tasks\_alter (). \* /

función store\_install\_tasks\_alter (& $ & $ de tareas, install\_state) {

...

/ / Incluir store.install.inc que contiene las tareas y formularios. include\_once 'store.install.inc';

}

Esto podría ser añadido a hook\_install\_tasks () así.

**Ejecutar el instalador desde la línea de comandos**

El instalador puede ejecutarse desde ambos, un navegador como un asistente, así como una parte de un script de instalación externa. Por ejemplo, el instalador puede ser utilizado como una parte de un sistema que crea automáticamente nuevos sitios. Esto es útil para los casos en que el

instalación se puede automatizar. Por defecto, el instalador se ejecuta en modo interactivo en el que se ejecuta como un asistente interactivo. El sistema es capaz de funcionar en un modo no-interactivo en el que se pasan la configuración en forma de matriz que precisar los detalles de cada paso. Por ejemplo, consulte el siguiente script que instala el sitio utilizando el perfil predeterminado:

<? Php

/ / Los ajustes para el instalador a utilizar al instalar Drupal $ settings = array (

* Esto anula la matriz PHP $ \_SERVER para que podamos decirle a
* Drupal la ruta de acceso al sitio. Esto es importante para los
* soporte multi-sitio.

'Servidor' => array (

'HTTP\_HOST' => 'localhost', / / ​​El nombre de dominio. 'SCRIPT\_NAME' =>'', / / ​​La ruta de acceso al sitio.

),

/ / Seleccionar el perfil y la configuración regional. 'parámetros' => array (

'Perfil' => 'mínimo',

**[360 ]**



*Capítulo 12*



'Locale' => 'en',

),

/ / Los valores a utilizar en cada uno de los formularios. 'formularios' => array (

'Install\_settings\_form' => array ('conductor' => 'sqlite', 'database' => 'test', 'username' =>'',

'Password' =>'', 'anfitrión' => 'localhost', 'puerto' =>'', 'db\_prefix' =>'',

),

/ / El formulario de configuración del sitio. 'Install\_configure\_form' => array (

'Site\_name' => 'Drupal Sitio', 'site\_mail' => 'email@example.com', 'cuenta' => array (

'Name' => 'admin',

'Mail' => 'email@example.com', 'pass' => array (

* En el formulario hay dos campos de contraseña. La
* instalador está llenando el formulario por lo que necesitamos
* Complete ambos campos de formulario.

'Pass1' => 'password', 'PASS2' => 'password',

),

),

* El país predeterminado y la zona horaria. 'Site\_default\_country' => 'EE.UU.', 'date\_default\_timezone' => 'America / Detroit,
* Habilitar URLs limpias.

'Clean\_url' => TRUE,

* Buscar actualizaciones utilizando el gestor de actualización.
* Los valores posibles son:
* - Array () = OFF,
* - Array (1) = comprobar si hay actualizaciones,
* - Array (1, 2) = comprobar si hay actualizaciones y notificar por
* email

**[361 ]**



*Perfiles de montaje*



'Update\_status\_module' => array (1, 2),

),

),

);

/ \*\*

\* El directorio raíz de instalación de Drupal. \* /

define ('DRUPAL\_ROOT', getcwd ());

/ \*\*

\* Global bandera para indicar que el sitio está en modo de instalación. \* /

define ('MAINTENANCE\_MODE', 'instalar');

* Cargue el programa de instalación e iniciar el proceso de instalación utilizando
* $ configuraciones.

DRUPAL\_ROOT require\_once. '/ Includes / install.core.inc'; install\_drupal ($ ajustes);

Esta secuencia de comandos se inicia mediante la creación de un $ ajustes matriz que proporciona información para el instalador y los valores de las formularios dentro del instalador. Después de los ajustes se definen, el instalador (includes / install.core.inc) Está incluido y puesto en marcha por llamar install\_drupal () con el $ ajustes se pasa pulg

**Resumen**

Hemos completado la creación de un perfil de instalación. Empezamos mirando dónde almacenar módulos y temas asociados al perfil. Nos mudamos a crear el perfil de instalar, que incluía la creación de una nueva tarea y la alteración de una tarea se incluye con el núcleo de Drupal.

En el camino, nos fijamos en cómo crear tipos de contenido, active bloques, y establecer variables.

Cerramos el capítulo de buscar la manera de optimizar la colocación de código y se veía en la forma de ejecutar una instalación de Drupal fuera el asistente de instalación basado en la web.

**[362 ]**



Base de datos Access

Aunque Drupal 7 ha hecho grandes saltos hacia adelante en cuanto a la flexibilidad de su almacenamiento de datos en la práctica, la gran mayoría de los sitios Drupal todavía dependen de una base de datos SQL para sus propios datos primarios y para la mayor parte de la configuración. En el pasado Drupal se ha basado en una capa muy delgada de base de datos que proporciona sólo la abstracción limitada más allá de la compatibilidad nativa de PHP para las bases de datos MySQL y PostgreSQL, que era una limitación grave para sitios complejos o de alto tráfico.

Drupal 7, sin embargo, cuenta con una nueva capa de base de datos de marca reescrito desde cero para proporcionar una amplia gama de características robustas en las áreas de seguridad, escalabilidad y flexibilidad desarrollador. Conocido tanto la lengua en la mejilla mientras

"Bases de datos: La nueva generación" o "DBTNG" durante el desarrollo, que ofrece soporte para muchas SQL características avanzadas, así como mejorado enormemente la portabilidad entre las bases de datos principales de SQL en el mercado. De hecho, Drupal 7 incluye ahora soporte para las tres bases de datos de código abierto líder (MySQL y variantes tales como Maria DB, PostgreSQL y SQLite) fuera de la caja y en fecha esta escritura complemento a los conductores están disponibles tanto para , Microsoft SQL Server y Oracle.

La API de base de datos está bien documentado, pero esta sección le proporciona una visión general de las principales características de la API de base de datos y la forma de utilizarlos para garantizar el código rápido, robusto. Vamos a suponer un conocimiento existente de SQL. Para obtener información más detallada consulte

http://drupal.org/developing/api/database y http://api.drupal.org/ api/group/database/7.

*Base de datos Access*



**Consultas básicas**

La mayoría de las consultas SELECT, en la práctica, bastante simple y no cambian. Drupal llama a estas consultas estáticas, y son muy fáciles de usar.

Por ejemplo, para obtener una lista de todos los módulos habilitados en el sistema, podríamos ejecutar la siguiente consulta:

$ Resultado = db\_query ("SELECT nombre, nombre FROM {sistema} DONDE type =: tipo y el estado =: status", array (': type' => 'módulo', ': status' => 1));



En la práctica, si queremos obtener esa información simplemente llamaríamos module\_list () en su lugar, pero para los propósitos de este ejemplo vamos a hacerlo de la forma manual.



La consulta se parece mucho a SQL normal que esperaríamos ver en ningún otro lugar, pero hay algunas cosas importantes a mencionar.

* Todos los nombres de tabla SQL se envuelven entre llaves. Que identifica a la cadena como nombre de la tabla a la capa de base de datos y permite que Drupal para añadir fácilmente un prefijo configurado para todas las tablas de una instancia de Drupal dado.
* No existe una sintaxis de MySQL específico (o cualquier sintaxis específica de base de datos) en cualquier parte de la consulta.
* No existen valores literales en la consulta. En cambio, los valores literales son especificados por los marcadores de posición. Los valores de los marcadores de posición se especifican en un array asociativo como segundo parámetro adb\_query ().

Esos marcadores son significativos. Ellos nos permiten separar la consulta de los valores de la consulta y pasarlos al servidor de base de datos por separado. El servidor de base de datos puede ensamblar los valores de cadena de consulta y de marcador de posición según sea necesario, con pleno conocimiento de qué tipo de datos tiene sentido en cada caso. Eso elimina la mayoría (aunque no todo) oportunidades para la inyección de SQL a partir de datos inesperados.

Hay otras tres cosas importantes que debe recordar acerca de los marcadores de posición:

* Los marcadores de posición deben ser únicos en una consulta, y deben comenzar con dos puntos.
* Los marcadores de posición no deben tener comillas alrededor de ellos, sin importar el tipo de datos. El servidor de base de datos se encargará de eso por nosotros.
* Los marcadores de posición se deben utilizar para todos los datos literales, incluso si no va a variar.

Este tercer punto es importante para la portabilidad entre bases de datos, como separación de valores literales permite a los conductores de bases de datos para hacer optimizaciones específicas de la base de datos.

**[364 ]**



*Apéndice A*



**Objetos de Resultados**

El valor de retorno de un db\_query () llamada es objeto de resultado. Con el fin de acceder a los datos devueltos por el servidor de base de datos, tenemos que iterar sobre el conjunto de resultados.

$ List = array ();

foreach ($ resultado as $ record) {

$ Lista [] = t ('@ name: @ nombre de archivo', array ('@ name' => $ nombre de registro->, '@ nombre' => $ record-> nombre de archivo,

));

}

Por defecto, cada $ Record en el conjunto de resultados es un stdClass objeto. Podemos, sin embargo, obtener el registro como una matriz asociativa contando la db\_query () llamamos a eso queremos arrays lugar. Así es como se hace:

$ Resultado = db\_query ("SELECT nombre, nombre FROM {sistema} DONDE type =: tipo y el estado =: status", array (': type' => 'módulo', ': status' => 1), array (' buscar '=> PDO :: FETCH\_ASSOC));

Aquí, especificamos un tercer parámetro a db\_query (), Que es otro array asociativo de opciones. Se especifica sólo una opción, el modo de búsqueda, el cual nos pusimos a PDO :: FETCH\_ASSOC. Esto le indica a la capa de base de datos que queremos matrices asociativas en lugar de stdClass objetos.

También podemos buscar un registro único, o incluso un solo campo:

* Obtener un registro único como un objeto. $ Record = $ result-> fetchObject ();
* Obtener un registro único como una matriz. $ Record = $ result-> fetchAssoc ();
* Recuperar sólo el primer campo del siguiente registro. $ Campo = $ result-> FetchField ();
* Obtener el conjunto de resultados a la vez en una matriz. $ registros = $ result-> fetchAll ();

Consulte la documentación en línea para obtener más información acerca de varias formularios de recuperar datos de un conjunto de resultados.

**[365 ]**



*Base de datos Access*



**Las consultas dinámicas**

Aunque la mayoría de las consultas SELECT son estáticas, a veces vamos a necesitar una consulta más flexible. Eso podría deberse a que la propia consulta puede cambiar dependiendo de los datos de usuario de entrada, porque queremos permitir a otros módulos para modificar nuestra consulta antes

se ejecuta, o queremos aprovechar algunas características de base de datos que se implementa de manera diferente en diferentes bases de datos. Para estos casos, Drupal proporciona un mecanismo para la creación de consultas dinámicas utilizando un generador de consultas robusto.

Para empezar, creamos un nuevo objeto de consulta con db\_select ():

$ Query = db\_select ('nodo', 'n');

El primer parámetro es el nombre de la tabla de base de la consulta y el segundo es el alias que queremos usar para ello. Luego llamamos a métodos adicionales en la $ Consulta objeto con el fin de construir la lógica de consulta que queremos crear de forma dinámica. Por ejemplo:

$ Query = db\_select ('nodo', 'n'); $ query-> campos ('n', array ('nid, título'));

$ U\_alias = $ query-> innerJoin ('usuarios', 'u', '% alias.uid = n.uid'); $ Query-> AddField (u\_alias $, 'nombre', 'username'); $ Query-> condición (". {$} U\_alias nombre", "Bob"); $ Query-> condición ('n.created', REQUEST\_TIME - 604800, '> ='); $ Query-> orderBy ('n.created', 'DESC');

$ Gama consulta-> (0, 5); $ Query-> addtag ​​('node\_access'); $ Resultado = $ query-> execute ();

La campos () método cuenta la consulta para seleccionar los campos en el segundo parámetro de la tabla en el primer parámetro. En este caso, a partir de la línea 2, nuestra consulta sería efectivamente leer:

SELECT n.nid AS nid, n.title AS titulo FROM {node} n

Tenga en cuenta que las llaves se añaden para nosotros de forma automática. También tenga en cuenta que los alias se crean para cada campo. Si queremos especificar un alias alternativa, tenemos que utilizar laaddField () método para que un campo. Veremos más sobre esto en breve. También puede unirse en contra de otras tablas utilizando elinnerJoin (),leftJoin (), Y rightJoin () métodos. También hay join (), Que es un alias de innerJoin (). La join () métodos toman la mesa para unirse en contra, su alias, y se unen las condiciones como parámetros en la forma de un fragmento de SQL. Tenga en cuenta que en este caso estamos usando la cadena % alias en la cláusula de combinación. Eso es porque mientras nos unimos en contra de la tabla de usuarios y pidiendo un alias de "u", podríamos terminar con un alias diferente si ese alias ya existe en esta consulta. Aunque estamos bastante seguros de que no es el caso aquí, podría ser el caso en query\_alter () hooks, así que es un buen hábito para entrar.

**[366 ]**



*Apéndice A*



La join () métodos todos vuelven el alias de la tabla que se utiliza realmente, así que podemos usar esa en llamadas a métodos posteriores. En este caso también vamos a seleccionar un campo de la tabla de usuarios, el nombre del usuario y alias a "nombre de usuario". Una vez más, ya que hay una ligera posibilidad de que el alias ya se podía utilizar, addField () devolverá el alias que realmente se utilizó para ese campo.

Nuestra consulta efectiva ahora se ve así:

SELECT n.nid AS nid, n.title AS título, u.name AS nombre\_usuario FROM {node} n INNER JOIN {usuarios} U en u.nid = n.nid

Ahora tenemos que restringir la consulta, es decir, añadir el DONDE cláusulas. Esto se hace con la condición () método, que lleva el nombre del campo, el valor a contrastar, y, opcionalmente, un comparador. El valor predeterminado es iguales. Las líneas de arriba, por lo tanto, añaden DONDE cláusulas de un nombre de usuario de 'Bob' y un tiempo de creación de nodos dentro de la última semana (es decir, donde la marca de tiempo de creación es mayor o igual a la hora actual menos el valor de segundo de siete días). Para más condicionales complejas también hay un método () cuando la toma un fragmento SQL.

Entonces nos decimos a la consulta iniciada por el tiempo de creación, en orden descendente (DESC) y para devolver sólo cinco resultados empezando por el registro 0, es decir, los cinco nodos de más reciente creación. Nuestra consulta SQL ahora se ve algo como esto:

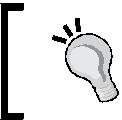
SELECT n.nid AS nid, n.title AS título, u.name AS nombre\_usuario FROM {node} n

INNER JOIN {usuarios} U en u.nid = n.nid WHERE (n.created> = 1286213869)

Y (u.name = 'Bob') ORDER BY DESC LIMIT n.created 5 OFFSET 0

Hay un método más importante llamar-addtag ​​(). Este método no afecta a la consulta directamente pero no marcar el tipo de consulta que es. Si una consulta se ha etiquetado a continuación, antes de que se convierte en una cadena SQL que se pasa a través de hook\_query\_alter () y hook\_query\_TAG\_alter (). Eso permite que otros módulos

una oportunidad para cambiar la consulta si es necesario. La node\_access etiqueta, que se utiliza aquí, es lo más importante, ya que permite que el sistema de acceso de nodo para modificar la consulta, para filtrar los nodos que el usuario actual no debería tener acceso.



Cuando se consulta la tabla de nodos, siempre nos una consulta dinámica con elnode\_access tag. Si no lo hace, entonces usted tiene un agujero de seguridad.

**[367 ]**



*Base de datos Access*



Finalmente ejecutamos la consulta. execute () toma la información que hemos proporcionado, ejecuta la consulta a través de hooks alter si es necesario, compila la cadena SQL correspondiente, y se ejecuta la consulta, devolviendo un objeto de resultado. El objeto de resultado es el mismo que el de regresar de una consulta estática.

También tenga en cuenta que la mayoría de los métodos de la selecta constructor devuelven el propio objeto de selección y por lo tanto son conexión en cadena. Las excepciones son eladdField () y join () métodos, como los que tienen que devolver un alias generados en su lugar. La consulta anterior podría, por tanto, también se han escrito como:

$ Query = db\_select ('nodo', 'n');

$ U\_alias = $ query-> innerJoin ('usuarios', 'u', '% alias.uid = n.uid'); $ Query-> AddField (u\_alias $, 'nombre', 'username');

$ Resultado = $ consulta

-> Campos ('n', array ('nid, título')); -> Condición (". {$} U\_alias nombre", "Bob"); -> Condición ('n.created', REQUEST\_TIME - 604800, '> ='); -> OrderBy ('n.created', 'DESC');

-> Rango (0, 5); -> Addtag ​​('node\_access') -> execute ();

El generador de consultas es capaz de crear consultas más complejas, también, incluyendo subconsultas, complejos AND y OR condiciones, entre otros. Consulte la documentación en línea para los detalles completos.

**Inserte consultas**

Aunque sólo consultas SELECT tienen versiones tanto estáticas como dinámicas, INSERT, UPDATE, DELETE, y MERGE consultas apoyar una versión dinámica. Esto es necesario a fin de apoyar la gama de bases de datos SQL en la naturaleza, muchos de los cuales requieren un manejo especial extra en algunos tipos de campo. Como un buen bono, la versión dinámica de estas consultas es a menudo mucho más fácil de trabajar que la versión estática sería y hace que sea fácil de añadir optimizaciones específicas de base de datos adicionales. Pronto nos ocuparemos de consultas Insert primero.

Al igual que con consultas de selección, consultas Insert comienzan con una función constructora y son conexión en cadena. De hecho, todos los métodos de las consultas de inserción son conexión en cadena.

$ Id = db\_insert («importaciones») -> campos (array (

'Name' => 'Groucho',

'Dirección' => '123 Casablanca Avenida. ',' Teléfono '=> '555-1212',

))

-> Execute ();

**[368 ]**



*Apéndice A*



La db\_insert () método crea un nuevo objeto de consulta de inserción para el importaciones tabla. Luego llamamos a la campos () método en ese objeto. campos () toma una matriz asociativa de valores a insertar. En este caso, estamos agregando un registro de uno de los grandes cómicos del mundo. A continuación, ejecutar la consulta, haciendo que se puede traducir en la cadena de consulta apropiado y ejecutado. Si hay un-incremento automático o en el campo "en serie" en la tabla de las importaciones, se devolverá el identificador generado. Eso es todo lo que hay que hacer.

db\_insert () puede obtener más de lujo, también. Por ejemplo, es compatible con las declaraciones de varios de inserción. Para hacer eso, primero tenemos que llamar campos () con una matriz indexada para especificar qué campos vamos a utilizar y luego llamar a la values ​​() método repetidamente con

una matriz asociativa para cada registro.

$ valores [] = array ('nombre' => 'Groucho',

'Dirección' => '123 Casablanca Avenida. ',' Teléfono '=> '555-1212',

);

valores de $ [] = array ('nombre' => 'Chico',

'Dirección' => '456 razas St. ',' teléfono '=> '555-1234',

);

$ valores [] = array ('nombre' => 'Harpo',

'Dirección' => '789 Horn Ave. ',' Teléfono '=> '555-1234',

);

$ valores [] = array ('nombre' => 'Zeppo',

'Dirección' => '22 University Way ',' teléfono '=> '555-3579',

);

$ Inserte = db\_insert («importaciones») -> campos (array ('nombre', 'Dirección', 'teléfono' => '555-1212 '));

foreach (valores $ como $ valor) {$ insert-> valores ($ valor);

}

$ Insert-> execute ();

En las bases de datos que soportan las declaraciones de varios de inserción, el código anterior se llevará a cabo en una sola consulta. Para aquellos que no lo hacen, van a ejecutar consultas separadas en una sola transacción. Eso los hace extremadamente potente y eficiente para las operaciones de importación masiva. Tenga en cuenta que en una consulta de múltiples inserto el valor de retorno deexecute () no está definido y debe ser ignorado.

**[369 ]**



*Base de datos Access*



**Consultas de actualización**

Consultas de actualización se parecen un híbrido de Insertar y Seleccionar declaraciones. Consisten en ambos campos para establecer sobre una mesa y condiciones para restringir la consulta.

db\_update («importaciones») -> condición ('nombre', 'Chico')

-> Campos (array ('dirección' => 'Go West St.')) -> execute ();

La condición () método funciona exactamente igual que el condición () método de consultas de selección, y podemos añadir múltiples si es necesario. campos () toma una matriz asociativa de valores para definir. Por consiguiente, la consulta anterior es equivalente a:

ACTUALIZACIÓN {} importaciones dirección SET = 'Go West St.' WHERE nombre = 'Chico';

Seguimos siempre queremos utilizar el enfoque dinámico en lugar de simplemente llamar db\_query (), Porque en algunas bases de datos (como PostgreSQL o Oracle) hay casos en que la consulta anterior no funcionaría y que tendríamos que ejecutar varias consultas con los valores consolidados y otros casos de borde extraño. Todos los que la manipulación es manejado por nosotros de forma automática por la capa de base de datos.

El valor de retorno de execute () para consultas de actualización es el número de registros que se han cambiado por la consulta. Tenga en cuenta que 'cambiado' no significa 'coincidente'. Si el DONDE parte de la consulta coincide con un registro pero si ese disco ya tiene los valores que se establece en, no se cambiará y no contarán para el valor de retorno de execute ().

**Eliminar consultas**

Eliminar consultas no debería ser una sorpresa, ya que consisten esencialmente sólo unDONDE cláusula:

db\_delete («importaciones») -> condición ('name' => 'Zeppo') -> execute ();

El valor de retorno de execute () para la consulta de eliminación es el número de registros que se han eliminado por la consulta.

**Combinar consultas**

Combinar consultas son una de las partes a veces olvidadas, de SQL. En parte, eso se debe a que las bases de datos de código abierto más populares no apoyan directamente a pesar de que son parte de la especificación de SQL.

**[370 ]**



*Apéndice A*



Una consulta de combinación, dice, en esencia: "Si existe este registro, actualizar con esta consulta lo contrario créelo con esta otra pregunta". La sintaxis es un poco prolijo, pero es un concepto muy poderoso. Es muy útil para establecer registros que pueden o no existir todavía, es decir, los datos que se fusionan en la tabla. También puede ser muy útil para incrementar los contadores.

Una verdadera consulta de combinación es atómica, es decir, se tiene la garantía de que va a funcionar como una sola operación ininterrumpida o fallar por completo. Dado que la mayoría de las bases de datos de Drupal funciona con no apoyan directamente Combinar consultas, Drupal ellos emula con varias consultas y transacciones, que en la mayoría de los casos es lo suficientemente cerca.

La sintaxis debe ser familiar con base en las otras consultas que hemos visto ya. El siguiente ejemplo es directamente de la variable del sistema:

db\_merge ('variable') -> clave (array ('nombre' => $ nombre))

-> Campos (array ('value' => serialize ($ valor))) -> execute ();

La tecla () método toma un array asociativo de pares campo / valor que son el eje de la consulta. La campos () método es acerca de los campos de configurar y funciona del mismo modo que lo hace en UpdatE o Insertar consultas. La consulta anterior se puede leer como "Si hay un disco en el que el 'nombre' de campo tiene el valor $ Name, Defina el campo "valor". Si no, insertar un nuevo registro con el nombre igual a$ Name y un valor igual a la cadena dada. "(¿No es la consulta anterior de manera mucho más fácil de decir?)

También podemos definir una lógica más compleja mediante el insertFields () y updateFields () métodos. Aquellos funciona exactamente igual que campos () pero, como hemos podría esperar, sólo se aplican cuando se toman las de inserción o actualización ramas.

db\_merge ("pueblo") -> clave (array ('job' => 'Speaker')) -> insertFields (array (

'Edad' => 31,

'Name' => 'Meredith',

))

-> UpdateFields (array ('nombre' => 'Tiffany',

))

-> Execute ();

En este caso, si ya existe un registro cuyo campo de trabajo es "Altavoz" su campo de nombre se actualizará para Tiffany. Si no es así, un nuevo récord se creará con 'trabajo' como altavoz y 'nombre' como Meredith. (Sí, este ejemplo es bastante artificial.)

**[371 ]**



*Base de datos Access*



**Asignaturas avanzadas**

Mientras que los cinco tipos básicos de consultas cubren la gran mayoría de nuestras necesidades de base de datos, hay otros dos temas avanzados que debemos cubrir: Transacciones y replicación maestro / esclavo. No hacemos más que tocar en cada brevemente.

**Transacciones**

Una transacción en una base de datos es una forma de envolver dos o más consultas entre sí y declaran que deben ser atómicas. Es decir, ya sea todo éxito o no tener éxito. Eso puede ser muy útil cuando, por ejemplo, el ahorro de un nodo; no queremos sólo algunos campos para que se escriba y luego un error de romper el proceso a medio camino. Vimos un ejemplo de ello en el capítulo 6. En pocas palabras, en Drupal empezamos una transacción mediante la creación de un objeto de transacción. Todo lo que hacemos a la base de datos es entonces parte de la transacción hasta que se destruye ese objeto, momento en el que toda la consulta se ha comprometido a la vez. En la mayoría de los casos, dejamos PHP destruir el objeto de transacción para nosotros cuando una función termina.

función mi\_funcion () {$ transaction = db\_transaction (); try {

$ Id = db\_insert ('ejemplo') -> campos (array (

=> 'Mystring' 'campo1', 'campo2' => 5,

))

-> Execute (); my\_other\_function ($ id); return $ id;

}

catch (Exception $ e) {$ transaction-> rollback (); watchdog\_exception ('tipo', $ e);

}

}

En la mayoría de los casos no necesitamos una transacción, y no tienen ningún efecto sobre consultas de selección.

Son más útiles cuando vamos a estar ejecutando muchas consultas de modificación de datos (INSERT, UPDATE, DELETE, y fusión) juntos, y el sistema se romperán si sólo algunos de ellos se ejecutan. Las importaciones, las reconstrucciones de las tablas de búsqueda, y otros módulos que se pueden ejecutar consultas a través de hook en el centro de nuestro proceso son buenos candidatos.

**[372 ]**



*Apéndice A*



**Los servidores esclavos**

Drupal también es compatible con la replicación de bases de datos maestro / esclavo. Algunas consultas de selección se pueden ejecutar en un servidor esclavo para descargar el trabajo a los servidores separados, permitiendo el sitio para escalar mejor. Sin embargo, la mayoría de los sitios no tienen un servidor esclavo configurado, así que ¿cómo podemos escribir código que funcione correctamente en ambos sentidos?

Drupal nos permite hacerlo mediante la especificación de "objetivos". El tercer parámetro a db\_query () o db\_select () es un conjunto de opciones que ajustar el comportamiento de la consulta. Vimos el ir a buscar clave anterior. La otra clave de interés es objetivo, Que especifica que la base de datos la variante del sistema debe tratar. Los valores permitidos son defecto (Que es el valor predeterminado) y esclavo. Si se especifica "esclavo", Drupal intentará ejecutar la consulta en un servidor esclavo. Si uno no está disponible, sin embargo, será en silencio caerá de nuevo a usar el servidor por defecto. Usando nuestro ejemplo anterior:

$ Resultado = db\_query ("SELECT nombre, nombre FROM {sistema} DONDE type =: tipo y el estado =: status", array (': type' => 'módulo', ': status' => 1), array (' buscar '=> PDO :: FETCH\_ASSOC,' target '=>' esclavo '));

La manera en que manejamos la consulta es idéntica.

Así que ¿por qué no todas las consultas de selección utilizan el servidor esclavo? Los datos de un servidor esclavo es siempre un poco detrás del servidor principal, por una fracción de segundo, o tanto como un minuto o dos, dependiendo de la configuración y el tráfico. No todosSeleccionar consultas pueden manejar sus datos de ser un poco rancio y no hay manera confiable de detectar automáticamente que son los que, por lo que por defecto todas las consultas utilizan el servidor maestro.

Sin embargo, hay otro truco bajo la manga que podemos utilizar. Después de escribir los datos, podemos llamar db\_ignore\_slave (). Eso hará que una nota en la sesión activa para desactivar el servidor esclavo para el usuario actual por un período de tiempo configurable. (El valor predeterminado es de cinco minutos.) De esta manera, Seleccionar ejecutar consultas sobre solicitudes de páginas posteriores por ese usuario siempre llegan los datos más actualizados, pero otros usuarios obtendrán los datos antiguos hasta que los servidores esclavos ponerse al día. Eso puede ser muy útil, por ejemplo, cuando un usuario envía un comentario. Ellos quieren ver el comentario de inmediato, pero no está mal si otros usuarios no lo ven durante 20-30 segundos.

**[373 ]**



*Base de datos Access*



**Resumen**

Aunque en la práctica es sorprendente la frecuencia con la que no tendremos que hacer frente a la base de datos directamente, lo hacemos de un modo u otro. En este apéndice hemos visto cómo aprovechar capa de base de datos de Drupal para construir potentes consultas, entre bases de datos que son, en muchos casos, más fácil de trabajar que el SQL, sin dejar de reflejo de SQL real.

* Hemos visto cómo escribir portable y segura consultas SQL arbitrarias.
* Hemos visto cómo permitir que otros módulos para modificar nuestras preguntas antes de que se ejecuten.
* Hemos visto cómo escribir consultas complejas que cambian según la entrada del usuario de una manera portátil.
* Ahora sabemos cómo escribir consultas de modificación de datos que se optimizan automáticamente para cualquier base de datos que se esté utilizando.
* Hemos visto cómo utilizar las transacciones para hacer nuestro código más robusto y tolerante a fallos.
* Entendemos cómo apoyar la replicación maestro / esclavo para permitir que los sitios para escalar a los altos niveles de tráfico.

**[374 ]**



Seguridad

A lo largo de este libro, hemos hablado en varias ocasiones la cuestión de la seguridad.

¿Qué hace que una buena práctica de seguridad? ¿Cómo hacemos para que el código sea más seguro? ¿Cómo protegemos contra ataques comunes? Eso es porque la seguridad no es una característica para ser atornillada en un programa. Es parte integrante del proceso de desarrollo.

Sin embargo, debido a que es un tema tan importante este apéndice se da una visión general de las herramientas de Drupal ofrece para ayudar a que el código sea más seguro, y el tipo de cosas que pensar cuando se escribe código para asegurar que nuestro código es tan seguro como

posible. Es de ninguna manera exhaustiva; muchos libros se han escrito exclusivamente en el tema de la seguridad. Sin embargo, se debe establecer una base sólida para abordar la escritura de código de una manera segura.

**Pensar de forma segura**

La seguridad es un proceso. Específicamente, es un proceso de desarrollo. El aspecto más importante de la seguridad es la forma en que nos acercamos al código que vamos a escribir. Tenemos que "pensar de forma segura" con el fin de escribir código robusto. Aunque hay muchos aspectos a pensar de forma segura, se puede resumir como "pensar paranoico".

La gran mayoría de las entradas en nuestro sistema va a ser sano y lo que esperamos, no es peligroso. Sin embargo, siempre habrá ese último 1% de que no es en absoluto lo que esperamos. Podría ser que alguien deliberadamente tratando de entrar en nuestro sitio web. Podría ser un robot de spam tratando de encontrar vulnerabilidades de modo que pueda convertir nuestro sitio en un cartel de drogas de proscripción falsos. Alternativamente, podría simplemente ser un usuario honesto que entró entrada que no esperábamos y cuenta. Todas estas cosas va a suceder

en algún momento durante la vida de un sitio web o una aplicación web, y la mejor manera de protegerse contra ellos es asumir que cualquier entrada es una sospecha de ataque hasta que podamos verificar lo contrario.

*Seguridad*



Siempre asuma que los datos de entrada no es segura hasta que ya menos que se ha procesado para asegurarse de que es seguro.

La mayoría de las vulnerabilidades de seguridad provienen de no correctamente comprobando si un pedazo de datos es lo que esperamos que sea. Recuerde también que hay muchos de los tipos de entrada.

* Cualquier información procedente del usuario, ya sea en el OBTENER consulta o una POSTE solicitud, se introduce y puede ser inseguro.
* Cualquier cookie envíe el usuario se introducen, y pueden ser falsificados.
* Todos los archivos se leen del disco son de entrada, que puede contener datos que no esperamos. Incluso si no es malicioso, formatos de datos inesperados pueden suponer un grave riesgo de seguridad.
* Los datos almacenados en una base de datos que no escribimos a pueden ser inseguros.
* Incluso los datos almacenados en la propia base de datos de Drupal puede ser inseguro, si no sabemos a ciencia cierta que lo limpiamos antes de guardarlo.



Los datos no tiene por qué ser maliciosas con el fin de ser una amenaza para la seguridad.

Asimismo, recuerda que nuestro sitio web puede no ser el blanco de un ataque. Una táctica muy común es para enviar comentarios sobre un sitio web que contiene código JavaScript que, cuando se ve, se hará cargo de el navegador web de un usuario o engañarlos para visitar otro sitio que descarga código malicioso en su ordenador. Aunque nuestro sitio no se vea perjudicada, todavía se utiliza como una forma de atacar a otro usuario. Nosotros no queremos eso.

**Filtrado contra escapar**

Hay dos maneras de tratar con datos potencialmente inseguros, es decir, la filtración y escape.

Filtrado implica despojar a las partes de los datos que podrían ser problemas, o forzar a los datos en una forma más simple, más seguro. La práctica común en PHP es filtrar todas las entradas en el sistema de manera inequívoca. Drupal tiene una táctica un poco diferente, en cuanto a ciertas piezas de contenido, como un cuerpo de nodo, no podemos saber el formato correcto de antemano. Si un usuario cambia una configuración de formato de texto, por ejemplo, no queremos que el texto previo de los interesados ​​que se pierde, simplemente se filtra de manera diferente. En cambio, la filtramos en la salida.

**[376 ]**



*Apéndice B*



**Filtración**

Las siguientes pautas, entonces nos deben guiar en cuanto a la manera de detectar correctamente los datos de entrada:

* Si sabemos de antemano que un determinado elemento de datos se supone que debe ser numérico, y lógicamente no tendría sentido de otra manera, convertirlo a un entero o un flotador con el fin de filtrar las cadenas inseguras.
* Si sabemos de antemano que hay un número fijo de valores posibles de una pieza de información podría contener, rechazar rotundamente cualquier valor que no son uno de

los de la lista de permitidos. En la mayoría de los casos, la forma de la API de Drupal lo hace por nosotros.

* No filtrar las etiquetas HTML al guardar datos de texto a la base de datos, como si esas etiquetas son adecuadas o no depende de donde son mostradas luego.
* Tratar a todos los datos textuales, incluso si se trata de nuestra propia base de datos, como el seguro, hasta filtrada en tiempo de visualización.

Drupal ofrece una serie de herramientas para el filtrado de datos textuales. Los más importantes son:

* filter\_xss () se tira a cabo todas las etiquetas HTML, excepto los de una lista especificada.

La lista por defecto es razonablemente seguro.

* filter\_xss\_admin () es el mismo que filter\_xss (), Pero con una muy lista de etiquetas permisiva. Se utiliza mejor para texto que sabemos que sólo se entró por un administrador de confianza.
* check\_markup () filtrará una parte del texto de acuerdo con un conjunto específico de filtro reglas, que se configuran a través de la interfaz de usuario administrativa que formatos de texto.
* check\_plain () adopta un enfoque más mazo para un fragmento de texto. En lugar de la eliminación de las etiquetas HTML de poca confianza, escapa todas las etiquetas HTML para que puedan aparecer, literalmente, en el navegador. Véase más abajo para más en escapar.

**Escapando HTML**

Escapar, por el contrario, no elimina el contenido de una hoja de datos, pero la codifica con un formato que otro sistema de espera para evitar confundir ese sistema. Por ejemplo, si queremos imprimir etiquetas HTML a la página de tal manera que el usuario puede ver, tenemos que escapar de la< y > caracteres utilizando los códigos de entidad HTML. Los dos sistemas más comunes que Drupal será el envío de datos a una base de datos SQL son y el navegador web, y ambos requieren enfoques diferentes.

**[377 ]**



*Seguridad*



Para la salida al navegador, check\_markup () puede estar configurado para escapar de HTML o de otros contenidos. La check\_plain () función toma un enfoque algo más maza, escapando todo el HTML en una cadena para que muestre textualmente en la página para que el usuario vea.

**Inyección SQL**

Para escribir en la base de datos, la capa de base de datos de Drupal se basa en el concepto de declaraciones preparadas. Declaraciones preparadas, entre otras cosas, se separan limpiamente la consulta SQL a partir de los datos variables en el mismo. Eso permite que el servidor de base de datos propia para construir una consulta con cordura mientras escapaba de la entrada en sí, evitando el ataque común conocido como "inyección SQL". Por ejemplo, un valor que contiene un apóstrofo provoca un error de sintaxis en una sentencia SQL (desde comillas simples tienen significado en SQL), en el mejor, o permite que una consulta SQL arbitrario extra para colarse en el comando en el peor.

Para evitar ese problema, nunca, nunca poner una variable en una cadena SQL directamente. Al escribir una consulta en la base de datos de Drupal, asegúrese siempre de que la parte de consulta es una cadena literal utilizando marcadores de posición y luego proporcionar valores para los marcadores de posición. Si lo hace, permitirá que la base de datos para separar la plantilla de consulta de contenido variable y evitar la inyección SQL. Si la consulta en sí es variable, utilice el generador de consultas dinámicas,db\_select ().



Inyección de SQL a partir de consultas mal escritos es el más común y el más fácilmente evitable, forma de vulnerabilidad de seguridad.

**Control de acceso Nodo**

A la pregunta de seguridad sobre todo en Drupal específico es cómo controlar el acceso a los nodos.

Mientras sistema de permisos de Drupal se encarga del caso de uso común de los nodos a nivel mundial legibles y acceso limitado a editar o eliminar nodos, hay un montón de casos en los que necesitamos de control de acceso más complejo. Sistema de acceso de los nodos de Drupal se encarga de los casos, pero requiere que atamos en ella cada vez que miramos los nodos, es decir, cada vez que ejecute una consulta que intenta encontrar registros en la tabla de nodos.

Puesto que hay muchas situaciones variadas en las que podrían utilizarse nodos, estos controles de acceso adicionales sólo se puede hacer a nivel de base de datos, es decir, en la propia consulta. Afortunadamente, Drupal permite a los módulos para manipular ciertos tipos de consultas antes de ser ejecutados para añadir cualquier tipo de filtración. Con el fin de permitir que Drupal para modificar nuestras preguntas, tenemos que "etiqueta" de manera apropiada.

**[378 ]**



*Apéndice B*



En primer lugar, cada vez que se escribe una consulta en la tabla de nodos, debemos utilizar una consulta dinámica, construida utilizando db\_select (). En segundo lugar, tenemos que llamar a la addtag ​​() método en el objeto de consulta y darle una etiqueta de node\_access. Por ejemplo:

$ Resultado = db\_select ("nodo") -> campos ("nodo") -> condición ("tipo", "página") -> addtag ​​('node\_access') -> execute ();

En el código anterior, estamos seleccionando todos los datos de la tabla de nodo para los nodos de la "página" de tipo. La etiqueta extra, sin embargo, permite que Drupal para alterar la consulta antes de ejecutarla y también para filtrar los nodos que el usuario actual no tiene acceso a la vista.



Utilice siempre una consulta dinámica con una etiqueta denode\_access al consultar la tabla de nodos para que Drupal puede aplicar filtros de seguridad adicionales que sean necesarias.



Ver http://drupal.org/node/310077 para más detalles sobre la forma de etiquetado y consulta Trabajos de transformación de la consulta, y cómo algunas otras etiquetas comunes funcionan.

**Manejo de código inseguro**

Tarde o temprano, usted toparse con código que es inseguro. Podría ser un módulo que usted ha escrito, o un módulo contribuido que has descargado de Drupal.org, o posiblemente incluso Drupal propio núcleo. Ningún código es perfectamente seguro y ningún desarrollador es perfecto, ya sea, por lo que va a suceder. Es sólo una cuestión de cuándo y qué podemos hacer al respecto.

Independientemente de cuyo código es, la manera de manejar la situación es más o menos la misma:

* No entre en pánico.
* Investigar el módulo en cuestión (ya sea el nuestro o no) para determinar exactamente cuál es el problema. A veces es un error muy simple que se puede arreglar fácilmente, y otras veces es muy sutil. Tenemos que averiguar si afecta a una sola versión del módulo o varias versiones, o si se aplica solamente a ciertas configuraciones (por ejemplo, sólo en servidores web Apache, sólo en Windows OS), entre otras cosas.
* Si somos capaces de hacerlo, armar un parche que soluciona el problema, pero no lo publiques a la cola de emisión.
* No entre en pánico.

**[379 ]**



*Seguridad*



* Enviar una descripción detallada del problema, junto con una revisión en caso de que se haya podido hacer una, al Equipo de Seguridad en Drupal security@drupal.org. Ellos coordinan todos los asuntos relacionados con la seguridad para el código alojados en drupal.org. Si no es uno de nuestros módulos, que se coordinarán con el mantenedor del módulo.
* Si se trata de uno de nuestros módulos, deje que el equipo de seguridad revisar el problema y el parche. Si están de acuerdo que es un agujero de seguridad, que van a trabajar con nosotros para investigar el parche o mejorarla y programar una versión de seguridad. Será hasta nosotros para escribir un Asesor de Seguridad (SA) también.
* Versiones de seguridad casi siempre salen los miércoles, así que esperen hasta que tengamos una luz verde por parte del equipo de seguridad. Cuando dicen que su clara para hacerlo, aplicamos el parche para todas las versiones afectadas del módulo y creamos una nueva versión. El equipo de seguridad se publicará la SA.
* ¿Mencionamos que no se preocupe?

Drupal funciona con un principio de transparencia total. Cuando se descubre un problema de seguridad, que se mantiene en secreto hasta que una solución esté disponible. Una vez que una solución está disponible, los detalles de la cuestión de la seguridad, incluido el código para arreglarlo y lo libera se ven afectados, se publica para todo el mundo lo vea. Este es un estándar de mejores prácticas de seguridad, ya que por un lado se reduce al mínimo las posibilidades de un aprendizaje bueno para nada bien a la vulnerabilidad antes de que haya una solución disponible, pero por el otro, una vez que una solución esté disponible, nos hacen lo más fácil posible para que las personas saben que necesitan actualizar su código para mantenerse seguro ante un atacante puede aprovecharse de ella.

Para obtener más información sobre la forma de manejar las cuestiones de seguridad en Drupal, consulte:

http://drupal.org/security-team

**Mantenerse al día**

Entonces, ¿cómo podemos mantener un sitio hasta al día con los últimos parches de seguridad? Hay dos claves para mantenerse en la cima de las necesidades de seguridad, y cualquier administrador del sitio deben hacer uso de los dos.

En primer lugar, todos los avisos Drupal seguridad se anuncian en Drupal.org en la siguiente dirección:

http://drupal.org/security

**[380 ]**



*Apéndice B*



Hay dos canales RSS para suscribirse y listas de correo electrónico que pueden unirse para ser notificado cada vez que una SA se emite. Todos los administradores de Drupal sitio y los desarrolladores deben suscribirse a una o la otra con el fin de recibir una notificación cuando se produce una liberación en materia de seguridad. Ambos son, afortunadamente, muy poco tráfico.

En segundo lugar, la propia Drupal incluye un módulo llamado Estado de la actualización que se conectará periódicamente para drupal.org para ver si hay una nueva versión de cualquier módulo instalado en nuestro sitio. Está habilitado por defecto, pero se puede desactivar. No desactive, sin embargo. De hecho, lo mejor es mantener la opción de notificación por correo electrónico habilitada también para recordar por e-mail cuando un módulo tiene una nueva versión de seguridad. No podemos mantener un sitio seguro si no sabe que tiene una vulnerabilidad, para empezar.

**Resumen**

La seguridad es un tema muy amplio, y que fácilmente podríamos pasar mucho más tiempo a discutir con mayor detalle. Sin embargo, por el momento, es suficiente con centrarse en un enfoque seguro para el desarrollo general.

* La seguridad es un proceso
* Todos los datos de entrada, de cualquier fuente, en caso de no ser de confianza hasta que se haya verificado y desinfectado
* Filtrar todos los datos, ya sea en la entrada o en la salida según sea apropiado
* Siempre filtrar o escapar de los datos enviados al navegador de una manera que tenga sentido, ya donde se va a utilizar
* Protege contra la inyección de SQL mediante declaraciones preparadas, y nunca, nunca poner una variable directamente en una consulta SQL
* Cuando se descubren vulnerabilidades de seguridad, no se asuste, pero seguir las mejores prácticas establecidas para informar y solucionar el problema
* Siempre estar al tanto de versiones de seguridad disponibles para los módulos de Drupal o Drupal

**[381 ]**



**Símbolos**

**$ Parámetro cuenta 254 $ element parámetro parámetro node 128 $ 201 $ variable de parámetro form\_state formulario 128 $ 254**

**$ Parámetro op 245, el argumento 254 $ options 290**

**. Archivo de información**

cerca de 29, 93, 345 CSS, añadiendo a 289

JavaScript, y agregó que 289 necesitan para el 29 de

escribiendo 29-32

**. Archivo de módulo** cerca de 29, 33 crear 34

**. Archivo de perfil** sobre 346

tareas de instalación, el almacenamiento 346

**<? Php?> Instrucción del procesador 35 {} node\_access mesa**

alrededor de 250, 261, 263

comparando, con hook\_node\_access () 250-253

la necesidad de 261, 262

permisos, el control de la reconstrucción de 248 273, 274

esquema 264

**{Node} tabla 243**

**La**

**acceso**

controlar, para los nodos 378, 379

Índice

**comprobaciones de acceso**

correr, en los formularios 233, 234 módulo de control de acceso reglas 264, 265

frente al módulo de acceso a las funciones de nodo 261 de acceso

declarando 221, 222

**Acquia Drupal**

URL 21

**método método addField () 366 addtag ​​() 367 bases de datos avanzadas, Drupal**

la replicación de bases de datos master / slave 373 transacciones 372

**Desarrollo ágil de software 26 AJAX**

añadiendo, a las formularios 305, 306

aplicando a los elementos en la página 307 de 308 comandos de 309-312

utilizando, en forma 235

**AJAX, comandos** ajax\_command\_after 309 ajax\_command\_alert 309 ajax\_command\_append 309 ajax\_command\_before 310 ajax\_command\_changed 310 ajax\_command\_css 310 ajax\_command\_data 310 ajax\_command\_html 310 ajax\_command\_prepend 311 ajax\_command\_remove 311 ajax\_command\_replace 311 ajax\_command\_restripe 311 ajax\_command\_settings 312

**ajax\_command\_after comando 309**

**comando comando ajax\_command\_alert 309 ajax\_command\_append comando 309 ajax\_command\_before 310 comando comando ajax\_command\_changed 310 ajax\_command\_css 310 comando ajax\_command\_data 310 comando ajax\_command\_html 310 comando ajax\_command\_prepend 311 comando ajax\_command\_remove 311 comando ajax\_command\_replace 311 comando ajax\_command\_restripe 311 ajax\_command\_settings comando 312 ajax\_get\_form función () 235**

**Devoluciones de llamada de AJAX**manejo de 235 utilizando 236, 237

**API hooks 63 obras de arte**

unos 152, 183 añadiendo 165, 167 crear 153, 154 eliminando 178-181 edición 177, 178 cuenta con 184 ahorro 172-174 visualización 176, 177 de trabajo 208-210

**obras de arte, y agregó** revisiones, manipulación 175 se someten de devolución de llamada de validación de devolución de llamada 171 170

**módulo artwork.controller.inc** la creación de 162

**artwork\_menu () hook 163 artwork\_page\_view () función 177 mesa artwork\_revision**

llamar a 154, la función 155 artwork\_type\_load () 161 artwork\_types () función 160 artwork\_uri función () 158 Clase ArtworkController 162 tipos de las ilustraciones

gestionar 163-165 pruebas automatizadas, Drupal unos 49

prueba, la creación de 50

**B**

**comportamientos** aproximadamente 303 ejemplo 303

**Bloquear API** sobre 42

hook\_block\_info () 43, 44 hook\_block\_view () 45-47 trabajar con 43

**bloques**

configurar 355-357 tematización 98

**subsistema de bloque** sobre 25

utilizando, en módulos de 25 bootstrapping 13 paquete de 152

**lógica de negocio**

contra presentación lógica 62, 63 de derivación permiso de acceso del nodo 263

**C**

**el almacenamiento en caché 292**

**Cascading Style Sheets.** *Ver* **Función check\_markup CSS () función 377 check\_plain () 41, 166, 377 cierres 32**

**CMF 8**

**Módulo CMS 8 Coder** características 23

**convenciones de codificación, la línea de comandos Drupal 30**

instalador, corriendo 360-362 módulo comment 18

**función 140 método formateadores complejos 201-204 complejo widget de 194-198 condición () 370 confirm\_form (), 141 contenidos**

manejo de 151

**Marco de gestión de contenido.** *Ver* **CMF**

**El gestor de contenidos.** *Ver* **Controlador CMS 156**

**[384 ]**



**núcleo de las bibliotecas 13 módulos básicos 14**

**Crear, Ver, Actualizar y Eliminar.** *Ver*

**CRUD**

**petición en sitios cruzados falsificaciones (CSRF) 229 CRUD 242**

**CSS** sobre 10

añadiendo, a. archivo de información 289 alterando 300

en línea, añadiendo en la página 294

**CSS\_DEFAULT, peso Drupal 292 CSS\_SYSTEM, peso Drupal 292 CSS\_THEME, peso Drupal archivos 292 CSS**

añadiendo, a las páginas 292

**CVS 31**

**D**

**base de datos de 15**

**Bases de datos de la próxima generación.** *Ver*

**Granularidad de datos DBTNG** cerca de 64, 65

65 ventajas ilustración 65

**implementación de la función método método db\_ignore\_slave () 373 db\_insert () 369 db\_select () constructor de la pregunta 378 db\_transaction () 174 363 DBTNG propiedad default\_formatter propiedad 186 186 default\_widget tema por defecto** edificio, los pasos 106

reutilización de 91, 92 consultas de eliminación 370

**dependencias [] Directiva 32 descripción de la propiedad del módulo 186 Devel Nodo de Acceso** aproximadamente 282-285

hook\_node\_access\_explain () 283, 284 bloques de usuario, proporcionando 284, 285

**módulo desarrollador** cerca de 22 características 23

**dimfield.module 185 distribuciones, Drupal**sobre 344

módulos, la venta por paquetes 345 directorio de perfil 344 la creación de 344

temas 345 del bloque doc 36

**bloque de documentación.** *Ver* **bloque doc**

**Módulos Doc estilo Doxygen** utilizando 36, 37

**Programa Doxygen 22 Drupal**

sobre 7

AJAX comandos 309-312 arquitectura 11, 12 pruebas automatizadas 49 caching 292

convenciones 30 contenidos manipulación 151 elementos de formulario personalizado de codificación 132 distribuciones 344, 345

Módulos Doc estilo Doxygen 36, 37 funciones 7

API de archivos 316

almacenamiento de archivos 314, 315 formularios, asegurando 229 funciones auxiliares, utilizando 305 Imagen API 326

la manipulación de imágenes funciona 326 Image Styles 331-333

JavaScript 287

API Library 295

Permisos de nodos, la concesión de permisos 242-244 211

plugins 287, 288 preprocesamiento 292 renderizable matriz 298 parches de seguridad, actualizaciones de 380 envoltura de secuencia 319 herramientas, artículos de 20 Página web del enlace 8

pesos 292

**Drupal, funciones de ayuda** comportamientos 303, 304

presentación temable 301, 302 cadenas traducibles 302, 303

**[385 ]**



**Drupal, subsytems** bloques 20

pruebas de código de 20 comentarios 18 entidades 19 campos 19

archivos 18 Formularios de API 19 menús 17 nodos 17 temas 16 usuarios 18

**Drupal, tecnologías** sobre 8

CSS API 10 base de datos de 10 bases de datos 9 HTML 10 JavaScript 10 MySQL 9

PHP 9

PHP Data Objects biblioteca PostgreSQL 9 9

RSS (Really Simple Syndication) 10 XML (eXtensible Markup Language) 10 XML-RPC 10

**Drupal, pesos**

CSS\_DEFAULT 292

CSS\_SYSTEM 292

CSS\_THEME 292 drupal\_access\_denied () de aproximadamente 224

de trabajo 226

**función 102 drupal\_add\_css (), la función 289 drupal\_add\_js ()**

utilizando 289, la función 293 drupal\_exit () sobre 225

función del ejemplo 225 drupal\_get\_form () sobre 128

utilizando 127

**función 102 drupal\_get\_path (), la función 290 drupal\_mail ()**

unos 141 argumentos 142, 143

la implementación de 142 electrónico, el envío de 141

**Función función drupal\_render () 128 77 drupal\_set\_message () 137 drupal\_static ()**

sobre 161 usando 161

**Arquitectura Drupal** aproximadamente 11, 12

bibliotecas del núcleo de 13 módulos de núcleo 14 de base de datos 15 hooks 13

tema del sistema 16

**Clase DrupalDefaultEntityController 156 Drupal Recomendaciones de seguridad**

URL 380

**DrupalStreamWrapperInterface 321 DrupalWebTestCase :: getInfo () de 52 DrupalWebTestCase :: setup () de 55**

**Drush 23, 28**

**consultas dinámicas 366-368**

**E**

**sistema de plantillas de correo electrónico 141**

**editar cualquier permiso contenido del artículo 257 elementos**

incapacitante, en forma de 230 entidades

sobre 151

creación, razona 151, 152 declarando 156-161 ejemplo 152

la búsqueda de 206-210 gestión del sistema 163 entidades 19

**función entity\_get\_controller () controlador 168 entidad 161**

**nivel de entidad de datos 207 Dirección de la entidad**

sobre 163

obras de arte, añadiendo 165, 167 obras de arte, eliminar 178 a 181 obras de arte, la edición de 177, 178 obras de arte, se muestran 176, 177

**[386 ]**



tipos de las ilustraciones, la gestión 163-165 nuevas obras de arte, añadiendo 167-170

**escapar** sobre 377

frente al filtrado 376 explotan la función () 198

**F**

**función fclose () función 319 feof () 319 API Campo**

almacenamiento de campo enchufable 185 de trabajo 184

**código de campo**

escritura, razona 183, 184 instancia de campo 184

**campos** sobre 184

declarando 185

mostrar, formateadores utilizan 199 encontrando 206-210

ajustes 188, 189 estructura, definiendo 186-188 validación 189, 190

**método de campos () 366 campos de sistema 19 estructuras exteriores** 186-188 definir el tipo de campo

sobre 184, 185 la creación de 185

**función file\_copy () 316 FILE\_EXISTS\_ERROR constante 317 FILE\_EXISTS\_RENAME constante 317 FILE\_EXISTS\_REPLACE función constante 317 file\_get\_contents () 317, 328 file\_link, hook tema 92**

**Función función file\_move () 316 file\_save\_data () 317, Función 329 file\_unmanaged\_copy () 316 file\_unmanaged\_save\_data () 329 file\_uri\_target () 323**

**API de archivos 316**

**Funciones de la API de archivos** sobre 316 file\_copy () 316

file\_get\_contents () 317 file\_move () 316 file\_save\_data () 317 file\_unmanaged\_copy () 316

**Módulo FileField 316 archivos [] directivas del sistema 32 archivos 18**

**almacenamiento de archivos, Drupal** aproximadamente 314 privado 314 pública 314

función temporal 314, la función 315 filter\_xss () 377 filter\_xss\_admin () 377 de filtrado

unos 376 377 directrices herramientas 377

frente a escapar 376 herramientas de filtrado, Drupal sobre 377

check\_markup () 377 check\_plain () 377 filter\_xss () 377 filter\_xss\_admin () 377 first\_block\_info (función) 43 fopen () de 319, 324 foreach () bucle 190

**API Form** sobre 126

función drupal\_get\_form (), utilizando 127 cuenta con 127

campos, exponiendo con widgets sistema 191 Ajustes 138, 139

**Formulario de Referencia de la API completa**

URL 127

**callback formatter\_view 201 formateadores**

sobre 184, 200

formateadores complejos 201-204 declarando 200

campos, mostrando 199 formateadores de un solo valor 200

**formulario de devolución de llamada** la creación de 305

**función de devolución de llamada de forma** la construcción de 128 a 130

**[387 ]**



**formar el manejo de 229 formularios**

controles de acceso, que se ejecuta en 233, 234 AJAX, añadiendo 305, 306

AJAX, con 235 elementos, desactivar 230 datos seguros, pasando por 231 de sujeción, en la presentación de Drupal 229 136, 137

**Formularios API** sobre 19, 229 de trabajo 229

**función fread () 319 pruebas funcionales 49**

**T**

**método de la función getExternalURL () 323 getInfo () 53**

**H**

**funciones de ayuda, Drupal** comportamientos 303, 304

presentación temable 301, 302 cadenas traducibles 302, 303

**hook** la aplicación de 14

**hook\_block\_info () 43, 44 hook\_block\_view () 43-47 hook\_css\_alter ()**

CSS, alterando 300 ejemplo 300

**hook\_element\_info 80 hook\_enable ()**

sobre 227

papel, creando 228 usando 227

**hook\_entity\_info () 156 hook\_field\_extra\_fields () 206 hook\_field\_info\_alter () Hook, propiedades** default\_formatter 186

default\_widget 186 Descripción 186 instance\_settings 186 etiqueta 186

ajustes 186

**hook\_form\_alter () 233 hook\_help ()**

sobre 38

La implementación de 38, 39, 120 hook\_image\_default\_styles () 339, 340 hook\_image\_effect\_info () aplicar 334

**hook\_init () 14 hook\_install\_tasks ()** sobre 348

propiedades 348 hook\_install\_tasks\_alter () 348 hook\_library ()

sobre 295

información, alterando 297

API Library, definiendo hook\_library\_alter 296 () 297 hook\_mail () la aplicación 144, 145 hook\_menu ()

formulario, que define 305

Página de devolución de llamada, definir 121-124 hook\_menu () aplicación crear 327 hook\_node\_access ()

aproximadamente 254, 260

módulos de control de acceso 259

comparando, con {node\_access} tabla 250-253

implementar, en vista del funcionamiento 260 invocando 247

Módulo de acceso de nodo, escribiendo 260 parámetros 254

módulo de control de acceso muestra las operaciones 254-259 visión 259

**hook\_node\_access (), los parámetros** $ 254 cuenta

$ Node 254 $ op 254 254

**hook\_node\_access\_explain () 283, 284 hook\_node\_access\_records () 266-268 hook\_node\_access\_records\_alter ()** sobre 279

utilizando 280, 281 hook\_node\_grants () sobre 269, 270

utilizando 249

**[388 ]**



**hook\_node\_grants\_alter ()** sobre 275

utilizando 275-277 hook\_page\_alter () 81 hook\_permission () sobre 124, 217

Permisos de módulos, definir 218, 219 necesitan de 217

utilizando 218 escritura 219

**hook\_query\_alter () 262, 263 hook\_stream\_wrappers () 320, 324 hook\_theme ()**

cerca de 86, 87 la implementación de 107

**hook\_theme\_registry\_alter 88 hook\_user\_view ()** implementar 213-216

**conectar la aplicación 33**

**HTML** sobre 10

escape 377

**html\_tag, tema conectar 92**

**HyperText Markup Language.** *Ver* **HTML**

**Yo**

**imagen**

nodo, adjuntando a la imagen 318, tema hook 92 image\_crop () 326

**función image\_desaturate () 326, la función de la función 329 image\_effect\_color\_validate () 336 image\_load () 326**

**image\_resize () función 326 image\_rotate () de función 326, 330 image\_save () 326, la función 330 image\_scale () 326, la función 330 image\_scale\_and\_crop () 326 image\_style, hook tema 92**

**API de imagen 326**

**Función módulo Imagecache 314, 326 imagecolorallocate () 338 efectos de imagen**

la creación de 334 a 338

**Módulo ImageField 326**

**Librería ImageMagick 326**

**funciones de manipulación de imágenes, Drupal** image\_crop () 326 image\_desaturate () 326 image\_resize () 326

image\_rotate () 326 image\_scale () 326 image\_scale\_and\_crop () 326

**función imagestring () Estilos 338 imágenes**

sobre 331 de trabajo 332, 333

**estilos de imagen**

crear, a partir módulo opcional 339-341 en línea 294

**innerJoin () método 366 código inseguro** manejo de 379, 380 de inserción consulta 368, 369 perfiles de instalación sobre 343

Beneficios 343 install sistema de tareas

aproximadamente 348 seleccionar 348

**instance\_settings propiedad 186 item\_list, hook tema 93**

**J**

**JavaScript** sobre 10, 287

añadiendo, a. Información de archivo 289 y añadió, a las páginas 288 alterando 299, 300

archivos, añadiendo 289 a 291 archivos, preprocesamiento 288 en línea, añadiendo en la página 294 pesos 290

**JavaScript archivos** añadiendo 289-291 preprocesamiento 288

**join () métodos 366, la función 367 jQuery.append () 308 jQuery 1.4.2 287**

**[389 ]**



**K**

**método clave () 371**

**L**

**Método propiedad label 186 leftJoin () 366 Library API**

sobre 287

definir, con la función de hook\_library () 296 utilizando 295

**enlaces, tema hook 93 list\_modules bloque** método propiedades 45 load () 162

**M**

**cuestiones electrónico** depuración 146

**la replicación de bases de datos master / slave 373 módulo menu\_block 92**

**rutas de menús** comodines, utilizando 125

**enrutador menú 17**

**sistema de menús 12, 17, 121 se funden consultas**

sobre 370 sintaxis 371

**módulo**

subsistema de bloque, usando 25 edificio, los pasos 26 la creación de 26

depuración 282

permitiendo, en proceso de instalación 347 diseño del sistema de archivos 28

estilos de imagen, la creación de 339-341 pruebas 282

**método module\_exists () función 347 module\_invoke ()** alrededor del 57

parámetros 57 module\_invoke\_all (función 175)

**MODULE\_preprocess\_page (directorio de módulos) Función 76**

la creación de 29

**nombres de los módulos** alrededor del 27 legible 27

de lectura mecánica 27 more\_link, hook tema 93

**función de preproceso multi-hook 73 MySQL 9**

**N**

**nueva obra de arte** añadiendo 167-170

**nodo** alrededor de 318

adjuntando, a la imagen de la creación de 318 318

**función node\_access ()** unos 243, 244 documentos 244, 245

hook\_node\_access (), invocando 247 caché estática, optimizando 246, 247 lista blanca 246

**NODE\_ACCESS\_ALLOW constantes 254 NODE\_ACCESS\_DENY constantes node\_access\_grants 254 () 249 NODE\_ACCESS\_IGNORE constantes 254 node\_access\_view\_all\_nodes () función 262 node\_page\_default () función 243 node\_query\_node\_access\_alter función ()**

**244**

**API Nodo de Acceso** unos 243 invocando 248-250 necesidad de 243

**Módulo de acceso de nodo** permitiendo 261, 262

frente a módulo de control de acceso 261 escritura 260

**sistema de acceso del nodo** sobre 241

frente al sistema de acceso del usuario del módulo de nodo 242 18

**Permisos de nodos, Drupal** concesión 242-244

**nodos**

el acceso, el control de 378, 379

**[390 ]**



**sistema de nodos 12, 17 datos que no son de campo** gestión 205, 206

**O**

**Función de programación orientada a objetos 9 overlay\_library () 295**

**P**

**Página de devolución de llamada**

la definición, la función hook\_menu () utiliza 121-124

**pager, tema hook 93 páginas**

JavaScript, añadiendo al 288 página del título

establecer 353 permisos sobre 211

afirmando, función user\_access () utiliza 212, 213

asignando 211

definir, para los módulos 218, 219 que permite, mediante programación cuenta de usuario 227, verificando 213-216

**datos persistente** gestión de 133-135

**las variables persistentes 133 PHP**

sobre 9

envoltura de flujo 319

las variables, que pasa a tener JavaScript 293

**Configuración de correo PHP 142 PHPTemplate 66 de base de datos PostgreSQL función 363 pre\_render** la creación de 100

**función de preproceso** alrededor del 68, 69

definir, con TEMPLATE\_ funciones preprocess prefijo 109-113, Archivos de plantilla de alrededor del 72 por

preproceso multi-hook funciona 73, orden de ejecución 74 preproceso 76

proceso funciona 74, 75 TEMPLATE\_ 72, 73

**funciones de preproceso, funciones temáticos** theme\_username función 69

**preprocesamiento 292 archivos privados 314**

**procesal directorio de perfiles de programación 9**

la creación de 344 perfiles de alrededor de 345

la creación de 345, 346 de dependencia, añadiendo 347

**perfiles, creando**

bloques, configuración 355-357 código, colocando 359 hook\_install (), utilizando 348 tarea de instalación, seleccionando 348 instalar el sistema de tareas 348 módulos, que permiten 347 tareas, alterando 354, 355 tareas, creando 349-353

**progress\_bar, hook tema de 93 archivos públicos 314**

**Q**

**consultas**

eliminar consultas 370 consultas dinámicas 366-368 queries INSERT 368, 369 de mezcla consulta 370 consultas al día 370

**R**

**range () método de archivo de 208 README 8 array renderizable** utilizando 298

**render elemento** frente a las variables 108

**Render propiedades de los elementos 79, 80 representan los elementos**

alrededor del 77, 78 hook\_element\_info 80, 81 hook\_page\_alter () 81 79 propiedades

**objeto de resultado 365**

**[391 ]**



**manejo de 175 revisiones de revisión**

manejo de 175 rightJoin () método 366

**De derecha a izquierda idiomas.** *Ver* **Módulo de acceso de papel idiomas RTL**

{} Node\_access mesa, la reconstrucción de 273, 274 crear 266

hook\_node\_access\_records (), utilizando 266-268 hook\_node\_grants (), utilizando 269, 270 consideraciones de seguridad 271, 272

**papeles** sobre 211

la creación de 228

**Idiomas RTL** sobre 103

hojas de estilo, creando para 104-106

**S**

**API de esquema** sobre 152 154 URL

de trabajo 153, 154

**Módulo de búsqueda** aproximadamente 217 ejemplo 217

**Módulo Permisos seguros 227 de seguridad**

alrededor de 375 376 vulnerabilidades

**Consultas SELECT 364 propiedad de configuración 186 Configuración () método simple prueba 54 49**

**formateadores de un solo valor 200 single\_blog\_block\_view (función 97)** edición 100

**función single\_blog\_list () 95 módulo single\_blog**

API, proporcionando 95, 96 edificio 93-95

**normas servidor esclavo de código 373 fuente**

<? Php?> Instrucción del procesador 35 sobre 35

35 Comentarios

funciones 35 sangría 35

espacios alrededor de operadores 35 de espaciamiento, en las estructuras de control de 35 variables 35

**Inyección SQL** aproximadamente 378 evitando 378 ª función () 351 consultas estática 364 store.info el archivo 346

**Función función stream\_mkdir () 321 stream\_open () 323 stream\_write () envoltorio 321 stream**

sobre 313, 319 creando 320-323

**las hojas de estilo**

adjuntando, para hacer arreglos de 102 crear, para idiomas RTL 104-106

**función de presentar 136 subsistemas, Drupal**

bloques de prueba 20 código 20 comentarios 18 entidades 19 campos 19

archivos 18 Formularios de API 19 menús 17 nodos 17 temas 16 usuarios 18

**función system\_element\_info () función 101 system\_settings\_form () 138, 139**

**T**

**t la función ()**

alrededor del 39, 97, 201, 351 cuenta con 40, 41

**tabla, tema hook 93 tarea**

aproximadamente 348 alterando 354, 355 crear 349 a 353

**[392 ]**



**TDD 26 tecnologías, Drupal**

aproximadamente 8 CSS 10

API de base de datos de bases de datos 10 9 10 HTML JavaScript MySQL 10 9

sistema operativo 11 PHP 9

PHP Data Objects Library 9 PostgreSQL 9

RSS (Really Simple Syndication) 10 servidor web 10

XML (eXtensible Markup Language) 10 XML-RPC 10

**TEMPLATE\_ prefijo**

función preproceso, definiendo 109-113 template\_preprocess () Función 75 archivos de plantilla

aproximadamente 70 a 72, 114 crear 115, 116 Funciones de preprocesamiento 72

**temporal de prueba de almacenamiento de archivos 315, la creación de**

sobre 50

caso de prueba, casos de prueba de escritura 51

patrones 51, 52 creación 54 escrito 51

**TestDrivenDevelopment.** *Ver* **Método de prueba TDD**

escribiendo 55-58 texto

fichas, aplicación en 147, 148 filtros de texto

sobre 358 ejemplo 358

**tema 16**

**función función de tema () de 47 theme\_item\_list () 67, 101 theme\_table () 204 theme\_username motores temáticos función 69 66**

**funciones de temas** unos 66, 67 operaciones 82, 83

funciones de preproceso 68, 69 theme\_item\_list () Función 67 tema anula 69

**hooks temáticos** alrededor del 63 file\_link 92 html\_tag imagen 92 92 92 image\_style item\_list 93 enlaces 93 more\_link 93 pager 93 93 progress\_bar registrarse 107 tabla 93 lista\_usuarios 93 user\_picture 93 nombre de usuario 93

**sugerencias hook tema 83, 84 overrides temáticos**

alrededor del 69, la función 70 bartik\_preprocess\_username () 70

**Registro tema** sobre 85 hook\_theme 86-88

hook\_theme\_registry\_alter 88, valora 89 de variable por defecto 85, 86

**tema del sistema 12, 16 tematización 62**

**simbólico** sobre 146

implementar, en texto 147, la función 148 token\_replace () 146 372 transacciones

**Función Twitpic 320 twitpic\_image\_munge () 327 Módulo de Twitpic 313**

**Sitio web Twitpic**

URL 313

**[393 ]**



**U**

**prueba de la unidad 49 al día 370 consultas uso ajax clase 307**

**función user\_access ()** alrededor de 212, 217, 221, 242 Ejemplo 222

permisos, afirmando 212, 213 utilizando 230

**lista\_usuarios, tema hook 93 user\_picture, tema conectar 93**

**función user\_role\_change\_permissions ()**

**228**

**función función user\_role\_delete 228 () 228 user\_role\_save () user\_warn.info archivo**

la creación de 120 archivos user\_warn.module crear 120

**user\_warn\_form\_submit (función) 136 sistema de acceso de los usuarios**

frente nodo del sistema de acceso a la cuenta 242 usuarios

verificando 213-216

**Bloquear usuario 284, 285 nombre de usuario, el tema hook 93 usuarios del sistema 18**

**Usuario Warn módulo** aproximadamente 119 crear 120 funciones 119

**V**

**función validar 136 variable\_get (función 133), la función 133 134 variable\_set (), 357 las variables**

que pasa, a partir de PHP a JavaScript 293 frente a hacer elemento 108

**W**

**función watermark\_effect () widget de 337**

sobre 184

Widgets complejos 194-198 declarando 191, 192

formulario de devolución de llamada 192, 194 widget\_settings\_form callback 194 caracteres comodín

utilizando, en rutas de menú 125

**X**

**Xdebug 21**

**Z**

**Zend depurador 21**

**[394 ]**



**Gracias por la compra de**

**Drupal 7 Desarrollo de Módulos**

**Acerca de Packt Publishing**

Packt, pronunciado 'lleno', publicó su primer libro "Mastering phpMyAdmin para la gestión eficaz de MySQL" en abril de 2004 y, posteriormente, continuó a especializarse en la publicación de libros altamente enfocados en tecnologías y soluciones específicas.

Nuestros libros y publicaciones comparten las experiencias de sus compañeros de los profesionales de TI en la adaptación y personalización de los sistemas de hoy en día, las aplicaciones y los marcos. Nuestros libros basados ​​solución le dan el conocimiento y el poder para personalizar el software y las tecnologías que está utilizando para hacer el trabajo. Libros Packt son más específicos y menos general que los libros de TI que ha visto en el pasado. Nuestro modelo de negocio único que nos permite proporcionarle información más enfocada, dándole más de lo que usted necesita saber, y menos de lo que no.

Packt es una compañía editorial todavía única moderna, que se centra en la producción de calidad, libros de vanguardia para las comunidades de desarrolladores, administradores y novatos por igual. Para obtener más información, por favor visite nuestro sitio web:www.packtpub.com.

**Acerca de Packt Open Source**

En 2010, Packt lanzó dos nuevas marcas, Packt Packt Open Source y Empresa, con el fin de continuar con su enfoque en la especialización. Este libro es parte de la marca Packt Open Source, el hogar de los libros publicados en el software construido alrededor de las licencias de código abierto, y ofreciendo información a cualquiera de los desarrolladores avanzados en ciernes diseñadores web. La marca de código abierto también se ejecuta de archivo Esquema de código abierto de Packt, por la que Packt da una regalía a cada proyecto Open Source sobre cuyo software de un libro se vende.

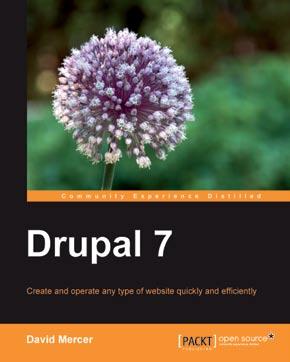
**Escribir para Packt**

Damos la bienvenida a todas las preguntas de la gente que está interesada en la edición. Propuestas de libros deben ser enviados a author@packtpub.com. Si su idea del libro se encuentra todavía en una fase inicial y que le gustaría discutirlo primero antes de escribir una propuesta de libro formal, póngase en contacto con nosotros; uno de nuestros editores puesta se pondrá en contacto con usted.

No estamos sólo en busca de autores publicados; si tiene fuertes habilidades técnicas, pero sin experiencia de la escritura, nuestros editores experimentados pueden ayudarle a desarrollar una carrera de escritor, o simplemente obtener una recompensa adicional para su experiencia.



**Drupal 7**



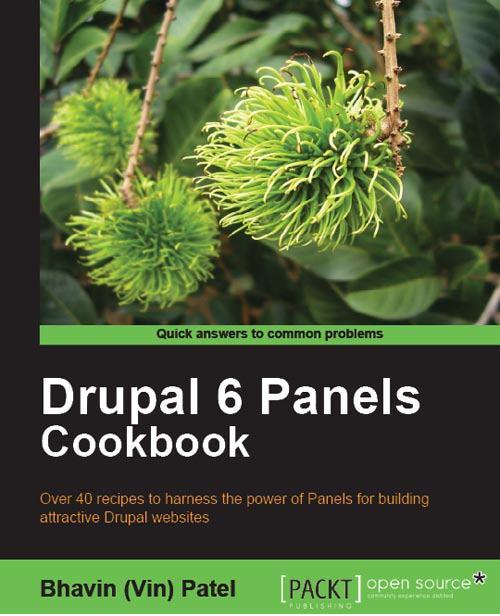
ISBN: 978-1-84951-286-2 Tapa blanda: 416 páginas

Guía básica integral para la instalación, configuración y construcción de un sitio web profesional de Drupal 7

1. Instalar, configurar e implementar una página web de Drupal 7
2. Fácil de añadir características interesantes y potentes
3. Diseñar y poner en práctica el aspecto de su sitio web y se siente
4. Promover, gestionar y mantener su sitio en línea



**Drupal 6 paneles Cookbook**



ISBN: 978-1-849511-18-6 Tapa blanda: 220 páginas

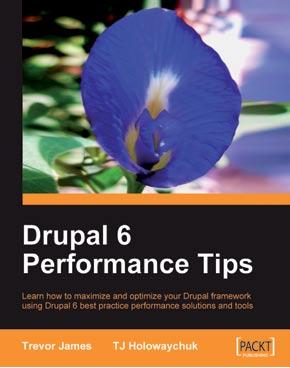
¡Más de 40 recetas para aprovechar el poder de los paneles para la construcción de atractivos sitios web con Drupal

1. Crear diseños de página complejos de forma rápida con los paneles
2. Combine Paneles con otros módulos de Drupal para crear sitios web de medios sociales dinámicos
3. Obtenga soluciones a los problemas de los "grupos especiales más comunes
4. Un enfoque práctico lleno de ejemplos del mundo real para enriquecer la comprensión
5. Parte del libro de cocina de la serie-cada receta de Packt es una secuencia cuidadosamente organizado de instrucciones para completar la tarea de manera más eficiente posible

Por favor, consulte www.PacktPub.com para obtener información sobre nuestros títulos



**Drupal 6 Consejos de rendimiento**



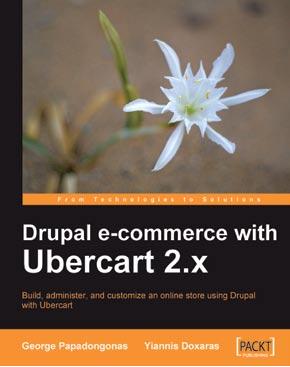
ISBN: 978-1-847195-84-5 Tapa blanda: 240 páginas

Aprende a maximizar y optimizar su marco Drupal usando Drupal 6 mejores soluciones y herramientas de rendimiento de la práctica

1. Supervisar el rendimiento de su sitio web con Drupal y mejorarlo
2. Configurar un entorno de varios sitios Drupal para un mejor rendimiento
3. Gran cantidad de ejemplos con explicaciones claras
4. Elegir y utilizar los mejores módulos de Drupal para mejorar el rendimiento de su sitio



**Drupal El comercio electrónico con Ubercart 2.x**



ISBN: 978-1-847199-20-1 Tapa blanda: 300 páginas

Crear, administrar y personalizar una tienda online con Drupal con Ubercart

1. Crea una poderosa tienda electrónica utilizando el galardonado CMS Drupal y la robusta e-commerce módulo Ubercart
2. Crear y gestionar el catálogo de productos e insertar los productos en modo manual o por lotes
3. Aplicar SEO (search engine optimization) a su e-shop y adoptar técnicas de marketing en Internet llave en mano
4. Implementar técnicas avanzadas como la comparación cross-selling, productos, códigos de descuento y precios segmentada

Por favor, consulte www.PacktPub.com para obtener información sobre nuestros títulos