Mastering Plone Documentation

Publicación 1.2.5a

Philip Bauer, Patrick Gerken

sept. 27, 2017

Índice general

Esta es la documentación del entrenamiento “Maestría en Plone”.

El entrenamiento “Maestría en Plone” está pensado para una semana para las personas que son nuevas en Plone o a las personas que quieren aprender acerca de las actuales mejores prácticas en el desarrollo de Plone. Se puede dividir en dos (02) entrenamientos:

- Un entrenamiento para principiante (2 a 3 días), que cubre los capítulos 1 al 18.
- Un entrenamiento avanzado (3 a 5 días), que cubre el resto de capítulos.

CAPÍTULO 1

Sobre “Maestría en Plone”

Esta entrenamiento fue creada por Philip Bauer y Patrick Gerken de la empresa starzel.de para crear una formación canónica para futuros desarrolladores de Plone. El objetivo es que cualquier persona con los conocimientos adecuados puede dar un entrenamiento basado en este y si lo desea pueda contribuir a mejorar este material. Se publica como código abierto en github y readthedocs.

Si desea ponerse en contacto con los autores originales para un entrenamiento, por favor, puede hacerlo por medio del correo team@starzel.de.

Próximos entrenamientos

Mastering Plone 5 Development: 2. - 6 de Marzo de 2015, en Múnich por Philip Bauer y Patrick Gerken

Entrenamientos previos

El entrenamiento “Maestría en Plone” se celebró públicamente en las siguientes ocasiones:

- Plone Conference 2014, Bristol
- Junio de 2014, Caracas
- Mayo de 2014, Múnich
- PythonBrasil/Plone Conference 2013, Brasilia
- PyCon DE 2012, Leipzig
- Plone Conference 2012, Arnhem
- PyCon De 2011, Lipsia

Entrenadores

Los siguientes entrenadores han dado entrenamiento basado en el material “Maestría en Plone”:

Leonardo Caballero Leonardo J. Caballero G. de Maracaibo, Venezuela, es Director Técnico de Covantec R.L. y Conectivo C.A. Leonardo mantiene las traducciones en español de más de 49 complementos de Plone, así como documentación en español para Plone. Ha contribuido con varios complementos de Plone que forman parte de PloneGov. Actualmente sirve a la Junta de Plone como Embajador de Plone, Leonardo también ha servido como

un miembro de la Junta Asesora y ha hablado o ayudado en la organización eventos de Plone y código abierto en toda América del Sur.

Philip Bauer Philip Bauer es un desarrollador web de Munich que se enamoró de Plone en 2005 y desde entonces trabaja casi exclusivamente con Plone. Un historiador por educación que derivó hacia la creación de sitios web en los años 90 y fundó la empresa Starzel.de en 2000. Es miembro de la Fundación Plone, ama la enseñanza y se dedica al Open Source. Entre otros proyectos relacionados con Plone, comenzó a crear el manual de entrenamiento "Maestría en Plone" para que todos puedan convertirse en programadores Plone.

Patrick Gerken Patrick Gerken trabaja con Python desde 2002. Comenzó a trabajar con aplicaciones Zope puras y ahora desarrolla principalmente con Plone, Pyramid y Javascript, además de hacer tareas de lo que se llama DevOps. Trabaja en Starzel.de.

Steve McMahon Steve McMahon es un miembro de la comunidad Plone hace mucho tiempo, colaborador y entrenador. Él es el creador del producto PloneFormGen, el cual es un generador de formulario a traves de la Web para Plone y el mantenedor del instalador unificado. Steve también escribió varios capítulos del libro 'Practical Plone' y es un orador e instructor experimentado.

Steffen Lindner Steffen Lindner comenzó a desarrollar Plone en 2006. Trabajó en pequeños sitios de Plone y también en enormes sitios de intranet. Como desarrollador de Open Source / Software Libre se unió al equipo de desarrolladores del núcleo de Plone en 2011 y trabaja en Starzel.de.

Usando la documentación para un entrenamiento

Siéntase libre para organizar un entrenamiento por ti mismo. Por favor, sea tan amable de aportar cualquier corrección de errores o mejoras que haya realizado en la documentación de su entrenamiento.

El entrenamiento se genera mediante la herramienta Sphinx y se basa en dos sabores:

por defecto La versión detallada utilizada para la documentación en línea y para el instructor. Para construirlo con Sphinx, ejecute el el comando `make html` o use la versión en línea.

presentation Una versión abreviada utilizada para el proyector durante un entrenamiento. Ese debe usar más lista de viñetas que texto detallado. Para construirlo con Sphinx, ejecute el el comando `make presentation`.

Nota: Anteponiendo un bloque con indentación de texto o código con `.. only:: presentation` usted puede controlar que este bloque es usado solamente en la versión presentación.

Para ocultar un bloque desde la versión presentation use `.. only:: not presentation`

Contenido sin un prefijo sera incluido en ambas versiones.

El tema readthedocs

Nos ajustamos ligeramente al tema readthedocs en el archivo `_static/custom.css` para que funcione mejor con proyectores:

- Comenzamos ocultar la barra de navegación mucho antes de manera que no interfiera con el texto.
- Ampliamos el ancho predeterminado del área de contenido.

Ejercicios

Algunos javascript adicionales muestran las soluciones ocultas para los ejercicios haciendo clic en ellos.

Simplemente escriba la solución con este markup:

```
..  admonition:: Solution
    :class: toggle
```

Aqui hay un ejemplo completo:

```
Exercise 1
^^^^^^^^^^

Your mission, should you choose to accept it...

..  admonition:: Solution
    :class: toggle

    To save the world with only seconds to spare do the following:

    .. code-block:: python

        from plone import api
```

Eso sera renderizado como algo así:

Ejercicio 1

Su misión, si decide aceptarla...

Solución

Para salvar al mundo con sólo unos segundos de sobra haga lo siguiente:

```
from plone import api
```

Construyendo la documentación localmente

Para construir la documentación siga estos pasos:

```
$ git clone https://github.com/plone/training.git
$ cd training
$ virtualenv-2.7 .
$ source bin/activate
$ pip install -r requirements.txt
$ make html
```

Ahora puede abrir lo generado en el directorio `_build/html/index.html`. Para construir la versión de presentación ejecute el comando `make presentation` en lugar del comando `make html`. Usted puede abrir la versión presentación en el directorio `presentation/index.html`.

Cosas que hacer antes de un entrenamiento (como entrenador)

- Preparar un servidor de correo electrónico para el registro de usuario (http://plone-training.readthedocs.io/en/legacy/features.html#configure-a-mailserver)

- Si usted sólo hace una parte del entrenamiento (Avanzado) prepare una base de datos con los pasos de los secciones anteriores. Tenga en cuenta que el archivo y el blobstorage en el Vagrant box está aquí: /home/vagrant/var/ (no con la ruta del directorio de buildout /vagrant/buildout/)

Contribuyendo

Todos **están invitados** a contribuir. Los cambios de errores menores pueden ser empujados directamente en el repositorio, los cambios más grandes deben haber como un pull requests y discutirlos previamente en tickets.

Licencia

El entrenamiento "Maestría en Plone" es licenciado bajo la Creative Commons Attribution 4.0 International License.

Asegúrese que usted ha completado el Acuerdo de contribuidor.

Si usted no ha completado un Acuerdo de colaborador, todavía puede aportar. Póngase en contacto con el equipo de documentación, por ejemplo a través de la lista de correo o directamente enviar un correo a plone-docs@lists.sourceforge.net. Básicamente, todo lo que necesitamos es la confirmación por escrito de que usted está de acuerdo que su contribución puede ser bajo licencia Creative Commons. También puede agregar en un comentario con su pull request "Yo, <nombre completo>, estoy acuerdo en que esta publicado bajo licencia Creative Commons 4.0 International BY".

Contenidos:

CAPÍTULO 2

Introducción

¿Quien eres tu?

Cuéntenos acerca de usted:

- Nombre, compañía, país...
- ¿Como ha sido tu experiencia en Plone?
- ¿Cual ha sido tu experiencia en desarrollo de aplicaciones?
- ¿Cuales son tus expectativas con este tutorial?
- ¿Cual es tu editor favorito?
- **Si este entrenamiento incluirá los capítulos de desarrollo:**
    - ¿Usted entiende lo que hace el siguiente código HTML?

```
<div class="hiddenStructure"
     tal:repeat="num python:range(1, 10, 5)"
     tal:content="structure num"
     tal:omit-tag="">
  This is some weird sh*t!
</div>
```

La respuesta es:

```
1 6
```

- ¿Usted sabe lo que podría retornar el siguiente código Python?:

```
[(i.Title, i.getURL()) for i in context.getFolderContents()]
```

¿Qué haremos?

Algunas tecnologías y herramientas que usaremos en el entrenamiento:

- Para el entrenamiento básico o principiante:
    - Virtualbox
    - Vagrant

    - Ubuntu linux
    - A través de la Web (TTW)
    - Buildout
    - Un poco de XML
    - Un poco de Python
- Para los capítulos avanzados:
    - Git
    - Github
    - Prueba Git (Una introducción agradable para git y github)
    - TAL
    - METAL
    - ZCML
    - Python
    - Dexterity
    - Viewlets
    - JQuery

¿Qué no haremos?

Nosotros no cubriremos los siguientes tópicos:

- Archetypes
- Portlets
- z3c.forms
- Theming
- Testing
- i18n y locales
- Desarrollo, hosting y cacheo de contenidos
- grok
- Referencias / Relaciones

Otros tópicos están ligeramente cubiertos:

- Arquitectura de componentes Zope
- GenericSetup
- ZODB
- Security
- Permissions
- Desempeño y Cacheo de contenidos

¿Qué esperar?

Al final de los dos primeros días de entrenamiento, conocerá muchas de las herramientas necesarias para la instalación, integración y configuración de Plone. Usted será capaz de instalar los paquetes complementarios y conocerá algo sobre las tecnologías que subyacen a Plone y sus historias. Usted estará listo para extender sus habilidades a través de la lectura de libros como Practical Plone y la documentación de Plone.

Al final de los segundo dos días, usted no será un completo programador profesional de Plone, pero conocerá algunas de las características más potentes de Plone y debe ser capaz de construir un sitio web más Plone y debería ser capaz de construir un sitio web más complejo con temas y paquetes personalizados. Usted debe también ser capaz de encontrar dónde buscar instrucciones para hacer tareas que no cubrimos en este entrenamiento. Usted conocerá la mayoría la mayoría de las principales tecnologías de núcleo involucradas en la programación de Plone.

Si quieres convertirte en un desarrollador profesional de Plone o un integrador altamente sofisticado de Plone, usted definitivamente debe leer el libro de Martins y después volver a leerlo de nuevo mientras actualmente este haciendo un complejo proyecto.

Más importante es ¡usted debe practicar sus habilidades y no detenerte aquí pero debe seguir delante! una vía recomendada para esto es debe realizar los pasos del tutorial todo-app.

Documentación

Sigue el entrenamiento en http://plone-training.readthedocs.io/en/legacy/

Nota: Usted puede utilizar esta presentación para copiar y pegar el código, pero usted memorizará más si escribe usted mismo.

Lecturas recomendadas

- Martin Aspeli: Professional Plone4 Development
- Practical Plone
- Referencia de Zope Page Templates

Nota:

- ¡Puede detenernos y pregunte cuando tenga alguna duda!
- Díganos si hablamos muy rápido, para frenar o no ser lo suficientemente fuerte.
- Uno de nosotros siempre está ahí para ayudarle si usted está atascado. Por favor, danos una señal si usted está atascado.
- Vamos a hacer algunas pausas, la primera será a las XX.
- ¿Dónde está la comida, los baños?
- Por favor, alguien tome el tiempo que tomemos para cada capítulo (título incluido)
- Por favor, alguien puede indicar a errores
- Nos puede contactar después del entrenamiento: team@starzel.de

---

CAPÍTULO 3

Instalación y configuración

Instalando Plone

Plone 4.3.x requiere Python 2.7 y varias otras herramientas del sistema que no todos los sistemas operativos proporciona por defecto. Por lo tanto la instalación de Plone es diferente en cada sistema. Aquí hay algunas maneras que Python se puede utilizar:

- utilizar un Python que viene instalado previamente en su sistema operativo (en la mayoría de GNU/Linux y Mac OS tienen uno incorporado)
- use el python buildout
- instale los paquetes construido en su sabor de Linux
- homebrew (Mac OS X)
- PyWin32 (Windows)

MacOS 10.8 y Ubuntu 14.04 tendrán instalado por defecto un Python 2.7. Los usuarios de estos sistemas son los afortunados. Para ejecutar una versión de Plone anterior a la 4.0 usted necesita Python 2.4. No siempre es fácil de instalar.

La mayoría de los desarrolladores suelen utilizar su sistema primario para desarrollar Plone. Para configuraciones complejas que a menudo usan máquinas virtuales en Linux.

- OS X: Usa el buildout de python para compilar todas las versiones necesarias de Python y la herramienta homebrew para instalar algunas herramientas de Linux.
- Linux: Dependiendo de su sabor de Linux que usted podría tener que construir Python por ti mismo e instalar algunas herramientas.
- Windows: Alan Runyan (uno de los fundadores Plone) lo utiliza. Un inconveniente: Plone parece funcionar mucho más lento en Windows.

Plone ofrece múltiples opciones para que se instale:

1. Instaladores de un clic para Mac y de Windows
2. Unified installers (todos los Unix, incluyendo MacOS)
3. Un kit de instalación vagrant/virtualbox (todas las plataformas)
4. Utilice su propio Buildout

Puede descargar todo esto en https://old.plone.org/products/plone/releases/4.3.4

Para el entrenamiento vamos a utilizar la opción 3 y 4 para instalar y ejecutar Plone. Vamos a crear nuestro propio Buildout y lo extenderemos como deseamos. Pero lo haremos en una máquina vagrant. Para sus primeros experimentos

propios se recomienda la opción 2 o 3 (si usted tiene una portátil con Windows o al encontrar problemas en su proceso de instalación). Más adelante usted debería ser capaz de utilizar su propio Buildout (lo cubriremos más adelante).

Ver también:

- http://docs.plone.org/4/en/manage/installing/installation.html

Alojando Plone

Si usted desea alojando un sitio real de Plone por si mismo entonces ejecutarlo desde su computadora portátil no es una opción viable.

Puede alojar Plone...

- con uno de los muchos proveedores de alojamiento
- en un servidor privado virtual
- en los servidores dedicados
- en heroku puede ejecutar Plone para uso *gratis* usando el Heroku buildpack para Plone
- en la nube (por ejemplo, el uso de Amazon EC2 o Codio.com)

Ver también:

- Requerimientos de la instalación de Plone : http://docs.plone.org/4/en/manage/installing/requirements.html
- Ejecute Plone en un plan de 5$ http://www.stevemcmahon.com/steves-blog/plone-on-5-a-month
- Dónde alojar Plone: https://old.plone.org/documentation/faq/where-can-i-host-my-plone-site

Despliegue en ambientes de producción

La manera en que estamos montando un sitio Plone durante esta clase puede ser adecuado para un sitio pequeño - o incluso uno muy grande que no está muy ocupado - pero es muy probable que desee hacer mucho más si estás usando Plone para ninguna demanda.

- El uso de un servidor web de producción como Apache o Nginx para la reescritura de URL, SSL y combinación de múltiples, las mejores soluciones de su clase todo en un solo sitio web.
- El almacenamiento en caché de proxy inverso con una herramienta como Varnish para mejorar el rendimiento del sitio.
- El balanceo de carga para hacer el mejor uso de núcleo múltiple de CPU e incluso múltiples servidores.
- Optimización cabeceras de caché y esquemas de caché interna de Plone con el paquete plone.app.caching.

Y, usted necesita aprender estrategias eficientes para la copia de seguridad y rotación del archivo de los registros.

Todos estos temas son introducidos en la Guía para el despliegue e instalación de Plone en producción.

CAPÍTULO 4

Instalando Plone para el entrenamiento

Tenga en cuenta que necesita una conexión rápida a Internet durante la instalación, ¡ya que tendrá que descargar una gran cantidad de datos!

Advertencia: Si siente el deseo de probar ambos métodos a continuación (con y sin Vagrant), ¡asegúrese de usar diferentes directorios `training`! Las dos instalaciones no coexisten bien.

Instalando Plone sin Vagrant

Advertencia: Si usted **no** está acostumbrado a ejecutar Plone en su computadora portátil omitir esta parte y continuar con *Instalar VirtualBox*.

Si usted **esta** experimentado con ejecutar Plone en su propio ordenador portátil, te animamos a hacerlo, ya que tendrá ciertos beneficios:

- Usted puede usar el editor, que usted usa.
- Puede utilizar la receta Buildout llamada *omelette* tener todo el código de Plone a su alcance.
- Usted no tiene que cambiar entre diferentes del sistema operativo durante el entrenamiento.

Si se siente cómodo, por favor, trabaje en su propia máquina con su propio Python. Pero **por favor** asegúrese de que usted tiene un sistema que funcione, ¡ya que no queremos que pierda un tiempo valioso!

Nota: Si también desea seguir el entrenamiento de JavaScript e instalar las herramientas de desarrollo de JavaScript, necesita NodeJS instalado en su computadora de desarrollo.

Nota: Asegúrese de que su sistema esté bien preparado e instalado todos los requisitos previos necesarios. Por ejemplo, en Ubuntu / Debian, debes instalar lo siguiente:

```
sudo apt-get install python-setuptools python-virtualenv python-dev build-essential libssl-dev libxml
sudo apt-get install libreadline-dev wv poppler-utils
sudo apt-get install git
```

Para mas información o en caso de problemas, por favor ver la instrucciones instalación oficial.

Instalar Plone para el entrenamiento debe ser de esta forma si usted utiliza su propio sistema operativo (Linux o Mac):

```
$ mkdir training
$ cd training
$ git clone https://github.com/collective/training_buildout.git buildout
$ cd buildout
$ git checkout plone4
$ virtualenv-2.7 py27
```

Ahora puede ejecutar el buildout por primera vez:

```
$ ./py27/bin/python bootstrap.py
$ ./bin/buildout
```

Esto tomará algún tiempo y producir una gran cantidad de mensaje en consola de comando debido a que descarga y configura Plone. Una vez hecho esto usted puede comenzar a instancia con el siguiente comando:

```
$ ./bin/instance fg
```

La salida por la consola de comando es similar a:

```
2015-09-24 15:51:02 INFO ZServer HTTP server started at Thu Sep 24 15:51:02 2015
        Hostname: 0.0.0.0
        Port: 8080
2015-09-24 15:51:05 WARNING PrintingMailHost Hold on to your hats folks, I'm a-patchin'
2015-09-24 15:51:05 WARNING PrintingMailHost

**********************************************************************

Monkey patching MailHosts to print e-mails to the terminal.

This is instead of sending them.

NO MAIL WILL BE SENT FROM ZOPE AT ALL!

Turn off debug mode or remove Products.PrintingMailHost from the eggs
or remove ENABLE_PRINTING_MAILHOST from the environment variables to
return to normal e-mail sending.

See https://pypi.python.org/pypi/Products.PrintingMailHost

**********************************************************************

2015-09-24 15:51:05 INFO ZODB.blob (54391) Blob directory `.../buildout/var/blobstorage` is unused ar
2015-09-24 15:51:05 INFO ZODB.blob (54391) Blob temporary directory '.../buildout/var/blobstorage/tmp
2015-09-24 15:51:09 INFO Plone OpenID system packages not installed, OpenID support not available
2015-09-24 15:51:11 INFO PloneFormGen Patching plone.app.portlets ColumnPortletManagerRenderer to not
2015-09-24 15:51:11 INFO Zope Ready to handle requests
```

La salida por consola debe decir `INFO Zope Ready to handle requests` lo cual significa su instalación esta en correctamente iniciada.

Si abre su navegador Web local en la dirección URL http://localhost:8080 se ve que Plone está ejecutando. Ahora cree un nuevo sitio Plone haciendo clic en "Crear un nuevo sitio Plone". El nombre de usuario y la contraseña son "admin" (Nota: ¡Nunca haga esto en un sitio real!).

Ahora tiene un sitio Plone en funcionamiento y puede continuar con el siguiente capítulo. Puede detener la instancia en ejecución en cualquier momento utilizando la combinación de teclas `ctrl + c`.

Advertencia: Si hay un mensaje de error, usted debe bien intenta arreglarlo o utiliza la instalación Vagrant y continuar en este capítulo.

Instalando Plone sin vagrant

Para no perder demasiado tiempo con la instalación y la depuración de las diferencias entre los sistemas, utilizamos una máquina virtual (Ubuntu 14.04) para ejecutar Plone durante el entrenamiento. Contamos con Vagrant y VirtualBox para dar el mismo medio de ambiente de desarrollo para todos.

Vagrant es una herramienta para la creación de entornos de desarrollo completos. Lo usamos en conjunto con Oracle's VirtualBox para crear y administrar un entorno virtual.

Instalar VirtualBox

Vagrant utiliza VirtualBox de Oracle para crear entornos virtuales. Aquí hay un enlace directo a la página de descarga: https://www.virtualbox.org/wiki/Downloads. Utilizamos VirtualBox 4.3.x.

Instalar y configurar Vagrant

Obtenga la última versión de https://www.vagrantup.com/downloads.html para su sistema operativo y proceda a instalarlo.

Nota: En Windows hay un error en la versión reciente de Vagrant. Aquí están las instrucciones de cómo evitar la advertencia `Vagrant could not detect VirtualBox! Make sure VirtualBox is properly installed`.

Ahora el sistema dispone de un comando `vagrant` que se puede ejecutar en el terminal.

Nota: No es necesario instalar `NodeJS` como se mencionó en la sección anterior. Nuestra configuración Vagrant ya lo hace para usted.

En primer lugar, crear un directorio en el que desea hacer el entrenamiento.

Advertencia: Si ya tiene un directorio llamado `training` porque ha seguido las instrucciones de **Instalación de Plone sin vagrant**, debe eliminarlo, renombrarlo o usar otro nombre.

```
$ mkdir training
$ cd training
```

Configure Vagrant para instalar automáticamente los actuales guests adicionales. Usted puede optar por omitir este paso si tiene algún problema con ella.

```
$ vagrant plugin install vagrant-vbguest
```

Ahora descargue `plone_training_config.zip` y copie su contenido en su directorio de entrenamiento.

```
$ wget https://raw.githubusercontent.com/plone/training/master/_static/plone_training_config.zip
$ unzip plone_training_config.zip
```

El directorio del entrenamiento ahora debería contener el archivo `Vagrantfile` y el directorio se `manifests` el cual contiene a su ves muchos archivos.

Ahora empieza la configuración de la máquina virtual que se configura en el archivo `Vagrantfile`:

```
$ vagrant up
```

Esto toma un **muuuuuy largo tiempo** (entre 10 minutos y 1 hora dependiendo de su conexión a Internet y la velocidad del sistema) debido a que hace todos los pasos siguientes:

- descarga una máquina virtual (Official Ubuntu Server 14.04 LTS, también llamado "Trusty Tahr")
- establece la VM
- actualiza la VM
- instala varios paquetes de sistema necesarios para el desarrollo de Plone
- descargas y desempaquetar el directorio buildout-cache del instalador unificado, para obtener todos los paquetes egg para Plone
- clona el directorio buildout `training` en el directorio `/vagrant/buildout`
- construye Plone usando los paquetes eggs del directorio buildout-cache

---

Nota: A veces esto se detiene con el mensaje:

```
Skipping because of failed dependencies
```

Si esto sucede o usted tiene la sensación de que algo ha ido mal y la instalación no ha finalizado correctamente por alguna razón usted necesita ejecutar para intentar el comando siguiente para repetir el proceso. Esto sólo se repetirá los pasos que no han terminado correctamente.

```
$ vagrant provision
```

Usted puede hacer esto varias veces para arreglar los problemas, por ejemplo, si su conexión a red es limitada y los pasos no se pudieron terminar debido a esto.

Nota: Si mientras se ejecuta vagrant obtiene un error similar a:

```
ssh_exchange_identification: read: Connection reset by peer
```

Es posible que la configuración se haya estancado porque el BIOS de su equipo requiere que se habilite la virtualización. Consulte con el fabricante de su computadora sobre cómo habilitar correctamente la virtualización. Vea: https://teamtreehouse.com/community/vagrant-ssh-sshexchangeidentification-read-connection-reset-by-peer

---

Una vez Vagrant termina el proceso de aprovisionamiento, se puede acceder a la máquina virtual en ejecución.

```
$ vagrant ssh
```

Nota: Si usted tiene que usar Windows tendrás que iniciar sesión para con putty. Conecte a vagrant@127.0.01 en el puerto 2222. Con el usuario **y** contraseña son `vagrant`.

Ahora está iniciado con el usuario vagrant en `/home/vagrant`. Vamos a hacer todos los pasos del entrenamiento como este usuario.

En cambio usamos nuestra propia instancia de Plone durante el entrenamiento. Está en el directorio `/vagrant/buildout/`. Inicie en modo foreground con el comando `./bin/instance fg`.

```
vagrant@training:~$ cd /vagrant/buildout
vagrant@training:/vagrant/buildout$ ./bin/instance fg
2015-09-24 15:51:02 INFO ZServer HTTP server started at Thu Sep 24 15:51:02 2015
        Hostname: 0.0.0.0
        Port: 8080
2015-09-24 15:51:05 WARNING PrintingMailHost Hold on to your hats folks, I'm a-patchin'
2015-09-24 15:51:05 WARNING PrintingMailHost

********************************************************************************

Monkey patching MailHosts to print e-mails to the terminal.

This is instead of sending them.

NO MAIL WILL BE SENT FROM ZOPE AT ALL!

Turn off debug mode or remove Products.PrintingMailHost from the eggs
or remove ENABLE_PRINTING_MAILHOST from the environment variables to
return to normal e-mail sending.

See https://pypi.python.org/pypi/Products.PrintingMailHost

********************************************************************************

2015-09-24 15:51:05 INFO ZODB.blob (54391) Blob directory `.../buildout/var/blobstorage` is unused ar
2015-09-24 15:51:05 INFO ZODB.blob (54391) Blob temporary directory '.../buildout/var/blobstorage/tmp
2015-09-24 15:51:09 INFO Plone OpenID system packages not installed, OpenID support not available
2015-09-24 15:51:11 INFO PloneFormGen Patching plone.app.portlets ColumnPortletManagerRenderer to not
2015-09-24 15:51:11 INFO Zope Ready to handle requests
```

Nota: En raras ocasiones cuando está utilizando OSX con un juego de caracteres UTF-8, al iniciar Plone puede fallar con el siguiente error:

```
ValueError: unknown locale: UTF-8
```

En ese caso, usted tiene que poner la configuración de idioma y del teclado localizados en el archivo .bash_profile del usuario vagrant a su entorno local (como `en_US.UTF-8` o `de_DE.UTF-8`).

```
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

Ahora la instancia Zope que estamos usando se está ejecutando. Puede detener la instancia en ejecución en cualquier momento utilizando `ctrl + c`.

Si no lo hace, no se preocupe, su shell no está bloqueado. Escriba el comando `reset` (incluso si usted no puede ver el símbolo de sistema) y pulse la tecla RETURN, y debe ser visible otra vez.

Si abre su navegador local en http://localhost:8080 se ve que Plone está ejecutando en vagrant. Esto funciona porque Virtualbox reenvía al puerto 8080 del sistema invitado (el vagrant de Ubuntu) para el sistema host (su sistema operativo normal). Ahora cree un nuevo sitio Plone haciendo clic en "Crear un nuevo sitio Plone". El nombre de usuario y la contraseña son "admin" (Nota: ¡Nunca haga esto en un sitio real!).

El Buildout para este Plone está en una carpeta compartida. Esto significa lo ejecutamos en la caja vagrant desde el directorio `/vagrant/buildout` pero también podemos acceder a este desde fuera del propio sistema operativo y el uso de nuestro editor favorito. Usted encontrará el directorio de `buildout` en el directorio del `training` que creó en el principio junto al archivo `Vagrantfile` y `manifests`.

Nota: La base de datos y los paquetes python no son accesibles en su propio sistema, ya que los archivos grandes no se puede utilizar enlaces simbólicos en los directorios compartidos. La base de datos se encuentra en `/home/vagrant/var`, los paquetes Python están en `/home/vagrant/packages`.

Si usted tiene cualquier problema o pregunta, por favor envíenos un correo electrónico a team@starzel.de o cree un ticket en https://github.com/plone/training/issues.

¿Qué hace Vagrant?

La instalación se realiza automáticamente por las herramientas vagrant y puppet. Si quieres saber qué pasos se hacen realmente, por favor, consulte el capítulo what_vagrant_does.

Nota: Atención y Manejo Vagrant

Tenga en cuenta las siguientes recomendaciones para usar sus entornos virtuales de Vagrant:

- Utilice los siguiente comando `vagrant suspend` o `vagrant halt` para poner la virtualbox a "sleep" o "power it off" antes de intentar iniciar otra instancia de Plone en cualquier otro lugar de su máquina, si usa el mismo puerto. Eso se debe a que vagrant "reserva" el puerto 8080, e incluso si se detuvo Plone en vagrant, e indica que el puerto todavía está en uso por el sistema operativo invitado.
- Si ha terminado con una caja vagrant, y desea eliminarla, siempre recuerde ejecutar el comando `vagrant destroy` en ella antes de borrar el directorio que la contiene. De lo contrario, dejará su "fantasma" en la lista de cajas administradas por vagrant y posiblemente ocupar espacio en disco en su máquina.
- Ver `vagrant help` para todos las opciones de comando disponibles, incluyendo `suspend`, `halt`, `destroy`, `up`, `ssh` y `resume`.

CAPÍTULO 5

La Anatomía de Plone

Python, Zope, CMF, Plone, ¿como se integran entre si?

Zope2

- Zope es un framework de aplicación Web, el cual Plone corre encima de él.
- Ese sirve aplicaciones que comunica con usuarios vía HTTP.

Antes de Zope, generalmente era un servidor Apache el que debía llamar a un script y dar la petición como una entrada. La secuencia de comando sería entonces simplemente imprimir HTML en la salida estándar. El servidor Apache retornó eso al usuario. La apertura de las conexiones de base de datos, verificando las restricciones de permisos, la generación de HTML válido, la configuración de almacenamiento en caché, la interpretación de los datos del formulario y todo lo que tiene que hacer por su cuenta. Cuando la segunda petición llega, usted tiene que hacer todo de nuevo.

Jim Fulton pensaba que esto era un poco tedioso. Así que escribió el código para manejar las peticiones. Él creía que el contenido del sitio es orientado a objetos y que la URL de alguna manera debe apuntar directamente a la jerarquía de objetos, por lo que escribió una base de datos orientada a objetos, llamada ZODB.

El ZODB es una base de datos ACID totalmente compatible con integridad transaccional automático. Se asigna automáticamente el recorrido en la jerarquía de objetos a rutas URL, así que no hay necesidad de objetos "cableados" o nodos de bases de datos a las direcciones URL. Esto da Plone sus fáciles de generar direcciones URL amigables a las técnicas SEO.

Transversal a través de la base de datos de objeto tiene la seguridad comprobada en todos los puntos a través de listas de control de acceso muy detallado.

Una pieza que falta el cual es importante y a la ves complicada es la: `Acquisition`.

La adquisición, es una especie de magia. Imagine un sistema de programación en el que no accede al sistema de archivos y donde usted no necesita importar código. Usted trabaja con objetos. Un objeto puede ser una carpeta que contiene más objetos, una página HTML, datos, u otro script. Para acceder a un objeto, lo que necesita saber dónde está el objeto. Los objetos se encuentran por las rutas que parecen direcciones URL, pero sin el nombre de dominio. Ahora la Adquisición permite escribir una ruta incompleta. Una ruta incompleta es una ruta relativa, ese no establece explícitamente cual es la ruta que inicia desde la raíz, ese se inicia en relación con donde esta el contenido del objeto – eso es el contexto. Si Zope no puede resolver la ruta de un objeto relativo a su código, ese trata la misma ruta en la carpeta que lo contiene. Y a continuación, la carpeta que contiene la carpeta.

Esto puede sonar extraño, ¿qué gano yo con esto?

Usted puede tener diferentes datos o códigos en función de su contexto. Imagínese que usted desea tener imágenes de cabecera diferentes para cada sección de la página, a veces, incluso diferentes para una sub-sección específica de su sitio. Así se define una ruta a la header_image y pone una imagen de cabecera en la raíz de su sitio. Si quieres

una carpeta a una imagen de cabecera diferente, se pone la imagen de cabecera en esta carpeta. Por favor, tómese un minuto para dejar que esto se asimile mejor y piense, lo que esto le permite hacer.

- formularios de contacto con diferentes direcciones de correo electrónico por sección.
- diferentes estilos CSS para diferentes partes de su sitio.
- Un sitio, varios clientes, todo se ve diferente para cada cliente.

Al igual que con toda la magia de programación, adquisición exige un precio. El código de Zope debe ser escrito cuidadosamente si es necesario para evitar heredar efectos secundarios a través de la adquisición. La comunidad Zope expresa esto con la máxima Python (Monty): Cuidado con el Spammish de la Adquisición.

Básicamente esto es Zope.

Ver también:

http://www.zope.org/the-world-of-zope y http://docs.zope.org/zope2/zope2book/

Content Management Framework

- CMF (Content Management Framework) es un complemento para Zope para construir sistemas de gestión de contenido (como Plone).

Después de muchos sitios web creados con éxito basado en Zope, una serie de requisitos que se repiten surgieron, y algunos desarrolladores Zope comenzaron a escribir CMF, el Content Management Framework.

La CMF ofrece muchos servicios que le ayudan a escribir un CMS basado en Zope. La mayoría de los objetos que ve en la ZMI son parte de la CMF de alguna manera.

Los desarrolladores detrás CMF no ven CMF como un producto listo para usar CMS. Ellos crearon un sitio CMS que era utilizable fuera de la caja, pero dejó deliberadamente feo, porque usted tiene que personalizar todas formas.

Todavía estamos en los tiempo de la prehistoria aquí. En ese entonces no existían los paquetes eggs (paquetes Python), Zope no consistía en mas de 100 componentes de software independientes, pero era una gran conjunto.

Muchas partes de Plone se derivan de la CMF, pero es una herencia mixta. El CMF es un proyecto independiente de software, y se ha movido a menudo más lentamente que Plone. Plone está eliminando gradualmente la dependencia de la mayoría de las partes de la CMF.

Zope Toolkit / Zope3

- Zope 3 fue originalmente creada como una profunda re-escritura de Zope.
- Plone usa partes provistas por el Zope Toolkit (ZTK).

Por desgracia, nadie empezó a usar Zope 3, nadie migró a Zope 3, porque nadie sabía cómo hacerlo.

Pero había muchas cosas útiles en Zope 3 que las personas querían usar en Zope 2, por lo que la comunidad Zope adaptado algunas partes para que pudieran utilizarlos en Zope 2. A veces, una clase wrapper de algún tipo era necesario, estos por lo general son ofrecidos por paquetes de los cinco espacios de nombres.

Para hacer la historia completa, ya que la gente se quedó en Zope 2, la comunidad Zope renombrado Zope 3 a Bluebream, por lo que la gente no piensa que Zope 3 era el futuro. No era nada más.

Ver también:

https://old.plone.org/documentation/faq/zope-3-and-plone

Zope Component Architecture (ZCA)

La Zope Component Architecture, que fue desarrollado como parte de Zope 3, es un sistema que permite componente de tipo pluggability y despacho complejo basado en objetos que implementan una interfaz (una descripción de una funcionalidad). Pyramid, y el servidor de aplicaciones Web Python independiente, utiliza la ZCA "bajo la capucha" para ejecutar despacho de vistas y otras tareas de configuración de aplicación.

Pyramid

- Pyramid es un framework de desarrollo de aplicaciones Web en Python, el cual a veces es visto como el sucesor natural de Zope.
- Eso hace menos que Zope, es muy configurable y usa la Zope Component Architecture.

Se puede utilizar con una base de datos relacional en lugar de ZODB si quieres, o usar las dos bases de datos o ninguno de ellos.

Aparte del hecho de que Pyramid no se vio obligado a soportar todas las funcionalidades legado, que puede hacer las cosas más complicadas, el desarrollador original tenía una postura muy diferente sobre cómo el software debe ser desarrollado. Mientras tanto Zope y Pyramid tienen una buena de la pruebas de software, Pyramid tiene una buena documentación; algo que estaba muy descuidado en Zope, y en ocasiones en Plone también.

Ya sea que si ¿la arquitectura de componentes es mejor en Pyramid o no?, no nos atrevemos a decir, pero nos gusta más. Pero tal vez es sólo porque fue documentada.

CAPÍTULO 6

El caso de estudio

Para este entrenamiento construiremos un Sitio Web para una conferencia ficticia de Plone.

Detalle

La conferencia de Plone toma lugar cada año y todos los desarrolladores al menos intentan ir.

Requerimientos

- Como un visitante, yo quiero encontrar información actualizada en la conferencia.
- Como visitante, yo quiero registrarme para la conferencia.
- Como visitante, yo quiero ver las charlas y ordenarlas por mis preferencias.
- Como un ponente, yo quiero ser capaz de enviar charlas.
- Como un ponente, yo quiero ver y editar mis charlas enviadas.
- Como miembro del jurado, yo quiero votar en las charlas.
- Como miembro del jurado, yo quiero decidir, cuales charlas aceptar, y cuales no.
- Como organizador, yo quiero ver una lista de todas las charlas propuestas.
- Como organizador, yo quiero tener un resumen acerca de cuantas personas están registradas.

Tenga en cuenta que todos nuestros requisitos deben conectar los roles con sus capacidades. Esto es importante porque vamos a querer limitar las capacidades de aquellos a los que asignamos roles particulares.

CAPÍTULO 7

Las características de Plone

En profundidad el Manual de usuario (en idioma Español): https://old.plone.org/countries/ve/publicaciones/plone-para-usuarios-principiantes

Ver también la versión original del manual (en idioma Ingles) en las siguientes direcciones URL: http://old.plone.org/documentation/manual/plone-4-user-manual y http://docs.plone.org/4/en/working-with-content/index.html

Iniciando y Deteniendo Plone

Tenemos el control de Plone con un pequeño script llamado "instance" a continuación un ejemplo de su uso:

```
$ ./bin/instance fg
```

Esto comando anterior, arranca Plone en el modo foreground para que podamos ver lo que está haciendo mediante el control de los mensajes de consola. Este es un método de desarrollo importante. Tenga en cuenta que cuando Plone se inicia en el modo foreground, es también automáticamente en el modo de desarrollo. Modo de desarrollo da una mejor respuesta, pero es mucho más lento, particularmente en Windows.

Usted puede detener el proceso presionando `ctrl + c`.

El script **`instance`** ofrece las siguientes opciones:

```
$ ./bin/instance fg
$ ./bin/instance start
$ ./bin/instance stop
$ ./bin/instance -O Plone debug
$ ./bin/instance -O Plone run myscript.py
4 ./bin/instance adduser
```

Dependiendo del equipo, puede tardar hasta un minuto hasta que Zope le dirá que ya está listo para servir peticiones. En un portátil decente debe estar en ejecución en menos de 15 segundos.

Una instalación estándar escucha en el puerto 8080, así que vamos a echar un vistazo a nuestro sitio Zope visitando http://localhost:8080

¡Como puede ver, no hay Plone todavía!

Tenemos un Zope corriendo con una base de datos pero no el sitio Plone. Pero, afortunadamente, hay un botón para crear un sitio Plone. Haga clic en el botón `Crear un nuevo sitio Plone` (inicio de sesión: admin:admin). Esto abre un formulario para crear un sitio Plone. Utilice `Plone` como el identificador del sitio.

Ahora tiene la opción de seleccionar algunos complementos antes de crear el sitio. Dado que vamos a utilizar Dexterity desde el principio seleccionamos el complemento `Dexterity-based Plone Default Types`.

De esta manera, incluso el contenido inicial en nuestra página será construido con dexterity por el complemento `plone.app.contenttypes` que será el predeterminado en Plone 5.

Usted será automáticamente redirigido al sitio nuevo.

Nota: Plone tiene muchos mensajes. Contienen información importante. ¡Léalos y asegúrese de entender su contenido!

Usuarios

Vamos a crear nuestros primeros usuarios en Plone. Hasta ahora hemos utilizado el usuario admin (admin:admin) configurado en el buildout. Este usuario es a menudo se le llama "el administrador de Zope" y con este usuario no se gestiona en Plone mas sólo es usado para la gestión de Zope. Por lo tanto el usuario le faltan algunas características como el correo electrónico y el nombre completo y no va a ser capaz de utilizar algunas de las características de Plone. Sin embargo, el usuario tiene todos los permisos posibles. Al igual que con el usuario root de un servidor, que es una mala práctica hacer uso innecesario del raíz de Zope. Se usa para crear sitios Plone y sus usuarios iniciales, pero no mucho más.

También puede agregar los usuarios Zope a través del terminal ejecutando el siguiente comando:

```
$ ./bin/instance adduser <someusername> <supersecretpassword>
```

De esa manera usted puede tener acceso a las bases de datos que recibe de los clientes donde no tienes usuario Plone.

Para agregar un nuevo usuario haga clic en el nombre *admin* ubicado en la esquina superior derecha y luego en *Configuración del sitio*. Este es el panel de control de Plone. También puede acceder a él por la dirección URL a http://localhost:8080/Plone/@@overview-controlpanel

Haga clic en *Usuarios y Grupos*, entonces añada un usuario. Si ha configurado un servidor de correo, Plone puede enviarle un correo electrónico con un enlace a un formulario donde puede elegir una contraseña. Establezca una contraseña aquí porque no ha configurado aun un servidor de correo.

Estableciendo a este usuario con su nombre un Administrador del sitio.

A continuación, cree otro usuario llamado `testuser`. Hágalo un usuario normal. Puede utilizar este usuario para ver cómo luce Plone y se comporta a los usuarios que no tienen permiso de administrador.

Ahora vamos a ver el sitio en tres (03) navegadores diferentes, con tres roles diferentes:

- como anónimo.
- como editor.
- como administrador.

Configurar un servidor de correo

Tenemos que configurar un servidor de correo desde temprano vamos a crear algunas acciones de contenido que envían mensajes de correo electrónico cuando el nuevo contenido se pone en nuestro sitio.

- Servidor: `mail.gocept.net`
- Nombre de usuario: `training@neww.de`
- Contraseña: `training2014`

Por favor, no abuses de esto. Vamos a deshabilitar esta cuenta después de la capacitación.

Tutorial de la interfaz de usuario

Vamos a ver lo que está ahí...

- *portal-top*:
  - *personaltools*: nombre, salir etc.
  - *logo*: con un enlace a la página principal
  - *buscar*
  - *navegación global*
- *portal-columns*: un contenedor que contienen:
  - *portal-column-one*: portlets (cajas configurables como herramientas para navegación, noticias etc.)
  - *portal-column-content*: el contenido y el editor
  - *barra de edición*: opciones de edición del contenido
  - *portal-column-two*: portlets
- *portal-footer*: viewlets

Estos son también los clases CSS de los respectivos de DIV del HTML. Si usted quiere hacer la temas visuales usted los necesita.

En la barra de edición, encontramos opciones que afectan el contexto actual...

- *Contenidos*
- *Visualizar*
- *Editar*
- *Reglas*
- *Compartir*
- *Mostrar*
- *Agregar nuevo...*
- *Estado*

Algunas opciones de la barra de edición sólo muestran cuando proceda; por ejemplo, la carpeta `Contenidos` y `Agregar nuevo...` sólo se muestran para carpetas. Las opción `Reglas` esta actualmente invisible porque no tenemos reglas de contenido disponible.

Tipos de Contenidos

Editar una página:

- *Edite el documento front-page*
- *Titulo* `Plone Conference 2014, Bristol`
- *Descripción* `Tutorial`
- *Texto* `...`

Crear un estructura del sitio:

- Agregar carpeta “El Evento” y que contenga...

    - Carpeta “Charlas”
    - Carpeta “Entrenamientos”
    - Carpeta “Sprint”
- En la ruta /news: Agregar una Noticia “¡Sitio Web de la Conferencia esta en linea!” con una imagen
- En la ruta /news: Agregar una Noticia “¡Enviar sus charlas!”
- En la ruta /news: Agregar un Evento “Fecha limite para enviar charlas” Fecha 10.10.2014
- Agregar carpeta llamada “Registro”
- Eliminar la carpeta llamada “Members” (Usuarios)
- Agregar carpeta llamada “Intranet”

Los tipos de contenidos por defecto:

- Documento
- Noticia
- Evento
- Archivo
- Imagen
- Enlace
- Carpeta
- Colección

Nota: Por favor, tenga en cuenta que usamos plone.app.contenttypes para el entrenamiento. Por lo tanto, los tipos se basan en Dexterity y un poco diferente de los tipos que se encuentran en un sitio Plone 4.3.x por defecto.

Carpetas

- Ir a ‘el-evento’
- explicar title/id/url
- explicar /folder_contents
- cambiar orden
- acciones masivas
- menú desplegable “Mostrar”
- default_pages
- Agregar una página a ‘el-evento’: “El Evento” y hacerlo su página por defecto
- El futuro: `wildcard.foldercontents`

Colecciones

- agregar nueva colección: “todos los contenidos que este pendiente por revisión”.
- explicar la colección por defecto de eventos en http://localhost:8080/Plone/events/aggregator/edit
- explicar Tópicos
- mencionar portlets de colección
- multiples rutas a consultar
- restricciones, ej. `/Plone/folder::1`

Reglas de Contenido

- ¡Crear una nueva regla “una nueva charla está presente”!
- Nuevo contenido en Carpeta “Charlas” -> Enviar correo a revisores.

Histórico

Mostrar y explicar; el control de versiones y su relación con los tipos.

Administrar usuarios y grupos

- agregar / editar / eliminar Usuarios
- roles
- grupos
  - Agregar grupo “Editores” y agregar el usuario ‘editor’ a ese.
  - Agregar grupo: `orga`
  - agregar grupo: `jury` y agregue usuario ‘jurymember’ a ese.

Flujos de trabajos

Echa un vistazo al menú desplegable `Estado` en la barra de edición en la página principal. Ahora, vaya a una de las carpetas que acaba de añadir. La página principal tiene el estado `Publicado` y el nuevo contenido es `Privado`.

Echemos un vistazo a las transiciones de estado disponibles para cada tipo. Podemos hacer un artículo publicado en privado y un artículo privado se pueda publicar. También podemos enviar un artículo para su revisión.

Cada uno de esos estados conecta los roles a permisos.

- En el estado `Publicado`, el contenido está disponible para los visitantes anónimos;
- En el estado `Privado`, el contenido es sólo visible para el autor (propietario) y los usuarios que tienen el rol local `Puede ver` para el contenido.

Un estado de flujo de trabajo es una asociación entre un rol y uno o más permisos. Pasar de un estado a otro es una `transición`. Las transiciones (como `Enviar a revisión`) pueden tener acciones - como la ejecución de una regla de contenido o un script - asociado a ellos.

Un conjunto completo de estados de flujo de trabajo y las transiciones constituyen un `flujo de trabajo`. Plone le permite seleccionar entre varios flujos de trabajo preconfigurados que son apropiados para los diferentes tipos de sitios. Tipos de contenido individuales pueden tener su propio flujo de trabajo. O, lo que es particularmente interesante, sin flujo de trabajo. En ese caso, que se aplica inicialmente a presentar y la subida de imágenes, el objeto de contenido hereda el flujo de trabajo de su contenedor.

Nota: Una rareza en el todo de los flujos de trabajo estándar de Plone: un elemento de contenido puede ser visible incluso si su contenedor no es. Haciendo un contenedor privado **no** hacen automáticamente su contenido privado.

Lea más en: http://docs.plone.org/4/en/working-with-content/collaboration-and-workflow/index.html

Copia de trabajo

El contenido publicado, incluso en un entorno de Intranet, puede plantear un problema especial para la edición. Puede ser necesario revisar antes de que los cambios se hacen disponible. De hecho, el autor original podría incluso no tener permiso para cambiar el documento sin revisión. O bien, puede que tenga que hacer una edición parcial. En cualquiera de los casos, puede ser deseable para que los cambios sean visibles inmediatamente.

El producto *Soporte de Copia de Trabajo* resuelve este problema mediante la adición una acción para *Retirar revisión* o *Guardar nueva revisión* de cambios para el contenido - disponible en el menú de acciones. Un elemento de contenido puede *guardar nueva revisión*, trabajado en una copia del elemento y después se *guardar nueva revisión* de nuevo. O *Cancelar retirada de revisión* si los cambios no eran aceptables. Hasta no guardar una nueva revisión de los cambios, el contenido anterior sera visible.

Mientras este complemento viene incorporado en Plone, el soporte de copia de trabajo no es una necesidad común. Así que, si usted lo necesita, usted necesita para activarlo a través de la página de configuración de Complementos. A menos que se active este complemento, las opciones de *Retirar revisión* / *Guardar nueva revisión* no son visibles.

Nota: Soporte de Copia de Trabajo no está disponible para los tipos de contenido creados a través de Dexterity. Esto está en el camino (es un trabajo en proceso).

Flujos de trabajo Placeful

Usted puede necesitar tener diferentes flujos de trabajo en diferentes partes de un sitio. Por ejemplo, creamos una carpeta intranet. Dado que este es para uso de nuestros organizadores de la conferencia - pero no para el público - el sencillo flujo de trabajo que deseamos utilizar para que el resto del sitio no será deseable aquí.

El paquete `Soporte de Copia de Trabajo` te da la posibilidad de configurar diferentes flujos de trabajo en las diferentes secciones de un sitio. Normalmente, se utiliza para configurar un flujo de trabajo especial en una carpeta que regirá todo bajo esa carpeta. Ya que tiene efecto en un "lugar" en un sitio, este mecanismo es a menudo llamado "flujo de trabajo Placeful".

Tanto como el Soporte de Copia de Trabajo, como el Soporte de Política de Flujo de trabajo Placeful viene incorporados en Plone, pero tiene que ser activadas a través de la página de configuración de Complementos. Una vez se ha añadido, una opción `Política...` aparecerá en el menú de Estado para permitir el establecimiento de una política de flujo de trabajo Placeful.

Nota: Soporte de Política de Flujos de trabajo aún no está disponible para los tipos de contenido Carpeta creados a través de Dexterity. Esto está en el camino (es un trabajo en proceso).

CAPÍTULO 8

Configurando y personalizando Plone a través de la Web

El panel de control

Las partes más importantes de Plone pueden ser configuradas en el panel de control.

- Haga clic en el nombre de usuario, en la esquina superior derecha de su pantalla
- Haga clic en “Configuración del Sitio”

Nosotros explicaremos cada página y mencionaremos algunas cosas que puedes hacer aquí.

1. Complementos (después)
2. Calendario
3. Registro de Configuración
4. Reglas de contenido (Ya lo sabemos)
5. Discusión
6. Edición
7. Errores
8. Filtrado HTML
9. Manejo de Imágenes
10. Idioma
11. Correo
12. Mantenimiento
13. Etiquetado
14. Navegación
15. Buscar
16. Seguridad
17. Sitio
18. Sindicación
19. Temas
20. Editor visual TinyMCE

21. Tipos
22. Usuarios y Grupos
23. Interfaz de administración de Zope (Aquí hay dragones, ¡Precaución!)

Por debajo en los enlaces encontraras información de las versiones en tu Plone, Zope y Python y un indicador de cualquier cosa que usted está funcionando en el modo de producción o desarrollo.

Portlets

- `@@manage-portlets`
- Interfaz de usuarios para editores inteligentes de contenido
- Varios tipos
- La configuración de Portlets es heredada
- Son administrados
- Ordenando / ponderación
- En el futuro: será reemplazado por tiles

Ejemplo:

- Valla a la dirección URL http://localhost:8080/Plone/@@manage-portlets
- Agregue un portlet "Patrocinadores" en el lado derecho.
- Remueve los nuevos portlets y agrega uno nuevo en el lado izquierdo.
- Ve a la carpeta de training: http://localhost:8080/Plone/training y has clic `Administrar portlets`
- Agregue un portlet estático. "Entrenamiento destacada: Conviértete en un Rockstar de Plone en la ¡Maestría en Plone!"

Viewlets

- `@@manage-viewlets`
- Viewlet no poseen Interfaz de usuario
- No recomendado para editores de contenido
- No es localmente agregable, no posee herencia configurable.
- Usado globalmente (depende del código)
- ¿Sera reemplazado por los tiles?
- El código es mucho más simple (crearemos uno mañana)
- En tiempo real en la herramienta administradores de viewlet, pueden ser anidadas (agregando un viewlet que contenga un administrador de viewlet)
- El reordenamiento a través de la Web solo puede ser dentro de su mismo administrador de viewlet
- el código decide cuando, dónde y que muestra

Portlets guarda datos, los Viewlets usualmente no lo hacen, los Viewlets a veces son usados para elementos de Interfaz de usuario.

Ejemplo:

- Valla a la dirección URL http://localhost:8080/Plone/@@manage-viewlets
- Ocultar Colofón (Ocultar pie de página)

ZMI

Valla a dirección URL http://localhost:8080/Plone/manage

Zope es la base de Plone. Aquí puedes acceder igualmente al funcionamiento interno de Zope y Plone.

Nota: Aquí fácilmente se puede romper su sitio así que usted debe saber lo que está haciendo.

Nosotros sólo cubrimos las tres partes de la personalización en el ZMI ahora. Más tarde, cuando agreguemos nuestro propio código fuente vamos a volver a la ZMI y lo buscaremos.

En algún momento tendrá que aprender, sobre todos los objetos que hay allí. Pero no hoy.

Acciones (portal_actions)

- Las acciones son principalmente enlaces. Pero enlaces **realmente flexibles**.
- Las acciones son configurable a través de la Web y a través de código fuente.
- Estas acciones son usualmente iteradas sobre viewlets y mostradas.

Ejemplos:

- Enlaces en el pie de página (site_actions)
- Acciones desplegables (folder_buttons)

Las acciones tienen propiedades como:

- descripción
- url
- i18n-domain
- condition
- permissions

site_actions

Estos son los enlaces al fondo de la página:

- Mapa del Sitio
- Accesibilidad
- Contacto
- Configuración del Sitio

Queremos un nuevo enlace a la información legal, llamado “Aviso Legal”.

- Ir a `site_actions` (lo sabemos porque lo chequeamos en `@@manage-viewlets`)
- Agregar un CMF Action, haciendo clic en el menú desplegable ubicado en la esquina superior derecha, seleccione `CMF Action` y luego hace clic en el botón `Add` y defina el ID como `Aviso legal`
- Asigna dirección URL a `string:${portal_url}/imprint`
- Deje *condition* en blanco
- Asigne el permiso a `View`
- Guardar

Explicar

- Verifique si el enlace de esta en la página funciona
- Cree un nuevo Documento llamado *Aviso legal* y publíquelo

Ver también:

http://docs.plone.org/4/en/develop/plone/functionality/actions.html

Navegación global

- La navegación horizontal es llamada `portal_tabs`
- Ir a `portal_actions` > `portal_tabs` aquí esta el enlace
- Edite `index_html`

¿Donde esta la navegación?

La navegación muestra objetos de contenido, los cuales están en el raíz del directorio de Plone, además de las acciones en la herramienta `portal_tabs`

Edite y explique `index_html`

Configurando la navegación por si mismo es hecho en cualquier parte: http://localhost:8080/Plone/@@navigation-controlpanel

Si hay tiempo explique:

- usuario > deshacer (¡chevere!)
- usuario > entrar/salir

Skins (portal_skins)

En la herramienta portal_skins podemos cambiar ciertas imágenes, archivos css y plantillas.

- portal_skins es una tecnología depreciada
- Haremos únicamente algunos mínimos cambios aquí.

Plone 5 tendrá una gran cantidad de funcionalidades que aún vive en la herramienta portal_skins.

Antes solíamos usar como parte del entrenamiento el complemento plone.app.themeeditor que tiene una interfaz de usuario mucho más agradable que la ZMI pero también tiene dependencias que no son compatibles con ZopeSkel y no es muy utilizada.

Cambie algunos archivos css

- Ir a ZMI
- Ir a la herramienta portal_skins
- Ir a la herramienta plone_styles
- Ir a `ploneCustom.css`
- Haga clic en `customize`

Ingrese el siguiente CSS:

```
#visual-portal-wrapper {
    margin: 0 auto;
    position: relative;
    width: 1024px;
}
```

Haga clic en `Save` y verifique los resultados en otra pestaña del navegador. ¿Cómo sucedió eso?

La interfaz de usuario deja mucho que desear. En un contexto profesional esto es inútil (sin control de versiones, no resaltado de sintaxis, etc. pp.). Pero todo el mundo usa la herramienta portal_skins, para hacer arreglos rápidos a los sitios que ya están en línea.

Vamos a añadir un poco más estilos CSS para hacer nuestro sitio un poco adaptativo:

```
@media only screen and (max-width: 980px) {
   #visual-portal-wrapper {
       position: relative;
       width: auto;
   }
}

@media only screen and (max-width: 768px) {
   #portal-columns > div {
       width: 97.75%;
       margin-left: -98.875%;
       clear: both;
   }

   .searchButton,
   .searchSection {
       display: none;
   }
}
```

Cambia el logotipo

Cambiemos el Logotipo.

- Baje un logotipo de la ploneconf: https://www.starzel.de/plone-tutorial/ploneconf-logo-2014/download
- Ir a `portal_skins` / `plone_images`
- Haga clic en `logo.png`, luego haga clic en `Customize` y suba el Logotipo.

Ver también:

http://docs.plone.org/4/en/adapt-and-extend/change-the-logo.html

la herramienta portal_view_customizations

Cambiar el pie de página

- Ir a `portal_view_customizations`
- Busque `plone.footer`, haga clic y personalice
- Reemplace el contenido con lo siguiente

```
<div i18n:domain="plone"
     id="portal-footer">
   <p>&copy; 2014 by me! |
     <a href="mailto:info@ploneconf.org">
      Contact us
     </a>
   </p>
</div>
```

Ver también:

http://docs.plone.org/4/en/adapt-and-extend/theming/templates_css/skin_layers.html

Registro de CSS (portal_css)

- Ir a ZMI > `portal_css`
- En el fondo esta el archivo `ploneCustom.css`
- Deshabilite `Development mode`: Los archivos CSS serán mezclados y teniendo un cache-key.

Nota: El JavaScripts Registry (la herramienta portal_javascripts) es muy similar.

La función de la mezcla resuelve un gran problema: nos gustaría desarrollar nuestros recursos CSS y JS de una forma granular, pero también nos gustaría reducir al mínimo su tamaño en las peticiones HTTP.

Nota: Al final de un proceso de desarrollo, un poco de reordenamiento para minimizar solicitudes pueden tener un efecto muy agradable. A menudo es posible reducir las solicitudes a un número muy pequeño para los visitantes anónimos.

Más herramientas en el ZMI

Hay muchos más elementos notables en el ZMI, los visitaremos más tarde.

- acl_users
- error_log
- la herramienta portal_properties
- la herramienta portal_setup
- la herramienta portal_workflow
- la herramienta portal_catalog

Resumen

Puedes configurar y personalizar muchas cosas en Plone a través de la Web. Las opciones más importantes son accesibles en el panel de control de Plone pero aun más están escondidas en la ZMI. La cantidad y presentación de la información es abrumadora, pero usted conseguirá la caída de ella a través de mucha práctica.

CAPÍTULO 9

Extendiendo Plone

Zope es extensible y también lo es Plone.

Si desea instalar un complemento, se va a instalar un paquete Egg - una forma de paquete Python. Los paquetes Eggs consisten en archivos Python junto con otros ficheros necesarios como plantillas de páginas y similares, y un poco de metadatos, que se incluye en un solo archivo.

Hay una gran variedad de paquetes compatibles con Plone disponibles. La mayor parte se enumeran en el Python Package Index. Una lista más navegable está disponible en la antigua lista complemento de Plone.org y la nueva lista complemento de Plone.org. El repositorio de código fuente para muchos complementos públicos Plone esta en la organización Collective de GitHub. También puede crear sus propios paquetes o mantener repositorios personalizados.

En Zope los paquetes Eggs son menores "en edad". Zope necesitaba algo como los paquetes eggs antes no había eggs, y los desarrolladores Zope escribieron su propio sistema. En la antigüedad los sistemas obsoletos Plone contenían una gran cantidad de código fuente no incluidos en un paquete egg. El código fuente antiguo no tenía metadatos para registrar las cosas, en vez que necesitaba un método de configuración especial. No necesitamos este método, pero podríamos verlo en otro código fuente. Normalmente se utiliza para registrar el código Archetypes. Los Archetypes es el viejo sistema de tipo de contenido. Ahora utilizamos el nuevo sistema de tipo de contenido Dexterity.

Tecnologías de extensión

¿Cómo extender Plone?

Esto depende de que tipo de extensión usted quiere crear.

- Puede crear extensiones con nuevos tipos de objetos para añadir a su sitio Plone. Por lo general, estos son los tipos de contenido.
- Puede crear una extensión que cambia o se extiende la funcionalidad. Por ejemplo, para cambiar la forma en que Plone muestra los resultados de búsqueda, o hacer a las imágenes que sean analizadas en búsqueda del algún texto que incluya en la misma mediante la adición de un convertidor de JPG a texto.

skin_folders

¿Te acuerdas del concepto Adquisición? Las Skin Folders se extiende los conceptos de Adquisición. Su sitio Plone tiene una carpeta denominada `portal_skins`. Esta carpeta tiene una serie de sub-carpetas. La carpeta `portal_skins` tiene una propiedad que define el orden en que las búsquedas de Plone para atributos u objetos en cada sub-carpeta.

El logo de Plone está en una carpeta skin.

Por defecto, su sitio tiene una carpeta `custom`, y los elementos son primero buscados en esa carpeta.

Para personalizar el logotipo, usted copia en la carpeta `custom`, y lo cambia allí. De esta manera usted puede cambiar las plantillas, los estilos CSS, imágenes y comportamientos, ya que un contenedor puede contener scripts python.

La personalización del estilo de la carpeta Skins se puede lograr a través de la Web usando la ZMI, o con paquetes de complementos. Muchos paquetes de estilo antiguo crean su propia carpeta de skin y la agregan a la capa de skin para Plone cuando se instala.

GenericSetup

El siguiente paso es *GenericSetup*. Como su nombre lo indica claramente, *GenericSetup* es parte de CMF.

Me temo que GenericSetup es difícil de dominar.

GenericSetup le permite definir la configuración persistente en archivos XML. *GenericSetup* analiza los archivos XML y actualiza la configuración persistente según la configuración. ¡Este es un paso que tiene que correr por su cuenta!

Usted verá muchos objetos en Zope o la ZMI que se pueden personalizar a través de la web. Si estás personalizaciones están listas, usted puede exportar su configuración a través *GenericSetup* e importarlo de nuevo.

Normalmente se usa directamente *GenericSetup* para cambiar los flujos de trabajo o añadir nuevas definiciones de tipo de contenido.

Los perfiles GenericSetup también pueden ser construidos en los paquetes de Python. Cada paquete que aparece en la lista de paquetes de complemento dentro de una instalación Plone tiene un perfil de GS que detalla cómo encaja en Plone. Los paquetes que forman parte de la propia Plone pueden tener perfiles GS, pero están excluidos de la lista activa / inactiva.

Arquitectura Componente

La última manera de extender Plone es a través de los *Componentes*.

Un poco de historia está a la orden.

Cuando comenzó Zope, el Diseño orientado a objetos fue la **bala de plata**, es decir, una solución rápida a un problema difícil.

Diseño orientado a objetos es bueno en el modelado herencia, pero se rompe cuando un objeto tiene varios aspectos que forman parte de múltiples taxonomías.

Algunos lenguajes de programación orientados a objetos como Python manejan esto a través de la herencia múltiple. Pero no es una buena manera de hacerlo. Los objetos Zope tienen más de 10 clases de base. Demasiados espacios de nombres hace al código difícil de mantener. ¿De dónde vino ese método / atributo?

Después de un tiempo, XML y los componentes se convirtieron en la próxima **bala de plata** (¿Alguien recuerda J2EE?).

Basado de sus experiencias con Zope en el pasado, ellos pensaron que un sistema de componente es configurado a través de XML podría ser la forma cuidar que el código sea más fácil de mantener.

A medida que los nuevos conceptos eran radicalmente diferentes de los viejos conceptos de Zope, los desarrolladores Zope cambió el nombre del nuevo proyecto de Zope 3. Pero no ganaron fuerza, la comunidad de alguna manera le cambió el nombre a Bluebream y esto lo extinguió.

Pero la arquitectura de componentes en sí es bastante acertada y el desarrollador Zope extrajo el Zope Toolkit. El Zope Toolkit es parte de Zope y los desarrolladores Plone lo utilizan ampliamente.

Esto es lo que usted desea utilizar.

¿Cuáles son los componentes?, ¿Que es el ZCML?

¿Cuál es la manera más simple absoluta para ampliar la funcionalidad?

Monkey Patching.

Eso significa que usted puedo cambiar el código en otros archivos mientras mi archivo se carga.

Si usted quiere tener un registro extensible de iconos para los distintos tipos de contenido, usted podría crear un diccionario global, y que implementa en un nuevo icono para un tipo de contenido diferente, podría añadir una entrada en mi diccionario durante el tiempo de importación.

Este enfoque, como subclases mediante herencia múltiple, no escala. Múltiples extensiones pueden sobrescribir los demás, usted debería explicar a la gente que tiene que reordenar las importaciones, y luego, de repente, se le importar característica A antes que B, B antes de C y C antes de A, o de lo contrario la aplicación no funcionará.

La arquitectura de componentes de Zope con su configuración ZCML es la respuesta para sus problemas.

Con ZCML declarar las utilidades, los adaptadores y las browser views en ZCML, el cual es un dialecto de XML que significa Zope Component Markup Language.

Los componentes se diferencian entre sí por las interfaces (definiciones formales de funcionalidad) que requieren o proporcionan.

Durante el inicio, Zope lee todas estas declaraciones ZCML, valida que no hay dos declaraciones que tratan de registrar los mismos componentes y sólo entonces registra todo. Todos los componentes están registrados por interfaces necesarias y proporcionadas. Los componentes con las mismas interfaces también pueden opcionalmente ser nombrados.

Esta es una buena cosa. ZCML es por cierto sólo *una* manera de declarar su configuración.

Grok proporciona otra manera, donde un poco de magia Python permite usar decoradores para registrar clases Python y funciones como componentes. Usted puede usar tanto ZCML y Grok juntos si usted lo desea.

A algunos les gusta Grok porque le permite hacer casi todo en sus archivos fuentes Python. No requiere cableado adicional XML. Si usted es alérgico a XML , Grok es su boleto al nirvana Python.

Tenga en cuenta que no todo el mundo le encanta Grok. Algunas partes de la comunidad Plone piensan que sólo puede haber un lenguaje de configuración, otros están en contra de añadir la gran dependencia relativa de Grok para Plone. Un problema real es el hecho de que no se puede personalizar componentes declarados con Grok con técnicas jbot (del que hablaremos más adelante). Grok no está permitido en el núcleo de Plone por estas razones.

La elección de Grok o no Grok es suya. En cualquier caso, si usted comienza a escribir una extensión que es reutilizable, convertir sus declaraciones Grok a ZCML para conseguir la máxima aceptación.

Personalmente, me resulta engorroso pero incluso para mí como un desarrollador que ofrece una buena ventaja: gracias a ZCML, casi nunca tengo dificultades para averiguar qué y dónde extensiones o personalizaciones. Para mí, los archivos ZCML son como una guía telefónica.

Extiende Plone con paquetes Complementos

- Hay más de 2000 complementos para Plone. Nosotros vamos a cubrir sólo un puñado de hoy.
- Usándolos puedes ahorrarte mucho tiempo
- El éxito de un proyecto a veces depende en encontrar el complemento adecuado
- Su uso, utilidad, calidad y complejidad varia mucho

¿Como buscar complementos?

- https://pypi.python.org/pypi - ¡Use el formulario de búsqueda!
- https://github.com/collective > 1200 repositorios
- https://github.com/plone > 260 repositorios
- http://news.gmane.org/gmane.comp.web.zope.plone.user
- Navegando en Google (por ejemplo Plone+Slider)
- Verifica una lista corta en Plone Paragon (Lanzado en Agosto de 2014)
- Pregunta en el canal irc y la lista de correo

Ver también:

- Una conversación sobre encontrar y administrar complementos: https://www.youtube.com/watch?v=Sc6NkqaSjqw

Algunos complementos notables

Products.PloneFormGen Un generador de formularios

collective.plonetruegallery Galerías de fotos con una gran selección de varios librerías Javascript

collective.cover Interfaz para crear páginas de inicios (landing pages) complejas

collective.geo Paquete flexible de complementos para georeferenciar contenido y mostrar mapas

collective.mailchimp Permite a los visitantes suscribirse a noticias Mailchimp

eea.facetednavigation Crear la navegación facetada y búsquedas a través de la Web.

webcouturier.dropdownmenu Convierte la navegación global en menú desplegables

collective.quickupload Subida de múltiples archivos usando el concepto drag & drop

Products.Doormat Un flexible pie de página basado en el patrón de diseño Doormat

collective.behavior.banner Agrega banners decorativos y pasarelas

plone.app.multilingual Permite sitios plurilingüe mediante la traducción de contenido

Plomino Un poderoso y flexible constructor de aplicaciones basado en Web para Plone

Instalando complementos

La instalación es un proceso de dos pasos.

Haciendo los paquetes Complementos disponibles para Zope

Primero, debemos asegurarnos que los paquetes Complementos estén disponible para Zope. Esto significa, que Zope puede importar el código. Buildout es responsable de esto.

Busca el archivo `buildout.cfg` en `/vagrant/buildout`.

Nota: Si estás utilizando nuestro kit Vagrant, la configuración Plone está disponible en una carpeta que se comparte entre los sistemas operativos anfitrión y huésped. Busque en su directorio de instalación Vagrant la carpeta `buildout`. Puede editar los archivos de configuración utilizando su editor de texto favorito en el sistema operativo huésped, luego cambiar en la máquina virtual para ejecutar buildout en el sistema operativo invitado.

En la sección `[instance]` existe una variable llamada `eggs`, la cual tiene una lista de *eggs* como valor. Agregue los siguientes paquetes eggs:

Ya hemos agregado los complementos que usaras ahora:

- `Products.PloneFormGen`
- `collective.plonetruegallery`

Generalmente, uno ingresa los paquetes eggs agregando una linea adicional por cada paquete egg en la configuración, Debes escribir el nombre del paquete egg de forma indentada, así el buildout entiende que la linea actual es parte de la ultima variable y no una nueva variable.

Si agregas nuevos complementos tendrás que ejecutar buildout y reiniciar el sitio:

```
$ cd /vagrant/buildout
$ bin/buildout
$ bin/instance fg
```

Ahora el código está disponible desde a dentro de Plone.

Instalando complementos en su sitio de Plone

Tu sitio de Plone no ha sido especificado para usar el complemento. Para esto debes activar los complementos en tu sitio de Plone.

Nota: ¿Por qué el paso adicional de activar el paquete complemento? Usted mi tiene varios sitios Plone en una única instalación de Zope. Es normal que desee activar algunos complementos en un sitio, otros complementos en otro sitio.

En el navegador, vaya a Configuración del sitio (atajo: añadir en su dirección URL `/@@overview-controlpanel` del sitio Plone), y abrir el panel `Complementos`. Usted verá que puede instalar los complementos allí.

Instala **PloneFormGen** y **Plone True Gallery** ahora.

Esto es lo que sucede: el perfil GenericSetup del producto se ha cargado. esto hace cosas como:

- Configurando nuevas acciones
- Registrando nuevos tipos de contenido
- Registrando archivos CSS y Javascripts
- Creando algunos objetos contenidos / configuración en tu sitio de Plone.

Miremos lo que hemos instalado.

PloneFormGen

Hay muchas formas de crear formularios en Plone:

- Puro: HTML y Python en una vista
- Usando framework: z3c.form, formlib, deform
- A través de la Web: Products.PloneFormGen

El concepto básico de PloneFormGen es usted construye un formulario mediante la adición de una carpeta de formulario, a esta agrega campos de formulario como elementos de contenido. Se añaden, eliminar, editar campos y mueven al igual que con cualquier otro tipo de contenido. Los envíos de formularios pueden ser enviados por correo electrónico de forma automática y / o guardarse para descargar, por ejemplo en un archivo CSV. Hay muchas complementos para PloneFormGen que proporcionan los tipos de campo adicionales y acciones.

Vamos a construir un formulario de registro:

- Activar PloneFormGen para este sitio a través del panel de configuración Complementos en la Configuración del sitio.
- Agrega un objeto del nuevo tipo 'Form Folder' en el raíz del sitio llamado "Registro"
- Guarde y ver el resultado, un sencillo formulario de contacto el cual puede personalizar
- Haga clic en QuickEdit (para la edición rápida)
- Remueva el campo "Subject"
- Añadir campos para preferencia de alimentos y el tamaño de la camisa
- Agregar un adaptador DataSave
- Personalizar el script de envío de correo

Nota: ¿Necesita CAPTCHA? Agregue el paquete `collective.recaptcha` a su buildout y PloneFormGen - PFG tendrá un campo de CAPTCHA.

¿Necesita cifrado? Añadir cifrado GPG para su sistema, añada una configuración GPG para el usuario demonio de Plone que incluye unas claves públicas para los objetivos de correo, y usted será capaz de cifrar el correo electrónico antes de enviar.

¿Piensa que PloneFormGen es demasiado complicado para sus editores del sitio? Los administradores (y estamos iniciado sesión de usuario como un administrador) ven un montón de opciones más complejas que son invisibles para editores del sitio.

Por cierto, mientras PloneFormGen es bueno en lo que hace, no es un buen modelo para el diseño de sus propias extensiones. Se fue creado antes de la arquitectura de componentes de Zope convirtió ampliamente utilizado. Los autores desean escribir de manera muy diferente si estaban empezando desde cero.

Agregar Galerías de fotos con collective.plonetruegallery

Para promocionar la conferencia queremos mostrar algunas fotos mostrando conferencias pasadas y la ciudad donde la conferencia se esta realizando.

En vez de crear tipos de contenido personalizados para galerías, este se integra con la funcionalidad de Plone para elegir diferentes vistas de tipos de contenidos en carpeta.

https://pypi.python.org/pypi/collective.plonetruegallery

- Activar el producto
- Habilitar el comportamiento `Plone True Gallery` en el tipo `Carpeta`: http://localhost:8080/Plone/dexterity-types/Folder/@@behaviors (sólo este paso es necesario porque PloneTrueGallery todavía no sabe sobre los nuevos tipos de contenidos en el paquete plone.app.contenttypes, entonces hay activamos para reemplazar viejos tipos de contenido de Plone con más nuevos, de estilo Dexterity.)
- Agregue una carpeta /el-evento/ubicacion
- Suba algunas fotos desde http://lorempixel.com/600/400/city/
- Habilite la vista `galleryview`

collective.plonetruegallery es un mejor modelo para escribir una extensión de Plone.

Internacionalización

Plone puede ejecutar el mismo sitio en múltiples lenguajes.

No estamos haciendo esto con el sitio de la conferencia ya que la *lengua franca* de la comunidad Plone es el Ingles.

Para esto debe usar https://pypi.python.org/pypi/plone.app.multilingual. Ese es el sucesor del Products.LinguaPlone (el cual solo usa tipos de contenidos Archetypes).

Nota: La construcción de un sitio multilingüe exige activar el producto `plone.app.multilingual`, pero no es necesario un complemento para construir un sitio en un solo idioma que no sea Inglés. Sólo tienes que seleccionar un lenguaje sitio diferente al crear un sitio Plone, y todos los mensajes básicos serán traducidos y LTR o RTL necesidades manejados.

Resumen

Ahora somos capaces de personalizar y ampliar muchas partes de nuestro sitio web. Incluso podemos instalar extensiones que añaden nuevas funcionalidades.

Pero:

- ¿Podemos enviar Charlas al jurado evaluador ahora?
- ¿Podemos crear listas con las propiedades mas importantes de cada tarea?

- ¿Podemos permitir a un miembro del jurado votar en las Charlas?

Algunas veces trabajamos con data estructurada. Hasta un grado podemos hacer todo a través de la Web, pero a algún punto nos encontramos con barreras, en la siguiente parte de nuestro entrenamiento, te enseñaremos como romper esas barreras.

CAPÍTULO 11

Temas

Nosotros no hacemos ningún tipo de temas y plantillas real durante el entrenamiento. Sólo vamos a explicar las opciones que dispone para esto.

Aplicar Temas a un sitio Plone tiene dos partes importantes:

- **Tematización estructural** : la construcción del esqueleto HTML de una página y obtener los elementos correctos de contenido dentro de los lugares correctos. Además, la disposición de la CSS y los elementos de Javascript para finalizar la presentación y proporcionar comportamientos dinámicos.
- **Plantillas**, que a su vez tiene dos aspectos:
  - Plantillas Viewlet, que podríamos pensar como el micro formato de la página. ¿Recuerda cuando miramos el mapa viewlet de una página? (Mira de nuevo a través de la dirección URL `@@manage-viewlets`.) Todas esas viewlets se proporcionan a través de plantillas individuales y editables.
  - Plantillas de vista de tipo de contenido. Cuando creamos un nuevo tipo de contenido o modificar una existente, normalmente queremos crear o modificar una vista de plantilla.

Tematización estructural se logra mejor a través del motor de temas Diazo. la herramienta Diazo proporciona un mecanismo basado en mapeo del contenido proporcionada por Plone en una o más diseños de páginas maestras.

Plantillas se logra mediante la edición de los archivos página de plantilla que nos permiten mezclar el contenido del objeto de la ZODB con HTML. Plone utiliza su propio (en realidad Zope) el Template Attribute Language (TAL) para este fin.

Ejemplo Diazo

- Activar Diazo vía el formulario del panel de control Complementos
- Valla al panel de control `Temas`
- Activar el tema Twitter Bootstrap
- Mire los cambios del sitio
- Remplace “localhost” en su dirección URL con “127.0.0.1”
- Vuelva al panel de control Temas, eche una mirada al panel Configuración Avanzada
- Desactivar el tema
- Copie el tema Twitter Bootstrap Example, use el botón `Copiar` para crear un tema en base a esta copia y poder modificar el tema para ver las reglas Diazo.

Ejemplo de Plantilla

- Use la ZMI para ver la herramienta portal_view_customizations
- Echa un vistazo a `plone.belowcontenttitle.documentbyline` — tener una idea de cómo se utiliza la lógica TAL para tirar en el contenido de contexto.
- Cambiar "History" a "Herstory" :)

Seleccionando la herramienta adecuada

Si todo lo que tienes es un martillo, todo parece un clavo

Hacer un buen trabajo con la tematización de Plone significa elegir la herramienta adecuada para el trabajo.

Si eres muy bueno con Diazo, que puede hacer casi todo con reglas Diazo. Es enteramente posible sustituir y reordenar los componentes más pequeños de un viewlet con una aplicación inteligente de una regla Diazo o un poco de XSL.

También es perfectamente posible hacer todo su tematización mediante la personalización de los archivos de plantilla. Después de todo, su sitio original Plone es tematizado (con un tema llamado Sunburst), incluso sin necesidad de activar Diazo. Antes Diazo ser unido a Plone (como un complemento en Plone 4.0), este es la forma en que Plone era tematizado.

Así que, ¿cuál es su estrategia?

- Para los sencillos temas del sitio son estructuralmente similares a los de fuera de la caja es Plone, sólo tiene que añadir CSS. Nada más se necesita.
- Para más temas complejos o uno en el se proporcionan con un tema de HTML, CSS y JS, use Diazo para mover las cosas, para poner las piezas del rompecabezas donde pertenecen.
- Si es necesario cambiar un viewlet o la vista de un tipo de contenido, utilice plantillas TAL.

Pero, **no es necesario** molestarse en aprender a cómo trabajar con los administradores viewlet de Plone. Sí, una vez fue necesario, pero Diazo es una mejor solución a este problema.

¿Quieres aprender realmente tematización?

Los buenos puntos de partida:

- Diazo (plone.app.theming): Este es la forma moderna para hacerlo y el tema por defecto en Plone 5 será un tema Diazo, esta disponible desde Plone 4.2 y versiones posteriores.
- Usted puede utilizar (y adaptar) una de las muchas temas Diazo disponible públicamente: https://pypi.python.org/pypi?%3Aaction=search&term=plonetheme&submit=search (intente con plonetheme.onegov por ejemplo)
- Creando un tema personalizado
- Un punto de partida puede ser el editor incorporados de temas Diazo
- Los temas a la vieja escuela (ampliando el tema por defecto integrado)
- Deliverance/XDV

Si usted busca un entrenamiento sobre Diazo se recomienda un entrenamiento por Chrissy Wainwright o Maik DerStappen

Ver también:

Diazo: How it Works por Steve McMahon desde la Plone Conference 2013 https://www.youtube.com/watch?v=Vvr26Q5IriE

Ver también:

http://docs.plone.org/4/en/adapt-and-extend/theming/index.html

CAPÍTULO 12

Dexterity - Parte I: A través de la Web

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/13_dexterity/ src/ploneconf.site
```

¿Qué es un tipo de contenido?

Un tipo de contenido es una variedad de objetos que pueden almacenar información y es editable por los usuarios. Tenemos diferentes tipos de contenido para reflejar los diferentes tipos de información sobre la que tenemos que recopilar y mostrar la información. Páginas, carpetas, eventos, noticias, archivos (binarios) y las imágenes son todos los tipos de contenido.

Es común en el desarrollo de un sitio Web usted necesitará versiones personalizadas de los tipos de contenido comunes, o tal vez incluso totalmente nuevos tipos.

¿Recuerda los requisitos para nuestro proyecto? Queríamos ser capaces de solicitar y editar charlas de la conferencia. Podríamos utilizar el tipo de contenido `Página` para ese propósito. Sin embargo, hay bits de información que necesitamos para asegurarnos de que obtenemos sobre una charla y no sería seguro de obtener esa información si sólo pedimos ponentes potenciales para crear una página. Además, vamos a querer ser capaz de mostrar las charlas que ofrecen esa información especial, y nosotros queremos ser capaces de mostrar las colecciones de charlas. Un tipo de contenido personalizado será ideal.

Los elementos de un tipo de contenido Plone

Cada tipo de contenido Plone tiene las siguientes partes:

Esquema Una definición de campos que comprenden un tipo de contenido; propiedades de un objeto.

FTI La "Factory Type Information" configura el tipo de contenido en Plone, asigna un nombre, un icono, características adicionales y posibles vistas.

Vistas Una vista es una representación del objeto y el contenido de sus campos que pueden prestarse en respuesta a una solicitud. Usted puede tener una o más vistas de un objeto. Algunos pueden ser visuales - destinados a la pantalla en forma de páginas Web - otros pueden estar destinados a satisfacer las peticiones AJAX y estar en formatos como JSON o XML.

Dexterity y Archetypes - Una comparación

Hay dos frameworks de contenido en Plone

- Dexterity: nuevo y el que vendrá por defecto
- Archetypes: los tipos de contenidos viejos, intentados y probados
- Archetypes: extenso sin embargo con complementos
- Plone 4.x: Archetypes es el predeterminado, con Dexterity disponible
- Plone 5.x: Dexterity es el predeterminado, con Archetypes disponible
- Por tanto, añadir y editar formularios se crean automáticamente a partir de un esquema

¿Cuales son las diferencias?

- Dexterity: Nuevo, más rápido, modular, hay magia oscura para los métodos getters y setters
- Archetypes tiene métodos setter / getter mágico - usa talk.getAudience() para el campo 'audience'
- Dexterity: los campos son atributos: talk.audience en vez de talk.getAudience()

A través de la Web:

- Dexterity tiene una buena experiencia a través de la Web.
- Archetypes no tiene experiencia a través de la Web, por defecto, solo vía productos adicionales de tercero sin soporte oficial.
- Modelado UML: ArchGenXML para Archetypes, agx para Dexterity

Enfoques para Desarrolladores:

- Esquema en Dexterity: A través de la Web, XML, Python. Interface = esquema, a menudo no hay clase necesaria
- Esquema en Archetypes: Esquemas solamente en Python
- Dexterity: Permisos fácil por cada campo. fácil personalización de formularios.
- Archetypes: Permisos por campo duros, los formularios personalizados aún más difícil.
- Si tiene que programar para los sitios antiguos ¡usted necesita saber Archetypes!
- Si empieza desde cero en nuevo sitios, ¡use Dexterity!.

Extendiendo:

- Dexterity tiene Comportamientos: fácilmente ampliable. Basta con activar un comportamiento a través de la Web y su tipo de contenido es, por ejemplo traducible (con el complemento plone.app.multilingual).
- Archetypes tiene el producto archetypes.schemaextender. Potente pero no tan flexible.

Sólo hemos utilizado Dexterity en los últimos años. Enseñamos Dexterity y no Archetypes porque es más accesible a los principiantes, tiene una gran experiencia a través de la Web y es el futuro.

Vistas:

- Tanto Dexterity y Archetypes tienen una vista predeterminada de tipos de contenido.
- Las Browser Views proveen vistas personalizadas.
- A través de la Web (en el futuro)
- Formularios de visualización

Instalación

Nota: Podemos omitir este paso desde que instalamos el producto `plone.app.contenttypes` activando este al momento de crear nuestro sitio Plone en el principio.

Usted no tiene que modificar el archivo buildout puesto este esta incorporado desde Plone 4.2 y versiones posteriores con Dexterity. Sólo tienes que activarlo en el panel de control para Complementos.

Esta vez, sin razón aparente que no sea cada vez se sienta más cómodo con el ZMI, utilizaremos la herramienta `portal_quickinstaller` para instalar Dexterity.

- Ir a la herramienta portal_quickinstaller en la ZMI.
- Instale el producto "Tipos de contenido Dexterity".

Modificando tipos existentes

- Ir a panel de control http://localhost:8080/Plone/@@dexterity-types
- Inspecciona algunos de los tipos de contenido existentes por defecto
- Seleccione el tipo de contenidos `Noticias` y agregar un nuevo campo `Hot News` de tipo `Sí/No`
- En otra pestaña agregar un tipo de contenido Noticia y entonces se ve el nuevo campo.
- Valla al editor de esquema y haga clic en Editar el modelo de campo XML.
- Note que el único campo en el esquema XML de la Noticia, es el que acaba de agregar. Todos los demás son proporcionados por los comportamientos.
- Editar el tipo de widget del formulario por lo que dice:

```
<form:widget type="z3c.form.browser.checkbox.SingleCheckBoxFieldWidget"/>
```

- Edite de nuevo el elemento de Noticia. El widget de cambiado de un campo de radio a una casilla de verificación.
- El nuevo campo `Hot News` no se visualiza cuando se representa la Noticia. Nosotros nos ocuparemos de esto más adelante.

Ver también:

http://docs.plone.org/4/en/external/plone.app.contenttypes/docs/README.html#extending-the-types

Creando tipos de contenidos a través de la Web

En este paso vamos a crear un tipo de contenido llamado *Talk* (para las charlas) y probarlo. Cuando esté listo vamos a mover el código fuente de la Web para el sistema de archivos y en nuestro propio complemento. Más tarde vamos a ampliar ese tipo, los comportamientos y añadir un viewlet de las Charlas.

- Agregar un nuevo tipo de contenido llamado "Talk" y algunos campos para el:
  - Agregue el campo "Type of talk", de tipo "Selección". Agregar opciones talk, keynote, training
  - Agregue el campo "Details", de tipo "Rich Text" con un tamaño máximo de 2000
  - Agregue el campo "Audience", de tipo "Selección múltiple". Agregar opciones: beginner, advanced, pro

        - Marque los comportamientos que están habilitados: Metadatos Dublin Core, Nombre a partir del título. ¿Tenemos todos necesitamos?
    - Pruebe el tipo de contenido
    - Regrese al panel de control http://localhost:8080/Plone/@@dexterity-types
    - Extiende el nuevo tipo
        - "Speaker", tipo: "Línea de texto (cadena de caracteres)"
        - "Email", tipo: "Línea de texto (cadena de caracteres)"
        - "Image", tipo: "Image", no requerido
        - "Speaker Biography", tipo: "Rich Text"
    - Probar otra vez

Aquí está el código completo de esquema XML creado por nuestras acciones.

```
<model xmlns:security="http://namespaces.plone.org/supermodel/security" xmlns:marshal="http://namespa
  <schema>
    <field name="type_of_talk" type="zope.schema.Choice">
      <description/>
      <title>Type of talk</title>
      <values>
        <element>Talk</element>
        <element>Training</element>
        <element>Keynote</element>
      </values>
    </field>
    <field name="details" type="plone.app.textfield.RichText">
      <description>Add a short description of the talk (max. 2000 characters)</description>
      <max_length>2000</max_length>
      <title>Details</title>
    </field>
    <field name="audience" type="zope.schema.Set">
      <description/>
      <title>Audience</title>
      <value_type type="zope.schema.Choice">
        <values>
          <element>Beginner</element>
          <element>Advanced</element>
          <element>Professionals</element>
        </values>
      </value_type>
    </field>
    <field name="speaker" type="zope.schema.TextLine">
      <description>Name (or names) of the speaker</description>
      <title>Speaker</title>
    </field>
    <field name="email" type="zope.schema.TextLine">
      <description>Adress of the speaker</description>
      <title>Email</title>
    </field>
    <field name="image" type="plone.namedfile.field.NamedBlobImage">
      <description/>
      <required>False</required>
      <title>Image</title>
    </field>
    <field name="speaker_biography" type="plone.app.textfield.RichText">
```

```
      <description/>
      <max_length>1000</max_length>
      <required>False</required>
      <title>Speaker Biography</title>
    </field>
  </schema>
</model>
```

Moviendo el tipo de contenido dentro del código fuente

Es impresionante que podemos hacer tanto a través de la Web. Pero también es un callejón sin salida si queremos volver a utilizar este tipo de contenido en otros sitios.

También, para el desarrollo profesional, queremos ser capaces de utilizar el control de versiones para nuestro trabajo, y vamos a querer ser capaz de añadir la clase de lógica de negocio que requerirá de programación.

Por lo tanto, vamos a última instancia querer mover nuestro nuevo tipo de contenido en un paquete Python. Nos faltan algunas habilidades para hacer eso, y vamos a cubrir aquellos en los próximos dos capítulos.

Ver también:

Dexterity Developer Manual

Ejercicios

Ejercicio 1

Modificar Documentos para permitir subir una imagen como la decoración (como las Noticias).

Solución

- Valla al panel de control Tipos de contenido Dexterity (http://localhost:8080/Plone/@@dexterity-types)
- Haga clic en *Página* (http://127.0.0.1:8080/Plone/dexterity-types/Document)
- Seleccione la pestaña *Comportamientos* (http://127.0.0.1:8080/Plone/dexterity-types/Document/@@behaviors)
- Marque la casilla junto a *Imagen Lead* y guardar.

Las imágenes se muestran por encima del título.

Ejercicio 2

Crear un nuevo tipo de contenido llamada *Speaker* y exportar el esquema a un archivo XML. Debe contener los siguientes datos:

- First Name
- Last Name
- Email
- Homepage (opcional)
- Biography (opcional)

- Company (opcional)
- Twitter-Name (opcional)
- IRC-Name (opcional)
- Image (opcional)

Podríamos utilizar este tipo de contenido después de convertir los ponentes en los usuarios de Plone. Podríamos luego enlazar a sus charlas.

Solución

El esquema debería lucir así:

```
<model xmlns:security="http://namespaces.plone.org/supermodel/security"
       xmlns:marshal="http://namespaces.plone.org/supermodel/marshal"
       xmlns:form="http://namespaces.plone.org/supermodel/form"
       xmlns="http://namespaces.plone.org/supermodel/schema">
  <schema>
    <field name="first_name" type="zope.schema.TextLine">
      <title>First Name</title>
    </field>
    <field name="last_name" type="zope.schema.TextLine">
      <title>Last Name</title>
    </field>
    <field name="email" type="zope.schema.TextLine">
      <title>Email</title>
    </field>
    <field name="homepage" type="zope.schema.TextLine">
      <required>False</required>
      <title>Homepage</title>
    </field>
    <field name="biography" type="plone.app.textfield.RichText">
      <required>False</required>
      <title>Biography</title>
    </field>
    <field name="company" type="zope.schema.TextLine">
      <required>False</required>
      <title>Company</title>
    </field>
    <field name="twitter_name" type="zope.schema.TextLine">
      <required>False</required>
      <title>Twitter-Name</title>
    </field>
    <field name="irc_name" type="zope.schema.TextLine">
      <required>False</required>
      <title>IRC-Name</title>
    </field>
    <field name="image" type="plone.namedfile.field.NamedBlobImage">
      <required>False</required>
      <title>Image</title>
    </field>
  </schema>
</model>
```

CAPÍTULO 13

Buildout - Parte I

Buildout compone su aplicación por usted, de acuerdo a sus reglas.

Para componer su aplicación es necesario definir los paquetes eggs que usted necesita, ¿qué versión?, ¿qué archivos de configuración Buildout tiene que generar para usted?, para descargar y compilar, así sucesivamente. Buildout descarga los paquetes eggs requeridos y resuelve todas las dependencias. Es posible que necesite cinco paquetes eggs diferentes, pero al final, Buildout tiene que instalar 300 paquetes eggs, todos con la versión correcta.

Buildout hace esto sin tocar el sistema Python o afectar a ningún otro paquete. Los comando creados por buildout traer todos los paquetes requeridos en el entorno Python. Cada comando crea mis diferentes bibliotecas o incluso diferentes versiones de la misma biblioteca.

Plone necesita carpetas para los archivos de registro, bases de datos y archivos de configuración. Buildout ensambla todo esto para usted.

Usted necesitará una gran cantidad de funcionalidades que Buildout no proporciona fuera de la caja, así usted necesitará varias extensiones. Algunas extensiones proporcionan una funcionalidad totalmente nueva, como mr.developer, la mejor manera de manejar sus repositorios de código fuente.

Sintaxis

La sintaxis de los archivos de configuración de despliegue es similar a los archivos clásicos ini. Usted escribe un nombre de parámetro, un signo igual y el valor. Si introduce otro valor en la siguiente línea y indentada, Buildout entiende que ambos valores pertenecen al nombre del parámetro y el parámetro almacena todos los valores en forma de lista.

Un Buildout consta de múltiples secciones. Las secciones comienzan con el nombre de la sección entre corchetes. Cada sección declara una parte diferente de su aplicación. Como analogía aproximada, su archivo Buildout es un libro de cocina con múltiples recetas.

Hay una sección especial, llamada *[buildout]*. En esta sección se puede cambiar el comportamiento de la propia Buildout. Las variables `parts` define, cuál de las secciones existentes deben ser efectivamente utilizadas.

Recetas

Buildout por si mismo no tiene idea de como instalar Zope. Buildout es un sistema basado en plugin, este viene con un pequeño conjunto de plugins para crear archivos de configuración y descargar paquetes eggs con sus dependencias y su versión apropiada. Para instalar un sitio Zope, necesitas un plugin de terceros. El plugin provee nuevas recetas que tu tienes que declarar y configurar en una sección.

Un ejemplo es esta sección.

```
[instance]
recipe = plone.recipe.zope2instance
user = admin:admin
```

Esto usa el paquete de python plone.recipe.zope2instance para crear y configurar la instancia Zope 2 la cual usamos para ejecutar Plone. Todas las lineas después de `recipe = xyz` son la configuración de la receta usada.

Ver también:

http://www.buildout.org/en/latest/docs/recipelist.html

Referencias

Buildout le permite usar referencias en la configuración. Una declaración de variable no sólo puede mantener el valor de variable, pero también una referencia al lugar donde buscar el valor de la variable.

Si usted tiene una gran instalación con muchos sitios Plone con pequeños cambios entre cada configuración, se puede generar una plantilla de configuración, y cada sitio hace referencia a todo, desde la plantilla y sobrescribe justo en lo que necesita ser cambiado.

Incluso en los buildouts más pequeños esta es una característica útil. Estamos utilizando collective.recipe.omelette. Una receta muy práctica, la cual crea un directorio virtual para facilitar la navegación al código fuente de cada paquete egg.

La receta omelette tiene que saber cuales son los paquetes eggs para hacer referencia. Queremos los mismos paquetes eggs usados por nuestra instancia Zope, por lo que nos referimos a los paquetes eggs de la instancia en lugar de repetir toda la lista.

Otro ejemplo: Digamos usted crea archivos de configuración para un servidor Web como Nginx, puede definir el puerto de destino para el Proxy inverso al mirar hacia arriba a partir de la receta zope2instance.

La configuración de sistemas complejos siempre implica una gran cantidad de duplicación de la información. El uso de referencias en la configuración buildout le permite minimizar estas duplicaciones.

Un ejemplo de la vida real

Examinemos el archivo `buildout.cfg` para el entrenamiento y miremos algunas de las variables más importantes:

```
[buildout]
extends =
    http://dist.plone.org/release/4.3.10/versions.cfg

# We add our own versions
    versions.cfg

versions = versions

extensions = mr.developer
# Tell mr.developer to ask before updating a checkout.
always-checkout = true
show-picked-versions = true
sources = sources

# Put checkouts in src-mrd. We keep our own package in src
```

```
sources-dir = src-mrd

# The directory this buildout is in. Modified when using vagrant.
buildout_dir = ${buildout:directory}

# We want to checkouts these eggs directly from github
auto-checkout =
#    ploneconf.site_sneak
#    starzel.votable_behavior
#    ploneconf.site

parts =
    checkversions
    codeintel
    instance
    mrbob
    packages
#    robot
    test
    zopepy
#    zopeskel

eggs =
    Plone
    Pillow

# development tools
    z3c.jbot
    plone.api
    plone.reload
    Products.PDBDebugMode
    plone.app.debugtoolbar
    Products.PrintingMailHost

# 3rd party addons
    Products.PloneFormGen
    collective.plonetruegallery
    collective.js.datatables
    eea.facetednavigation
    collective.behavior.banner

# dexterity default types
    plone.app.contenttypes

# The addon we develop in the training
#    ploneconf.site

# Voting on content
#    starzel.votable_behavior

zcml =

test-eggs +=
#    ploneconf.site [test]

[instance]
recipe = plone.recipe.zope2instance
user = admin:admin
```

```
http-address = 8080
debug-mode = on
verbose-security = on
deprecation-warnings = on
eggs = ${buildout:eggs}
zcml = ${buildout:zcml}
file-storage = ${buildout:buildout_dir}/var/filestorage/Data.fs
blob-storage = ${buildout:buildout_dir}/var/blobstorage

[test]
recipe = zc.recipe.testrunner
eggs = ${buildout:test-eggs}
defaults = ['--exit-with-status', '--auto-color', '--auto-progress']

[packages]
recipe = collective.recipe.omelette
eggs = ${buildout:eggs}
location = ${buildout:buildout_dir}/packages

[codeintel]
recipe = corneti.recipes.codeintel
eggs = ${buildout:eggs}

[checkversions]
recipe = zc.recipe.egg
eggs = z3c.checkversions [buildout]

[zopepy]
recipe = zc.recipe.egg
eggs = ${buildout:eggs}
interpreter = zopepy
scripts = zopepy

[zopeskel]
recipe = zc.recipe.egg
eggs =
    ZopeSkel
    Paste
    PasteDeploy
    PasteScript
    zopeskel.diazotheme
    zopeskel.dexterity
    zest.releaser
    ${buildout:eggs}

[mrbob]
recipe = zc.recipe.egg
eggs =
    mr.bob
    bobtemplates.plone

[sources]
# ploneconf.site = fs ploneconf.site full-path=${buildout:directory}/src/ploneconf.site
starzel.votable_behavior = git https://github.com/collective/starzel.votable_behavior.git pushurl=git
```

Cuando usted ejecuta el comando `./bin/buildout` sin argumentos, Buildout buscara por este archivo.

Echemos un vistazo más de cerca a algunas variables.

```
extends =
    http://dist.plone.org/release/4.3.10/versions.cfg
    versions.cfg
```

Esta línea le dice al Buildout lea más archivos de configuración. Puede hacer referencia a los archivos de configuración locales en el equipo o los archivos de configuración en Internet, accesible a través de protocolo HTTP. Puede utilizar varios archivos de configuración para compartir entre múltiples configuraciones Buildouts, o para separar los diferentes aspectos de su configuración en archivos diferentes. Ejemplos típicos son especificaciones de la versión o la configuración que difieren entre los diferentes entornos de instalación como desarrollo, calidad, producción.

```
eggs =
    Plone
    Pillow
    z3c.jbot
    plone.api
    plone.reload
    Products.PDBDebugMode
    plone.app.debugtoolbar
    Paste
    Products.PloneFormGen
    collective.plonetruegallery
    collective.js.datatables
    eea.facetednavigation
    collective.behavior.banner
    plone.app.contenttypes
#    ploneconf.site
#    starzel.votable_behavior
```

Esta es la lista de los paquetes eggs que nosotros configuramos para estar disponible para Zope. Estos paquetes eggs se ponen en el PYTHONPATH del script `bin/instance` con el cual se iniciara y detendrá con Plone.

El paquete egg `Plone` es una envoltura sin código fuente. Entre sus dependencias esta `Products.CMFPlone` que es el paquete egg en el cual esta propiamente de Plone.

El resto son complementos que ya hemos usado o usará más tarde. Los últimos paquetes eggs están comentadas por lo que no se instalarán por Buildout.

El archivo `versions.cfg` se incluye por la declaración `extends = ...` contiene las versiones que definimos:

```
[versions]
# dev tools
z3c.jbot = 0.7.2
plone.api = 1.1.0
plone.app.debugtoolbar = 1.0a3
...
```

Esta es otra sección especial. Se ha convertido en una sección especial de declaración. En nuestra sección `[buildout]` establecemos una variable `versions = versions`. Esto le dice a buildout, existe una sección denominada versions, conteniendo información de las versiones a usar. Cuando Buildout instala paquetes eggs se utilizará la versión definida en esta sección.

¡Hola mr.developer!

Hay muchas cosas más importantes que debe saber, y no podemos pasar por ellos en todos los detalles, pero quiero centrarme en una característica específica: **mr.developer**

Con mr.developer puede declarar los paquetes que desee comprobar desde o hacia un sistema de control de versiones y de cual dirección URL del repositorio. Usted puede comprobar código fuente desde diversos CVS como git, subver-

sion, bzr, hg y quizá más. También, se puede indicar a Buildout algún código fuente se encuentran en el sistema de archivos local sin aplicar control de versiones.

`mr.developer` viene con un comando, `./bin/develop`. Se puede utilizar para actualizar su código fuente, para comprobar los nuevos cambios y así sucesivamente. Usted puede activar y desactivar sus checkouts de sus código fuente. Si usted desarrolla sus extensiones en los paquetes eggs con checkouts separadas, lo cual es una buena práctica, usted puede planear publicaciones por tener todas los checkouts desactivados, y sólo activarlos, al escribir los cambios que requieren una nueva versión. Usted puede activar y desactivar los paquetes eggs a través del comando `develop` o vía la configuración Buildout. Siempre debe usar la forma Buildout. Su commit sirve como documentación.

Extensible

Usted puede haber notado que la mayoría, si no toda la funcionalidad sólo está disponible a través de plugins. Una de las cosas que sobresale en Buildout sin ningún plugin, es la resolución de dependencias. Usted puede ayudar a Plone en la resolución de dependencias, al declarar exactamente cual versión de un paquete egg usted desea. Esto es sólo un caso de uso. Otra es mucho más importante: Si usted quiere tener un Buildout repetible, el cual funcione de dos meses a partir de ahora también, se *debe* declarar todas sus versiones de paquete egg. De otro modo Buildout podría instalar las nuevas versiones.

Sea un McGuyver

Como puede ver, usted puede construir sistemas muy complejos con Buildout. Es hora de algunas advertencias. Sea selectivo en sus recetas. Supervisor es un programa para gestionar servidores en ejecución, es bastante bueno. No hay una receta para ello.

¡La configuración para esta receta es más complicada que la propia configuración del Supervisor en si! Mediante el uso de esta receta, se fuerza a los demás a comprender la sintaxis de configuración específica recetas y la sintaxis Supervisor. Para tales casos, collective.recipe.template presenta una mejor coincidencia.

Otro problema es el manejo de errores. ¿Buildout intenta instalar una rara dependencia que en realidad no quiere? Buildout no le dirá, lo que está viniendo.

Si hay un problema, siempre se puede ejecutar Buildout con la opción `-v`, para obtener una salida más detallada, a veces ayuda.

```
$ ./bin/buildout -v
```

Si se solicitan las versiones de paquetes eggs extraños, verifique la declaración de las dependencias de los paquetes eggs y su versión definida.

Algunas partes de Buildout interpretan nombres de paquete egg entre mayúsculas y minúsculas, otros no. Esto puede dar lugar a problemas de divertidos.

Verifique siempre el orden de su configuraciones extendidas, utilice siempre el comando `annotate` de Buildout para ver si se interpreta la configuración diferente a la que usted definió. Restringirse a los archivos de despliegue simples. Puede hacer referencia a variables de otras secciones, incluso se puede utilizar toda una sección como plantilla. Hemos aprendido que esto no funciona bien con jerarquías complejas y se tuvo que abandonar esa característica.

En el capítulo *Buildout - Parte II: Cómo prepararse para el despliegue* veremos una configuración de despliegue lista para entornos de producción de Plone que tiene muchas características útiles.

Ver también:

Documentación de Buildout

- http://docs.plone.org/4/en/old-reference-manuals/buildout/index.html

- http://www.buildout.org/en/latest/docs/index.html

Solución de problemas http://docs.plone.org/4/en/manage/troubleshooting/buildout.html

Un buildout mínimo para Plone 4 https://github.com/collective/minimalplone4

El archivo Buildout del instalador unificado tiene documentación valiosa como comentarios entre linea

- https://github.com/plone/Installers-UnifiedInstaller/blob/master/buildout_templates/buildout.cfg
- https://github.com/plone/Installers-UnifiedInstaller/blob/master/base_skeleton/base.cfg
- https://github.com/plone/Installers-UnifiedInstaller/blob/master/base_skeleton/develop.cfg

mr.developer https://pypi.python.org/pypi/mr.developer/

CAPÍTULO 14

Creando complementos para personalizar Plone

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/12_eggs1/ src/ploneconf.site
```

En esta parte vamos a tratar:

- Cree una distribución python personalizada llamada `ploneconf.site` para contener todo el código
- Modificar la configuración buildout para instalar esa distribución

Tópicos cubiertos:

- mr.bob y bobtemplates.plone
- la estructura de los paquetes eggs

Creando la distribución

Nuestro propio código tiene que ser organizado como un paquete egg. Un paquete egg es un archivo zip o un directorio que sigue ciertas convenciones. Vamos a utilizar bobtemplates.plone para crear un proyecto de esqueleto. Tan sólo hay que responder a las preguntas del generador, las cuales crearán archivos necesarios para trabajar.

Nosotros accedemos al directorio `src` y ejecutamos un script llamado `mrbob` ubicado en nuestros directorio `bin` del buildout.

```
$ mkdir src       # (if src does not exist already)
$ cd src
$ ../bin/mrbob -O ploneconf.site bobtemplates:plone_addon
```

Tenemos que responder a algunas preguntas sobre el complemento. Vamos a presionar la tecla `Enter` (es decir, elegir el valor por defecto) para todas las preguntas, excepto la 3era pregunta (donde ingresa su nombre de usuario github si tiene uno) y la 5ta pregunta (versión Plone), donde ingresa `4.3.10`.

```
--> What kind of package would you like to create? Choose between 'Basic', 'Dexterity', and 'Theme'.

--> Author's name [Philip Bauer]:

--> Author's email [bauer@starzel.de]:
```

```
--> Author's github username: fulv

--> Package description [An add-on for Plone]:

--> Plone version [4.3.9]: 4.3.10

Generated file structure at /vagrant/buildout/src/ploneconf.site
```

Si este es su primer paquete egg, éste es un momento muy especial. Vamos a crear el paquete egg con un script que genera una gran cantidad de archivos necesarios. Todos ellos son necesarios, pero a veces de una manera sutil. Se toma un tiempo entiendo su significado pleno. Sólo el año pasado he aprendido y entendido por qué debería tener un archivo `MANIFEST.in`. Usted puede vivir sin uno, pero confía en mí, te llevas mejor con un archivo de manifiesto adecuada.

Inspeccionando la distribución

En la carpeta `src` ahora hay una nueva carpeta `ploneconf.site` y ahí está la nueva distribución. Echemos un vistazo a algunos de los archivos:

bootstrap-buildout.py, buildout.cfg, travis.cfg, .travis.yml, .coveragerc Puede pasar por alto estos archivos por ahora. Ellos están aquí para crear un buildout sólo para este paquete egg para hacer las pruebas más fáciles. Una vez que empezamos a escribir las pruebas para este paquete tendremos que actualizar estos archivos a las actuales mejores prácticas y versiones.

README.rst, CHANGES.rst, CONTRIBUTORS.rst, docs/ La documentación y el archivo de registro de cambios, el archivo de la lista de contribuidores y el archivo de la licencia de su paquete egg van allí.

setup.py Este archivo configura el paquete, su nombre, las dependencias y algunos metadatos como el nombre del y correo electrónico del autor. Las dependencias listadas aquí son descargadas automáticamente por buildout.

src/ploneconf/site/ La distribución Python en si misma vive dentro de una estructura de carpetas especial. Eso parece confuso, pero es necesario para tener buena habilidad de prueba. Nuestra distribución contiene un paquete de espacio de nombres llamado *ploneconf.site* y debido a esto hay una carpeta `ploneconf` con un archivo `__init__.py` y allí hay otra carpeta `site` y ahí finalmente está nuestro código. Desde la perspectiva de buildout, nuestro código está en la ruta `<su directorio buildout>/src/ploneconf.site/src/ploneconf/site/<real code>`

Nota: A menos que discutamos el buildout, de ahora en adelante, omitiremos silenciosamente estas carpetas al describir los archivos y asumiremos que `<su directorio buildout>/src/ploneconf.site/src/ploneconf/site/` ¡es el directorio raíz de nuestra distribución Python!

---

configure.zcml (src/ploneconf/site/configure.zcml) La guía de los paquetes. De su lectura se puede averiguar qué funcionalidad está registrado a través de la arquitectura de componentes.

setuphandlers.py (src/ploneconf/site/setuphandlers.py) Esto contiene el código que se ejecuta automáticamente al instalar y desinstalar nuestro complemento.

interfaces.py (src/ploneconf/site/interfaces.py) Aquí se define un browserlayer en una simple clase de python. Lo necesitaremos más tarde.

testing.py Esta contiene la configuración para correr las pruebas del paquete.

tests/ Esta contiene las pruebas del paquete.

browser/ Este directorio es un paquete python (porque tiene un `__init__.py`) y por convención contiene la mayoría de las cosas que son visibles en el navegador.

browser/configure.zcml La guía del paquete del browser. Aquí se registran vistas, recursos y sobre-escrituras de Plone.

browser/overrides/ Este complemento ya está configurado para permitir la sustitución o sobre-escritura de las plantillas por defecto de Plone existentes.

browser/static/ Un directorio que contiene los recursos estáticos (archivos de imágenes, CSS y JS). Los archivos aquí serán accesibles a través de las URLs como `++resource++ploneconf.site/myawesome.css`

profiles/default/ La carpeta contiene el perfil GenericSetup. Durante el entrenamiento pondrán algunos archivos XML ahí que tienen la configuración para el sitio.

profiles/default/metadata.xml Número de versión y dependencias que son auto-instalado cuando esta instalando su propio complemento.

Incluyendo el paquete egg en Plone

Antes de poder utilizar nuestro nuevo complemento tenemos que indicarle de la existencia de este a Plone. Entonces edite el archivo `buildout.cfg` y descomente el paquete egg `ploneconf.site` en las secciones *eggs* y *sources*:

```
eggs =
    Plone
    ...
    ploneconf.site
#    starzel.votable_behavior

...

[sources]
collective.behavior.banner = git https://github.com/collective/collective.behavior.banner.git pushurl
ploneconf.site = fs ploneconf.site full-path=${buildout:directory}/src/ploneconf.site
# starzel.votable_behavior = git https://github.com/collective/starzel.votable_behavior.git pushurl=
```

Esto le dice a Buildout agregue el paquete egg `ploneconf.site`. Dado que también se encuentra en la sección *sources* en Buildout no intentará descargarlo de PYPI pero esperará en `src/ploneconf.site`. La opción *fs* le permite añadir paquetes en el sistema de ficheros sin un sistema de control de versiones, o con uno compatible.

Ahora ejecute buildout para reconfigurar Plone con la configuración actualizada:

```
$ ./bin/buildout
```

Después reinicie Plone con el comando `./bin/instance fg` el nuevo complemento *ploneconf.site* está disponible para instalar como *PloneFormGen* o *Plone True Gallery*.

No vamos a instalarlo ahora, ya que no añadimos aun nada de nuestro propio código fuente o configuración todavía. Vamos a hacer eso.

Volver Dexterity: para mover tipos de contenidos a código fuente

¿Recuerda el tipo de contenido *Talk* que hemos creado a través de la Web con Dexterity? Vamos a pasar ese nuevo tipo de contenido a dentro de nuestro paquete egg para que pueda ser instalado en otros sitios sin manipulación a través de la Web.

Pasos:

- Volver al panel del control Tipos de contenido Dexterity

- Haga clic en la casilla junto al Tipo de contenido *Talk* y hace clic en el botón *Exportar los perfiles del tipo* y guardarlo en el archivo
- Elimine el Tipo de contenido *Talk* desde el panel del control Tipos de contenido Dexterity en su sitio Plone ante de instalarlo desde el sistema de archivo
- Extrae los archivos desde el archivo tar exportado y agréguelo entonces en nuestro paquete en la ruta `ploneconf/site/profiles/default/`

El archivo `ploneconf/site/profiles/default/types.xml` le dice a que allí tiene un nuevo tipo de contenido definido en el archivo `talk.xml`.

```
<?xml version="1.0"?>
<object name="portal_types" meta_type="Plone Types Tool">
 <property name="title">Controls the available content types in your portal</property>
 <object name="talk" meta_type="Dexterity FTI"/>
 <!-- -*- more types can be added here -*- -->
</object>
```

Tras la instalación, Plone lee el archivo `ploneconf/site/profiles/default/types/talk.xml` y registra un nuevo tipo de contenido en la herramienta `portal_types` (puede encontrar esta en el ZMI) con la información tomada de ese archivo.

```
  <?xml version="1.0"?>
  <object name="talk" meta_type="Dexterity FTI" i18n:domain="plone"
     xmlns:i18n="http://xml.zope.org/namespaces/i18n">
   <property name="title" i18n:translate="">Talk</property>
   <property name="description" i18n:translate="">None</property>
   <property name="icon_expr">string:${portal_url}/document_icon.png</property>
   <property name="factory">talk</property>
   <property name="add_view_expr">string:${folder_url}/++add++talk</property>
   <property name="link_target"></property>
   <property name="immediate_view">view</property>
   <property name="global_allow">True</property>
   <property name="filter_content_types">True</property>
   <property name="allowed_content_types"/>
   <property name="allow_discussion">False</property>
   <property name="default_view">view</property>
   <property name="view_methods">
    <element value="view"/>
   </property>
   <property name="default_view_fallback">False</property>
   <property name="add_permission">cmf.AddPortalContent</property>
   <property name="klass">plone.dexterity.content.Container</property>
   <property name="behaviors">
    <element value="plone.app.dexterity.behaviors.metadata.IDublinCore"/>
    <element value="plone.app.content.interfaces.INameFromTitle"/>
   </property>
   <property name="schema"></property>
   <property
      name="model_source">&lt;model xmlns:security="http://namespaces.plone.org/supermodel/security"
      &lt;schema&gt;
        &lt;field name="type_of_talk" type="zope.schema.Choice"&gt;
          &lt;description/&gt;
          &lt;title&gt;Type of talk&lt;/title&gt;
          &lt;values&gt;
            &lt;element&gt;Talk&lt;/element&gt;
            &lt;element&gt;Training&lt;/element&gt;
            &lt;element&gt;Keynote&lt;/element&gt;
          &lt;/values&gt;
```

```
        &lt;/field&gt;
        &lt;field name="details" type="plone.app.textfield.RichText"&gt;
          &lt;description&gt;Add a short description of the talk (max. 2000 characters)&lt;/descripti
          &lt;max_length&gt;2000&lt;/max_length&gt;
          &lt;title&gt;Details&lt;/title&gt;
        &lt;/field&gt;
        &lt;field name="audience" type="zope.schema.Set"&gt;
          &lt;description/&gt;
          &lt;title&gt;Audience&lt;/title&gt;
          &lt;value_type type="zope.schema.Choice"&gt;
            &lt;values&gt;
              &lt;element&gt;Beginner&lt;/element&gt;
              &lt;element&gt;Advanced&lt;/element&gt;
              &lt;element&gt;Professionals&lt;/element&gt;
            &lt;/values&gt;
          &lt;/value_type&gt;
        &lt;/field&gt;
        &lt;field name="speaker" type="zope.schema.TextLine"&gt;
          &lt;description&gt;Name (or names) of the speaker&lt;/description&gt;
          &lt;title&gt;Speaker&lt;/title&gt;
        &lt;/field&gt;
        &lt;field name="email" type="zope.schema.TextLine"&gt;
          &lt;description&gt;Adress of the speaker&lt;/description&gt;
          &lt;title&gt;Email&lt;/title&gt;
        &lt;/field&gt;
        &lt;field name="image" type="plone.namedfile.field.NamedBlobImage"&gt;
          &lt;description/&gt;
          &lt;required&gt;False&lt;/required&gt;
          &lt;title&gt;Image&lt;/title&gt;
        &lt;/field&gt;
        &lt;field name="speaker_biography" type="plone.app.textfield.RichText"&gt;
          &lt;description/&gt;
          &lt;max_length&gt;1000&lt;/max_length&gt;
          &lt;required&gt;False&lt;/required&gt;
          &lt;title&gt;Speaker Biography&lt;/title&gt;
        &lt;/field&gt;
      &lt;/schema&gt;
    &lt;/model&gt;</property>
    <property name="model_file"></property>
    <property name="schema_policy">dexterity</property>
    <alias from="(Default)" to="(dynamic view)" />
    <alias from="edit" to="@@edit" />
    <alias from="sharing" to="@@sharing" />
    <alias from="view" to="(selected layout)" />
    <action title="View" action_id="view" category="object" condition_expr=""
       description="" icon_expr="" link_target="" url_expr="string:${object_url}"
       visible="True">
      <permission value="View" />
    </action>
    <action title="Edit" action_id="edit" category="object" condition_expr=""
       description="" icon_expr="" link_target=""
       url_expr="string:${object_url}/edit" visible="True">
      <permission value="Modify portal content" />
    </action>
  </object>
```

Ahora nuestro paquete tiene algunos contenidos reales. Por lo tanto, tendremos que volver a instalarlo (si está instalado antes).

- Reinicie Plone.
- Reinstale el paquete ploneconf.site (desactive y active).
- Valla a la ZMI y vea la definición del nuevo tipo de contenido en la herramienta `portal_types`.
- Pruebe el tipo de contenido agregando un objeto o la edición de una de las entradas antiguas.
- Mire cómo se presentan los tipo de contenidos talks en el navegador.

CAPÍTULO 15

Vistas - Parte I

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/14_views_1/ src/ploneconf.site
```

Una simple browser view

Antes de escribir la vista de tipo de contenido talk en sí, retrocedemos y hablamos *un poco* sobre las vistas y las plantillas.

Una vista en Plone suele ser un `BrowserView`. Puede contener mucho código de python, pero primero nos centraremos en la plantilla.

Edite el archivo `browser/configure.zcml` y registre una nueva vista llamada *training*:

```
<configure
  xmlns="http://namespaces.zope.org/zope"
  xmlns:browser="http://namespaces.zope.org/browser"
  xmlns:plone="http://namespaces.plone.org/plone"
  i18n_domain="ploneconf.site">

  <!-- Set overrides folder for Just-a-Bunch-Of-Templates product -->
  <include package="z3c.jbot" file="meta.zcml" />
  <browser:jbot
    directory="overrides"
    layer="ploneconf.site.interfaces.IPloneconfSiteLayer"
    />

  <!-- Publish static files -->
  <browser:resourceDirectory
    name="ploneconf.site"
    directory="static"
    />

  <browser:page
    name="training"
    for="*"
    template="templates/training.pt"
```

```
24      permission="zope2.View"
25      />
26
27  </configure>
```

Agregue un archivo `browser/templates/training.pt`:

```
<h1>Hello World</h1>
```

- Reiniciar Plone y abra en la dirección http://localhost:8080/Plone/@@training.
- Usted debería ver "Hello World".

Ahora tenemos todo en su lugar para aprender sobre las Zope Page Templates - ZPT.

Nota: La vista `training` no tiene ninguna clase python registrada para ella pero sólo una plantilla. Actúa como si tuviera una clase vacía python heredando de `Products.Five.browser.BrowserView`, pero la forma en que sucede es realmente es un poco de magia...

CAPÍTULO 16

Zope Page Templates

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/16_zpt_2/ src/ploneconf.site
```

Plantillas de páginas son archivos HTML con alguna información adicional, escritas en TAL, METAL y TALES.

Las plantillas de página debe ser XML válido.

Los tres lenguajes son:

- TAL: “Template Attribute Language”
  - Plantillas XML / HTML usando con atributos especiales
  - Usando TAL nosotros podemos incluir expresiones en HTML
- TALES: “TAL Expression Syntax”
  - Define la sintaxis y la semántica de estas expresiones
- METAL: “Macro Expansion for TAL”
  - esto nos permite combinar, la reutilización y las plantillas anidadas juntas

TAL y METAL se escriben como atributos HTML (url, src, title). TALES están escritos como los valores de los atributos de HTML. Una declaración TAL típica tiene este aspecto:

```
<title tal:content="context/title">
    The Title of the content
</title>
```

Se utiliza para modificar la salida:

```
<p tal:content="string:I love red">I love blue</p>
```

resultado en:

```
<p>I love red</p>
```

Vamos a intentarlo. Abra el archivo `demoview.pt` y agregue:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      lang="en"
      i18n:domain="ploneconf.site">
```

```
<body>

    <p>red</p>

</body>
</html>
```

TAL y TALES

Vamos a añadir un poco de magia y modificar la etiqueta <p>:

```
<p tal:content="string:blue">red</p>
```

Esto resultará en el siguiente código HTML:

```
<p>blue</p>
```

Sin reiniciar Plone abra en el navegador http://localhost:8080/Plone/@@demoview.

Lo mismo ocurre con los atributos. Remplace en la línea <p> con el siguiente código:

```
<a href="http://www.mssharepointconference.com"
   tal:define="a_fine_url string:http://www.ploneconf.org"
   tal:attributes="href a_fine_url"
   tal:content="string:A even better conference">
    A sharepoint conference
</a>
```

resultado en:

```
<a href="http://www.ploneconf.org">
    A even better conference
</a>
```

Nosotros hemos utilizado los tres atributos TAL aquí. Esta es la lista completa de los atributos TAL:

tal:define definir las variables. Nosotros definimos la variable url para la cadena "http://www.ploneconf.org"

tal:content reemplazar el contenido de un elemento. Hemos sustituido el valor por defecto de contenido con algunas como "A even better conference"

tal:attributes cambiar dinámicamente atributos de los elementos. Nosotros hemos establecido el atributo href html a la variable `a_fine_url`

tal:condition pruebas, si la expresión es verdadera o falsa.

tal:repeat repite un elemento iterable, en nuestro caso, la lista de charlas.

tal:replace reemplazar el contenido de un elemento como `tal:content` , pero se elimina el elemento sólo dejando el contenido.

tal:omit-tag eliminar un elemento, dejando el contenido del elemento.

tal:on-error controlar los errores.

Expresiones Python

Hasta ahora sólo utilizamos una expresión TALES (el bit `string:`). Usemos una expresión TALES diferente ahora. Con `python:` podemos usar código python. Un ejemplo sencillo:

```
<p tal:define="title context/title"
   tal:content="python:title.upper()">
   A big title
</p>
```

Y otro ejemplo:

```
<p tal:define="talks python:['Dexterity for the win!',
                             'Deco is the future',
                             'A keynote on some weird topic',
                             'The talk that I did not submit']"
   tal:content="python:talks[0]">
    A talk
</p>
```

Con expresiones python

- sólo se puede escribir declaraciones individuales
- usted podría importar cosas pero no debe (ejemplo: `tal:define="something modules/Products.PythonScripts/something;`).

tal:condition

`tal:condition` pruebas, si la expresión es verdadera o falsa.

- Si es true, entonces la etiqueta se representa.
- Si es false, entonces la etiqueta **y todos sus hijos** se retiran y ya no se evalúa.
- Podemos invertir la lógica anteponiendo un `not:` a la expresión.

Vamos a añadir otro atributo TAL para nuestro ejemplo anterior:

```
tal:condition="talks"
```

También pudimos probar que el número de charlas:

```
tal:condition="python:len(talks) >= 1"
```

o si una cierta charla está en la lista de charlas:

```
tal:condition="python:'Deco is the future' in talks"
```

tal:repeat

Probemos otra sentencia:

```
<p tal:define="talks python:['Dexterity for the win!',
                             'Deco is the future',
                             'A keynote on some weird topic',
                             'The talk that I did not submit']"
   tal:repeat="talk talks"
   tal:content="talk">
   A talk
</p>
```

`tal:repeat` repite un elemento iterable, en nuestro caso, la lista de charlas.

Cambiamos el formato un poco para construir una lista en la que hay una lista <li> para cada charla:

```
<ul tal:define="talks python:['Dexterity for the win!',
                              'Deco is the future',
                              'A keynote on some weird topic',
                              'The talk that I did not submit']">
    <li tal:repeat="talk talks"
        tal:content="talk">
          A talk
    </li>
    <li tal:condition="not:talks">
          Sorry, no talks yet.
    </li>
</ul>
```

Expresiones de rutas

En cuanto TALES hasta ahora hemos utilizado la `string:` o `python:` o sólo las variables. La expresión siguiente y más comunes son de path-expressions. Opcionalmente se puede iniciar path-expression con el `path:`

Cada expresión de ruta comienza con un nombre de variable. Puede ser tanto un objeto como `context`, `view`, `template` o una variable definida anteriormente como `charla`.

Después de la variable se añade una barra `/` y el nombre de un sub-objeto, atributo o método invocable. El carácter '/' se utiliza para terminar el nombre de un objeto y el inicio del nombre de la propiedad. Mismas propiedades pueden ser objetos que a su vez tienen propiedades.

```
<p tal:content="context/title"></p>
```

Nos puede encadenar varios de los elementos para obtener la información que queremos.

```
<p tal:content="context/REQUEST/form"></p>
```

Esto devolverá el valor de la forma de diccionario del objeto HTTPRequest. Útil para la forma de manipulación.

El carácter `|` (operador lógico "o") se utiliza para encontrar un valor alternativo a un camino si la primera ruta se evalúe como `nothing` o no existe.

```
<p tal:content="context/title | context/id"></p>
```

Esto devuelve el id del contexto si no tiene título.

```
<p tal:replace="talk/average_rating | nothing"></p>
```

Esto devuelve nada si no hay 'average_rating' para una charla. Lo que no funciona es `tal:content="python:talk['average_rating'] or ''"`. ¿Quién sabe lo que esto produciría?

Conseguiremos `KeyError: 'average_rating'`. Es muy mala práctica, use el carácter `|` con demasiada frecuencia, ya que se tragará errores como un error tipográfico en `tal:content="talk/averange_ratting | nothing"` y es posible que se preguntan por qué no existen calificaciones más tarde...

No puede y no debe utilizarlo para evitar errores como un bloque try / except.

Hay varias **variables incorporado** que se pueden utilizar en caminos:

El más utilizado es `nothing` que es el equivalente a `None`

```
<p tal:replace="nothing">
    this comment will not be rendered
</p>
```

Un diccionario de todas las variables disponibles es `CONTEXTS`

```
<dl tal:define="path_variables_dict CONTEXTS">
  <tal:vars tal:repeat="variable path_variables_dict">
    <dt tal:content="variable"></dt>
    <dd tal:content="python:path_variables_dict[variable]"></dd>
  </tal:vars>
</dl>
```

Muy útil para depuración :-)

Puro bloques TAL

Podemos utilizar atributos TAL sin marcas HTML. Esto es útil cuando no es necesario añadir las etiquetas para el marcado

Sintaxis:

```
<tal:block attribute="expression">some content</tal:block>
```

Ejemplos:

```
<tal:block define="id template/id">
...
  <b tal:content="id">The id of the template</b>
...
</tal:block>

<tal:news condition="python:context.content_type == 'News Item'">
    This text is only visible if the context is a News Item
</tal:news>
```

Manejo de datos complejos en plantillas

Vamos a pasar a un poco más complejo de datos. Y a otro atributo TAL:

tal:replace reemplazar el contenido de un elemento y elimina el elemento dejando sólo el contenido.

Ejemplo:

```
<p>
    <img tal:define="tag string:<img src='https://plone.org/logo.png'>"
         tal:replace="tag">
</p>
```

este resulta en:

```
<p>
    &lt;img src='https://plone.org/logo.png'&gt;
</p>
```

`tal:replace` cae su propia base de etiquetas a favor del resultado de la expresión TALES. Por lo tanto el original `<img... >` se sustituye. Pero el resultado se escapó por defecto.

Para prevenir que se escapen utilizamos `structure`

```
<p>
    <img tal:define="tag string:<img src='https://plone.org/logo.png'>"
```

```
        tal:replace="structure tag">
</p>
```

Ahora vamos a emular una estructura típica Plone mediante la creación de un diccionario.

```
  <table tal:define="talks python:[{'title':'Dexterity for the win!',
                                    'subjects':('content-types', 'dexterity')},
                                   {'title':'Deco is the future',
                                    'subjects':('layout', 'deco')},
                                   {'title':'The State of Plone',
                                    'subjects':('keynote',) },
                                   {'title':'Diazo designs dont suck!',
                                    'subjects':('design', 'diazo', 'xslt')}
                                  ]">
      <tr>
          <th>Title</th>
          <th>Topics</th>
      </tr>
      <tr tal:repeat="talk talks">
          <td tal:content="talk/title">A talk</td>
          <td tal:define="subjects talk/subjects">
              <span tal:repeat="subject subjects"
                    tal:replace="subject">
              </span>
          </td>
      </tr>
  </table>
```

Nosotros emulamos una lista de charlas y mostrar información en una tabla. Nos pondremos en contacto a la lista de las charlas más tarde, cuando se utilizan los verdaderos objetos charlas que hemos creado con dexterity.

Para completar la lista que aquí están los atributos TAL que no hemos utilizado todavía:

tal:omit-tag Omite las etiquetas de los elementos, dejando sólo el contenido interno.

tal:on-error controlar los errores.

Cuando un elemento tiene múltiples sentencias, se ejecutan en este orden:

1. define
2. condition
3. repeat
4. content o replace
5. attributes
6. omit-tag

METAL y macros

¿Por qué nuestra vista es tan feo?, ¿Cómo conseguimos nuestro HTML para representar en la interfaz de usuario de Plone?

Utilizamos METAL (extensión macro para TAL) para definir ranuras que podemos llenar y macros que podemos reutilizar.

Agregamos a la etiqueta `<html>`:

```
metal:use-macro="context/main_template/macros/master"
```

Y a continuación, ajustar el código que queremos poner en el área de contenido de Plone en:

```
<metal:content-core fill-slot="content-core">
    ...
</metal:content-core>
```

Esto pondrá a nuestro código de una sección definida en el main_template llamado "content-core".

Ahora la plantilla debería tener el siguiente aspecto:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
      lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="ploneconf.site">
<body>

<metal:content-core fill-slot="content-core">

<table tal:define="talks python:[{'title':'Dexterity for the win!',
                                  'subjects':('content-types', 'dexterity')},
                                 {'title':'Deco is the future',
                                  'subjects':('layout', 'deco')},
                                 {'title':'The State of Plone',
                                  'subjects':('keynote',) },
                                 {'title':'Diazo designs are great',
                                  'subjects':('design', 'diazo', 'xslt')}
                                ]">
    <tr>
        <th>Title</th>
        <th>Topics</th>
    </tr>
    <tr tal:repeat="talk talks">
        <td tal:content="talk/title">A talk</td>
        <td tal:define="subjects talk/subjects">
            <span tal:repeat="subject subjects"
                  tal:replace="subject">
            </span>
        </td>
    </tr>
</table>

</metal:content-core>

</body>
</html>
```

Nota: Desde la vista demoview solamente es utilizada el contenido de la plantilla, no por el contexto que le toca no tiene mucho sentido tener la barra de edición. Nos lo ocultamos estableciendo la variable correspondiente en el request actual con el python a 1: `request.set('disable_border', 1)`.

La manera más fácil de hacer esto es definir una variable dummy. Dummy porque nunca se usa, salvo que nos permita ejecutar algún código.

```
<metal:block fill-slot="top_slot"
    tal:define="dummy python:request.set('disable_border', 1)" />
```

Macros en browser-views

Define un macro en un nuevo archivo `macros.pt`

```
<div metal:define-macro="my_macro">
    <p>I can be reused</p>
</div>
```

Registrarlo como un BrowserView (esta vez sin una clase Python) en sentencias ZCML:

```
<browser:page
  for="*"
  name="ploneconf.site.macros"
  template="templates/macros.pt"
  permission="zope2.View"
  />
```

Reuse el macro en la plantilla `demoview.pt`:

```
<div metal:use-macro="view/context/@@ploneconf.site.macros/my_macro">
    Instead of this the content of the macro will appear...
</div>
```

Accediendo a Plone desde la plantilla

En nuestra plantilla tenemos acceso al objeto de contexto en el que la vista se llama en adelante, la propia browser-views (es decir, todos los métodos de python vamos a poner en el punto de vista más adelante), los objetos request y response y con ellas podemos conseguir casi cualquier cosa.

En las plantillas también podemos acceder a otras browser-views. Algunos de los que existen para facilitar el acceso a los métodos que a menudo necesitamos:

```
tal:define="context_state context/@@plone_context_state;
            portal_state context/@@plone_portal_state;
            plone_tools context/@@plone_tools;
            plone_view context/@@plone;"
```

`@@plone_context_state` El BrowserView `plone.app.layout.globals.context.ContextState` tiene métodos útiles que tienen que ver con el objeto de contexto actual como `is_default_page`

`@@plone_portal_state` El BrowserView `plone.app.layout.globals.portal.PortalState` tiene métodos para el portal como `portal_url`

`@@plone_tools` El BrowserView `plone.app.layout.globals.tools.Tools` da acceso a las herramientas más importante como el `plone_tools/catalog`

Estos son muy utilizados y hay muchos más.

Lo que echamos de menos

Hay algunas cosas que no nos cubrimos hasta el momento:

`tal:condition="exists:expression"` comprueba si existe un objeto o un atributo (rara vez utilizada)

`tal:condition="nocall:context"` que explícitamente no llamar a un método invocable.

Si nos referimos a objetos de contenido sin usar el modificador nocall: estos objetos son innecesariamente representados en la memoria como la expresión se evaluada.

`i18n:translate` y `i18n:domain` las cadenas que ponemos en las plantillas se pueden traducir de forma automática.

Hay mucho más acerca de TAL, TALES y METAL que no hemos cubierto. Usted solamente aprenderá si sigues leyendo, escribiendo y personalización de plantillas.

Ver también:

- http://docs.plone.org/4/en/adapt-and-extend/theming/templates_css/template_basics.html
- Usando Zope Page Templates: http://docs.zope.org/zope2/zope2book/ZPT.html
- Referencia de Zope Page Templates: http://docs.zope.org/zope2/zope2book/AppendixC.html

Chameleon

Chameleon es el sucesor de TAL y serán incorporado en Plone 5.

- Plip para Chameleon: https://dev.plone.org/ticket/12198
- Página principal: http://www.pagetemplates.org/
- Capa de Integración para Plone: five.pt

En Plone 4 nosotros usaremos ZPT por defecto.

Personalizar plantillas existentes

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/16_zpt_2/ src/ploneconf.site
```

Para profundizar más en los datos real de plone, nosotros ahora miraremos algunas plantillas existentes y las personalizaremos.

El archivo newsitem.pt

Queremos mostrar la fecha de una Noticia se publica. De esta manera la gente puede ver a simple vista que la está mirando es una noticia de actualidad o una noticia antigua.

Para ello vamos a personalizar las plantillas que es utilizanda para representar una Noticia.

Básicamente haremos lo mismo que cuando se utilizó en `portal_skins` (personalizamos el pie de página), pero ahora vamos a hacerlo todo a mano en nuestro paquete.

- Cree el directorio `browser/template_overrides`
- Agregue el siguiente en el archivo `browser/configure.zcml`:

```
<browser:jbot directory="template_overrides" />
```

- Para completar, agregue `z3c.jbot` a las dependencias en el archivo `setup.py` a la lista `install_requires`.
- Buscar el archivo `plone/app/contenttypes/browser/templates/newsitem.pt` en el directorio `omelette` (en la instalación Vagrant esto esta en `/home/vagrant/omelette`).
- Cópielo dentro de la nueva carpeta, ejecutando el siguiente comando Linux: `cp /home/vagrant/omelette/plone/app/contenttypes/browser/templates/newsitem.pt /vagrant/buildout/src/ploneconf.site/ploneconf/site/browser/template_overrides`
- Renombre el nuevo archivo desde `newsitem.pt` a `plone.app.contenttypes.browser.templates.newsitem.pt`
- Reinicie Plone

Ahora Plone debería usar el nuevo archivo sobrescribiendo el original.

Edite la plantilla `plone.app.contenttypes.browser.templates.newsitem.pt` e inserte el siguiente código antes el `<div id="parent-fieldname-text"...`:

```
<p tal:content="python: context.Date()">
    The current Date
</p>
```

- Abra un existente elemento de Noticia en el navegador Web

Esto mostrara algo como esto: `2013-10-02 19:21:15`. No es muy amigable. Vamos a ampliar el código y utilizar una de las muchas funciones helpers que ofrece Plone.

```
<p tal:define="toLocalizedTime nocall:context/@@plone/toLocalizedTime;
               date python:context.Date()"
   tal:content="python:toLocalizedTime(date)">
        The current Date in its local short-format
</p>
```

Ahora debería verse la fecha en un formato amigable como `17.02.2013`.

- Con `nocall:` prevenimos que el método `toLocalizedTime` sea llamado, ya que sólo queremos que esté disponible para su uso.
- El método `toLocalizedTime` es proveído por la BrowserView `Products.CMFPlone.browser.ploneview.Plone` y ejecuta el objeto Date (tipo Fecha) a través del `translation_service` de Plone y regresa la Fecha en el actual formato de locales, así transformando `2013-02-17 19:21:15` en `17.02.2013`.

En antigua versiones de Plone nosotros usamos `python:context.toLocalizedTime(context.Date(), longFormat=False)`. Que llamo al script python `toLocalizedTime.py` en la Carpeta `Products/CMFPlone/skins/plone_scripts/`.

Esa carpeta `plone_scripts` todavía tiene una gran cantidad de scripts útiles que son ampliamente utilizados. Pero todos ellos están obsoletos y honestamente se espera retirar en Plone 5 los cuales sean reemplazados por métodos Python adecuados en BrowserViews.

El archivo summary_view.pt

Utilizamos la vista "Vista de resumen" a la lista de noticias publicadas. También deben tener la fecha. La plantilla asociada con esa vista es `summary_view.pt`.

Echemos un vistazo a la plantilla folder_summary_view.pt:

```
plone/app/contenttypes/browser/templates/summary_view.pt
```

Cópielo al directorio `browser/template_overrides/` y renómbrelo a `plone.app.contenttypes.browser.templates.summary_view.pt`.

Agregue lo siguiente después de la linea 29:

```
<p tal:condition="python:item_type == 'News Item'"
   tal:content="python:toLocalizedTime(item.Date())">
        News date
</p>
```

El método `toLocalizedTime` ya está definido en la plantilla cuya macro usa esta plantillas. ¿Porqué es eso?

El secreto es la linea del archivo `summary_view.pt`:

```
<metal:block use-macro="context/standard_view/macros/entries">
```

`use-macro` le dice a Plone reusar el macro `entries` desde la vista `standard_view` la cual se encuentra en la plantilla `plone/app/contenttypes/browser/templates/standard_view.pt`.

Las plantillas `summary_view.pt` y `folder_summary_view.pt` (las cuales es lo mismo, pero para las carpetas, no para las colecciones) están muy difundidas y también ampliamente personalizadas, para requisitos particulares, por lo que también podría llegar a conocerlo un poco.

Nuestra adiciones hace que la fecha de los objetos respectivo que las iteraciones de plantilla (por lo tanto `item` en lugar de `context` desde `context` sería la colección de la agregación de las noticias).

La fecha solamente es mostrada si la variable `item_type` (definida en la linea 42 de la vista `standard_view.pt`) sea `News Item`.

Hay mucho más en `standard_view.pt` y `summary_view.pt` pero vamos a dejar las cosas así.

Nota: En defecto de Plone sin `plone.app.contenttypes` esto sería `folder_summary_view.pt`, una plantilla skin de Archetypes se puede encontrar en la carpeta `Products/CMFPlone/skins/plone_content/`. La plantilla personalizada sería `Products.CMFPlone.skins.plone_content.folder_summary_view.pt`.

La plantilla Archetypes de Noticia es `newsitems_view.pt` de la misma carpeta. La plantilla personalizada tendría que ser nombrada `Products.CMFPlone.skins.plone_content.folder_summary_view.pt`.

¡Tenga en cuenta que no sólo los nombres han cambiado, sino también el contenido!

Buscando la correcta plantilla

Hemos cambiado la pantalla de la lista de elementos de noticias en http://localhost:8080/Plone/news. Pero, ¿cómo sabemos que plantilla para personalizar?

Si usted no sabe con qué plantilla utiliza la página que estás viendo, usted puede hacer una conjetura, iniciar una sesión de depuración o use el paquete `plone.app.debugtoolbar` para obtener una vista que ofrece información de depuración del actual contexto.

1. Pudimos comprobar el HTML con la herramienta Firebug y buscar una estructura en el área de contenido que parece único. También podríamos buscar la clase CSS del cuerpo

```
<body class="template-summary_view portaltype-collection site-Plone section-news subsection-aggr
```

   La clase `template-summary_view` nos dice que el nombre de la vista (pero no necesariamente el nombre de la plantilla) el cual es `summary_view`. Así que podríamos buscar en todos los archivos `*.zcml` la cadena `name="summary_view"` o buscar todas las plantillas llama `summary_view.pt` y probablemente encontrar la vista y también la plantilla correspondiente. Pero probablemente sólo porque esa no nos diría si ya está siendo sobrescrita la plantilla.

2. El método es más seguro está usando el paquete `plone.app.debugtoolbar`. Ya lo tenemos en nuestra configuración buildout definidos y sólo hay que instalarlo. Ese agregue un menú desplegable "Debug" en la parte superior de la página. La sección "Publicado" muestra la ruta completa a la plantilla que se utiliza para representar la página que estás viendo.

3. La sesión de depuración para encontrar la plantilla es un poco más complicado. Entonces tenemos el producto `Products.PDBDebugMode` en nuestra configuración buildout podemos llamar `/pdb` desde nuestra página.

   El objeto que las URL apunta por defecto `self.context`. Pero el primer problema es que la dirección URL que estamos viendo no es la dirección URL de la colección en la que queremos modificar ya que la colección es la página por defecto de la carpeta `news`.

```
>>> (Pdb) self.context
<Folder at /Plone/news>
>>> (Pdb) obj = self.context.aggregator
>>> (Pdb) obj
<Collection at /Plone/news/aggregator>
>>> (Pdb) context_state = obj.restrictedTraverse('@@plone_context_state')
>>> (Pdb) template_id = context_state.view_template_id()
>>> (Pdb) template_id
'summary_view'
>>> (Pdb) view = obj.restrictedTraverse('summary_view')
>>> (Pdb) view
<Products.Five.metaclass.SimpleViewClass from /Users/philip/.cache/buildout/eggs/plone.app.conte
>>> view.index.filename
u'/Users/philip/workspace/training_without_vagrant/src/ploneconf.site/ploneconf/site/browser/tem
```

Ahora puede ver que nosotros personalizamos la plantilla.

Plantillas skins

¿Por qué no siempre sólo utilizamos plantillas? Porque lo que se quiere hacer es algo más complicado que obtener un atributo de formulario en el contexto y hacer su valor en alguna etiqueta HTML.

Es una tecnología obsoleta llamada 'skin-templates' que le permite añadir simplemente alguna página en la plantilla (por ejemplo, 'old_style_template.pt') a una carpeta determinada en el ZMI o su paquete egg) y se puede acceder a ella en el navegador mediante la dirección URL como esta http://localhost:8080/Plone/old_style_template y se mostrara. Pero nosotros no lo usamos y usted también no hay que a pesar de que estos de las skin-templates siguen siendo todo Plone.

Desde que usamos el paquete `plone.app.contenttypes` no encontramos nunca más muchas plantillas skin cuando se trata de contenidos. Pero más a menudo que usted no tendrás que personalizar un sitio antiguo que aún utiliza la plantillas skin.

Las plantillas Skins y los script Python están ubicados en la herramienta portal_skin están descontinuados porque:

- ellos son Python restringido
- no tienen buena manera de adjuntar código Python a ellos
- ellos son siempre invocable para todo el mundo (ellos no pueden ser fácilmente unidos a una interfaz)

Vistas - Parte II: Una vista por defecto para la “charla”

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/17_views_2/ src/ploneconf.site
```

Vista de clases

Anteriormente hemos escrito una vista de demostración que también utilizamos para experimentar con las plantillas de página. Vamos a echar un vistazo a la ZCML y la Page Template de nuevo. Yo he ampliado el código sólo un poco.

El archivo `browser/configure.zcml`

```
<configure xmlns="http://namespaces.zope.org/zope"
    xmlns:browser="http://namespaces.zope.org/browser"
    i18n_domain="ploneconf.site">

    <browser:page
       name="demoview"
       for="*"
       layer="zope.interface.Interface"
       class=".views.DemoView"
       template="templates/demoview.pt"
       permission="zope2.View"
       />

</configure>
```

El archivo `browser/views.py`

```
from Products.Five.browser import BrowserView

class DemoView(BrowserView):
    """ This does nothing so far
    """

    def __init__(self, context, request):
        self.context = context
        self.request = request
```

```
    def __call__(self):
        # Implement your own actions

        # This renders the template that was registered in zcml like this:
        #   template="templates/demoview.pt"
        return super(DemoView, self).__call__()
        # If you don't register a template in zcml the Superclass of
        # DemoView will have no __call__-method!
        # In that case you have to call the template like this:
        # from Products.Five.browser.pagetemplatefile import ViewPageTemplateFile
        # class DemoView(BrowserView):
        # template = ViewPageTemplateFile('templates/demoview.pt')
        # def __call__(self):
        #    return self.template()
```

¿Recuerdas el término MultiAdapter? La browser page es sólo un MultiAdapter. La declaración ZCML `browser:page` registra un MultiAdapter y agrega cosas adicionales que se necesitan para una browser view.

Un adaptador adapta cosas, un MultiAdapter adapta múltiple cosas.

Al introducir una dirección URL, Zope trata de encontrar un objeto para él. Al final, cuando Zope no encuentra ningún objeto más, pero todavía hay un elemento de ruta, o no hay más elementos de rutas, Zope busca un adaptador que va a responder a la solicitud.

El adaptador se adapta la solicitud y el objeto que Zope ha encontrado con la dirección URL. La clase de adaptador se crea instanciada con los objetos que han de adaptarse, entonces se llama.

El código fuente anterior hace la misma cosa que la implementación estándar haría. Hace que el contexto y la solicitud estén disponibles como variables en el objeto.

Yo he escrito estos métodos porque es importante entender algunos conceptos importantes.

El método init es llamado mientras Zope es todavía esta *tratando* de encontrar una vista. En esa fase, la seguridad no se ha resuelto. El código no comprueba la seguridad. Por razones históricas, muchos errores que se producen en el método init puede resultar en una página no encuentre el error en lugar de una excepción.

No hagas mucho a todos en el método init. En su lugar usted tiene la garantía que el método call es llamado antes de cualquier otra cosa (excepto el método init). Tiene los controles de seguridad en su lugar y así sucesivamente.

Desde un punto de vista práctico, considere el método call en su método init, la mayor diferencia es que este método debe retornar el HTML listo. Deje a su clase base manejar la generación de HTML.

La vista por defecto

Ahora nosotros finalmente agregamos la vista por defecto para las charlas en views.py

El archivo `browser/configure.zcml`

```
<browser:page
   name="talkview"
   for="*"
   layer="zope.interface.Interface"
   class=".views.TalkView"
   template="templates/talkview.pt"
   permission="zope2.View"
   />
```

El archivo `browser/views.py`

```
from plone.dexterity.browser.view import DefaultView

...

class TalkView(DefaultView):
    """ The default view for talks
    """
```

La clase base DefaultView en el paquete plone.dexterity solamente existe para los Objetos Dextertity y tiene algo de gran utilidad a disposición de la plantilla:

- view.w es un diccionario de todos los widgets de pantalla, con clave de nombres de campos. Esto incluye widgets de conjunto de campos alternativos.
- view.widgets contiene una lista de widgets en esquema de ordenar para el conjunto de campos predeterminado.
- view.groups contiene una lista de conjunto de campos con el fin de ordenar conjunto de campo.
- view.fieldsets contiene un nombre de conjunto de campo mapeado un diccionario a conjunto de campo
- En un conjunto de campos (grupo), puede acceder a una lista de widgets para obtener los widgets en ese conjunto de campo

Nota: La vista `plone.dexterity.browser.view.DefaultView` tiene las mismas características que el equivalente Grok `plone.directives.dexterity.DisplayForm`.

La plantilla `templates/talkview.pt` usa el patrón `view/w/<fieldname>/render` para hacer algunos widgets.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
    lang="en"
    metal:use-macro="context/main_template/macros/master"
    i18n:domain="ploneconf.site">
<body>
    <metal:content-core fill-slot="content-core">
        <p>Suitable for <em tal:replace="structure view/w/audience/render"></em>
        </p>

        <div tal:content="structure view/w/details/render" />

        <div tal:content="context/speaker">
            User
        </div>
    </metal:content-core>
</body>
</html>
```

Después de un reinicio, podemos probar nuestra vista, yendo a un tipo de contenido talk agregado y agrégale a la dirección URL /talkview.

Debemos decirle a Plone, que la vista talkview debe ser utilizado como la vista predeterminada para las charlas en lugar de la vista incorporada.

Esta es una configuración que puede cambiar en tiempo de ejecución y se almacena en la base de datos, como tal, también es gestionado por perfiles GenericSetup.

Nota: Para cambiarlo a través de la Web valla a la ZMI (http://localhost:8080/Plone/manage), luego vaya a la herramienta `portal_types` y seleccione el tipo para el cual el nuevo tipo de contenido debe ser seleccionable (*talk*).

Ahora agregue `talkview` a la lista *Available view methods*. Ahora la nueva vista está disponible en el menú *Mostrar*. Para que sea la vista predeterminada ingrese esta vista en la sección `Default view method`.

Abra el archivo `profiles/default/types/talk.xml`:

```
...
<property name="default_view">talkview</property>
<property name="view_methods">
    <element value="talkview"/>
    <element value="view"/>
</property>
...
```

Vamos a tener que reinstalar nuestro complemento o ejecutar el paso GenericSetup llamado `typeinfo` entonces Plone aprende sobre el cambio.

También podríamos decirle a Plone sobre esto en el ZMI: http://localhost:8080/Plone/portal_types/talk/manage_propertiesForm

Mejoremos la vista talkview para mostrar toda la información que queremos.

El archivo `templates/talkview.pt`:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="ploneconf.site">
<body>
    <metal:content-core fill-slot="content-core">

        <p>
            <span tal:content="context/type_of_talk">
                Talk
            </span>
            suitable for
            <span tal:replace="structure view/w/audience/render">
                Audience
            </span>
        </p>

        <div tal:content="structure view/w/details/render">
            Details
        </div>

        <div class="newsImageContainer">
            <img tal:condition="python:getattr(context, 'image', None)"
                 tal:attributes="src string:${context/absolute_url}/@@images/image/thumb" />
        </div>

        <div>
            <a class="email-link" tal:attributes="href string:mailto:${context/email}">
                <strong tal:content="context/speaker">
                    Jane Doe
                </strong>
            </a>
            <div tal:content="structure view/w/speaker_biography/render">
                Biography
            </div>
        </div>

    </metal:content-core>
```

```
38 </body>
39 </html>
```

Ejercicio

Agregar el nuevo campo "room" con el tipo de contenido Talk (a través de la Web) y mostrarlo debajo del campo Audience en la browser view, debe contener los siguientes datos:

- Título: Room
- Posible valores: Room 101, Room 102, Auditorium

Solución

- Valla a la dirección URL http://localhost:8080/Plone/dexterity-types/talk/@@fields y agregue los nuevos campos
- Agregar el nuevo HTML siguiente en la parte de audiencia:

```
<p>
    <span tal:replace="structure view/w/room/render">
        Room
    </span>
</p>
```

Ver también:

http://docs.plone.org/4/en/develop/plone/views/browserviews.html

CAPÍTULO 19

Vistas - Parte III: Una lista de Charlas

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/18_views_3/ src/ploneconf.site
```

Ahora no queremos proporcionar información sobre un elemento específico, sino en varios elementos. ¿Y ahora qué? No podemos mirar a varios elementos al mismo tiempo que el contexto.

Usando portal_catalog

Digamos que queremos mostrar una lista de todas las charlas que se presentaron durante la conferencia. Sólo podemos ir a la carpeta y seleccione un método de visualización que nos convenga. Pero ninguno lo hace porque queremos para mostrar al público objetivo en nuestro listado.

Así que tenemos que conseguir todas las charlas. Para ello utilizamos la vista de la clase Python para consultar en el catálogo las charlas.

El catálogo es como un motor de búsqueda por el contenido de nuestro sitio. Contiene información sobre todos los objetos, así como algunos de sus atributos como título, descripción, desde estado de flujo de trabajo, palabras clave que fueron etiquetadas, autor, tipo de contenido, su ruta en el sitio, etc. Sin embargo, no se mantiene el contenido de campos "pesado" como imágenes o archivos, campos texto enriquecidos y el campo que acabamos de definir nosotros mismos.

Es la forma más rápida de obtener el contenido que existe en el sitio y hacer algo con él. A partir de los resultados del catálogo podemos conseguir los mismos objetos, pero a menudo no lo necesita, pero sólo las propiedades que los resultados ya tienen.

El archivo `browser/configure.zcml`

```
<browser:page
   name="talklistview"
   for="*"
   layer="zope.interface.Interface"
   class=".views.TalkListView"
   template="templates/talklistview.pt"
   permission="zope2.View"
   />
```

El archivo `browser/views.py`

```
from Products.Five.browser import BrowserView
from plone import api
from plone.dexterity.browser.view import DefaultView


class DemoView(BrowserView):
    """ This does nothing so far
    """


class TalkView(DefaultView):
    """ The default view for talks
    """


class TalkListView(BrowserView):
    """ A list of talks
    """

    def talks(self):
        results = []
        portal_catalog = api.portal.get_tool('portal_catalog')
        current_path = "/".join(self.context.getPhysicalPath())

        brains = portal_catalog(portal_type="talk",
                                path=current_path)
        for brain in brains:
            talk = brain.getObject()
            results.append({
                'title': brain.Title,
                'description': brain.Description,
                'url': brain.getURL(),
                'audience': ', '.join(talk.audience),
                'type_of_talk': talk.type_of_talk,
                'speaker': talk.speaker,
                'uuid': brain.UID,
                })
        return results
```

Nosotros consultamos el catálogo para dos cosas:

- `portal_type = "talk"`
- `path = "/".join(self.context.getPhysicalPath())`

Obtenemos la ruta del contexto actual para consultar sólo para los objetos en la ruta actual. De lo contrario íbamos a obtener todas las charlas en todo el sitio. Si nos trasladamos algunas charlas a una parte diferente del sitio (por ejemplo, un sub-conferencia para universidades con una especial lista charla) puede ser que no quiera ver así en nuestra lista.

Nosotros iteramos sobre la lista de resultados que el catálogo nos devuelve.

Creamos un diccionario que contiene toda la información que queremos mostrar en la plantilla. De esta manera no tiene que poner ninguna lógica compleja en la plantilla.

Cerebros y objetos

Los objetos normalmente no se cargan en la memoria, pero permanecen latentes en el base de datos ZODB. Despertar objetos hasta puede ser lento, especialmente si usted está despertando una gran cantidad de objetos. Afortunadamente

nuestros tipos de contenidos no son especialmente pesados, ya que son:

- objetos dexterity que son más ligeros que sus hermanos archetypes
- relativamente pocos, ya que no tenemos miles de charlas en nuestra conferencia

Queremos mostrar la audiencia destinada de la charla, pero esos atributos de las charlas no está en el catálogo. Esto es por qué necesitamos llegar a los objetos mismos.

También podríamos agregar un nuevo índice al catálogo que agregará 'audience' a las propiedades de los cerebros. Tenemos que ponderar los pros y contras:

- charlas son importantes y por lo tanto más probable es que siempre este en memoria
- prevenir la inflamación del catálogo con índices

Nota: El código para agregar este índice se vería como el siguiente:

```
from plone.indexer.decorator import indexer
from ploneconf.site.talk import ITalk

@indexer(ITalk)
def talk_audience(object, **kw):
     return object.audience
```

Tendríamos que registrar esta función factory como un adaptador llamado en el archivo `configure.zcml`. Suponiendo que haya puesto el código fuente anterior en un archivo llamado indexers.py

```
<adapter name="audience" factory=".indexers.talk_audience" />
```

Vamos a agregar algunos índices más adelante.

¿Por qué utilizar el catálogo en absoluto?, comprueba los permisos, y sólo devuelve las charlas que el usuario actual puede ver. Puede ser que sean privados u ocultos a usted, ya que son parte de una súper conferencia secreta para los desarrolladores del núcleo de Plone (¡no hay tal cosa!).

La mayoría de los objetos en Plone son como diccionarios, por lo que podían hacer `context.values()` para obtener todos sus contenidos.

Por razones históricas, algunos atributos de los cerebros y los objetos se escriben de manera diferente:

```
>>> obj = brain.getObject()

>>> obj.title
u'Talk-submission is open!'

>>> brain.Title == obj.title
True

>>> brain.title == obj.title
False
```

¿Quién puede adivinar lo que `brain.title` volverán ya que el cerebro no tiene ese atributo?

Nota: Respuesta: La Adquisición obtendrá el atributo de servidor principal más cercano. `brain.__parent__` es `<CatalogTool at /Plone/portal_catalog>`. El atributo `title` del `portal_catalog` es 'Indexado en todo el contenido en el sitio'.

La Adquisición puede ser nocivo. Los cerebros no tienen el atributo 'getLayout' `brain.getLayout()`:

```
>>> brain.getLayout()
'folder_listing'

>>> obj.getLayout()
'newsitem_view'

>>> brain.getLayout
<bound method PloneSite.getLayout of <PloneSite at /Plone>>
```

Lo mismo es cierto para los métodos:

```
>>> obj.absolute_url()
'http://localhost:8080/Plone/news/talk-submission-is-open'
>>> brain.getURL() == obj.absolute_url()
True
>>> brain.getPath() == '/'.join(obj.getPhysicalPath())
True
```

Consultando el catálogo

Hay muchos índices de catálogo para consultar. He aquí algunos ejemplos:

```
>>> portal_catalog = getToolByName(self.context, 'portal_catalog')
>>> portal_catalog(Subject=('cats', 'dogs'))
[]
>>> portal_catalog(review_state='pending')
[]
```

Llamando al catálogo sin parámetros devuelven todo el sitio:

```
>>> portal_catalog()
[<Products.ZCatalog.Catalog.mybrains object at 0x1085a11f0>, <Products.ZCatalog.Catalog.mybrains obje
```

Ver también:

http://docs.plone.org/4/en/develop/plone/searching_and_indexing/query.html

Ejercicios

Ya que ahora saben cómo consultar el catálogo es el momento para un poco de ejercicio.

Ejercicio 1

Agregue un método `get_news` a `TalkListView` que devuelve una lista de los cerebros de todos los elementos de Noticias que se publican y ordenarlos en el orden de su publicación fecha.

Solución

```
1 def get_news(self):
2
3     portal_catalog = api.portal.get_tool('portal_catalog')
4     return portal_catalog(
5         portal_type='News Item',
```

```
        review_state='published',
        sort_on='effective',
    )
```

Ejercicio 2

Agregar un método que regresa todas los tipos de contenidos Talk de tipo keynotes publicadas como objetos.

Solución

```
def keynotes(self):

    portal_catalog = api.portal.get_tool('portal_catalog')
    brains = portal_catalog(
        portal_type='Talk',
        review_state='published')
    results = []
    for brain in brains:
        # There is no catalog-index for type_of_talk so we must check
        # the objects themselves.
        talk = brain.getObject()
        if talk.type_of_talk == 'Keynote':
            results.append(talk)
    return results
```

La plantilla para la lista

A continuación, crear una plantilla en la usa los resultados del método 'talks'.

Trate de mantener la lógica mayormente en Python. Esto es por dos razones:

Legibilidad: Es mucho más fácil de leer Python que complejas estructuras de TAL.

Velocidad: El código Python es más rápido que el código ejecutado en las plantillas. También es fácil de añadir almacenamiento en caché a los métodos.

Don't Repeat Yourself (DRY): En Python se puede volver a utilizar métodos y refactorizar código fácilmente. Refactorizando TAL generalmente significa tener que hacer grandes cambios en la estructura HTML que se traduce en diffs incomprensible.

El esquema MVC no se aplica directamente a Plone pero míralo de esta manera:

Modelo: el objeto

Vista: la plantilla

Controlador: la vista

La vista y el controlador están muy mezclados en Plone. Especialmente cuando nos fijamos en algunos de los códigos más antiguos de Plone verás que la política de mantener la lógica en Python y representación en las plantillas no siempre se cumple.

¡Pero usted debe, no obstante, que lo haga! Usted va a terminar con más que suficiente lógica en las plantillas de todos modos.

Agregar esta sencilla tabla al archivo `templates/talklistview.pt`:

```
<table class="listing">
    <thead>
        <tr>
            <th>
                Title
            </th>
            <th>
                Speaker
            </th>
            <th>
                Audience
            </th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>
               The 7 sins of plone-development
            </td>
            <td>
                Philip Bauer
            </td>
            <td>
                Advanced
            </td>
        </tr>
    </tbody>
</table>
```

Después usted lo transforma dentro de una lista:

```
<table class="listing" id="talks">
    <thead>
        <tr>
            <th>
                Title
            </th>
            <th>
                Speaker
            </th>
            <th>
                Audience
            </th>
        </tr>
    </thead>
    <tbody>
        <tr tal:repeat="talk view/talks">
            <td>
                <a href=""
                   tal:attributes="href talk/url;
                                   title talk/description"
                   tal:content="talk/title">
                   The 7 sins of plone-development
                </a>
            </td>
            <td tal:content="talk/speaker">
                Philip Bauer
```

```
            </td>
            <td tal:content="talk/audience">
                Advanced
            </td>
        </tr>
        <tr tal:condition="not:view/talks">
            <td colspan=3>
                No talks so far :-(
            </td>
        </tr>
    </tbody>
</table>
```

Hay algunas cosas que necesitan explicación:

tal:repeat="talk view/talks" Este itera sobre la lista de diccionarios devueltos por la vista. La vista `view/talks` llama al método `talks` de nuestro vista y cada `talk` es a su vez regresa uno de los diccionarios que son devueltos por este método. Desde las expresiones de ruta TAL para la búsqueda de los valores en los diccionarios es la misma que los atributos de los objetos se puede escribir `talk/somekey` como pudimos `view/somemethod`. Práctico, pero a veces irritante ya de mirar la page template solos a menudo no tenemos forma de saber si algo es un atributo, un método o el valor de un diccionario. Sería una buena práctica para escribir `tal:repeat="talk python:view.talks()"`.

tal:content="talk/speaker" 'speaker' es una clave en el diccionario 'talk'. También podríamos escribir `tal:content="python:talk['speaker']"`

tal:condition="not:view/talks" Esta es una alternativa en caso de no se devuelven las charlas. A continuación, devolver una lista vacía (¿recuerda `results = []`?)

Ejercicio

Modificar la vista para solamente expresiones Python.

Solución

```
<table class="listing" id="talks">
    <thead>
        <tr>
            <th>
                Title
            </th>
            <th>
                Speaker
            </th>
            <th>
                Audience
            </th>
        </tr>
    </thead>
    <tbody tal:define="talks python:view.talks()">
        <tr tal:repeat="talk talks">
            <td>
                <a href=""
                   tal:attributes="href python:talk['url'];
                                   title python:talk['description']"
                   tal:content="python:talk['title']">
                   The 7 sins of plone-development
```

```
                </a>
            </td>
            <td tal:content="python:talk['speaker']">
                Philip Bauer
            </td>
            <td tal:content="python:talk['audience']">
                Advanced
            </td>
        </tr>
        <tr tal:condition="python:not talks">
            <td colspan=3>
                No talks so far :-(
            </td>
        </tr>
    </tbody>
</table>
```

Para seguir el principio "don't repeat yourself" nosotros definimos `talks` vez de llamar al método dos veces.

Configuración de una vista personalizada como vista por defecto en un objeto

No queremos tener siempre que a anexar /@@talklistview a nuestra carpeta para obtener la vista. Hay una manera muy fácil de configurar la vista de la carpeta con el ZMI.

Si agregamos `/manage_propertiesForm` podemos establecer la propiedad "layout" para la vista `talklistview`.

Para hacer vistas configurables para que los editores pueden elegir entonces tenemos que registrar la vista para el tipo de contenido que nos ocupa, en que es FTI. Para activar si todas las carpetas tiene esta vista se añade un nuevo archivo `profiles/default/types/Folder.xml`

```
<?xml version="1.0"?>
<object name="Folder">
 <property name="view_methods" purge="False">
  <element value="talklistview"/>
 </property>
  <alias from="@@talklistview" to="talklistview"/>
</object>
```

Después de volver a aplicar el perfil typeinfo del complemento (o, simplemente, volver a instalarlo) el tipo de contenido "Carpeta" se amplía con nuestro método de vista adicional y aparece en la lista desplegable Mostrar.

La opción `purge="False"` anexa a la vista de los ya existentes en lugar de reemplazarlos.

Agregando un poco de Javascript (collective.js.datatables)

Aquí se utiliza una de las muchas agradables característica construida en Plone. La class="listing" da la tabla un buen estilo y hace que la tabla se puede ordenar con un poco de Javascript.

Pero podríamos mejorar esa tabla aún más mediante el uso de una buena librería Javascript llamada "datatables". Incluso podría llegar a ser parte del core de Plone en algún momento.

Al igual que para muchas librerías Javascript, ya hay un paquete que hace la integración de Plone por nosotros: `collective.js.datatables`. Al igual que todos los paquetes Python usted lo puede encontrar en PYPI: https://pypi.python.org/pypi/collective.js.datatables

Ya hemos añadido el complemento a nuestra buildout, sólo tienes que activarlo en nuestra plantilla.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="ploneconf.site">
<body>

<metal:head fill-slot="javascript_head_slot">
    <link rel="stylesheet" type="text/css" media="screen" href="++resource++jquery.datatables/media/c

    <script type="text/javascript" src="++resource++jquery.datatables.js"></script>
    <script type="text/javascript">
        $(document).ready(function(){
            var oTable = $('#talks').dataTable({
            });
        })
    </script>
</metal:head>

<metal:content-core fill-slot="content-core">

    <table class="listing" id="talks">
        <thead>
            <tr>
                <th>
                    Title
                </th>
                <th>
                    Speaker
                </th>
                <th>
                    Audience
                </th>
            </tr>
        </thead>
        <tbody>
            <tr tal:repeat="talk view/talks">
                <td>
                    <a href=""
                       tal:attributes="href talk/url;
                                       title talk/description"
                       tal:content="talk/title">
                       The 7 sins of plone-development
                    </a>
                </td>
                <td tal:content="talk/speaker">
                    Philip Bauer
                </td>
                <td tal:content="talk/audience">
                    Advanced
                </td>
            </tr>
            <tr tal:condition="not:view/talks">
                <td colspan=3>
                    No talks so far :-(
```

```
                </td>
            </tr>
        </tbody>
    </table>

</metal:content-core>
</body>
</html>
```

Nosotros no necesitamos la clase CSS `listing` nunca más, ya que podría entrar en conflicto con la librería datatables (no es así, pero aún así ...).

La documentación de la librería datatables está más allá de nuestro entrenamiento.

Utilizamos METAL de nuevo pero esta vez para llenar un slot diferente. El "javascript_head_slot" es parte de área <head> del HTML en Plone y se puede ampliar de esta manera. También podríamos simplemente poner el código fuente en línea, pero con HTML muy bien ordenado es una buena práctica.

Hagamos una prueba con la siguiente dirección URL: http://localhost:8080/Plone/talklistview

---

Nota: Añadimos el archivo `jquery.datatables.js` directamente a la ranura HEAD del HTML sin utilizar un registro en la herramienta Plone JavaScript registry (portal_javascript). Al utilizar el registro podría permitir la fusión de los archivos js y almacenamiento en caché avanzado. Un perfil GenericSetup está incluido en el paquete collective.js.datatables.

Resumen

Hemos creado un bonito listado, que se puede llamar en cualquier lugar en el sitio web.

CAPÍTULO 20

Comportamientos

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/19_behaviors_1/ src/ploneconf.site
```

Usted puede extender la funcionalidad de su objeto Dexterity solo creando un adaptador que se adapte a su objeto Dexterity para agregar otra característica o aspecto.

Pero si usted desea utilizar este adaptador, debe saber saber de alguna manera que un objeto implementa este. Además, usted no puede fácilmente agregar más campos a un objeto con ese enfoque.

Enfoque Dexterity

Dexterity tiene una solución para ello, con adaptadores especiales los cuales se llaman y registran por el nombre del comportamiento.

Un comportamiento puede ser agregado a cualquier tipo de contenido a través de la Web y durante el tiempo de ejecución.

Todas las vistas predeterminadas deben conocer el concepto de comportamientos y cuando renderizar los formularios, las vistas también comprueban si hay comportamientos referenciados con el contexto actual y si estos comportamientos tienen un esquema propio, esos campos se mostraran adicionalmente.

Nombres y Teoría

El nombre del comportamiento no es un término estándar en el desarrollo de software. Pero es una buena idea pensar en un comportamiento como un aspecto. Usted está agregando un aspecto a tu tipo de contenido y usted quiere escribir su aspecto de tal manera, que funciona independientemente del tipo de contenido en el que se aplica el aspecto. Usted no debe tener dependencias a campos específicos de su objeto o de otros comportamientos.

Como un objeto le permite aplicar el principio Abierto/cerrado a los objetos de Dexterity.

Ejemplo práctico

Así que, vamos a escribir nuestro propio pequeño comportamiento.

En el futuro, queremos que nuestra presentación este representada también en Lanyrd (un Directorio Conferencia Social - Lanyrd.com). Por ahora nos limitaremos a ofrecer un enlace para que los visitantes puedan colaborar fácilmente con el sitio Lanyrd.

Así que por ahora, nuestro comportamiento sólo añade un nuevo campo para almacenar la dirección URL del servicio Lanyrd.

Queremos mantener una estructura limpia, así que creamos primero un directorio llamado `behaviors`, y luego lo incluimos dentro de las declaraciones ZCML de nuestro archivo `configure.zcml`.

```
<include package=".behaviors" />
```

Entonces, agregamos un archivo vació `behaviors/__init__.py` y un archivo `behaviors/configure.zcml` conteniendo el siguiente código:

Referencia avanzada

La documentación original tiene código doctest, así que no hay documentación y ninguna prueba depurable.
Puede ser un poco confuso cuándo se utilizar la factory, o las interfaces marker y cuando no.
Si usted no define un factory, sus atributos se almacenan directamente en el objeto. Esto puede dar lugar a comportamientos no deseados.
Esto se puede evitar mediante el uso de la factory `plone.behavior.AnnotationStorage`. Éste almacena sus atributos en una Annotation. Pero entonces usted *debe* utilizar una interfaz marker si usted quiere tener viewlets, browser views o portlets personalizados.
Sin ella, no tendrías interfaz contra el cual podría registrar sus vistas.

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:plone="http://namespaces.plone.org/plone"
    i18n_domain="ploneconf.site">

  <plone:behavior
      title="Social Behavior"
      description="Adds a link to lanyrd"
      provides=".social.ISocial"
      />

</configure>
```

Y un archivo `behaviors/social.py` conteniendo el siguiente código:

```
from plone.supermodel import model, directives
from plone.autoform.interfaces import IFormFieldProvider
from zope import schema
from zope.interface import alsoProvides


class ISocial(model.Schema):

    directives.fieldset(
        'social',
        label=u'Social',
        fields=('lanyrd',),
    )

    lanyrd = schema.URI(
        title=u"Lanyrd-link",
        description=u"Add URL",
```

```
        required=False,
    )

alsoProvides(ISocial, IFormFieldProvider)
```

Vamos a llevarlo a través de este paso a paso.

1. Registramos un comportamiento en el archivo *behaviors/configure.zcml*. Nosotros decimos para qué tipo de contenido este comportamiento es válido. Para hacer esto, a través de la Web o en el perfil GenericSetup.
2. Creamos una interfaz marker en *behaviors/social.py* para nuestro comportamiento y hacerlo también un esquema el cual contiene los campos que queremos declarar. Podríamos simplemente utilizar definiciones de campos de esquema en una clase `zope.interface`, pero utilizaremos un formulario extendido desde el paquete plone.supermodel, de lo contrario nosotros podríamos no usar las características del conjunto de campos.
3. También añadimos un fieldset para que nuestros campos no se mezclen con los campos normales del objeto.
4. Añadimos un campo de esquema URI normal para almacenar la dirección URI del servicio Lanyrd.
5. Nosotros marcamos nuestro esquema como una clase que también implementa la interfaz IFormFieldProvider. Se trata de una interfaz marker, nosotros no necesitamos implementar nada para proporcionar la interfaz.

Agregándolo a nuestra Talk

Podríamos añadir este comportamiento ahora a través del panel de control de Plone llamado `Tipos de contenidos Dexterity`. Pero en cambio, lo haremos directamente de forma apropiada en nuestro perfil GenericSetup

Debemos agregar el comportamiento al archivo `profiles/default/types/talk.xml`:

```
<?xml version="1.0"?>
<object name="talk" meta_type="Dexterity FTI" i18n:domain="plone"
   xmlns:i18n="http://xml.zope.org/namespaces/i18n">
   ...
 <property name="behaviors">
  <element value="plone.app.dexterity.behaviors.metadata.IDublinCore"/>
  <element value="plone.app.content.interfaces.INameFromTitle"/>
  <element value="ploneconf.site.behaviors.social.ISocial"/>
 </property>
 ...
</object>
```

CAPÍTULO 21

Escribiendo Viewlets

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/20_viewlets_1/ src/ploneconf.site
```

Un viewlet para el comportamiento social

Un viewlet no es una vista, pero es un fragmento de HTML y la lógica Python que se puede colocar en varios lugares en el sitio. Estos lugares se llaman `ViewletManager`.

- Inspeccione y administre los viewlets existentes yendo a la dirección URL http://localhost:8080/Plone/@@manage-viewlets.
- Ya hemos personalizado un viewlet (el archivo `collophon.pt`). Ahora agregamos un nuevo viewlet.
- Los viewlets no guardan los datos (los portlets si lo hacen).
- Los viewlets no tienen interfaz de usuario (los portlets lo hacen).

El viewlet social-viewlet

Vamos a añadir un enlace al sitio que utiliza la información que hemos recopilado utilizando el comportamiento social.

Registramos el viewlet en el archivo `browser/configure.zcml`, agregando el siguiente código ZCML:

```
<browser:viewlet
  name="social"
  for="ploneconf.site.behaviors.social.ISocial"
  manager="plone.app.layout.viewlets.interfaces.IBelowContentTitle"
  class=".viewlets.SocialViewlet"
  layer="zope.interface.Interface"
  template="templates/social_viewlet.pt"
  permission="zope2.View"
  />
```

Esto registra un viewlet llamado `social`. Es visible en todos los contenidos que implementa la interfaz `ISocial` de nuestro comportamiento. También es una buena práctica de obligar a la BrowserLayer llamada

`IPloneconfSiteLayer` de nuestro complemento para que sólo sea mostrado si nuestro complemento esta instalado en realidad.

La clase viewlet `SocialViewlet` se espera en un archivo `browser/viewlets.py`, agregando el siguiente código Python:

```
from plone.app.layout.viewlets import ViewletBase

class SocialViewlet(ViewletBase):
    pass
```

Esta clase no hace nada excepto hacer que la plantilla asociada (Esa tenemos aún que escribirla)

Nota: Si utilizamos `Grok` no necesitaríamos registrar los viewlets en el archivo `configure.zcml` pero si hay que hacerlo en código Python. Nos gustaría añadir un archivo `viewlets.py` que contiene la clase viewlet, la cual se presenta a continuación:

```
from five import grok
from plone.app.layout.viewlets import interfaces as viewletIFs
from zope.component import Interface

class SocialViewlet(grok.Viewlet):
    grok.viewletmanager(viewletIFs.IBelowContentTitle)
```

Esto haría lo mismo que el código anterior utilizando el paradigma de `Grok` de la convención sobre la configuración. En las browser views la referencia es llamada `view`, usted tenga en cuenta que en las viewlets Grok se llama `viewlets` (en ese caso `viewlet/lanyrd_link`).

Vamos a agregar el archivo de la plantilla restante `templates/social_viewlet.pt`.

```
<div id="social-links">
    <a href="#"
       class="lanyrd-link"
       tal:define="link view/lanyrd_link"
       tal:condition="link"
       tal:attributes="href link">
         See this talk on Lanyrd!
    </a>
</div>
```

Como se puede ver esto no es un documento válido HTML. Eso no es necesario, porque no queremos una `view` completa aquí, sólo un fragmento de código HTML.

Hay una sentencia `tal:define`, para la consulta *view/lanyrd_link*. Al igual en la page templates de la plantilla tiene acceso a su clase.

Entonces, ahora tenemos que ampliar el Viewlet Social para añadir el atributo que le falta, agregando el siguiente código Python:

¿Por qué no acceder directamente contexto?

En este ejemplo, `ISocial(self.context)` devuelve el contexto directamente. Todavía es bueno utilizar este idioma por dos razones:

1. Se deja en claro, que sólo queremos usar el aspecto ISocial del objeto.
2. Si decidimos utilizar una factory, por ejemplo, para almacenar nuestros atributos en una annotation, nosotros deberíamos *no* conseguir nuestro contexto, pero si el adaptador.

Por lo tanto, en este ejemplo, usted puede simplemente escribir `return self.context.lanyrd`.

```
from plone.app.layout.viewlets import ViewletBase
from ploneconf.site.behaviors.social import ISocial

class SocialViewlet(ViewletBase):

    def lanyrd_link(self):
        adapted = ISocial(self.context)
        return adapted.lanyrd
```

Hasta ahora, hemos hecho:

- Registrar el viewlet al contenido que tiene la interfaz ISocial.
- Adaptar el objeto a su comportamiento para ser capaz de acceder a los campos del comportamiento.
- Devolver el enlace.

Ejercicio 1

Registrar una viewlet `number_of_talks` en el pie de página que es visible sólo para los administradores del sitio (el permiso que usted busca es `cmf.ManagePortal`). Utilice sólo una plantilla (sin clase) para mostrar el número de charlas ya enviadas. Sugerencia: Utilice Aquisition para obtener al catálogo (Usted sabe, no debe hacer esto, pero hay un montón de código por ahí que lo hace...)

Solución

Registrar el viewlet en el archivo `browser/configure.zcml`, agregando el siguiente código ZCML:

```
<browser:viewlet
  name="number_of_talks"
  for="*"
  manager="plone.app.layout.viewlets.interfaces.IPortalFooter"
  layer="zope.interface.Interface"
  template="templates/number_of_talks.pt"
  permission="cmf.ManagePortal"
  />
```

Para los parámetros `for` y `layer` el `*` es la abreviatura de `zope.interface.Interface` y el mismo efecto que la omisión de ellos: El viewlet se mostrará para todos los tipos de páginas y para todos los sitios de Plone en su instancia Zope.

Agregar el archivo de la plantilla `browser/templates/number_of_talks.pt`, agregando el siguiente código HTML:

```
<div class="number_of_talks"
     tal:define="catalog python:context.portal_catalog;
                 talks python:len(catalog(portal_type='talk'));">
    There are <span tal:replace="talks" /> talks.
</div>
```

`python:context.portal_catalog` devolverá el catálogo a través de la adquisición. Tenga cuidado si usted desea usar expresiones de rutas: `content/portal_catalog` llama el catálogo (y devuelve todos los cerebros). Es necesario para evitar esto usando `nocall:content/portal_catalog`.

Basándose en Adquisición es una mala idea. Sería mucho mejor utilizar la helper view llamada `plone_tools` desde el módulo `plone/app/layout/globals/tools.py` para obtener el catálogo.

```
<div class="number_of_talks"
     tal:define="catalog context/@@plone_tools/catalog;
                 talks python:len(catalog(portal_type='talk'));">
    There are <span tal:replace="talks" /> talks.
</div>
```

`context/@@plone_tools/catalog` atraviesa a la vista `plone_tools` y llama a su método `catalog`. En expresiones Python debería verse así:

```
<div class="number_of_talks"
     tal:define="catalog python:context.restrictedTraverse('plone_tools').catalog();
                 talks python:len(catalog(portal_type='talk'));">
    There are <span tal:replace="talks" /> talks.
</div>
```

No es una buena práctica consultar el catálogo dentro de una plantilla, ya que incluso la simple lógica como esta debería vivir en Python. Pero es muy poderoso si está depurando o necesita una solución rápida y sucia.

Ejercicio 2

Registrar una viewlet `days_to_conference` en la cabecera. Utilice una clase y una plantilla para mostrar el número de días hasta la conferencia. Usted consigue muchos puntos de bonificación si se muestra en un formato agradable (pensar “En 2 días” y “Último mes”) utilizando un Javascript existentes o biblioteca Python.

CAPÍTULO 22

Programando en Plone

plone.api

La herramienta mas importante actualmente para desarrolladores Plone es el complemento plone.api , el cual cubre el 20 % de las tareas que cualquier desarrollador Plone que normalmente hace en 80 % de su tiempo. Si no estas seguro de como manejar alguna tarea, verifica primero si plone.api tiene una solución para ti.

El api esta divido en 5 secciones. Este es un ejemplo de cada uno:

- *Contenido*: Crear contenido
- *Portal*: Enviar E-Mail
- *Grupos*: Asignar roles al grupo
- *Usuarios*: Obtener roles de usuario
- *Entorno*: Cambiar roles dentro de un bloque

plone.api aun no es parte del núcleo de Plone, Por lo tanto, no veras ningún uso de plone.api dentro de Plone. Pero estamos seguros que sera parte de Plone.

En el código existente, a veces encontraras métodos que no significan nada para usted. Usted tiene que usar la fuente para averiguar que hacen.

Algunos de estos siguientes métodos serán reemplazados por plone.api en el futuro:

- `Products.CMFCore.utils.getToolByName` -> `api.portal.get_tool`
- `zope.component.getMultiAdapter` -> `api.content.get_view`

Herramientas del portal

Algunas partes de Plone son módulos muy complejos entre si (ej. el blanco mecanismo de versionado de `Products.CMFEditions`). Algunos de ellos tienen una api que usted aprenderá a usar tarde o temprano.

Aquí tienes unos ejemplos:

portal_catalog `unrestrictedSearchResults()` regresa los resultados de búsqueda sin verificar si el usuario actual tiene permiso de acceso a los objetos.

`uniqueValuesFor()` regresa todas las entradas en un índice

portal_setup `runAllExportSteps()` genera un archivo comprimido en formato Tar conteniendo los artefactos de todos los pasos de exportación.

portal_quickinstaller `isProductInstalled()` verifica si un producto esta instalado.

Usualmente la mejor forma para aprender acerca de la herramienta de un api es mirando en el archivo `interfaces.py` en el respectivo paquete y leer la documentación incluida en formato docstrings.

Depurando

Aquí hay algunas herramientas y técnicas que usamos frecuentemente cuando desarrollamos y depuramos. Usamos algunas de ellas en varias situaciones durante el entrenamiento.

Rastreos y el registro El log (y la consola cuando ejecutamos en modo `foreground`) recolecta todos los mensajes de log que Plone imprime. Cuando ocurre una excepción Plone lanza un rastreo. La mayoría del tiempo el rastreo es todo lo que necesitas para averiguar que esta sucediendo. También agregar tu propia información de rastreo al log es muy simple.

pdb El depurador de Python `pdb`, es la herramienta simple mas importante para nosotros cuando programemos. ¡Solo agrega `import pdb; pdb.set_trace()` en tu código y empieza a depurar!

ipdb Mejorar `pdb` con el poder de IPython, por ejemplo, la implementación del tabulador, resaltado de sintaxis, mejores rastreos y la introspección. También funciona muy bien con el complemento `Products.PDBDebugMode`.

Products.PDBDebugMode Un complemento que tiene dos características esenciales para su uso.

Post-mortem debugging: le lanza en un `pdb` cuando ocurra una excepción. de esta forma puedes averiguar que es lo que esta mal.

pdb-view: simplemente agregando `/pdb` a una dirección URL, le lanza a una sesión pdb con el contexto actual como `self.context`. De allí puedes hacer casi todo.

Modo Depuración Cuando se inicia Plone usando el comando `./bin/instance debug -O Plone` usted terminara en una sesión de comando del depurador Interactivo.

plone.app.debugtoolbar Un complemento que permite inspeccionar todo muy cercanamente. Incluso tiene una consola interactiva y un probador para expresiones TALES.

plone.reload Un complemento que permite recargar el código que has cambiado sin reiniciar el sitio. Esto es también es usado por el complemento `plone.app.debugtoolbar`.

Products.PrintingMailHost Un complemento que previene a Plone enviar mensajes. Los mensajes son registrados como eventos en el log del servicio.

Products.Ienablesettrace Complemento que permite usar `pdb` y `ipdb` en scripts python skin. Eso es muy útil.

`verbose-security = on` Una opción para la receta *plone.recipe.zope2instance* que hace a los logs de las razones detalladas sobre por que un usuario no puede ser autorizado a ver algo.

`./bin/buildout annotate` Una opción que cuando se ejecute buildout realiza un log de los paquetes traídos y sus versiones.

Sentry Sentry es una aplicación de log de errores que puede usar usted mismo. Agrega rastreos de muchas fuentes y (aquí viene la característica esencial para su uso) incluso los valores de las variables en el Stack trace. Lo usamos en todos nuestro sitios en producción.

zopepy Buildout puede crear un shell script de Python para ti que tiene los paquetes de su sitio Plone en su PYTHONPATH. Puede agrega la parte Buildout a la suya con la siguiente configuración:

```
[zopepy]
recipe = zc.recipe.egg
eggs = ${instance:eggs}
interpreter = zopepy
```

CAPÍTULO 23

IDEs y editores

¡Plone consiste de más de 20.000 archivos! Usted necesita una herramienta para administrarlas. Ningún entorno de desarrollo esta completo sin un buen editor.

La gente elige sus propios editores. Utilice cualquiera con el que se sienta cómodo y productivo. Estos son los editores más utilizados en la comunidad Plone:

- Sublime.
- PyCharm.
- Wing IDE.
- PyDev para Eclipse
- Aptana Studio.
- vim.
- emacs.
- Textmate.

Algunas características que la mayoría de los editores tienen de una forma u otra son esenciales desarrollando en Plone.

- **Buscar en Proyecto** (SublimeText 3: `cmd + shift + f`)
- **Buscar archivos en Proyecto** (SublimeText 3: `cmd + t`)
- **Buscar clases y métodos en Proyecto** (SublimeText 3: `cmd + shift + r`)
- **Ir a Definición** (SublimeText3 con codeintel: `alt + click`)
- **Poderoso mecanismo de búsqueda y remplazo**

La capacidad de realizar una *búsqueda a texto completo* a través del código completo de Plone es inestimable. gracias a la receta `omelette`, con una unidad de estado sólido y mucha RAM puedes buscar en el código de Plone en 3 segundos.

Nota: Algunos IDEs y editores pueden ser extendidos para cumplir completamente con todas sus características. Aquí hay algunos paquetes recomendados cuando se usa Sublime Text 3:

- SublimeCodeIntel (Ir a Definición)
- BracketHighlighter
- GitGutter
- FileDiffs

- SublimeLinter con SublimeLinter-flake8 ...
- INI (sintaxis para archivos ini)
- SideBarEnhancements
- MacTerminal
- SyncedSideBar

CAPÍTULO 24

Tipos Dexterity - Parte II: Creciendo

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/23_dexterity_2/ src/ploneconf.site
```

Los tipos de contenidos talks existentes todavía le faltan algunas funcionalidades que queremos utilizar.

En esta parte vamos a tratar:

- Añadir una interfaz marker para nuestro tipo de contenido talk.
- Crear catálogo de índices personalizados.
- Hacer consultas al catálogo a estos.
- Permitir a algunas características más por defecto para nuestro tipo de contenido.

Agregar una interfaz marker para el tipo de contenido talk

Interfaces Marcador

El tipo de contenido *Talk* no es aun ciudadano de primera clase porque no implementa su propia interfaz. Las interfaces son como etiquetas de nombre, diciendo a otros elementos quien y que eres y que puedes hacer. Una interfaz marker es como una etiqueta. Los tipo de contenido talks actualmente tienen una interfaz marker auto generado en `plone.dexterity.schema.generated.Plone_0_talk`.

El problema es que el nombre de la instancia `Plone` es parte del nombre de la interfaz. si tu ahora mueves estos tipos a un sitio con otro nombre, el código que usa en estas interfaces no podrán ser capaces de encontrar estos objetos.

Para solventar esto agregamos un nuevo archivo `interfaces.py`:

```
1 from zope.interface import Interface
2
3
4 class ITalk(Interface):
5     """Marker interface for Talks
6     """
```

`ITalk` es una interfaz marker. Podemos anexar Vistas y Viewlets al contenido que provee estas interfaces. Ahora veremos como proveer esta interfaz. Hay dos soluciones para esto.

1. Dejemos que sean las instancias de una clase que implementa esta interfaz.
2. Registra esta interfaz como un comportamiento y habilite este en el tipo de contenido Talk.

La primera opción tiene un inconveniente importante: Sólo nuevas charlas pueden ser instancias de una nueva clase. Nos gustaría o podríamos tal vez tener que migrar a las charlas existentes o eliminarlas.

Así que vamos a registrar la interfaz como un comportamiento en el archivo `behaviors/configure.zcml`

```
<plone:behavior
  title="Talk"
  description="Marker interface for talks to be able to bind views to."
  provides="..interfaces.ITalk"
  />
```

Y habilite ese comportamiento en la definición de tipo de contenido en su configuración GenericSetup en el archivo `profiles/default/types/talk.xml`

```
<property name="behaviors">
 <element value="plone.app.dexterity.behaviors.metadata.IDublinCore"/>
 <element value="plone.app.content.interfaces.INameFromTitle"/>
 <element value="ploneconf.site.behaviors.social.ISocial"/>
 <element value="ploneconf.site.interfaces.ITalk"/>
</property>
```

Vuelva a instalar el complemento, aplicar el comportamiento a mano o ejecutar una paso de actualización GenericSetup (ver más abajo) y la interfaz estará allí.

Entonces podemos con seguridad unir la vista `talkview` a la nueva interfaz marker, agregando el siguiente código ZCML:

```
<browser:page
  name="talkview"
  for="ploneconf.site.interfaces.ITalk"
  layer="zope.interface.Interface"
  class=".views.TalkView"
  template="templates/talkview.pt"
  permission="zope2.View"
  />
```

Ahora la vista `/talkview` puede solamente se usada en objetos que implementan dicha interfaz. Ahora también podemos consultar el catálogo para los objetos que proporcionan esta interfaz `catalog(object_provides="ploneconf.site.interfaces.ITalk")`. Las vistas *talklistview* y *demoview* no reciben esta restricción, ya que no sólo son utilizados en los tipos de contenidos talks.

Nota: Sólo para completarlo, esto es lo que tendría que pasar por la primera opción:

- Crear una nueva clase que herede desde `plone.dexterity.content.Container` e implementa la interfaz marker.

```
from plone.dexterity.content import Container
from ploneconf.site.interfaces import ITalk
from zope.interface import implements

class Talk(Container):
    implements(ITalk)
```

- Modificar la clase para los nuevos tipos de contenidos talks en el archivo `profiles/default/types/talk.xml`

```
...
<property name="add_permission">cmf.AddPortalContent</property>
<property name="klass">ploneconf.site.content.talk.Talk</property>
<property name="behaviors">
...
```

- Crear un paso de actualización para modificar la clase de tipo de contenido existente. Un código ejemplo de como hacer esto lo encuentra en el paquete ftw.upgrade.

Pasos de actualización

Cuando los proyectos se desarrollan a veces tendrá que modificar varias cosas mientras que el sitio ya este arriba, lleno de contenido y usuarios. Los pasos de actualización son piezas de código que se ejecutan al actualizar de una versión de un complemento a una más reciente. Ellos pueden hacer casi cualquier cosa.

Nosotros crearemos un paso de actualización que:

- Ejecute el paso `typeinfo` (es decir, carga los almacenes de configuración GenericSetup en `profiles/default/types.xml` y `profiles/default/types/...` por lo que no tiene que volver a instalar el complemento para tener nuestros cambios hechos) y
- haga una limpieza de algún contenido que pudiera estar esparcidos alrededor del sitio en las primeras etapas de su creación. Nos movemos todas los tipos de contenidos talks en una carpeta llamadas `talks` (a menos que ya están allí).

Los pasos de actualización son usualmente registrados en en su propio archivo ZCML. Cree el archivo `upgrades.zcml`, agregando el siguiente código ZCML:

```
<configure
  xmlns="http://namespaces.zope.org/zope"
  xmlns:i18n="http://namespaces.zope.org/i18n"
  xmlns:genericsetup="http://namespaces.zope.org/genericsetup"
  i18n_domain="ploneconf.site">

  <genericsetup:upgradeStep
    title="Update and cleanup talks"
    description="Updates typeinfo and moves talks to a folder 'talks'"
    source="1"
    destination="1001"
    handler="ploneconf.site.upgrades.upgrade_site"
    sortkey="1"
    profile="ploneconf.site:default"
    />

</configure>
```

El paso de actualización cambia la versión número del perfil GenericSetup del complemento `ploneconf.site` de 1 a 1001. La versión se almacena en `profiles/default/metadata.xml`. Cámbielo a:

```
<version>1001</version>
```

Incluir el nuevo archivo `upgrades.zcml` en nuestro archivo `configure.zcml`, agregando el siguiente código ZCML:

```
<include file="upgrades.zcml" />
```

Ahora GenericSetup espera el código como un método `upgrade_site` en el archivo `upgrades.py`. Vamos a crearlo, agregando el siguiente código Python:

```
from plone import api
import logging

default_profile = 'profile-ploneconf.site:default'
logger = logging.getLogger('ploneconf.site')


def upgrade_site(setup):
    setup.runImportStepFromProfile(default_profile, 'typeinfo')
    catalog = api.portal.get_tool('portal_catalog')
    portal = api.portal.get()
    if 'the-event' not in portal:
        theevent = api.content.create(
            container=portal,
            type='Folder',
            id='the-event',
            title='The event')
    else:
        theevent = portal['the-event']
    if 'talks' not in theevent:
        talks = api.content.create(
            container=theevent,
            type='Folder',
            id='talks',
            title='Talks')
    else:
        talks = theevent['talks']
    talks_url = talks.absolute_url()
    brains = catalog(portal_type='talk')
    for brain in brains:
        if talks_url in brain.getURL():
            continue
        obj = brain.getObject()
        logger.info('Moving %s to %s' % (obj.absolute_url(), talks.absolute_url()))
        api.content.move(
            source=obj,
            target=talks,
            safe_id=True)
```

Después de iniciar de nuevo el sitio, nosotros podemos ejecutar el paso de actualización:

- Valla al panel control Complementos en la dirección URL http://localhost:8080/Plone/prefs_install_products_form. Ahora debe haber una advertencia **This add-on has been upgraded. Old profile version was 1. New profile version is 1001** y un botón al lado de él.
- Ejecutar el paso de actualización haciendo clic en el.

En la consola debería ver mensajes de registro como este:

```
INFO ploneconf.site Moving http://localhost:8080/Plone/old-talk1 to http://localhost:8080/Plone/the-e
```

Alternativamente, usted puede seleccionar los pasos de actualización para ejecutarlo de la siguiente manera:

- En la ZMI valla a la herramienta *portal_setup*
- Valla a la pestaña *Upgrades*
- Seleccione *ploneconf.site* desde la lista desplegable y haga clic *Choose profile*

- Ejecutar el paso de actualización.

Ver también:

http://docs.plone.org/4/en/develop/addons/components/genericsetup.html#id1

Nota: Actualizando desde una versión anterior de Plone a uno más reciente también puede ejecutar los pasos de actualización del paquete `plone.app.upgrade`. Usted debe ser capaz de actualizar un sitio limpio de Plone 2.5 a Plone 5.0a2 con un solo clic.

Para un ejemplo ver el paso de actualización para Plone 5.0a1 https://github.com/plone/plone.app.upgrade/blob/master/plone/app/upgrade/v50/alphas.py#L23

Agregar una browserlayer

Una BrowserLayer es otro ejemplo de la interfaz marker. Las BrowserLayers nos permiten fácilmente habilitar y deshabilitar opiniones y funcionalidad de otro sitio basado en complementos y temas instalados.

Como queremos que las características que escribimos sólo estén disponible cuando el paquete `ploneconf.site` actualmente este instalado podemos asociarlos a una BrowserLayer.

En el archivo `interfaces.py` nosotros agregamos el siguiente código Python:

```
class IPloneconfSiteLayer(Interface):
    """Marker interface for the Browserlayer
    """
```

Nosotros registramos la BrowserLayer en GenericSetup en el archivo `profiles/default/browserlayer.xml`

```
<?xml version="1.0"?>
<layers>
  <layer name="ploneconf.site"
    interface="ploneconf.site.interfaces.IPloneconfSiteLayer" />
</layers>
```

Después de volver a instalar el complemento podemos obligar a las vistas `talkview`, `demoview` y `talklistview` a que sean parte de nuestra layer. Aquí hay un ejemplo usando la vista `talkview`.

```
<browser:page
  name="talklistview"
  for="ploneconf.site.interfaces.ITalk"
  layer="..interfaces.IPloneconfSiteLayer"
  class=".views.TalkListView"
  template="templates/talklistview.pt"
  permission="zope2.View"
  />
```

Tenga en cuenta el ruta relativa Python `..interfaces.IPloneconfSiteLayer`. Es equivalente a la ruta absoluta `ploneconf.site.interfaces.IPloneconfSiteLayer`.

Ver también:

http://docs.plone.org/4/en/develop/plone/views/layers.html

Ejercicio

¿Es necesario vincular el viewlet social desde el capítulo 20 de este nuevo Browser-Layer?

Solución

No, no haría ninguna diferencia desde el viewlet ya está asociada a la interfaz marker `ploneconf.site.behaviors.social.ISocial`.

Agregar índices del catálogo

En la vista *talklistview* nosotros tuvimos que despertar todos los objetos para acceder a algunos de sus atributos. Eso está bien si no tenemos muchos objetos y ellos son objetos Dexterity ligeros. Si tuviéramos miles de objetos que esto podría no ser una buena idea.

En lugar de cargar a todos en la memoria usaremos índices del catálogo para obtener los datos que queremos mostrar.

Agregar un nuevo archivo `profiles/default/catalog.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object name="portal_catalog">
  <index name="type_of_talk" meta_type="FieldIndex">
    <indexed_attr value="type_of_talk"/>
  </index>
  <index name="speaker" meta_type="FieldIndex">
    <indexed_attr value="speaker"/>
  </index>
  <index name="audience" meta_type="KeywordIndex">
    <indexed_attr value="audience"/>
  </index>

  <column value="audience" />
  <column value="type_of_talk" />
  <column value="speaker" />
</object>
```

Esto agrega los nuevos índices para los tres campos que queremos mostrar en el listado. No es que *audience* es un `KeywordIndex` porque el campo es de varios valores, pero queremos una entrada de índice independiente para cada valor a un objeto.

La entrada `column ..` nos permite visualizar estos valores de estos índices en la vista `tableview` de colecciones.

Nota: Hasta Plone 4.3.2 agregar índices en el archivo `catalog.xml` era perjudicial porque ¡al volver a instalar el complemento purgaba los índices! Ver mas detalles en la dirección URL https://www.starzel.de/blog/a-reminder-about-catalog-indexes.

Para ejecutar código personalizado adicional al volver instalar un complemento que puedes usar un archivo setuphandler.py.

- Volver a instalar el complemento
- Valla a la URL http://localhost:8080/Plone/portal_catalog/manage_catalogIndexes para inspeccionar el ingreso de datos e inspeccionar los nuevos índices al catálogo

Ver también:

http://docs.plone.org/4/en/develop/plone/searching_and_indexing/indexing.html

Consulta para índices personalizados

Los nuevos índices se comportan como los que Plone tiene construido en:

```
>>> (Pdb) from Products.CMFCore.utils import getToolByName
>>> (Pdb) catalog = getToolByName(self.context, 'portal_catalog')
>>> (Pdb) catalog(type_of_talk='Keynote')
[<Products.ZCatalog.Catalog.mybrains object at 0x10737b9a8>, <Products.ZCatalog.Catalog.mybrains obje
>>> (Pdb) catalog(audience=('Advanced', 'Professionals'))
[<Products.ZCatalog.Catalog.mybrains object at 0x10737b870>, <Products.ZCatalog.Catalog.mybrains obje
>>> (Pdb) brain = catalog(type_of_talk='Keynote')[0]
>>> (Pdb) brain.speaker
u'David Glick'
```

Ahora podemos usar los nuevos índices para mejorar la vista `talklistview` así que no tenemos que despertar los objetos nunca más.

```
class TalkListView(BrowserView):
    """ A list of talks
    """

    def talks(self):
        results = []
        portal_catalog = getToolByName(self.context, 'portal_catalog')
        current_path = "/".join(self.context.getPhysicalPath())

        brains = portal_catalog(portal_type="talk",
                                path=current_path)
        for brain in brains:
            results.append({
                'title': brain.Title,
                'description': brain.Description,
                'url': brain.getURL(),
                'audience': ', '.join(brain.audience or []),
                'type_of_talk': brain.type_of_talk,
                'speaker': brain.speaker,
                'uuid': brain.UID,
                })
        return results
```

La plantilla no necesita ser cambiado y el resultado no cambió también.

Agregar criterio de colección

Para ser capaz de buscar contenido en tipo de contenidos *Colecciones* utilizando los nuevos índices tendríamos que registrarlos como criterios para el widget de cadena de consulta que usan las *Colecciones*.

Agregar un nuevo archivo `profiles/default/registry.xml`, agregando el siguiente código XML:

```
<registry>
  <records interface="plone.app.querystring.interfaces.IQueryField"
           prefix="plone.app.querystring.field.audience">
    <value key="title">Audience</value>
    <value key="description">A custom speaker index</value>
    <value key="enabled">True</value>
    <value key="sortable">False</value>
    <value key="operations">
```

```
      <element>plone.app.querystring.operation.string.is</element>
    </value>
    <value key="group">Metadata</value>
  </records>
  <records interface="plone.app.querystring.interfaces.IQueryField"
           prefix="plone.app.querystring.field.type_of_talk">
    <value key="title">Type of Talk</value>
    <value key="description">A custom index</value>
    <value key="enabled">True</value>
    <value key="sortable">False</value>
    <value key="operations">
      <element>plone.app.querystring.operation.string.is</element>
    </value>
    <value key="group">Metadata</value>
  </records>
</registry>
```

Ver también:

http://docs.plone.org/4/en/develop/plone/functionality/collections.html#add-new-collection-criteria-new-style-plone-app-collection-ins

Añadir más características a través de GenericSetup

Habilitar el control de versiones y una visión del diff para los tipos de contenidos talks a través de GenericSetup.

Agregar nuevo archivo `profiles/default/repositorytool.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<repositorytool>
  <policymap>
    <type name="talk">
      <policy name="at_edit_autoversion"/>
      <policy name="version_on_revert"/>
    </type>
  </policymap>
</repositorytool>
```

Agregar nuevo archivo `profiles/default/diff_tool.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object>
  <difftypes>
    <type portal_type="talk">
      <field name="any" difftype="Compound Diff for Dexterity types"/>
    </type>
  </difftypes>
</object>
```

CAPÍTULO 25

Búsqueda personalizada

Podemos usar los índices creados en el último capítulo para mejorar la lista de tipos de contenidos talks.

Si los capítulos acerca de las vistas le parece complejo, ahora mostraremos el complemento de búsqueda personalizado como una gran alternativa hasta que se sientas cómodo escribiendo vistas y plantillas. También hay muchas herramientas que permitan agregar impresionantes búsquedas personalizadas y listados de contenidos a través de la Web en Plone.

eea.facetednavigation

- Instale el complemento eea.facetednavigation.
- Habilita navegación en facetas en una nueva carpeta "Descubriendo charlas" haciendo clic en el menú de edición *Acciones* > *Enable faceted navigation*.
- Haga clic en la pestaña *Faceted criteria* para configurarla.
  - Selecciona "Talk" para *Portal type*, oculta *Results per page*.
  - Agregue un widget de selección a la izquierda y usa el índice de catalogo *Audience* para eso.
  - Agrega un widget de selección para `speaker`.
  - Agregue un widget de radio para `type_of_talk`.
  - Otros widgets notables son: `tagcloud`, `a-z`, `search`.

Ejemplos:

- http://www.dipf.de/en/research/projects
- https://mountaineers.org/learn/find-courses-clinics-seminars
- http://www.dynajet.de/en/hochdruckreiniger

Ver también:

Usaremos el nuevo índice de catálogo para proveer la data de los widgets y buscar los resultados. Para otros casos de uso podremos usar entre los vocabularios integrados (https://pypi.python.org/pypi/plone.app.vocabularies) o crear vocabularios personalizados para este.

- Vocabularios personalizados a través de la Web usando Products.ATVocabularyManager
- Programando usando vocabularios: http://docs.plone.org/4/en/external/plone.app.dexterity/docs/advanced/vocabularies.html

CAPÍTULO 26

Convertir charlas en eventos

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/25_events/ src/ploneconf.site
```

Se nos olvidó algo: Una lista de los tipos de contenidos talks es especialmente excelente si usted puede ordenarlo por sus preferencias. Pero si un visitante decide que quiere ver una charla que necesita saber cuándo va a tener lugar.

Necesitamos un programa y para ello necesitamos almacenar la información de cuando una charla va a suceder.

Afortunadamente el tipo *Evento* es basado en comportamiento reusable desde el paquete plone.app.event.

En este capítulo vamos a tratar:

- Habilitar este comportamiento para las charlas.
- Escribir un paso de actualización (upgrade step) para evitar un error en el paquete `plone.app.event`.
- Mostrar la fecha en la vista `talkview`.

Primero nosotros habilitamos el comportamiento `IEventBasic` para los tipos de contenidos talksen el archivo `profiles/default/types/talk.xml`, agregando el siguiente código XML:

```
<property name="behaviors">
  <element value="plone.app.dexterity.behaviors.metadata.IDublinCore"/>
  <element value="plone.app.content.interfaces.INameFromTitle"/>
  <element value="ploneconf.site.behavior.social.ISocial"/>
  <element value="ploneconf.site.interfaces.ITalk"/>
  <element value="plone.app.event.dx.behaviors.IEventBasic"/>
</property>
```

Después activamos el comportamiento manualmente o vuelta a instalar el complemento, podríamos añadir nuevos tipos de contenidos `talks` con el nuevo campo de `start` y `end`.

Pero debido a un error en plone.app.event no se espera que los objetos existentes obtengan el comportamiento. Desde los tipos de contenidos existentes no tienen valores en los campos `start` y `end` nos encontraremos con un `traceback` en línea de validación cuando editamos estos campos. Para solucionar esto creamos un paso de actualización que establece alguna una fecha inicial.

Registrar el nuevo paso de actualización (upgrade step) en el archivo `upgrades.zcml`, agregando el siguiente código ZCML:

```
<genericsetup:upgradeStep
  title="Add event-behavior to talks"
  description=""
  source="1001"
  destination="1002"
  handler="ploneconf.site.upgrades.turn_talks_to_events"
  sortkey="1"
  profile="ploneconf.site:default"
  />
```

Cambiar la versión del perfil a 1002 en el archivo `profiles/default/metadata.xml`

Escribir el paso de actualización (upgrade sted) en el archivo `upgrades.py`, agregando el siguiente código Python:

```
# -*- coding: UTF-8 -*-
from plone import api

import datetime
import logging
import pytz


default_profile = 'profile-ploneconf.site:default'

logger = logging.getLogger('ploneconf.site')

def upgrade_site(setup):
    setup.runImportStepFromProfile(default_profile, 'typeinfo')
    catalog = api.portal.get_tool('portal_catalog')
    portal = api.portal.get()
    if 'the-event' not in portal:
        theevent = api.content.create(
            container=portal,
            type='Folder',
            id='the-event',
            title='The event')
    else:
        theevent = portal['the-event']
    if 'talks' not in theevent:
        talks = api.content.create(
            container=theevent,
            type='Folder',
            id='talks',
            title='Talks')
    else:
        talks = theevent['talks']
    talks_url = talks.absolute_url()
    brains = catalog(portal_type='talk')
    for brain in brains:
        if talks_url in brain.getURL():
            continue
        obj = brain.getObject()
        logger.info('Moving %s to %s' % (obj.absolute_url(), talks.absolute_url()))
        api.content.move(
            source=obj,
            target=talks,
            safe_id=True)

def turn_talks_to_events(setup):
```

```
    """Set a start- and end-date for old events to work around a
    bug in plone.app.event 1.1.1
    """
    api.portal.set_registry_record(
        'plone.app.event.portal_timezone',
        'Europe/London')
    setup.runImportStepFromProfile(default_profile, 'typeinfo')

    tz = pytz.timezone("Europe/London")
    now = tz.localize(datetime.datetime.now())
    date = now + datetime.timedelta(days=30)
    date = date.replace(minute=0, second=0, microsecond=0)

    catalog = api.portal.get_tool('portal_catalog')
    brains = catalog(portal_type='talk')
    for brain in brains:
        obj = brain.getObject()
        if not getattr(obj, 'start', False):
            obj.start = obj.end = date
            obj.timezone = "Europe/London"
```

Después que ejecutamos el paso de actualización ahora podemos añadir un tiempo para los existentes eventos. Para mostrar estos reutilizamos una vista de resumen para los eventos por defecto como se documenta en http://ploneappevent.readthedocs.io/en/latest/development.html#reusing-the-event-summary-view-to-list-basic-event-information

Edite el archivo `browser/templates/talkview.pt`, agregando el siguiente código ZPT:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="ploneconf.site">
<body>
    <metal:content-core fill-slot="content-core" tal:define="widgets view/w">

        <tal:eventsummary replace="structure context/@@event_summary"/>

        <p>
            <span tal:content="context/type_of_talk">
                Talk
            </span>
            suitable for
            <span tal:replace="structure widgets/audience/render">
                Audience
            </span>
        </p>

        <div tal:content="structure widgets/details/render">
            Details
        </div>

        <div class="newsImageContainer">
            <img tal:condition="python:getattr(context, 'image', None)"
                 tal:attributes="src string:${context/absolute_url}/@@images/image/thumb" />
        </div>

        <div>
            <a class="email-link" tal:attributes="href string:mailto:${context/email}">
                <strong tal:content="context/speaker">
                    Jane Doe
```

```
32                  </strong>
33              </a>
34              <div tal:content="structure widgets/speaker_biography/render">
35                  Biography
36              </div>
37          </div>
38
39      </metal:content-core>
40  </body>
41  </html>
```

CAPÍTULO 27

Usar contenido generado

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/26_user_generated_content/ src/ploneconf.site
```

¿Cómo hacer que los potenciales ponentes envíen sus charlas? Dejamos que se registren en el sitio y se le otorgan derecho a crear tipos de contenidos Talk. Para ello volvemos a cambiar el sitio a través de la Web.

En este capítulo vamos a:

- Permitir auto registro de usuario.
- Restringir tipos de contenidos en la carpeta `talks` de las charlas.
- Otorgar roles locales.
- Crear un flujo de trabajo para los tipos de contenidos Talk.

Auto-registro

- Valla al panel de control de Seguridad en la dirección URL http://localhost:8080/Plone/@@security-controlpanel y Habilitar auto-registro
- Deja "Habilitar carpetas de usuario" apagado a menos que quieras un sitio comunitario.

Restringir tipos de contenidos

- En la carpeta `talks` seleccione Restricciones... en el menú *Añadir nuevo...*. Solo permita agregar tipos de contenidos `Talks`.

Otorgar roles locales

- Valla a la pestaña *Compartir* y otorgar el rol *Puede agregar* al grupo Usuarios conectados. Ahora cualquier usuario puede agregar contenido en esta carpeta (y solamente en esta carpeta).

Ahora todos los Usuarios conectados pueden crear y enviar charlas en esta carpeta con el permiso del flujo de trabajo por defecto.

Un flujo de trabajo personalizado para las charlas

Todavía tenemos que solucionar un problema: los usuarios autenticados pueden ver todos los tipos de contenidos `talks`, incluso los de otros usuarios en el estado privado. Ya que no queremos que esto vamos a crear un flujo de trabajo modificado para los tipos de contenidos `talks`. El nuevo flujo de trabajo sólo les permitirá ver y editar los tipos de contenidos `talks` que han creado ellos mismos y no los de otros usuarios.

- Valla a la herramienta `ZMI` > `portal_workflow`
- Ver como los tipos de contenidos `talks` tiene el mismo flujo de trabajo como la mayoría del contenido `(Por defecto)`
- Valla a la pestaña *Contents*, marque la casilla próxima a `simple_publication_workflow`, haga clic en `Copy` y `Paste`.
- Renombrar el nuevo flujo de trabajo desde *copy_of_simple_publication_workflow* a *talks_workflow*.
- Edite el flujo de trabajo haciendo clic en: Change the Title to *Talks Workflow*.
- Haga clic en la pestaña *States* y haga clic en *private* y edite este estado. En la próximo vista seleccione en la pestaña *Permissions*.
- Encuentra en la columna de la tabla para el rol *Colaborador* y quitar los permisos para `Access contents information` y `View`. Tenga en cuenta que el rol *Propietario* (es decir, el rol Creador) todavía tiene algunos permisos.
- Hacer lo mismo para el estado *pending*.
- Valla a la herramienta *portal_workflow* y establecer el nuevo flujo de trabajo `talks_workflow` para los tipos de contenidos `talks`. Haga clic *Change* y entonces *Update security settings*.

Nota: El complemento plone.app.workflowmanager provee una más agradable de interfaz de usuario para esto. El problema es que necesita una pantalla grande para él y que puede ser muy confuso también.

Hecho.

Mover los cambios el sistema de archivo.

No queremos hacer estos pasos para cada nueva conferencia manualmente para que nos movemos los cambios en nuestro paquete.

Flujo de trabajo

- Exporte el paso GenericSetup *Workflow Tool* en la siguiente dirección URL http://localhost:8080/Plone/portal_setup/manage_exportSteps.
- Lance el archivo `workflows.xml` dentro del directorio `profiles/default`.
- Lance el archivo `workflows/talks_workflow/definition.xml` en `profiles/default/workflows/talks_workflow/definition.xml`. Los otros son sólo definiciones de los flujos de trabajo por defecto y sólo queremos cosas en nuestro paquete que cambia Plone.

Auto-registro

Esto tiene que suceder en Python en una archivo setuphandler.py personalizable, ya que aún no existe un entorno exportable para esto.

Registrar un paso de importación en el archivo `configure.zcml`. Será ejecutará automáticamente al volver a instalar el complemento.

```
<genericsetup:importStep
  name="ploneconf.site"
  title="ploneconf.site special import handlers"
  description=""
  handler="ploneconf.site.setuphandlers.setupVarious">
    <depends name="typeinfo"/>
</genericsetup:importStep>
```

Tenga en cuenta que el módulo `setuphandler` tiene una dependencia en *typeinfo* porque sólo permitirá la creación de tipos de contenidos `talks`. Para este tipo ya tiene que existir.

Crear un nuevo archivo `setuphandlers.py`, agregando el siguiente código Python:

```
# -*- coding: UTF-8 -*-
from plone.app.controlpanel.security import ISecuritySchema
from plone import api

import logging

PROFILE_ID = 'profile-ploneconf.site:default'
logger = logging.getLogger('ploneconf.site')


def setupVarious(context):

    # Ordinarily, GenericSetup handlers check for the existence of XML files.
    # Here, we are not parsing an XML file, but we use this text file as a
    # flag to check that we actually meant for this import step to be run.
    # The file is found in profiles/default.

    if context.readDataFile('ploneconf.site_various.txt') is None:
        return

    site = api.portal.get()
    set_up_security(site)


def set_up_security(site):
    secSchema = ISecuritySchema(site)
    secSchema.set_enable_self_reg(True)
```

Agregue el archivo marcador `profile/default/ploneconf.site_various.txt` usado en la linea 18, usando el código anterior, con el siguiente contenido:

```
The ploneconf.site_various step is run if this file is present in the profile
```

Otorgar roles locales

Ya que se aplica sólo a una determinada carpeta en el sitio nosotros normalmente no escribe código para ello.

Pero podemos añadir fácilmente un método para el módulo `setuphandler` el cual cree la carpeta y se establece un cierto ajuste por ello.

Aquí hay un ejemplo:

```
# -*- coding: UTF-8 -*-
from plone.app.controlpanel.security import ISecuritySchema
from plone import api
from Products.CMFPlone.interfaces.constrains import ISelectableConstrainTypes
from plone.app.dexterity.behaviors import constrains

import logging

PROFILE_ID = 'profile-ploneconf.site:default'
logger = logging.getLogger('ploneconf.site')


def setupVarious(context):

    # Ordinarily, GenericSetup handlers check for the existence of XML files.
    # Here, we are not parsing an XML file, but we use this text file as a
    # flag to check that we actually meant for this import step to be run.
    # The file is found in profiles/default.

    if context.readDataFile('ploneconf.site_various.txt') is None:
        return

    site = api.portal.get()
    set_up_security(site)
    set_up_content(site)


def set_up_security(site):
    secSchema = ISecuritySchema(site)
    secSchema.set_enable_self_reg(True)


def set_up_content(site):
    """Create and configure some initial content"""
    if 'talks' in site:
        return
    talks = api.content.create(
        container=site,
        type='Folder',
        id='talks',
        title='Talks')
    api.content.transition(talks, 'publish')
    api.group.grant_roles(
        groupname='AuthenticatedUsers',
        roles=['Contributor'],
        obj=talks)
    # Enable constraining
    behavior = ISelectableConstrainTypes(talks)
    behavior.setConstrainTypesMode(constrains.ENABLED)
    behavior.setLocallyAllowedTypes(['talk'])
    behavior.setImmediatelyAddableTypes(['talk'])
    logger.info("Created and configured %s" % talks.absolute_url())
```

Recursos

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/27_resources/ src/ploneconf.site
```

Todavía no hemos hablado de los archivos CSS y Javascript. Por el momento estos son considerados recursos estáticos.

Puede declarar y acceder a recursos estáticos con direcciones URLs especiales. El archivo `configure.zcml` de nuestro paquete ya tiene una declaración de recursos:

```
<browser:resourceDirectory
  name="ploneconf.site"
  directory="resources" />
```

Ahora todos los archivos que ponemos en la carpeta llamada `resources` se pueden encontrar a través de la dirección URL http://localhost:8080/Plone/++resource++ploneconf.site/something.js

Vamos a crear un archivo `ploneconf.css` y un archivo `ploneconf.js` en la carpeta `resources`.

```
#visual-portal-wrapper {
    margin: 0 auto;
    position: relative;
    width: 1024px;
}

@media only screen and (max-width: 980px) {
   #visual-portal-wrapper {
       position: relative;
       width: auto;
   }
}

@media only screen and (max-width: 768px) {
   #portal-columns > div {
       width: 97.75%;
       margin-left: -98.875%;
       clear: both;
   }

   .searchButton,
   .searchSection {
```

```
        display: none;
    }
}
```

Si accedemos a la dirección URL http://localhost:8080/Plone/++resource++ploneconf.site/ploneconf.css para ver nuestro archivo CSS.

¿Cómo hacer que nuestros archivos Javascript y CSS sean usados cuando visiten la página? Añadiendo directamente en el HTML no es una buena solución, teniendo muchos archivos CSS y JS ralentiza carga de la página.

Con las herramientas `portal_css` y `portal_javascript` Plone tiene administradores de recursos que son capaces de *combinar* y *comprimir archivos* JS y CSS. Los recursos pueden ser añadidos de forma condicional y Plone detiene automáticamente la combinación de archivos cuando se está depurando Plone en modo `foreground`.

Tenemos que registrar nuestros recursos dentro de un perfil GenericSetup.

Agregar un nuevo archivo `profiles/default/cssregistry.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object name="portal_css">
  <stylesheet
      title=""
      applyPrefix="False"
      authenticated="False"
      bundle=""
      cacheable="True"
      compression="safe"
      conditionalcomment=""
      cookable="True"
      enabled="True"
      expression=""
      id="++resource++ploneconf.site/ploneconf.css"
      media=""
      rel="stylesheet"
      rendering="import"/>
</object>
```

Agregar un nuevo archivo `profiles/default/jsregistry.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object name="portal_javascripts">
  <javascript
    authenticated="False"
    bundle=""
    cacheable="True"
    compression="safe"
    conditionalcomment=""
    cookable="True"
    enabled="on"
    expression=""
    id="++resource++ploneconf.site/ploneconf.js"
    inline="False"/>
</object>
```

CAPÍTULO 29

Usando comportamientos de terceros

Agregar el comportamiento de banner con el complemento `collective.behavior.banner`

Hay un montón de complementos en Plone para diapositivas / banners / teasers. Nosotros pensamos que debe haber un mejor complemento y fue creado `collective.behavior.banner`.

Al igual muchos complementos que aún no se ha lanzado en PyPI pero sólo existe como código fuente en GitHub.

El buildout del entrenamiento contiene una sección `[sources]` que le dice buildout para descargar un complemento específico no disponible en PyPI sino en algún repositorio de código (generalmente GitHub):

```
[sources]
collective.behavior.banner = git https://github.com/collective/collective.behavior.banner.git pushurl
```

Definiendo la revisión nos salva de ser sorprendidos por algunos cambios en el código fuente que puede que no querríamos tener.

Después de añadir el código fuente, tenemos que añadir el paquete `Egg` al archivo buildout:

```
eggs =
    Plone
    ...
    collective.behavior.banner
    ...
```

Y vuelva a ejecutar el comando `./bin/buildout`

- Instalar el complemento
- Crear un nuevo tipo de contenido Dexterity `Banner` con **sólo** el comportamiento `Banner` habilitado.
- Crear una carpeta llamada `banners`
- Agregar dos banners dentro de la carpeta usando imágenes tomadas desde la dirección URL http://lorempixel.com/800/150/
- Agregar el comportamiento `Slider` al tipo de contenido por defecto `Página (Document)`
- Edita el la página frontpage y enlace a los nuevos banners.

CAPÍTULO 30

Tipos Dexterity - III: Python

¡Obtén el código!

Obtén el código para este capítulo (Más información) usando este comando en el directorio buildout:

```
cp -R src/ploneconf.site_sneak/chapters/29_dexterity_3/ src/ploneconf.site
```

Sin patrocinadores una conferencia sería difícil de financiar además de que es una buena oportunidad para las empresas Plone para hacer publicidad de sus servicios.

En esta parte vamos a tratar:

- Crear el tipo de contenido *sponsor* que tiene un esquema Python
- Crear un viewlet que muestra los patrocinadores ordenados por el nivel
- Discutir las escalas de imagen.

Primero creamos el esquema para el nuevo tipo de contenido. En lugar del código XML ahora usamos código Python. Cree una nueva carpeta `content` con un archivo vacío `__init__.py` en esta. Ahora agregue un nuevo archivo `content/sponsor.py`, con el siguiente código Python:

```
from plone.app.textfield import RichText
from plone.autoform import directives
from plone.namedfile import field as namedfile
from plone.supermodel.directives import fieldset
from plone.supermodel import model
from z3c.form.browser.radio import RadioFieldWidget
from zope import schema
from zope.schema.vocabulary import SimpleVocabulary
from zope.schema.vocabulary import SimpleTerm

from ploneconf.site import MessageFactory as _


LevelVocabulary = SimpleVocabulary(
    [SimpleTerm(value=u'platinum', title=_(u'Platinum Sponsor')),
     SimpleTerm(value=u'gold', title=_(u'Gold Sponsor')),
     SimpleTerm(value=u'silver', title=_(u'Silver Sponsor')),
     SimpleTerm(value=u'bronze', title=_(u'Bronze Sponsor'))]
    )


class ISponsor(model.Schema):
```

```
    """Dexterity-Schema for Sponsors
    """

    directives.widget(level=RadioFieldWidget)
    level = schema.Choice(
        title=_(u"Sponsoring Level"),
        vocabulary=LevelVocabulary,
        required=True
    )

    text = RichText(
        title=_(u"Text"),
        required=False
    )

    url = schema.URI(
        title=_(u"Link"),
        required=False
    )

    fieldset('Images', fields=['logo', 'advertisment'])
    logo = namedfile.NamedBlobImage(
        title=_(u"Logo"),
        required=False,
    )

    advertisment = namedfile.NamedBlobImage(
        title=_(u"Advertisment (Gold-sponsors and above)"),
        required=False,
    )

    directives.read_permission(notes="cmf.ManagePortal")
    directives.write_permission(notes="cmf.ManagePortal")
    notes = RichText(
        title=_(u"Secret Notes (only for site-admins)"),
        required=False
    )
```

Algunas cosas son notables aquí:

- Los campos en el esquema son en su mayoría basado en el paquete `zope.schema`. Una referencia de campos está disponible en la siguiente dirección URL: http://docs.plone.org/4/en/external/plone.app.dexterity/docs/reference/fields.html
- En `directives.widget(level=RadioFieldWidget)` cambiamos el widget predeterminado para un campo de elección de un menú desplegable para casilla de selección simple. Una referencia incompleta de widgets está disponible en la siguiente dirección URL: http://docs.plone.org/4/en/external/plone.app.dexterity/docs/reference/widgets.html
- `LevelVocabulary` se utiliza para crear las opciones que se utilizan en el campo `level`. De esta manera se podría traducir fácilmente el valor mostrado.
- `fieldset('Images', fields=['logo', 'advertisment'])` mueve las dos campos de imágenes a otra pestaña.
- `directives.read_permission(...)` establece el permiso de lectura y escritura, para el campo `note` a los usuarios que pueden agregar nuevos usuarios de tipoMiembro. Por lo general, este permiso sólo se concede a Administradores de Sitio y Administradores. Lo utilizamos para almacenar información que no debe ser visible públicamente. Tenga en cuenta que `obj.note` sigue siendo accesible en plantillas y código Python. Sólo

utilizando el widget (como lo hacemos en la vista posterior) comprueba el permiso.

- Nosotros no usaremos Grok aquí

Segundo nosotros creamos el *Factory Type Information - FTI* para el nuevo tipo de contenido en el archivo `profiles/default/types/sponsor.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object name="sponsor" meta_type="Dexterity FTI" i18n:domain="plone"
   xmlns:i18n="http://xml.zope.org/namespaces/i18n">
 <property name="title" i18n:translate="">Sponsor</property>
 <property name="description" i18n:translate="">None</property>
 <property name="icon_expr">string:${portal_url}/document_icon.png</property>
 <property name="factory">sponsor</property>
 <property name="add_view_expr">string:${folder_url}/++add++sponsor</property>
 <property name="link_target"></property>
 <property name="immediate_view">view</property>
 <property name="global_allow">True</property>
 <property name="filter_content_types">True</property>
 <property name="allowed_content_types" />
 <property name="allow_discussion">False</property>
 <property name="default_view">view</property>
 <property name="view_methods">
  <element value="view" />
 </property>
 <property name="default_view_fallback">False</property>
 <property name="add_permission">cmf.AddPortalContent</property>
 <property name="klass">plone.dexterity.content.Container</property>
 <property name="behaviors">
  <element value="plone.app.dexterity.behaviors.metadata.IDublinCore" />
  <element value="plone.app.content.interfaces.INameFromTitle" />
 </property>
 <property name="schema">ploneconf.site.content.sponsor.ISponsor</property>
 <property name="model_source"></property>
 <property name="model_file"></property>
 <property name="schema_policy">dexterity</property>
 <alias from="(Default)" to="(dynamic view)" />
 <alias from="edit" to="@@edit" />
 <alias from="sharing" to="@@sharing" />
 <alias from="view" to="(selected layout)" />
 <action title="View" action_id="view" category="object" condition_expr=""
    description="" icon_expr="" link_target="" url_expr="string:${object_url}"
    visible="True">
  <permission value="View" />
 </action>
 <action title="Edit" action_id="edit" category="object" condition_expr=""
    description="" icon_expr="" link_target=""
    url_expr="string:${object_url}/edit" visible="True">
  <permission value="Modify portal content" />
 </action>
</object>
```

Entonces nosotros registramos el FTI en el archivo `profiles/default/types.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<object name="portal_types" meta_type="Plone Types Tool">
 <property name="title">Controls the available content types in your portal</property>
 <object name="talk" meta_type="Dexterity FTI" />
 <object name="sponsor" meta_type="Dexterity FTI" />
```

```
 <!-- -*- more types can be added here -*- -->
</object>
```

Después volvemos a instalar nuestro paquete y entonces nosotros podemos crear un tipos de contenido `Sponsor`. Utilizamos la vista predeterminada proporcionada por Dexterity ya que mostremos a los patrocinadores en un viewlet.

En cambio, nosotros nos ajustamos la vista predeterminada con algunas sentencias CSS. Agregue lo siguiente al archivo `resources/ploneconf.css`, con el siguiente código CSS:

```
.template-view.portaltype-sponsor .named-image-widget img {
    width: 100%;
    height: auto;
}

.template-view.portaltype-sponsor fieldset#folder-listing {
    display: none;
}
```

Si quisiéramos una vista personalizada para los tipos de contenidos `Sponsor` podría tenerlo con esta apariencia.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en"
      metal:use-macro="context/main_template/macros/master"
      i18n:domain="ploneconf.site">
<body>
  <metal:content-core fill-slot="content-core">
    <h3 tal:content="structure view/w/level/render">
      Level
    </h3>

    <div tal:content="structure view/w/text/render">
      Text
    </div>

    <div class="newsImageContainer">
      <a tal:attributes="href context/url">
        <img tal:condition="python:getattr(context, 'logo', None)"
             tal:attributes="src string:${context/absolute_url}/@@images/logo/preview" />
      </a>
    </div>

    <div>
      <a tal:attributes="href context/url">
        Website
      </a>

      <img tal:condition="python:getattr(context, 'advertisment', None)"
           tal:attributes="src string:${context/absolute_url}/@@images/advertisment/preview" />

      <div tal:condition="python: 'notes' in view.w"
           tal:content="structure view/w/notes/render">
        Notes
      </div>

    </div>
  </metal:content-core>
</body>
</html>
```

Nota: Tenga en cuenta que tenemos que manejar el campo con permisos especiales: `tal:condition="python: 'notes' in view.w"` comprueba si la conveniencia diccionario `w` proporcionada por la clase base `DefaultView` sostiene el widget para el campo `note`. Si el usuario actual no tiene permiso `cmf.ManagePortal` será omitido del diccionario y sale un error ya que `notes` no sería una clave en `w`. A primera comprobación si falta trabajamos alrededor de eso.

Nosotros mostramos los tipos de contenidos `Sponsor` en la parte inferior de la página Web en un viewlet.

Registrar el viewlet en el archivo `browser/configure.zcml`, agregando el siguiente código ZCML:

```
<browser:viewlet
  name="sponsorsviewlet"
  manager="plone.app.layout.viewlets.interfaces.IPortalFooter"
  for="*"
  layer="..interfaces.IPloneconfSiteLayer"
  class=".viewlets.SponsorsViewlet"
  template="templates/sponsors_viewlet.pt"
  permission="zope2.View"
  />
```

Agregue en la clase viewlet en el archivo `browser/viewlets.py`, con el siguiente código Python:

```
from collections import OrderedDict
from plone import api
from plone.app.layout.viewlets.common import ViewletBase
from plone.memoize import ram
from ploneconf.site.behaviors.social import ISocial
from ploneconf.site.content.sponsor import LevelVocabulary
from random import shuffle
from time import time


class SocialViewlet(ViewletBase):

    def lanyrd_link(self):
        adapted = ISocial(self.context)
        return adapted.lanyrd


class SponsorsViewlet(ViewletBase):

    @ram.cache(lambda *args: time() // (60 * 60))
    def _sponsors(self):
        catalog = api.portal.get_tool('portal_catalog')
        brains = catalog(portal_type='sponsor')
        results = []
        for brain in brains:
            obj = brain.getObject()
            scales = api.content.get_view(
                name='images',
                context=obj,
                request=self.request)
            scale = scales.scale(
                'logo',
                width=200,
                height=80,
                direction='down')
```

```
            tag = scale.tag() if scale else ''
            if not tag:
                # only display sponsors with a logo
                continue
            results.append(dict(
                title=brain.Title,
                description=brain.Description,
                tag=tag,
                url=obj.url or obj.absolute_url(),
                level=obj.level
            ))
        return results

    def sponsors(self):
        sponsors = self._sponsors()
        if not sponsors:
            return
        results = OrderedDict()
        levels = [i.value for i in LevelVocabulary]
        for level in levels:
            level_sponsors = []
            for sponsor in sponsors:
                if level == sponsor['level']:
                    level_sponsors.append(sponsor)
            if not level_sponsors:
                continue
            shuffle(level_sponsors)
            results[level] = level_sponsors
        return results
```

- `_sponsors` regresa una lista de diccionarios conteniendo todo la información necesaria sobre los tipos de contenidos `Sponsor`.
- `_sponsors` se almacena en caché durante una hora usando plone.memoize. De esta manera no es necesario mantener todos los objetos tipos de contenidos `Sponsor` en la memoria todo el tiempo. También podríamos almacenar en caché hasta que uno de los tipos de contenidos `Sponsor` sea modificado:

```
    ...
    def _sponsors_cachekey(method, self):
        catalog = api.portal.get_tool('portal_catalog')
        brains = catalog(portal_type='sponsor')
        cachekey = sum([int(i.modified) for i in brains])
        return cachekey

    @ram.cache(_sponsors_cachekey)
    def _sponsors(self):
        catalog = api.portal.get_tool('portal_catalog')
    ...
```

- Creamos la etiqueta HTML `IMG` completa utilizando una escala personalizada (200x80) utilizando la vista `images` del paquete `plone.namedfile`. Este escalas en realidad los logotipos y las guarda como nuevos registros blobs.
- En `sponsors` regresamos un diccionario ordenado de una listas aleatorias de diccionarios (que contiene la información sobre los tipos de contenidos `Sponsor`).

Ver también:

http://docs.plone.org/4/en/develop/plone/images/content.html#for-plone-4

Agregar la plantilla en el archivo `browser/templates/sponsors_viewlet.pt`, con el siguiente código ZPT:

```
<div metal:define-macro="portal_sponsorbox"
     i18n:domain="ploneconf.site">
    <div id="portal-sponsorbox"
         tal:define="sponsors view/sponsors;">
        <div tal:repeat="level sponsors"
             tal:attributes="id python:'level-' + level"
             tal:condition="sponsors">
            <h3 tal:content="python: level.capitalize()">
                Level
            </h3>
            <tal:images tal:define="items python:sponsors[level];"
                        tal:repeat="item items">
                <div class="sponsor">
                    <a href=""
                       tal:attributes="href python:item['url'];
                                       title python:item['title'];">
                        <img tal:replace="structure python:item['tag']" />
                    </a>
                </div>
            </tal:images>
            <div class="visualClear"><!-- --></div>
        </div>
    </div>
</div>
```

Ahora agregue algunas sentencias CSS para estilizar la apariencia. Edite el archivo `resources/ploneconf.css`, agregando el siguiente código CSS:

```
.sponsor {
    float: left;
    margin: 0 1em 1em 0;
}

.sponsor:hover {
    box-shadow: 0 0 8px #000000;
    -moz-box-shadow: 0 0 8px #000000;
    -webkit-box-shadow: 0 0 8px #000000;
}
```

CAPÍTULO 31

Creando paquetes reusables con Eggs

Nosotros ya hemos creado un paquete `Egg` anteriormente.

Ahora vamos a crear una característica que es totalmente independiente de nuestro sitio ploneconf y puede ser reutilizado en otros paquetes.

Para hacer la distinción clara, esto no es un paquete de los nombres `ploneconf` pero desde `starzel`.

Vamos a agregar el comportamiento de votaciones.

Para esto nosotros necesitamos:

- Un comportamiento que almacene esos datos en registros de anotaciones.
- Un viewlet para presentar los votos.
- Un poco de código Javascript.
- Un poco de CSS.
- Algunos helper views para que el código Javascript puede comunicarse con Plone.

Nos ubicamos dentro del directorio `src` y ejecutamos el script llamado `zopeskel` desde nuestro directorio `bin` del proyecto, con los siguientes comando:

```
$ mkdir src
$ cd src
$ ../bin/zopeskel
```

Esto devuelve una lista de plantillas disponibles que podríamos utilizar. Elegimos la plantilla `dexterity` ya que es bastante pequeña, pero ya cuenta con algunas de las dependencias adecuadas que necesitamos, con el siguiente comando:

```
$ ../bin/zopeskel dexterity
```

Nosotros respondemos algunas preguntas:

- Enter project name: `starzel.votable_behavior`
- Expert Mode? (What question mode would you like? (easy/expert/all)?) ['easy']: `easy`
- Version (Version number for project) ['1.0']: `1.0.0`
- Description (One-line description of the project) ['Example Dexterity Product']: `Voting Behavior`
- Grok-Based? (True/False: Use grok conventions to simplify coding?) [True]: `False`
- Use relations? (True/False: include support for relations?) [False]: `False`

Tenemos que modificar ligeramente los archivos generados.

En `setup.py`, eliminamos por completo las variables `setup_requires` y `paster_plugins`. Estos son necesarios para las características que se usan muy poco y añade una gran cantidad de código en el directorio de fuentes que no queremos. Para la lista `install_requires`, añadimos una entrada para `plone.api`.

Nosotros eliminamos el archivo `tests.py`. Este ofrece un sistema de pruebas descontinuado, no queremos que empieces a partir de ahí.

El archivo `profiles/default/types.xml` lo eliminamos también. No vamos a definir nuevos tipos de contenidos en este paquete.

CAPÍTULO 32

Más comportamientos complejos

Usando Anotaciones

Vamos a almacenar la información en una anotación. No porque se necesita, sino porque se encuentra el código que utiliza anotaciones y la necesidad de entender las implicaciones.

Annotations en Zope / Plone significa que los datos no serán almacenados directamente en un objeto, sino de una manera indirecta y con multiples espacios de nombres para que varios paquetes pueden almacenar información bajo el mismo atributo, sin colisionar.

Así que usando anotaciones evita los conflictos de nombres. El costo es una indirección. El diccionario es persistente por lo que tiene que ser almacenado por separado. Además, se podría dar atributos un nombre que contiene un prefijo de espacio de nombres para evitar colisiones de nombres.

Usando Esquema

El atributo donde almacenamos nuestros datos será declarado como un campo de esquema. Marcamos el campo como un campo omitido, porque no vamos a crear widgets del paquete `z3c.form` para mostrarlos. Nosotros ofrecemos un esquema, porque muchos otros paquetes utilizan la información de esquema para obtener el conocimiento de los campos pertinentes.

Por ejemplo, cuando los archivos se han migrado a `blobs`, los nuevos objetos tuvieron que ser creados y cada campo de esquema fue copiado. El código no puede saber acerca de nuestro campo, excepto si proporcionamos información de esquema.

Escribiendo código

Para iniciar, creamos un directorio `behavior` con un archivo vació `behavior/__init__.py`.

Luego debemos, como siempre, registrar nuestro ZCML.

Primero, agrega la información de que existe otro archivo ZCML en `configure.zcml`, con el siguiente código ZCML:

```
<configure
      ...>

  ...
  <include package=".behavior" />
```

```
  ...

</configure>
```

Luego, cree el archivo `behavior/configure.zcml`, agregando el siguiente código ZCML:

```
<configure
    xmlns="http://namespaces.zope.org/zope"
    xmlns:plone="http://namespaces.plone.org/plone">

  <plone:behavior
      title="Voting"
      description="Allow voting for an item"
      provides="starzel.votable_behavior.interfaces.IVoting"
      factory=".voting.Vote"
      marker="starzel.votable_behavior.interfaces.IVotable"
      />

</configure>
```

Hay algunas diferencias importantes de nuestro primer comportamiento:

- Hay una interfaz marker.
- Hay una factory.

La factory es una clase que proporciona la lógica de comportamiento y da acceso a los atributos que ofrecemos. Las factories en la tierra de Plone / Zope se recuperan mediante la adaptación de un objeto con una interfaz. Si usted quiere su comportamiento, usted escribiría `IVoting(object)`

Pero para que esto funcione, el objeto *no* puede estar implementando la interfaz IVoting, porque si lo haría, ¡ `IVoting(object)` devolvería el objeto en sí mismo!. Si yo necesito una interfaz marker de los objetos proporcionando mi comportamiento, yo debo proporcionar uno, para esto usamos el atributo marker. Mi objeto implementa `IVotable` y debido a esto, podemos escribir `views` y `viewlets` sólo para este tipo de contenido.

Las interfaces necesitan ser escritas, en nuestro caso en un archivo `interfaces.py`, agregando el siguiente código Python:

```
# encoding=utf-8
from plone import api
from plone.autoform import directives as form
from plone.autoform.interfaces import IFormFieldProvider
from plone.supermodel import model
from plone.supermodel import directives
from zope import schema
from zope.interface import alsoProvides
from zope.interface import Interface

class IVotableLayer(Interface):
    """Marker interface for the Browserlayer
    """

# Ivotable is the marker interface for contenttypes who support this behavior
class IVotable(Interface):
    pass

# This is the behaviors interface. When doing IVoting(object),you receive an
# adapter
class IVoting(model.Schema):
    if not api.env.debug_mode():
```

```
        form.omitted("votes")
        form.omitted("voted")

    directives.fieldset(
        'debug',
        label=u'debug',
        fields=('votes', 'voted'),
    )

    votes = schema.Dict(title=u"Vote info",
                        key_type=schema.TextLine(title=u"Voted number"),
                        value_type=schema.Int(title=u"Voted so often"),
                        required=False)
    voted = schema.List(title=u"Vote hashes",
                        value_type=schema.TextLine(),
                        required=False)

    def vote(request):
        """
        Store the vote information, store the request hash to ensure
        that the user does not vote twice
        """

    def average_vote():
        """
        Return the average voting for an item
        """

    def has_votes():
        """
        Return whether anybody ever voted for this item
        """

    def already_voted(request):
        """
        Return the information wether a person already voted.
        This is not very high level and can be tricked out easily
        """

    def clear():
        """
        Clear the votes. Should only be called by admins
        """

alsoProvides(IVoting, IFormFieldProvider)
```

Se trata de una gran cantidad de código. El `IVotableLayer` nosotros lo necesitaremos más tarde para las `viewlets` y los `browser views`. Permite agregar aquí mismo. La interfaz `IVotable` es la interfaz marker simple. Sólo se utiliza para enlazar los `browser views` y `viewlets` a tipos de contenido que proporcionan nuestro comportamiento, por lo que no hay código necesario.

La clase `IVoting` es más compleja, como se puede ver. Mientras `IVoting` es sólo una interfaz, utilizamos `plone.supermodel.model.Schema` para las características avanzadas Dexterity. El paquete `zope.schema` no proporciona medios para ocultar campos. La directivas `form.omitted` de `plone.autoform` nos permite a nosotros anotar esta información adicional para que los auto-formularios renderizados puedan utilizar la información adicional.

Hacemos esta omitir condicional. Si ejecutamos Plone en modo de depuración, seremos capaces de ver los datos

internos en el formulario de edición.

Creamos los campos esquema mínimos para nuestras estructuras de datos internas. Por una pequeña prueba, yo le quité las directivas omitida de formulario y abrí la vista de edición de un tipo de contenido `talks` que utiliza el comportamiento. Después de ver la fealdad, yo decidí que debía proporcionar al menos mínimo de información. Los títulos y requerido son puramente opcional, pero muy útil si no se omitirán los campos, algo que puede ser útil al depurar el comportamiento. Más tarde, cuando ponemos en práctica el comportamiento, los atributos `votes` y `voted` se apliquen de tal manera que no se puede simplemente modificar estos campos, que son de sólo lectura.

Luego definimos la API que vamos a usar en los `browser views` y las `viewlets`.

La última línea se asegura de que los campos esquema sean conocidos por otros paquetes. Siempre que algún código quiere todos los esquemas de un objeto, que recibe el esquema definido directamente sobre el objeto y los esquemas adicional. Los esquemas adicionales se compilan mediante la búsqueda de comportamientos y si ofrecen la funcionalidad `IFormFieldProvider`. Sólo entonces nuestros campos son conocidos como campos de esquema.

Ahora la única cosa que falta es el comportamiento, la cual debemos colocar en archivo `behavior/voting.py`, con el siguiente código Python:

```
# encoding=utf-8
from hashlib import md5
from persistent.dict import PersistentDict
from persistent.list import PersistentList
from zope.annotation.interfaces import IAnnotations

KEY = "starzel.votable_behavior.behavior.voting.Vote"


class Vote(object):
    def __init__(self, context):
        self.context = context
        annotations = IAnnotations(context)
        if KEY not in annotations.keys():
            annotations[KEY] = PersistentDict({
                "voted": PersistentList(),
                'votes': PersistentDict()
                })
        self.annotations = annotations[KEY]

    @property
    def votes(self):
        return self.annotations['votes']

    @property
    def voted(self):
        return self.annotations['voted']
```

En nuestro método `__init__` obtenemos las *anotaciones* del objeto. Buscamos los datos con una específica clave.

La clave en este ejemplo es el mismo que lo que obtendría con `__name__+Vote.__name__`. Pero no vamos a crear un nombre dinámico, esto sería muy inteligente y hábil es malo.

Al declarar un nombre estático, no vamos a tener problemas si reestructuramos el código.

Usted puede ver, que inicializamos los datos si no existe. Trabajamos con `PersistentDict` y `PersistentList`. Para entender por qué hacemos esto, es importante entender cómo funciona el `ZODB`.

Ver también:

El `ZODB` puede almacenar objetos. Tiene un objeto raíz especial que usted nunca toca. Cualquier cosa que usted almacena donde, formará parte del objeto raíz, excepto si se trata de un sublclassing objeto `persistent.Persistent`.

Entonces se almacenará de forma independiente.

Tenga cuenta que los objetos persistentes Zope/ZODB cuando se cambia un atributo en él y marcar como cambiado. Los objetos modificados se guardarán en la base de datos. Esto sucede automáticamente. Cada `request` inicia una transacción y después de nuestro código corrió y el servidor Zope está preparando para enviar de nuevo la respuesta que nosotros generamos, la transacción sera enviada y todo lo cambiamos, será salvo.

Ahora, si tienen un diccionario normal sobre un objeto persistente, y usted sólo va a cambiar el diccionario, el objeto persistente no tiene manera de saber, si el diccionario se ha cambiado. Esto sucede de vez en cuando.

Así que una solución es cambiar el atributo especial `_p_changed` a `True` en el objeto persistente, o utilizar un `PersistentDict`. Eso es lo que estamos haciendo aquí.

Puede encontrar más información en la documentación de la ZODB, en particular, Reglas para Clases Persistentes

A continuación ofrecemos los campos internos a través de propiedades. El uso de este formulario de propiedad, hace que lean únicamente la propiedad, ya que no nos definimos manipuladores de escritura. Nosotros no los necesitamos entonces para que nosotros no los agregaremos.

Como se ha visto en la declaración del esquema, si ejecuta su sitio en modo de depuración, verá un campo de edición para estos campos. Pero si tratar de cambiar estos campos abra una excepción.

Continuemos con este archivo:

```
    def _hash(self, request):
        """
        This hash can be tricked out by changing IP addresses and might allow
        only a single person of a big company to vote
        """
        hash_ = md5()
        hash_.update(request.getClientAddr())
        for key in ["User-Agent", "Accept-Language",
                    "Accept-Encoding"]:
            hash_.update(request.getHeader(key))
        return hash_.hexdigest()

    def vote(self, vote, request):
        if self.already_voted(request):
            raise KeyError("You may not vote twice")
        vote = int(vote)
        self.annotations['voted'].append(self._hash(request))
        votes = self.annotations['votes']
        if vote not in votes:
            votes[vote] = 1
        else:
            votes[vote] += 1

    def average_vote(self):
        if not has_votes(self):
            return 0
        total_votes = sum(self.annotations['votes'].values())
        total_points = sum([vote * count for (vote, count) in
                            self.annotations['votes'].items()])
        return float(total_points) / total_votes

    def has_votes(self):
        return len(self.annotations.get('votes', [])) != 0

    def already_voted(self, request):
        return self._hash(request) in self.annotations['voted']

```

```
38      def clear(self):
39          annotations = IAnnotations(self.context)
40          annotations[KEY] = PersistentDict({'voted': PersistentList(),
41                                             'votes': PersistentDict()})
42          self.annotations = annotations[KEY]
```

Empezamos con un poco del método `helper` que no está expuesta a través de la interfaz. No queremos que la gente vote dos veces. Hay muchas formas para asegurar esto y cada uno tiene defectos.

Elegimos esta manera de mostrar cómo acceder a la información de la `request` del navegador del usuario que nos envió. En primer lugar, tenemos la dirección IP del usuario, entonces podemos acceder a un pequeño conjunto de encabezados desde el navegador de los usuarios y generar una suma de comprobación MD5 de esto.

El método de votación, quiere un voto y una `request`. Comprobamos las condiciones previas, a continuación, convertimos el voto a un número entero, almacenar la `request` que tiene `voted` y los votos en el diccionario `votes`. Sólo contamos allí, con qué frecuencia se ha dado ninguna votación.

Todo lo demás es simplemente código Python aburrido.

Un viewlet para el comportamiento de votaciones

Viewlet de Votaciones

- El viewlet para la interfaz IVoteable.
- La plantilla del viewlet.
- Agregar jQuery incluyen declaraciones.
- Guardar el voto en el objeto usando anotaciones.

Anteriormente hemos agregado la lógica que salva los votos en los objetos. Nosotros ahora podemos crear la interfaz de usuario para esto.

Puesto que queremos utilizar la interfaz de usuario en más de una página (no sólo la vista `talk-view` sino también la vista `talk-listing`) debemos ponerla en alguna parte.

- Para manejar el ingreso de datos del usuario nosotros no usamos un formulario pero si links y ajax.
- La votación en si, es en su efecto manejada por otra vista.

Registramos el viewlet en el archivo `browser/configure.zcml`, agregando el siguiente código ZCML:

```
<configure xmlns="http://namespaces.zope.org/zope"
    xmlns:browser="http://namespaces.zope.org/browser">

    ...

  <browser:viewlet
    name="voting"
    for="starzel.votable_behavior.interfaces.IVoting"
    manager="plone.app.layout.viewlets.interfaces.IBelowContentTitle"
    layer="..interfaces.IVotableLayer"
    class=".viewlets.Vote"
    template="templates/voting_viewlet.pt"
    permission="zope2.View"
    />

    ....

</configure>
```

Nosotros extendemos en el archivo `browser/viewlets.py`, agregando el siguiente código Python:

```
from plone.app.layout.viewlets import common as base


class Vote(base.ViewletBase):
    pass
```

Esto agregará un `viewlet` a una ranura debajo del título y esperará una plantilla `voting_viewlet.pt` en una carpeta `browser/templates`.

Vamos a agregar el archivo de la plantilla `templates/social_viewlet.pt` sin ninguna lógica.

```
<div class="voting">
    Wanna vote? Write code!
</div>

<script type="text/javascript">
  jq(document).ready(function(){
    // please add some jQuery-magic
  });
</script>
```

- Reinicie Plone.
- Muestre el viewlet.

Escribiendo el código del Viewlet

Actualizar en la clase `viewlet` que contiene la lógica necesaria en el archivo `browser/viewlets.py`, con el siguiente código Python:

```
from plone.app.layout.viewlets import common as base
from Products.CMFCore.permissions import ViewManagementScreens
from Products.CMFCore.utils import getToolByName

from starzel.votable_behavior.interfaces import IVoting


class Vote(base.ViewletBase):

    vote = None
    is_manager = None

    def update(self):
        super(Vote, self).update()

        if self.vote is None:
            self.vote = IVoting(self.context)
        if self.is_manager is None:
            membership_tool = getToolByName(self.context, 'portal_membership')
            self.is_manager = membership_tool.checkPermission(
                ViewManagementScreens, self.context)

    def voted(self):
        return self.vote.already_voted(self.request)

    def average(self):
        return self.vote.average_vote()
```

```
28
29      def has_votes(self):
30          return self.vote.has_votes()
```

La plantilla

Y extiende la plantilla en el archivo `browser/templates/voting_viewlet.pt`, con el siguiente código ZPT:

```
1  <tal:snippet omit-tag="">
2    <div class="voting">
3      <div id="current_rating" tal:condition="viewlet/has_votes">
4        The average vote for this talk is <span tal:content="viewlet/average">200</span>
5      </div>
6      <div id="alreadyvoted" class="voting_option">
7        You already voted this talk. Thank you!
8      </div>
9      <div id="notyetvoted" class="voting_option">
10       What do you think of this talk?
11       <div class="votes"><span id="voting_plus">+1</span> <span id="voting_neutral">0</span> <span id
12       </div>
13     </div>
14     <div id="no_ratings" tal:condition="not: viewlet/has_votes">
15       This talk has not been voted yet. Be the first!
16     </div>
17     <div id="delete_votings" tal:condition="viewlet/is_manager">
18       Delete all votings
19     </div>
20     <div id="delete_votings2" class="areyousure warning"
21          tal:condition="viewlet/is_manager"
22          >
23       Are you sure?
24     </div>
25     <a href="#" class="hiddenStructure" id="context_url"
26        tal:attributes="href context/absolute_url"></a>
27     <span id="voted" tal:condition="viewlet/voted"></span>
28   </div>
29   <script type="text/javascript">
30     $(document).ready(function(){
31       starzel_votablebehavior.init_voting_viewlet($(".voting"));
32     });
33   </script>
34 </tal:snippet>
```

Tenemos muchas piezas pequeñas, la mayoría de las cuales serán ocultadas por javascript a menos que sea necesario. Al proporcionar toda esta información de estado en HTML, podemos utilizar herramientas de traducción estándar para traducir. Traducir cadenas en javascript requiere trabajo adicional.

Nosotros necesitamos algunos sentencias CSS que almacenaremos en el archivo `static/starzel_votablebehavior.css`, con el siguiente código CSS:

```
1  .voting {
2      float: right;
3      border: 1px solid #ddd;
4      background-color: #DDDDDD;
5      padding: 0.5em 1em;
6  }
7
```

```
.voting .voting_option {
    display: None;
}

.areyousure {
    display: None;
}

.voting div.votes span {
    border: 0 solid #DDDDDD;
    cursor: pointer;
    float: left;
    margin: 0 0.2em;
    padding: 0 0.5em;
}

.votes {
    display: inline;
    float: right;
}

.voting #voting_plus {
    background-color: LimeGreen;
}

.voting #voting_neutral {
    background-color: yellow;
}

.voting #voting_negative {
    background-color: red;
}
```

El código javascript

Para que funcione en el navegador, algunas sentencias Javascript en el archivo `static/starzel_votablebehavior.js`, con el siguiente código Javascript:

```
/*global location: false, window: false, jQuery: false */
(function ($, starzel_votablebehavior) {
    "use strict";
    starzel_votablebehavior.init_voting_viewlet = function (context) {
        var notyetvoted = context.find("#notyetvoted"),
            alreadyvoted = context.find("#alreadyvoted"),
            delete_votings = context.find("#delete_votings"),
            delete_votings2 = context.find("#delete_votings2");

        if (context.find("#voted").length !== 0) {
            alreadyvoted.show();
        } else {
            notyetvoted.show();
        }

        function vote(rating) {
            return function inner_vote() {
                $.post(context.find("#context_url").attr('href') + '/vote', {
```

```
                    rating: rating
                }, function () {
                    location.reload();
                });
            };
        }

        context.find("#voting_plus").click(vote(1));
        context.find("#voting_neutral").click(vote(0));
        context.find("#voting_negative").click(vote(-1));

        delete_votings.click(function () {
            delete_votings2.toggle();
        });
        delete_votings2.click(function () {
            $.post(context.find("#context_url").attr("href") + "/clearvotes", function () {
                location.reload();
            });
        });
    };
}(jQuery, window.starzel_votablebehavior = window.starzel_votablebehavior || {}));
```

Este código Javascript se adhiere a las reglas `jshint` de crockfort, por lo que todas las variables se declaran al principio del método. Mostramos y ocultamos bastantes pequeños elementos html aquí.

Escribiendo 2 simples helpers view

Nuestro código Javascript se comunica con nuestro sitio llamando a las vistas que aún no existen. Estas vistas no necesitan mostrar código html, pero deben devolver un estatus válido. Las excepciones establecen el estado correcto y no están siendo mostradas por Javascript, por lo que nos conviene bien.

Como puede recordar, el método `vote` puede devolver una excepción, si alguien vota dos veces. No capturamos esta excepción. El usuario nunca verá esta excepción.

Ver también:

La captura de excepciones contiene un comportamiento contra-intuitivo para los nuevos desarrolladores.

```
try:
    something()
except:
    fix_something()
```

Zope reclama algunas excepciones para sí mismos. Eso necesita que funcionen correctamente.

Por ejemplo, si dos solicitudes intentan modificar algo al mismo tiempo, una solicitud lanzará una excepción, a `ConflictError`.

Zope captura la excepción, espera una cantidad de tiempo aleatoria e intenta procesar la solicitud nuevamente, hasta tres veces. Si capturas esa excepción, estás en problemas, así que no lo hagas. Nunca.

Como tantas veces, debemos extender en el archivo `browser/configure.zcml`, agregando el siguiente código ZCML:

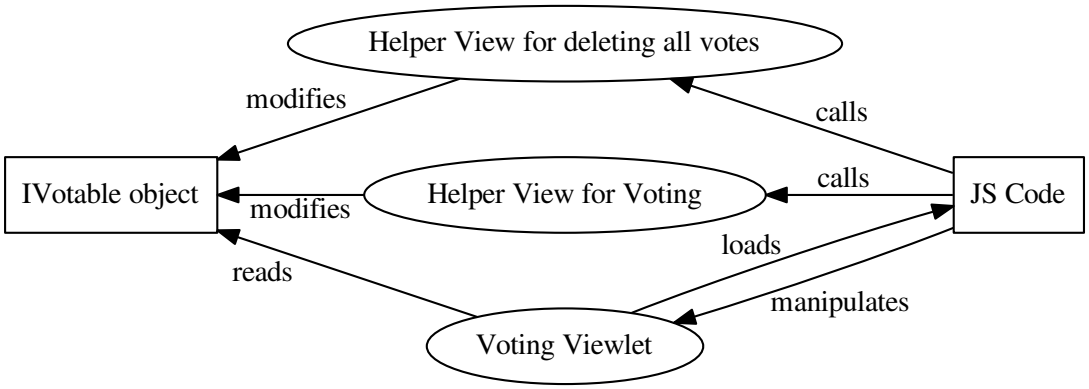
```
...

<browser:page
  name="vote"
```

```
  for="starzel.votable_behavior.interfaces.IVotable"
  layer="..interfaces.IVotableLayer"
  class=".vote.Vote"
  permission="zope2.View"
  />

<browser:page
  name="clearvotes"
  for="starzel.votable_behavior.interfaces.IVotable"
  layer="..interfaces.IVotableLayer"
  class=".vote.ClearVotes"
  permission="zope2.ViewManagementScreens"
  />

...
```

A continuación, agregamos nuestras vistas simples en el archivo `browser/vote.py`, con el siguiente código Python:

```
from zope.publisher.browser import BrowserPage

from starzel.votable_behavior.interfaces import IVoting


class Vote(BrowserPage):

    def __call__(self, rating):
        voting = IVoting(self.context)
        voting.vote(rating, self.request)
        return "success"


class ClearVotes(BrowserPage):

    def __call__(self):
        voting = IVoting(self.context)
        voting.clear()
        return "success"
```

Se han creado muchas partes móviles. Aquí hay un pequeño resumen:



CAPÍTULO 34

Haciendo nuestro paquete reusable

Esta paquete contiene algunos problemas.

- Sin la configuración de permisos, los usuarios no pueden personalizar quién y cuándo los usuarios pueden votar.
- Hacemos cosas, pero no provocamos eventos. Los eventos permiten que otros reaccionen.

Agregando permisos

Los permisos tienen una larga historia, hay dos tipos de permisos.

En Zope2, un permiso era sólo una cadena.

En ZTK, un permiso es un objeto que se obtiene registrado como una Utilidad.

Debemos apoyar a ambos, en algunos casos tenemos que referenciar el permiso por su versión Zope2, en algunos por su versión ZTK.

Por suerte, hay una declaración ZCML para registrar un permiso de ambas maneras en un solo paso.

Ver también:

El registro de configuración quería resolver un problema, pero ahora nos topamos con un problema que no se resolvió correctamente.

Nuestro permiso es una utilidad. Nuestras browser views declaran este permiso como un requisito para verlas.

Cuando nuestras browser views se registren, los permisos deben existir ya. Si intenta registrar los permisos después de las vistas, Zope no se iniciará porque no sabe acerca de los permisos.

Modificar el archivo `configure.zcml`, con el siguiente código ZCML:

```
<configure
  ...>

  <includeDependencies package="." />

  <permission
      id="starzel.votable_behavior.view_vote"
      title="starzel.votable_behavior: View Vote"
      />

  <permission
      id="starzel.votable_behavior.do_vote"
      title="starzel.votable_behavior: Do Vote"
```

```
      />

  <include package=".browser" />

  ...

</configure>
```

En algunos lugares tenemos que hacer referencia a las cadenas de permisos de Zope 2. Es la mejor práctica proporcionar una variable estática para esto.

Proporcionamos esto en el archivo `__init__.py`, con el siguiente código Python:

```
...
DoVote = 'starzel.votable_behavior: Do Vote'
ViewVote = 'starzel.votable_behavior: View Vote'
```

Usando nuestros permisos

Como puede ver, creamos dos permisos, uno para votar, uno para ver los votos.

Si uno no puede ver las votaciones, no necesita acceso al viewlet de votos.

Mientras estamos en ello, si uno no puede votar, no necesita acceso a la helper view para presentar realmente un voto.

Podemos agregar esta restricción al archivo `browser/configure.zcml`, con el siguiente código ZCML:

```
<configure
  xmlns="http://namespaces.zope.org/zope"
  xmlns:browser="http://namespaces.zope.org/browser"
  i18n_domain="starzel.votable_behavior">

  <browser:viewlet
    name="voting"
    for="starzel.votable_behavior.interfaces.IVotable"
    manager="plone.app.layout.viewlets.interfaces.IBelowContentTitle"
    template="templates/voting_viewlet.pt"
    layer="..interfaces.IVotableLayer"
    class=".viewlets.Vote"
    permission="starzel.votable_behavior.view_vote"
    />

  <browser:page
    name="vote"
    for="starzel.votable_behavior.interfaces.IVotable"
    layer="..interfaces.IVotableLayer"
    class=".vote.Vote"
    permission="starzel.votable_behavior.do_vote"
    />

  ...

</configure>
```

Estamos configurando componentes, por lo que usamos el nombre de componente del permiso, que es el `id` parte de la declaración que añadimos anteriormente.

Ver también:

Entonces, ¿qué sucede, si no protegemos la browser view para votar?

La persona podría votar, mediante la elaboración manual de la URL. Las Browser Views ejecutar código sin ninguna restricción, es su trabajo tener cuidado de la seguridad.

Pero... si una persona no tiene acceso al objeto en absoluto, tal vez porque el sitio está configurado para que los usuarios anónimos no puedan acceder a objetos privados, los usuarios no autorizados no podrán enviar un voto.

Es decir, porque Zope comprueba los permisos de seguridad al intentar encontrar el objeto correcto. Si no puede encontrar el objeto debido a las restricciones de seguridad no se cumplen, ninguna vista mal, nunca ser llamado, ya que habría sido el siguiente paso.

Ahora protegemos nuestras `views` y `viewlets`. Todavía mostramos la opción de votar.

Debemos agregar una condición en nuestra plantilla de página, y debemos proporcionar la información de la condición en nuestra clase viewlet.

Mueva al archivo `browser/viewlets.py`, con el siguiente código Python:

```
...

from starzel.votable_behavior import DoVote


class Vote(base.ViewletBase):

     ...
     can_vote = None

    def update(self):

        ...

        if self.is_manager is None:
            membership_tool = getToolByName(self.context, 'portal_membership')
            self.is_manager = membership_tool.checkPermission(
                ViewManagementScreens, self.context)
            self.can_vote = membership_tool.checkPermission(
                DoVote, self.context)


...
```

Y la plantilla en el archivo `browser/templates/voting_viewlet.pt`, con el siguiente código ZPT:

```
<tal:snippet omit-tag="">
  <div class="voting">

    ...

    <div id="notyetvoted" class="voting_option"
            tal:condition="view/can_vote">
      What do you think of this talk?
      <div class="votes"><span id="voting_plus">+1</span> <span id="voting_neutral">0</span> <span id
      </div>
    </div>
    <div id="no_ratings" tal:condition="not: view/has_votes">
      This talk has not been voted yet.<span tal:omit-tag="" tal:condition="view/can_vote"> Be the fi
    </div>

  ...

```

```
  </div>

...

</tal:snippet>
```

A veces se producen errores sutiles debido a los cambios, en este caso, me di cuenta de que sólo debería animar a la gente a votar, ¡si a ellos se les permite votar!

Proporcionar valores por defecto

¿Ya terminamos?, ¿Quién puede votar ahora?

Tenemos que decirle a alguien.

En el ZTK, un permiso es un objeto que se obtiene registrado como una Utilidad.

La configuración persistente se gestiona en otro archivo: `profiles/default/rolemap.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<rolemap>
  <permissions>
    <permission name="starzel.votable_behavior: View Vote" acquire="True">
      <role name="Anonymous"/>
    </permission>
    <permission name="starzel.votable_behavior: Do Vote" acquire="True">
      <role name="Anonymous"/>
    </permission>
  </permissions>
</rolemap>
```

Usando starzel.votable_behavior en ploneconf.site

- Queremos usar el comportamiento de votación, para que nuestros usuarios revisores puedan votar.
- Para mostrar cómo usar los eventos, vamos a publicar automáticamente los tipos de contenidos talks que han alcanzado una cierta puntuación.
- No vamos a dejar que todos voten todo.

Primero, debemos agregar nuestro paquete como una dependencia al paquete `ploneconf.site`.

Hacemos esto en dos lugares. La descripción del paquete egg en el archivo `setup.py` necesita `starzel.votable_behavior` como una dependencia. De lo contrario ningún código fuente estará disponible.

La configuración persistente necesita ser instalada cuando instalamos nuestro sitio. Esto se configura en GenericSetup.

Comenzamos con la edición del archivo `setup.py`, con el siguiente código:

```
...
zip_safe=False,
install_requires=[
    'setuptools',
    'plone.app.dexterity [relations]',
    'plone.app.relationfield',
    'plone.namedfile [blobs]',
    'starzel.votable_behavior',
    # -*- Extra requirements: -*-
],
...
```

A continuación modificamos el archivo `profiles/default/metadata.xml`, con el siguiente código XML:

```
<metadata>
  <version>1002</version>
    <dependencies>
      <dependency>profile-plone.app.dexterity:default</dependency>
      <dependency>profile-plone.app.relationfield:default</dependency>
      <dependency>profile-starzel.votable_behavior:default</dependency>
    </dependencies>
</metadata>
```

Qué nombre tan raro. `profile-` es un prefijo que siempre necesitarás hoy en día. Luego viene el nombre del paquete egg, y la parte después de los dos puntos es el nombre del perfil. El nombre del perfil se define en configuraciones ZCML. Hasta ahora he tropezado sólo sobre un paquete donde el nombre del directorio de perfil era diferente al nombre del perfil GenericSetup.

Ahora el paquete está allí, pero nada es votable. Esto se debe a que ningún tipo de contenido declara utilizar este comportamiento. Podemos añadir este comportamiento a través del panel de control, exportar la configuración y

almacenarla en nuestro paquete egg. Vamos a añadirlo a mano ahora.

Tenemos que agregar el comportamiento al tipo de contenidos talks, haremos esto en el archivo `profiles/default/types/talk.xml`.

Nota: Administrando las dependencias en el archivo `metadata.xml` es una buena práctica. No podemos confiar en recordar lo que tendríamos que hacer a mano. Por ejemplo, ¿recuerdas que tuvimos que añadir para seleccionar `Tipos predeterminados de Plone basados en Dexterity` al crear un nuevo sitio de Plone?

En lugar de eso, debemos agregar `<dependency>profile-plone.app.contenttypes:plone-content</dependency` como la documentación de plone.app.contenttypes recomienda.

```
<property name="behaviors">
  <element value="plone.app.dexterity.behaviors.metadata.IDublinCore"/>
  <element value="plone.app.content.interfaces.INameFromTitle"/>
  <element value="starzel.votable_behavior.interfaces.IVoting"/>
</property>
```

Ahora nosotros podemos reinstalarlo en nuestro sitio Plone.

Todo el mundo puede votar ahora en los tipos de contenidos `talks`. Eso no es lo que queríamos. En realidad, sólo queremos que los revisores voten en las charlas pendientes. Esto significa que, dependiendo del estado del flujo de trabajo, el permiso tiene que cambiar. Por suerte, los flujos de trabajo se pueden configurar para hacer precisamente eso. Los tipos de contenidos `talks` ya tienen su propio flujo de trabajo. Así que no interferiremos con otros paquetes.

En primer lugar, tenemos que decirle al flujo de trabajo que gestionará más permisos. A continuación, tenemos que configurar para cada estado, que el rol de usuario tiene ahora los dos nuevos permisos.

Esa es una configuración muy detallada, tal vez usted quiere hacerlo en la interfaz web y exportar la configuración. Por otro lado, es fácil cometer un simple error en ambos sentidos. Yo acabo de presentar la forma XML aquí.

La configuración del flujo de trabajo está en el archivo `profiles/default/workfows/talks_workflow.xml`, con el siguiente código XML:

```
<?xml version="1.0"?>
<dc-workflow workflow_id="talks_workflow" title="Talks Workflow" description=" - Simple workflow that
 <permission>Access contents information</permission>
 <permission>Change portal events</permission>
 <permission>Modify portal content</permission>
 <permission>View</permission>
 <permission>starzel.votable_behavior: View Vote</permission>
 <permission>starzel.votable_behavior: Do Vote</permission>
 <state state_id="pending" title="Pending review">
  <description>Waiting to be reviewed, not editable by the owner.</description>
  ...
  <permission-map name="starzel.votable_behavior: View Vote" acquired="False">
   <permission-role>Site Administrator</permission-role>
   <permission-role>Manager</permission-role>
   <permission-role>Reviewer</permission-role>
  </permission-map>
  <permission-map name="starzel.votable_behavior: Do Vote" acquired="False">
   <permission-role>Site Administrator</permission-role>
   <permission-role>Manager</permission-role>
   <permission-role>Reviewer</permission-role>
  </permission-map>
  ...
 </state>
 <state state_id="private" title="Private">
  <description>Can only be seen and edited by the owner.</description>
```

```
  ...
  <permission-map name="starzel.votable_behavior: View Vote" acquired="False">
   <permission-role>Site Administrator</permission-role>
   <permission-role>Manager</permission-role>
  </permission-map>
  <permission-map name="starzel.votable_behavior: Do Vote" acquired="False">
   <permission-role>Site Administrator</permission-role>
   <permission-role>Manager</permission-role>
  </permission-map>
  ...
 </state>
 <state state_id="published" title="Published">
  <description>Visible to everyone, editable by the owner.</description>
  ...
  <permission-map name="starzel.votable_behavior: View Vote" acquired="False">
   <permission-role>Site Administrator</permission-role>
   <permission-role>Manager</permission-role>
  </permission-map>
  <permission-map name="starzel.votable_behavior: Do Vote" acquired="False">
  </permission-map>
  ...
 </state>
  ...
</dc-workflow>
```

Tenemos que volver a instalar nuestro producto de nuevo.

Pero esta vez, esto no es suficiente. Los permisos se actualizan en los cambios de flujo de trabajo. Mientras no ocurra un cambio en el flujo de trabajo, los tipos de contenidos `talks` tienen los mismos permisos que nunca.

Afortunadamente, para eso hay un botón "*Update security settings*" en la vista Flujo de trabajo de la ZMI.

Después de hacer clic en este, sólo los usuarios Administradores y los usuarios Revisores pueden ver la funcionalidad de votación.

Por último, añadimos nuestra tonta función para autorizar los tipos de contenidos `talks`.

Rápidamente terminas escribiendo muchos manejadores de eventos, por lo que ponemos todo en un directorio para los manejadores de eventos.

Para los eventos nosotros necesitamos un directorio `events`.

Crear el directorio `events` y agregar un archivo vació `events/__init__.py`.

Lo próximo, registramos el directorio `events` en el archivo `configure.zcml`

```
<include package=".events" />
```

Entonces, escribimos la configuración ZCML para los eventos dentro del archivo `events/configure.zcml`, agregando el siguiente código ZCML:

```
<configure
    xmlns="http://namespaces.zope.org/zope">

  <subscriber
    for="starzel.votable_behavior.interfaces.IVotable
         zope.lifecycleevent.IObjectModifiedEvent"
    handler=".votable.votable_update"
    />

</configure>
```

Esto parece un `MultiAdapter`. Queremos ser notificados cuando un objeto `IVotable` se modifica. Nuestro método recibirá el objeto `votable`, y el evento en sí.

Y finalmente, nuestro manipulador del evento en el archivo `events/votable.py`, con el siguiente código:

```
from plone.api.content import transition
from plone.api.content import get_state
from starzel.votable_behavior.interfaces import IVoting


def votable_update(votable_object, event):
    votable = IVoting(votable_object)
    if get_state(votable_object) == 'pending':
        if votable.average_vote() > 0.5:
            transition(votable_object, transition='publish')
```

Estamos usando un montón de sentencias plone api aquí. La Plone API hace el código una brisa fresca. Además, no hay nada realmente interesante. Sólo haremos algo, si el estado de flujo de trabajo está pendiente y el voto promedio es superior a 0,5. Como puede ver, el método `transition` no quiere el estado de destino, sino la transición para mover el estado al estado de destino.

No hay nada especial en marcha.

CAPÍTULO 36

Buildout - Parte II: Cómo prepararse para el despliegue

El buildout de starzel

Miremos la configuración buildout que usaremos para nuestros proyectos: https://github.com/starzel/buildout

Tiene algunas características notables:

- Se extiende a archivos en github por todos los proyectos de la misma versión

```
[buildout]
extends =
    https://raw.github.com/starzel/buildout/4.3.3/linkto/base.cfg
```

- Trabajo mínimo para instalar un nuevo proyecto.
- Preestablecidos para desarrollo, pruebas, implantación y producción.

Una instalación de despliegue

- servidor zeo (zeoserver) y clientes zeo (zeoclients).
- balanceador de carga con haproxy.
- monitoreo de red con nagios.
- cacheo de contenidos con varnish.
- monitoreo.
- supervisor de procesos.
- respaldo del despliegue.
- logrotate los registros de eventos.
- precompilador de codigo Python.
- tareas crontab (cronjobs).

Otras herramientas a usar

- Fabric (administra sitios).

- Sentry (monitoreo de errores).
- Ansible (Administra e instala servidores y herramientas).
- Nagios (monitoreo de servidores).
- Jenkins (pruebas continuas de software).
- Piwik (estadísticas).
- Gitlab (repositorio y revisión de código).
- Redmine (sistema de ticket y wiki).

CAPÍTULO 37

El futuro de Plone

- Plone 5.
- Mapa de ruta de Plone hasta el año 2020.

El proceso Plone y la Comunidad Plone.

Plips: https://dev.plone.org/report/24

CAPÍTULO 38

Opcional

- Pruebas
- Paquete `zc3.form`.
- Los portlets.
- La ZCA en profundidad.
- La ZODB.
- Más y más campos complejos.
- Relaciones.
- Formularios de edición personalizados.
- Paneles de control.
- Paquete `plone.app.registry`.
- Seguridad (permisos / roles / flujo de trabajo).
- Permisos personalizados.
- Flujo de trabajo personalizado.
- Usuarios, autenticación, perfiles de miembros, ldap.
- Cacheo (con el paquete `plone.app.caching`).
- GenericSetup.
- Pasos de actualización.
- Migraciones.
- Procesamiento asíncrono.
- Hablando con APIs externas.
- Mockup.
- Personalizando el tema barcelonetta.
- Paquete `plone.app.widgets`.
- deployment_code.
- Despliegue
    - Opciones de almacenamiento.

- respaldo
- Configuraciones buildouts de despliegue.
- Cacheo (con varnish).

Por favor, tenga en cuenta que este documento no está completo sin la palabra hablada de un entrenador. Aunque tratamos de incluir las partes más importantes de lo que enseñamos en la narración, no puede considerarse completa sin la palabra hablada.

CAPÍTULO 39

Cambios

1.2.5 (sin publicar)

- Revisión de parte uno para la Ploneconf 2014, Bristol [smcmahon]

1.2.4 (2014-10-03)

- Revisado para la Ploneconf 2014, Bristol [pbauer, gomez]
- Agregue los primeros ejercicios y cree archivos css + js para expandir / colapsar las soluciones de las practicas [pbauer]
- Corregir la compilación local con el paquete rtd-Theme [pbauer]
- Agregar traducción al Español. [macagua]
- Agregar soporte para las traducciones en https://www.transifex.com/plone/plone4-training/ [macagua]

1.2.3 (2014-07-11)

- Mover las fuentes de https://github.com/plone/training y renderizarlo en http://plone-training.readthedocs.io/en/legacy/ [pbauer]
- integrar con docs.plone.org y papyrus [do3cc, hellfish2]
- Cambiar licencia a https://creativecommons.org/licenses/by/4.0/ [pbauer]
- Documento de como contribuir [pbauer, hellfish2]
- Actualizar introducción [pbauer, hellfish2]

1.2.2 (2014-06-01)

- Arreglar todos los errores encontrados durante el entrenamiento en Mayo 2014 [pbauer, hellfish2]
- Mover archivos rst al repositorio https://github.com/starzel/training [pbauer]

1.2.1 (2014-05-30)

- Publicar una versión verbosa en http://starzel.github.io/training/index.html [pbauer, hellfish2]
- Agregar un comando bash, para copiar el código de ploneconf.site_sneak a ploneconf.site para cada capítulo [pbauer, hellfish2]
- Incluir vagrant-setup como un archivo Zip [pbauer, hellfish2]
- Serios pequeños bugfixes [pbauer, hellfish2]

1.2 (2014-05-23)

- Ampliamente expandido y reescrito para un entrenamiento en Mayo 2014 [pbauer, do3cc, hellfish2]
- Remover grok [pbauer, hellfish2]
- usar plone.app.contenttypes desde el principio [pbauer, hellfish2]
- usar plone.api [pbauer, hellfish2]
- Reescribir vagrant-setup [pbauer, hellfish2]
- Eliminar el uso de plone.app.themeeditor [pbauer, hellfish2]
- Agregue más capítulos:
  - Tipos Dexterity II: Creciendo
  - Usar contenido generado
  - Programando en Plone
  - Búsqueda Personalizada
  - Eventos
  - Usando comportamientos de complementos de terceros
  - Tipos Dexterity III: python
  - ...

  [pbauer, do3cc]

1.1 (Octubre 2013)

- Revisado y ampliado para la Ploneconf 2013, Brasilia [pbauer, do3cc]

1.0 (Octubre, 2012)

- Primera versión bajo el titulo 'Mastering Plone' para la Ploneconf 2012, Arnhem [pbauer, do3cc]

0.2 Octubre 2011

- Ampliado como el Tutorial Plone para la PyCon De 2011, Leipzig [pbauer]

0.1 (Octubre 2009)

- Partes iniciales creadas para el Plone-Einsteigerkurs (http://www.plone.de/trainings/einsteiger-kurs/kursuebersicht) [pbauer, hellfish2]

CAPÍTULO 40

Mapa de ruta

1.3

Agregar nuevo capítulos:

- Paquete z3c.forms (incluyendo paneles de control y el paquete plone.app.registry).
- Testing (incluyendo robot tests).
- Relaciones (incluyendo widgets, indexes personalizados).
- Seguridad (incluyendo permisos personalizados y roles).

Todo

Todos son mantenidos como incidencias en el repositorio github.