

MASSIMO DI PIERRO

WEB2PY MANUAL

3RD EDITION ALPHA SPANISH (MACHINE TRANSLATED)

MDP PRESS

Copyright 2008, 2009, 2010 by Massimo Di Pierro. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600, or on the web at www.copyright.com. Requests to the Copyright owner for permission should be addressed to:

Massimo Di Pierro
School of Computing
DePaul University
243 S Wabash Ave
Chicago, IL 60604 (USA)
Email: mdipierro@cs.depaul.edu

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

Library of Congress Cataloging-in-Publication Data:

ISBN-0000

Printed in the United States of America.

To my family

Contents

1	Introducción	27
1.1	Principios	30
1.2	Web Marcos	31
1.3	Modelo-Vista-Controlador	33
1.4	¿Por qué web2py	37
1.5	Seguridad	39
1.6	En el cuadro de	41
1.7	Licencia	43
1.8	Agradecimientos	45
1.9	Acerca de este libro	47
1.10	Elementos de Estilo	49

2 El lenguaje Python	51
2.1 Acerca de Python	51
2.2 Puesta en marcha	52
2.3 ayuda, dir	53
2.4 Tipos	54
2.4.1 str	55
2.4.2 lista	56
2.4.3 tupla	57
2.4.4 dict	58
2.5 Acerca de sangría	60
2.6 de... en	60
2.7 mientras que	62
2.8 def... regresar	62
2.9 si... elif... else	64
2.10 intenta... excepto... si no... por fin	64
2.11 clase	67
2.12 Características especiales, métodos y operadores	68
2.13 Archivo de entrada / salida	69

2.14 lambda	70
2.15 exec, eval	72
2.16 importación	73
2.16.1 os	74
2.16.2 sistemas	75
2.16.3 fecha y hora	75
2.16.4 tiempo	76
2.16.5 cPickle	77
3 Información general	79
3.1 Inicio	79
3.2 Di Hola	84
3.3 Vamos a contar	89
3.4 Say My Name	91
3.5 Devoluciones de datos	93
3.6 Una imagen de blog	96
3.7 Agregar CRUD	112
3.8 Adición de autenticación	114

3.9	Configuring el diseño	116
3.10	Un Wiki	116
3.11	Más en admin	129
3.11.1	<i>site</i>	129
3.11.2	<i>about</i>	132
3.11.3	<i>edit</i>	133
3.11.4	<i>errors</i>	136
3.11.5	<i>Mercurial</i>	140
3.12	Más en appadmin	140
4	El Núcleo	143
4.1	Opciones de línea de comandos	143
4.2	Despacho	147
4.3	Bibliotecas	151
4.4	Aplicaciones	156
4.5	API	158
4.6	solicitud	160
4.7	respuesta	164

4.8	sesión	167
4.9	cache	169
4.10	URL	171
4.11	HTTP y redirigir	174
4.12	T e Internacionalización	176
4.13	Galletas	178
4.14	Aplicación init	179
4.15	URL Rewrite	180
4.16	Rutas a un error	184
4.17	Cron	184
4.18	Antecedentes Los procesos y las colas de tareas	188
4.19	Tercera Parte módulos	189
4.20	Entorno de ejecución	191
4.21	Cooperación	193
4.22	WSGI	194
4.22.1	Exteriores Middleware	195
5	Las Vistas	197

5.1	Sintaxis básica	200
5.1.1	de... en	200
5.1.2	mientras que	201
5.1.3	si... elif... else	201
5.1.4	intentar otra cosa... excepto..... finalmente	202
5.1.5	def... regresar	203
5.2	Ayudantes del HTML	204
5.2.1	XML	205
5.2.2	Construido en Ayudantes	207
5.3	EMBELLECER	221
5.4	Del lado del servidor DOM y Analizar	221
5.5	Diseño de página	224
5.5.1	Diseño de página predeterminado	226
5.5.2	Personalizar el diseño predeterminado	229
5.6	Las funciones en Vista	231
5.7	Bloques de Vista	232
5.8	Usando el sistema de plantillas para generar mensajes de correo electrónico	233

6 La Capa de abstracción de base de datos	235
6.1 Dependencias	235
6.2 Cadenas de conexión	237
6.3 Palabras clave reservadas	240
6.4 DAL, de mesa, campo	241
6.5 Registro de Representación	242
6.6 Migraciones	248
6.7 La fijación de Migraciones Roto	249
6.8 insertar	250
6.9 ejecuta o deshace	251
6.10 Raw SQL	252
6.11 caída	253
6.12 Índices	253
6.13 Legado de bases de datos y tablas con llave	254
6.14 De transacciones distribuidas	255
6.15 Manual de Cargas	256
6.16 Consulta, Set, Filas	256
6.17 seleccionar	257

6.18	Computarizada Campos	268
6.19	Campos Virtuales	269
6.20	Una relación de muchos	271
6.21	Muchos a muchos	275
6.22	Otros operadores	276
6.23	La generación de SQL primas	279
6.24	Exportación e importación de datos	280
6.25	Selecciona el almacenamiento en caché	286
6.26	Auto-Referencia y alias	286
6.27	Cuadro Herencia	288
7	Los formularios y validadores	291
7.1	FORMA	292
7.1.1	Ocultos campos	296
7.1.2	keepvalues	298
7.1.3	onvalidation	298
7.1.4	Formularios y redirección	299
7.1.5	Múltiples formas por página	300

7.1.6	Compartiendo las formas	301
7.2	SQLFORM	302
7.2.1	SQLFORM e insertar, actualizar o borrar	307
7.2.2	SQLFORM en HTML	309
7.2.3	SQLFORM y Cargas	310
7.2.4	Almacenar el nombre de archivo original	313
7.2.5	autodelete	314
7.2.6	Enlaces a los expedientes de referencia a	314
7.2.7	Pre-llenar el formulario	316
7.2.8	SQLFORM sin base de datos de E / S	317
7.3	SQLFORM.factory	318
7.4	CRUD	319
7.4.1	Configuración	321
7.4.2	Mensajes	324
7.4.3	Métodos	325
7.4.4	Registro de control de versiones	327
7.5	Formularios personalizados	328
7.5.1	CSS convenios	330

7.5.2	Ocultar los errores	331
7.6	Validadores	332
7.6.1	Validadores	333
7.6.2	Base de datos de los validadores	346
7.6.3	Validadores personalizados	348
7.6.4	Validadores con dependencias	350
7.7	Reproductores	350
7.7.1	Autocompletar Widget	352
8	Control de Acceso	355
8.1	Autenticación	357
8.1.1	Restricciones en materia de registro	361
8.1.2	Integración con OpenID, Facebook, etc	361
8.1.3	CAPTCHA y reCAPTCHA	364
8.1.4	Personalización de la autenticación	365
8.1.5	Cambiar el nombre de autenticación tablas	366
8.1.6	Otros métodos sesión y formularios de ingreso	367
8.2	Auth y correo electrónico	373

8.2.1	Correo electrónico de depuración	374
8.2.2	Mensajes de correo electrónico forma Google App Engine	375
8.2.3	Más ejemplos de correo electrónico	375
8.2.4	x509 y el cifrado PGP	376
8.3	Autorización	377
8.3.1	Decoradores	379
8.3.2	La combinación de Requerimientos	381
8.3.3	Autorización y CRUD	381
8.3.4	Autorización y descargas	383
8.3.5	Control de acceso y autenticación básica	383
8.3.6	Configuración y mensajes	384
8.4	Servicio Central de autenticación	390
9	Servicios	395
9.1	Representación de un diccionario	395
9.1.1	HTML, XML y JSON	396
9.1.2	Genérico Vistas	397
9.1.3	Representación de Filas	398

9.1.4	Formatos personalizados	399
9.1.5	RSS	399
9.1.6	CSV	402
9.2	Pide a procedimiento remoto	403
9.2.1	XMLRPC	405
9.2.2	JSONRPC y Pijamas	407
9.2.3	AMFRPC	411
9.2.4	SOAP	414
9.3	Bajo nivel de API y otras recetas	414
9.3.1	simplejson	415
9.3.2	PyRTF	416
9.3.3	ReportLab y PDF	417
9.4	Servicios y autenticación	418
10	Ajax Recetas	421
10.1	web2py_ajax.html	421
10.2	Efectos jQuery	426
10.2.1	Campos condicional en las formas	430

10.2.2 Confirmación en Suprimir	432
10.3 La función de ajax	433
10.3.1 Eval objetivo	434
10.3.2 Auto-realización	435
10.3.3 Formulario de Ajax	438
10.3.4 Voto y Valoración	439
11 Recetas de implementación	443
11.1 Linux / Unix	447
11.1.1 Un paso de implementación de Producción	447
11.1.2 Apache configuración	447
11.1.3 mod_wsgi	449
11.1.4 mod_wsgi y SSL	452
11.1.5 mod_proxy	454
11.1.6 Inicio como Daemon Linux	456
11.1.7 Lighttpd	457
11.1.8 Alojamiento Compartido con mod_python	459
11.1.9 Cherokee con FastGGI	460

11.1.10 PostgreSQL	462
11.2 Windows	463
11.2.1 Apache y mod_wsgi	464
11.2.2 Inicio como servicio de Windows	467
11.3 Asegurar Sesiones y admin	468
11.4 Eficiencia y Escalabilidad	469
11.4.1 Eficiencia trucos	470
11.4.2 Sesiones en base de datos	471
11.4.3 HAProxy una alta disponibilidad de equilibrador de carga	472
11.4.4 Limpieza de Sesiones	474
11.4.5 Cargar archivos en la base de datos	475
11.4.6 Recogida de entradas	476
11.4.7 Memcache	478
11.4.8 Sesiones en Memcache	479
11.4.9 Eliminación de aplicaciones	479
11.4.10 Uso de bases de datos replicadas Servidores	479
11.5 Google App Engine	481

11.5.1 Configuración	482
11.5.2 Ejecución e implementación	484
11.5.3 Configuración del controlador	485
11.5.4 Evite el sistema de archivos	486
11.5.5 Memcache	487
11.5.6 Problemas de base de datos	487
11.5.7 JUEGO y HTTPS	488
12 Otras recetas	489
12.1 Actualizar	489
12.2 Cómo distribuir sus aplicaciones como Binarios	490
12.3 Obtención de una dirección URL externa	491
12.4 Bastante fechas	491
12.5 Geocodificación	491
12.6 Paginación	492
12.7 httpserver.log y el formato de archivo de registro	493
12.8 Poblar la base de datos con un maniquí de datos	495
12.9 Enviar un SMS	496

12.10 A aceptar pagos con tarjeta de crédito	496
12.11 Twitter API	498
12.12 Streaming de archivos virtual	499
12.13 Jython	500
13 Componentes y Plugins	503
13.1 Componentes	504
13.2 Plugins	511
13.2.1 Componente Plugins	515
13.2.2 Plugin Manager	517
13.2.3 Diseño de plugins	519
13.3 plugin_wiki	520
13.3.1 Markmin sintaxis	523
13.3.2 Page Permisos	526
13.3.3 Páginas especiales	526
13.3.4 Configuración plugin_wiki	528
13.3.5 Reproductores actual	529
13.3.6 Ampliación de Widgets	535

Bibliography**539**

Preface

Estoy muy preocupado por el crecimiento de la brecha digital y los problemas que ello supone para el libre flujo de información. La capacidad de construir buenas aplicaciones web de calidad, rápida y eficaz es de importancia crítica para permitir que más personas interactuar con Internet de una manera no trivial y prevenir gran jugadear de monopolizar la información. Por eso comencé a trabajar en web2py.

En general, los efectos de cualquier framework de desarrollo web es hacer el desarrollo web más fácil, más rápido y evitar que los desarrolladores de cometer errores (en particular, en asuntos relacionados con la seguridad). En web2py que solucionemos estos problemas:

Ease of use si el objetivo primordial de web2py. Para nosotros esto significa la reducción del aprendizaje y el tiempo de implementación. Por eso hemos construido web2py como completo framework sin dependencias. No requiere instalación y no tiene ningún archivo de configuración. Se incluye todo lo necesario para que se inicie como un servidor web rápido multi-hilo (aunque apoyamos los terceros también), una transacción de base de datos de seguridad (apoyamos muchos los terceros también), una potente capa de abstracción de base de datos, y una web- del IDE que le permite acceder a todas las funcionalidades web2py. También hemos reducido el API al mínimo (12 objetos del núcleo solamente) así que es poco para que usted pueda memorizar. desarrollo *Faster* es el objetivo secundario para web2py. Por esta razón

hemos creado web2py de modo que cada función tiene un comportamiento por defecto (que puede ser anulado). Por ejemplo, si usted acaba de crear un modelo (es decir, elegir la forma en la base de datos debe almacenar los datos), web2py crea automáticamente una base de datos basados en la web la interfaz de administración a éste, si se agregan acciones del controlador (la lógica de su aplicación), automáticamente tiene por defecto puntos de vista presenta la salida en HTML, XML, JSON, RSS, etc web2py también genera automáticamente crear / modificar / suprimir las formas que se pueden personalizar en su punto de vista (la capa de presentación de su aplicación).

Security es el último pero no menos importante de las metas de web2py. web2py deja muy poca libertad a los desarrolladores en asuntos relacionados con la seguridad. La base de datos Capa de abstracción impide Inyecciones SQL. La plantilla del idioma, impiden Cross Site Scripting vulnerabilidades. Los formularios generados por ofrecer web2py validación de campo y el bloque de Cross Site Request falsificaciones. Contraseña siempre se almacenan encriptados. Las sesiones son almacenadas en el servidor de forma predeterminada para impedir Cookie Manipulaciones y cookies de sesión son uuid para prevenir robo de cookies.

Internamente, web2py tiene un diseño de arriba hacia abajo frente a la concepción de abajo hacia arriba de otros marcos. No se construye mediante la adición de una capa sobre otra. Se construye desde la perspectiva del usuario y se ha optimizado constantemente en el interior para llegar a ser más rápido y ágil, manteniendo siempre *backward compatibility*. Estoy feliz de decir que web2py es hoy uno de los marcos web más rápido y también uno de los más pequeños. Más de 50 personas han contribuido con código y parches para hacer web2py lo que es hoy y todos trabajamos constantemente para que sea aún mejor.

web2py es gratuito para su uso. Si usted se beneficia de ella, esperamos que se sienta un poco más como contribución a la sociedad en cualquier forma que usted elija.

1

Introducción

web2py (1) es una web de código abierto libre de marco para el desarrollo ágil de aplicaciones basadas en web seguras bases de datos, sino que está escrito en Python (2) y programable en Python. web2py es un completo framework, lo que significa que contiene todos los componentes necesarios para construir plenamente las aplicaciones web funcional. web2py está diseñado para guiar a un desarrollador web a aplicar buenas prácticas de ingeniería de software, tales como el uso del Modelo Vista Controlador (MVC). web2py separa la representación de datos (el modelo) a partir de la presentación de datos (la vista) y también de la lógica de la aplicación y flujo de trabajo (el controlador). web2py proporciona bibliotecas para ayudar a los desarrolladores diseñar, implementar y probar cada una de estas tres partes por separado, y los hace trabajar juntos. web2py se construye para la seguridad. Esto significa que automáticamente resuelve muchos de los problemas que pueden dar lugar a vulnerabilidades de seguridad, siguiendo las prácticas establecidas así. Por ejemplo, valida todas las entradas (para evitar las inyecciones), escapa a toda la producción (para evitar-site scripting de cruz), cambia el nombre de los archivos cargados (para prevenir ataques de directorio transversal). Web2py hojas de solicitud a la elección de los desarrolladores de poco en los asuntos relacionados con la seguridad. web2py incluye una base de datos Capa de abstracción (DAL) que escribe SQL (3) dinámi-

camente para que usted, el desarrollador no tiene que hacerlo. El DAL sabe cómo generar SQL de forma transparente para SQLite (4), MySQL (6), PostgreSQL (5), MSSQL (7), FireBird (8), Oracle (9), IBM DB2 (10), Informix (11) y Ingres (12). El DAL también pueden generar llamadas a funciones para Google BigTable cuando se ejecuta en el Google App Engine (GAE) (13). Una vez que una o más tablas de bases de datos están definidos, web2py también genera una base de datos de administración de la web la interfaz totalmente funcional basado para acceder a la base de datos y las tablas. web2py difiere de otros frameworks web que es el único marco para abrazar plenamente el paradigma Web 2.0, donde la red es el ordenador. De hecho, web2py no requiere instalación o configuración, sino que funciona en cualquier arquitectura que se puede ejecutar Python (Windows, Windows CE, Mac OS X, iPhone, y Unix / Linux), y el desarrollo, despliegue y las fases de mantenimiento para las aplicaciones se puede hacer mediante una interfaz web local o remoto. web2py se ejecuta con CPython (la aplicación C) y Jython (la aplicación Java), las versiones de 2,4, 2,5 y 2,6 aunque el apoyo de sólo el 2,5 "officially" otra cosa que no puede garantizar la compatibilidad para aplicaciones. web2py proporciona un sistema de billeteaje. Si ocurre un error, los billetes se expiden para el usuario, y el error se registra para el administrador. web2py es de código abierto y liberado bajo la licencia GPL2.0, pero las aplicaciones desarrolladas con web2py no están sujetos a ninguna restricción de licencia. De hecho, siempre y cuando no contengan el código fuente web2py, no se consideran "derivative work". web2py también permite al desarrollador bytecode-compilar aplicaciones y distribuirlos como de código cerrado, aunque se requieren para ejecutar web2py. La licencia web2py incluye una excepción que permite a los desarrolladores web para enviar sus productos con originales compilados web2py binarios-pre, sin el acompañamiento código fuente.

Otra característica de web2py, es que nosotros, sus desarrolladores, se comprometen a mantener la compatibilidad en futuras versiones. Lo hemos hecho desde la primera versión de web2py en octubre de 2007. Las nuevas funciones se han añadido y los errores se han corregido, pero si un programa trabajó con web2py 1,0, el programa seguirá funcionando en la actualidad.

Éstos son algunos ejemplos de consejos de web2py que ilustran su potencia y simplicidad. El siguiente código:

```
1 db.define_table('person', Field('name'), Field('image', 'upload'))
```

crea una tabla de base de datos llamada, algo "person" with two fields: "name", a string; and "image" que necesita ser subido (la imagen real). Si la tabla ya existe, pero no coincide con esta definición, se altera adecuadamente.

Dada la tabla definida anteriormente, el siguiente código:

```
1 form = crud.create(db.person)
```

crea un formulario de inserción de esta tabla que permite a los usuarios subir imágenes. También valida un formulario presentado, cambia el nombre de la imagen cargada en forma segura, las tiendas de la imagen en un archivo, inserta el correspondiente registro en la base de datos, impide la presentación de matrimonio, y, finalmente, modifica la forma misma mediante la adición de mensajes de error si los datos presentados por el usuario no pasa la validación.

El siguiente código:

```
1 @auth.requires_permission('read', 'person')
2 def f(): ....
```

impide que los visitantes accedan a la `f` funcionará a menos que el visitante es un miembro de un grupo cuyos miembros tienen permisos para "read" records of table "person". Si el visitante no está conectado, él se dirige a una página de entrada (siempre por defecto por web2py).

El siguiente código incorpora un elemento de página.

```
1 {{=LOAD('other_controller','function.load',ajax=True, ajax_trap=True)}}
```

Permite web2py para cargar en una vista de los contenidos generados por la función other_controller (funciona con cualquier función). Se carga el con-

tenido a través de Ajax, se incrusta en la página actual (utilizando el diseño actual y no el diseño de la función other_controller), las trampas de todos los formularios que figuran en el contenido cargado de tal modo que también son enviados a través de Ajax sin recargar la página. También puede cargar contenido de las aplicaciones no web2py.

El ayudante de carga permite muy diseño modular de aplicaciones y se discute con cierto detalle en el último capítulo de este libro.

1.1 *Principios*

de programación Python típicamente sigue estos principios básicos:

- No te repitas (DRY).
- Debe haber una sola manera de hacer las cosas.
- Explícito es mejor que implícito.

web2py plenamente abarca los dos primeros principios, forzando al desarrollador utilizar buenas prácticas de ingeniería de software que desalientan la repetición de código. guías web2py el desarrollador a través de casi todas las tareas comunes en el desarrollo de aplicaciones web (la creación y el procesamiento de formularios, gestión de sesiones, cookies, errores, etc.) web2py se diferencia de otros marcos de relación con el tercer principio, que a veces entra en conflicto con los otros dos. En particular, web2py no importa, pero las aplicaciones de usuario los ejecuta en un contexto predefinido. Este contexto expone las palabras clave de Python, así como las palabras clave web2py.

Para algunos esto puede parecer magia, pero no debería. En la práctica sólo algunos módulos ya se han importado sin ti hacerlo. Web2py está tratando de evitar la característica molesta de otros marcos de esa fuerza al desarrollador a importar los mismos módulos en la parte superior de la cada modelo

y el controlador. web2py, mediante la importación de sus propios módulos, ahorra tiempo y evita errores, siguiendo así el espíritu de "don't repeat yourself" y "there should be only one way of doing things".

Si el desarrollador desea utilizar estos módulos de Python o de terceros módulos, los módulos deben ser importados de forma explícita, como en cualquier otro programa de Python.

1.2 Web Marcos

En su nivel más fundamental, una aplicación web se compone de un conjunto de programas (o funciones) que se ejecutan cuando el URL correspondiente es visitado. La salida del programa se devuelve al visitante y prestados por el navegador.

El propósito de los marcos Web es permitir a los desarrolladores crear nueva aplicación rápida, fácil y sin hacer errores. Hecho esto probando APIs y herramientas que reducir y simplificar la cantidad de código que se requiere.

Los dos enfoques clásicos para el desarrollo de aplicaciones web son:

- Generación de HTML (14; 15) programación.
- Incrustación de código en páginas HTML.

El primer modelo es el seguido, por ejemplo, a principios de scripts CGI. El segundo modelo es seguido, por ejemplo, PHP (16) (si el código en PHP, uno-como el lenguaje C), ASP (donde el código está en Visual Basic), y JSP (si el código en Java).

Aquí presentamos un ejemplo de un programa PHP que, cuando se ejecuta, recupera datos de una base de datos y devuelve una página HTML que muestra los registros seleccionados:

```

1 <html><body><h1>Records</h1><?
2   mysql_connect(localhost,username,password);
3   @mysql_select_db(database) or die( "Unable to select database");
4   $query="SELECT * FROM contacts";
5   $result=mysql_query($query);
6   mysql_close();
7   $i=0;
8   while ($i < mysql_numrows($result)) {
9     $name=mysql_result($result,$i,"name");
10    $phone=mysql_result($result,$i,"phone");
11    echo "<b>$name</b><br>Phone:$phone<br /><br /><hr /><br />";
12    $i++;
13  }
14 ?></body></html>

```

El problema con este enfoque es que el código está embebido en páginas HTML, pero este mismo código muy También es necesario para generar ingresos adicionales HTML y generar las instrucciones SQL para consultar la base de datos, de enredo de múltiples capas de la aplicación y lo que hace difícil de leer y mantener. La situación es aún peor para las aplicaciones Ajax y la complejidad aumenta con el número de páginas (archivos) que componen la aplicación.

La funcionalidad del ejemplo anterior se puede expresar en web2py con dos líneas de código Python:

```

1 def index():
2   return HTML(BODY(H1('Records'), db().select(db.contacts.ALL)))

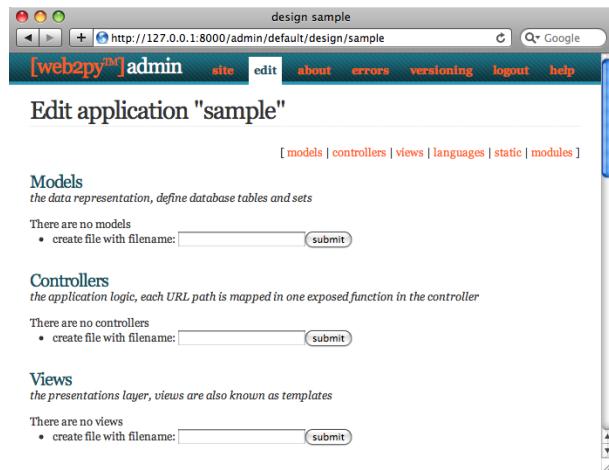
```

En este sencillo ejemplo, la estructura de la página HTML está representado por la programación, HTML, BODY y H1 objetos, el db de base de datos se consulta por el comando select, finalmente, todo se serializa en HTML. Tenga en cuenta que db no es una palabra clave, pero un usuario variable definida. Vamos a utilizar ese nombre constantemente para referirse a una conexión de base de datos para evitar confusiones. marcos Web suelen ser catalogados como uno de dos tipos: un marco "glued" se construye mediante el ensamblaje (pegando) varias partes componentes de otros. Un marco "full-stack" se construye mediante la creación de componentes diseñados específicamente para trabajar juntos y estar bien integrado. web2py es un completo frame-

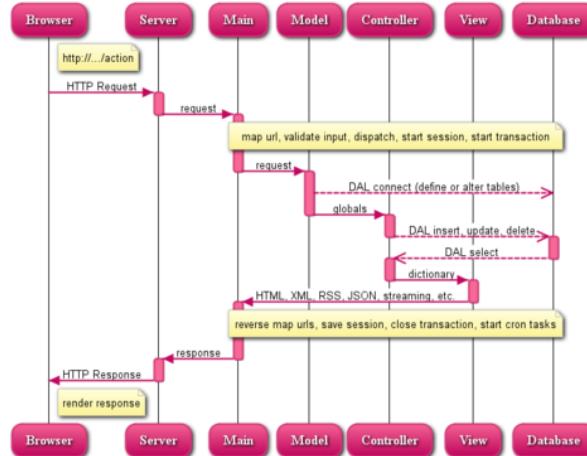
work. Actualmente casi todos sus componentes se construyen de cero y diseñados para trabajar conjuntamente, pero funcionan tan bien fuera del marco web2py completo. Por ejemplo, la capa de abstracción de base de datos (DAL) o del lenguaje del template se puede utilizar independientemente del marco web2py mediante la importación de `gluon.sql` o `gluon.template` en sus propias aplicaciones Python. `gluon` es el nombre de la carpeta que contiene web2py bibliotecas del sistema. Algunas bibliotecas web2py, como la construcción y el procesamiento de formularios de las tablas de base de datos, tienen dependencias en otras partes de web2py. web2py también puede trabajar con el partido de Python bibliotecas de sesiones, incluidos otros idiomas y Dals plantilla, pero no será tan bien integrado como los componentes originales.

1.3 *Modelo-Vista-Controlador*

fuerzas web2py el promotor a la representación por separado los datos (el modelo), la presentación de datos (la vista) y el flujo de trabajo de aplicación (el controlador). Consideremos de nuevo el ejemplo anterior y ver la forma de construir una aplicación web2py a su alrededor. He aquí un ejemplo de la interfaz de edición web2py MVC:



El flujo de trabajo típico de una petición de web2py se describe en el siguiente diagrama:



En el diagrama:

- El servidor puede ser el web2py incorporada en el servidor web o un servidor de terceros, tales como Apache. El servidor maneja multi-threading.

- "main" es la principal aplicación WSGI. Realiza todas las tareas comunes y envuelve las aplicaciones de usuario. Se trata de cookies, sesiones, transacciones, url de enrutamiento y enrutamiento inverso, de despacho. Puede servir y escuchar los archivos estáticos si el servidor web no está haciendo ya.
- Los Modelos, Vistas y componentes del controlador constará de la solicitud del usuario.
- No puede haber múltiples aplicaciones alojadas en la instancia web2py misma.
- Las líneas punteadas representan la comunicación con el motor de base de datos (o motores). Las consultas de base de datos se puede escribir en SQL primas (desaconsejado) o utilizando la base de datos web2py Capa de abstracción (recomendado), de modo que el código de aplicación web2py no depende de que el motor de base de datos específica.
- Los mapas despachador de la dirección URL solicitada en una llamada a la función en el controlador. La salida de la función puede ser una cadena o un diccionario de símbolos (una tabla hash). Los datos en el diccionario se representa por un punto de vista. Si el visitante solicita una página HTML (por defecto), el diccionario se representa en una página HTML. Si el visitante solicita la misma página en XML, web2py trata de encontrar una visión que puede hacer que el diccionario en formato XML. El desarrollador puede crear vistas para hacer páginas en cualquiera de los protocolos que ya soportados (HTML, XML, JSON, RSS, CSV, RTF) o de protocolos adicionales a medida.
- Todas las llamadas se envuelven en una transacción, así como cualquier excepción no detectada hace que la transacción no pasaran. Si la solicitud se realiza correctamente, la transacción se confirma.
- web2py también se ocupa de las sesiones y cookies de sesión de forma automática, y cuando una transacción se confirma, la sesión también se almacena, a menos que se especifique lo contrario.
- Es posible registrar las tareas periódicas (cron) para ejecutarse a horas fijas y / o después de la realización de determinadas acciones. De esta manera es posible a largo plazo y calcular intensivos tareas en segundo plano sin ralentizar la navegación.

Ésta es una aplicación MVC mínima y completa que consta de tres archivos:

"db.py" es el modelo:

```

1 db = DAL('sqlite://storage.sqlite')
2 db.define_table('contacts',
3     Field('name'),
4     Field('phone'))
```

Se conecta a la base de datos (en este ejemplo una base de datos SQLite almacenada en el archivo `storage.sqlite`) y define una tabla llamada `contacts`. Si la tabla no existe, web2py la crea y, de forma transparente y en el fondo, genera código SQL en el dialecto SQL adecuado para el motor de base de datos específica utilizada. El desarrollador puede ver el SQL generado, pero no es necesario cambiar el código si el base de datos de back-end, que por defecto a SQLite, se sustituye con MySQL, PostgreSQL, MSSQL, Firebird, Oracle, DB2, Informix, Interbase, Ingres, o Big Tablas de Google en la App Engine de Google.

Una vez que una tabla está definida y creada, web2py también genera una base de datos de administración de la web la interfaz totalmente funcional basado para acceder a la base de datos y las tablas. Se llama **appadmin**.

"default.py" es el controlador:

```

1 def contacts():
2     return dict(records=db().select(db.contacts.ALL))
```

En web2py, las direcciones URL se asignan a los módulos de Python y llamadas a funciones. En este caso, el controlador contiene una sola función (o "action") `contacts` llamado. Una acción puede devolver una cadena (el sitio web devolverán) o un diccionario de Python (un conjunto de pares `key:value`). Si la función devuelve un diccionario, se pasa a una vista con el mismo nombre que el controlador / función, que a su vez, se hace. En este ejemplo, la función realiza una `contacts select` base de datos y devuelve los registros resultantes como un valor asociado a la `records` clave de diccionario.

"default/contacts.html" es el panorama:

```

1 {{extend 'layout.html'}}
2 <h1>Records</h1>
3 {{for record in records:}}
4 {{=record.name}}: {{=record.phone}}<br />
5 {{pass}}

```

Este punto de vista es llamado automáticamente por web2py después de la función del controlador asociado (acción) es ejecutado. El propósito de este punto de vista es hacer que las variables en el diccionario records=... volvió a HTML. La vista de archivo está escrito en HTML, pero incrusta código Python delimitado por el {{ y }} delimitadores especiales. Esto es muy diferente del ejemplo de código PHP, ya que el único código incrustado en el código HTML es el archivo "presentation layer" code. The "layout.html" se hace referencia en la parte superior de la vista es proporcionada por web2py y constituye la estructura básica para todas las aplicaciones web2py. El archivo de diseño puede ser fácilmente modificado o reemplazado.

1.4 ¿Por qué web2py

web2py es uno de los muchos marcos de aplicaciones web, pero tiene características únicas y convincentes. web2py se desarrolló originalmente como una herramienta de enseñanza, con el primario siguientes motivaciones:

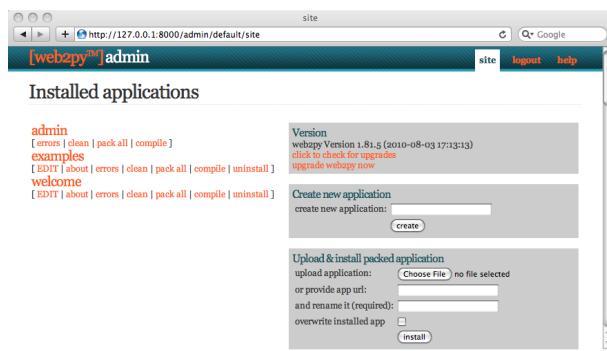
- Fácil para los usuarios a aprender sobre el desarrollo del lado del servidor web sin comprometer la funcionalidad. Por esta razón web2py no requiere instalación, sin necesidad de configuración, no tiene dependencias (excepto para la distribución de código fuente, que requiere Python 2.5 (o posterior 2.x) y sus módulos de biblioteca estándar), y expone la mayor parte de su funcionalidad a través de una interfaz web.
- web2py se ha mantenido estable desde el primer día, dado que sigue un diseño de arriba hacia abajo, es decir, su API fue diseñado antes de su ejecución. A pesar de que la nueva funcionalidad se ha añadido, nunca ha

roto web2py compatibilidad hacia atrás, y no se romperá la compatibilidad cuando se añade una funcionalidad adicional en el futuro.

- web2py forma pro activa, los problemas de seguridad más importantes que aquejan muchas aplicaciones web modernas, según lo determinado por OWASP (19) a continuación.
- web2py es luz. Sus bibliotecas centrales, incluyendo la capa de abstracción de bases de datos, la plantilla del lenguaje y la totalidad del importe de ayuda para que 300 KB. La fuente de todo el código incluyendo aplicaciones muestra y las cantidades imágenes a 2.0MB.
- web2py ocupa poco espacio y es muy rápido. Utiliza el servidor Rocket (22) WSGI Web desarrollada por Timothy Farrell. Es 30% más rápido que Apache con mod_proxy. Nuestras pruebas también indican que, en un promedio de PC, que sirve una página dinámica media sin acceso de base de datos en aproximadamente 10 ms. La DAL ha generales muy bajos, por lo general inferior al 3%.

(17; 18) WSGI significa servidor Web Gateway Interface. Es un emergente estándar de Python para la comunicación entre un servidor web y aplicaciones Python).

Aquí hay una captura de pantalla de la interfaz principal de **admin** web2py:



1.5 Seguridad

El Abierto de aplicaciones Web Project (19) Seguridad (OWASP) es una comunidad mundial de abierta y libre se centró en la mejora de la seguridad del software de aplicación.

OWASP ha incluido entre los diez principales temas de seguridad que ponen en riesgo las aplicaciones web. Esta lista se reproduce aquí, junto con una descripción de cómo cada tema es abordado por web2py:

- web2py "Cross Site Scripting (XSS): XSS flaws occur whenever an application takes user supplied data and sends it to a web browser without first validating or encoding that content. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface web sites, possibly introduce worms, etc.", por defecto, escapa a todas las variables representan en el punto de vista, la prevención de XSS.
- web2py "Injection Flaws: Injection flaws, particularly SQL injection, are common in web applications. Injection occurs when user-supplied data is sent to an interpreter as part of a command or query. The attacker's hostile data tricks the interpreter into executing unintended commands or changing data." incluye una capa de abstracción de base de datos que hace posible inyección de código SQL. Normalmente, los estados de SQL no están escritas por el desarrollador. En su lugar, SQL se genera dinámicamente por el DAL, asegurando que todos los datos insertados correctamente escapado.
- web2py "Malicious File Execution: Code vulnerable to remote file inclusion (RFI) allows attackers to include hostile code and data, resulting in devastating attacks, such as total server compromise." permite que las funciones sólo se expone a ser ejecutado, la prevención de la ejecución de archivos maliciosos funciones importado nunca son expuestos; acciones sólo están expuestos. web2py la interfaz de administración basada en web hace que sea muy fácil hacer un seguimiento de lo que se expone y lo que no.

- web2py "Insecure Direct Object Reference: A direct object reference occurs when a developer exposes a reference to an internal implementation object, such as a file, directory, database record, or key, as a URL or form parameter. Attackers can manipulate those references to access other objects without authorization." no expone todos los objetos internos y, además, web2py valida todas las direcciones URL, con lo que la prevención de ataques de directorio transversal. web2py también proporciona un mecanismo simple para crear formularios que validar automáticamente todos los valores de entrada.
- "Cross Site Request Forgery (CSRF): A CSRF attack forces a logged-on victim's browser to send a pre-authenticated request to a vulnerable web application, which then forces the victim's browser to perform a hostile action to the benefit of the attacker. CSRF can be as powerful as the web application that it attacks". web2py impide CSRF, así como la presentación de las formas accidentales doble mediante la asignación de una muestra al azar de una sola vez a cada formulario. Por otra parte web2py utiliza UUID para cookie de sesión.
- "Information Leakage and Improper Error Handling: Applications can unintentionally leak information about their configuration, internal workings, or violate privacy through a variety of application problems. Attackers use this weakness to steal sensitive data, or conduct more serious attacks." web2py incluye un sistema de billeteo. Ningún error puede resultar en código que se está expuesto a los usuarios. Todos los errores son registrados y los billetes se expiden para el usuario que permite el seguimiento de error. Errores y códigos fuente, son accesibles sólo para el administrador.
- "Broken Authentication and Session Management: Account credentials and session tokens are often not properly protected. Attackers compromise passwords, keys, or authentication tokens to assume other users' identities." web2py provides a built-in mechanism for administrator authentication, and it manages sessions independently for each application. The administrative interface also forces the use of secure session cookies when the client is not "localhost". Para las aplicaciones, que incluye un potente control de acceso basado en roles API.

- web2py "Insecure Cryptographic Storage: Web applications rarely use cryptographic functions properly to protect data and credentials. Attackers use weakly protected data to conduct identity theft and other crimes, such as credit card fraud." utiliza el MD5 o HMAC la + algoritmos hash SHA-512 para proteger las contraseñas almacenadas. Otros algoritmos están también disponibles.
- web2py "Insecure Communications: Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications." incluye el SSL enabled (21) Rocket WSGI servidor, pero también puede usar Apache o Lighttpd y mod_ssl para proporcionar cifrado SSL de las comunicaciones.
- "Failure to Restrict URL Access: Frequently an application only protects sensitive functionality by preventing the display of links or URLs to unauthorized users. Attackers can use this weakness to access and perform unauthorized operations by accessing those URLs directly." mapas web2py solicitudes de URL a los módulos de Python y funciones. web2py proporciona un mecanismo para que especifique cuáles son las funciones son públicas y que requieren autenticación y autorización. El papel incluye control de acceso basado API permite a los desarrolladores para restringir el acceso a cualquier función sobre la base de entrada, permisos de pertenencia a grupos o colectivos a partir. Los permisos son muy granulares y se puede combinar con CRUD para permitir, por ejemplo, para dar acceso a tablas específicas y / o registros.

web2py fue revisado para la seguridad y usted puede encontrar el resultado de la revisión a ref. (20).

1.6 En el cuadro de

Puede descargar web2py desde el sitio web oficial:

¹ <http://www.web2py.com>

web2py se compone de los siguientes componentes:

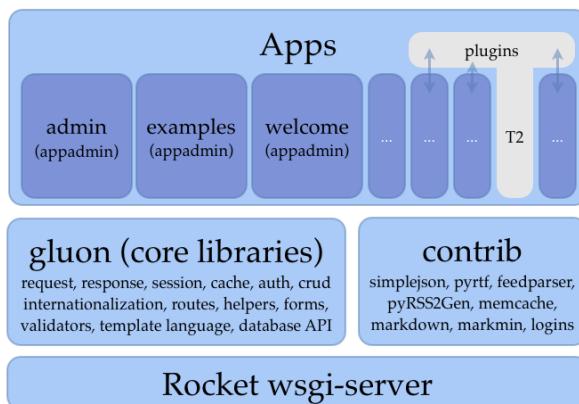
- **libraries**: proporcionar funcionalidad básica del web2py y son accesibles mediante programación.
- **web server**: el servidor Rocket WSGI web.
- la aplicación **admin**: se utiliza para crear, diseñar y administrar otras aplicaciones web2py. **admin** ofrecen una completa basada en la web Entorno de desarrollo integrado (IDE) para la creación de aplicaciones web2py. También incluye otras funcionalidades, como la prueba basada en web y un shell basado en la web.
- la aplicación **examples**: contiene documentación y ejemplos interactivos. **examples** es un clon de la página web web2py oficiales, e incluye la documentación Epydoc y la Esfinge.
- la aplicación **welcome**: el andamio plantilla básica para cualquier otra aplicación. Por defecto se incluye un menú en cascada CSS puro y autenticación de usuarios (capítulo 8).

web2py se distribuye en el código fuente como en binario para Microsoft Windows y para Mac OS X.

La distribución de código fuente puede ser utilizado en cualquier plataforma en la que Python ejecuta e incluye los componentes mencionados. Para ejecutar el código fuente, necesita Python 2.5 pre-instalado en el sistema. También es necesario uno de los motores de base de datos compatible instalado. Para el ensayo y la demanda en aplicaciones de la luz, puede utilizar la base de datos SQLite, incluido en Python 2.5.

Las versiones binarias de web2py (para Windows y Mac OS X) son una de Python 2.5 intérprete y la base de datos SQLite. Técnicamente, estos dos no son componentes de web2py. Su inclusión en las distribuciones binarias le permite quedarse sin web2py de la caja.

La siguiente imagen muestra la estructura general web2py:



1.7 Licencia

web2py está licenciado bajo la licencia GPL versión 2. El texto completo de la licencia si está disponible en ref. (31).

La licencia incluye pero no se limita a los siguientes artículos:

1. Usted puede copiar y distribuir copias exactas del código fuente del Programa, según lo has recibido, por cualquier medio, siempre que adecuada y bien visible publique en cada copia un anuncio de copyright adecuado y un repudio de garantía, mantenga intactos todos los anuncios que se refieran a esta Licencia ya la ausencia de garantías, y proporcione a cualquier otro receptor del programa una copia de esta licencia junto con el Programa.

[...]

4. Usted no puede copiar, modificar, sublicenciar o distribuir el Programa excepto como prevé expresamente esta Licencia. Cualquier intento de copiar, modificar, sublicenciar o distribuir el Programa es nulo, y terminará automáticamente sus derechos bajo esta Licencia.

[...]

11. Como el programa se licencia libre de cargas, NO HAY GARANTÍA PARA EL PROGRAMA, en la medida permitida por la ley. Excepto cuando así lo declaró por escrito los titulares del copyright Y / u otras partes proporcionan EL PROGRAMA "AS IS" SIN GARANTÍA DE NINGÚN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO, PERO NO LIMITADO A, LAS GARANTÍAS IMPLÍCITAS DE COMERCIALIZACIÓN Y APTITUD PARA UN PROPÓSITO PARTICULAR. Los riesgos en cuanto a la calidad y los resultados del programa está contigo. Si el programa resultara defectuoso, usted asume el costo de cualquier servicio, reparación o corrección.
12. EN NINGÚN CASO MENOS exija la ley aplicable o haya sido acordado por escrito, ningún tenedor del copyright ni ninguna otra parte que modifique y / o redistribuya el Programa en los lugares permitidos, serán responsables ante usted por daños, INCLUYENDO CUALQUIER GENERAL, especiales, incidentales o DAÑOS DERIVADOS DERIVADAS DEL USO O INCAPACIDAD DE USAR EL PROGRAMA (incluyendo pero no limitado a pérdida de datos o la generación incorrecta o pérdidas sufridas por usted o terceras partes o un fallo del Programa al funcionar con cualquier otro programa), incluso si dicho tenedor u otra parte ha sido ADVERTIDO DE LA POSIBILIDAD DE TALES DAÑOS. web2py incluye algún código de terceros (por ejemplo, el intérprete de Python, el servidor web de cohete, y algunas bibliotecas JavaScript). Sus autores respectivos y las licencias son reconocidos en el diario oficial website (1) y en el propio código.

Las aplicaciones desarrolladas con web2py, siempre y cuando no incluir el código fuente web2py, no se consideran obras derivadas. Esto significa que no están sujetos a la licencia GPLv2, y usted puede distribuir las aplicaciones que desarrollaron bajo cualquier licencia que usted elija, incluyendo un código cerrado y / o licencia comercial.

License Commercial Exception

La licencia web2py también incluye una excepción comercial:

Usted puede distribuir una aplicación que se desarrolló con web2py junto con un oficial sin modificar la distribución binaria de web2py, que también puede descargarse de la página oficial website (1), siempre y cuando quede claro en la licencia de su solicitud de qué archivos pertenecen a la aplicación y qué archivos pertenecen a web2py.

1.8 *Agradecimientos*

web2py fue originalmente desarrollado por y propiedad de Massimo Di Pierro. La primera versión (1.0) fue lanzado en octubre de 2007. Desde entonces ha sido adoptado por muchos usuarios, algunos de los cuales han contribuido también los informes de error, prueba, depuración, parches y corrección de textos de este libro.

Algunos de los principales contribuyentes son, en orden alfabético por apellido:

Nezhdanov Alexey, Justen Alvaro, Csipa Atila, Ferrett Bill, Boris, Brian Meredyk Manojlovic, Bruno Rocha, Haese Carsten, Chris Clark, Chris Acero, Christian Howes Foster, Christopher Smiga, Lazell CJ, Craig Younkins, Lin Daniel, Denes Lengyel, Douglas Soares de Andrade, Boon Fran, Francisco Gama, Yanowski Fred, Dumpleton Graham, Szabolcs Gyuris, HAMDY Abdel-Badeea, Hans Donner, Murx Hans, Hans C. c. Stockhausen, Ian Reinhart Geiser, Benn Jonathan, Jonathan Lundell, Goldfoot Josh, Jachuf José, Yang Keith, Smith Kyle, Limodou, Leuthi Marcel, Marcello Della Longa, Reingart Mariano, Larsen Marcos, Marcos Moore, Gritsch Markus, Hufsky Martin, Banach Mateusz, Michael Willis, Comitini Michele, congellar Nathan, Niall Sweeny, Niccolo Polo, Nicolas Bruxer, Ondrej Tales, Pai, Phyto Arkar Lwin, Bhattacharyya Robin, Ruijun Luo (Iceberg), Scott Roberts, Sergey Podlesnyi, Aina Sharriff, Sriram, Durbha, Hankins Sterling, Stuart Rackham, Yusupov Telman, Burgess Thadeus, Tim Michelsen, Farrell Timoteo, Yair Eshel, Yarko Tymciurak, Jo Younghyun, Zahariash.

Estoy seguro de que he olvidado a alguien, así que pido disculpas.

En particular, agradezco Jonathan, Iceberg, Nathan, y Thadeus de sus importantes contribuciones a web2py y Álvaro, Bruno, Denes, Felipe, Graham, Jonathan, Hans, Kyle, Marcos, Miguel, Richard, Robin, romana, Scott, Shane, Sharriff, Sriram, Sterling, Stuart, Thadeus Yarko y para la corrección diversos capítulos de este libro. Su contribución fue muy valiosa. Si encuentra algún error en este libro, son exclusivamente culpa mía, probablemente introducido por un proceso de edición de última hora. También agradezco a Ryan Stoffen, de Wiley Soluciones de aprendizaje para ayudar con la publicación de la primera edición de este libro. web2py contiene el código de los siguientes autores, a quien quiero dar las gracias a:

Guido van Rossum para Python (2), Peter Hunt, Richard Gordon, Timothy Farrell para el servidor web Rocket (22), Christopher Dolivet para EditArea (23), Bob Ippolito de simplejson (25), Cusack Simon Edwards y de donaciones para pyRTF (26), Dalke Scientific Software para pyRSS2Gen (27), Mark Pilgrim por feedparser (28), Trent Mick para markdown2 (29), Allan Saddi para fcgi.py, Evan Martin para la memcache module (30) Python, John Resig para jQuery (32).

El logotipo utilizado en la portada de este libro fue diseñado por Peter Kirchner a jóvenes diseñadores.

Doy las gracias a Helmut Epp (rector de la Universidad DePaul), David Miller (decano de la Facultad de Informática y Medios Digitales de la Universidad DePaul), y Estia Eichten (Miembro de MetaCryption LLC), por su confianza y apoyo continuos.

Por último, quiero agradecer a mi esposa, Claudia, y mi hijo, Marco, por aguantarme a mí durante las muchas horas que he pasado en desarrollo web2py, intercambiar e-mails con los usuarios y colaboradores, y escribir este libro. Este libro está dedicado a ellos.

1.9 Acerca de este libro

Este libro incluye los siguientes capítulos, además de esta introducción:

- El capítulo 2 es una introducción a Python minimalist. Asume un conocimiento de ambos procedimientos y conceptos orientados a objetos de programación como bucles, condiciones, llamadas a funciones y clases, y cubre los aspectos básicos de sintaxis de Python. También incluye ejemplos de módulos de Python que se utilizan en todo el libro. Si ya sabes Python, puede omitir el capítulo 2.
- El capítulo 3 muestra cómo iniciar web2py, habla de la interfaz administrativa, y guía al lector a través de varios ejemplos de complejidad creciente: una aplicación que devuelve una cadena, una aplicación de mostrador, un blog de imágenes y una aplicación plena soplado wiki que acepta la subida de la imagen y comentarios, proporciona autenticación, autorización, servicios web y un canal RSS. Al leer este capítulo, es posible que necesite consultar el capítulo 2 de la sintaxis de Python y en general a los capítulos siguientes para una referencia más detallada sobre las funciones que se utilizan.
- El capítulo 4 abarca de manera más sistemática la estructura del núcleo y las bibliotecas: mapeo de URL, la solicitud, la respuesta, sesiones, cacheint, CRON, la internacionalización y flujo de trabajo general.
- El capítulo 5 es una referencia para la plantilla lenguaje utilizado para construir puntos de vista. Se muestra cómo incrustar código Python dentro de código HTML, y se muestra el uso de ayudantes (los objetos que pueden generar HTML).
- Capítulo 6 se refiere a la Capa de abstracción de base de datos, o DAL. La sintaxis de la DAL se presenta a través de una serie de ejemplos.
- El capítulo 7 trata de las formas, la validación de formularios y procesamiento de formularios. FORM es el ayudante de bajo nivel para la creación de forma. SQLFORM es el constructor de alto nivel de formulario. En el capítulo 7 se discute la nueva Crear / Leer / Actualizar / Eliminar (CRUD) de la API.

- Capítulo 8 se refiere a la autenticación, autorización y extensible acceso basado en roles Mecanismo de control disponibles en web2py. La configuración del correo y CAPTCHA también se discuten aquí, ya que son utilizados por la autenticación. En la tercera edición del libro hemos añadido una amplia cobertura de la integración con mecanismos de autenticación de tercero como OpenID, OAuth, Google, Facebook, Linkedin, etc
- El capítulo 9 trata sobre la creación de servicios web en web2py. Proporcionamos ejemplos de integración con el kit de herramientas web de Google a través de Pijamas, y con Adobe Flash a través de PyAMF.
- Capítulo 10 está a punto web2py y recetas jQuery. web2py está diseñado principalmente para la programación del lado del servidor, pero incluye jQuery, puesto que hemos encontrado para ser los mejores de la biblioteca de código abierto disponible para efectos de JavaScript y Ajax. En este capítulo se explica cómo utilizar con eficacia con jQuery web2py.
- Capítulo 11 habla sobre la implementación de producción de aplicaciones web2py. Estamos principalmente frente a tres escenarios de producción posible: en un servidor web Linux o un conjunto de servidores (que se considera la alternativa de implementación principal), funcionando como un servicio en un entorno de Microsoft Windows, y el despliegue de las aplicaciones de Google Engine (GAE). En este capítulo, se discute cuestiones de seguridad y escalabilidad.
- El capítulo 12 contiene una variedad de recetas de otros para resolver tareas específicas, incluyendo actualizaciones, geocodificación, paginación, Twitter API, y mucho más.
- Capítulo 13 se añadió en la tercera edición del libro y analiza los componentes web2py y plugins como un medio para crear aplicaciones modulares. Se presenta un ejemplo de un plugin que implementa la funcionalidad de los más comunes, tales como gráficos, comentarios, etiquetas, y las páginas wiki.

Este libro sólo abarca funcionalidades básicas web2py y la API que se incluye con web2py. Este libro no cubre aparatos web2py (es decir, presentando solicitudes de lista).

Puede descargar aplicaciones web2py de la (34) sitio web correspondiente.

Usted puede encontrar temas adicionales discutidos en AlterEgo (35), el interactivo web2py FAQ.

1.10 Elementos de Estilo

Ref. (36) contiene buenas prácticas de estilo al programar con Python. Usted encontrará que web2py no siempre siguen estas reglas. Esto no es a causa de omisiones o negligencia, es nuestra creencia de que los usuarios de web2py deben seguir estas reglas y lo apoyamos. Decidimos no seguir algunas de estas normas en la definición de los objetos auxiliares web2py con el fin de minimizar la probabilidad de conflicto de nombres con objetos definidos por el usuario.

Por ejemplo, la clase que representa una `<div>` DIV se llama, mientras que según la referencia de estilo de Python que debería haber sido llamada Div. Creemos que, para este ejemplo concreto que el uso de una mayúscula- "DIV" todo es una opción más natural. Por otra parte, este enfoque deja programadores de software libre para crear una clase llamada "Div" si deciden hacerlo. Nuestros mapas de sintaxis también naturalmente en la notación de la mayoría de los navegadores DOM (incluyendo, por ejemplo, Firefox).

De acuerdo con el Manual de Estilo de Python, todo en mayúsculas cadenas deberían ser utilizados para las constantes y no variables. Continuando con nuestro ejemplo, incluso teniendo en cuenta que DIV es una clase, es una clase especial que nunca debe ser modificados por el usuario ya que al hacerlo se rompería otras aplicaciones web2py. Por lo tanto, creemos que esto califica la clase DIV como algo que debe ser tratada como una constante, más que justifica nuestra elección de la notación.

En resumen, los convenios se siguen los siguientes:

- ayudantes HTML y validadores son todo en mayúsculas por las razones expuestas más arriba (por ejemplo, DIV, A, FORM, URL).
- El T es objeto traductor mayúsculas a pesar de que es una instancia de una clase y no una clase en sí. Lógicamente el objeto traductor realiza una acción similar a los ayudantes HTML, afecta a la actualización de parte de la presentación. Además, las necesidades de T para ser fácil de localizar en el código y debe tener un nombre corto.
- clases DAL seguir la guía de estilo de Python (primera letra en mayúscula), por ejemplo Table, Field, Query, Row, Rows, etc

En los demás casos creemos que hemos seguido, en la medida de lo posible, la Guía de Estilo de Python (PEP8). Por ejemplo, todos los objetos de instancia son minúsculas (solicitud, respuesta, período de sesiones, la memoria caché), y todas las clases internas se capitalizan.

En todos los ejemplos de este libro, las palabras clave web2py se muestran en negrita, mientras que las cadenas y los comentarios se muestran en cursiva.

2

El lenguaje Python

2.1 Acerca de Python

Python es un uso general y muy a nivel de lenguaje de programación de alto. Su filosofía de diseño hace hincapié en la productividad del programador y legibilidad del código. Tiene una sintaxis minimalista central con comandos básicos muy pocos y la semántica simple, pero también tiene una biblioteca estándar amplio y completo, incluyendo una interfaz de programación (API) para muchos de los operativos subyacentes del sistema (OS) funciones. El código en Python, mientras que minimalista, define objetos tales como listas ligadas (`list`), tuplas (`tuple`), tablas hash (`dict`), y números enteros arbitrariamente larga (`long`).

Python soporta múltiples paradigmas de programación. Estos están orientados a objetos (`class`), imperativo (`def`) y funcionales (`lambda`) de programación. Python tiene un sistema de tipo dinámico y gestión de memoria automática mediante el conteo de referencias (similar a Perl, Ruby y Scheme).

Python fue lanzado por primera vez por Guido van Rossum en 1991. La lengua tiene un modelo de desarrollo abierto y basado en la comunidad ges-

tionados por la sin fines de lucro Python Software Foundation. Hay muchos intérpretes y compiladores que implementan el lenguaje Python, incluyendo uno en Java (Jython), pero, en esta breve reseña, nos referimos a la implementación en C de referencia creada por Guido.

Usted puede encontrar muchos tutoriales, la documentación oficial y la colección de referencias de la lengua en la página oficial de Python website (2).

Para más referencias Python, podemos recomendar los libros de ref. (37) y ref. (38).

Usted puede saltarse este capítulo, si usted ya está familiarizado con el lenguaje Python.

2.2 *Puesta en marcha*

Las distribuciones binarias de web2py para Microsoft Windows o Apple OS X vienen empaquetados con el intérprete de Python integrado en el propio archivo de distribución.

Usted puede iniciar en Windows con el comando siguiente (tipo en el símbolo de DOS):

```
1 web2py.exe -S welcome
```

El Apple OS X, ingrese el tipo de comando siguiente en una ventana de terminal (asumiendo que usted está en la misma carpeta que web2py.app):

```
1 ./web2py.app/Contents/MacOS/web2py -S welcome
```

En un sistema Linux o Unix otra caja, lo más probable es que usted tiene ya instalado Python. De ser así, a un tipo de línea de comandos:

```
1 python web2py.py -S welcome
```

Si no tienes Python 2.5 (o posterior 2.x) ya está instalado, usted tendrá que descargarlo e instalarlo antes de ejecutar web2py.

La opción de línea de comandos `-S welcome` instruye web2py para ejecutar el intérprete de comandos interactivo como si los comandos fueran ejecutados en un controlador para la aplicación `welcome`, la aplicación web2py andamios. Esto expone a casi todas las clases web2py, objetos y funciones en su caso. Es la única diferencia entre los de línea de comandos interactiva web2py y la normal línea de comandos Python.

La interfaz de administración también proporciona un shell basado en la Web para cada aplicación. Puede acceder a la una para la aplicación en la "welcome".

¹ <http://127.0.0.1:8000/admin/shell/index/welcome>

Puedes probar todos los ejemplos en este capítulo con la consola normal o la shell basado en la web.

2.3 *ayuda, dir*

El lenguaje Python proporciona dos comandos para obtener documentación acerca de los objetos definidos en el alcance actual, tanto integrados y definidos por el usuario.

Podemos pedir `help` acerca de un objeto, por ejemplo "1":

```
1 >>> help(1)
2 Help on int object:
3
4 class int(object)
5 |   int(x[, base]) -> integer
6 |
7 |   Convert a string or number to an integer, if possible. A floating point
8 |   argument will be truncated towards zero (this does not include a string
9 |   representation of a floating point number!) When converting a string, use
10 |   the optional base. It is an error to supply a base when converting a
```

```

11 | non-string. If the argument is outside the integer range a long object
12 | will be returned instead.
13 |
14 | Methods defined here:
15 |
16 | __abs__(...)
17 |     x.__abs__() <=> abs(x)
18 ...

```

y, desde "1" es un entero, se obtiene una descripción acerca de la clase `int` y todos sus métodos. Aquí la salida se ha truncado porque es muy largo y detallado.

Del mismo modo, podemos obtener una lista de métodos del objeto "1" con el comando `dir`:

```

1 >>> dir(1)
2 ['__abs__', '__add__', '__and__', '__class__', '__cmp__', '__coerce__',
3 '__delattr__', '__div__', '__divmod__', '__doc__', '__float__',
4 '__floordiv__', '__getattribute__', '__getnewargs__', '__hash__', '__hex__',
5 '__index__', '__init__', '__int__', '__invert__', '__long__', '__lshift__',
6 '__mod__', '__mul__', '__neg__', '__new__', '__nonzero__', '__oct__',
7 '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdiv__',
8 '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
9 '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__rpow__', '__rrshift__',
10 '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__',
11 '__str__', '__sub__', '__truediv__', '__xor__']

```

2.4 Tipos

Python es un lenguaje escrito de forma dinámica, lo que significa que las variables no tienen un tipo y por lo tanto no tienen que ser declarados. Valores, por el contrario, tienen un tipo. Puede consultar una variable para el tipo de valor que contiene:

```

1 >>> a = 3
2 >>> print type(a)
3 <type 'int'>
4 >>> a = 3.14
5 >>> print type(a)

```

```

6 <type 'float'>
7 >>> a = 'hello python'
8 >>> print type(a)
9 <type 'str'>

```

Python también incluye, de forma nativa, estructuras de datos tales como listas y diccionarios.

2.4.1 str

Python admite el uso de dos diferentes tipos de cadenas: cadenas de caracteres ASCII y cadenas Unicode. cadenas de caracteres ASCII están delimitados por '...', ..." or by "...". Triple cita a delimitar cadenas de varias líneas. cadenas Unicode comenzar con una u seguido de la cadena que contiene caracteres Unicode. Una cadena Unicode se pueden convertir en una cadena ASCII eligiendo una codificación por ejemplo:

```

1 >>> a = 'this is an ASCII string'
2 >>> b = u'This is a Unicode string'
3 >>> a.encode('utf8')

```

Después de ejecutar esos tres comandos, el a resultante es una cadena ASCII UTF8 almacenar caracteres codificados. Por diseño, web2py utiliza UTF8 cadenas codificadas internamente.

También es posible escribir las variables en cadenas de diversas maneras:

```

1 >>> print 'number is ' + str(3)
2 number is 3
3 >>> print 'number is %s' % (3)
4 number is 3
5 >>> print 'number is %(number)s' % dict(number=3)
6 number is 3

```

La notación última es más explícita y menos propenso a errores, y es preferible.

Muchos objetos de Python, por ejemplo, los números, se pueden serializar en cadenas utilizando `str` o `repr`. Estos dos comandos son muy similares pero que producen resultados ligeramente distintos. Por ejemplo:

```

1 >>> for i in [3, 'hello']:
2         print str(i), repr(i)
3 3
4 hello 'hello'
```

Para las clases definidas por el usuario, `str` y `repr` se puede definir / redefinido mediante los operadores especiales `__str__` y `__repr__`. Estos se describen brevemente más adelante, para más, consulte el funcionario documentation (39) Python. `repr` siempre tiene un valor predeterminado.

Otra característica importante de una cadena de Python es que, como un lista, es un objeto iterable.

```

1 >>> for i in 'hello':
2         print i
3 h
4 e
5 l
6 l
7 o
```

2.4.2 lista

Los principales métodos de una lista de Python son añadir, insertar y borrar:

```

1 >>> a = [1, 2, 3]
2 >>> print type(a)
3 <type 'list'>
4 >>> a.append(8)
5 >>> a.insert(2, 7)
6 >>> del a[0]
7 >>> print a
8 [2, 7, 3, 8]
9 >>> print len(a)
10 4
```

Las listas pueden ser cortados en lonchas:

```

1 >>> print a[:3]
2 [2, 7, 3]
3 >>> print a[1:]
4 [7, 3, 8]
5 >>> print a[-2:]
6 [3, 8]
```

y concatenados:

```

1 >>> a = [2, 3]
2 >>> b = [5, 6]
3 >>> print a + b
4 [2, 3, 5, 6]
```

Una lista es repetible, que puede circular sobre ella:

```

1 >>> a = [1, 2, 3]
2 >>> for i in a:
3         print i
4 1
5 2
6 3
```

Los elementos de una lista no tiene que ser del mismo tipo, ya que pueden ser cualquier tipo de objeto de Python.

2.4.3 tupla

Una tupla es como una lista, pero su tamaño y los elementos son inmutables, mientras que en una lista que son mutables. Si un elemento de la tupla es un objeto, los atributos de objetos son mutables. Una tupla está delimitado por paréntesis.

```

1 >>> a = (1, 2, 3)
```

Así, mientras que esto funciona para una lista:

```

1 >>> a = [1, 2, 3]
2 >>> a[1] = 5
3 >>> print a
4 [1, 5, 3]

```

la asignación de elemento no funciona para una tupla:

```

1 >>> a = (1, 2, 3)
2 >>> print a[1]
3 2
4 >>> a[1] = 5
5 Traceback (most recent call last):
6   File "<stdin>", line 1, in <module>
7 TypeError: 'tuple' object does not support item assignment

```

La tupla, como la lista, es un objeto iterable. Tenga en cuenta que una tupla consistente en un solo elemento debe incluir una coma, como se muestra a continuación:

```

1 >>> a = (1)
2 >>> print type(a)
3 <type 'int'>
4 >>> a = (1,)
5 >>> print type(a)
6 <type 'tuple'>

```

Las tuplas son muy útiles para el envasado eficiente de los objetos a causa de su inmutabilidad, y los soportes suelen ser opcionales:

```

1 >>> a = 2, 3, 'hello'
2 >>> x, y, z = a
3 >>> print x
4 2
5 >>> print z
6 hello

```

2.4.4 dict

Un Python-dict ionary es una tabla hash que se asigna un objeto clave para un objeto de valor. Por ejemplo:

```

1 >>> a = {'k':'v', 'k2':3}
2 >>> a['k']
3 v
4 >>> a['k2']
5 3
6 >>> a.has_key('k')
7 True
8 >>> a.has_key('v')
9 False

```

Las claves pueden ser de cualquier tipo hashable (int, string, o cualquier objeto cuya clase implementa el método de `__hash__`). Los valores pueden ser de cualquier tipo. Claves y valores diferentes en el mismo diccionario no tienen por qué ser del mismo tipo. Si las claves son caracteres alfanuméricos, un diccionario también puede ser declarada con la sintaxis alternativa:

```

1 >>> a = dict(k='v', h2=3)
2 >>> a['k']
3 v
4 >>> print a
5 {'k': 'v', 'h2': 3}

```

Métodos son útiles `has_key`, `keys`, `values` y `items`:

```

1 >>> a = dict(k='v', k2=3)
2 >>> print a.keys()
3 ['k', 'k2']
4 >>> print a.values()
5 ['v', 3]
6 >>> print a.items()
7 [('k', 'v'), ('k2', 3)]

```

El método `items` produce una lista de tuplas, cada uno con una clave y su valor asociado. Elementos de diccionario y elementos de la lista se pueden eliminar con el comando `del`:

```

1 >>> a = [1, 2, 3]
2 >>> del a[1]
3 >>> print a
4 [1, 3]
5 >>> a = dict(k='v', h2=3)
6 >>> del a['h2']
7 >>> print a
8 {'k': 'v'}

```

Internamente, Python utiliza el operador `hash` para convertir objetos en números enteros, y utiliza ese número entero para determinar dónde almacenar el valor.

```

1 >>> hash("hello world")
2 -1500746465

```

2.5 Acerca de sangría

Python utiliza sangría para delimitar bloques de código. Un bloque comienza con una línea que termina en dos puntos, y continúa para todas las líneas que tienen un sangrado similar o superior en la línea siguiente. Por ejemplo:

```

1 >>> i = 0
2 >>> while i < 3:
3     >>>     print i
4     >>>     i = i + 1
5 >>>
6 0
7 1
8 2

```

Es común el uso de cuatro espacios para cada nivel de sangría. Es bueno no mezclar la política de una fichas con espacios, o usted puede tener problemas.

2.6 de... en

En Python, puede circular por encima de objetos iterable:

```

1 >>> a = [0, 1, 'hello', 'python']
2 >>> for i in a:
3     >>>         print i
4 0
5 1
6 hello
7 python

```

Un acceso directo común es `xrange`, el cual genera una amplia iterable sin almacenar toda la lista de elementos.

```

1 >>> for i in xrange(0, 4):
2         print i
3
4 0
5 1
6 2
7 3

```

Esto es equivalente a la de C / C ++ / C # / sintaxis de Java:

```

1 for(int i=0; i<4; i=i+1) { print(i); }

```

Otro comando útil es `enumerate`, que cuenta, mientras que bucle:

```

1 >>> a = [0, 1, 'hello', 'python']
2 >>> for i, j in enumerate(a):
3         print i, j
4
5 0 0
6 1 1
7 2 hello
8 3 python

```

También hay una palabra clave de `range(a, b, c)` que devuelve una lista de números enteros a partir de la `a` valor, por incremento `c`, y terminando con el último valor más pequeño que `b`, a por defecto `0` y por defecto `c` a `1`. `xrange` es similar, aunque no genere la lista, sólo un iterador sobre la lista, por lo que es mejor para recorrer.

Puede saltar de un bucle que utiliza `break`

```

1 >>> for i in [1, 2, 3]:
2         print i
3         break
4
5 1

```

Usted puede saltar a la próxima iteración del bucle sin ejecutar el bloque de código completo con `continue`

```

1 >>> for i in [1, 2, 3]:
2         print i

```

```

3     continue
4     print 'test'
5 1
6 2
7 3

```

2.7 *mientras que*

El bucle `while` en Python funciona mucho como lo hace en muchos otros lenguajes de programación, haciendo un bucle en un número indefinido de veces y probar una condición antes de cada iteración. Si la condición es `False`, el ciclo termina.

```

1 >>> i = 0
2 >>> while i < 10:
3         i = i + 1
4 >>> print i
5 10

```

No hay construcción de `loop...until` en Python.

2.8 *def... regresar*

Aquí es una típica función de Python:

```

1 >>> def f(a, b=2):
2         return a + b
3 >>> print f(4)
4 6

```

No es necesario (o camino) para especificar tipos o de los argumentos del tipo de cambio (`s`).

Parámetros de las funciones pueden tener valores por defecto y puede devolver varios objetos:

```

1 >>> def f(a, b=2):
2     return a + b, a - b
3 >>> x, y = f(5)
4 >>> print x
5 7
6 >>> print y
7 3

```

Parámetros de las funciones se pueden pasar explícitamente por nombre:

```

1 >>> def f(a, b=2):
2     return a + b, a - b
3 >>> x, y = f(b=5, a=2)
4 >>> print x
5 7
6 >>> print y
7 -3

```

Las funciones pueden tener un número variable de argumentos:

```

1 >>> def f(*a, **b):
2     return a, b
3 >>> x, y = f(3, 'hello', c=4, test='world')
4 >>> print x
5 (3, 'hello')
6 >>> print y
7 {'c':4, 'test':'world'}

```

Aquí los argumentos no se pasan por su nombre (3, 'hola') se almacenan en a lista y argumentos que se pasan por su nombre (c y test) se almacenan en el diccionario de b.

En el caso contrario, una lista o tupla se puede pasar a una función que requiere argumentos posicionales individuales por desembalar ellos:

```

1 >>> def f(a, b):
2     return a + b
3 >>> c = (1, 2)
4 >>> print f(*c)
5 3

```

y un diccionario se pueden desempaquetar para entregar argumentos clave:

```

1 >>> def f(a, b):
2     return a + b
3 >>> c = {'a':1, 'b':2}
4 >>> print f(**c)
5 3

```

2.9 si... elif... else

El uso de los condicionales en Python es intuitivo:

```

1 >>> for i in range(3):
2     if i == 0:
3         print 'zero'
4     elif i == 1:
5         print 'one'
6     else:
7         print 'other'
8 zero
9 one
10 other

```

"elif" means "else if". Ambas cláusulas `elif` y `else` son opcionales. No puede haber más de una `elif`, pero sólo una declaración de `else`. condiciones complejas pueden ser creados usando el, `not` y `and` or operadores.

```

1 >>> for i in range(3):
2     if i == 0 or (i == 1 and i + 1 == 2):
3         print '0 or 1'

```

2.10 intenta... excepto... si no... por fin

puede lanzar Python - perdón, levante - Excepciones:

```

1 >>> try:
2     a = 1 / 0
3 >>> except Exception, e
4     print 'oops: %s' % e
5 >>> else:

```

```

6 >>>     print 'no problem here'
7 >>> finally:
8 >>>     print 'done'
9 oops: integer division or modulo by zero
10 done

```

Si la excepción es capturado por la cláusula de `except`, que se ejecuta, mientras que la cláusula de `else` no lo es. Si no se lanza una excepción, la cláusula de `except` no se ejecuta, pero es el `else`. La cláusula de `finally` se ejecuta siempre.

No puede haber múltiples cláusulas `except` para diferentes posibles excepciones:

```

1 >>> try:
2 >>>     raise SyntaxError
3 >>> except ValueError:
4 >>>     print 'value error'
5 >>> except SyntaxError:
6 >>>     print 'syntax error'
7 syntax error

```

Las cláusulas de `else` y `finally` son opcionales.

Aquí está una lista de excepciones incorporadas Python + HTTP (definido por web2py)

```

1 BaseException
2   +-+ HTTP (defined by web2py)
3   +-+ SystemExit
4   +-+ KeyboardInterrupt
5   +-+ Exception
6     +-+ GeneratorExit
7     +-+ StopIteration
8     +-+ StandardError
9       |   +-+ ArithmeticError
10      |   |   +-+ FloatingPointError
11      |   |   +-+ OverflowError
12      |   |   +-+ ZeroDivisionError
13      |   +-+ AssertionError
14      |   +-+ AttributeError
15      |   +-+ EnvironmentError
16      |   |   +-+ IOError

```

```

17     |     +-+ OSError
18     |     |     +-+ WindowsError (Windows)
19     |     |     +-+ VMSError (VMS)
20     +-+ EOFError
21     +-+ ImportError
22     +-+ LookupError
23     |     +-+ IndexError
24     |     |     +-+ KeyError
25     +-+ MemoryError
26     +-+ NameError
27     |     +-+ UnboundLocalError
28     +-+ ReferenceError
29     +-+ RuntimeError
30     |     +-+ NotImplementedError
31     +-+ SyntaxError
32     |     +-+ IndentationError
33     |     |     +-+ TabError
34     +-+ SystemError
35     +-+ TypeError
36     +-+ ValueError
37     |     +-+ UnicodeError
38     |     |     +-+ UnicodeDecodeError
39     |     |     +-+ UnicodeEncodeError
40     |     |     +-+ UnicodeTranslateError
41     +-+ Warning
42     |     +-+ DeprecationWarning
43     |     +-+ PendingDeprecationWarning
44     |     +-+ RuntimeWarning
45     |     +-+ SyntaxWarning
46     |     +-+ UserWarning
47     |     +-+ FutureWarning
48     +-+ ImportWarning
49     +-+ UnicodeWarning

```

Para obtener una descripción detallada de cada uno de ellos, consulte la documentación oficial de Python. web2py expone sólo una nueva excepción, llamada `HTTP`. Cuando levantó, hace que el programa devuelva una página de error HTTP (para más sobre este consulte el Capítulo 4).

Cualquier objeto puede ser planteado como una excepción, pero es una buena práctica de levantar objetos que se extienden a una de las excepciones incorporadas.

2.11 clase

Debido a que Python es tipado dinámico, las clases de Python y los objetos pueden parecer extrañas. De hecho, no es necesario definir las variables miembro (atributos) cuando se declara una clase, y las distintas instancias de la misma clase pueden tener diferentes atributos. Los atributos se asocia generalmente con la instancia, no la clase (excepto cuando ha sido declarado como "class attributes", which is the same as "static member variables" en C + + / Java).

Aquí está un ejemplo:

```

1 >>> class MyClass(object): pass
2 >>> myinstance = MyClass()
3 >>> myinstance.myvariable = 3
4 >>> print myinstance.myvariable
5 3

```

Tenga en cuenta que `pass` es un comando no hace nada. En este caso se utiliza para definir una clase que contiene `MyClass` nada. `MyClass()` llamadas al constructor de la clase (en este caso el constructor por defecto) y devuelve un objeto, una instancia de la clase. El `(object)` en la definición de clase indica que nuestra clase extiende la clase incorporada `object`. Esto no es necesario, pero es una buena práctica.

Aquí está un complejo de clase más:

```

1 >>> class MyClass(object):
2 >>>     z = 2
3 >>>     def __init__(self, a, b):
4 >>>         self.x = a, self.y = b
5 >>>     def add(self):
6 >>>         return self.x + self.y + self.z
7 >>> myinstance = MyClass(3, 4)
8 >>> print myinstance.add()
9 9

```

Las funciones declaradas dentro de la clase son los métodos. Algunos métodos tienen nombres especiales reservados. Por ejemplo, `__init__` es el con-

structor. Todas las variables son variables locales del método, excepto las variables declaradas fuera de los métodos. Por ejemplo, `z` es un equivalente *class variable, static member variable* a un C++ que tiene el mismo valor para todas las instancias de la clase.

Tenga en cuenta que tiene 3 argumentos `__init__` y `add` tiene una, y sin embargo, los llamamos con `z` y `o`, respectivamente argumentos. El primer argumento representa, por convención, el nombre local dentro del método utilizado para referirse al objeto actual. Aquí usamos `self` para referirse al objeto actual, pero podríamos haber utilizado cualquier otro nombre. `self` juega el mismo papel que `*this` en C++ o en Java `this`, `self`, pero no es una palabra clave reservada.

Esta sintaxis es necesario para evitar la ambigüedad al declarar clases anidadas, como una clase que es local a un método dentro de otra clase.

2.12 Características especiales, métodos y operadores

Clase atributos, métodos, y los operadores de partida con un doble subrayado habitualmente se destinan a ser privado, aunque esto es una convención que no se cumple por el intérprete.

Algunos de ellos son palabras clave reservadas y tienen un significado especial.

Aquí, como ejemplo, son tres de ellos:

- `__len__`
- `__getitem__`
- `__setitem__` Pueden utilizarse, por ejemplo, para crear un objeto contenedor que actúa como una lista:

¹ `>>> class MyList(object)`

```

2 >>> def __init__(self, *a): self.a = a
3 >>> def __len__(self): return len(self.a)
4 >>> def __getitem__(self, i): return self.a[i]
5 >>> def __setitem__(self, i, j): self.a[i] = j
6 >>> b = MyList(3, 4, 5)
7 >>> print b[1]
8 4
9 >>> a[1] = 7
10 >>> print b.a
11 [3, 7, 5]

```

Otros operadores especiales incluyen `__getattr__` y `__setattr__`, que definen el obtener y establecer los atributos de la clase, y `__sum__` y `__sub__`, que los operadores aritméticos sobrecarga. Para el uso de estos operadores, remitimos al lector a los libros más avanzados en este tema. Ya hemos mencionado los operadores especiales `__str__` y `__repr__`.

2.13 Archivo de entrada / salida

En Python se puede abrir y escribir en un archivo con:

```

1 >>> file = open('myfile.txt', 'w')
2 >>> file.write('hello world')

```

Del mismo modo, usted puede leer desde el archivo con:

```

1 >>> file = open('myfile.txt', 'r')
2 >>> print file.read()
3 hello world

```

Alternativamente, se puede leer en el modo de binarios con "rb", write in binary mode with "wb", and open the file in append mode "a", utilizando notación estándar C.

El comando `read` tiene un argumento opcional, que es el número de bytes. También puede ir a cualquier lugar en un archivo con `seek`.

Usted puede leer desde el archivo con la `read`

```
1 >>> print file.seek(6)
2 >>> print file.read()
3 world
```

y se puede cerrar el archivo con:

```
1 >>> file.close()
```

aunque a menudo esto no es necesario, ya que un archivo se cierra automáticamente cuando la variable que hace referencia a su viaje de alcance.

Al utilizar web2py, usted no sabe dónde está el directorio actual es, porque depende de cómo se configura web2py. El `request.folder` variable contiene la ruta de acceso a la aplicación actual. Caminos se pueden concatenar con la `os.path.join` de comandos, discuten a continuación.

2.14 lambda

Hay casos en los que pueda necesitar para generar dinámicamente una función sin nombre. Esto se puede hacer con la palabra clave `lambda`:

```
1 >>> a = lambda b: b + 2
2 >>> print a(3)
3 5
```

La expresión "`lambda a: [b]`" literalmente dice lo "a function with arguments [a] that returns [b]". Incluso si la función no tiene nombre, se puede almacenar en una variable, por lo que adquiere un nombre. Técnicamente esto es diferente que el uso de `def`, porque es la variable de referencia a la función que tiene un nombre, no la propia función.

¿Quién necesita lambdas? En realidad son muy útiles porque permiten volver a factor de una función en otra función mediante el establecimiento de

argumentos por defecto, sin definir una nueva función real, pero temporal. Por ejemplo:

```
1 >>> def f(a, b): return a + b
2 >>> g = lambda a: f(a, 3)
3 >>> g(2)
4 5
```

Aquí es más compleja y más convincente de la aplicación. Suponga que tiene una función que comprueba si su argumento es primo:

```
1 def isprime(number):
2     for p in range(2, number):
3         if number % p:
4             return False
5     return True
```

Esta función es, obviamente, consume tiempo.

Suponga que tiene una función de almacenamiento en caché que `cache.ram` tiene tres argumentos: una clave, una función y un número de segundos.

```
1 value = cache.ram('key', f, 60)
```

La primera vez que se le llama, llama a la función de `f()`, almacenes de la salida en un diccionario en la memoria (digamos "d"), y lo devuelve para que el valor es:

```
1 value = d['key']=f()
```

La segunda vez que se llama, si la clave está en el diccionario y más no que el número de segundos especificado (60), devuelve el valor correspondiente sin realizar la llamada a función.

```
1 value = d['key']
```

¿Cómo caché de la salida de la `isprime` función por cualquier entrada? He aquí cómo:

```
1 >>> number = 7
```

```

2 >>> print cache.ram(str(number), lambda: isprime(number), seconds)
3 True
4 >>> print cache.ram(str(number), lambda: isprime(number), seconds)
5 True

```

La salida es siempre el mismo, pero el `cache.ram` primera vez se llama, se llama `isprime`, la segunda no lo es.

La existencia de `lambda` permite re-factoring una función existente en términos de un conjunto diferente de argumentos. `cache.ram` y `cache.disk` son web2py caché funciones.

2.15 exec, eval

A diferencia de Java, Python es un lenguaje interpretado realmente. Esto significa que tiene la capacidad de ejecutar sentencias de Python almacenadas en cadenas. Por ejemplo:

```

1 >>> a = "print 'hello world'"
2 >>> exec(a)
3 'hello world'

```

¿Qué ha pasado? El `exec` cuenta la función de intérprete para llamarse a sí misma y ejecutar el contenido de la cadena pasada como argumento. También es posible ejecutar el contenido de una cadena dentro de un contexto definido por los símbolos en un diccionario:

```

1 >>> a = "print b"
2 >>> c = dict(b=3)
3 >>> exec(a, {}, c)
4 3

```

Aquí el intérprete, al ejecutar el `a` cadena, ve los símbolos definidos en `c` (`b` en el ejemplo), pero no ve `c` o a sí mismos. Esto es diferente a un entorno

restringido, ya que `exec` no limita lo que el código interno se puede hacer, sino que sólo define el conjunto de variables visibles en el código.

Una función relacionada es `eval`, que funciona muy parecido a `exec`, salvo que espera que el argumento para evaluar a un valor y devuelve ese valor.

```

1 >>> a = "3*4"
2 >>> b = eval(a)
3 >>> print b
4 12

```

2.16 importación

El verdadero poder de Python es en la biblioteca de sus módulos. Constituyen un conjunto grande y coherente de Application Programming Interfaces (APIs) para muchas bibliotecas del sistema (a menudo de una manera independiente del sistema operativo).

Por ejemplo, si usted necesita utilizar un generador de números aleatorios, que puede hacer:

```

1 >>> import random
2 >>> print random.randint(0, 9)
3 5

```

Esto imprime un entero aleatorio entre 0 y 9 (incluyendo 9), 5 en el ejemplo. El `randint` función está definida en el módulo de `random`. También es posible importar un objeto de un módulo en el espacio de nombres actual:

```

1 >>> from random import randint
2 >>> print randint(0, 9)

```

o importar todos los objetos de un módulo en el espacio de nombres actual:

```

1 >>> from random import *
2 >>> print randint(0, 9)

```

o importar todo en un espacio de nombres definido recientemente:

```
1 >>> import random as myrand
2 >>> print myrand.randint(0, 9)
```

En el resto de este libro, se utilizará principalmente objetos definidos en módulos `os`, `sys`, `datetime`, `time` y `cPickle`.

Todos los objetos web2py son accesibles por un módulo llamado gluon, y que es el tema de los capítulos posteriores. Internamente, web2py muchos usos módulos de Python (por ejemplo, thread), pero rara vez necesitan acceder a ellos directamente.

En las siguientes subsecciones consideramos aquellos módulos que son más útiles.

2.16.1 os

Este módulo proporciona una interfaz con el API del sistema operativo. Por ejemplo:

```
1 >>> import os
2 >>> os.chdir('..')
3 >>> os.unlink('filename_to_be_deleted')
```

— Algunas de las funciones `os`, `chdir`, tales como, NO DEBE ser utilizado en web2py porque no son "thread-safe". — `os.path.join` es muy útil ya que permite la concatenación de caminos en una manera independiente del sistema operativo:

```
1 >>> import os
2 >>> a = os.path.join('path', 'sub_path')
3 >>> print a
4 path/sub_path
```

variables de entorno del sistema se pueden consultar en:

```
1 >>> print os.environ
```

que es un diccionario de sólo lectura.

2.16.2 sistemas

El módulo contiene sys muchas variables y funciones, pero la que más uso es sys.path. Contiene una lista de rutas en Python busca módulos. Cuando intenta importar un módulo, Python lo busca en todas las carpetas que figuran en el sys.path. Si instala módulos adicionales en algún lugar y quieren Python para encontrarlos, debe anexar la ruta a esa ubicación en la sys.path.

```
1 >>> import sys
2 >>> sys.path.append('path/to/my/modules')
```

Cuando se ejecuta web2py, Python permanece residente en memoria, y sólo hay una sys.path, mientras que hay muchos hilos de servicio las solicitudes HTTP. Para evitar una pérdida de memoria, lo mejor es comprobar si hay una ruta ya está presente antes de añadir:

```
1 >>> path = 'path/to/my/modules'
2 >>> if not path in sys.path:
3 >>>     sys.path.append(path)
```

2.16.3 fecha y hora

El uso del módulo de fecha y hora se ilustra mejor con algunos ejemplos:

```
1 >>> import datetime
2 >>> print datetime.datetime.today()
3 2008-07-04 14:03:90
4 >>> print datetime.date.today()
5 2008-07-04
```

En ocasiones es posible que necesite los datos de sello de tiempo sobre la base de la hora UTC en oposición a la hora local. En este caso, puede usar la siguiente función:

```

1 >>> import datetime
2 >>> print datetime.datetime.utcnow()
3 2008-07-04 14:03:90

```

Los módulos de fecha y hora contiene varias clases: la fecha, fecha y hora, la hora y timedelta. La diferencia entre dos fechas o dos de fecha y hora o dos objetos de tiempo es un timedelta:

```

1 >>> a = datetime.datetime(2008, 1, 1, 20, 30)
2 >>> b = datetime.datetime(2008, 1, 2, 20, 30)
3 >>> c = b - a
4 >>> print c.days
5 1

```

En web2py, fecha y fecha y hora se utilizan para almacenar los tipos correspondientes cuando se pasan a SQL o de regresar de la base de datos.

2.16.4 tiempo

El módulo de tiempo difiere de date' and datetime », ya que representa el tiempo en segundos de la época (principios de 1970).

```

1 >>> import time
2 >>> t = time.time()
3 1215138737.571

```

Consulte la documentación de Python para funciones de conversión entre el tiempo en segundos y el tiempo como una datetime.

2.16.5 cPickle

Este es un módulo muy poderoso. Proporciona funciones que se pueden serializar casi cualquier objeto de Python, incluyendo objetos auto-referencial. Por ejemplo, vamos a construir un objeto extraño:

```
1 >>> class MyClass(object): pass
2 >>> myinstance = MyClass()
3 >>> myinstance.x = 'something'
4 >>> a = [1 ,2, {'hello':'world'}, [3, 4, [myinstance]]]
```

y ahora:

```
1 >>> import cPickle
2 >>> b = cPickle.dumps(a)
3 >>> c = cPickle.loads(b)
```

En este ejemplo, b es una representación de cadena de a, c y es una copia de a generados por de-serializar b. cPickle también puede serializar y de-serializar de un archivo:

```
1 >>> cPickle.dumps(a, open('myfile.pickle', 'wb'))
2 >>> c = cPickle.loads(open('myfile.pickle', 'rb'))
```


3

Información general

3.1 Inicio

web2py viene en paquetes binarios para Windows y Mac OS X. También hay una versión de código fuente que se ejecuta en Windows, Mac, Linux y otros sistemas Unix. El Windows y OS X versiones binarias incluir la necesaria intérprete de Python. El paquete del código fuente de Python supone que ya está instalado en el equipo. web2py no requiere instalación. Para empezar, descomprimir el archivo zip descargado para su sistema operativo y ejecutar el archivo web2py correspondiente.

En Windows, ejecute:

```
1 web2py.exe
```

En OS X, ejecuta:

```
1 open web2py.app
```

En Unix y Linux, ejecute desde el código fuente, escriba:

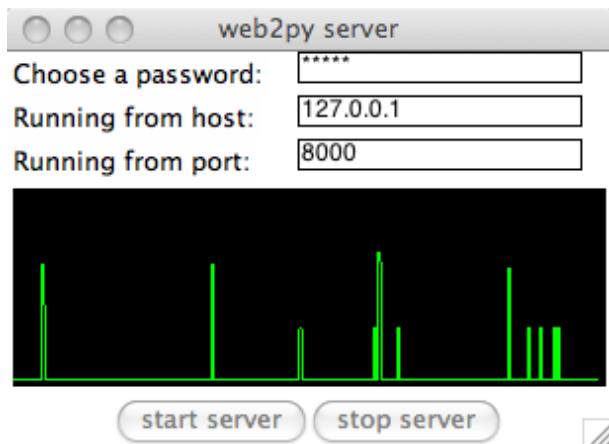
```
1 python2.5 web2py.py
```

El programa web2py acepta varias opciones de línea de comandos que se describen más adelante.

Forma predeterminada, al iniciar, web2py muestra una ventana de inicio:



y luego muestra un widget de interfaz gráfica de usuario que le pide que elija una contraseña de administrador de una sola vez, la dirección IP de la interfaz de red que se utilizará para el servidor web y un número de puerto desde donde servir peticiones. De forma predeterminada, web2py ejecuta su servidor web en 127.0.0.1:8000 (el puerto 8000 en localhost), pero se puede correr en cualquier dirección IP disponible y el puerto. Puede consultar la Dirección IP de su interfaz de red abriendo una línea de comandos y escribir en Windows o ipconfig ifconfig en OS X y Linux. A partir de ahora suponemos web2py se está ejecutando en localhost (127.0.0.1:8000). 0.0.0.0:80 utilizar para ejecutar web2py público en cualquiera de las interfaces de red.



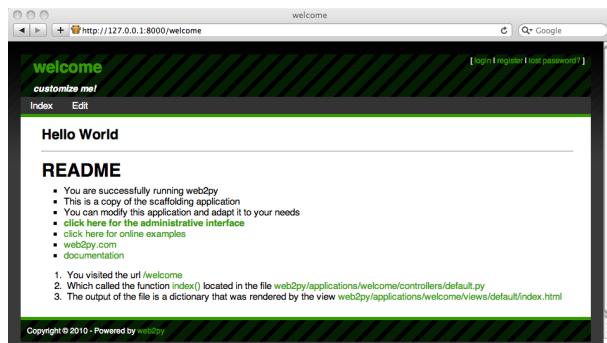
Si usted no proporciona una contraseña de administrador, la interfaz de administración está deshabilitada. Esta es una medida de seguridad para evitar exponer públicamente la interfaz de administración.

La interfaz administrativa, **admin**, sólo es accesible desde localhost a menos que ejecute web2py detrás de Apache con mod_proxy. Si detecta un proxy **admin**, la cookie de sesión se establece para asegurar y entrada **admin** no funciona a menos que la comunicación entre el cliente y el proxy va a través de HTTPS. Esta es otra medida de seguridad. Todas las comunicaciones entre el cliente y el **admin** siempre debe ser local o cifrados, de lo contrario un atacante podría ser capaz de realizar un "man-in-medio" del ataque o un ataque de reproducción y ejecutar código arbitrario en el servidor.

Después de la contraseña de administración se ha fijado, web2py pone en marcha el navegador de Internet en la página:

¹ <http://127.0.0.1:8000/>

Si el equipo no tiene un navegador por defecto, abra un navegador web e introduzca la dirección URL.

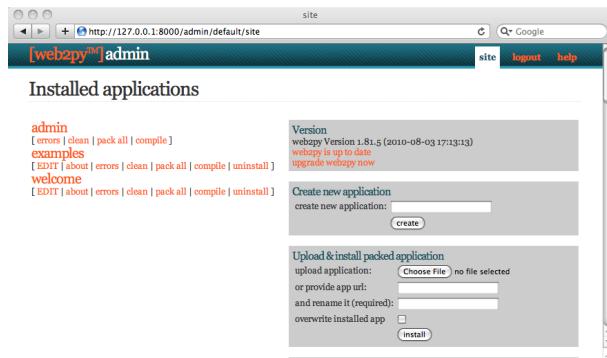


Al hacer clic en "administrative interface" le lleva a la página de acceso para la interfaz de administración.



La contraseña del administrador es la misma que la contraseña que eligió en el inicio. Tenga en cuenta que sólo hay un administrador, por lo que sólo una contraseña de administrador. Por razones de seguridad, el desarrollador se le pide que elija una contraseña nueva cada vez que se inicia a menos que el web2py <recycle> opción se especifica. Esto es distinto de el mecanismo de autenticación en aplicaciones web2py.

Después de que el administrador inicia sesión en web2py, el navegador es redirigido a la página de "site".



Esta página muestra todas las aplicaciones instaladas web2py y permite al administrador para gestionarlos. Web2py viene con tres aplicaciones:

- Una aplicación **admin**, la que está usando en este momento.
- Una aplicación **examples**, con la documentación en línea interactivas y una réplica de la página web oficial web2py.
- Una aplicación de **welcome**. Este es el modelo básico para cualquier aplicación web2py otros. Se hace referencia a que la aplicación andamios. Esta es también la aplicación que da la bienvenida a un usuario en el inicio.

listos para usar aplicaciones web2py se denominan web2py *appliances*. Puede descargar muchos aparatos disponibles libremente en (34). web2py usuarios son alentados a inscribir nuevos aparatos, ya sea en código abierto o de código cerrado (compilado y empaquetado) forma.

En la aplicación de la página **admin site**, puede realizar las siguientes operaciones:

- **install** una solicitud completando el formulario en la parte inferior derecha de la página. Asigne un nombre a la aplicación, seleccione el archivo que contiene una solicitud envasados o el URL donde se encuentra la aplicación, y haga clic en "submit".

- **uninstall** una aplicación haciendo clic en el botón correspondiente. Hay una página de confirmación.
- **create** una nueva solicitud de elegir un nombre y haciendo clic en "submit".
- **package** una solicitud de distribución haciendo clic en el botón correspondiente. Una aplicación se descarga un archivo que contiene alquitrán de todo, incluyendo la base de datos. Usted nunca debe descomprimir este archivo, es automáticamente sin envasar por web2py cuando uno lo instala utilizando **admin**.
- **clean up** archivos temporales de una aplicación, tales como sesiones, los errores y los archivos de caché.
- **EDIT** una aplicación.

*Cuando se crea una nueva solicitud **admin** se inicia como un clon de la "welcome" scaffolding app and it already provides a "models/db.py" which creates a SQLite database, connects to it, instantiates Auth, Crud and Service, configures them. It also provides a "controller/default.py" which exposes actions "index", "download", "user" for user management, and "call" para los servicios. En algunos de los siguientes asumimos estos archivos se han eliminado o se sobrescribe.*

3.2 Di Hola

Aquí, como un ejemplo, hemos creado una aplicación web simple que muestra el mensaje de "Hello from MyApp" to the user. We will call this application "myapp". También vamos a agregar un contador que cuenta las veces que el mismo usuario visita la página.

Puede crear una nueva aplicación con sólo escribir su nombre en el buscador de la parte superior derecha de la página en **site admin**.

Create new application

create new application:

Después de pulsar [enviar], la aplicación se crea como una copia de la aplicación integrada de bienvenida.

Installed applications

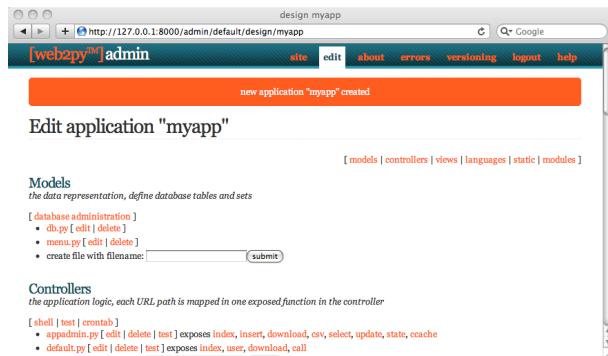
admin [errors clean pack all compile] examples [EDIT about errors clean pack all compile uninstall] myapp [EDIT about errors clean pack all compile uninstall] welcome [EDIT about errors clean pack all compile uninstall]	Version web2py Version 1. click to check for upgrade web2py	Create new application create new application
Upload & install upload application or provide app url		

Para ejecutar la aplicación, visite:

```
1 http://127.0.0.1:8000/myapp
```

Ahora usted tiene una copia de la solicitud de la recepción.

Para editar una aplicación, haga clic en el botón *edit* para la aplicación de nueva creación.



La página **EDIT** le dice qué está dentro de la aplicación. Cada aplicación web2py consta de ciertos archivos, la mayoría de los cuales pertenecen a una de cinco categorías:

- **models**: describir la representación de datos.
- **controllers**: describir la lógica de la aplicación y flujo de trabajo.
- **views**: describir la presentación de datos.
- **languages**: describir la manera de trasladar la presentación de solicitud a otros idiomas.
- **modules**: módulos de Python que pertenecen a la solicitud.
- **static files**: imágenes estáticas, CSS files (40; 41; 42), JavaScript files (43; 44), etc

Todo está perfectamente organizado según el patrón de diseño Modelo-Vista-Controlador. Cada sección en la página *edit* corresponde a una subcarpeta en la carpeta de la aplicación.

Tenga en cuenta que títulos de las secciones se cambian su contenido. Los nombres de carpeta en archivos estáticos también son plegables.

Cada archivo enumerado en la sección correspondiente a un archivo ubicado físicamente en la subcarpeta. Cualquier operación que se realiza en un archivo

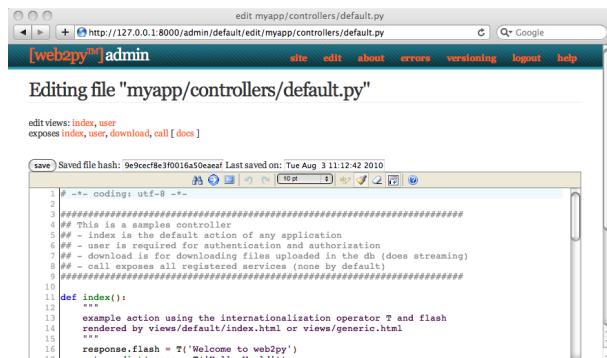
a través de la interfaz de **admin** (crear, editar, eliminar) se puede realizar directamente desde el intérprete de comandos usando su editor favorito.

La aplicación contiene otros tipos de archivos (base de datos, archivos de sesión, los archivos de error, etc), pero que no figuran en la página de *edit*, porque no son creados o modificados por el administrador. Son creados y modificados por la propia demanda.

Los controladores contienen la lógica y el flujo de trabajo de la aplicación. Cada dirección se designa en una llamada a una de las funciones de los controladores (acciones). Hay dos controladores por defecto: La función `appadmin.py` and `"default.py"`. `appadmin`:bold provides the database administrative interface; we do not need it now. `"default.py"` is the controller that you need to edit, the one that is called by default when no controller is specified in the URL. Edit the "index" de la siguiente manera:

```
1 def index():
2     return "Hello from MyApp"
```

Esto es lo que el editor en línea será similar a:



The screenshot shows a web browser window with the following details:

- Title Bar:** edit myapp/controllers/default.py
- Address Bar:** http://127.0.0.1:8000/admin/default/edit/myapp/controllers/default.py
- Header:** [web2py] admin site edit about errors versioning logout help
- Page Title:** Editing file "myapp/controllers/default.py"
- Toolbar:** save, undo, redo, cut, copy, paste, find, select all, font size (10pt), bold, italic, underline, etc.
- Message:** edit views: index, user exposes index, user, download, call [docs]
- Code Editor:**

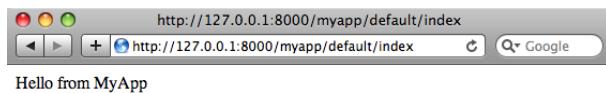
```
1 # -*- coding: utf-8 -*-
2 #
3 ##### This is a samples controller
4 ## - index is the default action of any application
5 ## - auth is automatically created to handle authentication and authorization
6 ## - download is for downloading files uploaded in the db (does streaming)
7 ## - call exposes all registered services (none by default)
8 ##
9 #####
10
11 def index():
12     """
13         example action using the internationalization operator T and flash
14         rendered by views/default/index.html or views/generic.html
15     """
16     response.flash = T('Welcome to web2py')
```

Guardar y volver a la página de *edit*. Haga clic en el enlace del índice para visitar la página de nueva creación.

Cuando visite la dirección

```
1 http://127.0.0.1:8000/myapp/default/index
```

la acción index del controlador por defecto de la aplicación miaplicacion se llama. Devuelve una cadena que muestra el navegador para nosotros. Se debe tener este aspecto:



Ahora, edita la función "index" de la siguiente manera:

```
1 def index():
2     return dict(message="Hello from MyApp")
```

También desde la página *edit*, modifique la vista predeterminada / índice (el nuevo archivo asociado a la acción) y, en este archivo, escriba a:

```
1 <html>
2     <head></head>
3     <body>
4         <h1>{{=message}}</h1>
5     </body>
6 </html>
```

Ahora, la acción devuelve un diccionario la definición de una message. Cuando una acción devuelve un diccionario, web2py busca una vista con el nombre

[controller]/[function].[extension] y lo ejecuta. Aquí [extensión] es la prórroga solicitada. Si no se especifica la extensión, por defecto se "html", y eso es lo que vamos a asumir aquí. Bajo este supuesto, la vista es un archivo HTML que se incluye el código Python usando especiales () etiquetas. En

particular, en el ejemplo, el `{=message}` instruye web2py para reemplazar el código marcado con el valor de la `message` devuelto por la acción. Nótese que aquí no `message` es una palabra clave web2py sino que se define en la acción. Hasta ahora no hemos utilizado las palabras clave web2py.

Si no encuentra web2py la opinión solicitada, utiliza la vista de "generic.html" que viene con cada aplicación.

If another extension of a "html" is specified ("json" for example), and the view file "[controller]/[function].json" is not found, web2py looks for the view "generic.json". web2py viene con generic.html, generic.json, generic.xml y generic.rss. Estos puntos de vista genérico puede ser modificado para cada aplicación individual, y vistas adicionales se pueden agregar fácilmente.

Lea más sobre este tema en el capítulo 9.

Si nos remontamos a la "EDIT" y haga clic en el índice, ahora veremos la siguiente página HTML:



3.3 Vamos a contar

Ahora vamos a agregar un contador a esta página que cuenten cuántas veces el mismo visitante muestra la página. web2py forma automática y transpar-

ente seguimiento de los visitantes a través de sesiones y cookies. Para cada nuevo visitante, se crea una sesión y le asigna un único "session_id". La sesión es un contenedor para las variables que se almacenan en el servidor. El identificador único se envía al navegador a través de una cookie. Cuando el visitante solicita otra página desde la misma aplicación, el navegador envía la cookie, se recuperará por web2py, y el correspondiente período de sesiones se restaura.

Para utilizar el período de sesiones, modificar el controlador predeterminado:

```

1 def index():
2     if not session.counter:
3         session.counter = 1
4     else:
5         session.counter += 1
6     return dict(message="Hello from MyApp", counter=session.counter)

```

Tenga en cuenta que `counter` no es una palabra clave, pero web2py `session` es. Estamos pidiendo web2py para comprobar si existe una variable de contador en la sesión y, si no, para crear uno y lo ponen a 1. Si el contador está ahí, pedimos web2py para aumentar el contador en 1. Finalmente pasamos el valor del contador a la vista.

Una forma más compacta de codificar la misma función es la siguiente:

```

1 def index():
2     session.counter = (session.counter or 0) + 1
3     return dict(message="Hello from MyApp", counter=session.counter)

```

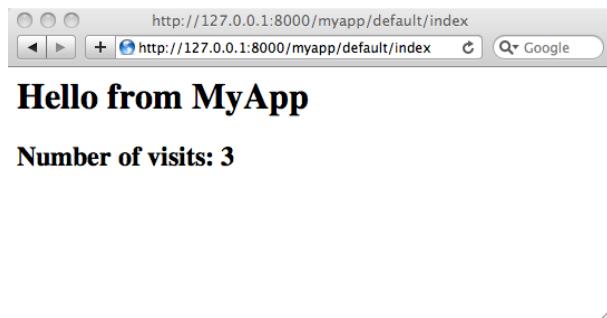
Ahora modificar el fin de añadir una línea que muestra el valor del contador:

```

1 <html>
2     <head></head>
3     <body>
4         <h1>{{=message}}</h1>
5         <h2>Number of visits: {{=counter}}</h2>
6     </body>
7 </html>

```

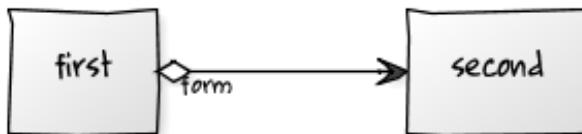
Cuando usted visita la página de índice otra vez (y otra vez) debe obtener la siguiente página HTML:



El contador se asocia a cada visitante, y se incrementa cada vez que el visitante vuelve a cargar la página. Diferentes visitantes ver distintos mostradores.

3.4 Say My Name

Ahora crea dos páginas (primera y segunda), donde la primera página crea un formulario, le pregunta el nombre del visitante, y redirige a la segunda página, que acoge a los visitantes por su nombre.



Escribe las acciones correspondientes en el controlador por defecto:

```

1 def first():
2     return dict()
3
4 def second():
5     return dict()
  
```

A continuación, cree una vista para el "default/first.html" primera acción:

Models

Controllers

Views

the presentations layer, views are also known as templates

- `__init__.py` [edit | delete]
- `appadmin.html` [edit | delete] extends `layout.html`
- `default/index.html` [edit | delete]
- `default/user.html` [edit | delete] extends `layout.html`
- `generic.html` [edit | delete] extends `layout.html`
- `generic.json` [edit | delete]
- `generic.load` [edit | delete]
- `generic.rss` [edit | delete]
- `generic.xml` [edit | delete]
- `layout.html` [edit | delete] includes `web2py_ajax.html`
- `web2py_ajax.html` [edit | delete]
- create file with filename:

y escriba:

```

1 {{extend 'layout.html'}}
2 What is your name?
3 <form action="second">
4   <input name="visitor_name" />
5   <input type="submit" />
6 </form>
```

Finalmente, cree una vista "default/second.html" para la segunda acción:

```

1 {{extend 'layout.html'}}
2 <h1>Hello {{=request.vars.visitor_name}}</h1>
```

En ambos puntos de vista se ha ampliado el punto de vista básico "layout.html" que viene con web2py. La vista de diseño mantiene el aspecto de las dos páginas coherente. El archivo de diseño se pueden editar y volver a montarse fácilmente, ya que principalmente contiene código HTML.

Si ahora visitar la primera página, escriba su nombre:



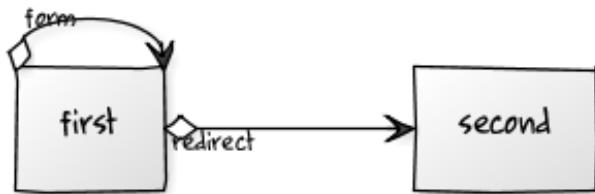
y envía el formulario, recibirá un saludo:



3.5 Devoluciones de datos

El mecanismo para el envío de formularios que utilizamos antes es muy común, pero no es una buena práctica de programación. Todas las entradas deben ser validados y, en el ejemplo anterior, la carga de validación caería en la segunda acción. Así, la acción que realiza la validación es diferente de la acción que generó el formulario. Esto puede causar que la redundancia en el código.

Un modelo mejor para el envío del formulario es enviar los formularios a la misma acción que los genera, en nuestro ejemplo la página de "first". The "first" action should receive the variables, process them, store them server side, and redirect the visitor to the "second", que recupera las variables. Este mecanismo se denomina *postback*.



Puede modificar el controlador por defecto de la siguiente manera para implementar auto-presentación:

```

1 def first():
2     if request.vars.visitor_name:
3         session.visitor_name = request.vars.visitor_name
4         redirect(URL('second'))
5     return dict()
6
7 def second():
8     return dict()

```

En consecuencia, es necesario modificar la vista "default/first.html":

```

1 {{extend 'layout.html'}}
2 What is your name?
3 <form>
4     <input name="visitor_name" />
5     <input type="submit" />
6 </form>

```

y la vista "default/second.html" necesita recuperar los datos de la session lugar de desde el `request.vars`:

```

1 {{extend 'layout.html'}}
2 <h1>Hello {{=session.visitor_name or "anonymous"}}</h1>

```

Desde el punto de vista del visitante, la auto-presentación se comporta exactamente igual que la implementación anterior. No hemos añadido la validación todavía, pero ahora está claro que la validación debe ser realizada por la primera acción.

Este enfoque también es mejor porque el nombre del visitante permanece en

el período de sesiones, y se puede acceder por todas las acciones y puntos de vista en las aplicaciones sin tener que pasar alrededor de forma explícita.

Tenga en cuenta que si el "second" action is ever called before a visitor name is set, it will display "Hello anonymous" session.visitor_name None porque los rendimientos. También podríamos agregar el siguiente código en el controlador (dentro o fuera de la función second:

```
1 if not request.function=='first' and not session.visitor_name:
2     redirect(URL('first'))
```

Se trata de un mecanismo general que puede utilizar para hacer cumplir la autorización de los controladores, a pesar de ver el capítulo 8 de un método más de gran alcance.

Con web2py podemos avanzar un paso más allá y pedir web2py para generar la forma para nosotros, incluida la validación. web2py provee helpers (FORM, INPUT, TEXTAREA y SELECT / OPCIÓN) con los mismos nombres como el equivalente etiquetas HTML. Pueden ser utilizados para construir las formas, ya sea en el controlador o en la vista.

Por ejemplo, aquí hay una forma posible de volver a escribir la primera acción:

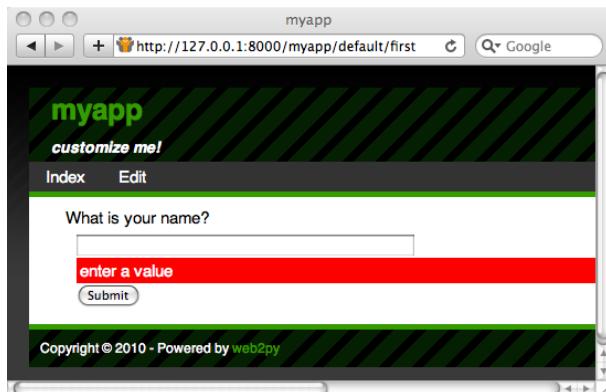
```
1 def first():
2     form = FORM(INPUT(_name='visitor_name', requires=IS_NOT_EMPTY()),
3                 INPUT(_type='submit'))
4     if form.accepts(request.vars, session):
5         session.visitor_name = form.vars.visitor_name
6         redirect(URL('second'))
7     return dict(form=form)
```

cuando decimos que la etiqueta de formulario contiene dos etiquetas INPUT. Los atributos de las etiquetas de entrada son especificados por los argumentos con nombre a partir de subrayado. El argumento requires no es un atributo de etiqueta (porque no comienza por guión bajo) pero establece un validador para el valor de visitor_name.

El objeto `form` puede ser fácilmente serializado en HTML mediante la inserción en la vista de "default/first.html".

```
1 {{extend 'layout.html'}}
2 What is your name?
3 {{=form}}
```

El método se aplica `form.accepts` los validadores. Si el formulario de auto-envía supera la validación, que almacena las variables de la sesión y vuelve a dirigir como antes. Si el formulario no pasa la validación, los mensajes de error se insertan en la forma y muestra al usuario, se muestra a continuación:



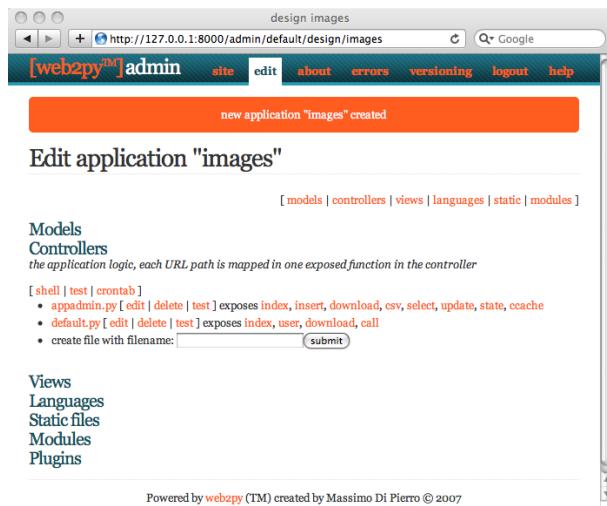
En la siguiente sección vamos a mostrar cómo las formas se pueden generar automáticamente a partir de un modelo.

3.6 Una imagen de blog

Aquí, como otro ejemplo, queremos crear una aplicación web que permite al administrador para enviar imágenes y darles un nombre, y permite a los visitantes del sitio web para ver las imágenes y presentar sus observaciones.

Al igual que antes, crear la nueva aplicación desde la página de **site** en **admin**

y vaya a la página *edit*:



Comenzamos creando un modelo, una representación de los datos persistentes en la aplicación (las imágenes para subir, sus nombres, y los comentarios). Primero, usted necesita para crear o editar un archivo de modelo que, por falta de imaginación, que llamamos "db.py". We assume the code below will replace any existing code in "db.py". Modelos y controladores deberán tener una extensión .py, ya que son el código Python. Si la extensión no está previsto, es incluido por web2py. Vistas así, tener una extensión .html, ya que principalmente contienen código HTML.

Editar el botón "db.py" file by clicking the corresponding "edit":



y escriba lo siguiente:

```

1 db = DAL("sqlite://storage.sqlite")
2
3 db.define_table('image',
4     Field('title'),
5     Field('file', 'upload'))
6
7 db.define_table('comment',
8     Field('image_id', db.image),
9     Field('author'),
10    Field('email'),
11    Field('body', 'text'))
12
13 db.image.title.requires = IS_NOT_IN_DB(db, db.image.title)
14 db.comment.image_id.requires = IS_IN_DB(db, db.image.id, '%(title)s')
15 db.comment.author.requires = IS_NOT_EMPTY()
16 db.comment.email.requires = IS_EMAIL()
17 db.comment.body.requires = IS_NOT_EMPTY()
18
19 db.comment.image_id.writable = db.comment.image_id.readable = False

```

Vamos a analizar esta línea por línea.

La línea 1 define una variable global llamada `db` que representa la conexión de base de datos. En este caso se trata de una conexión a una base de datos SQLite almacenada en el archivo de "applications/images/databases/storage.sqlite". En el caso de SQLite, si la base de datos no existe, se crea. Puede cambiar el nombre del archivo, así como el nombre de la `db` variable global, pero es conveniente darles el mismo nombre, para que sea fácil de recordar.

Líneas 3-5 definir una tabla de "image". `define_table` es un método del objeto `db`. El primer argumento, "image", is the name of the table we are defining. The other arguments are the fields belonging to that table. This table has a field called "title", a field called "file", and a field called "id" that serves as the table primary key ("id" is not explicitly declared because all tables have an id field by default). The field "title" is a string, and the field "file" is of type "upload". "upload" es un tipo especial de campo utilizado por la capa de abstracción de datos web2py (DAL) para almacenar los nombres de los archivos subidos. web2py sabe cómo subir archivos (a través de streaming si son grandes), cambiar el nombre de un lugar seguro, y almacenarlos.

Cuando una tabla se define, web2py toma una de varias acciones posibles: a) si la tabla no existe, se crea la tabla; b) si la tabla existe y no se corresponde con la definición, el cuadro es el mismo cambio, y si un campo tiene un tipo diferente, web2py intenta convertir su contenido, c) si la tabla existe y corresponde a la definición, web2py no hace nada.

Este comportamiento se denomina "migration". En las migraciones web2py son automáticas, pero se puede desactivar para cada tabla pasando `migrate=False` como el último argumento de `define_table`.

Líneas 11.07 definir otra tabla llamada "comment". Un comentario "author", an "email" (we intend to store the email address of the author of the comment), a "body" of type "text" (we intend to use it to store the actual comment posted by the author), and an "image_id" tiene un campo de tipo de referencia que señala a `db.image.id` través del campo.

En las líneas 13 `db.image.title` "title" of table "image" representa el campo. El `requires` atributo le permite establecer los requisitos y limitaciones a las que se harán cumplir por las formas web2py. Aquí se requiere que el "title" es único

```
IS_NOT_IN_DB(db, db.image.title)
```

Los objetos que representan estas restricciones son llamados los validadores.

validadores múltiples pueden agruparse en una lista. Los validadores se ejecutan en el orden en que aparecen. `IS_NOT_IN_DB(a, b)` es un validador especial que comprueba que el valor de una `b` de campo para un nuevo registro no aparece ya en `a`.

La línea 14 requiere que el campo está en "image_id" of table "comment" `db.image.id`. En cuanto a la base de datos se refiere, que ya había declarado esto cuando se definió el "comment" tabla. Ahora estamos diciendo de forma explícita el modelo que esta condición debe ser aplicada por web2py, también, a nivel de procesamiento de formularios cuando un nuevo comentario sea publicado, de manera que los valores no válidos no se propagan a partir de formas de entrada a la base de datos. También exigimos que el "image_id" be represented by the "title", '`%(title)s`', del registro correspondiente.

La línea 18 indica que el "image_id" of table "comment" campo no deben consignarse en los formularios, `writable=False` y ni siquiera en forma de sólo lectura, `readable=False`.

El significado de los validadores en las líneas 15-17 debería ser obvia.

Aviso de que el

```
1 db.comment.image_id.requires = IS_IN_DB(db, db.image.id, '%(title)s')
```

validador puede ser omitido (y sería automática) es que íbamos a especificar un formato para la representación de una imagen:

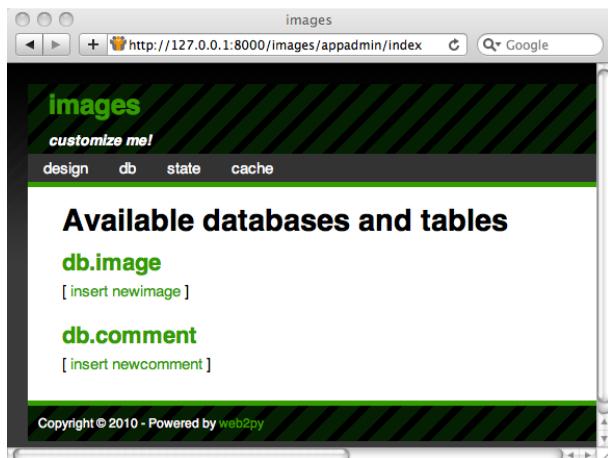
```
1 db.define_table('image', ..., format='%(title)s')
```

donde el formato puede ser una cadena o una función que toma un registro y devuelve una cadena.

Una vez que el modelo está definida, si no hay errores, web2py crea una interfaz de administración aplicación para administrar la base de datos. Usted acceder a él a través del enlace en la página de "database administration" *edit* o directamente:

1 http://127.0.0.1:8000/images/appadmin

Aquí está una captura de pantalla de la interfaz de **appadmin**:



Esta interfaz se codifica en el controlador llamado "appadmin.py" and the corresponding view "appadmin.html". De ahora en adelante, nos referiremos a esta interfaz simplemente como **appadmin**. Permite al administrador para insertar nuevos registros de base de datos, editar y borrar los registros existentes, mesas de exploración, se une a la base de datos y llevar a cabo.

El **appadmin** primera vez que se accede, el modelo se ejecuta y se crean las tablas. El DAL web2py traduce el código Python en instrucciones SQL que son específicos de la parte posterior, seleccione la base de datos de gama (SQLite en este ejemplo). Usted puede ver el SQL generado desde la página *edit* haciendo clic en el "sql.log" link under "models". Tenga en cuenta que el vínculo no está presente hasta que las tablas se han creado.

```

[edit]
1. timestamp: 2010-08-04T06:52:21.990465
2. CREATE TABLE image(
3.     id INTEGER PRIMARY KEY AUTOINCREMENT,
4.     title CHAR(512),
5.     file CHAR(512)
6. );
7. success!
8. timestamp: 2010-08-04T06:52:21.999322
9. CREATE TABLE comment(
10.    id INTEGER PRIMARY KEY AUTOINCREMENT,
11.    image_id INTEGER REFERENCES image(id) ON DELETE CASCADE,
12.    author CHAR(512),
13.    email CHAR(512),
14.    body TEXT
15. );
16. success!
17.

```

Si se va a modificar el modelo **appadmin** y el acceso de nuevo, web2py generaría SQL para alterar las tablas existentes. El SQL generado se registra en "sql.log".

Ahora vuelve a **appadmin** y tratar de insertar un registro nuevo de la imagen:

database db table image

New Record

Title:

File: earth

web2py se ha traducido en la carpeta `db.image.file` "upload" field into an upload form for the file. When the form is submitted and an image file is uploaded, the file is renamed in a secure way that preserves the extension, it is saved with the new name under the application "uploads", y el nuevo nombre se almacena en el campo de `db.image.file`. Este proceso está diseñado

para prevenir los ataques de directorio transversal.

Tenga en cuenta que cada tipo de campo se representa por una *widget*. reproductores por defecto se puede reemplazar.

Al hacer clic en un nombre de tabla en **appadmin**, web2py realiza una selección de todos los registros en la tabla actual, señaladas por el

```
1 db.image.id > 0
```

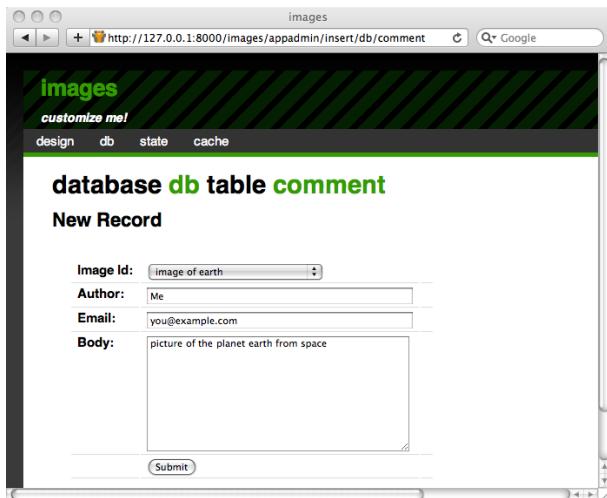
consulta DAL y deja el resultado.

The screenshot shows a web browser window titled "images" with the URL <http://127.0.0.1:8000/images/appadmin/select/db?query=db.image.id>0>. The page contains a form with fields for "Query" (set to "db.image.id>0"), "Update", and "Delete". Below the form is a note about building queries using JOIN, AND, OR, NOT, and update expressions. A table titled "1 selected" displays one row of data: image.id (1), image.title (image of earth), and image.file (file). At the bottom, there is an "Import/Export" section with options to export as CSV or import from CSV.

image.id	image.title	image.file
1	image of earth	file

Puede seleccionar un conjunto diferente de registros por parte de editar una consulta SQL y presionando "apply".

Para modificar o borrar un solo registro, haga clic en el número de identificador de registro.



Debido al validador `IS_IN_DB`, "image_id" el campo de referencia se representa mediante un menú desplegable. Los elementos de la lista desplegable se almacenan como teclas (`db.image.id`), pero están representados por sus `db.image.title`, según lo especificado por el validador.

Validadores son objetos poderosos que saben cómo representar los campos, los valores del filtro de campo, generar errores, y los valores de formato extraído del campo.

La figura siguiente muestra lo que sucede cuando envía un formulario que no pasa la validación:

Image Id:	<input type="text" value="image of earth"/>
Author:	<input type="text" value="enter a value"/>
Email:	<input type="text" value="enter a valid email address"/>
Body:	<input type="text" value="enter a value"/>

Las mismas formas que se generan automáticamente por **appadmin** también se puede generar mediante programación a través de la ayuda SQLFORM y está integrada en las aplicaciones de usuario. Estas formas son CSS de usar, y puede ser personalizado.

Cada aplicación tiene su propia **appadmin**, por lo tanto, **appadmin** misma puede ser modificada sin afectar a otras aplicaciones.

Hasta ahora, la aplicación sabe cómo almacenar los datos, y hemos visto cómo acceder a la base de datos a través de **appadmin**. Acceso a la **appadmin** está restringido al administrador, y no pretende ser una interfaz web para la aplicación de producción, de ahí la siguiente parte de este recorrido. En concreto queremos crear:

- Una página que muestra "index" todas las imágenes disponibles ordenados por títulos y enlaces a páginas de detalles de las imágenes.
- Una página que muestra "show/[id]" al visitante la imagen solicitada y permite al visitante ver y enviar comentarios.
- Una acción "download/[name]" para descargar las imágenes subidas.

Esto se representa esquemáticamente en:



Volver a la página de *edit* y "default.py" editar el controlador, en sustitución de su contenido con el texto siguiente:

```

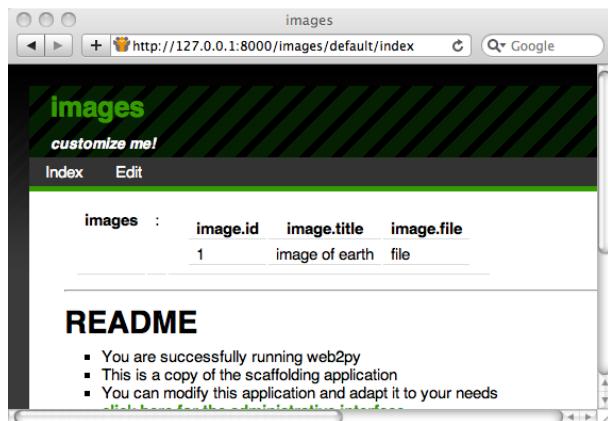
1 def index():
2     images = db().select(db.image.ALL, orderby=db.image.title)
3     return dict(images=images)

```

Esta acción devuelve un diccionario. Las claves de los elementos en el diccionario se interpretan como variables pasadas a la vista asociada a la acción. Si no hay ningún punto de vista, la acción se representa por el punto de vista "generic.html" que se entrega con todas las aplicaciones web2py.

El índice de acciones realiza una selección de todos los campos (db.image.ALL) de imagen en la mesa, ordenados por db.image.title. El resultado de la selección es un objeto que contiene los registros Rows. Asignar a una variable local llamada `images` devuelto por la acción a la vista. `images` es repetible y sus elementos son las filas seleccionadas. Para cada fila de las columnas se puede acceder como diccionarios: `images[0]['title']` o equivalente como `images[0].title`.

Si usted no escribe una opinión, el diccionario es traducido por "views/-generic.html" y una llamada a la acción index se vería así:



No ha creado una vista para esta acción, sin embargo, hace tan web2py el conjunto de registros en forma de tabla sin formato.

Proceda a crear una vista para la acción index. Volver al administrador, editar "default/index.html" y reemplazar su contenido con el texto siguiente:

```

1 {{extend 'layout.html'}}
2 <h1>Current Images</h1>
3 <ul>
4 {{for image in images:}}
5 {{=LI(A(image.title, _href=URL("show", args=image.id)))}}
6 {{pass}}
7 </ul>
```

La primera cosa a notar es que esta perspectiva es HTML puro, con placas especiales {...}. El código incrustado en {...} es puro código Python con una advertencia: el sangrado es irrelevante. Los bloques de código de inicio con las líneas que terminan en dos puntos (:) y terminan en las líneas que empiezan con la palabra clave de pass. En algunos casos, el final de un bloque es evidente por el contexto y el uso de pass no es necesario.

Las líneas 5-7 lazo sobre las filas de la imagen y para cada fila de visualización de la imagen:

```
1 LI(A(image.title, _href=URL('show', args=image.id))
```

Esta es una etiqueta `...` que contiene una etiqueta que contiene el `...` `image.title`. El valor de la referencia de hipertexto (atributo href) es:

```
1 URL('show', args=image.id)
```

es decir, la dirección URL en la misma aplicación y de control según la solicitud actual que llama a la función llamada "show", pasando un solo argumento a la función, `args=image.id`. LI, A, etc son ayudantes web2py que se asignan a las correspondientes etiquetas HTML. Sus argumentos no identificado se interpretan como objetos a serializar y se incluirán en `innerHTML` de la etiqueta. Nombrado argumentos que comienzan con un guión bajo (por ejemplo, `_href`) se interpretan como atributos de la etiqueta, pero sin el carácter de subrayado. Por ejemplo `_href` es el atributo `href`, `_class class` es el atributo, etc

Como ejemplo, la siguiente declaración:

```
1 {{=LI(A('something', _href=URL('show', args=123)))}}
```

se representa como:

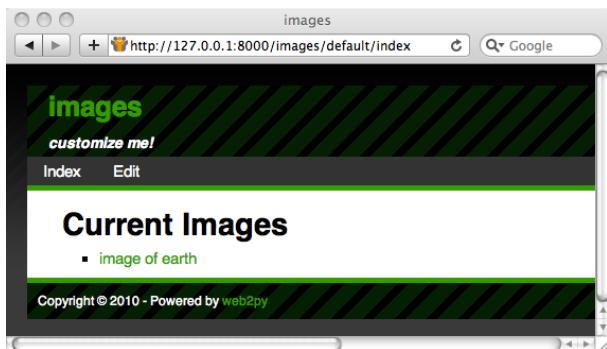
```
1 <li><a href="/images/default/show/123">something</a></li>
```

Un puñado de ayudantes (`INPUT`, `TEXTAREA`, `OPTION` y `SELECT`) también apoyan algunos atributos especiales llamados que no empiezan con guión bajo (`value`, `requires` y). Son importantes para la creación de formularios personalizados y se discutirá más adelante.

Volver a la página de `edit.`, Indicando que "default.py exposes index". By clicking on "index", se puede visitar la recién creada página:

```
1 http://127.0.0.1:8000/images/default/index
```

que se parece a:



Si hace clic en el enlace del nombre de la imagen, se le dirige a:

```
1 http://127.0.0.1:8000/images/default/show/1
```

y esto resulta en un error, ya que aún no ha creado una acción llamada "show" in controller "default.py".

Vamos a editar el controlador de "default.py" y reemplace su contenido con:

```
1 def index():
2     images = db().select(db.image.ALL, orderby=db.image.title)
3     return dict(images=images)
4
5 def show():
6     image = db(db.image.id==request.args(0)).select().first()
7     form = SQLFORM(db.comment)
8     form.vars.image_id = image.id
9     if form.accepts(request.vars, session):
10         response.flash = 'your comment is posted'
11     comments = db(db.comment.image_id==image.id).select()
12     return dict(image=image, comments=comments, form=form)
13
14 def download():
15     return response.download(request, db)
```

El controlador contiene dos acciones: "show" and "download". La acción "show" selecciona la imagen con el id analizados del args solicitud y todos los comentarios relacionados con la imagen. "show" then passes everything to the view "default/show.html".

El ID de la imagen que hace referencia:

```
1 URL('show', args=image.id)
```

en "default/index.html", se puede acceder como: `request.args(0)` "show" de la acción.

La acción "download" espera un nombre de archivo en `request.args(0)`, construye una ruta a la ubicación donde se supone que ese archivo a ser, y lo envía de vuelta al cliente. Si el archivo es demasiado grande, que transmite el archivo sin incurrir en ninguna sobrecarga de la memoria.

Notificación de las siguientes afirmaciones:

- La línea 7 crea un formulario de inserción SQLFORM para la tabla `db.comment` usando sólo los campos especificados.
- La línea 8 se establece el valor para el campo de referencia, que no forma parte del formulario de entrada, ya que no está en la lista de los campos mencionados anteriormente.
- La línea 9 procesa el formulario enviado (las variables de formulario presentado en `request.vars`) en el actual período de sesiones (la sesión se utiliza para prevenir el envío de matrimonio, y para hacer cumplir de navegación). Si las variables de formulario presentado son validados, el comentario se añade en la tabla de `db.comment`, de lo contrario el formulario se modifica para incluir mensajes de error (por ejemplo, si la dirección de correo electrónico del autor no es válido). Todo esto se hace en la línea 9!.
- La línea 10 sólo se ejecuta si el formulario es aceptado, después de que el acta está inserta en la tabla de base de datos. `response.flash` es una variable web2py que se muestra en los puntos de vista y se utiliza para notificar al usuario de que algo pasó.
- La línea 11 selecciona todos los comentarios que hacen referencia a la imagen actual.

El controlador "download" action is already defined in the "default.py" de la solicitud de andamios.

El "download" action does not return a dictionary, so it does not need a view. The "show" action, though, should have a view, so return to admin:bold and create a new view called "default/show.html".

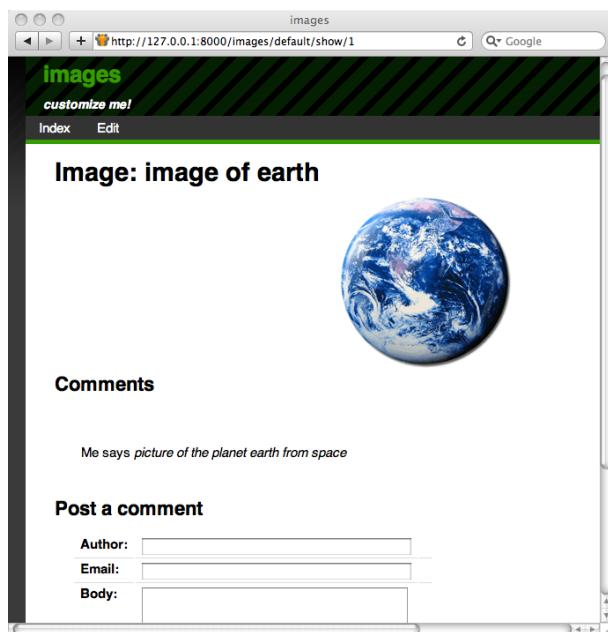
Editar este archivo nuevo y reemplazar su contenido con el texto siguiente:

```

1 {{extend 'layout.html'}}
2 <h1>Image: {{=image.title}}</h1>
3 <center>
4 
6 </center>
7 {{if len(comments):}}
8     <h2>Comments</h2><br /><p>
9         {{for comment in comments:}}
10            <p>{{=comment.author}} says <i>{{=comment.body}}</i></p>
11            {{pass}}</p>
12 {{else:}}
13     <h2>No comments posted yet</h2>
14 {{pass}}
15 <h2>Post a comment</h2>
16 {{=form}}}
```

Esta vista muestra el **image.file** llamando a la acción dentro de una etiqueta "download" . Si hay observaciones, se realiza un bucle sobre ellos y muestra cada uno.

Así es como todo lo que van a aparecer a un visitante.



Cuando un usuario envía un comentario a través de esta página, el comentario se almacena en la base de datos y se adjuntará en la parte inferior de la página.

3.7 Agregar CRUD

web2py también proporciona un CRUD (Crear / Leer / Actualizar / Borrar) de la API que simplifica aún más formas. Para utilizar CRUD es necesario definirlo en alguna parte, como en el módulo "db.py":

```
1 from gluon.tools import Crud
2 crud = Crud(globals(), db)
```

Estas dos líneas se encuentran ya en la aplicación andamios.

El objeto crud proporciona a nivel de métodos de alta, por ejemplo:

```
1 form = crud.create(table)
```

que se puede utilizar para sustituir el modelo de programación:

```
1 form = SQLFORM(table)
2 if form.accepts(request.post_vars,session):
3     session.flash = '...'
4     redirect('...')
```

Aquí, volvemos a escribir la acción anterior utilizando "show" porquería y haciendo algunas mejoras más:

```
1 def show():
2     image = db.image(request.args(0)) or redirect(URL('index'))
3     db.comment.image_id.default = image.id
4     form = crud.create(db.comment,
5                         message='your comment is posted',
6                         next=URL(args=image.id))
7     comments = db(db.comment.image_id==image.id).select()
8     return dict(image=image, comments=comments, form=form)
```

En primer lugar anuncio que hemos utilizado la sintaxis

```
1 db.image(request.args(0)) or redirect(...)
```

a buscar el expediente requerido. Dado que ninguno de retorno 'table(id)' si el registro no se encuentra, podemos utilizar `or redirect(...)`, en este caso en una línea.

El argumento de `next crud.create` es la URL para redirigir a después de que el formulario es aceptado. El argumento `message` es la que se mostrará después de la aceptación. Puede leer más sobre CRUD en el capítulo 7.

3.8 Adición de autenticación

El API para web2py basada en funciones de control de acceso es muy sofisticado, pero por ahora nos limitaremos a restringir el acceso a la acción show a los usuarios autenticados, se aplaza una discusión más detallada del capítulo 8.

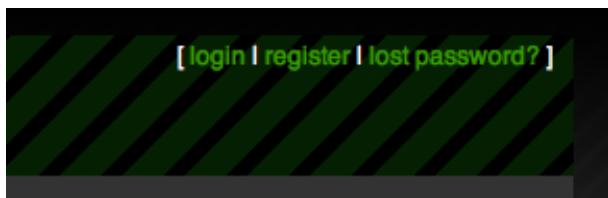
Para limitar el acceso a los usuarios autenticados, tenemos que completar tres pasos. En un modelo, por ejemplo, "db.py", tenemos que añadir:

```
1 from gluon.tools import Auth  
2 auth = Auth(globals(), db)  
3 auth.define_tables()
```

En nuestro controlador, tenemos que añadir una acción:

```
1 def user():
2     return dict(form=auth())
```

Esto es suficiente para permitir la conexión, registro, cerrar sesión, etc páginas. El diseño predeterminado también mostrará las opciones a las páginas correspondientes en la esquina superior derecha.



Ahora podemos decorar las funciones que usted quiere restringir, por ejemplo:

```

7 comments = db(db.comment.image_id==image.id).select()
8 return dict(image=image, comments=comments, form=form)

```

Cualquier intento de acceso

```

1 http://127.0.0.1:8000/images/default/show/[image_id]

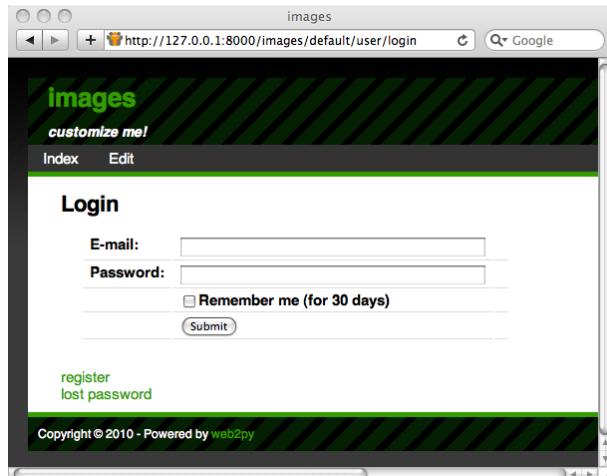
```

requerirá entrada. Si el usuario no está conectado, el usuario será redirigido a

```

1 http://127.0.0.1:8000/images/default/user/login

```



La función user también expone, entre otras, las siguientes acciones:

```

1 http://127.0.0.1:8000/images/default/user/logout
2 http://127.0.0.1:8000/images/default/user/register
3 http://127.0.0.1:8000/images/default/user/profile
4 http://127.0.0.1:8000/images/default/user/change_password
5 http://127.0.0.1:8000/images/default/user/request_reset_password
6 http://127.0.0.1:8000/images/default/user/retrieve_username
7 http://127.0.0.1:8000/images/default/user/retrieve_password
8 http://127.0.0.1:8000/images/default/user/verify_email
9 http://127.0.0.1:8000/images/default/user/impersonate
10 http://127.0.0.1:8000/images/default/user/notAuthorized

```

Ahora, una vez que el usuario primero debe registrarse para poder acceder y leer / enviar comentarios.

Tanto el objeto auth y user la función ya están definidos en la solicitud de andamios. El objeto auth es altamente personalizable y se puede tratar con la verificación de correo electrónico, las aprobaciones de registro, CAPTCHA, y entrada métodos alternativos.

3.9 Configuring el diseño

Usted puede configurar el diseño de default editando "views/layout.html" but you can also configure it without editing the HTML. In fact the "static/base.css" stylesheet if very well documented and described in Chapter 5. You can change color, columns, size, borders and background without editing the HTML. If you want to edit the menu, the title or the subtitle, you can do so in any model file. The scaffolding app, sets default values of these parameters in the file "models/menu.py":

```

1 response.title = request.application
2 response.subtitle = T('customize me!')
3 response.meta.author = 'you'
4 response.meta.description = 'describe your app'
5 response.meta.keywords = 'bla bla bla'
6 response.menu = [ [ 'Index', False, URL('index') ] ]

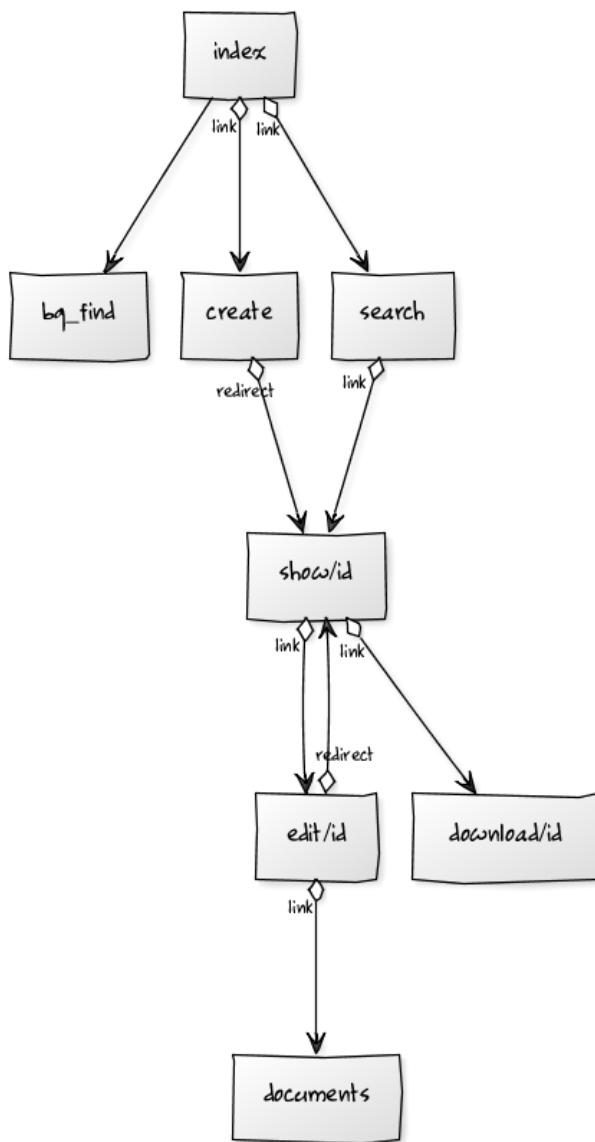
```

3.10 Un Wiki

En esta sección, se construye un wiki, a partir de cero y sin utilizar la funcionalidad extendida proporcionada por plugin_wiki que se describe en el capítulo 13. El visitante podrá crear páginas, buscar por ellos (por título), y editarlos. El visitante también podrá enviar comentarios (exactamente como en las aplicaciones anteriores), así como publicar documentos (como archivos

adjuntos a las páginas) y vincularlos a las páginas. Por convenio, se adopta la sintaxis Markmin de nuestra sintaxis wiki. También pondrá en marcha una página de búsqueda con el Ajax, un feed RSS de las páginas, y un controlador para buscar las páginas a través de XML-RPC (46).

El diagrama siguiente, se enumeran las acciones que se necesitan para implementar y los vínculos que la intención de construir entre ellos.



Comience por crear una aplicación nueva andamios, dándole el nombre "my-wiki".

El modelo debe contener tres tablas: la página, el comentario, y el documento. Ambos comentarios y la página de referencia del documento, ya que pertenecen a la página. Un documento contiene un campo de archivo de subida como en el tipo de solicitud de imágenes anteriores.

Aquí está el modelo completo:

```
1 db = DAL('sqlite://storage.sqlite')
2
3 from gluon.tools import *
4 auth = Auth(globals(),db)
5 auth.define_tables()
6 crud = Crud(globals(),db)
7
8 db.define_table('page',
9     Field('title'),
10    Field('body', 'text'),
11    Field('created_on', 'datetime', default=request.now),
12    Field('created_by', db.auth_user, default=auth.user_id),
13    format='%(title)s')
14
15 db.define_table('comment',
16     Field('page_id', db.page),
17     Field('body', 'text'),
18     Field('created_on', 'datetime', default=request.now),
19     Field('created_by', db.auth_user, default=auth.user_id))
20
21 db.define_table('document',
22     Field('page_id', db.page),
23     Field('name'),
24     Field('file', 'upload'),
25     Field('created_on', 'datetime', default=request.now),
26     Field('created_by', db.auth_user, default=auth.user_id),
27     format='%(name)s')
28
29 db.page.title.requires = IS_NOT_IN_DB(db, 'page.title')
30 db.page.body.requires = IS_NOT_EMPTY()
31 db.page.created_by.readable = db.page.created_by.writable = False
32 db.page.created_on.readable = db.page.created_on.writable = False
33
34 db.comment.body.requires = IS_NOT_EMPTY()
35 db.comment.page_id.readable = db.comment.page_id.writable = False
36 db.comment.created_by.readable = db.comment.created_by.writable = False
37 db.comment.created_on.readable = db.comment.created_on.writable = False
38
39 db.document.name.requires = IS_NOT_IN_DB(db, 'document.name')
40 db.document.page_id.readable = db.document.page_id.writable = False
```

```

41 db.document.created_by.readable = db.document.created_by.writable = False
42 db.document.created_on.readable = db.document.created_on.writable = False

```

Edite el controlador de "default.py" y crear las siguientes acciones:

- Índice: Lista de todas las páginas wiki
- crear: puesto de otra página wiki
- mostrar: muestra una página wiki y sus comentarios, y añadir comentarios
- editar: editar una página existente
- documentos: gestión de los documentos adjuntos a una página
- descargar: descargar un documento (como en el ejemplo, imágenes)
- Búsqueda: mostrar un cuadro de búsqueda y, a través de una devolución de llamada Ajax, devolver todos los títulos coincidentes tal y como el visitante tipos
- bg_find: la función de devolución de llamada Ajax. Devuelve el código HTML que se incrusta en la página de búsqueda, mientras que los tipos de visitantes.

Aquí está el controlador de "default.py":

```

1 def index():
2     """ this controller returns a dictionary rendered by the view
3         it lists all wiki pages
4     >>> index().has_key('pages')
5     True
6     """
7     pages = db().select(db.page.id,db.page.title,orderby=db.page.title)
8     return dict(pages=pages)
9
10 @auth.requires_login()
11 def create():
12     "creates a new empty wiki page"
13     form = crud.create(db.page, next = URL('index'))
14     return dict(form=form)
15
16 def show():
17     "shows a wiki page"
18     this_page = db.page(request.args(0)) or redirect(URL('index'))

```

```

19     db.comment.page_id.default = this_page.id
20     form = crud.create(db.comment) if auth.user else None
21     pagecomments = db(db.comment.page_id==this_page.id).select()
22     return dict(page=this_page, comments=pagecomments, form=form)
23
24 @auth.requires_login()
25 def edit():
26     "edit an existing wiki page"
27     this_page = db.page(request.args(0)) or redirect(URL('index'))
28     form = crud.update(db.page, this_page,
29                         next = URL('show', args=request.args))
30     return dict(form=form)
31
32 @auth.requires_login()
33 def documents():
34     "lists all documents attached to a certain page"
35     this_page = db.page(request.args(0)) or redirect(URL('index'))
36     db.document.page_id.default = this_page.id
37     form = crud.create(db.document)
38     pagedocuments = db(db.document.page_id==this_page.id).select()
39     return dict(page=this_page, documents=pagedocuments, form=form)
40
41 def user():
42     return dict(form=auth())
43
44 def download():
45     "allows downloading of documents"
46     return response.download(request, db)
47
48 def search():
49     "an ajax wiki search page"
50     return dict(form=FORM(INPUT(_id='keyword', _name='keyword',
51                             _onkeyup="ajax('bg_find', ['keyword'], 'target');")),
52                 target_div=DIV(_id='target'))
53
54 def bg_find():
55     "an ajax callback that returns a <ul> of links to wiki pages"
56     pattern = '%' + request.vars.keyword.lower() + '%'
57     pages = db(db.page.title.lower().like(pattern))\
58             .select(orderby=db.page.title)
59     items = [A(row.title, _href=URL('show', args=row.id)) \
60             for row in pages]
61     return UL(*items).xml()

```

Líneas 2-6 ofrecer un comentario de la acción index. Líneas 4-5 dentro del comentario se interpreta como código python prueba (doctest). Las pruebas se pueden ejecutar a través de la interfaz de administración. En este caso los

ensayos se comprueba que la acción index se ejecuta sin errores.

Las líneas 18, 27 y 35 intentan buscar un registro page con el id en request.args(0).

Línea 13, 20 y 37 define y el proceso de crear formularios, para una nueva página y un comentario nuevo y un nuevo documento, respectivamente.

La línea 28 define y procesar un formulario de actualización de una página wiki.

Algunos magia que sucede en la línea 51. El atributo de la onkeyup "keyword" etiqueta INPUT está establecido. Cada vez que el visitante suelta una tecla, el código JavaScript insertado en el atributo onkeyup se ejecuta en el cliente. Aquí está el código JavaScript:

```
1 ajax('bg_find', ['keyword'], 'target');
```

ajax es una función de JavaScript definida en el archivo de "web2py_ajax.html" which is included by the default "layout.html". It takes three parameters: the URL of the action that performs the synchronous callback ("bg_find"), a list of the IDs of variables to be sent to the callback ([{"keyword"}]), and the ID where the response has to be inserted ("target").

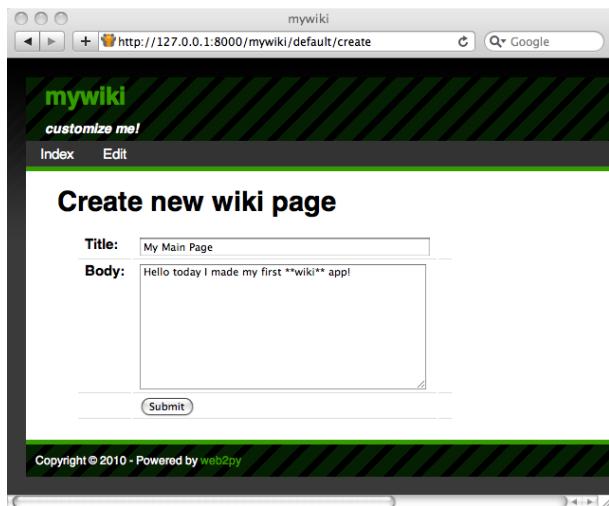
Tan pronto como algo que escriba en el cuadro de búsqueda y soltar una tecla, el cliente llama al servidor y envía el contenido de la "palabra clave" sobre el terreno, y, cuando la severa respuesta, la respuesta está incrustado en la propia página como innerHTML el 'objetivo' etiqueta.

La etiqueta 'objetivo' es un DIV definido en la línea 75. Podría haber sido definidos en el punto de vista también.

Aquí está el código para la vista "default/create.html":

```
1 {{extend 'layout.html'}}
2 <h1>Create new wiki page</h1>
3 {{=form}}
```

Si usted visita la página **create**, ve lo siguiente:



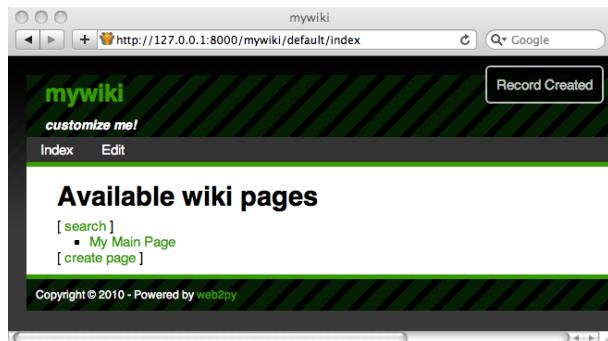
Aquí está el código para la vista "default/index.html":

```

1 {{extend 'layout.html'}}
2 <h1>Available wiki pages</h1>
3 [ {{=A('search', _href=URL('search'))}} ]<br />
4 <ul>{{for page in pages:}}
5     {{=LI(A(page.title, _href=URL('show', args=page.id)))}}
6 {{/for}}
7 [ {{=A('create page', _href=URL('create'))}} ]

```

Se genera la siguiente página:



Aquí está el código para el "default/show.html" view:

```

1 {{extend 'layout.html'}}
2 <h1>{{=page.title}}</h1>
3 [ {{=A('edit', _href=URL('edit', args=request.args))}}
4 | {{=A('documents', _href=URL('documents', args=request.args))}} ]<br />
5 {{=MARKMIN(page.body)}}
6 <h2>Comments</h2>
7 {{for comment in comments:}}
8   <p>{{=db.auth_user[comment.created_by].first_name}} on {{=comment.created_on}}
9     says <i>{{=comment.body}}</i></p>
10 {{/pass}}
11 <h2>Post a comment</h2>
12 {{=form}}

```

Si desea utilizar la sintaxis de Markdown en lugar de la sintaxis markmin, de `gluon.contrib.markdown.WIKI` importación y usarlo en vez de la ayuda `MARKMIN`. Alternativamente, usted puede optar por aceptar HTML prima en lugar de la sintaxis markmin. En este caso, usted tendría que reemplazar:

```

1 {{=MARKMIN(page.body)}}

```

con:

```

1 {{=XML(page.body)}}

```

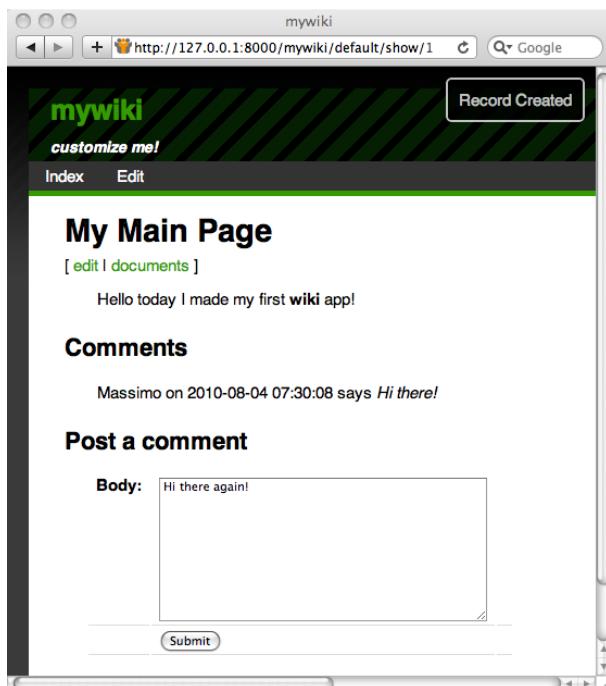
(de modo que el XML no se escapó, como por web2py comportamiento por defecto).

Esto se puede hacer mejor con:

```
1 {{=XML(page.body, sanitize=True)}}
```

Con el establecimiento de `sanitize=True`, decirle web2py escapar inseguros etiquetas XML como "`<script>`", y así evitar las vulnerabilidades XSS.

Ahora bien, si, desde la página de índice, haga clic sobre un título de la página, se puede ver la página que ha creado:



Aquí está el código para la vista "default/edit.html":

```
1 {{extend 'layout.html'}}
2 <h1>Edit wiki page</h1>
3 [ {{=A('show', _href=URL('show', args=request.args))}} ]<br />
4 {{=form}}
```

Genera una página que se ve casi idéntico al crear la página.

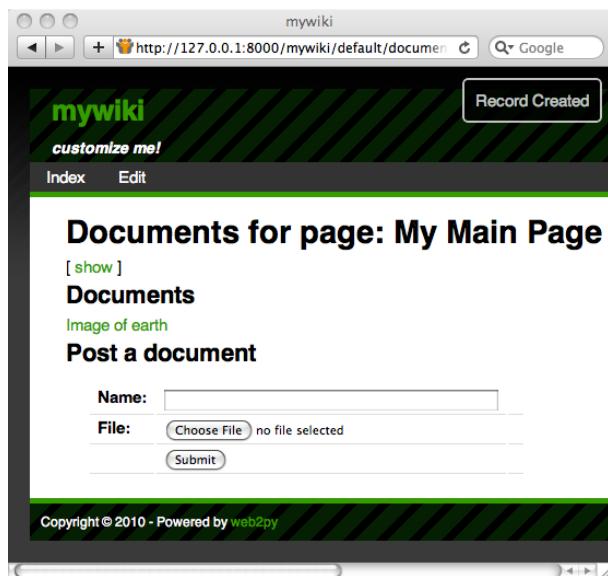
Aquí está el código para la vista "default/documents.html":

```

1 {{extend 'layout.html'}}
2 <h1>Documents for page: {{=page.title}}</h1>
3 [ {{=A('show', _href=URL('show', args=request.args))}} ]<br />
4 <h2>Documents</h2>
5 {{for document in documents:}}
6     {{=A(document.name, _href=URL('download', args=document.file))}}
7     <br />
8 {{/for}}
9 <h2>Post a document</h2>
10 {{=form}}

```

Si, desde la página "show", hacer clic en los documentos, ahora puede administrar los documentos adjuntos a la página.



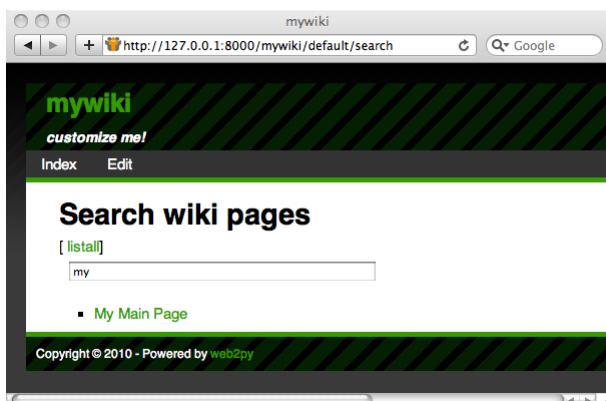
Finalmente aquí está el código de la vista "default/search.html":

```

1 {{extend 'layout.html'}}
2 <h1>Search wiki pages</h1>
3 [ {{=A('listall', _href=URL('index'))}} ]<br />
4 {{=form}}<br />{{=target_div}}

```

que genera el formulario siguiente Ajax de búsqueda:



También puede tratar de llamar a la acción de devolución de llamada directamente visitando, por ejemplo, el siguiente URL:

```
1 http://127.0.0.1:8000/mywiki/default/search?keyword=wiki
```

Si nos fijamos en la página de origen ve el HTML devuelto por la devolución de llamada:

```
1 <ul><li><a href="/mywiki/default/show/4">I made a Wiki</a></li></ul>
```

Creación de un feed RSS de las páginas almacenadas utilizando web2py es fácil porque incluye web2py gluon.contrib.rss2. Sólo añadir las siguientes medidas para el controlador por defecto:

```
1 def news():
2     "generates rss feed from the wiki pages"
3     pages = db().select(db.page.ALL, orderby=db.page.title)
4     return dict(
5         title = 'mywiki rss feed',
6         link = 'http://127.0.0.1:8000/mywiki/default/index',
7         description = 'mywiki news',
8         created_on = request.now,
9         items = [
10             dict(title = row.title,
11                  link = URL('show', args=row.id),
12                  description = MARKMIN(row.body).xml()),
```

```

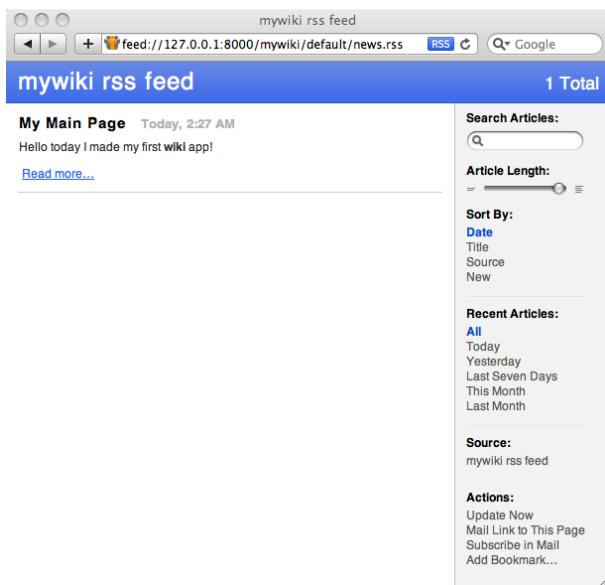
13     created_on = row.created_on
14 ) for row in pages])

```

y cuando usted visita la página de

```
1 http://127.0.0.1:8000/mywiki/default/news.rss
```

usted ve el alimento (la salida exacta depende del lector de feeds). Observe que el dict se convierte automáticamente en RSS, gracias a la ampliación rss. En la URL.



web2py también incluye feedparser para leer feeds de terceros.

Por último, vamos a añadir un controlador de XML-RPC que permite la búsqueda de la programación wiki:

```

1 service=Service(globals())
2
3 @service.xmlrpc
4 def find_by(keyword):
5     "finds pages that contain keyword for XML-RPC"

```

```

6     return db(db.page.title.lower().like('%' + keyword + '%'))\
7         .select().as_list()
8
9 def call():
10    "exposes all registered services, including XML-RPC"
11    return service()

```

Aquí, la acción controlador simplemente publica (a través de XML-RPC), las funciones especificadas en la lista. En este caso, `find_by`. `find_by` no es una acción (ya que toma un argumento). Se consulta la base de datos con `.select()` y, a continuación extractos de los registros como una lista con retornos `.response` y la lista.

He aquí un ejemplo de cómo acceder al controlador de XML-RPC desde un programa externo Python.

```

1 >>> import xmlrpclib
2 >>> server = xmlrpclib.ServerProxy(
3     'http://127.0.0.1:8000/mywiki/default/call/xmlrpc')
4 >>> for item in server.find_by('wiki'):
5     print item.created_on, item.title

```

El controlador se puede acceder desde muchos otros lenguajes de programación que entienden XML-RPC, incluyendo C, C++, C# y Java.

3.11 Más en admin

La interfaz de administración proporciona una funcionalidad adicional que vamos a reseñar brevemente aquí.

3.11.1 site

Esta página muestra todas las aplicaciones instaladas. Hay dos formas en la parte inferior.

El primero de ellos permite la creación de una nueva solicitud, especificando su nombre.

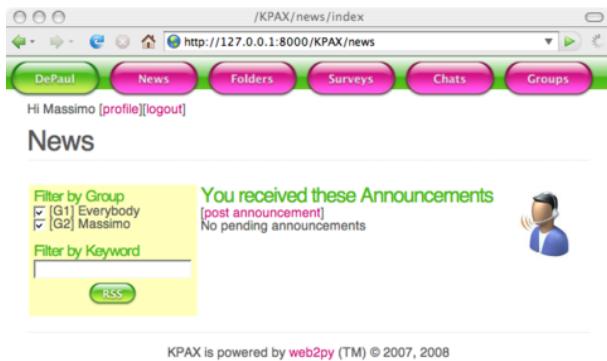
La segunda forma permite cargar una aplicación existente de un archivo local o una URL remota. Al cargar una aplicación, es necesario especificar un nombre para él. Esto puede ser su nombre original, pero no necesita ser. Esto permite instalar múltiples copias de la misma aplicación. Puedes intentar, por ejemplo, para cargar el contenido KPAX sistema de gestión de:

```
1 http://web2py.com/appliances/default/download/app.source.221663266939.tar
```

Subido aplicaciones pueden ser archivos .tar (convención de edad) y los archivos de .w2p (nuevo convenio). Estos últimos son archivos tar comprimidos con gzip. Se pueden comprimir manualmente con tar zxvf [filename] aunque esto nunca es necesario.

Una vez subida exitosa, se muestra web2py la Suma de control MD5 del fichero subido. Usted puede utilizarlo para verificar que el archivo no ha sido dañado durante la subida. El nombre KPAX aparecerá en la lista de aplicaciones instaladas.

Haga clic en el nombre KPAX de administración para su creación y funcionamiento.



Los expedientes de solicitud se almacenan como archivos W2P (tar comprimido con gzip), pero no están destinadas a untar de alquitrán o de forma manual; web2py lo hace por usted.

A cada solicitud de la página *site* le permite:

- Desinstalar la aplicación.
- Saltar a la página de *about* (leer abajo).
- Saltar a la página de *edit* (leer abajo).
- Saltar a la página de *errors* (leer abajo).
- Limpie los archivos temporales (períodos de sesiones, los errores y los archivos cache.disk).
- Pack todos. Esto devuelve un archivo tar que contiene una copia completa de la solicitud. Le sugerimos que limpiar los archivos temporales antes de empacar una aplicación.
- Compile la aplicación. Si no hay errores, esta opción bytecode a compilar todos los modelos, controladores y puntos de vista. Debido a que las opiniones pueden extenderse e incluir otros puntos de vista en un árbol, antes de la compilación bytecode, la vista de árbol por cada controlador

se derrumbó en un solo archivo. El efecto neto es que una aplicación de código de bytes compilado es más rápido, porque no hay más análisis de plantillas o sustituciones de cadenas que ocurren en tiempo de ejecución.

- Pack compilado. Esta opción sólo está presentes en bytecode compilado de aplicaciones. Permite embalaje a la solicitud sin el código fuente para ser distribuidos como código cerrado. Tenga en cuenta que Python (como cualquier otro lenguaje de programación) técnicamente se puede descompilar, por lo que la compilación no ofrece una protección completa del código fuente. Sin embargo, descompilación puede ser difícil y puede ser ilegal.
- Retire compilado. Simplemente elimina el bytecode compilado modelos, vistas y controladores de la aplicación. Si la solicitud fue empaquetado con el código fuente o diseñados a nivel local, no hay daño en la eliminación de los archivos compilados bytecode, y la aplicación seguirá trabajando. Si la aplicación se instaló formar un archivo comprimido compilado, entonces esto no es seguro, porque no hay código fuente para revertir, y la solicitud ya no hay trabajo.

Todas las funcionalidades disponibles en la página de administración del sitio web2py también se puede acceder mediante programación a través de la API definida en el módulo de gluon/admin.py. Basta con abrir un shell de python e importar este módulo.

3.11.2 *about*

La ficha *about* permite editar la descripción de la solicitud y su licencia. Estos son escritos, respectivamente, en el ACERCA DE LICENCIA y los archivos en la carpeta de la aplicación.

The screenshot shows a web browser window with the title 'about welcome'. The URL in the address bar is 'http://127.0.0.1:8000/admin/default/about/welcome'. The page has a dark blue header with the text '[web2py™]admin' and several menu items: site, edit, about, errors (which is highlighted in red), versioning, logout, and help.

About application "welcome"

About welcome
[edit]

Write something about this app. Developed with web2py.

License for welcome
[edit]

This is a sample license. You can write here anything you want as long as you do not violate web2py copyright, trademark and license.

Powered by web2py (TM) created by Massimo Di Pierro © 2007

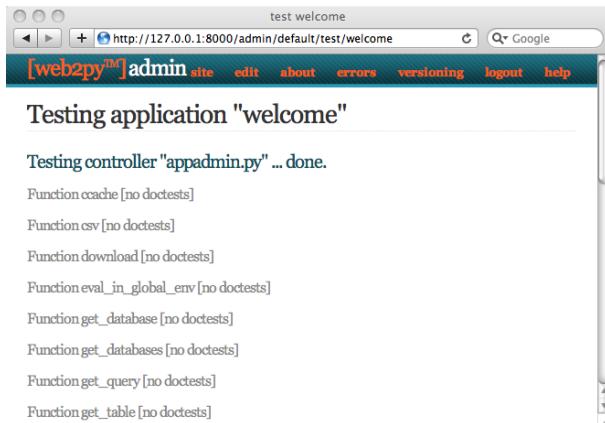
Puede utilizar la sintaxis MARKMIN o gluon.contrib.markdown.WIKI de estos archivos como se describe en ref. (29).

3.11.3 edit

Ha utilizado la página *edit* ya en este capítulo. Aquí queremos destacar un poco más las funcionalidades de la página *edit*.

- Si hace clic en cualquier nombre de archivo, se puede ver el contenido del archivo con resaltado de sintaxis.
- Si hace clic en editar, puede editar el archivo a través de una interfaz web.
- Si haces clic en eliminar, puede eliminar el archivo (permanentemente).
- Si hace clic en la prueba, web2py hará pruebas. Las pruebas son escritas por el programador usando Python doctests, y cada función debe tener sus propias pruebas.
- Usted puede agregar archivos de idiomas, la aplicación exploración para descubrir todas las cadenas, y edite las traducciones de cadena mediante la interfaz Web.
- Si los archivos estáticos se organizan en carpetas y subcarpetas, la jerarquía de carpetas se puede activar haciendo clic en un nombre de carpeta.

La imagen siguiente muestra la salida de la página de prueba para la aplicación de bienvenida.



The screenshot shows a web browser window with the title "test welcome". The address bar displays "http://127.0.0.1:8000/admin/default/test/welcome". The page header includes the [web2py™] logo and navigation links for admin, site, edit, about, errors, versioning, logout, and help. The main content area is titled "Testing application 'welcome'" and contains the text "Testing controller 'appadmin.py' ... done." followed by a list of function names with their status: Function cache [no doctests], Function csv [no doctests], Function download [no doctests], Function eval_in_global_env [no doctests], Function get_database [no doctests], Function get_databases [no doctests], Function get_query [no doctests], and Function get_table [no doctests].

```
test welcome
http://127.0.0.1:8000/admin/default/test/welcome
[web2py™]admin site edit about errors versioning logout help

Testing application "welcome"

Testing controller "appadmin.py" ... done.

Function cache [no doctests]
Function csv [no doctests]
Function download [no doctests]
Function eval_in_global_env [no doctests]
Function get_database [no doctests]
Function get_databases [no doctests]
Function get_query [no doctests]
Function get_table [no doctests]
```

La siguiente imagen muestra la ficha idiomas para la aplicación de bienvenida.

Languages
translation strings for the application

- [update all languages]
 - es-es.py [edit | delete]
 - fr-ca.py [edit | delete]
 - fr-fr.py [edit | delete]
 - hi-hi.py [edit | delete]
 - hu-hu.py [edit | delete]
 - hu.py [edit | delete]
 - it-it.py [edit | delete]
 - it.py [edit | delete]
 - pl-pl.py [edit | delete]
 - pl.py [edit | delete]
 - pt-br.py [edit | delete]
 - pt-pt.py [edit | delete]
 - pt.py [edit | delete]
 - ru-ru.py [edit | delete]
 - zh-tw.py [edit | delete]
 - create file with filename: submit

La imagen de abajo muestra cómo editar un archivo de idioma, en este caso, el "it" (italiano) de idiomas para la aplicación de bienvenida.

Editing Language file "welcome/languages/it.py"

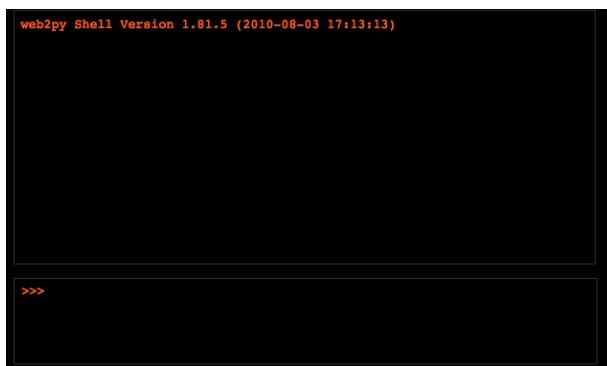
Original/Translation
 "update" is an optional expression like "field1=newvalue". You cannot update or delete the results of a JOIN
 "update" è un'espressione opzionale come "campo1='nuovo valore'". Non si può fare "update" o "delete" dei risultati di un JOIN

%s rows deleted
 %s right ("record") cancellate

concha

Si hace clic en el enlace "shell" en la ficha controladores en *edit*, web2py se abrirá un shell basado en la web de Python y ejecutará los modelos para la

aplicación actual. Esto le permite hablar de forma interactiva a su aplicación.



crontab

Igual que en la ficha controladores en *edit* existe una relación "crontab". Al hacer clic en este enlace podrás editar el archivo crontab web2py. Esto sigue la misma sintaxis que el crontab de Unix, pero no se basan en Unix. De hecho, sólo requiere de web2py y funciona en Windows también. Le permite registrar las acciones que deben ejecutarse en segundo plano en la fecha prevista veces. Para más información sobre esto, consulte el capítulo siguiente.

3.11.4 errors

Cuando la programación web2py , inevitablemente se cometen errores e introducir errores. web2py ayuda de dos maneras: 1) que le permite crear pruebas para cada función que se puede ejecutar en el navegador desde la página de *edit*, y 2) cuando un error se manifiesta, los billetes se expiden para el visitante y el error se registra.

A propósito introducir un error en la aplicación de imágenes como se muestra a continuación:

```
1 def index():
```

```

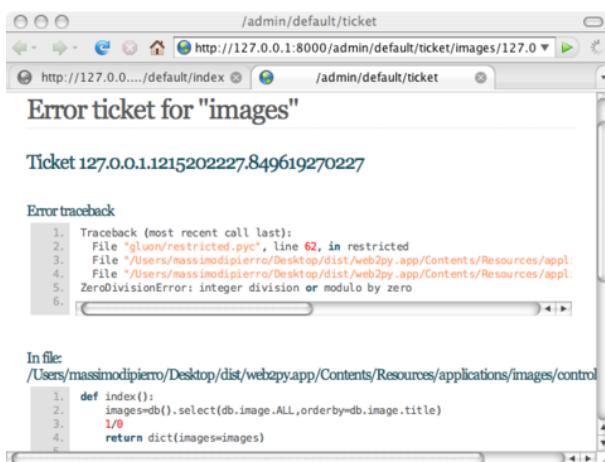
2     images = db().select(db.image.ALL,orderby=db.image.title)
3
4     return dict(images=images)

```

Al acceder a la acción de índice, las entradas siguientes:



Sólo el administrador puede acceder a las entradas:



El billete se muestra el rastreo, y el contenido del archivo que causó el problema. Si el error se produce en un punto de vista, muestra la vista web2py convierte de HTML en código Python. Esto permite identificar fácilmente la estructura lógica del archivo.

Tenga en cuenta que en todas partes muestra **admin** sintaxis resaltado de código (por ejemplo, en los informes de errores, palabras clave web2py se

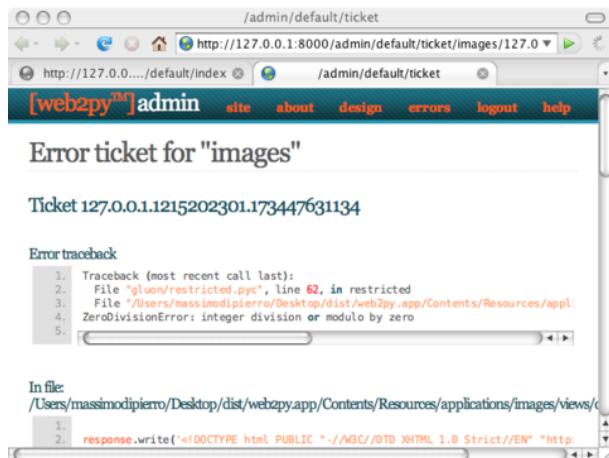
muestran en naranja). Si hace clic en una palabra clave web2py, se le redirige a una página de documentación sobre la palabra clave.

Si usted fija el / o error 1 en la acción e introducir un índice en la vista de lista:

```

1 {{extend 'layout.html'}}
2
3 <h1>Current Images</h1>
4 <ul>
5 {{for image in images:}}
6 {{1/0}}
7 {{=LI(A(image.title, _href=URL("show", args=image.id)))}}
8 {{/pass}}
9 </ul>
```

que las entradas siguientes:



Tenga en cuenta que web2py ha convertido a la vista desde un archivo HTML en Python, y por lo tanto, el error se describe en el pasaje se refiere al código generado Python y NO en el archivo de vista original. que las entradas siguientes:

```

43.         response.write("<class='inactive'>")
44.         pass
45.     response.write('>',escape=False)
46.     response.write('<name>')
47.     response.write('</a></li>\n    ',escape=False)
48.     pass
49.     response.write('\n      </ul>\n    </div>\n  <div id="main_inner" clu
50.     pass
51.     response.write('\n      </div>\n    <div id="main">\n      <div id="main_inner" clu
52.     if response.flash:
53.         response.write('\n          <div id="flash">',escape=False)
54.         response.write(response.flash)
55.         response.write('</div>\n      ',escape=False)
56.         pass
57.     response.write('\n          \n          <h1>Current Images</h1>\n          <ul>\n            ')
58.     for image in images:
59.         response.write('
60.           1/>
61.           <li>
62.             <a href="URL(r=request,f="show",args=[image.id])">
63.               <img alt="image.title" />
64.             </a>
65.           </li>
66.         response.write('
67.           </ul>
68.         </div>
69.       </div>
70.     </div>
71.   </div>
72. 
```

Esto puede parecer confuso al principio, pero en la práctica hace que la depuración más fácil, porque la sangría Python pone de relieve la estructura lógica del código que incrustados en los puntos de vista.

El código se muestra en la parte inferior de la misma página.

Todos los boletos se enumeran en la página admin en *errors* para cada aplicación:

Delete Ticket	Date and Time
<input type="checkbox"/> 127.0.0.1.1215202227.849619270227	2008-07-04 15:10:27
<input type="checkbox"/> 127.0.0.1.1215202301.17344763134	2008-07-04 15:11:41

Powered by **webzpy**™ created by Massimo Di Pierro © 2007, 2008

3.11.5 Mercurial

Si está ejecutando desde el código fuente y tiene el control de versiones Mercurial bibliotecas instaladas:

```
1 easy_install mercurial
```

a continuación, la interfaz de administración muestra un elemento más del menú botón llamado "mercurial". It automatically creates a local Mercurial repository for the application. Pressing the "commit" en la página se comprometen a la aplicación actual.

Esta característica es experimental y se mejorará en el futuro.

3.12 Más en appadmin

appadmin no está destinada a ser expuesta al público. Está diseñado para ayudar a proporcionar un fácil acceso a la base de datos. Se compone de dos archivos: un controlador de "appadmin.py" and a view "appadmin.html" que son utilizados por todas las acciones en el controlador.

El controlador **appadmin** es relativamente pequeño y fácil de leer, sino que constituye un ejemplo en el diseño de una interfaz de base de datos.

appadmin muestra las bases de datos están disponibles y las tablas que existen en cada base de datos. Usted puede insertar registros y la lista de todos los registros de cada tabla por separado. salida pagina **appadmin** 100 registros a la vez.

Una vez que un conjunto de registros está seleccionada, el encabezado de las páginas de los cambios, lo que le permite actualizar o eliminar los registros seleccionados.

Para actualizar los registros, entrar en una misión SQL en el campo de cadena de consulta:

```
1 title = 'test'
```

donde los valores de cadena debe ir entre comillas simples. campos múltiples pueden ser separados por comas.

Para borrar un registro, haga clic en la casilla correspondiente y confirmar que está seguro.

appadmin también puede realizar si el filtro se une a SQL contiene una condición SQL que involucra a dos o más tablas. Por ejemplo, pruebe:

```
1 db.image.id == db.comment.image_id
```

web2py pasa esto a lo largo de la DAL, y entiende que la consulta se unirán dos mesas, por lo que ambos cuadros se seleccionan con una operación INNER JOIN. Aquí está la salida:

Rows in table

Query:

Update:

Delete:

The "query" is a condition like "db.table1.field1=='value'". Something like "db.table1.field1==db.table2.field2" results in a SQL JOIN. Use (...) & (...) for AND, (...) | (...) for OR, and ~ (...) for NOT to build more complex queries. "update" is an optional expression like "field1='newvalue'". You cannot update or delete the results of a JOIN

1 selected

comment_id	comment.image_id	comment.author	comment.email	comment.body	image_id	image.title	im...
1	1	Me	you@example.com	picture of th...	1	image of earth	file

Si hace clic en el número de un campo id, se obtiene una página de edición del registro con el ID correspondiente.

Si hace clic en el número de un campo de referencia, se obtiene una página de edición del registro referenciado.

No se puede actualizar o borrar registros seleccionados por una combinación porque implican registros de varias tablas y esto sería ambigua.

4

El Núcleo

4.1 Opciones de línea de comandos

Es posible saltarse la interfaz gráfica de usuario y comenzar a web2py directamente desde la línea de comandos escribiendo algo así como:

```
1 python web2py.py -a 'your password' -i 127.0.0.1 -p 8000
```

Cuando se inicia web2py, crea un archivo llamado "parameters_8000.py" where it stores the hashed password. If you use "<ask>" como la contraseña, web2py le pide por ello.

Para seguridad adicional, usted puede comenzar a web2py con:

```
1 python web2py.py -a '<recycle>' -i 127.0.0.1 -p 8000
```

En este caso web2py vuelve a utilizar la contraseña con algoritmo hash almacenado previamente. Si no se proporciona la contraseña, o si el archivo es eliminado "parameters_8000.py", la interfaz de administración basada en web está deshabilitada.

En algunos sistemas Unix / Linux, si la contraseña es

```
1 <pam_user:some_user>
```

web2py PAM utiliza la contraseña de la cuenta del sistema operativo de la `some_user` para autenticar al administrador, a no ser bloqueado por la configuración de PAM.

Web2py normalmente se ejecuta con CPython (la implementación C de la intérprete de Python creado por Guido van Rossum), pero también se puede ejecutar con Jython (la aplicación Java del intérprete). Esta última posibilidad permite el uso de web2py en el contexto de una infraestructura de J2EE. Para utilizar Jython, basta con sustituir "python web2py.py..." with "jython web2py.py". Detalles sobre la instalación de Jython, zxJDBC los módulos necesarios para acceder a las bases de datos se puede encontrar en el capítulo 12.

La secuencia de comandos "web2py.py" puede tomar muchos argumentos de línea de comandos que especifica el número máximo de subprocessos, lo que permite de SSL, etc Para obtener una lista completa de tipo:

```
1 >>> python web2py.py -h
2 Usage: python web2py.py
3
4 web2py Web Framework startup script. ATTENTION: unless a password
5 is specified (-a 'passwd'), web2py will attempt to run a GUI.
6 In this case command line options are ignored.
7
8 Options:
9   --version           show program's version number and exit
10  -h, --help            show this help message and exit
11  -i IP, --ip=IP        ip address of the server (127.0.0.1)
12  -p PORT, --port=PORT  port of server (8000)
13  -a PASSWORD, --password=PASSWORD
14                           password to be used for administration
15                           use -a "<recycle>" to reuse the last
16                           password
17  -u UPGRADE, --upgrade=UPGRADE
18                           -u yes: upgrade applications and exit
19  -c SSL_CERTIFICATE, --ssl_certificate=SSL_CERTIFICATE
20                           file that contains ssl certificate
21  -k SSL_PRIVATE_KEY, --ssl_private_key=SSL_PRIVATE_KEY
22                           file that contains ssl private key
```

```

23 -d PID_FILENAME, --pid_filename=PID_FILENAME
24                                     file to store the pid of the server
25 -l LOG_FILENAME, --log_filename=LOG_FILENAME
26                                     file to log connections
27 -n NUMTHREADS, --numthreads=NUMTHREADS
28                                     number of threads
29 -s SERVER_NAME, --server_name=SERVER_NAME
30                                     server name for the web server
31 -q REQUEST_QUEUE_SIZE, --request_queue_size=REQUEST_QUEUE_SIZE
32                                     max number of queued requests when server unavailable
33 -o TIMEOUT, --timeout=TIMEOUT
34                                     timeout for individual request (10 seconds)
35 -z SHUTDOWN_TIMEOUT, --shutdown_timeout=SHUTDOWN_TIMEOUT
36                                     timeout on shutdown of server (5 seconds)
37 -f FOLDER, --folder=FOLDER
38                                     folder from which to run web2py
39 -v, --verbose           increase --test verbosity
40 -Q, --quiet            disable all output
41 -D DEBUGLEVEL, --debug=DEBUGLEVEL
42                                     set debug output level (0-100, 0 means all,
43                                     100 means none; default is 30)
44 -S APPNAME, --shell=APPNAME
45                                     run web2py in interactive shell or IPython
46                                     (if installed) with specified appname
47 -P, --plain             only use plain python shell; should be used
48                                     with --shell option
49 -M, --import_models    auto import model files; default is False;
50                                     should be used with --shell option
51 -R PYTHON_FILE, --run=PYTHON_FILE
52                                     run PYTHON_FILE in web2py environment;
53                                     should be used with --shell option
54 -T TEST_PATH, --test=TEST_PATH
55                                     run doctests in web2py environment;
56                                     TEST_PATH like a/c/f (c,f optional)
57 -W WINSERVICE, --winservice=WINSERVICE
58                                     -W install|start|stop as Windows service
59 -C, --cron               trigger a cron run manually; usually invoked
60                                     from a system crontab
61 -N, --no-cron            do not start cron automatically
62 -L CONFIG, --config=CONFIG
63                                     config file
64 -F PROFILER_FILENAME, --profiler=PROFILER_FILENAME
65                                     profiler filename
66 -t, --taskbar            use web2py gui and run in taskbar
67                                     (system tray)
68 --nogui                 text-only, no GUI
69 -A ARGS, --args=ARGS    should be followed by a list of arguments to be passed
70                                     to script, to be used with -S, -A must be the last
71                                     option

```

```

72 --interfaces=INTERFACES
73           allows multiple interfaces to be served

```

opciones minúscula se utilizan para configurar el servidor web. La opción **-L** cuenta web2py para leer las opciones de configuración de un archivo, se instala **-W** web2py como un servicio de Windows, mientras que **-S**, **-P** y **-M** opciones de iniciar un intérprete de comandos interactivo de Python. La opción **-T** encuentra y se ejecuta doctests controlador en un entorno de ejecución web2py. Por ejemplo, el siguiente ejemplo se ejecuta doctests de todos los controladores en la aplicación "welcome":

```

1 python web2py.py -vT welcome

```

si ejecuta web2py como servicio de Windows, **-W**, no es conveniente pasar de la configuración mediante argumentos de línea de comandos. Por esta razón, en la carpeta web2py hay un ejemplo de archivo de configuración "options_std.py" para el servidor web interno:

```

1 import socket, os
2 ip = '127.0.0.1'
3 port = 8000
4 password = '<recycle>' ### <recycle> means use the previous password
5 pid_filename = 'httpserver.pid'
6 log_filename = 'httpserver.log'
7 ssl_certificate = " ### path to certificate file
8 ssl_private_key = " ### path to private key file
9 numthreads = 10
10 server_name = socket.gethostname()
11 request_queue_size = 5
12 timeout = 10
13 shutdown_timeout = 5
14 folder = os.getcwd()

```

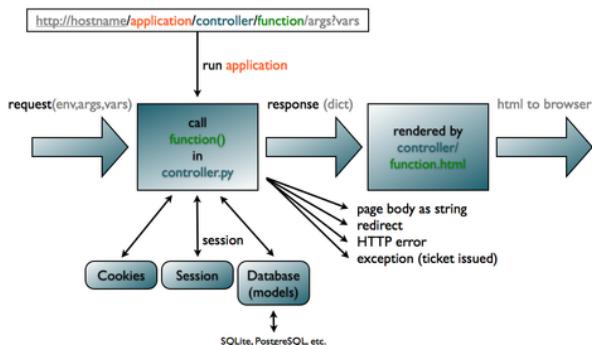
Este archivo contiene los valores predeterminados web2py. Si modifica este archivo, tiene que importarlo explícitamente con la opción de línea de comandos **-L**. Sólo funciona si ejecuta web2py como servicio de Windows.

4.2 Despacho

mapas web2py una dirección URL de la forma:

```
1 http://127.0.0.1:8000/a/c/f.html
```

a la función en `f()` "c.py" in application "a" controlador. Si `f` no está presente, por defecto web2py a la función de controlador de `index`. Si `c` no está presente, por defecto web2py al controlador "default.py", a y si no está presente, por defecto web2py a la aplicación `init`. Si no hay ninguna aplicación `init`, web2py intenta ejecutar la aplicación `welcome`. Esto se muestra esquemáticamente en la siguiente imagen:



De forma predeterminada, cualquier nueva solicitud también crea una nueva sesión. Además, una cookie de sesión se devuelve al navegador del cliente para realizar un seguimiento de la sesión.

El `.html` extensión es opcional; `.html` se asume por defecto. La extensión determina la extensión de la vista que hace que la salida de la función de controlador de `f()`. Permite el mismo contenido para ser servido en múltiples formatos (`html`, `xml`, `json`, `rss`, etc.)

Funciones que toman argumentos o comenzar con un guión bajo doble no se hubiere ofrecido públicamente y sólo se puede llamar por otras funciones.

Hay una excepción para las URL de la forma:

```
1 http://127.0.0.1:8000/a/static/filename
```

No hay controlador llamado "static". web2py interprets this as a request for the file called "filename" in the subfolder "static" of the application "a".

Cuando se descargan los archivos estáticos, web2py no crea una sesión, ni tampoco emitir una "cookie" o ejecutar los modelos. web2py siempre arroja archivos estáticos en trozos de 1 MB, y envía PARCIAL DE CONTENIDO cuando el cliente envía una solicitud de GAMA para un subconjunto del archivo. web2py también es compatible con el protocolo IF_MODIFIED_SINCE, y que no envíe el archivo si no está ya almacenada en la caché del navegador y si el archivo no ha cambiado desde esa versión.

mapas web2py GET / POST peticiones de la forma:

```
1 http://127.0.0.1:8000/a/c/f.html/x/y/z?p=1&q=2
```

para funcionar en f "c.py" a controlador en la aplicación, y almacena los parámetros de URL en la variable de la siguiente manera request:

```
1 request.args = ['x', 'y', 'z']
```

y:

```
1 request.vars = {'p':1, 'q':2}
```

y:

```
1 request.application = 'a'
2 request.controller = 'c'
3 request.function = 'f'
```

En el ejemplo anterior, tanto `request.args[i]` y `request.args(i)` se puede utilizar para recuperar el elemento " " de la `request.args`, pero mientras el primero produce una excepción si la lista no tiene ese tipo de índice, éste regresa Ninguno en este caso.

```
1 request.url
```

almacena la dirección URL completa de la solicitud actual (sin incluir las variables GET).

```
1 request.ajax
```

Falso por defecto pero es True si web2py determina que la acción fue convocada por una petición Ajax.

Si la petición HTTP es un GET, a continuación, se establece en `request.env.request_method` "GET", y si es un POST, `request.env.request_method` se establece en "POST". Las variables de consulta de URL se almacenan en el diccionario `request.vars` de almacenamiento, sino que también se almacenan en `request.get_vars` (a raíz de una petición GET) o `request.post_vars` (a raíz de una petición POST). Tiendas web2py WSGI y variables de entorno web2py en `request.env`, por ejemplo:

```
1 request.env.path_info = 'a/c/f'
```

y encabezados HTTP en las variables de entorno, por ejemplo:

```
1 request.env.http_host = '127.0.0.1:8000'
```

Observe que web2py valida todas las URL para evitar ataques de directorio transversal.

URL sólo pueden contener caracteres alfanuméricos, guiones, barras, el `args` puede contener puntos no consecutivos. Los espacios se sustituirán por guiones bajos antes de la validación. Si la sintaxis de URL no es válida, devuelve web2py un error HTTP 400 message (47; 48).

Si la dirección corresponde a una solicitud de un archivo estático, web2py simplemente lee y devuelve (corrientes) el archivo solicitado.

Si la URL no solicita un fichero estático web2py procesa la solicitud en el siguiente orden:

- Analiza las cookies.
- Crea un entorno en el que para ejecutar la función.
- Inicializa `request`, `response`, `cache`.
- Abre el `session` existente o crea uno nuevo.
- Ejecuta los modelos pertenecientes a la aplicación solicitada.
- Ejecuta la función controladora acción solicitada.
- Si la función devuelve un diccionario ejecuta, la vista asociada.
- En caso de éxito, obliga a todas las transacciones abiertas.
- Guarda la sesión.
- Devuelve una respuesta HTTP.

Observe que el controlador y la vista se ejecutan en los diferentes ejemplares de un mismo entorno, por lo tanto, la vista no ve el controlador, pero ve los modelos y las variables que considera devuelto por la función acción del controlador.

Si una excepción (que no sean HTTP) se eleva, web2py hace lo siguiente:

- Almacena el rastreo, en un error de archivo y le asigna un número de ticket a la misma.
- Revierte todas las transacciones abiertas.
- Devuelve una página de error de presentación de informes el número de billete.

Si la excepción es una excepción HTTP, esto se supone que es el comportamiento previsto (por ejemplo, redirigir una HTTP), y todas las transacciones

de bases de datos abiertas están comprometidos. El comportamiento después de que se especifica en la excepción HTTP sí mismo. La clase de excepción HTTP no es una norma excepción Python, sino que se define por web2py.

4.3 Bibliotecas

Las bibliotecas web2py están expuestos a las aplicaciones de usuario como objetos globales. Por ejemplo (`request`, `response`, `session`, `cache`), clases (ayudantes, validadores, DAL API), y funciones (`T` y `redirect`).

Estos objetos se definen en los archivos principales siguientes:

```

1 web2py.py
2 gluon/__init__.py    gluon/highlight.py    gluon/restricted.py  gluon/streamer.py
3 gluon/admin.py       gluon/html.py        gluon/rewrite.py    gluon/template.py
4 gluon/cache.py       gluon/http.py        gluon/rocket.py    gluon/storage.py
5 gluon/cfs.py         gluon/import_all.py  gluon/sanitizer.py gluon/tools.py
6 gluon/compileapp.py  gluon/languages.py   gluon/serializers.py gluon/utils.py
7 gluon/contenttype.py gluon/main.py       gluon/settings.py  gluon/validators.py
8 gluon/dal.py         gluon/myregex.py   gluon/shell.py    gluon/widget.py
9 gluon/decoder.py    gluon/newcron.py   gluon/sql.py      gluon/winservice.py
10 gluon/fileutils.py  gluon/portalocker.py gluon/sqlhtml.py  gluon/xmlrpc.py
11 gluon/globals.py    gluon/reserved_sql_keywords.py

```

El andamio tar comprimido con gzip aplicación que se suministran con web2py es

```

1 welcome.w2p

```

Se crea durante la instalación y sobrescribir de cualquier actualización.

La primera vez que inicie web2py, dos nuevas carpetas se crean: El depósito y aplicaciones. El archivo "welcome" app is zipped into a "welcome.w2p" para ser utilizados como andamios aplicación. La carpeta se utiliza como depósito para almacenamiento temporal de la instalación y desinstalación de aplicaciones.

web2py unidad de pruebas están en

```
1 gluon/tests/
```

Hay controladores para la conexión con los servidores web diferentes:

```
1 cghandler.py
2 gaehandler.py
3 fcgihandler.py
4 wsgihandler.py
5 modpythonhandler.py
6 gluon/contrib/gateways/__init__.py
7 gluon/contrib/gateways/fcgi.py
```

(Fcgi.py fue desarrollado por Allan Saddi)

Hay dos ficheros de ejemplo:

```
1 options_std.py
2 routes.example.py
```

El primero es un archivo de configuración opcional que se puede pasar a web2py.py con la opción de -L. El segundo es un ejemplo de un archivo de asignación de dirección URL. Se carga automáticamente cuando renombrado "routes.py".

Los archivos de

```
1 app.yaml
2 index.yaml
```

son archivos de configuración necesarios para la implementación en el motor de Google App. Probablemente no es necesario modificarlos, pero usted puede leer más sobre ellos en las páginas de Google Documentación.

También hay librerías adicionales, por lo general desarrollada por un tercero:

feedparser (28) por Mark Pilgrim para leer feeds RSS y Atom:

```
1 gluon/contrib/__init__.py
2 gluon/contrib/feedparser.py
```

markdown2 (29) por Trent Mick de marcas wiki:

```
1 gluon/contrib/markdown/__init__.py
2 gluon/contrib/markdown/markdown2.py
```

markmin marcado:

```
1 gluon/contrib/markmin.py
```

pysimplesoap es un ligero servidor de aplicación SOAP creado por Mariano Reingart:

```
1 gluon/contrib/pysimplesoap/
```

API Python **memcache** (30) por Martin Evan:

```
1 gluon/contrib/memcache/__init__.py
2 gluon/contrib/memcache/memcache.py
```

gql, un puerto de la DAL a la App Engine de Google:

```
1 gluon/contrib/gql.py
```

memdb, un puerto de la DAL en la parte superior de memcache:

```
1 gluon/contrib/memdb.py
```

gae_memcache es un API para usar memcache en la App Engine de Google:

```
1 gluon/contrib/gae_memcache.py
```

pyrtf (26) para la generación de formato de texto enriquecido (RTF) los documentos, desarrollado por Simon Cusack y revisado por Grant Edwards:

```
1 gluon/contrib/pyrtf
2 gluon/contrib/pyrtf/__init__.py
3 gluon/contrib/pyrtf/Constants.py
4 gluon/contrib/pyrtf/Elements.py
5 gluon/contrib/pyrtf/PropertySets.py
6 gluon/contrib/pyrtf/README
7 gluon/contrib/pyrtf/Renderer.py
8 gluon/contrib/pyrtf/Styles.py
```

PyRSS2Gen (27) desarrollado por Dalke Scientific Software, para generar feeds RSS:

```
1 gluon/contrib/rss2.py
```

simplejson (25) por Bob Ippolito, la biblioteca estándar para analizar y escribir objetos JSON:

```
1 gluon/contrib/simplejson/__init__.py
2 gluon/contrib/simplejson/decoder.py
3 gluon/contrib/simplejson/encoder.py
4 gluon/contrib/simplejson/jsonfilter.py
5 gluon/contrib/simplejson/scanner.py
```

AuthorizeNet (93) API proporciona a aceptar pagos con tarjeta de crédito a través de la red

```
1 gluon/contrib/AuthorizeNet.py
```

Authorize.net

PAM (72) autenticación API creada por Chris AtLee:

```
1 gluon/contrib/pam.py
```

Un clasificador Bayesiano para poblar la base de datos con datos ficticios para propósitos de prueba:

```
1 gluon/contrib/populate.py
```

Un archivo que permite la interacción con la barra de tareas de Windows, cuando web2py se ejecuta como un servicio:

```
1 gluon/contrib/taskbar_widget.py
```

login_methods Facultativo y login_forms que se utilizarán para la autenticación:

```
1 gluon/contrib/login_methods/__init__.py
2 gluon/contrib/login_methods/basic_auth.py
```

```

3 gluon/contrib/login_methods/cas_auth.py
4 gluon/contrib/login_methods/email_auth.py
5 gluon/contrib/login_methods/extended_login_form.py
6 gluon/contrib/login_methods/gae_google_account.py
7 gluon/contrib/login_methods/ldap_auth.py
8 gluon/contrib/login_methods/linkedin_account.py
9 gluon/contrib/login_methods/oauth20_account.py
10 gluon/contrib/login_methods/openid_auth.py
11 gluon/contrib/login_methods/pam_auth.py
12 gluon/contrib/login_methods/rpx_account.py

```

web2py también contiene una carpeta con scripts útiles:

```

1 scripts/setup-web2py-fedora.sh
2 scripts/setup-web2py-ubuntu.sh
3 scripts/cleancss.py
4 scripts/cleanhtml.py
5 scripts/contentparser.py
6 scripts/repair.py
7 scripts/sessions2trash.py
8 scripts/sync_languages.py
9 scripts/tickets2db.py
10 ...

```

Los dos primeros son particularmente útiles porque intentar una instalación completa y la configuración de un entorno de producción web2py desde cero. Este aspecto se analiza en el capítulo 12, pero son más o menos auto-dокументado.

Por último web2py incluye estos archivos necesarios para construir las distribuciones binarias.

```

1 Makefile
2 setup_exe.py
3 setup_app.py

```

Estos son los scripts de configuración para **py2exe** y **py2app**, respectivamente, y que sólo son necesarios para construir las distribuciones binarias de web2py. QUE NUNCA DEBE NEET para ejecutarlas.

En resumen, las bibliotecas web2py ofrecer las siguientes funcionalidades:

- Mapa de las URL en las llamadas a funciones.
- Mando pasar y devolver parámetros a través de HTTP.
- Realizar la validación de estos parámetros.
- Proteger las aplicaciones de las cuestiones de seguridad más actualizada.
- Manejar la persistencia de datos (base de datos, la sesión, caché, cookies).
- Realizar traducciones de cadenas para varios idiomas soportados.
- Generar HTML mediante programación (por ejemplo, de las tablas de base de datos).
- Generar SQL a través de una capa de abstracción de base de datos (DAL).
- Generar formato de texto enriquecido (RTF) de salida.
- Generar valores separados por comas (CSV) de salida de las tablas de base de datos.
- Generar Really Simple Syndication (RSS).
- Generar JavaScript Object Notation (JSON) las cuerdas de serialización para Ajax.
- Traducir wiki marcas (Markdown) a HTML.
- Expone XML-RPC servicios web.
- Cargar y descargar archivos de gran tamaño a través de streaming.

aplicaciones web2py contener archivos adicionales, en particular, bibliotecas JavaScript de terceros, como jQuery, calendario, y EditArea nicEdit. Sus autores son reconocidos en los propios archivos.

4.4 *Aplicaciones*

Las aplicaciones desarrolladas en web2py se componen de las siguientes partes:

- **models** describir una representación de los datos como las tablas de bases de datos y las relaciones entre tablas.

- **controllers** describir la lógica de la aplicación y flujo de trabajo.
- **views** describen cómo los datos deben ser presentados al usuario mediante HTML y JavaScript.
- **languages** describir cómo traducir las cadenas en la aplicación en varios idiomas soportados.
- **static files** no requieren tratamiento (por ejemplo, imágenes, hojas de estilo CSS, etc.)
- **ABOUT** y **README** documentos se explica por sí mismo.
- error de la tienda **errors** informes generados por la aplicación.
- almacenar información relacionada **sessions** a cada usuario en particular.
- **databases** bases de datos SQLite y almacenar información de la tabla adicional.
- almacenar en caché los elementos **cache** aplicación.
- **modules** son opcionales otros módulos de Python.
- archivos **private** se accede por los controladores, pero no directamente por el promotor.
- archivos **uploads** se accede por los modelos, pero no directamente por el promotor (por ejemplo, los archivos subidos por los usuarios de la aplicación).
- **tests** es un directorio para almacenar scripts de prueba y accesorios, y se burla.

Los modelos, vistas, controladores, los idiomas y los archivos estáticos son accesibles a través de la administración web [diseño] de la interfaz. ACERCA, README, y los errores son también accesibles por medio de la interfaz de administración por los elementos del menú correspondiente. Sesiones, caché, módulos y archivos privados son accesibles a las aplicaciones, pero no a través de la interfaz de administración.

Todo está perfectamente organizado en una estructura de directorios claro que se replica para cada aplicación web2py instalado, aunque el usuario nunca tiene que acceder al sistema de ficheros directamente:

```

1 __init__.py ABOUT LICENSE models views
2 controllers modules private tests cron
3 cache errors upload sessions static

```

"`__init__.py`" es un archivo vacío que se requiere a fin de que Python (y web2py) para importar los módulos en el directorio `modules`.

Observe que la aplicación **admin** proporciona simplemente una interfaz web para aplicaciones web2py del sistema de archivos del servidor. web2py aplicaciones también pueden ser creados y desarrollados a partir de la línea de comandos, usted no tiene que utilizar el navegador de interfaz de **admin**. Una nueva aplicación se pueden crear manualmente mediante la replicación de la estructura de directorios más arriba en, por ejemplo, "applications/newapp/" (o simplemente descomprimir el archivo en el directorio welcome.w2p aplicaciones informáticas). Los expedientes de solicitud también se pueden crear y modificar desde la línea de comandos sin tener que utilizar la interfaz web de **admin**.

4.5 API

Los modelos, controladores y puntos de vista se ejecutan en un entorno donde los objetos siguientes ya se han importado por nosotros:

Global Objects:

```

1 request, response, session, cache

```

Navigation:

```

1 redirect, HTTP

```

Internationalization:

```

1 T

```

Helpers:

```

1 XML, URL, BEAUTIFY
2
3 A, B, BEAUTIFY, BODY, BR, CENTER, CODE, DIV, EM, EMBED,
4 FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HEAD, HR, HTML,
5 I, IFRAME, IMG, INPUT, LABEL, LEGEND, LI, LINK, OL, UL,
6 MARKMIN, MENU, META, OBJECT, ON, OPTION, P, PRE, SCRIPT,
7 OPTGROUP, SELECT, SPAN, STYLE, TABLE, TAG, TD, TEXTAREA,
8 TH, THEAD, TBODY, TFOOT, TITLE, TR, TT, URL, XHTML,
9 xmlescape, embed64

```

Validators:

```

1 CLEANUP, CRYPT, IS_ALPHANUMERIC, IS_DATE_IN_RANGE, IS_DATE,
2 IS_DATETIME_IN_RANGE, IS_DATETIME, IS_DECIMAL_IN_RANGE,
3 IS_EMAIL, IS_EMPTY_OR, IS_EXPR, IS_FLOAT_IN_RANGE, IS_IMAGE,
4 IS_IN_DB, IS_IN_SET, IS_INT_IN_RANGE, IS_IPV4, IS_LENGTH,
5 IS_LIST_OF, IS_LOWER, IS_MATCH, IS_EQUAL_TO, IS_NOT_EMPTY,
6 IS_NOT_IN_DB, IS_NULL_OR, IS_SLUG, IS_STRONG, IS_TIME,
7 IS_UPLOAD_FILENAME, IS_UPPER, IS_URL

```

Database:

```

1 DAL, Field

```

Por compatibilidad con versiones SQLDB=DAL y SQLField=Field. Le animamos a usar la nueva sintaxis DAL y Field, en lugar de la sintaxis de edad.

Otros objetos y los módulos se definen en las bibliotecas, pero no se importan automáticamente ya que no se utilizan con tanta frecuencia.

El núcleo de las entidades del API en el entorno de ejecución web2py son request, response, session, cache, URL, HTTP, redirect y T, y se comentan a continuación.

Unos cuantos objetos y funciones, incluyendo **Auth**, **Crud** y **Service**, "gluon/tools.py" se definen en y deben ser importados en caso necesario:

```

1 from gluon.tools import Auth, Crud, Service

```

4.6 solicitud

El objeto `request` es una instancia de la clase web2py ubicua que se llama `gluon.storage.Storage`, que amplía la clase Python `dict`. Se trata básicamente de un diccionario, pero los valores de partida también se puede acceder como atributos:

```
1 request.vars
```

es el mismo:

```
1 request['vars']
```

A diferencia de un diccionario, si un atributo (o clave) no existe, no se producirá una excepción. En su lugar, devuelve `None`. `request` tiene los siguientes elementos y atributos, algunos de los cuales son también una instancia de la clase `Storage`:

- **`request.cookies`**: un objeto que contiene las cookies `Cookie.SimpleCookie()` pasado con la petición HTTP. Actúa como un diccionario de cookies. Cada cookie es un objeto bocado.
- **`request.env`**: `Storage` contiene un objeto de las variables de entorno se pasa al controlador, incluyendo variables de cabecera HTTP de la petición HTTP estándar y parámetros WSGI. Las variables de entorno son todos convertidos a minúsculas, y los puntos se convierten en relieve para facilitar su memorización.
- **`request.application`**: el nombre de la aplicación solicitada (analizada a partir de `request.env.path_info`).
- **`request.controller`**: el nombre del responsable del tratamiento requerido (analizada desde el `request.env.path_info`).
- **`request.function`**: el nombre de la función solicitada (analizado desde el `request.env.path_info`).
- **`request.extension`**: la extensión de la acción solicitada. Su valor predeterminado es "html". Si la función de controlador devuelve un diccionario y

no especifica un punto de vista, esto se utiliza para determinar la extensión de la vista de archivos que se traduzcan en el diccionario (analizada desde el `request.env.path_info`).

- **`request.folder`**: el directorio de la aplicación. Por ejemplo, si la solicitud es "welcome", `request.folder` se establece en el "/path/to/welcome" ruta absoluta. En sus programas, siempre debe utilizar esta variable y la función `os.path.join` para construir rutas de acceso a los archivos que necesita acceder. Aunque web2py siempre usa rutas absolutas, es una buena regla para cambiar nunca de forma explícita la carpeta de trabajo actual (sea lo que sea), pues esto no es una práctica thread-safe.
- **`request.now`**: `datetime.datetime` un objeto de almacenar el sello de tiempo de la solicitud actual.
- **`request.args`**: una lista de los componentes del paso siguiente URL el nombre de función del controlador; `request.env.path_info.split('/')[3:]` equivalente a
- **`request.vars`**: `gluon.storage.Storage` un objeto que contiene el HTTP GET y HTTP POST variables de la consulta.
- **`request.get_vars`**: `gluon.storage.Storage` un objeto que contenga solamente las variables de la consulta HTTP GET.
- **`request.post_vars`**: `gluon.storage.Storage` contiene un objeto únicamente las variables de consulta HTTP POST.
- **`request.client`**: La dirección IP del cliente que determine `request.env.remote_addr` o `request.env.http_x_forwarded_for` si está presente. Si bien esto es útil, no se debe confiar ya que el `http_x_forwarded_for` pueden ser falseadas.
- escuchar **`request.body`**: un archivo de sólo lectura que contiene el cuerpo de la petición HTTP. Esto es interpretado automáticamente para obtener la `request.post_vars` y luego rebobinado. Se puede leer con el `request.body.read()`.
- **`request.wsgi`** un gancho que permite llamar a aplicaciones de terceros WSGI de las acciones en el interior.

A modo de ejemplo, la siguiente llamada en un sistema típico:

¹ `http://127.0.0.1:8000/examples/default/status/x/y/z?p=1&q=2`

resultados en la siguiente tabla:

variable	value
request.application	examples
request.controller	default
request.function	index
request.extension	html
request.view	status
request.folder	applications/examples/
request.args	['x', 'y', 'z']
request.vars	<Storage {'p': 1, 'q': 2}>
request.get_vars	<Storage {'p': 1, 'q': 2}>
request.post_vars	<Storage {}>
request.wsgi	hook
request.env.content_length	0
request.env.content_type	text/xml;text/html;
request.env.http_accept	gzip, deflate
request.env.http_accept_encoding	en
request.env.http_accept_language	session_id_examples=127.0.0.1.119725
request.env.http_cookie	127.0.0.1:8000
request.env.http_host	10
request.env.http_max_forwards	http://web2py.com/
request.env.http_referer	Mozilla/5.0
request.env.http_user_agent	1.1 web2py.com
request.env.http_via	76.224.34.5
request.env.http_x_forwarded_for	web2py.com
request.env.http_x_forwarded_host	127.0.0.1
request.env.http_x_forwarded_server	/examples/simple_examples/status
request.env.path_info	remote_addr:127.0.0.1
request.env.query_string	GET
request.env.request_method	
request.env.script_name	127.0.0.1
request.env.server_name	8000
request.env.server_port	HTTP/1.1
request.env.server_protocol	/Users/mdipierro/web2py
request.env.web2py_path	Version 1.81.5
request.env.we2bpy_version	(optional, defined only if GAE detected)
request.env.web2py_runtime_gae	<open file, mode 'w' at >
request.env.wsgi_errors	
request.env.wsgi_input	
request.env.wsgi_multiprocess	False
request.env.wsgi_multithread	True
request.env.wsgi_run_once	False
request.env.wsgi_url_scheme	http
request.env.wsgi_version	10

¿Qué variables de entorno se definen realmente depende del servidor web. Aquí estamos asumiendo que el servidor incorporado Rocket WSGI. El con-

junto de variables no es muy diferente cuando se utiliza el servidor web Apache.

Las variables `request.env.http_*` se analizan desde el encabezado de solicitud HTTP.

Las variables `request.env.web2py_*`. Estos no se analizan desde el entorno de servidor web, pero son creados por web2py en caso de que sus aplicaciones necesitan saber acerca de la ubicación web2py y la versión, y si se está ejecutando en el motor de Google App (porque optimizaciones específicas que sean necesarias).

Observe también las variables `request.env.wsgi_*`. Son específicos para el adaptador de WSGI.

4.7 *respuesta*

`response` es otro ejemplo de la clase `Storage`. Contiene la siguiente información:

- **`response.body`:** `StringIO` un objeto en el que web2py escribe el cuerpo de la página de salida. NUNCA cambiar esta variable.
- **`response.cookies`:** similar a la `request.cookies`, pero mientras que el segundo contiene las cookies enviadas desde el cliente al servidor, el primero contiene las cookies enviadas por el servidor al cliente. La cookie de sesión se controla de forma automática.
- **`response.download(request, db)`:** un método de ejecución de la función de controlador que permite la descarga de los archivos subidos.
- **`response.files`:** una lista de CSS y JS requeridos por la página. Automáticamente se vinculará en el encabezado de la "layout.html" estándar. Para incluir un nuevo CSS o JS archivo, sólo añadir a esta lista. Se manejará duplicados. El orden es significativo.

- **response.flash:** parámetro opcional que puede ser incluido en los puntos de vista. Normalmente se utiliza para notificar al usuario acerca de algo que sucedió.
- **response.headers:** dict uno de los encabezados de respuesta HTTP.
- **response.menu:** parámetro opcional que puede ser incluido en los puntos de vista, normalmente usado para pasar un árbol de menús de navegación para la vista. Puede ser prestados por el ayudante MENU.
- **response.meta:** un objeto de almacenamiento (como un diccionario) que contiene información opcional de almacenamiento como meta `response.meta.author`, `response.meta.description`, `response.meta.keywords` y. El contenido de la variable de meta se coloca automáticamente en la etiqueta META efectiva por parte del código en "web2py_ajax.html" with is included by default by "views/layout.html".
- **response.postprocessing:** esta es una lista de funciones, vacía por defecto. Estas funciones se utilizan para filtrar el objeto de respuesta en la salida de una acción, antes de la salida se representa por la vista. Se puede utilizar para implementar el soporte para la plantilla de otros idiomas.
- **response.render(view, vars):** un método utilizado para llamar a la vista de forma explícita dentro del controlador. `view` es un parámetro opcional que es el nombre del archivo de vista, `vars` es un diccionario de valores con nombre pasó a la vista.
- **response.session_file:** secuencia de archivo que contiene la sesión.
- **response.session_file_name:** nombre del archivo donde se guarda la sesión.
- **response.session_id:** el id de la sesión actual. Se determina de forma automática. NUNCA cambiar esta variable.
- **response.session_id_name:** el nombre de la cookie de sesión para esta aplicación. NUNCA cambiar esta variable.
- **response.status:** el código de estado HTTP entero que se pasa a la respuesta. El valor predeterminado es 200 (OK).
- **response.stream(file, chunk_size):** cuando un controlador devuelve, web2py transmite el contenido del archivo de nuevo al cliente en bloques de tamaño `chunk_size`.

- **response.subtitle**: parámetro opcional que puede ser incluido en los puntos de vista. Debe contener el subtítulo de la página.
- **response.title**: parámetro opcional que puede ser incluido en los puntos de vista. Debe contener el título de la página y se dictarán por la etiqueta HTML de título en la cabecera.
- **response._vars**: esta variable sólo es accesible en una vista, no en la acción. Contiene el valor devuelto por la ayuda a la vista.
- **response.view**: el nombre de la plantilla de vista que debe representar la página. Esto se establece de forma predeterminada en:

```
1 "%s/%s.%s" % (request.controller, request.function, request.extension)
```

o, si el archivo anterior no se puede encontrar, para

```
1 "generic.%s" % (request.extension)
```

Cambie el valor de esta variable para modificar la vista de archivo asociado con una acción en particular.

- **response.xmlrpc(request, methods)**: cuando un controlador devuelve, esta función expone los métodos a través de XML-RPC (46). Esta función es obsoleta desde un mejor mecanismo está disponible y se describe en el capítulo 9.
- **response.write(text)**: un método para escribir texto en el cuerpo de la página de salida.

Desde **response** gluon.storage.Storage es un objeto que puede ser utilizado para almacenar otros atributos que es posible que desee pasar a la vista. Si bien no existe ninguna restricción técnica nuestra recomendación es sólo para almacenar las variables que han de ser prestados por todas las páginas del diseño general ("layout.html").

De todos modos, le recomendamos que se adhieren a las variables figuran en esta lista:

```

1 response.title
2 response.subtitle
3 response.flash
4 response.menu
5 response.meta.author
6 response.meta.description
7 response.meta.keywords
8 response.meta.*

```

porque esto hará que sea más fácil para usted para reemplazar el archivo norma "layout.html" que viene con web2py con otro archivo de diseño, que utiliza el mismo conjunto de variables.

Antiguas versiones de `response.author` usuario web2py lugar de `response.meta.author` y similar para los atributos de otros meta.

4.8 sesión

`session` es otro ejemplo de la clase `Storage`. Lo que se almacena en `session`, por ejemplo:

```

1 session.myvariable = "hello"

```

puede ser recuperada en un momento posterior:

```

1 a = session.myvariable

```

siempre y cuando el código se ejecuta en el mismo período de sesiones por el mismo usuario (siempre que el usuario no ha eliminado las cookies de sesión y la sesión no expire). Porque `session Storage` es un objeto, tratando de acceder a un atributo / clave que no se ha establecido no plantea una excepción, sino que devuelve `None` en su lugar.

El objeto de sesión tiene dos métodos importantes. Uno de ellos es **forget**:

```

1 session.forget()

```

Se dice que no web2py para guardar la sesión. Esto debería ser usado en esos controladores, cuyas acciones se les llama a menudo y no es necesario para rastrear la actividad del usuario.

El otro método es **connect**:

```
1 session.connect(request, response, db, masterapp=None)
```

donde `db` es el nombre de una conexión de base de datos abierta (como los devueltos por el DAL). Se dice web2py que desea guardar las sesiones en la base de datos y no en el sistema de archivos. web2py crea una tabla:

```
1 db.define_table('web2py_session',
2                 Field('locked', 'boolean', default=False),
3                 Field('client_ip'),
4                 Field('created_datetime', 'datetime', default=now),
5                 Field('modified_datetime', 'datetime'),
6                 Field('unique_key'),
7                 Field('session_data', 'text'))
```

y tiendas de cPickled sesiones en el campo de `session_data`.

El `masterapp=None` opción, por defecto, dice web2py para tratar de recuperar una sesión existente para la aplicación con el nombre de `request.application`, en la aplicación en ejecución.

Si desea que dos o más solicitudes a las sesiones de acción, establecer `masterapp` con el nombre de la aplicación principal.

Puede comprobar el estado de su solicitud en cualquier momento mediante la impresión de las variables `request`, `session` y `response` sistema. Una forma de hacerlo es crear una acción específica:

```
1 def status():
2     return dict(request=request, session=session, response=response)
```

4.9 cache

cache un objeto global también está disponible en el entorno de ejecución web2py. Cuenta con dos atributos:

- **cache.ram**: el caché de la aplicación en la memoria principal.
- **cache.disk**: el caché de la aplicación en el disco. cache es exigible, esto permite que sea utilizado como decorador para almacenar en caché las acciones y opiniones.

El siguiente ejemplo la función de caché en la memoria RAM `time.ctime()`:

```
1 def cache_in_ram():
2     import time
3     t = cache.ram('time', lambda: time.ctime(), time_expire=5)
4     return dict(time=t, link=A('click me', _href=request.url))
```

La salida de la `lambda: time.ctime()` se almacena en caché en la memoria RAM durante 5 segundos. El 'time' cadena se utiliza como clave de caché.

El siguiente ejemplo la función de caché en el disco `time.ctime()`:

```
1 def cache_on_disk():
2     import time
3     t = cache.disk('time', lambda: time.ctime(), time_expire=5)
4     return dict(time=t, link=A('click me', _href=request.url))
```

La salida de la `lambda: time.ctime()` se almacena en caché en el disco (utilizando el módulo de lado) durante 5 segundos.

Los escondites siguiente ejemplo la función `time.ctime()` tanto la memoria RAM y disco:

```
1 def cache_in_ram_and_disk():
2     import time
3     t = cache.ram('time', lambda: cache.disk('time',
4                                     lambda: time.ctime(), time_expire=5),
5                                     time_expire=5)
6     return dict(time=t, link=A('click me', _href=request.url))
```

La salida de la `lambda: time.ctime()` se almacena en caché en el disco (utilizando el módulo de lado) y luego en la memoria RAM durante 5 segundos. web2py mira primero en la memoria RAM y si no existe se ve en el disco. Si no lo está en la memoria RAM o el disco, `lambda: time.ctime()` se ejecuta y la memoria caché se actualiza. Esta técnica es útil en un entorno multiproceso. Las dos veces no tiene que ser el mismo.

El ejemplo siguiente es el almacenamiento en caché en la memoria RAM de la salida de la función de controlador (pero no la vista):

```

1 @cache(request.env.path_info, time_expire=5, cache_model=cache.ram)
2 def cache_controller_in_ram():
3     import time
4     t = time.ctime()
5     return dict(time=t, link=A('click me', _href=request.url))

```

El diccionario devuelto por `cache_controller_in_ram` se almacena en caché en la memoria RAM durante 5 segundos. Tenga en cuenta que el resultado de seleccionar una base de datos no puede ser almacenada en caché sin haber sido serializados. Una mejor manera es guardar la base de datos directamente utilizando el método de `select cache` argumento.

El siguiente ejemplo es el almacenamiento en caché de la salida de la función de controlador en el disco (pero no la vista):

```

1 @cache(request.env.path_info, time_expire=5, cache_model=cache.disk)
2 def cache_controller_on_disk():
3     import time
4     t = time.ctime()
5     return dict(time=t, link=A('click to reload',
6                                _href=request.url))

```

El diccionario devuelto por `cache_controller_on_disk` se almacena en caché en el disco durante 5 segundos. Recuerde que no puede almacenar en caché web2py un diccionario que contiene objetos no-pickleable.

También es posible almacenar en caché las vistas. El truco es hacer que la vista en la función de controlador, de modo que el controlador devuelve una cadena. Esto se hace mediante la devolución `response.render(d)`, donde es

el diccionario que la intención de pasar a la vista:

El `time_expire` se puede establecer en `0` para forzar una actualización de la caché y `None` para evitar que el contenido de cada vez que expira.

Usted puede limpiar la caché o más variables con una

```
1 cache.ram.clear(regex='...')
```

donde `regex` es una expresión regular matching todas las teclas que desea quitar de la caché.

El siguiente ejemplo, la caché de salida de la función de controlador en la memoria RAM (incluyendo el punto de vista prestados):

```
1 @cache(request.env.path_info, time_expire=5, cache_model=cache.ram)
2 def cache_controller_and_view():
3     import time
4     t = time.ctime()
5     d = dict(time=t, link=A('click to reload', _href=request.url))
6     return response.render(d)
```

`response.render(d)` devuelve la vista representada como una cadena que está en caché durante 5 segundos. Esta es la mejor y más rápida forma de almacenamiento en caché.

También es posible definir otros mecanismos de almacenamiento en caché como memcache. Memcache está disponible a través `gluon.contrib.memcache` y se analiza con más detalles en el capítulo 11.

4.10 URL

La función de `URL` es una de las funciones más importantes en web2py. Genera rutas internas dirección de las acciones y los archivos estáticos.

Aquí está un ejemplo:

```
1 URL('f')
```

se asigna en

```
1 /[application]/[controller]/f
```

Observe que el resultado de la función `URL` depende del nombre de la aplicación actual, el controlador de llamada y otros parámetros. web2py admite la asignación de URL y revertir la asignación de URL. Asignación de URL permite redefinir el formato de URL externa. Si utiliza la función `URL` para generar todas las direcciones URL internas, a continuación, adiciones o modificaciones de asignaciones de dirección se previenen la ruptura de los vínculos dentro de la aplicación web2py.

Puede pasar parámetros adicionales a la función de `URL`, es decir, términos adicionales en la dirección URL (`args`) y las variables de consulta de URL (`vars`):

```
1 URL('f', args=['x', 'y'], vars=dict(z='t'))
```

se asigna en

```
1 /[application]/[controller]/f/x/y?z=t
```

Los atributos `args` son automáticamente analizados, decodificado, y finalmente almacenados en `request.args` por web2py. Del mismo modo, la `vars` se analiza, decodifica, y se almacena en `request.vars`. `args` y `vars` proporcionan el mecanismo básico por el cual intercambia información con el cliente web2py el navegador.

Si `args` contiene sólo un elemento, no hay necesidad de pasar de una lista.

También puede utilizar la función `URL` para generar direcciones URL a las acciones en otros controladores y otras aplicaciones:

```
1 URL('a', 'c', 'f', args=['x', 'y'], vars=dict(z='t'))
```

se asigna en

```
1 /a/c/f/x/y?z=t
```

También es posible especificar la aplicación, control y operan de acuerdo con los argumentos con nombre:

```
1 URL(a='a', c='c', f='f')
```

Si el nombre de la aplicación una falta de la aplicación actual se supone.

```
1 URL('c', 'f')
```

Si el nombre del controlador no se encuentra, el actual se supone.

```
1 URL('f')
```

En lugar de pasar el nombre de una función de control también es posible pasar de la función misma

```
1 URL(f)
```

Por los motivos antes mencionados, siempre debe utilizar la función de `URL` para generar direcciones URL de archivos estáticos para sus aplicaciones. Los archivos estáticos se almacenan en la subcarpeta `static` de la aplicación (que es donde van cuando se cargan mediante la interfaz de administración). `web2py` establece un virtual "estática" controlador de cuyo trabajo consiste en recuperar los archivos de la subcarpeta `static`, determinar su tipo de contenido, y transmitir el archivo al cliente. El ejemplo siguiente genera la dirección del archivo estático "image.png":

```
1 URL('static', 'image.png')
```

se asigna en

```
1 /[application]/static/image.png
```

No es necesario para codificar / escapar de los argumentos `args` y `vars`, esto se hace automáticamente.

4.11 HTTP y redirigir

web2py sólo define una nueva excepción llamada `HTTP`. Esta excepción se puede levantar en cualquier parte de un modelo, un controlador, o una vista con el comando:

```
1 raise HTTP(400, "my message")
```

Hace que el flujo de control para saltar fuera de usuario el código, de vuelta a web2py, y devolver una respuesta HTTP como:

```
1 HTTP/1.1 400 BAD REQUEST
2 Date: Sat, 05 Jul 2008 19:36:22 GMT
3 Server: Rocket WSGI Server
4 Content-Type: text/html
5 Via: 1.1 127.0.0.1:8000
6 Connection: close
7 Transfer-Encoding: chunked
8
9 my message
```

El primer argumento de la `HTTP` es el código de estado HTTP. El segundo argumento es la cadena que se devuelve como el cuerpo de la respuesta. Adicional llamado argumentos opcionales se utilizan para construir la cabecera HTTP de respuesta. Por ejemplo:

```
1 raise HTTP(400, 'my message', test='hello')
```

genera:

```
1 HTTP/1.1 400 BAD REQUEST
2 Date: Sat, 05 Jul 2008 19:36:22 GMT
3 Server: Rocket WSGI Server
4 Content-Type: text/html
5 Via: 1.1 127.0.0.1:8000
6 Connection: close
7 Transfer-Encoding: chunked
8 test: hello
9
10 my message
```

Si no desea confirmar la transacción de base de datos abierta, rollback antes de levantar la excepción.

Cualquier otra excepción que las causas HTTP web2py para revertir cualquier transacción de base de datos abierta, el registro de rastreo de error, expedir un billete para el visitante, y devolver una página de error estándar.

Esto quiere decir que HTTP sólo puede ser usado para control de flujo entre páginas. Otras excepciones han de ser capturados por la aplicación, de lo contrario son multados por web2py.

El comando:

```
1 redirect('http://www.web2py.com')
```

es simplemente un atajo para:

```
1 raise HTTP(303,
2     'You are being redirected <a href="%s">here</a>' % location,
3     Location='http://www.web2py.com')
```

Los argumentos con nombre del método inicializador HTTP se traducen en directivas encabezado HTTP, en este caso, la redirección de ubicación de destino. redirect tiene un segundo argumento opcional, que es el código de estado HTTP para la redirección (303 por defecto). Cambiar este número a 307 para una redirección temporal o 301 para una redirección permanente.

La forma más común de redirigir el usuario es redirigir a otras páginas en la misma aplicación y (opcionalmente) pasar parámetros:

```
1 redirect(URL('index',args=(1,2,3),vars=dict(a='b')))
```

4.12 T e Internacionalización

El `T` es el objeto traductor de idiomas. Constituye una única instancia global de la clase `web2py gluon.language.translator`. Todas las constantes de cadena (y sólo las constantes de cadena) debe ser marcada por la `T`, por ejemplo:

```
1 a = T("hello world")
```

Las cadenas que están marcados con el `T` son identificados por web2py como la necesidad de traducción de idiomas y se debe traducir cuando el código (en el modelo, controlador, o vista) se ejecuta. Si la cadena a traducir no es una constante, sino variable, que se añadirá al archivo de traducción en tiempo de ejecución (salvo el GAME), que se tradujo más tarde.

El objeto `T` también puede contener interpolados variables, por ejemplo:

```
1 a = T("hello %(name)s", dict(name="Massimo"))
```

La primera cadena se traduce de acuerdo con el archivo de idioma solicitado y la variable se sustituye `name` independientemente de la lengua.

La concatenación de cadenas de traducción no es una buena idea, por eso web2py no le permite hacer:

```
1 T("blah ") + name + T(" blah") # invalid!
```

pero permite:

```
1 T("blah %(name)s blah", dict(name='Tim'))
```

o el

```
1 T("blah %(name)s blah") % dict(name='Tim')
```

sintaxis alternativa

En ambos casos, la traducción se produce antes de que el nombre de la variable se sustituye en la ranura de la "%(name)s". El siguiente en su lugar no debe ser utilizada:

```
1 T("blah %(name)s blah" % dict(name='Tim'))
```

porque la traducción se produciría después de la sustitución.

El idioma requerido es determinado por el campo "Accept-Language" en la cabecera HTTP, pero esta selección se puede sobrescribir mediante programación solicitando un archivo específico, por ejemplo:

```
1 T.force('it-it')
```

que lee el archivo de idioma "languages/it-it.py". archivos de idioma puede ser creado y editado a través de la interfaz administrativa.

Normalmente, la traducción cadena es evaluada perezosamente cuando la vista se representa, por lo que el traductor `force` método no se debe llamar dentro de un punto de vista.

Es posible desactivar a través de la evaluación perezosa

```
1 T.lazy = False
```

De esta manera, las cadenas se tradujo de inmediato por el operador T basado en el actualmente aceptado o forzado el lenguaje.

Un problema común es la siguiente. La aplicación original está en Inglés. Supongamos que hay un archivo de traducción (por ejemplo archivo italiano, "it-it.py") and the HTTP client declares that it accepts both English (en) and Italian (it-it) in that order. The following unwanted situation occurs: web2py does not know the default is written in English (en). Therefore, it prefers translating everything into Italian (it-it) because it only found the Italian translation file. If it had not found the "it-it.py", habría utilizado las cadenas de idioma por defecto (Inglés).

Hay dos soluciones para este problema: crear un lenguaje de traducción para Inglés, que sería redundante e innecesario, o mejor, dicen web2py que los idiomas deberían utilizar las cadenas de idioma por defecto (las cadenas codificadas en la aplicación). Esto se puede hacer con:

```
1 T.set_current_language('en', 'en-en')
```

Almacena en un `T.current_languages` lista de lenguas que no requieren traducción y las fuerzas de una recarga de los archivos de idioma.

Tenga en cuenta que "it" and "it-it" son lenguas diferentes desde el punto de vista de web2py. En apoyo de los dos, uno necesitaría dos archivos de traducción, siempre en minúsculas. Lo mismo es cierto para todos los demás idiomas.

El lenguaje aceptado actualmente se almacena en

```
1 T.accepted_language
```

Lamentamos que (...) `T` no sólo traducir cadenas sino que también puede traducirse variables:

```
1 >>> a="test"
2 >>> print T(a)
```

En este caso, el "test" mundo en traducir, pero si no se encuentra y si el sistema de archivos se puede escribir, se lo añadiremos a la lista de palabras a traducir en el archivo de idioma.

4.13 Galletas

web2py utiliza las cookies para el manejo de módulos de Python "cookies";

Las cookies del navegador está en `request.cookies` y las cookies enviadas por el servidor están en `response.cookies`.

Usted puede configurar una cookie de la siguiente manera:

```
1 response.cookies['mycookie'] = 'somevalue'
2 response.cookies['mycookie']['expires'] = 24 * 3600
3 response.cookies['mycookie']['path'] = '/'
```

La segunda línea indica al navegador la cookie para mantener durante 24 horas. La tercera línea indica al navegador que envíe la cookie de nuevo a cualquier aplicación (ruta URL) en el dominio actual.

La cookie puede ser protegido con:

```
1 response.cookies['mycookie']['secure'] = True
```

Una cookie de seguridad sólo se envía de vuelta a través de HTTPS y no a través de HTTP.

La cookie se puede recuperar con:

```
1 if request.cookies.has_key('mycookie'):
2     value = request.cookies['mycookie'].value
```

A menos que las sesiones son personas con discapacidad, web2py, debajo del capó, establece la cookie siguiente y lo utiliza para manejar las sesiones:

```
1 response.cookies[response.session_id_name] = response.session_id
2 response.cookies[response.session_id_name]['path'] = "/"
```

4.14 Aplicación init

Al implementar web2py, tendrá que establecer una aplicación predeterminada, es decir, la aplicación que se inicia cuando hay un vacío en la ruta URL, como en:

```
1 http://127.0.0.1:8000
```

De forma predeterminada, cuando se enfrentan a una ruta de acceso vacío, web2py busca una aplicación llamada **init**. Si no hay una aplicación de inicio busca una aplicación llamada **welcome**.

Aquí hay tres maneras de establecer la aplicación por defecto:

- Llame a su "init" aplicación por defecto.
- Hacer un enlace simbólico desde "applications/init" a la carpeta de la aplicación.
- El uso de reescritura de URL como se comenta en la siguiente sección.

4.15 URL Rewrite

web2py tiene la capacidad de reescribir la dirección URL de las solicitudes entrantes antes de llamar a la acción del controlador (asignación de URL), ya la inversa, web2py puede volver a escribir la ruta URL generada por la función URL (reverse mapping URL). Una de las razones para hacer esto es para el manejo de URLs legado, otro es el de simplificar los caminos y hacerlos más cortos.

Para utilizar esta función, cree un nuevo archivo en la carpeta llamada "web2py" "routes.py" y definen dos listas (o tuplas) de 2-tuplas `routes_in` y `routes_out`. Cada tupla contiene dos elementos: el patrón se va a sustituir y la cadena que lo reemplaza. Por ejemplo:

```

1 routes_in = (
2     ('/testme', '/examples/default/index'),
3 )
4 routes_out = (
5     ('/examples/default/index', '/testme'),
6 )

```

Con estas rutas, la dirección URL:

¹ <http://127.0.0.1:8000/testme>

se asigna a:

```
1 http://127.0.0.1:8000/examples/default/index
```

Para el visitante, todos los enlaces a la URL de la página se parece a `/testme`.

Los patrones tienen la misma sintaxis que las expresiones regulares de Python. Por ejemplo:

```
1 ('.*\.php', '/init/default/index'),
```

mapas de todas las URL que termina en ".php" a la página de índice.

Algunas veces usted quiere deshacerse del prefijo aplicación de las URLs, porque va a exponer sólo una aplicación. Esto se puede lograr con:

```
1 routes_in = (
2   ('/(?P<any>.*)', '/init/\g<any>'),
3 )
4 routes_out = (
5   ('/init/(?P<any>.*)', '/\g<any>'),
6 )
```

También existe una sintaxis alternativa que puede ser mezclada con la notación de expresión regular anterior. Consiste en utilizar en vez de `name` (`?P<name>[\w_]+`) o `\g<name>`. Por ejemplo:

```
1 routes_in = (
2   ('/$c/$f', '/init/$c/$f'),
3 )
4 routes_out = (
5   ('/init/$c/$f', '/$c/$f'),
6 )
```

También eliminaría el prefijo "/example" aplicación en todas las direcciones URL.

Usando la notación \$, automáticamente puede asignar a la `routes_in` `routes_out`, siempre y cuando no utilice ningún expresiones regulares. Por ejemplo:

```

1 routes_in = (
2   ('/$c/$f', '/init/$c/$f'),
3 )
4
5 routes_out = [(x, y) for (y, x) in routes_in]

```

Si hay varias rutas, la primera para que coincida con la URL se ejecuta. Si no hay ningún patrón de los partidos, el camino se deja sin cambios.

Usted puede utilizar `$anything` para coincidir con nada hasta que el final de la fila.

Aquí está una "routes.py" mínima para el manejo de robots favicon y pide:

```

1 routes_in = (
2   ('/favicon.ico', '/examples/static/favicon.ico'),
3   ('/robots.txt', '/examples/static/robots.txt'),
4 )
5 routes_out = ()

```

Aquí está un ejemplo más complejo que expone una "myapp" sola aplicación sin prefijos innecesario, sino que también expone **admin**, **appadmin** y estático:

```

1 routes_in = (
2   ('/admin/$anything', '/admin/$anything'),
3   ('/static/$anything', '/myapp/static/$anything'),
4   ('/appadmin/$anything', '/myapp/appadmin/$anything'),
5   ('/favicon.ico', '/myapp/static/favicon.ico'),
6   ('/robots.txt', '/myapp/static/robots.txt'),
7 )
8 routes_out = [(x, y) for (y, x) in routes_in[:-2]]

```

La sintaxis general de las rutas es más compleja que los ejemplos simples que hemos visto hasta ahora. Aquí está una más general y representativo ejemplo:

```

1 routes_in = (
2   ('140|191|.\d+\.\d+:https://www.web2py.com:POST /(?P<any>.*)\.php',
3   '/test/default/index?vars=g<any>'),
4 )

```

Al trazar https POST peticiones para acoger `www.web2py.com` de una dirección IP remota se pongan en venta el

```
1 140\.191\.\d+\.\d+
```

de expresiones regulares solicitando una página en que aparezca la expresión regular

```
1 /( ?P<any>.* )\.php !
```

en

```
1 /test/default/index?vars=\g<any>
```

`\g<any>` donde se sustituye por la expresión regular coincidente.

La sintaxis general es

```
1 [remote address]:[protocol]://[host]:[method] [path]
```

La expresión completa se corresponde como una expresión regular, por lo que debe ser siempre `".` escapó y cualquier subexpresión coincidente puede ser capturado utilizando `(?P<...>...)` acuerdo con la sintaxis regex Python.

Esto permite redirigir las solicitudes basadas en la dirección IP del cliente o dominio, en función del tipo de la petición, sobre el método, y el camino. También permite asignar diferentes máquinas virtuales en diferentes aplicaciones. Cualquier subexpresión emparejado se puede utilizar para construir la URL de destino y, finalmente, pasa como una variable GET.

Todos los servidores web más importantes, como Apache y lighttpd, también tienen la capacidad para reescribir URL. En un entorno de producción le sugerimos que el servidor web realiza la reescritura de URL.

4.16 Rutas a un error

También puede utilizar "routes.py" para redirigir al visitante a acciones especiales en caso de que haya un error en el servidor. Puede especificar esta asignación a nivel mundial, para cada aplicación, para cada código de error, para cada aplicación y el código de error. Aquí está un ejemplo:

```

1 routes_onerror = [
2     ('init/400', '/init/default/login'),
3     ('init/*', '/init/static/fail.html'),
4     ('*/404', '/init/static/cantfind.html'),
5     ('*/*', '/init/error/index')
6 ]

```

Para cada tupla se corresponde con la primera cadena contra la "[appname]/[error code]". Si un partido se encuentra el usuario es redirigido a la URL en la segunda cadena de la tupla coincidente. En el caso de un billete fue emitido, el boleto se pasa a la nueva dirección URL como una variable llamada GET billete. errores incomparables mostrar una página de error por defecto. Esta página de error por defecto también se puede personalizar en:

```

1 error_message = '<html><body><h1>Invalid request</h1></body></html>'
2 error_message_ticket = '''<html><body><h1>Internal error</h1>
3     Ticket issued: <a href="/admin/default/ticket/%(ticket)s"
4     target="_blank">%s</a></body></html>'''
```

La primera variable contiene el mensaje de error cuando una aplicación no válida se solicita. La segunda variable contiene el mensaje de error cuando los billetes se expiden.

4.17 Cron

El cron web2py proporciona la capacidad para aplicaciones para ejecutar tareas en horas preestablecidas, en la forma de una plataforma independiente.

Para cada aplicación, la funcionalidad cron se define mediante un archivo

crontab "app/cron/crontab", siguiendo la sintaxis definida en la ref. (45) (con algunas extensiones que se web2py específico).

Esto significa que cada aplicación puede tener una configuración de cron por separado y que de configuración de cron se puede cambiar desde dentro web2py sin afectar al huésped sistema operativo en sí.

Aquí está un ejemplo:

```

1 0-59/1 * * * * root python /path/to/python/script.py
2 30      3 * * * root *applications/admin/cron/db_vacuum.py
3 */30    * * * * root **applications/admin/cron/something.py
4 @reboot root    *mycontroller/myfunction
5 @hourly root   *applications/admin/cron/expire_sessions.py

```

Las dos últimas líneas en este ejemplo, uso de extensiones a la sintaxis cron regular para proporcionar funcionalidad adicional web2py. cron web2py tiene una sintaxis algunos detalles adicionales para apoyar web2py aplicación.

Si la tarea / script se precede con un asterisco (*) y termina con la ".py", que se ejecutará en el entorno web2py. Esto significa que usted tendrá todos los controladores y modelos a su disposición. Si utiliza dos asteriscos (**), los modelos no se ejecutarán. Esta es la forma recomendada de convocatoria, ya que tiene menos gastos generales y evita posibles problemas de seguridad.

Observe que las secuencias de comandos / funciones ejecutadas en el medio ambiente requieren una web2py db.commit() manual al final de la función o la operación se revertirá. web2py no genera entradas o seguimientos significativa en el modo de concha (en el que se ejecuta cron). Asegúrese de que su código se ejecuta sin errores web2py antes de configurarlo como una tarea cron, como usted probablemente no será capaz de ver cuando se ejecuta desde cron.

Además, tenga cuidado de cómo utiliza modelos. Si bien la ejecución que ocurre en un proceso separado, los bloqueos de base de datos hay que tener en cuenta a fin de evitar las páginas de espera para las tareas cron que esté

bloqueando la base de datos. Utilice la sintaxis `**` si no es necesario utilizar la base de datos en su tarea cron.

También puede llamar a una función de control. No hay necesidad de especificar una ruta. El responsable del tratamiento y la función será la de la solicitud invoca. Tenga especial cuidado acerca de las advertencias enumeradas anteriormente. Ejemplo:

```
1 */30 * * * * root *mycontroller/myfunction
```

Si se especifica `@reboot` en el primer campo en el archivo crontab, la tarea que se ejecutará sólo una vez, en el inicio web2py. Usted puede utilizar esta función si desea pre-cache, cheque o inicializar los datos de una aplicación en el arranque web2py. Tenga en cuenta que las tareas cron se ejecutan en paralelo con la aplicación — si la aplicación no está preparada para atender las solicitudes hasta que la tarea cron está terminado, se debe implementar controles para reflejar esto. Ejemplo:

```
1 @reboot * * * * root *mycontroller/myfunction
```

Dependiendo de cómo se invoca web2py, hay cuatro modos de funcionamiento para el cron web2py.

- cron Soft: disponible en todos los modos de ejecución
- cron duro: disponible si se utiliza el built-in del servidor web (ya sea directamente o través de mod_proxy de Apache)
- cron exterior: disponible si usted tiene acceso a la función del sistema cron propia
- No cron

El valor predeterminado es cron duro si está utilizando el built-in del servidor web y, en todos los demás casos, el valor predeterminado es cron suave. cron Soft es el valor por defecto si usted está usando CGI, FastCGI o WSGI. Sus tareas serán ejecutadas en la primera convocatoria (carga de la página) para web2py después del tiempo especificado en el crontab (pero después de

procesar la página, así que no hay retraso para el usuario es visible). Obviamente, existe cierta incertidumbre con exactitud cuando la tarea se ejecutará en función del tráfico, el sitio recibe. Además, la tarea cron puede obtener interrumpe si el servidor web tiene un tiempo de espera de carga de página establecidos. Si estas limitaciones no son aceptables, consulte "external cron". cron Soft es un último recurso razonable, pero si su servidor web permite a otros métodos de cron, que conviene preferir el cron suave. cron duro es el valor por defecto si usted está usando el built-in del servidor web (ya sea directamente o través de mod_proxy Apache). Hard cron se ejecuta en un hilo paralelo, así que a diferencia de cron suave no hay limitaciones con respecto al tiempo de ejecución o precisión el tiempo de ejecución. cron externa no es por defecto en cualquier escenario, pero requiere que usted tenga acceso a las instalaciones del sistema cron. Se ejecuta en un proceso paralelo, por lo que ninguna de las limitaciones de cron suave aplicar. Esta es la forma recomendada de utilizar cron en WSGI o FastCGI.

Ejemplo de la línea para añadir el crontab del sistema, (normalmente / etc / crontab):

```
1 0-59/1 * * * * web2py cd /var/www/web2py/ && python web2py.py -C -D 1 >> /tmp/cron.  
output 2>&1
```

Si está ejecutando cron externo, asegúrese de agregar comandos a la N-parámetro de la línea a su script de inicio o Ajuste web2py lo que no hay colisión de múltiples tipos de cron.

En caso de que usted no necesita ninguna funcionalidad cron dentro de un proceso en particular, puede utilizar la N-parámetro de línea de comandos para deshabilitarlo. Tenga en cuenta que esto puede desactivar algunas tareas de mantenimiento (como la limpieza automática de directorios sesión). El uso más común de esta función:

- Usted ya ha configurado cron externa que se activa con el sistema (más común con configuraciones WSGI)
- Si desea depurar su aplicación sin interferir cron ya sea con acciones o con

salida

4.18 Antecedentes Los procesos y las colas de tareas

Mientras que cron es útil para ejecutar tareas a intervalos regulares de tiempo no siempre es la mejor solución para ejecutar una tarea de fondo. Para ello web2py proporciona la capacidad de ejecutar cualquier script python como si fuera un controlador de información privilegiada:

```
1 python web2py.py -S app -M -N -R applications/app/private/myscript.py -A a b c
```

donde dice -S app web2py para ejecutar "myscript.py" as "app", -M cuenta web2py para ejecutar los modelos, -N dice que no web2py ejecutar el cron, y pasa -A a b c argumentosopcionales de línea de comandos para sys.argv=['a','b','c'] "myscript.py".

Un caso de prueba típica consiste en la tramitación de una cola.

Considere este modelo

```
1 db.define_table('queue',
2     Field('status'),
3     Field('email'),
4     Field('subject'),
5     Field('message'))
```

y una aplicación que encola los mensajes que se envían por

```
1 db.queue.insert(status='pending',
2                  email='you@example.com',
3                  subject='test',
4                  message='test')
```

La secuencia de comandos en proceso de fondo que envía los correos electrónicos podrían

```
1 # in file myscript.py
```

```

2 import time
3 while True:
4     rows = db(db.queue.status=='pending').select()
5     for row in rows:
6         if mail.send(to=row.email,
7                     subject=row.subject,
8                     message=row.message):
9             row.update_record(status='sent')
10        else:
11            row.update_record(status='failed')
12        db.commit()
13    time.sleep(60) # check every minute

```

Tenga en cuenta que el objeto `mail` se define en el archivo de `db.py` en la aplicación de andamios, debido a la opción de `-M` es visible aquí. Es posible que tenga que ser configurado en `db.py` para que funcione correctamente. Observe también que es importante para cometer cualquier cambio lo antes posible con el fin no bloquear la base de datos a otros procesos concurrentes.

Este tipo de proceso de fondo no debería ser ejecutado a través de cron (excepto tal vez para iniciar el sistema cron @) porque usted necesita estar seguro de que no más de una ocasión se está ejecutando al mismo tiempo. Con cron es posible que si el proceso comienza en la iteración 1 cron y no se completa por iteración cron 2, por lo cron comienza de nuevo, y otra vez, y otra vez lo que se obstruye el servidor de correo.

4.19 Tercera Parte módulos

`web2py` está escrito en Python, por lo que puede importar y utilizar cualquier módulo de Python, incluyendo módulos de terceros. Simplemente tiene que ser capaz de encontrarlos.

Los módulos pueden ser instalados en el directorio oficial de Python, o cualquier lugar "site-packages" su aplicación pueda encontrar.

Los módulos en "site-packages" directory are, as the name suggests, site-

level packages. Applications requiring site-packages are not portable unless these modules are installed separately. The advantage of having modules in "site-packages" is that multiple applications can share them. Let's consider, for example, the plotting package called "matplotlib". Usted puede instalar desde el shell con el comando `easy_install PEAK`:

```
1 easy_install py-matplotlib
```

y entonces usted puede importar a cualquier modelo / controlador / ver con:

```
1 import matplotlib
```

Windows web2py distribuciones binarias tiene un sitio en paquetes en la carpeta de nivel superior. La distribución binaria Mac tiene una carpeta de sitio los paquetes en la carpeta:

`web2py.app/Contents/Resources/site-packages`

También puede instalar los paquetes manualmente en la aplicación "modules" folder. The advantage is that the module will be automatically copied and distributed with the application. If the application is called "test", you can import "mymodule" con: web2py también proporciona una función de `local_import`. He aquí un ejemplo de uso:

```
1 mymodule = local_import(mymodule)
```

La función busca en la carpeta `mymodule` aplicación módulos locales y las importaciones con el nombre en el lado izquierdo de la igualdad.

Esta función toma tres argumentos: `name`, `reload` y `app`. Cuando se especifica `reload=True`, se vuelve a importar el módulo en cada solicitud, de lo contrario el proceso de python sólo se importa el módulo una vez. El valor predeterminado es `reload=False`. `app` es el nombre de la aplicación desde la que importar el módulo, por defecto se `request.application`.

La razón de esta función es que, desde un servidor puede ejecutar muchos casos web2py usted no desea agregar módulos a sus trayectorias `sys.path`

más la búsqueda pasaría a depender de la orden.

4.20 Entorno de ejecución

Si bien todo lo descrito aquí funciona bien, le recomendamos lugar la construcción de su aplicación utilizando componentes, como se describe en el capítulo 13.

modelo web2py y archivos de los controladores no son los módulos de Python en que no pueden ser importados mediante la instrucción Python `import`. La razón de esto es que los modelos y los controladores están diseñados para ser ejecutados en un ambiente preparado que ha sido pre-pobladas con web2py objetos globales (solicitud, respuesta, período de sesiones, la caché y T) y funciones auxiliares. Esto es necesario porque Python es una forma estática (léxico) Idioma ámbito, mientras que el medio ambiente web2py se crea de forma dinámica. web2py `exec_environment` proporciona la función que le permite a los modelos de acceso y controladores directamente. `exec_environment` crea un entorno de ejecución web2py, carga el archivo en él y luego devuelve un objeto que contiene el almacenamiento medio ambiente. El objeto de almacenamiento también sirve como un mecanismo de espacio de nombres. Cualquier archivo de Python diseñados para ser ejecutados en el entorno de ejecución se puede cargar con `exec_environment`. Usos para `exec_environment` incluyen:

- Acceso a datos (modelos) de otras aplicaciones.
- Obtener acceso a objetos globales de otros modelos o controladores.
- Ejecución de las funciones del controlador de los controladores de otros.
- Cargando bibliotecas auxiliares en todo el sitio.

Este ejemplo lee filas de la tabla en la aplicación user cas:

```
1 from gluon.shell import exec_environment
```

```

2 cas = exec_environment('applications/cas/models/db.py')
3 rows = cas.db().select(cas.db.user.ALL)

```

Otro ejemplo: suponga que tiene un controlador que contiene "other.py":

```

1 def some_action():
2     return dict(remote_addr=request.env.remote_addr)

```

Aquí está cómo usted puede llamar a esta acción de otro controlador (o de la concha web2py):

```

1 from gluon.shell import exec_environment
2 other = exec_environment('applications/app/controllers/other.py', request=request)
3 result = other.some_action()

```

En la partida 2, `request=request` es opcional. Tiene el efecto de pasar la solicitud actual para el medio ambiente de la "other". Sin este argumento, el medio ambiente debería contener un nuevo y vacío (aparte de `request.folder` objeto de solicitud). También es posible pasar de una respuesta y un objeto de sesión para `exec_environment`. Tenga cuidado al solicitar que pasa, la respuesta y los objetos de sesión — modificación introducida por la acción llamada o codificación de las dependencias en las medidas que se podrían llevar a efectos secundarios inesperados.

La llamada a la función en la línea 3 no se ejecuta el punto de vista, sino que simplemente devuelve el diccionario, a menos `response.render` se llama explícitamente por "some_action".

Una advertencia final: no utilice inadecuadamente `exec_environment`. Si desea que los resultados de las acciones en otra aplicación, probablemente debería aplicar una API XML-RPC (aplicación de una API XML-RPC con web2py es casi trivial). No utilice `exec_environment` como un mecanismo de redirección, el uso del auxiliar `redirect`.

4.21 Cooperación

Hay muchas maneras de las aplicaciones pueden cooperar:

- Las aplicaciones pueden conectarse a la misma base de datos y por lo tanto comparten las tablas. No es necesario que todas las tablas de la base de datos se definen todas las aplicaciones, sino que deben ser definidos por las aplicaciones que los utilizan. Todas las aplicaciones que utilizan la misma mesa, pero uno, debe definir la tabla con `migrate=False`.
- Las aplicaciones se pueden integrar los componentes de otras acciones con la CARGA ayudado (que se describe en el capítulo 13).
- Las aplicaciones pueden compartir sesiones.
- Las aplicaciones pueden llamarse unos a otros las acciones de forma remota a través de XML-RPC.
- Las aplicaciones pueden acceder a todas las demás archivos a través del sistema de archivos (suponiendo que comparten el mismo sistema de ficheros).
- Las aplicaciones pueden llamarse unos a otros las acciones a nivel local utilizando `exec_environment` como se discutió anteriormente.
- Las solicitudes se pueden importar los demás módulos usando la sintaxis:
- Las aplicaciones pueden importar cualquier módulo en la ruta de búsqueda `PYTHONPATH`, `sys.path`.

Una aplicación puede cargar el período de sesiones de otra aplicación usando el comando:

```
1 session.connect(request, response, masterapp='appname', db=db)
```

Aquí "appname" es el nombre de la aplicación principal, la que establece el `session_id` inicial en la cookie. `db` es una conexión de base de datos a la base de datos que contiene la tabla de sesiones (`web2py_session`). Todas las aplicaciones que las sesiones de participación debe utilizar la misma base de datos para el almacenamiento de sesión.

Una aplicación puede cargar un módulo de otra aplicación utilizando

```
othermodule = local_import('othermodule', app='otherapp')
```

Si un módulo de funciones necesita acceder a uno de los objetos del núcleo (solicitud, respuesta, período de sesiones, la caché y T), los objetos se debe pasar explícitamente a la función. No deje que el módulo de crear otra instancia de los objetos básicos. De lo contrario, la función no se comporta como se espera.

4.22 WSGI

web2py y WSGI tienen relación de amor odio. Nuestra es de tipo prospectivo que WSGI fue desarrollado como un protocolo para conectar los servidores web a las aplicaciones web de una forma portátil y la usamos para ese fin. web2py en su núcleo es una aplicación WSGI: gluon.main.wsgibase. Algunos desarrolladores han llevado a sus límites WSGI como un protocolo para las comunicaciones y middleware de desarrollar aplicaciones web como una cebolla con muchas capas (cada capa es un middleware WSGI desarrollado de forma independiente en todo el marco). web2py no adopta esta estructura interna. Esto se debe a que sentimos la funcionalidad principal de un marcos (errores de cookies de sesión manipulación,,, transacciones, su expedición), puede ser mejor optimizada para la velocidad y la seguridad si se manejan por una capa única global.

Sin embargo, web2py le permite usar aplicaciones de terceros WSGI y middleware de tres maneras (y sus combinaciones):

- Puede editar el archivo y "wsgihandler.py" incluyen cualquier tercero WSGI middleware.
- Puede conectar el middleware de terceros WSGI a actuaciones específicas en sus aplicaciones.
- Usted puede llamar a un tercero WSGI aplicación de sus acciones.

La única limitación es que no se puede utilizar middleware tercero para reemplazar las funciones básicas web2py.

4.22.1 Exteriores Middleware

Considere el archivo "wsgibase.py":

```

1 #...
2 LOGGING = False
3 #...
4 if LOGGING:
5     application = gluon.main.appfactory(wsgiapp=gluon.main.wsgibase,
6                                         logfilename='httpserver.log',
7                                         profilerfilename=None)
8 else:
9     application = gluon.main.wsgibase

```

Cuando entrada se establece en True, gluon.main.wsgibase está envuelto por el gluon.main.appfactory función de middleware. Proporciona el registro en el archivo de "httpserver.log". De manera similar se puede añadir ningún middleware de terceros. Nos referimos a la documentación oficial de WSGI para más detalles.

Interior Middleware

Dado cualquier acción en sus controladores (por ejemplo, index) y las aplicaciones de un middleware de terceros (por ejemplo MyMiddleware que convierte la salida a mayúsculas, puede utilizar un decorador web2py aplicar el middleware para esa acción. He aquí un ejemplo:

```

1 class MyMiddleware:
2     """converts output to upper case"""
3     def __init__(self,app):
4         self.app = app
5     def __call__(self,environ, start_response):
6         items = self.app(environ, start_response)
7         return [item.upper() for item in items]
8
9 @request.wsgi.middleware(MyMiddleware)
10 def index():

```

```
11     return 'hello world'
```

No podemos prometer que todo el middleware de terceros trabajará con este mecanismo.

Llamar a aplicaciones WSGI

Es fácil llamar a la aplicación de una acción WSGI web2py. Aquí está un ejemplo:

```
1 def test_wsgi_app(environ, start_response):
2     """this is a test WSGI app"""
3     status = '200 OK'
4     response_headers = [('Content-type','text/plain'),
5                          ('Content-Length','13')]
6     start_response(status, response_headers)
7     return ['hello world!\n']
8
9 def index():
10    """a test action that call the previous app and escapes output"""
11    items = test_wsgi_app(request.wsgi.environ,
12                          request.wsgi.start_response)
13    for item in items:
14        response.write(item,escape=False)
15    return response.body.getvalue()
```

En este caso la acción `index` `test_wsgi_app` llamadas y se escapa el valor devuelto antes de devolverla. Tenga en cuenta que `index` no es en sí una aplicación WSGI y debería utilizar la normal web2py API (por ejemplo, `response.write` escribir en el socket).

5

Las Vistas

web2py utiliza Python para sus modelos, controladores y puntos de vista, a pesar de que utiliza una sintaxis ligeramente modificada Python en los puntos de vista para permitir la lectura del código, sin imponer ninguna restricción sobre el correcto uso de Python.

El propósito de una opinión es que el código de inserción (Python) en un documento HTML. En general, esto plantea algunos problemas:

- ¿Cómo debe ser escapado código incrustado?
- En caso de sangrado se basa en Python o reglas HTML?

web2py {{ ... }} utiliza para escapar de código Python incrustado en HTML. La ventaja de usar llaves en lugar de paréntesis angulares es que es transparente para todos los editores de HTML comunes. Esto permite al desarrollador utilizar los editores para crear vistas web2py.

Puesto que el desarrollador es incrustación de código Python dentro de código HTML, el documento debe tener una sangría de acuerdo con las normas HTML, y no normas Python. Por lo tanto, se permite sin sangrar Python

dentro de las etiquetas {{ ... }}. Desde Python normalmente usa sangría para delimitar bloques de código, necesitamos una forma diferente a ellos delimitar, por eso la plantilla web2py lenguaje hace uso de la palabra clave de `pass` Python.

Un bloque de código comienza con una línea que termina con dos puntos y termina con una línea que comienza con la pass. El pass palabra clave no es necesario cuando el final de la manzana es evidente por el contexto.

Aquí está un ejemplo:

```

1 {{{
2 if i == 0:
3     response.write('i is 0')
4 else:
5     response.write('i is not 0')
6 pass
7 }}}

```

Tenga en cuenta que `pass` es una palabra clave de Python, no una palabra clave web2py. Algunos editores de Python, como Emacs, el uso de la palabra clave `pass` para significar la división de bloques y lo utilizan para el código nuevo guión de forma automática.

El lenguaje del template web2py hace exactamente lo mismo. Cuando encuentra algo así como:

```

1 <html><body>
2 {{for x in range(10):}}{{=x}}hello<br />{{pass}}
3 </body></html>

```

que se traduce en un programa:

```

1 response.write("""<html><body>""", escape=False)
2 for x in range(10):
3     response.write(x)
4     response.write("""hello<br />""", escape=False)
5 response.write("""</body></html>""", escape=False)

```

`response.write` escribe en la `response.body`.

Cuando hay un error en una vista web2py, el informe de error muestra el código generado no vista, la visión real de como está escrito por el desarrollador. Esto ayuda al desarrollador a depurar el código, poniendo de relieve el código real que se ejecuta (que es algo que se puede depurar con un editor HTML o el inspector DOM del navegador).

También tenga en cuenta que:

```
1 {{=x}}
```

genera

```
1 response.write(x)
```

Las variables se inyecta en el HTML de esta manera se escapó de forma predeterminada. El escape se ignora si `x` XML es un objeto, aun cuando escape se establece en True.

Aquí está un ejemplo que introduce el ayudante `H1`:

```
1 {{=H1(i)}}
```

que se traduce a:

```
1 response.write(H1(i))
```

en la evaluación, el objeto `H1` y sus componentes son de forma recursiva en serie, se escapó y por escrito al cuerpo de la respuesta. Las etiquetas generadas por `H1` y el interior HTML no se escapan. Este mecanismo garantiza que todo el texto y sólo texto — aparece en la página web está siempre se escapaba, evitando así que las vulnerabilidades XSS. Al mismo tiempo, el código es simple y fácil de depurar.

El `response.write(obj, escape=True)` método toma dos argumentos, el objeto a ser escrito y si tiene que ser escapado (True establece por defecto). Si `obj`

.xml() tiene un método, se le llama y el resultado por escrito al cuerpo de la respuesta (el argumento se omite escape). De lo contrario, utiliza el método __str__ del objeto para serializar y, si el argumento de escape es True, se le escapa. Todos los objetos integrados auxiliares (`H1` en el ejemplo) son objetos que saben cómo serializar a través del método .xml().

Todo esto se hace de forma transparente. Usted nunca necesita (y nunca debe) llamar al método `response.write` explícitamente.

5.1 Sintaxis básica

La plantilla de web2py lenguaje soporta todas las estructuras de control de Python. Aquí les ofrecemos algunos ejemplos de cada uno de ellos. Se pueden anidar según la práctica habitual de programación.

5.1.1 de... en

En las plantillas se puede circular a través de cualquier objeto iterable:

```

1 {{items = ['a', 'b', 'c']}}
2 <ul>
3 {{for item in items:}}<li>{{=item}}</li>{{pass}}
4 </ul>
```

que produce:

```

1 <ul>
2 <li>a</li>
3 <li>b</li>
4 <li>c</li>
5 </ul>
```

Aquí `item` es cualquier objeto iterable como una lista de Python, tupla de Python, o filas de objetos, o cualquier objeto que se implementa como un

iterador. Los elementos que se muestran son primeramente serializada y escapó.

5.1.2 *mientras que*

Usted puede crear un bucle que utiliza la palabra clave al mismo tiempo:

```

1 {{k = 3}}
2 <ul>
3 {{while k > 0:}}<li>{{=k}} {{k = k - 1}}</li>{{pass}}
4 </ul>
```

que produce:

```

1 <ul>
2 <li>3</li>
3 <li>2</li>
4 <li>1</li>
5 </ul>
```

5.1.3 *si... elif... else*

Puede utilizar las cláusulas condicionales:

```

1 {{{
2 import random
3 k = random.randint(0, 100)
4 }}}
5 <h2>
6 {{=k}}
7 {{if k % 2:}}is odd{{else:}}is even{{pass}}
8 </h2>
```

que produce:

```

1 <h2>
2 45 is odd
3 </h2>
```

Puesto que es evidente que cierra el bloque `else if` en primer lugar, no hay necesidad de una declaración `pass`, y el uso de una sería incorrecto. Sin embargo, de forma explícita, debe cerrar el bloque con una `else pass`.

Recuerde que en Python "else if" `elif` está escrito como en el ejemplo siguiente:

```

1 {{{
2 import random
3 k = random.randint(0, 100)
4 }}
5 <h2>
6 {{=k}}
7 {{if k % 4 == 0:}}is divisible by 4
8 {{elif k % 2 == 0:}}is even
9 {{else:}}is odd
10 {{pass}}
11 </h2>
```

Produce:

```

1 <h2>
2 64 is divisible by 4
3 </h2>
```

5.1.4 intentar otra cosa... excepto..... finalmente

También es posible utilizar las declaraciones `try...except` en puntos de vista con una advertencia. Consideremos el siguiente ejemplo:

```

1 {{try:}}
2 Hello {{= 1 / 0}}
3 {{except:}}
4 division by zero
5 {{else:}}
6 no division by zero
7 {{finally}}
8 <br />
9 {{pass}}
```

Se producirá el siguiente resultado:

```

1 Hello
2 division by zero
3 <br />

```

Este ejemplo ilustra que toda la salida generada antes se produce una excepción es representado (incluyendo la salida anterior a la excepción) dentro del bloque try. "Hello" está escrito, ya que precede a la excepción.

5.1.5 def... regresar

El lenguaje del template web2py permite al desarrollador definir y ejecutar las funciones que pueden devolver cualquier objeto de Python o un texto / html cadena. Aquí consideramos dos ejemplos:

```

1 {{def itemize1(link): return LI(A(link, _href="http://" + link))}}
2 <ul>
3 {{=itemize1('www.google.com')}}
4 </ul>

```

produce el resultado siguiente:

```

1 <ul>
2 <li><a href="http://www.google.com">www.google.com</a></li>
3 </ul>

```

La función devuelve un objeto de ayuda `itemize1` que se inserta en el lugar donde se llama a la función.

Consideremos ahora el siguiente código:

```

1 {{def itemize2(link):}}
2 <li><a href="{{link}}>{{=link}}</a></li>
3 {{return}}
4 <ul>
5 {{itemize2('www.google.com')}}
6 </ul>

```

Se produce exactamente el mismo resultado que el anterior. En este caso, el `itemize2` función representa un trozo de HTML que se va a reemplazar la etiqueta web2py donde se llama a la función. Observe que no hay '=' en frente de la llamada a la `itemize2`, ya que la función no devuelve el texto, pero lo escribe directamente en la respuesta.

Hay una advertencia: las funciones definidas por el interior de una visión debe terminar con una sentencia `return`, o la sangría automática se producirá un error.

5.2 Ayudantes del HTML

Considere el siguiente código en una vista:

```
1 {{=DIV('this', 'is', 'a', 'test', _id='123', _class='myclass')}}
```

que se representa como:

```
1 <div id="123" class="myclass">thisisatest</div>
```

`DIV` es una clase de ayuda, es decir, algo que puede ser usado para construir HTML mediante programación. Corresponde a la etiqueta HTML `<div>`. argumentos posicionales se interpretan como objetos contenidos entre las etiquetas se abren y cierran. Nombrado argumentos que comienzan con un guión bajo son interpretadas como etiquetas HTML tag atributos (sin el guión bajo). Algunos ayudantes también tienen argumentos con nombre que no comienzan con guión bajo, estos argumentos se etiqueta específica.

El siguiente grupo de ayudantes:

A, B, BEAUTIFY, BODY, BR, CENTER, CODE, DIV, EM, EMBED, FIELDSET, FORM, H1, H2, H3, H4, H5, H6, HEAD, HR, HTML, I, IFRAME, IMG, INPUT, LABEL, LEGEND, LI, LINK, OL, UL, MARKMIN, MENU, META, OBJECT, ON, OPTION, P, PRE, SCRIPT, OPTGROUP, SELECT, SPAN, STYLE, TABLE, TAG, TD, TEXTAREA, TH, THEAD, TBODY, TFOOT, TITLE, TR, TT, URL, XHTML,

XML, xmlescape, embed64 se puede utilizar para construir expresiones complejas que a continuación se puede serializar a XML (49) (50). Por ejemplo:

```
1 {{=DIV(B(I("hello ", "<world>"))), _class="myclass")}}
```

se representa:

```
1 <div class="myclass"><b><i>hello &lt;world&gt;</i></b></div>
```

El ayudantes mecanismo web2py es más que un sistema para generar HTML sin concatenar cadenas. Proporciona una representación del lado del servidor del Modelo de Objetos de Documento (DOM).

"Los componentes pueden ser objetos de referencia a través de su posición, y actuar como ayudantes listas con respecto a sus componentes:

```
1 >>> a = DIV(SPAN('a', 'b'), 'c')
2 >>> print a
3 <div><span>ab</span>c</div>
4 >>> del a[1]
5 >>> a.append(B('x'))
6 >>> a[0][0] = 'y'
7 >>> print a
8 <div><span>yb</span><b>x</b></div>
```

Atributos de los ayudantes se puede hacer referencia por su nombre y actuar como ayudantes diccionarios con respecto a sus atributos:

```
1 >>> a = DIV(SPAN('a', 'b'), 'c')
2 >>> a['_class'] = 's'
3 >>> a[0]['_class'] = 't'
4 >>> print a
5 <div class="s"><span class="t">ab</span>c</div>
```

5.2.1 XML

XML es un objeto que se utiliza para encapsular texto que no debe escapar. El texto puede o no contener XML válido. Por ejemplo, podría contener JavaScript.

El texto de este ejemplo es de escape:

```
1 >>> print DIV("<b>hello</b>")
2 &lt;b&gt;hello&lt;/b&gt;
```

mediante el uso de XML puede prevenir que se escapan:

```
1 >>> print DIV(XML("<b>hello</b>"))
2 <b>hello</b>
```

A veces quieras hacer HTML almacenados en una variable, pero el HTML pueden contener etiquetas peligrosas tales como secuencias de comandos:

```
1 >>> print XML('<script>alert("unsafe!")</script>')
2 <script>alert("unsafe!")</script>
```

Un-escapó de entrada ejecutable como éste (por ejemplo, entró en el cuerpo de un comentario en un blog) es peligroso, porque puede ser usado para generar Cross Site Scripting (XSS) en contra de otros visitantes a la página.

El ayudante XML web2py puede desinfectar nuestro texto para evitar las inyecciones y escapar de todas las etiquetas, salvo las que se permita explícitamente. Aquí está un ejemplo:

```
1 >>> print XML('<script>alert("unsafe!")</script>', sanitize=True)
2 &lt;script&gt;alert("unsafe!")&lt;/script&gt;
```

Los constructores XML, por defecto, tenga en cuenta el contenido de algunas etiquetas y algunos de sus atributos de seguridad. Puede reemplazar los valores predeterminados utilizando los argumentos opcionales `permitted_tags` y `allowed_attributes`. Éstos son los valores por defecto de los argumentosopcionales de la ayuda XML.

```
1 XML(text, sanitize=False,
2     permitted_tags=['a', 'b', 'blockquote', 'br/', 'i', 'li',
3     'ol', 'ul', 'p', 'cite', 'code', 'pre', 'img/'],
4     allowed_attributes={'a': ['href', 'title'],
5     'img': ['src', 'alt'], 'blockquote': ['type']})
```

5.2.2 Construido en Ayudantes

A

Esta ayuda se utiliza para crear enlaces.

```
1 >>> print A('<click>', XML('<b>me</b>'),
2           _href='http://www.web2py.com')
3 <a href='http://www.web2py.com'>&lt;click&gt;<b>me</b></a>
```

B

Esta ayuda hace que su contenido en negrita.

```
1 >>> print B('<hello>', XML('<i>world</i>'), _class='test', _id=0)
2 <b id="0" class="test">&lt;hello&gt;<i>world</i></b>
```

CUERPO

Este ayudante hace que el cuerpo de una página.

```
1 >>> print BODY('<hello>', XML('<b>world</b>'), _bgcolor='red')
2 <body bgcolor="red">&lt;hello&gt;<b>world</b></body>
```

CENTRO

Este ayudante centra su contenido.

```
1 >>> print CENTER('<hello>', XML('<b>world</b>'),
2                  _class='test', _id=0)
3 <center id="0" class="test">&lt;hello&gt;<b>world</b></center>
```

CÓDIGO

Este ayudante realiza resaltado de sintaxis para Python, C, C + +, código HTML y web2py, y es preferible a PRE para listados de código. CODE también tiene la capacidad para crear vínculos con el web2py documentación de la API.

He aquí un ejemplo de destacar las secciones de código en Python.

```

1 >>> print CODE('print "hello"', language='python').xml()
2 <table><tr valign="top"><td style="width:40px; text-align: right;"><pre style=
3   font-size: 11px;
4   font-family: Bitstream Vera Sans Mono,monospace;
5   background-color: transparent;
6   margin: 0;
7   padding: 5px;
8   border: none;
9   background-color: #E0E0E0;
10  color: #A0A0A0;
11  ">1.</pre></td><td><pre style="
12    font-size: 11px;
13    font-family: Bitstream Vera Sans Mono,monospace;
14    background-color: transparent;
15    margin: 0;
16    padding: 5px;
17    border: none;
18    overflow: auto;
19  "><span style="color:#185369; font-weight: bold">print </span>
20  <span style="color: #FF9966">"hello"</span></pre></td></tr>
21 </table>
```

Aquí hay un ejemplo similar para

```

1 >>> print CODE(
2 >>>   '<html><body>{{=request.env.remote_add}}</body></html>',
3 >>>   language='html')
```



```

1 <table>...<code>...
2 <html><body>{{=request.env.remote_add}}</body></html>
3 ...</code>...</table>
```

HTML

Estos son los argumentos por defecto para el ayudante CODE:

```
>>> CODE("print 'hello world'", language='python', link=None, counter=1, styles={})
```

Los valores admitidos para el argumento se `language` "python", "html_plain", "c", "cpp", "web2py", and "html". The "html" language interprets {{ and }} tags as "web2py" code, while "html_plain" no lo hace.

Si un valor se especifica `link`, "/examples/global/vars/", web2py API references in the code are linked to documentation at the link URL. For example "request" would be linked to "/examples/global/vars/request". In the above example, the link URL is handled by the "var" action in the "global.py" controller that is distributed as part of the web2py "examples", por ejemplo la aplicación.

El argumento `counter` se utiliza para la numeración de líneas. Se puede establecer en cualquier de estos tres valores diferentes. Puede ser `None` sin números de línea, un valor numérico que especifica el número de salida, o una cadena. Si el contador se establece en una cadena, se interpreta como un sistema, y no hay números de línea.

DIV

Todos los asistentes además de `XML` se derivan de `DIV` y heredar sus métodos básicos.

```
>>> print DIV('<hello>', XML('<b>world</b>'), _class='test', _id=0)
<div id="0" class="test">&lt;hello&gt;<b>world</b></div>
```

EM

Destaca su contenido.

```
>>> print EM('<hello>', XML('<b>world</b>'), _class='test', _id=0)
<em id="0" class="test">&lt;hello&gt;<b>world</b></em>
```

FIELDSET

Esto se utiliza para crear un campo de entrada junto con su etiqueta.

```
>>> print FIELDSET('Height:', INPUT(_name='height'), _class='test')
<fieldset class="test">Height:<input name="height" /></fieldset>
```

FORMA

Este es uno de los ayudantes más importantes. En su forma simple, sólo

hace que una etiqueta de <form>...</form>, pero debido a los ayudantes son objetos y tienen conocimiento de lo que contienen, pueden procesar los formularios presentados (por ejemplo, realizar la validación de los campos). Esto se discutirá en detalle en el capítulo 7.

```
1 >>> print FORM(INPUT(_type='submit'), _action='', _method='post')
2 <form enctype="multipart/form-data" action="" method="post">
3   <input type="submit" /></form>
```

El "enctype" es "multipart/form-data" de forma predeterminada.

El constructor de una FORM, SQLFORM y, también pueden tomar un argumento especial llamada . Cuando el diccionario se pasa como , sus elementos se convierten en campos de entrada "hidden". Por ejemplo:

```
1 >>> print FORM(hidden=dict(a='b'))
2 <form enctype="multipart/form-data" action="" method="post">
3   <input value="b" type="hidden" name="a" /></form>
```

H1, H2, H3, H4, H5, H6

Estos ayudantes son para títulos de los párrafos y subtítulos:

```
1 >>> print H1('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <h1 id="0" class="test">&lt;hello&gt;<b>world</b></h1>
```

CABEZA

Para etiquetar la cabeza de una página HTML.

```
1 >>> print HEAD(TITLE('<hello>', XML('<b>world</b>')))
2 <head><title>&lt;hello&gt;<b>world</b></title></head>
```

HTML

Este ayudante es un poco diferente. Además de hacer las etiquetas <html>, que antepone la etiqueta con un doctype string (52; 53; 54).

```
1 >>> print HTML(BODY('<hello>', XML('<b>world</b>')))
2 <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
```

```

3      "http://www.w3.org/TR/html4/loose.dtd">
4  <html><body>&lt;hello&gt;<b>world</b></body></html>
```

El helper HTML también tiene algunos argumentos opcionales adicionales que tienen el defecto siguientes:

```
1 HTML(..., lang='en', doctype='transitional')
```

DOCTYPE donde puede ser "estricta", "de transición", "conjunto de marcos", "HTML5", o una cadena de DOCTYPE completo.

XHTML

XHTML es similar al HTML, pero crea un DOCTYPE XHTML en su lugar.

```
1 XHTML(..., lang='en', doctype='transitional', xmlns='http://www.w3.org/1999/xhtml')
```

DOCTYPE donde puede ser "estricta", "de transición", "conjunto de marcos", o una cadena de DOCTYPE completo.

ENTRADA

Crea una etiqueta de <input .../>. Una etiqueta de entrada no puede contener otras etiquetas, y está cerrado por /> en vez de >. La etiqueta de entrada tiene un atributo opcional _type que se pueden establecer para "text" (the default), "submit", "checkbox", or "radio".

```

1 >>> print INPUT(_name='test', _value='a')
2 <input value="a" name="test" />
```

También tiene un especial argumento opcional llamado "value", distinct from "_value". The latter sets the default value for the input field; the former sets its current value. For an input of type "text", los aumentos a la segunda:

```

1 >>> print INPUT(_name='test', _value='a', value='b')
2 <input value="b" name="test" />
```

Para la radio INPUT botones selectivamente establece el atributo "checked":

```

1 >>> for v in ['a', 'b', 'c']:
2     print INPUT(_type='radio', _name='test', _value=v, value='b'), v
3 <input value="a" type="radio" name="test" /> a
4 <input value="b" type="radio" checked="checked" name="test" /> b
5 <input value="c" type="radio" name="test" /> c

```

y lo mismo para casillas de verificación:

```

1 >>> print INPUT(_type='checkbox', _name='test', _value='a', value=True)
2 <input value="a" type="checkbox" checked="checked" name="test" />
3 >>> print INPUT(_type='checkbox', _name='test', _value='a', value=False)
4 <input value="a" type="checkbox" name="test" />

```

IFRAME

Esta ayuda incluye otra página web en la página actual. La url de la página se indica otro a través del atributo "_src".

```

1 >>> print IFRAME(_src='http://www.web2py.com')
2 <iframe src="http://www.web2py.com"></iframe>

```

ETIQUETA

Se utiliza para crear una etiqueta Etiqueta para un campo de entrada.

```

1 >>> print LABEL('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <label id="0" class="test">&lt;hello&gt;<b>world</b></label>

```

LI

Hace un elemento de la lista y debe estar contenido en una etiqueta de UL o OL.

```

1 >>> print LI('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <li id="0" class="test">&lt;hello&gt;<b>world</b></li>

```

LEYENDA

Se utiliza para crear una etiqueta de leyenda para un campo de un formulario.

```
1 >>> print LEGEND('Name', _for='somefield')
2 <legend for="somefield">Name</legend>
```

META

Para ser utilizado para la creación de etiquetas META en la cabeza HTML. Por ejemplo:

```
1 >>> print META(_name='security', _content='high')
2 <meta name="security" content="high" />
```

MARKMIN

Implementa la sintaxis markmin wiki. Convierte la entrada de texto en el HTML de salida de acuerdo a las normas markmin se describe en el ejemplo siguiente:

```
1 >>> print MARKMIN("this is **bold** or ''italic'' and this [[a link http://web2py.
     com]]")
2 <p>this is <b>bold</b> or <i>italic</i> and
3 this <a href="http://web2py.com">a link</a></p>
```

La sintaxis markmin se describe en este archivo que se incluye con web2py:

```
1 http://127.0.0.1:8000/examples/static/markmin.html
```

y algunos ejemplos se muestran en el capítulo 13 en el contexto de plugin_wiki que utiliza MARKMIN ampliamente.

OBJETO

Se utiliza para incrustar objetos (por ejemplo, un reproductor flash) en el código HTML.

```
1 >>> print OBJECT('<hello>', XML('<b>world</b>'),
2      _src='http://www.web2py.com')
3 <object src="http://www.web2py.com">&lt;hello&gt;<b>world</b></object>
```

OL

Está parado para la lista ordenada. La lista debe contener etiquetas LI. OL argumentos que no son objetos LI son automáticamente cerrados en las etiquetas

```
1 >>> print OL('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <ol id="0" class="test"><li>&lt;hello&gt;world</b></li></ol>
```

ON

Esto está aquí para dar compatibilidad y es simplemente un alias para True. Se utiliza exclusivamente para casillas de verificación y obsoleta desde True es más Pythonic.

```
1 >>> print INPUT(_type='checkbox', _name='test', _checked=ON)
2 <input checked="checked" type="checkbox" name="test" />
```

OPTGROUP

Permite múltiples opciones tro grupo en un SELECT y es útil para personalizar los campos usando CSS.

```
1 >>> print SELECT('a', OPTGROUP('b', 'c'))
2 <select>
3   <option value="a">a</option>
4   <optgroup>
5     <option value="b">b</option>
6     <option value="c">c</option>
7   </optgroup>
8 </select>
```

OPCIÓN

Esto sólo se debe utilizar como parte de un SELECT combinación de OPCIÓN.

```
1 >>> print OPTION('<hello>', XML('<b>world</b>'), _value='a')
2 <option value="a">&lt;hello&gt;world</b></option>
```

Al igual que en el caso de INPUT, web2py hace una distinción entre "_value" (the value of the OPTION), and "value" (the current value of the enclosing select). If they are equal, the option is "selected".

```

1 >>> print SELECT('a', 'b', value='b'):
2 <select>
3 <option value="a">a</option>
4 <option value="b" selected="selected">b</option>
5 </select>
```

P

Esto es para marcar un párrafo.

```

1 >>> print P('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <p id="0" class="test">&lt;hello&gt;<b>world</b></p>
```

PRE

Genera una etiqueta <pre>...</pre> para mostrar el texto preformateado. El ayudante CODE general, es preferible para los listados de código.

```

1 >>> print PRE('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <pre id="0" class="test">&lt;hello&gt;<b>world</b></pre>
```

SCRIPT

Se trata de incluir o enlazar una secuencia de comandos, como JavaScript. El contenido entre las etiquetas se representa como un comentario HTML, en beneficio de los viejos exploradores de verdad.

```

1 >>> print SCRIPT('alert("hello world");', _language='javascript')
2 <script language="javascript"><!--
3 alert("hello world");
4 //--></script>
```

SELECCIONAR

Hace una etiqueta de <select>...</select>. Esta se utiliza con la ayuda

OPTION. Estos argumentos SELECT que no son objetos OPTION se convierten automáticamente a las opciones.

```

1 >>> print SELECT('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <select id="0" class="test">
3   <option value="&lt;hello&gt;">&lt;hello&gt;</option>
4   <option value="&lt;b&gt;world&lt;/b&gt;"><b>world</b></option>
5 </select>
```

SPAN

Al igual que DIV pero que se utilizan para etiquetar en línea (en lugar de bloquear) los contenidos.

```

1 >>> print SPAN('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <span id="0" class="test">&lt;hello&gt;<b>world</b></span>
```

ESTILO

Similares a la escritura, sino que se utiliza para incluir o enlace código CSS. Aquí se incluye el CSS:

```

1 >>> print STYLE(XML('body {color: white}'))
2 <style><!--
3 body { color: white }
4 //--></style>
```

y aquí está vinculado:

```

1 >>> print STYLE(_src='style.css')
2 <style src="style.css"><!--
3 //--></style>
```

TABLE, TR, TD

Estas etiquetas (junto con la THEAD opcional, TBODY y TF0OTER ayudantes) se utilizan para construir las tablas HTML.

```

1 >>> print TABLE(TR(TD('a'), TD('b')), TR(TD('c'), TD('d')))
2 <table><tr><td>a</td><td>b</td></tr><tr><td>c</td><td>d</td></tr></table>
```

TR espera que el contenido TD, argumentos que no son objetos TD se convierten automáticamente.

```
1 >>> print TABLE(TR('a', 'b'), TR('c', 'd'))
2 <table><tr><td>a</td><td>b</td></tr><tr><td>c</td><td>d</td></tr></table>
```

Es fácil convertir una matriz de Python en una tabla HTML con la función de Python * argumentos notación, que los mapas de la lista de elementos a los argumentos de la función de posición.

Aquí, vamos a hacerlo línea por línea:

```
1 >>> table = [['a', 'b'], ['c', 'd']]
2 >>> print TABLE(TR(*table[0]), TR(*table[1]))
3 <table><tr><td>a</td><td>b</td></tr><tr><td>c</td><td>d</td></tr></table>
```

Aquí lo que hacemos todas las líneas a la vez:

```
1 >>> table = [['a', 'b'], ['c', 'd']]
2 >>> print TABLE(*[TR(*rows) for rows in table])
3 <table><tr><td>a</td><td>b</td></tr><tr><td>c</td><td>d</td></tr></table>
```

TBODY

Esto se utiliza para etiquetar las filas contenidas en el cuerpo de la tabla, en lugar de filas de encabezado o pie de página. Es opcional.

```
1 >>> print TBODY(TR('<hello>'), _class='test', _id=0)
2 <tbody id="0" class="test"><tr><td>&lt;hello&gt;</td></tr></tbody>
```

TEXTAREA

Esta ayuda hace que una etiqueta de <textarea>...</textarea>.

```
1 >>> print TEXTAREA('<hello>', XML('<b>world</b>'), _class='test')
2 <textarea class="test" cols="40" rows="10">&lt;hello&gt;<b>world</b></textarea>
```

La única salvedad es que su "value" opcional reemplaza su contenido (los centros de HTML)

```

1 >>> print TEXTAREA(value=<hello world>, _class="test")
2 <textarea class="test" cols="40" rows="10">&lt;hello world&gt;</textarea>

```

TFOOT

Esta se utiliza para etiquetar pie de filas de la tabla.

```

1 >>> print TFOOT(TR(TD('<hello>'), _class='test', _id=0)
2 <tfoot id="0" class="test"><tr><td>&lt;hello&gt;</td></tr></tfoot>

```

TH

Esto se utiliza en lugar de TD en encabezados de tabla.

```

1 >>> print TH('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <th id="0" class="test">&lt;hello&gt;<b>world</b></th>

```

THEAD

Esto se utiliza para etiquetar encabezado filas de la tabla.

```

1 >>> print THEAD(TR(TD('<hello>'), _class='test', _id=0)
2 <thead id="0" class="test"><tr><td>&lt;hello&gt;</td></tr></thead>

```

TÍTULO

Esto se utiliza para etiquetar el título de una página en un encabezado HTML.

```

1 >>> print TITLE('<hello>', XML('<b>world</b>'))
2 <title>&lt;hello&gt;<b>world</b></title>

```

TR

Etiquetas un registro de la tabla. Cabe prestados dentro de una tabla y contienen etiquetas <td>...</td>. TR argumentos que no son objetos TD se convertirán automáticamente.

```

1 >>> print TR('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <tr id="0" class="test"><td>&lt;hello&gt;</td><td><b>world</b></td></tr>

```

TT

Etiquetas de texto como máquina de escribir (monoespaciado) de texto.

```
>>> print TT('<hello>', XML('<b>world</b>'), _class='test', _id=0)
<tt id="0" class="test">&lt;hello&gt;<b>world</b></tt>
```

UL

Significa una lista desordenada y debe contener elementos LI. Si su contenido no está etiquetado como LI, UL lo hace automáticamente.

```
>>> print UL('<hello>', XML('<b>world</b>'), _class='test', _id=0)
<ul id="0" class="test"><li>&lt;hello&gt;</li><li><b>world</b></li></ul>
```

Ayudantes del personal

A veces es necesario para generar código XML personalizado. web2py proporciona TAG, un generador de tag universal.

```
1 {{=TAG.name('a', 'b', _c='d')}}
```

genera el siguiente código XML

```
1 <name c="d">ab</name>
```

"a" and "b" and "d" Los argumentos son automáticamente escapados, el uso del auxiliar XML para suprimir este comportamiento. Usando TAG puede generar HTML / etiquetas XML no estuviera ya previsto por la API. Etiquetas pueden ser anidadas, y se serializan con str(). una sintaxis equivalente es:

```
1 {{=TAG['name']('a', 'b', c='d')}}
```

Tenga en cuenta que TAG es un objeto, y TAG.name o TAG['name'] es una función que devuelve una clase de ayuda temporal.

MENÚ

El ayudante MENÚ toma una lista de las listas de la forma `response.menu` (como se describe en el capítulo 4) y genera una estructura de árbol utilizando listas desordenadas que representa el menú. Por ejemplo:

```

1 >>> print MENU([['One', False, 'link1'], ['Two', False, 'link2']])
2 <ul class="web2py-menu web2py-menu-vertical">
3   <li><a href="link1">One</a></li>
4   <li><a href="link2">Two</a></li>
5 </ul>
```

Cada elemento del menú puede tener un cuarto argumento que es un submenú anidado (y así sucesivamente de forma recursiva):

```

1 >>> print MENU([['One', False, 'link1'], [['Two', False, 'link2']]])
2 <ul class="web2py-menu web2py-menu-vertical">
3   <li class="web2py-menu-expand">
4     <a href="link1">One</a>
5     <ul class="web2py-menu-vertical">
6       <li><a href="link2">Two</a></li>
7     </ul>
8   </li>
9 </ul>
```

El ayudante MENÚ toma los argumentos opcionales siguientes:

- `incumplimientos _class: "web2py-menu web2py-menu-vertical"` y establece que la clase de los elementos de UL exterior.
- `incumplimientos ul_class: "web2py-menu-vertical"` y establece que la clase de los elementos UL interior.
- `incumplimientos li_class: "web2py-menu-expand"` y establece que la clase de los elementos interiores LI.

El "base.css" de la solicitud comprende los andamios de base los siguientes tipos de menús: "web2py-menu web2py-menu-vertical" and "web2py-menu web2py-menu-horizontal".

5.3 EMBELLECER

`BEAUTIFY` se utiliza para construir representaciones de objetos HTML compuesto, como listas, tuplas y diccionarios:

```
1 {{=BEAUTIFY({ "a": ["hello", XML("world")], "b":(1, 2)})}}
```

`BEAUTIFY` devuelve un objeto XML como serializable a XML, con una bonita representación de su argumento del constructor. En este caso, la representación XML de:

```
1 {"a": ["hello", XML("world")], "b":(1, 2)}
```

hará como:

```
1 <table>
2 <tr><td>a</td><td>:</td><td>hello<br />world</td></tr>
3 <tr><td>b</td><td>:</td><td>1<br />2</td></tr>
4 </table>
```

5.4 Del lado del servidor DOM y Analizar

elementos

El ayudante DIV y todos los ayudantes derivados proporcionar un tres métodos de búsqueda: `element` y `elements`. `element` devuelve el primer elemento secundario se pongan en venta una condición especificada (o `None` si no hay coincidencia). `elements` devuelve una lista de todos los niños se pongan en venta.

`element` y `elements` utiliza la misma sintaxis para especificar la condición de concordancia que permite tres posibilidades que se pueden mezclar y emparejar: expresiones como jQuery, partido por valor de atributo exacta, coinciden con las expresiones regulares.

He aquí un ejemplo simple:

```

1 >>> a = DIV(DIV(DIV('a', _id='target',_class='abc')))
2 >>> d = a.elements('div#target')
3 >>> d[0] = 'changed'
4 >>> print a
5 <div><div><div id="target" class="abc">changed</div></div></div>

```

El argumento sin nombre de los elementos es una cadena que puede contener: el nombre de una etiqueta, el identificador de una etiqueta precedido por un símbolo de la libra, la clase precedido por un punto, el valor explícito de un atributo entre corchetes.

Aquí hay 4 formas equivalentes para buscar la etiqueta anterior por id:

```

1 >>> d = a.elements('#target')
2 >>> d = a.elements('div#target')
3 >>> d = a.elements('div[id=target]')
4 >>> d = a.elements('div',_id='target')

```

Aquí hay 4 formas equivalentes para buscar la etiqueta anterior por clase:

```

1 >>> d = a.elements('.abc')
2 >>> d = a.elements('div.abc')
3 >>> d = a.elements('div[class=abc]')
4 >>> d = a.elements('div',_class='abc')

```

Cualquier atributo puede ser usado para localizar un elemento (no sólo `id` y `class`), incluyendo varios atributos (el elemento de función puede tomar múltiples argumentos con nombre), pero sólo el elemento de primer apareamiento se devolverán.

Uso de la sintaxis de jQuery "div#target" es posible especificar criterios de búsqueda separados entre sí por un espacio:

```

1 >>> a = DIV(SPAN('a', _id='t1'),div('b',_class='c2'))
2 >>> d = a.elements('span#t1', div#c2')

```

o, equivalentemente,

```

1 >>> a = DIV(SPAN('a', _id='t1'),div('b',_class='c2'))
2 >>> d = a.elements('span#t1', 'div#c2')

```

Si el valor del atributo de búsqueda se especifica mediante un argumento de nombre, puede ser una cadena o una expresión regular:

```
1 >>> a = DIV(SPAN('a', _id='test123'),div('b',_class='c2'))
2 >>> d = a.elements('span',_id=re.compile('test\d{3}'))
```

Un argumento especial llamado de la DIV (y derivados) ayudantes es `find`. Se puede utilizar para especificar un valor de búsqueda o una búsqueda de expresiones regulares en el contenido de texto de la etiqueta. Por ejemplo:

```
1 >>> a = DIV(SPAN('abcde'),div('fghij'))
2 >>> d = a.elements(find='bcd')
3 >>> print d[0]
4 <span>abcde</span>
```

O

```
1 >>> a = DIV(SPAN('abcde'),div('fghij'))
2 >>> d = a.elements(find=re.compile('fg|w\{3\}'))
3 >>> print d[0]
4 <div>fghij</div>
```

padre

`parent` devuelve el parent del elemento actual.

```
1 >>> a = DIV(SPAN('a'),DIV('b'))
2 >>> d = a.element('a').parent()
3 >>> d['_class']='abc'
4 >>> print a
5 <div class="abc"><span>a</span><div>b</div></div>
```

aplanar

El método recursivo `aplanar` serializa el contenido de los hijos de un elemento dado en el texto normal (sin etiquetas):

```
1 >>> a = DIV(SPAN('this',DIV('is',B('a'))),SPAN('test'))
2 >>> print a.flatten()
3 thisisatest
```

Aplanar se puede pasar un argumento opcional, `render`, es decir, una función que hace / se aplana el contenido mediante un protocolo diferente. Este es un ejemplo para serializar algunas etiquetas en la sintaxis wiki Markmin:

```

1 >>> a = DIV(H1('title'),P('example of a ',A('link','_href="#test")))
2 >>> from gluon.html import markmin_serializer
3 >>> print a.flatten(render=markmin_serializer)
4 # titles
5
6 example of [[a link #test]]

```

En el momento de la escritura que ofrecemos `markmin_serializer` y `markdown_serializer`.
 ##### Analizar

El objeto de TAG también es un XML / HTML parser. Puede leer el texto y convertirlo en una estructura de árbol de ayudantes. Esto permite la manipulación con la API de arriba:

```

1 >>> html = '<h1>Title</h1><p>this is a <span>test</span></p>'
2 >>> parsed_html = TAG(html)
3 >>> parsed_html.element('span')[0]='TEST'
4 >>> print parsed_html
5 <h1>Title</h1><p>this is a <span>TEST</span></p>

```

5.5 Diseño de página

Las vistas se pueden ampliar e incluir otros puntos de vista en una estructura de árbol.

Por ejemplo podemos pensar en una "index.html" extends "layout.html" and includes "body.html" vista. Al mismo tiempo "layout.html" may include "header.html" and "footer.html".

La raíz del árbol es lo que llamamos una vista de diseño. Al igual que cualquier otro archivo de plantilla HTML, puede modificar utilizando la interfaz administrativa web2py. El "layout.html" nombre del archivo es sólo una convención.

Aquí hay una página minimalista que se extiende la opinión de "layout.html" view and includes the "page.html":

```
1 {{extend 'layout.html'}}
2 <h1>Hello World</h1>
3 {{include 'page.html'}}
```

El archivo de diseño extendida debe contener una directiva {{include}}, algo así como:

```
1 <html><head><title>Page Title</title></head>
2   <body>
3     {{include}}
4   </body>
5 </head>
```

Cuando la vista se le llama, la extendida (diseño) para ver se carga, y la vista llamando sustituye a la Directiva {{include}} en el interior del trazado. El proceso continúa de forma recursiva hasta que todos se extienden e incluyen directivas han sido procesados. La plantilla resultante se traduce en código Python.

extend y include son especiales plantilla de directivas no, los comandos de Python.

Diseños se utilizan para encapsular común página (encabezados, pies de página, menús), y aunque no son obligatorios, que hará que su aplicación más fácil de escribir y mantener. En particular, le sugerimos escribir diseños que se aprovechan de las siguientes variables que se pueden establecer en el controlador. Uso de variables bien conocidas éstas ayudarán a hacer sus diseños intercambiables:

```
1 response.title
2 response.subtitle
3 response.meta.author
4 response.meta.keywords
5 response.meta.description
6 response.flash
7 response.menu
8 response.files
```

A excepción de `menu` y `files` se trata de todas las cadenas y su significado debe ser obvia.

`response.menu` menú es una lista de 3-tuplas o 4 tuplas. Los tres elementos son: el nombre del vínculo, un booleano que representa si el enlace está activo (es el vínculo actual) y la URL de la página enlazada. Por ejemplo:

```
1 response.menu = [ ('Google', False, 'http://www.google.com', []),
2                   ('Index', True, URL('index'), [])]
```

el elemento de la tupla cuarto es un sub-menú opcional.

`response.files` es una lista de archivos CSS y JS que son necesarios para su página.

También te recomendamos que utilice:

```
1 {{include 'web2py_ajax.html'}}
```

HTML en la cabeza, ya que se incluirá la colección de jQuery y definir algunas funciones JavaScript compatibles con versiones anteriores para efectos especiales y Ajax. "web2py_ajax.html" `response.meta` incluye las etiquetas en la vista, la base de jQuery, el calendario de jQuery y CSS incluye todas las necesarias y JS `response.files`.

5.5.1 Diseño de página predeterminado

Aquí hay una mínima "views/layout.html" que viene con el **welcome** anadimos web2py aplicación, y cualquier nueva aplicación tendrá un diseño predeterminado similares:

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="{{=T.accepted_language or 'en'
4   ''}}">
5   <head>
```

```
6      <link rel="shortcut icon"
7          href="{{=URL(request.application,'static','favicon.ico')}}" type="image/vnd.microsoft.icon">
8      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
9
10     {{##### require CSS and JS files for this page (read info in base.css) }}
11     {{response.files.append(URL(request.application,'static','base.css'))}}
12     {{response.files.append(URL(request.application,'static','superfish.js'))}}
13
14     {{##### include web2py specific js code (jquery, calendar, form stuff) }}
15     {{include 'web2py_ajax.html'}}
16
17 </head>
18 <body>
19     <div class="flash">{{=response.flash or ''}}</div>
20     <div class="ez-mr wrapper" id="layout">
21
22         {{##### Layout 3 from http://www.ez-css.org/layouts }}
23         <div class="ez-wr">
24             <div class="ez-box" id="header">
25
26                 {{try:}}{{=auth.navbar(action=URL(request.application,'default','user'))}}{{
27                     except:pass}}
28                     <h1>
29                         <a href="">{{=response.title or 'response.title'}}</a>
30                     </h1>
31                     <h2>
32                         {{=response.subtitle or 'response.subtitle'}}
33                     </h2>
34             </div>
35             <div class="ez-box" id="statusbar">
36
37                 {{##### superfish menu }}
38                 {{=MENU(response.menu,_class='sf-menu')}}
```

```

54     </div>
55     <div class="ez-box" id="footer">
56         {{=T('Copyright')}} &#169; 2010 -
57         {{=T('Powered by')}} <a href="http://www.web2py.com">web2py</a>
58     </div>
59     </div>
60   </div>
61 </body>
62 </html>
```

Hay algunas características de este diseño por defecto que lo hacen muy fácil de usar y personalizar:

- {{#...}} son comentarios especiales que no van a aparecer en el código HTML de la página.
- Se muestra tanto `response.title` y `response.subtitle` que se puede establecer en un modelo. Si no se establecen, adopta el nombre de la aplicación como el título
- Se incluye el archivo `web2py_ajax.html` en el encabezado
- Se requiere de dos archivos de manera explícita: "base.css" and "superfish.js". El primero contiene la CSS para la página completa y está muy bien documentado y personalizable. Este último contiene las Juventudes Socialistas de la cascada por defecto del menú.
- El `{{=auth.navbar(...)}}` muestra una bienvenida al usuario actual y enlaces a las funciones de autenticación, como inicio de sesión, cierre de sesión, el registro, cambiar contraseña, etc dependiendo del contexto. Se coloca en una `{{try:}}...{{except:pass}}` autoridades en caso de que no está definido.
- El `{{=MENU(response.menu)}}` muestra la estructura del menú como `...`.
- Hay una secuencia de comandos explícitos para activar el menú en cascada Superfish y puede ser removido si no es necesario.
- `{{include}}` se sustituye por el contenido de la vista se extiende cuando se represente la página.
- Por defecto se utiliza un diseño de tres columnas de página y el uso de las siguientes identificaciones DIV: Los juegos de "header", "left_sidebar",

"content", and "right_sidebar", "footer" although the provided "base.css" los anchos de las barras laterales a cero.

- Se utiliza la convención de ez.css de diseño CSS nombres definidos en (51) ref. En particular, se utiliza el número de plantilla 3. El ez.css minimizada está incluido en "base.css".

5.5.2 Personalizar el diseño predeterminado

Personalizar el diseño predeterminado sin editar es muy fácil debido a el archivo "static/base.css" está muy bien documentada.

En particular, se organiza en los siguientes sub-secciones:

- ez.css
- restablecer etiquetas comunes
- elegir fuentes por defecto
- elegir estilo de vínculo
- añadir la línea de fondo a la mesa filas
- etiquetas de negrita y centrado en ocasiones
- hacer todos los campos de entrada del mismo tamaño
- añadir una separación adecuada entre h1-h6 y texto
- siempre sangrar la primera línea y agregar espacio a continuación los párrafos
- balas y el estilo de los números y guión
- forma y el relleno de la tabla
- bloques de código
- izquierdo y derecho de relleno texto citado
- diseño de la página de alineación, ancho y relleno (cambiar esta situación para los espacios)

- ancho de las columnas (cambiar esta opción para utilizar left_sidebar y right_sidebar)
- imágenes de fondo y cambiar los colores (esto por los colores)
- estilo de menú (por superfish.js)
- web2py específicos (. flash,. error)

Para cambiar el left_sidebar, el contenido, anchos right_sidebar, simplemente edita parte de "base.css":

```

1 ****柱子宽度 ****
2 #left_sidebar { width: 0px; }
3 #content { width: 840px; }
4 #right_sidebar { width: 0px; }
```

Para cambiar los colores e imágenes de fondo, simplemente edita la siguiente parte:

```

1 ***** 背景图像 和 颜色 ****
2 body { background: url('images/background.png') repeat-x #3A3A3A; }
3
4 a { color: #349C01; }
5 .auth_navbar {
6   top: 0px;
7   float: right;
8   padding: 3px 10px 3px 10px;
9   font-size: 0.9em;
10 }
11 code { color: green; background: black; }
12 input:focus, textarea:focus { background: #ccffcc; }
13 #layout { background: white; }
14 #header, #footer { color: white; background: url('images/header.png') repeat
15   #111111; }
16 #header h1 { color: #349C01; }
17 #header h2 { color: white; font-style: italic; font-size: 14px; }
18 #statusbar { background: #333333; border-bottom: 5px #349C01 solid; }
19 #statusbar a { color: white; }
20 #footer { border-top: 5px #349C01 solid; }
```

El menú está construido de una manera de color neutro pero se puede cambiar eso.

Por supuesto, también puede reemplazar completamente los archivos "layout.html" and "base.css" con el suyo.

5.6 Las funciones en Vista

Considere esto "layout.html":

```

1 <html>
2   <body>
3     {{include}} <!-- must come before the two blocks below -->
4     <div class="sidebar">
5       {{if 'mysidebar' in globals():}}{{mysidebar()}}{{else:}}
6         my default sidebar
7       {{pass}}
8     </body>
9   </html>
```

y esto se extiende vista

```

1 {{def mysidebar():}}
2 my new sidebar!!!
3 {{return}}
4 {{extend 'layout.html'}}
5 Hello World!!!
```

Notificación de la función definida antes de la instrucción {{extend...}}. Observe también la función se incluye en la vista ampliada sin el prefijo =.

El código genera el resultado siguiente:

```

1 <html>
2   <body>
3     Hello World!!!
4     <div class="sidebar">
5       {{block mysidebar}}
6         my new sidebar!!!
7       {{end}}
8     </body>
9   </html>
```

Observe que las funciones se definen en HTML (aunque también pueden contener código Python) de modo que `response.write` se utiliza para escribir su contenido (las funciones no devuelven el contenido). Esta es la razón por la disposición pide la opinión de la función utilizando en lugar de `{{{mysidebar()}}}` `==>{{=mysidebar()}}`. Las funciones definidas en esta forma puede llevar argumentos.

5.7 Bloques de Vista

Otra forma de hacer una visión más modular es mediante el uso de `{{block...}}`s, y este mecanismo es una alternativa al mecanismo discutido en la sección anterior.

Considere esto "layout.html":

```

1 <html>
2   <body>
3     {{include}} <!-- must come before the two blocks below -->
4     <div class="sidebar">
5       {{block mysidebar}}
6         my default sidebar
7       {{end}}
8     </body>
9   </html>
```

y esto se extiende vista

```

1 {{extend 'layout.html'}}
2 Hello World!!!
3 {{block mysidebar}}
4 my new sidebar!!!
5 {{end}}
```

Se genera el siguiente resultado:

```

1 <html>
2   <body>
3     Hello World!!!
4     <div class="sidebar">
```

```

5    {{block mysidebar}}
6        my new sidebar!!!
7    {{end}}
8    </body>
9 </html>
```

Usted puede tener muchos bloques y si el bloque A está presente la opinión extendida, pero no en la vista de la ampliación, el contenido de la vista extendida se usa.

5.8 Usando el sistema de plantillas para generar mensajes de correo electrónico

Es posible utilizar el sistema de plantillas para generar mensajes de correo electrónico. Por ejemplo, considere la tabla de base de datos de

```
1 db.define_table('person', Field('name'))
```

en la que desea enviar a cada persona en la base de datos el siguiente mensaje, almacena en un archivo de vista "message.html":

```

1 Dear {{=person.name}},
2 You have won the second prize, a set of steak knives.
```

Usted puede lograr esto en la siguiente forma

```

1 >>> from gluon.tools import Mail
2 >>> mail = Mail(globals())
3 >>> mail.settings.server = 'smtp.gmail.com:587'
4 >>> mail.settings.sender = '...@somewhere.com'
5 >>> mail.settings.login = None or 'username:password'
6 >>> for person in db(db.person.id>0).select():
7 >>>     context = dict(person=person)
8 >>>     message = response.render('message.html', context)
9 >>>     mail.send(to=['who@example.com'],
10 >>>                 subject='None',
11 >>>                 message=message)
```

La mayoría del trabajo se hace en la declaración de

```
1 response.render('message.html', context)
```

Vuelve el "file.html" with the variables defined in the dictionary "context" punto de vista, y devuelve una cadena con el texto del correo electrónico prestados. El contexto es un diccionario que contiene las variables que serán visibles para el archivo de plantilla.

Si el mensaje se inicia con <html> </html> y termina con el correo electrónico será un correo electrónico HTML.

El mismo mecanismo que se utiliza para generar texto correo electrónico también puede ser usado para generar SMS o cualquier otro tipo de mensaje basado en una plantilla.

6

La Capa de abstracción de base de datos

6.1 Dependencias

web2py viene con una capa de abstracción de base de datos (DAL), una API que los mapas Python objetos en objetos de base de datos como las consultas, tablas y registros. El DAL genera dinámicamente el SQL en tiempo real utilizando el dialecto especificado para la base de datos back-end, de modo que usted no tiene que escribir código SQL o aprender diferentes dialectos SQL (SQL se utiliza el término genérico), y la aplicación será portable entre diferentes tipos de bases de datos. En el momento de escribir estas líneas, las bases de datos soportadas son SQLite (que viene con Python y web2py así), PostgreSQL, MySQL, Oracle, MSSQL, Firebird, DB2, Informix, Ingres y y (parcialmente) el Google App Engine (GAE). JUEGO es tratado como un caso particular en el capítulo 11.

La distribución binaria de Windows funciona fuera de la caja con SQLite y MySQL. La distribución binaria Mac funciona fuera de la caja con SQLite. Para usar cualquier base de datos de otras de gama, de ejecución de la fuente de la distribución e instalar el controlador apropiado para la parte final requerida.

Una vez que el controlador adecuado está instalado, inicie web2py de la fuente, y que se encuentra el conductor. Aquí está una lista de los conductores:

database	driver (source)
SQLite	sqlite3 or pysqlite2 or zxJDBC (56) (on Jython)
PostgreSQL	psycopg2 (57) or zxJDBC (56) (on Jython)
MySQL	MySQLDb (58)
Oracle	cx_Oracle (59)
MSSQL	pyodbc (60)
FireBird	kinterbasdb (61)
DB2	pyodbc (60)
Informix	informixdb (62)
Ingres	ingresdbi (63)

web2py define las siguientes clases que componen el DAL:

DAL representa una conexión de base de datos. Por ejemplo:

```
1 db = DAL('sqlite://storage.db')
```

Table representa una tabla de base de datos. No es directamente una instancia de la tabla, sino que crea una instancia que **DAL.define_table**.

```
1 db.define_table('mytable', Field('myfield'))
```

Los métodos más importantes de una tabla son:

.insert, .truncate, .drop, .import_from_csv_file y.

Field representa un campo de base de datos. Se pueden crear instancias y pasa como un argumento para **DAL.define_table**.

DAL Rows es el objeto devuelto por una base de datos seleccione. Se puede considerar como una lista de filas **Row**:

```
1 rows = db(db.mytable.myfield!=None).select()
```

Row contiene valores de campo.

```

1 for row in rows:
2     print row.myfield

```

Query es un objeto que representa una cláusula de "where" SQL:

```

1 myquery = (db.mytable.myfield != None) | (db.mytable.myfield > 'A')

```

Set es un objeto que representa un conjunto de registros. Sus métodos son importantes `count`, `select`, `update`, `delete` y. Por ejemplo:

```

1 myset = db(myquery)
2 rows = myset.select()
3 myset.update(myfield='somevalue')
4 myset.delete()

```

Expression es algo así como una `orderby` o `groupby` un expresiones. La clase de campo se deriva de la expresión. He aquí un ejemplo.

```

1 myorder = db.mytable.myfield.upper() | db.mytable.id
2 db().select(db.table.ALL, orderby=myorder)

```

6.2 Cadenas de conexión

La conexión con la base de datos se establece mediante la creación de una instancia del objeto DAL:

```

1 >>> db = DAL('sqlite://storage.db', pool_size=0)

```

db no es una palabra clave, es una variable local que almacena el objeto de conexión DAL. Usted es libre de darle un nombre diferente. El constructor de DAL requiere un único argumento, la cadena de conexión. La cadena de conexión es el único código web2py que depende de una base de datos específica de servidor. Éstos son ejemplos de cadenas de conexión para tipos específicos de bases de datos de apoyo de back-end (en todos los casos, se supone la base de datos se está ejecutando desde localhost en el puerto por defecto y se llama "test"):

SQLite	sqlite://storage.db
MySQL	mysql://username:password@localhost/test
PostgreSQL	postgres://username:password@localhost/test
MSSQL	mssql://username:password@localhost/test
FireBird	firebird://username:password@localhost/test
Oracle	oracle://username:password@test
DB2	db2://username:password@test
Ingres	ingres://username:password@localhost/test
DB2	db2://username:password@test
Informix	informix://username:password@test
Google App Engine	gae

Observe que en la base de datos SQLite se compone de un solo archivo. Si no existe, se crea. Este archivo está bloqueado cada vez que se accede. En el caso de MySQL, PostgreSQL, MSSQL, Firebird, Oracle, DB2, Informix el "test" base de datos debe ser creado fuera web2py. Una vez establecida la conexión, se web2py crear, modificar, eliminar tablas apropiadamente.

También es posible establecer la cadena de conexión para `None`. En este caso DAL no se conectará a cualquier base de datos de back-end, pero el API todavía se puede acceder para su análisis. Ejemplos de esto se discutirá en el capítulo 7.

Agrupación de conexiones

El segundo argumento del constructor DAL es el `pool_size`, el valor predeterminado es 0.

Debido a que es lento para establecer una conexión nueva base de datos para cada solicitud. Para evitar esto, web2py implementa un mecanismo de agrupación de conexiones. Cuando se establece una conexión, después de que la página ha sido notificado y termine la transacción, la conexión no está cerrada, pero va en una piscina. Cuando la petición http siguiente llega, web2py trata de escoger una conexión de la agrupación y el uso que uno de una nueva transacción. Si no hay conexiones disponibles de la piscina, una nueva conexión está establecida.

El parámetro `pool_size` es ignorado por SQLite y la caza.

Conexiones en las piscinas son compartidos entre los hilos de forma secuencial, en el sentido de que puede ser utilizada por dos diferentes, pero no simultáneos. Sólo hay una piscina para cada proceso web2py.

Cuando se inicia web2py, la piscina está siempre vacío. La piscina crece hasta el mínimo entre el valor de `pool_size` y el número máximo de solicitudes simultáneas. Esto significa que si `pool_size=10`, pero nuestro servidor no recibe más de 5 solicitudes simultáneas, entonces el tamaño del grupo en sí sólo crecerá a 5. Si `pool_size=0` entonces la agrupación de conexiones no se utiliza.

La agrupación de conexiones se tiene en cuenta para SQLite, ya que no se traduce en una ventaja.

Fallos en la conexión

Si web2py no se puede conectar a la base de datos espera un segundo y lo intenta de nuevo hasta 5 veces antes de declarar un fracaso. En caso de agrupación de conexiones es posible que una conexión agrupada que permanece abierta, pero sin utilizar durante algún tiempo se cerraron a finales de bases de datos. Gracias a la función de reintento web2py intenta volver a establecer estas conexiones caídas.

Bases de datos replicadas

El primer argumento de `DAL(...)` puede ser una lista de URIs. En este caso web2py intenta conectarse a cada uno de ellos. El propósito principal de esto es tratar con los servidores de bases de datos múltiples y distribuir el trabajo entre ellos). He aquí un caso de uso típico:

```
1 db=DAL(['mysql://...1','mysql://...2','mysql://...3'])
```

En este caso, el DAL intenta conectarse a la primera y, en caso de fallo, que será el segundo y el primero. Esto también se puede utilizar distribuir la

carga en un maestro-esclavo de configuración de base de datos. Vamos a hablar más sobre esto en el capítulo 11 en el contexto de la escalabilidad.

6.3 Palabras clave reservadas

También hay otro argumento que se puede pasar al constructor DAL para comprobar los nombres de tabla y los nombres de columna con palabras clave reservadas de SQL en bases de datos de destino de back-end.

Este argumento es `check_reserved` y el valor predeterminado es Ninguno.

Esta es una lista de cadenas que contienen la base de datos los nombres de adaptadores de servidor.

El nombre del adaptador es el mismo que se utiliza en la cadena de conexión DAL. Así que si desea comprobar contra PostgreSQL y MSSQL a continuación, la cadena de conexión sería el siguiente:

```
1 db = DAL('sqlite://storage.db',
2           check_reserved=['postgres', 'mssql'])
```

El DAL explorará las palabras clave en el mismo orden de la lista.

Hay dos opciones adicionales "all" and "common". Si se especifica todo, comprobará contra todas las palabras clave de SQL conocida. Si especifica común, sólo se echa en contra comunes tales como palabras clave de SQL `SELECT`, `INSERT`, `UPDATE`, etc

Para los fines de soporte en la espalda también puede especificar si desea verificar contra las palabras clave de SQL no reservados, así. En este caso se anexa al nombre de `_nonreserved`. Por ejemplo:

```
1 check_reserved=['postgres', 'postgres_nonreserved']
```

Los backends de base de datos tras el apoyo palabras reservadas de cheques.

PostgreSQL	postgres(_nonreserved)
MySQL	mysql
FireBird	firebird(_nonreserved)
MSSQL	mssql
Oracle	oracle

6.4 DAL, de mesa, campo

La mejor manera de entender la API DAL es probar cada función manualmente. Esto se puede hacer de forma interactiva a través de la cáscara web2py, aunque en última instancia, el código va DAL en los modelos y controladores.

Comience por crear una conexión. Por el bien de ejemplo, puede utilizar SQLite. Nada en esta discusión los cambios al cambiar el motor de fondo.

```
1 >>> db = DAL('sqlite://storage.db')
```

La base de datos ya está conectado y la conexión se almacena en la variable global db.

En cualquier momento puede recuperar la cadena de conexión.

```
1 >>> print db._uri
2 sqlite://storage.db
```

y el nombre de base de datos

```
1 >>> print db._dbname
2 sqlite
```

La cadena de conexión se denomina _uri, ya que es una instancia de un identificador uniforme de recursos.

El DAL permite múltiples conexiones con la misma base de datos o con bases de datos diferentes, incluso bases de datos de diferentes tipos. Por ahora, vamos a suponer la presencia de una única base de datos ya que esta es la situación más común.

El método más importante de un DAL es `define_table`:

```
>>> db.define_table('person', Field('name'))
```

Se define, tiendas y devuelve un objeto llamado `Table "person"` containing a field (column) "name". Este objeto también se puede acceder a través de `db.person`, por lo que no es necesario coger el valor de retorno.

6.5 Registro de Representación

Es opcional pero recomendado para especificar un formato para la representación de los registros:

```
>>> db.define_table('person', Field('name'), format='%(name)s')
```

O

```
>>> db.define_table('person', Field('name'), format='%(name)s %(id)s')
```

o los más complejos, incluso utilizando una función:

```
>>> db.define_table('person', Field('name'),
2      format=lambda r: r.name or 'anonymous')
```

El atributo formato se utilizará para dos propósitos:

- Representar a los registros que se hace referencia en seleccionar la opción / desplegables.
- Para establecer el atributo `db.other_table.person.represent` para todos los campos referencia a esta tabla. Esto significa que no se mostrará SQLTABLE

referencias por id pero utilizará la representación formato preferido en su lugar.

Estos son los valores por defecto de un constructor de campo:

```

1 Field(name, 'string', length=None, default=None,
2      required=False, requires='<default>',
3      ondelete='CASCADE', notnull=False, unique=False,
4      uploadfield=True, widget=None, label=None, comment=None,
5      writable=True, readable=True, update=None, authorize=None,
6      autodelete=False, represent=None, compute=None, uploadseparate=None)

```

No todos ellos son relevantes para cada campo. "length" is relevant only for fields of type "string". "uploadfield" and "authorize" are relevant only for fields of type "upload". "ondelete" is relevant only for fields of type "reference" and "upload".

- `length` establece la longitud máxima de un campo de "string", "password" or "upload". Si no se especifica `length` un valor por defecto se utiliza el valor por defecto, pero no se garantiza que sea compatible con versiones anteriores. *To avoid unwanted migrations on upgrades, we recommend that you always specify the length for string, password and upload fields.*
- `default` establece el valor predeterminado para el campo. El valor predeterminado es utilizado para realizar una inserción si un valor no se especifica explícitamente. También se utiliza para llenar los formularios de pre-construido a partir de la tabla que usa SQLFORM.
- `required` cuenta la DAL que no se debe permitir insertar en esta tabla si el valor de este campo no se especifica explícitamente.
- `requires` es un validador o una lista de validadores. Esto no es utilizado por el DAL, pero es utilizado por SQLFORM. Los validadores por defecto para los tipos dados se muestran en la tabla siguiente:

field type	default field validators
string	IS_LENGTH(length)
blob	None
boolean	None
integer	IS_INT_IN_RANGE(-1e100, 1e100)
double	IS_FLOAT_IN_RANGE(-1e100, 1e100)
decimal(n,m)	IS_DECIMAL_IN_RANGE(-1e100, 1e100)
date	IS_DATE()
time	IS_TIME()
datetime	IS_DATETIME()
password	None
upload	None
reference	IS_IN_DB(db, referenced_id_field, referenced_table_format)

Decimal requiere y devuelve valores Decimal como objetos, tal como se define en el módulo de Python decimal. SQLite no controla el tipo decimal tan internamente la tratamos como una double. El (n, m) y el número de dígitos antes y después del punto decimal.

Observe que requires=... se aplica en el nivel de las formas, required=True se aplica en el nivel de LAD (insertar), mientras que notnull, unique y ondelete se aplican a nivel de la base de datos. Aunque a veces puede parecer redundante, es importante mantener la distinción cuando se programa con el DAL.

- Se traduce en la ondelete "ON DELETE" SQL statement. By default "CASCADE" cuenta la base de datos que cuando se elimina un registro, también deberá eliminar todos los registros que se refieren a ella.
- se traduce en la notnull=True "NOT NULL" instrucción SQL. Evita que la base de datos de la inserción de valores nulos para el campo.
- se traduce en la unique=True "UNIQUE" instrucción SQL y se asegura de que los valores de este campo son únicos dentro de la tabla. Se aplican a nivel de base de datos.
- uploadfield sólo se aplica a los campos de la carpeta "upload". A field of type "upload" stores the name of a file saved somewhere else, by default on the filesystem under the application "uploads/" tipo. Si uploadfield está definida, entonces el archivo se almacena en un campo blob en la misma

mesa y el valor de la `uploadfield` es el nombre del campo blob. Esto se discutirá en detalle más adelante en el contexto de SQLFORM.

- `uploadseparate` establece en True, si se subir archivos en diferentes subcarpetas de la carpeta "upload". Esto se ha optimizado para evitar demasiados archivos en la misma carpeta / subcarpeta. ATENCIÓN: No se puede cambiar el valor de la `uploadseparate` de True a False sin dañar al sistema. web2py bien utiliza las subcarpetas por separado o no. Cambiar el comportamiento después de los archivos se han subido evitará web2py de poder recuperar esos archivos. Si esto ocurre, es posible mover archivos y solucionar el problema pero esto no se describe aquí.
- `widget` debe ser uno de los objetos widget disponibles, incluidos los widgets personalizados, por ejemplo: `SQLFORM.widgets.string.widget`. Una lista de widgets disponibles se discutirá más adelante. Cada tipo de campo tiene un widget por defecto.
- `label` es una cadena (o algo que se puede serializar en una cadena) que contiene la etiqueta que se utilizará para este campo en las formas autogenerado.
- `comment` es una cadena (o algo que se puede serializar en una cadena) que contiene un comentario asociado a este campo, y se mostrará a la derecha del campo de entrada en las formas autogenerado.
- `writable` si un campo se puede escribir, puede ser editado en autogenerado crear y actualizar las formas.
- `readable` si un campo es legible, será visible en forma de solo lectura. Si un campo no es ni leer ni escribir, no se mostrará en crear y actualizar las formas.
- `update` contiene el valor predeterminado para este campo cuando el registro se actualiza.
- `compute` es una función opcional. Si se inserta un registro y no hay ningún valor para un campo que tiene un atributo de cálculo, el valor se calcula al pasar el registro (como un diccionario) para calcular la función.
- `authorize` se puede utilizar para requerir control de acceso en el campo correspondiente, para los campos "upload" solamente. Se discutirá con más detalle en el contexto de la autenticación y autorización.

- autodelete determina si el archivo subido correspondientes se suprimirá cuando la referencia a grabar el archivo es eliminado. Para los campos "upload" solamente.
- represent puede Ninguno o puede apuntar a una función que toma un valor de campo y devuelve una representación alternativa para el valor del campo. Ejemplos:

```

1 db.mytable.name.represent = lambda name: name.capitalize()
2 db.mytable.other_id.represent = lambda id: db.other(id).somefield
3 db.mytable.some_uploadfield.represent = lambda value: \
4     A('get it', _href=URL('download', args=value))

```

"blob" campos también son especiales. De forma predeterminada, los datos binarios codificados en base64 es antes de ser almacenado en el campo de base de datos real, y es decodificado cuando se extrae. Esto tiene el efecto negativo de la utilización de un 25% más espacio de almacenamiento de lo necesario en los campos blob, pero tiene dos ventajas. En promedio se reduce la cantidad de datos transmitidos entre el servidor y web2py base de datos y hacer la comunicación independientes de los convenios escapar de respaldo específico de usuarios finales.

Puede consultar la base de datos de las tablas existentes:

```

1 >>> print db.tables
2 ['person']

```

También puede consultar una tabla para los campos existentes:

```

1 >>> print db.person.fields
2 ['id', 'name']

```

No declarar un campo denominado "id", because one is created by web2py anyway. Every table has a field called "id" by default. It is an auto-increment integer field (starting at 1) used for cross-reference and for making every record unique, so "id" es una clave principal. (Nota: el id a partir de 1 ha vuelto específico fin. Por ejemplo, esto no se aplica al motor de Google App (GAE).)

Opcionalmente, puede definir un campo de type='id' y web2py utilizará este campo como el incremento de campo id-auto. Esto no es recomendable, excepto cuando se accede a tablas de bases de datos heredados. Con algunas limitaciones, también puede utilizar diferentes claves principales y esto se discute en la sección de "Legacy databases and keyed tables".

Usted puede consultar el tipo de una tabla:

```
1 >>> print type(db.person)
2 <class 'gluon.sql.Table'>
```

y puede acceder a una tabla desde la conexión con DAL:

```
1 >>> print type(db['person'])
2 <class 'gluon.sql.Table'>
```

Del mismo modo se puede acceder a los campos de su nombre en múltiples formas equivalentes:

```
1 >>> print type(db.person.name)
2 <class 'gluon.sql.Field'>
3 >>> print type(db.person['name'])
4 <class 'gluon.sql.Field'>
5 >>> print type(db['person']['name'])
6 <class 'gluon.sql.Field'>
```

Dado un campo, puede acceder a los atributos establecidos en su definición:

```
1 >>> print db.person.name.type
2 string
3 >>> print db.person.name.unique
4 False
5 >>> print db.person.name.notnull
6 False
7 >>> print db.person.name.length
8 32
```

incluyendo su tabla padre, nombre de tabla, y conexión a los padres:

```
1 >>> db.person.name._table == db.person
2 True
3 >>> db.person.name._tablename == 'person'
4 True
```

```
5 >>> db.person.name._db == db
6 True
```

6.6 *Migraciones*

controles `define_table` o no existe la tabla correspondiente. Si no lo hace, se genera el código SQL para crear y ejecuta el código SQL. Si la tabla no existe, pero difiere de la que se define, genera el código SQL para alterar la tabla y lo ejecuta. Si un campo ha cambiado el tipo pero no el nombre, que tratará de convertir los datos (Si no desea que esto, es necesario redefinir la tabla dos veces, la primera vez, dejando caer el web2py campo por su eliminación, y la segunda vez agregar el campo que acaba de definir de manera que web2py puede crearla.). Si la tabla existe y coincide con la definición actual, que lo dejen en paz. En todos los casos se creará el objeto `db.person` que representa la tabla.

Nos referimos a este comportamiento como una "migration". web2py logs all migrations and migration attempts in the file "databases/sql.log".

El primer argumento de la `define_table` es siempre el nombre de la tabla. Los argumentos no identificado otros son los campos (campo). La función también tiene un último argumento opcional llamado "migrate" que se debe a que se refiere explícitamente por su nombre como en:

```
>>> db.define_table('person', Field('name'), migrate='person.table')
```

El valor de migrar es el nombre del archivo (en los archivos de "databases" folder for the application) where web2py stores internal migration information for this table. These files are very important and should never be removed except when the entire database is dropped. In this case, the ".table" tienen que ser removidos manualmente. De forma predeterminada, la migración se establece en True. Web2py Esto hace que para generar el nombre del fichero de un hash de la cadena de conexión. Si la migración se establece

en `False`, el la migración no se realiza y web2py asume que la tabla existe en el almacén de datos y que contiene (al menos) a los puntos indicados en el `define_table`. La mejor práctica es dar un nombre explícito a la mesa de migrar.

No puede haber dos tablas en la misma solicitud con el mismo nombre de archivo de migrar.

6.7 *La fijación de Migraciones Roto*

Hay dos problemas comunes de las migraciones y hay formas de recuperarse de ellas.

Un problema es específico con SQLite. SQLite no hace cumplir tipos de columna y no se puede quitar columnas. Esto significa que si tiene una columna de tipo cadena y que lo quite, no es realmente eliminado. Si agrega la columna de nuevo con un tipo diferente (por ejemplo fecha y hora) al final tiene una columna datetime que contiene cadenas (basura, a efectos prácticos). web2py no se queja de esto porque no sabe lo que está en la base de datos, hasta que se intenta recuperar los registros y falla.

Si se vuelve a web2py un error en la función `gluon.sql.parse` cuando los registros de la selección, este es el problema: datos dañados en una columna debido a la cuestión anterior.

La solución consiste en la actualización de todos los registros de la tabla y actualización de los valores de la columna en cuestión con `ninguno`.

El otro problema es más genérico, pero típico con MySQL. MySQL no permite más de un `ALTER TABLE` en una transacción. Esto significa que hay que romper web2py transacciones complejas en otras más pequeñas (un `ALTER TABLE` en el momento) y se comprometan de una pieza en el momento. Por tanto, es posible que parte de una transacción compleja se compromete

y una parte no dejando web2py en un estado dañado. ¿Por qué parte de una transacción no? Porque, por ejemplo, se supone modificar una tabla y la conversión de una columna de cadena en una columna de fecha y hora, web2py intenta convertir los datos, pero los datos no se pueden convertir. ¿Qué sucede con web2py? Se pone confundido acerca de cuál es exactamente la estructura de la tabla almacenados en la base de datos.

La solución consiste en deshabilitar las migraciones para todas las tablas y permitir migraciones falsas:

```
1 db.define_table(...,migrate=False,fake_migrate=True)
```

Esto volverá a generar metadatos web2py alrededor de la mesa de acuerdo con la definición de la tabla. Pruebe varias definiciones mesa para ver cuál funciona (el que antes de la migración y el no una después de la migración no). Una vez conseguido quitar el atributo de `fake_migrate=True`.

Antes de intentar solucionar los problemas de migración que es prudente hacer una copia de "aplicaciones / yourapp / bases de datos / table *.sql" archivos.

6.8 *insertar*

Dada una tabla, puede insertar registros

```
1 >>> db.person.insert(name="Alex")
2 1
3 >>> db.person.insert(name="Bob")
4 2
```

Inserte devuelve el valor único de cada "id" registro insertado.

Usted puede truncar la tabla, es decir, eliminar todos los registros y restablecer el contador del ello.

```
1 >>> db.person.truncate()
```

Ahora, si inserta un registro nuevo, el contador vuelve al 1 (esto es de fondo específico y no se aplica a GAME):

```
1 >>> db.person.insert(name="Alex")
2 1
```

web2py también ofrece una

```
1 >>> db.person.bulk_insert([{'name':'Alex'}, {'name':'John'}, {'name':'Tim'}])
2 [3,4,5]
```

pies bulk_insert / método

Toma una lista de diccionarios de campos para la introducción y lleva a cabo inserciones de múltiples al mismo tiempo. Devuelve el ID de los registros insertados. En las bases de datos relacionales apoyo no hay ninguna ventaja en el uso de esta función en lugar de un bucle y realizar inserciones individuales, sino en Google App Engine, que hay una ventaja mayor velocidad.

6.9 ejecuta o deshace

No crean, borran, insertar, truncar, eliminar o actualizar operación es efectivamente comprometidos hasta que emita el comando commit

```
1 >>> db.commit()
```

Para comprobarlo vamos a insertar un nuevo registro:

```
1 >>> db.person.insert(name="Bob")
2 2
```

y hacer retroceder, es decir, ignora todas las operaciones desde la última confirmación:

```
1 >>> db.rollback()
```

Si ahora inserte de nuevo, el contador de nuevo se establece en 2, ya que la inserción anterior se deshizo.

```
1 >>> db.person.insert(name="Bob")
2
```

Código en los modelos, vistas y controladores se incluye en el código web2py que tiene este aspecto:

```
1 try:
2     execute models, controller function and view
3 except:
4     rollback all connections
5     log the traceback
6     send a ticket to the visitor
7 else:
8     commit all connections
9     save cookies, sessions and return the page
```

No hay necesidad de llamar siempre `commit` o `rollback` de forma explícita en web2py a menos que uno necesita un control más granular.

6.10 Raw SQL

ExecuteSQL

El DAL le permite emitir comandos SQL de forma explícita.

```
1 >>> print db.executesql('SELECT * FROM person; ')
2 [(1, u'Massimo'), (2, u'Massimo')]
```

En este caso, los valores de retorno no se analizan o transformado por la DAL, y el formato depende del controlador de base de datos específica. Este uso con selección normalmente no es necesario, pero es más común con los índices. `executesql` tiene dos argumentos opcionales: `placeholders` y `as_dict`. `placeholders` es una secuencia opcional de valores que deben sustituirse o, si es compatible con el controlador de DB, un diccionario con claves marcadores de posición se pongan en venta el nombre de su SQL.

Si `as_dict` se establece en `True`, y los resultados del cursor devuelto por el controlador de DB se convertirán en una secuencia de diccionarios vehículo con los nombres de campo db. Resultados regresó con `as_dict = True` are las mismas regresó al aplicar `.to_list()` a una normal de selección.

```
1 [{field1: value1, field2: value2}, {field1: value1b, field2: value2b}]
```

`_lastsql`

Si SQL fue ejecutado manualmente utilizando `ExecuteSQL` o fue SQL generado por el DAL, siempre se puede encontrar el código SQL en `db._lastsql`. Esto es útil para propósitos de depuración:

```
1 >>> rows = db().select(db.person.ALL)
2 >>> print db._lastsql
3 SELECT person.id, person.name FROM person;
```

*Nunca web2py genera consultas utilizando el " * " operador. web2py siempre es explícita cuando los campos de selección.*

6.11 caída

Por último, puedes colocar mesas y todos los datos se perderán:

```
1 >>> db.person.drop()
```

6.12 Índices

Actualmente la API DAL no proporciona un comando para crear índices en tablas, pero esto se puede hacer utilizando el comando `executesql`. Esto se debe a la existencia de índices puede hacer migraciones complejas, y es mejor

tratar con ellos de forma explícita. Los índices pueden ser necesarios para los campos que se utilizan en las consultas periódicas.

He aquí un ejemplo de cómo crear un índice usando SQL en SQLite:

```

1 >>> db = DAL('sqlite://storage.db')
2 >>> db.define_table('person', Field('name'))
3 >>> db.executesql('CREATE INDEX IF NOT EXISTS myidx ON person name;')
```

Otros dialectos base de datos tienen sintaxis muy similar, pero no puede apoyar la directiva "IF NOT EXISTS" opcional.

6.13 Legado de bases de datos y tablas con llave

web2py puede conectarse a bases de datos heredadas en ciertas condiciones.

La forma más fácil es cuando se cumplen estas condiciones:

- Cada tabla debe tener un único campo de incremento automático entero llamado "id"
- Los registros deben hacer referencia únicamente a partir de la esfera "id".

Al acceder a una tabla existente, es decir, un cuadro no creada por web2py en la aplicación actual, siempre se establece migrate=False.

Si la tabla tiene un legado de campo entero de incremento automático, pero no se llama "id", web2py aún puede acceder, pero la definición de tabla debe contener explícitamente como `Field('....','id')` que... es el nombre del campo entero de incremento automático.

Por último, si la tabla legado utiliza una clave principal que no es un incremento automático campo id es posible utilizar una "keyed table", por ejemplo:

```

1 db.define_table('account',
2     Field('accnum','integer'),
3     Field('acctype'),
4     Field('accdesc'),
5     primarykey=['accnum','acctype'],
6     migrate=False)

```

En este ejemplo el atributo PrimaryKey es una lista de campo que constituye la clave principal. En el momento de la escritura, no podemos garantizar que el atributo primarykey trabaja con todas las mesas de legado existentes y cada apoyado backend de base de datos. Para simplificar, se recomienda, si es posible, creando una vista de base de datos que tiene un campo id de incremento automático.

6.14 De transacciones distribuidas

En el momento de escribir esta función sólo es compatible con PostgreSQL, MySQL y Firebird, ya que exponen API para dos fases comete.

Asumiendo que tiene dos (o más) conexiones a distintas bases de datos PostgreSQL, por ejemplo:

```

1 db_a = DAL('postgres://... ')
2 db_b = DAL('postgres://... ')

```

En los modelos o los controladores, puede comprometer al mismo tiempo que:

```
1 DAL.distributed_transaction_commit(db_a, db_b)
```

En caso de fallo, esta función se deshace y plantea una `Exception`.

En los controladores, cuando una acción de retorno, si tiene dos conexiones distintas y usted no llama a la función anterior, web2py los compromete por

separado. Esto significa que hay una posibilidad de que uno de los comete tiene éxito y no una. La transacción distribuida impide que esto suceda.

6.15 Manual de Cargas

Considere el siguiente modelo:

```
1 db.define_table('myfile',Field('image','upload'))
```

Normalmente, la inserción se gestionan de forma automática a través de un SQLFORM o un formulario de porquería (que es un SQLFORM), pero de vez en cuando ya tienes el archivo en el sistema de archivos y quieres subirlo mediante programación. Esto puede hacerse de esta manera:

```
1 stream = open(filename,'rb')
2 db myfile.insert(image=db myfile.image.store(stream,filename))
```

El método de la `store` objeto de campo de carga tiene una secuencia de archivo y un nombre de archivo. Utiliza el nombre de archivo para determinar la extensión (tipo) del archivo, crea un nombre temporal para el archivo (siguiendo el mecanismo de web2py upload) y carga el contenido del archivo en este nuevo archivo temporal (bajo la carpeta de archivos a menos que se especifique lo contrario). Devuelve el nombre temporal nueva que se va a las tiendas en el campo de la tabla `image` `db myfile`.

6.16 Consulta, Set, Filas

Vamos a considerar de nuevo la tabla definida (y se dejó caer), previamente e inserte tres registros:

```
1 >>> db.define_table('person', Field('name'))
2 >>> db.person.insert(name="Alex")
3 1
4 >>> db.person.insert(name="Bob")
```

```

5 2
6 >>> db.person.insert(name="Carl")
7 3

```

Puede almacenar la tabla en una variable. Por ejemplo, con `person` variable, que puede hacer:

```

1 >>> person = db.person

```

También puede almacenar un campo en una variable como la `name`. Por ejemplo, también puedes hacer:

```

1 >>> name = person.name

```

Incluso puede crear una consulta (con operadores como `==`, `!=`, `<`, `>`, `<=`, `>=`, `Como`, `pertenece`) y guardar la consulta en una variable `q` como en:

```

1 >>> q = name=='Alex'

```

Cuando se llama a `db` con una consulta, se define un conjunto de registros. Se puede almacenar en una variable de `s` y escribir:

```

1 >>> s = db(q)

```

Observe que no hay ninguna consulta de base de datos se ha realizado hasta ahora. DAL + consulta simplemente definir un conjunto de registros en esta `db` que coinciden con la consulta. Web2py determina a partir de la consulta que el cuadro (o cuadros) están involucrados y, de hecho, no hay necesidad de precisar que.

6.17 seleccionar

Dado un conjunto, `s`, usted puede traer los registros con los `select` comando:

```

1 >>> rows = s.select()

```

Devuelve un objeto de iterable gluon.sql.Rows clase cuyos elementos son objetos de fila. gluon.sql.Row objetos actúan como diccionarios, pero sus elementos también se puede acceder como atributos, como el ex gluon.storage.Storage.The difieren de esta última debido a que sus valores son de sólo lectura.

Las filas de objetos permite recorrer el resultado de la selección e impresión de los valores de los campos seleccionados para cada fila:

```

1 >>> for row in rows:
2         print row.id, row.name
3 Alex

```

Usted puede hacer todos los pasos en una declaración:

```

1 >>> for row in db(db.person.name=='Alex').select():
2         print row.name
3 Alex

```

El comando de selección pueden incluir argumentos. Todos los argumentos sin nombre se interpretan como los nombres de los campos que desea buscar. Por ejemplo, puede ser explícito sobre obtener "id" and field "name" campo:

```

1 >>> for row in db().select(db.person.id, db.person.name):
2         print row.name
3 Alex
4 Bob
5 Carl

```

La tabla de atributos TODOS le permite especificar todos los campos:

```

1 >>> for row in db().select(db.person.ALL):
2         print row.name
3 Alex
4 Bob
5 Carl

```

Observe que no hay cadena de consulta pasa a db. web2py entiende que si desea que todos los campos de la tabla de la persona sin información adicional a continuación, desea que todos los registros de la persona mesa.

Una sintaxis alternativa equivalente es la siguiente:

```

1 >>> for row in db(db.person.id > 0).select():
2     print row.name
3 Alex
4 Bob
5 Carl

```

web2py y entiende que si usted pide para todos los registros de la persona tabla (`id>0`), sin información adicional, a continuación, desea que todos los ámbitos de la persona mesa.

Atajos

El DAL admite varios códigos, lo cual simplifica los accesos directos. En particular:

```

1 myrecord = db.mytable[id]

```

devuelve el registro con el `id` dar si es que existe. Si la `id` no existe, devuelve `None`. La declaración anterior es equivalente a

```

1 myrecord = db(db.mytable.id==id).select().first()

```

Puede eliminar los registros por id:

```

1 del db.mytable[id]

```

y esto es equivalente a

```

1 db(db.mytable.id==id).delete()

```

y elimina el registro con el `id` dado, si existe.

Usted puede insertar registros:

```

1 db.mytable[0] = dict(myfield='somevalue')

```

Es equivalente a

```

1 db.mytable.insert(myfield='somevalue')

```

y crea un nuevo registro con los valores de campo especificado por el diccionario en la mano derecha.

Usted puede actualizar los registros:

```
1 db.mytable[id] = dict(myfield='somevalue')
```

lo que equivale a

```
1 db(db.mytable.id==id).update(myfield='somevalue')
```

y actualiza un registro existente con los valores de campo especificado por el diccionario en la mano derecha.

Obtención de un registro

Sin embargo, otra sintaxis conveniente es el siguiente:

```
1 record = db.mytable(id)
2 record = db.mytable(db.mytable.id==id)
3 record = db.mytable(id,myfield='somevalue')
```

Al parecer similar a la sintaxis anterior `db.mytable[id]` es más flexible y segura. En primer lugar, comprueba si `id` es si no un entero (o `str(id)` es un `int`) y devuelve `None` (nunca se plantea una excepción). También permite especificar varias condiciones que el registro debe cumplir. Si no se cumplen, sino que también devuelve `None`.

Recursivas selecciona

Considere la persona tabla anterior y una nueva tabla "dog" referencing a "person":

```
1 >>> db.define_table('dog', Field('name'), Field('owner',db.person))
```

y un simple seleccionará de la tabla:

```
1 >>> dogs = db(db.dog.id>0).select()
```

Para cada fila de los perros es posible no sólo buscar campos de la tabla seleccionada (perro) pero también en relación a las tablas (recursivamente):

```
1 >>> for dog in dogs: print dog.info, dog.owner.name
```

Aquí `dog.owner.name` requiere una base de datos de seleccionar para cada perro en `perros` y por lo tanto, ineficiente. Le recomendamos que utilice siempre que sea posible se une en lugar de recursiva selecciona, no obstante, esto es conveniente y práctico cuando el acceso a los registros individuales.

También puede hacerlo al revés, mediante la selección de los perros que se hace referencia por una persona:

```
1 person = db.person(id)
2 for dog in person.dog.select(orderby=db.dog.name):
3     print person.name, 'owns', dog.name
```

En esta última expresiones `person.dog` es un atajo para

```
1 db(db.dog.owner==person.id)
```

es decir, el Conjunto de `dogs` referencia el `person` actual. Esta sintaxis se rompe si la tabla de referencia tiene múltiples referencia a la tabla referenciada. En este caso hay que ser más explícito y el uso de una consulta completa.

Seriación de filas en Vista

El resultado de una selección se pueden mostrar en una vista con la siguiente sintaxis:

```
1 {{extend 'layout.html'}}
2 <h1>Records</h2>
3 {{=db().select(db.person.ALL)}}
```

y se convierte automáticamente en una tabla HTML con un encabezado que contiene la columna de nombres y una fila por registro. Las filas se marcan como alterna "even" and class "odd" clase. Bajo el capó, las filas se convierte primero en un objeto SQLTABLE (que no debe confundirse con el cuadro) y

serializado entonces. Los valores extraídos de la base de datos también están formateados por los validadores, asociada al campo y se escapó entonces. (Nota: El uso de una db de esta manera en un punto de vista es por lo general no se considera una buena práctica MVC).

Sin embargo, es posible ya veces conveniente llamar SQLTABLE explícitamente.

El constructor SQLTABLE toma los argumentos opcionales siguientes:

- `linkto` la dirección URL o una acción que se utiliza para vincular los campos de referencia (por defecto en Ninguno)
- `upload` la dirección o la acción de descarga para permitir la descarga de los archivos cargados (por defecto en Ninguno)
- `headers` un campo de la cartografía diccionario de nombres para sus etiquetas que se utilizan como encabezados (por defecto para {}). También puede ser una instrucción. Actualmente soportamos `headers='fieldname:capitalize'`.
- `truncate` el número de caracteres para truncar los valores de largo en la tabla (por defecto es 16)
- `columns` la lista de nombres de campos que se muestran como columnas. Aquellos que no figuran no se muestran. (Por defecto a todos).
- ayudante de genéricos `**attributes` atributos que se pasa al objeto de mesa más externa.

Aquí está un ejemplo:

```

1 {{extend 'layout.html'}}
2 <h1>Records</h2>
3 {{=SQLTABLE(db().select(db.person.ALL),
4     headers='fieldname:capitalize',
5     truncate=100,
6     upload=URL('download'))}}
7 }}
```

orderby, groupby, limitby, distinto

El comando `select` lleva cinco argumentos opcionales: `orderby`, `groupby`, `limitby`, `izquierda` y `caché`. Aquí hablamos de los tres primeros.

Usted puede obtener los registros ordenados por nombre:

```

1 >>> for row in db().select(db.person.ALL, orderby=db.person.name):
2     print row.name
3 Alex
4 Bob
5 Carl

```

Usted puede obtener los registros ordenados por nombre en orden inverso (nótese la tilde):

```

1 >>> for row in db().select(db.person.ALL, orderby=db.person.name):
2     print row.name
3 Carl
4 Bob
5 Alex

```

Y usted puede ordenar los registros de acuerdo con varios campos mediante la concatenación de ellos con una "|":

```

1 >>> for row in db().select(db.person.ALL, orderby=db.person.name|db.person.id):
2     print row.name
3 Carl
4 Bob
5 Alex

```

Usando `groupby` junto con `orderby`, puede agrupar los registros con el mismo valor para el campo especificado (esto es backend específico, y no en el juego):

```

1 >>> for row in db().select(db.person.ALL, orderby=db.person.name,
2                             groupby=db.person.name):
3     print row.name
4 Alex
5 Bob
6 Carl

```

Con el `distinct=True` argumento, se puede especificar que sólo desea seleccionar registros distintos. Esto tiene el mismo efecto que la agrupación utilizando todos los campos especificados, excepto que no requiere clasificación.

Al utilizar distintas es importante no seleccionar TODOS los campos, en particular, no para seleccionar el campo "id", de lo contrario todos los registros serán siempre distintas.

Aquí está un ejemplo:

```

1 >>> for row in db().select(db.person.name, distinct=True):
2     print row.name
3 Alex
4 Bob
5 Carl

```

Con limitby, puede seleccionar un subconjunto de los registros (en este caso, los dos primeros a partir de cero):

```

1 >>> for row in db().select(db.person.ALL, limitby=(0, 2)):
2     print row.name
3 Alex
4 Bob

```

En la actualidad, "limitby" es sólo parcialmente compatible con MSSQL desde la base de datos Microsoft no proporciona un mecanismo para buscar un subconjunto de registros que no empiezan a 0.

Operadores lógicos

Las consultas se pueden combinar usando el operador binario y "&":

```

1 >>> rows = db((db.person.name=='Alex') | (db.person.id>3)).select()
2 >>> for row in rows: print row.id, row.name
3 4 Alex

```

y el operador OR binaria "|":

```

1 >>> rows = db((db.person.name=='Alex') | (db.person.id>3)).select()
2 >>> for row in rows: print row.id, row.name
3 1 Alex

```

Usted puede anular una consulta (o sub-query) con el operador binario !=:

```

1 >>> rows = db((db.person.name!='Alex') | (db.person.id>3)).select()

```

```

1 >>> for row in rows: print row.id, row.name
2 Bob
3 Carl

```

o por la negación explícita con el operador unario |:

```

1 >>> rows = db((db.person.name=='Alex') | (db.person.id>3)).select()
2 >>> for row in rows: print row.id, row.name
3 Bob
4 3 Carl

```

Debido a las restricciones en la sobrecarga de los operadores Python "AND" and "OR", estos no se pueden utilizar en la formación de consultas. Los operadores binarios se debe utilizar en su lugar.

contar, borrar, actualizar

Usted puede contar con los registros de un conjunto:

```

1 >>> print db(db.person.id > 0).count()
2 3

```

Puede eliminar los registros de un conjunto:

```

1 >>> db(db.person.id > 3).delete()

```

Y usted puede actualizar todos los registros de un conjunto al pasar argumentos con nombre que corresponde a los campos que necesitan ser actualizados:

```

1 >>> db(db.person.id > 3).update(name='Ken')

```

Expresiones

El valor asignado una instrucción de actualización puede ser una expresión. Por ejemplo, considerar este modelo

```

1 >>> db.define_table('person',
2     Field('name'),
3     Field('visits', 'integer', default=0))

```

```

4 >>> db(db.person.name == 'Massimo').update(
5     visits = db.person.visits + 1)

```

Los valores utilizados en las consultas también pueden ser expresiones

```

1 >>> db.define_table('person',
2     Field('name'),
3     Field('visits', 'integer', default=0),
4     Field('clicks', 'integer', default=0))
5 >>> db(db.person.visits == db.person.clicks + 1).delete()

```

update_record web2py también permite la actualización de un registro único que ya está en la memoria utilizando `update_record`

```

1 >>> rows = db(db.person.id > 2).select()
2 >>> row = rows[0]
3 >>> row.update_record(name='Curt')

```

Esto no debe confundirse con

```

1 >>> row.update(name='Curt')

```

porque para una sola fila, el método de actualizaciones `update` el objeto de fila, pero no el registro de la base, como en el caso de `update_record`.

primera y la última

Dado un objeto que contiene los registros Filas:

```

1 >>> rows = db(query).select()
2
3 >>> first_row = rows.first()
4 >>> last_row = rows.last()

```

son equivalentes a

```

1 >>> first_row = rows[0] if len(rows)>0 else None
2 >>> last_row = rows[-1] if len(rows)>0 else None

```

as_dict y as_list

Un objeto de fila se pueden serializar en un diccionario tradicional utilizando el método de `as_dict()` y filas de un objeto se puede serializar en una lista de diccionarios utilizando el método de `as_list()`. He aquí algunos ejemplos:

```
1 >>> rows = db(query).select()
2 >>> rows_list = rows.as_list()
3 >>> first_row_dict = rows.first().as_dict()
```

Estos métodos son convenientes para pasar a las filas y / o puntos de vista genérico para almacenar las filas en las sesiones (ya que los objetos Filas mismos no se puede serializar desde incluirán una referencia a una conexión abierta DB):

```
1 >>> rows = db(query).select()
2 >>> session.rows = rows # not allowed!
3 >>> session.rows = rows.as_list() # allowed!
```

encontrar, excluir, especie

Hay veces que uno necesita para llevar a cabo dos selecciona y una contiene un subconjunto de un anterior seleccione. En este caso no tiene sentido para acceder a la base de datos de nuevo. El `find`, `exclude` y `sort` objetos permiten manipular objetos filas y una generación de otra sin el acceso a la base de datos. Más en concreto:

- `find` devuelve un nuevo conjunto de filas filtradas por una condición y deja el original sin cambios.
- `exclude` devuelve un nuevo conjunto de filas filtradas por una condición y los elimina del Filas original.
- `sort` devuelve un nuevo conjunto de filas, ordenados por una condición y deja el original sin cambios.

Todos estos métodos tienen un solo argumento, una función que actúa sobre cada fila individual.

He aquí un ejemplo de uso:

```

1 >>> db.define_table('person',Field('name'))
2 >>> db.insert(name='John')
3 >>> db.insert(name='Max')
4 >>> db.insert(name='Alex')
5 >>> rows = db(db.person.id>0).select()
6 >>> for row in rows.find(lambda row: row.name[0]=='M'): print row.name
7 Max
8 >>> print len(rows)
9 3
10 >>> for row in rows.extract(lambda row: row.name[0]=='M'): print row.name
11 Max
12 >>> print len(rows)
13 2
14 >>> for row in rows.sort(lambda row: row.name): print row.name
15 Alex
16 John

```

Se pueden combinar:

```

1 >>> rows = db(db.person.id>0).select()
2 >>> rows = rows.find(lambda row: 'x' in row.name).sort(lambda row: row.name)
3 >>> for row in rows: print row.name
4 Alex
5 Max

```

6.18 Computarizada Campos

DAL campos pueden tener un atributo `compute`. Esta debe ser una función (o lambda) que toma un objeto de fila y devuelve un valor para el campo. Cuando un nuevo registro se inserta (o actualizado), si un valor para el campo no se proporciona, web2py intenta computar desde el campo otros valores utilizando la función de `compute`. Aquí está un ejemplo:

```

1 >>> db.define_table('item',
2     Field('unit_price','double'),
3     Field('quantity','integer'),
4     Field('total_price',compute=lambda r: r['unit_price']*r['quantity']))
5 >>> r = db.item.insert(unit_price=1.99, quantity=5)
6 >>> print r.total_price
7 9.95

```

Observe que el valor calculado se almacena en la db y no se calcula sobre la recuperación, como en el caso de los campos virtuales, que se describe más adelante. Dos aplicaciones típicas de los campos calculados son:

- en aplicaciones wiki, para almacenar la entrada procesada de texto como html wiki, para evitar la re-procesamiento en cada solicitud
- para buscar, para calcular los valores normalizados de un campo, que se utilizará para la búsqueda.

6.19 Campos Virtuales

campos virtuales son también campos calculados (como en el apartado anterior), pero difieren de las "virtual" porque son en el sentido de que no se almacenan en la db y que se calculan todos los registros de tiempo se extraen de la base de datos. que puede ser usado para simplificar el código del usuario sin utilizar sistemas de almacenamiento adicional, pero que no se puede utilizar para buscar.

Con el fin de definir uno o más campo virtual, usted tiene que definir una clase de contenedor, es una instancia y un vínculo a una tabla oa una selección. Por ejemplo, considere la siguiente tabla:

```

1 >>> db.define_table('item',
2                     Field('unit_price','double'),
3                     Field('quantity','integer'),
```

Uno puede definir un campo virtual precio total como

```

1 >>> class MyVirtualFields:
2         def total_price(self):
3             return self.item.unit_price*self.item.quantity
4
>>> db.item.virtualfields.append(MyVirtualFields())
```

Observe que cada método de la clase que toma un único argumento (auto) es un campo virtual. self se refiere a cada fila de uno de los selecciona-

dos. los valores de campo son referidos por la ruta completa como en `self.item.unit_price`. La tabla está vinculada a los campos virtual, añadiendo una instancia de la clase atribuir `virtualfields` de la tabla. campos virtuales también pueden acceder a los campos recursiva como en

```

1 >>> db.define_table('item',
2     Field('unit_price','double'))
3 >>> db.define_table('order_item',
4     Field('item',db.item),
5     Field('quantity','integer'))
6 >>> class MyVirtualFields:
7     def total_price(self):
8         return self.order_item.item.unit_price*self.order_item.quantity
9 >>> db.order_item.virtualfields.append(MyVirtualFields())

```

Observe el campo `self.order_item.item.unit_price` acceso recursivo, en el `self` es el registro de bucle.

También pueden actuar sobre el resultado de un JOIN

```

1 >>> db.define_table('item',
2     Field('unit_price','double'))
3 >>> db.define_table('order_item',
4     Field('item',db.item),
5     Field('quantity','integer'))
6 >>> rows = db(db.order_item.item==db.item.id).select()
7 >>> class MyVirtualFields:
8     def total_price(self):
9         return self.item.unit_price*self.order_item.quantity
10 >>> rows.setvirtualfields(order_item=MyVirtualFields())
11 >>> for row in rows: print row.order_item.total_price

```

Observe cómo en este caso la sintaxis es diferente. Los accesos campo virtual tanto `self.item.unit_price` y `self.order_item.quantity`, que pertenecen a la junta de selección. El campo virtual se une a las filas de la tabla mediante el método de las filas `setvirtualfields` objeto. Este método tiene un número arbitrario de argumentos con nombre y puede ser usado para fijar varios campos virtual, definido en múltiples clases, que se adjuntarán a las tablas múltiples:

```

1 >>> class MyVirtualFields1:
2     def discounted_unit_price(self):
3         return self.item.unit_price*0.90

```

```

4 >>> class MyVirtualFields2:
5     def total_price(self):
6         return self.item.unit_price*self.order_item.quantity
7     def discounted_total_price(self):
8         return self.item.discounted_unit_price*self.order_item.quantity
9 >>> rows.setvirtualfields(item=MyVirtualFields1(),order_item=MyVirtualFields2())
10 >>> for row in rows: print row.order_item.discounted_total_price

```

campos virtuales pueden ser perezoso, todo lo que necesitan hacer es devolver una función y se puede acceder llamando a la función:

```

1 >>> db.define_table('item',
2                     Field('unit_price','double'),
3                     Field('quantity','integer'),
4 >>> class MyVirtualFields:
5     def lazy_total_price(self):
6         def lazy(self=self):
7             return self.item.unit_price*self.item.quantity
8         return lazy
9 >>> db.item.virtualfields.append(MyVirtualFields())
10 >>> for item in db(db.item.id>0).select(): print item.lazy_total_price()

```

o menos usando una función lambda:

```

1 >>> class MyVirtualFields:
2     def lazy_total_price(self):
3         return lambda self=self: self.item.unit_price*self.item.quantity

```

6.20 Una relación de muchos

Para ilustrar la forma de aplicar una de muchas relaciones con la web2py DAL, definir otro "dog" that refers to the table "person" cuadro en el que volvamos a definir aquí:

```

1 >>> db.define_table('person',
2                     Field('name'),
3                     format='%(name)s')
4 >>> db.define_table('dog',
5                     Field('name'),
6                     Field('owner', db.person),
7                     format='%(name)s')

```

"dog" cuadro tiene dos campos, el nombre del perro y el dueño del perro. Cuando un tipo de campo es otra tabla, se pretende que la referencia sobre el terreno de la otra tabla por su ID. De hecho, usted puede imprimir el valor de tipo real y recibe:

```
1 >>> print db.dog.owner.type
2 reference person
```

Ahora, inserte tres perros, dos de propiedad de Alex y uno de Bob:

```
1 >>> db.dog.insert(name='Skipper', owner=1)
2 1
3 >>> db.dog.insert(name='Snoopy', owner=1)
4 2
5 >>> db.dog.insert(name='Puppy', owner=2)
6 3
```

Usted puede seleccionar como lo hizo para cualquier otra tabla:

```
1 >>> for row in db(db.dog.owner==1).select():
2     print row.name
3 Skipper
4 Snoopy
```

Debido a que un perro tiene una referencia a una persona, una persona puede tener muchos perros, por lo que un registro de persona mesa ahora adquiere un perro nuevo atributo, que es un conjunto, que define a los perros de esa persona. Esto permite recorrer todas las personas y traer a sus perros con facilidad:

```
1 >>> for person in db().select(db.person.ALL):
2         print person.name
3         for dog in person.dog.select():
4             print '    ', dog.name
5 Alex
6     Skipper
7     Snoopy
8 Bob
9     Puppy
10    Carl
```

Combinaciones internas

Otra forma de conseguir un resultado similar es el uso de una combinación, en concreto un INNER JOIN. web2py une realiza de forma automática y transparente cuando la consulta se unirán dos o más tablas como en el ejemplo siguiente:

```

1 >>> rows = db(db.person.id==db.dog.owner).select()
2 >>> for row in rows:
3     print row.person.name, 'has', row.dog.name
4 Alex has Skipper
5 Alex has Snoopy
6 Bob has Puppy

```

Observe que web2py hizo una combinación, por lo que las filas contienen ahora dos discos, uno de cada mesa, unidos entre sí. Debido a que los dos registros pueden tener campos con nombres en conflicto, tiene que especificar la tabla en la extracción del valor de campo de una fila. Esto significa que si antes se podía hacer:

```
1 row.name
```

y era obvio que si este era el nombre de una persona o un perro, en el resultado de una combinación que tiene que ser más explícito y decir:

```
1 row.person.name
```

O:

```
1 row.dog.name
```

Left Outer Join

Tenga en cuenta que Carl no aparecían en la lista anterior porque no tiene perros. Si tiene la intención de seleccionar a las personas (si tienen perros o no) y sus perros (si es que tienen alguno), entonces usted necesita para llevar a cabo un LEFT OUTER JOIN. Esto se hace utilizando el argumento de la "left" comando select. Aquí está un ejemplo:

```

1 >>> rows=db().select(db.person.ALL, db.dog.ALL, left=db.dog.on(db.person.id==db.dog
   .owner))
2 >>> for row in rows:

```

```

3     print row.person.name, 'has', row.dog.name
4 Alex has Skipper
5 Alex has Snoopy
6 Bob has Puppy
7 Carl has None

```

donde:

```

1 left = db.dog.on(...

```

hace la izquierda consulta de combinación. Aquí el argumento de la `db.dog.on` es la condición necesaria para la unión (el mismo utilizado anteriormente para la combinación interna). En el caso de una combinación a la izquierda, es necesario ser explícitos acerca de los campos que seleccione.

Agrupar y Contar

Al hacer combinaciones, a veces quiere agrupar las filas de acuerdo a determinados criterios y contar con ellos. Por ejemplo, contar el número de perros de propiedad de cada persona. web2py permite esto también. En primer lugar, es necesario contar con un operador. En segundo lugar, desea unirse a la mesa de la persona a la tabla de perro por su dueño. En tercer lugar, desea seleccionar todas las filas (persona + perro), agruparlos por persona, y el recuento de ellos, mientras la agrupación:

```

1 >>> count = db.person.id.count()
2 >>> for row in db(db.person.id==db.dog.owner).select(db.person.name, count, groupby
   =db.person.id):
3     print row.person.name, row[count]
4 Alex 2
5 Bob 1

```

Aviso de que el operador cuenta (que es integrado) se utiliza como un campo. El único problema aquí está en cómo recuperar la información. Cada fila contiene claramente una persona y el conde, pero el recuento no es un campo de una persona ni es una tabla. Entonces, ¿dónde va? Se entra en el almacenamiento de objetos que representan el registro con una clave igual a la expresión de consulta en sí.

6.21 Muchos a muchos

En los ejemplos anteriores, nos permite un perro tener un titular, pero una persona puede tener muchos perros. ¿Qué pasa si Patrón era propiedad de Alex y Curt? Esto requiere una relación de muchos a muchos, y se realiza a través de una tabla intermedia que vincula a una persona a un perro a través de una relación de propiedad.

Aquí está cómo hacerlo:

```

1 >>> db.define_table('person',
2                     Field('name'))
3 >>> db.define_table('dog',
4                     Field('name'))
5 >>> db.define_table('ownership',
6                     Field('person', db.person),
7                     Field('dog', db.dog))

```

la relación de propiedad existentes ahora puede ser reescrita como:

```

1 >>> db.ownership.insert(person=1, dog=1) # Alex owns Skipper
2 >>> db.ownership.insert(person=1, dog=2) # Alex owns Snoopy
3 >>> db.ownership.insert(person=2, dog=3) # Bob owns Puppy

```

Ahora puede agregar la nueva relación que Curt co-propietario de Patrón:

```

1 >>> db.ownership.insert(person=3, dog=1) # Curt owns Skipper too

```

Porque ahora tiene una relación de tres vías entre las tablas, puede ser conveniente definir un nuevo conjunto en los que realizar las operaciones:

```

1 >>> persons_and_dogs = db((db.person.id==db.ownership.person) | (db.dog.id==db.
                           ownership.dog))

```

Ahora es fácil para seleccionar todas las personas y sus perros de la nueva serie:

```

1 >>> for row in persons_and_dogs.select():
2         print row.person.name, row.dog.name
3 Alex Skipper

```

```

4 Alex Snoopy
5 Bob Puppy
6 Curt Skipper

```

Del mismo modo, puedes buscar por todos los perros de propiedad de Alex:

```

1 >>> for row in persons_and_dogs(db.person.name=='Alex').select():
2     print row.dog.name
3 Skipper
4 Snoopy

```

y todos los propietarios de Patrón:

```

1 >>> for row in persons_and_dogs(db.dog.name=='Skipper').select():
2     print row.owner.name
3 Alex
4 Curt

```

Una alternativa más ligera a las relaciones Muchos Muchos 2 es una de marcado. Etiquetar es discutido en el contexto del validador IS_IN_DB. Etiquetado de obras, incluso en backends de base de datos que no admite JOIN como el Google App Engine.

6.22 Otros operadores

web2py tiene otros operadores que proporcionan una API para acceder a los operadores de SQL equivalente. Definamos otra tabla "log" para almacenar los eventos de seguridad, su marca de tiempo y la gravedad, donde la gravedad es un número entero.

```

1 >>> db.define_table('log',
2                     Field('event'),
3                     Field('timestamp', 'datetime'),
4                     Field('severity', 'integer'))

```

Al igual que antes, inserte algunos eventos, una "port scan", an "xss injection" and an "unauthorized login". Por el bien de el ejemplo, puede registrar los

eventos con la misma estampa, pero con diferente gravedad (1, 2, 3, respectivamente).

```

1 >>> import datetime
2 >>> now = datetime.datetime.now()
3 >>> print db.log.insert(event='port scan', timestamp=now, severity=1)
4 1
5 >>> print db.log.insert(event='xss injection', timestamp=now, severity=2)
6 2
7 >>> print db.log.insert(event='unauthorized login', timestamp=now, severity=3)
8 3

```

como, superior, inferior

Los campos tienen un operador como que se puede utilizar para que coincida con cadenas:

```

1 >>> for row in db(db.log.event.like('port%')).select():
2     print row.event
3 port scan

```

Aquí "port%" indicates a string starting with "port". The percent sign character, "%", is a wild-card character that means "any sequence of characters".

Del mismo modo, puede utilizar los métodos de `upper` y `lower` para convertir el valor del campo a mayúsculas o minúsculas, y también se puede combinar con el operador LIKE.

```

1 >>> for row in db(db.log.event.upper().like('PORT%')).select():
2     print row.event
3 port scan

```

año, mes, día, hora, minutos, segundos

Los campos de fecha y de fecha y hora que día, mes y año métodos. Los campos de fecha y hora y el tiempo han horas minutos y segundos métodos. Aquí está un ejemplo:

```

1 >>> for row in db(db.log.timestamp.year()==2009).select():
2     print row.event
3 port scan

```

```

4  xss injection
5  unauthorized login

```

pertenece

El SQL operador IN se realiza a través del método que pertenece devuelve true cuando el valor del campo pertenece al conjunto especificado (lista de tuplas):

```

1 >>> for row in db(db.log.severity.belongs((1, 2))).select():
2     print row.event
3 port scan
4 xss injection

```

El DAL también permite un anidadas seleccione como el argumento de la operadora pertenece. La única salvedad es que la selección anidada tiene que ser un `_select`, select no es una, y sólo un campo tiene que ser seleccionado de manera explícita, la que define el conjunto.

```

1 >>> bad_days = db(db.log.severity==3).select(db.log.timestamp)
2 >>> for row in db(db.log.timestamp.belongs(bad_days)).select():
3     print row.event
4 port scan
5 xss injection
6 unauthorized login

```

Anteriormente , que ha usado el operador cuenta para contar registros. Del mismo modo, puede utilizar el operador suma para agregar (suma) los valores de un campo específico de un grupo de registros. Al igual que en el caso de contar, el resultado de una suma se recupera a través del objeto de almacen:

```

1 >>> sum = db.log.severity.sum()
2 >>> print db().select(sum).first()[sum]
3 6

```

6.23 La generación de SQL primas

A veces es necesario para generar el SQL, pero no ejecutarlo. Esto es fácil de ver con web2py ya que cada comando que realiza la base de datos de E / S tiene un comando equivalente que no, y simplemente devuelve el SQL que habría sido ejecutado. Estos comandos tienen el mismo nombre y la sintaxis como los funcionales, sino que empiezan con un guion bajo:

Aquí está `_insert`

```
>>> print db.person._insert(name='Alex')
2 INSERT INTO person(name) VALUES ('Alex');
```

Aquí está `_count`

```
>>> print db(db.person.name=='Alex')._count()
2 SELECT count(*) FROM person WHERE person.name='Alex';
```

Aquí está `_select`

```
>>> print db(db.person.name=='Alex')._select()
2 SELECT person.id, person.name FROM person WHERE person.name='Alex';
```

Aquí está `_delete`

```
>>> print db(db.person.name=='Alex')._delete()
2 DELETE FROM person WHERE person.name='Alex';
```

Y finalmente, aquí está `_update`

```
>>> print db(db.person.name=='Alex')._update()
2 UPDATE person SET WHERE person.name='Alex';
```

Por otra parte usted puede utilizar siempre `db._lastsql` para devolver el código SQL más reciente, ya se ha ejecutado de forma manual utilizando `ExecuteSQL` o fue SQL generado por el DAL.

6.24 Exportación e importación de datos

CSV (un cuadro a la vez)

Cuando un objeto se DALRows convierte en una cadena es automáticamente serializado en formato CSV:

```
1 >>> rows = db(db.person.id==db.dog.owner).select()
2 >>> print rows
3 person.id,person.name,dog.id,dog.name,dog.owner
4 1,Alex,1,Skipper,1
5 1,Alex,2,Snoopy,1
6 2,Bob,3,Puppy,2
```

Usted puede serializar una sola tabla en formato CSV y lo almacena en un archivo "test.csv":

```
>>> open('test.csv', 'w').write(str(db(db.person.id).select()))
```

y usted puede leerlo de nuevo con:

```
>>> db.person.import_from_csv_file(open('test.csv', 'r'))
```

Cuando se importen, web2py busca los nombres de campo en el encabezado del CSV. En este ejemplo, se encuentra dos columnas: los campos de "person.id" and "person.name". It ignores the "person." prefix, and it ignores the "id". A continuación, todos los registros se agregan y se les asignará nuevos ID. Ambas operaciones se pueden realizar a través de la interfaz web appadmin.

CSV (todas las tablas a la vez)

En web2py, puede copia de seguridad / restaurar una base de datos con dos comandos:

Para exportar:

```
>>> db.export_to_csv_file(open('somefile.csv', 'wb'))
```

Para importar:

```
1 >>> db.import_from_csv_file(open('somefile.csv', 'rb'))
```

Este mecanismo se puede utilizar incluso si la base de datos de importación es de un tipo diferente de la base de datos de exportación. Los datos se almacenan en "somefile.csv" como un archivo CSV en el que cada tabla empieza con una línea que indica el nombre de tabla, y otra línea con los nombres de campos:

```
1 TABLE tablename
2 field1, field2, field3, ...
```

Dos mesas están separadas \r\n\r\n. El archivo debe terminar con la línea de

```
1 END
```

El archivo no incluye los archivos subidos si estas no se almacenan en la base de datos. En cualquier caso, es bastante fácil de comprimir la carpeta "uploads" por separado.

Cuando se importen, los nuevos registros se añadirán a la base de datos si no está vacío. En general, los nuevos registros importados no tendrá el mismo registro de identificación que el original (guardado), pero los registros web2py recuperará referencias para que no se rompen, incluso si los valores de id puede cambiar.

Si una tabla contiene un campo denominado "uuid", este campo se utiliza para identificar duplicados. Además, si un registro importado tiene el "uuid" mismo que un registro existente, el récord anterior se actualizará.

CSV y sincronización de base de datos remota

Considere el siguiente modelo:

```
1 db = DAL('sqlite:memory:')
2 db.define_table('person',
3     Field('name'),
```

```

4     format='%(name)s')
5 db.define_table('dog',
6     Field('owner', db.person),
7     Field('name'),
8     format='%(name)s')
9
10 if not db(db.person.id>0).count():
11     id = db.person.insert(name="Massimo")
12     db.dog.insert(owner=id, name="Snoopy")

```

Cada registro se identifica por un nombre de usuario y al que hace referencia dicho ID. Si usted tiene dos copias de la base de datos utilizada por las distintas instalaciones web2py, el ID es único sólo dentro de cada base de datos y no a través de las bases de datos. Este es un problema cuando la fusión de registros de diferentes bases de datos.

Con el fin de hacer un registro único de identificación en bases de datos, deben:

- tener un identificador único (UUID),
- tienen una marca de tiempo (para averiguar cuál es la más reciente si varias copias),
- referencia el UUID en lugar de Ello.

Esto se puede lograr sin modificar web2py. Esto es lo que hay que hacer:

1. Cambiar el modelo anterior a:

```

1 db.define_table('person',
2     Field('uuid', length=64, default=uuid.uuid4()),
3     Field('modified_on', 'datetime', default=now),
4     Field('name'),
5     format='%(name)s')
6
7 db.define_table('dog',
8     Field('uuid', length=64, default=uuid.uuid4()),
9     Field('modified_on', 'datetime', default=now),
10    Field('owner', length=64),
11    Field('name'),
12    format='%(name)s')

```

```

13 db.dog.owner.requires = IS_IN_DB(db,'person.uuid','%(name)s')
14
15
16 if not db(db.person.id).count():
17     id = uuid.uuid4()
18     db.person.insert(name="Massimo", uuid=id)
19     db.dog.insert(owner=id, name="Snoopy")

```

2. Crear una acción de controlador para exportar la base de datos:

```

1 def export():
2     s = StringIO.StringIO()
3     db.export_to_csv_file(s)
4     response.headers['Content-Type'] = 'text/csv'
5     return s.getvalue()

```

3. Crear una acción de controlador para importar una copia guardada de la base de datos y registros de sincronización:

```

1 def import_and_sync():
2     form = FORM(INPUT(_type='file', _name='data'), INPUT(_type='submit'))
3     if form.accepts(request.vars):
4         db.import_from_csv_file(form.vars.data.file,unique=False)
5         # for every table
6         for table in db.tables:
7             # for every uuid, delete all but the latest
8             items = db(db[table].id>0).select(db[table].id,
9                                                 db[table].uuid,
10                                                orderby=db[table].modified_on,
11                                                groupby=db[table].uuid)
12             for item in items:
13                 db((db[table].uuid==item.uuid)& \
14                     (db[table].id!=item.id)).delete()
15     return dict(form=form)

```

4. Crear un índice de forma manual para hacer la búsqueda mediante uuid más rápido.

Aviso de que los pasos 2 y 3 de trabajo para todos los modelos de base de datos, no son específicos para este ejemplo.

Alternativamente, puede utilizar XML-RPC para exportar / importar el archivo.

Si la referencia registros subido los ficheros, necesita exportar / importar el contenido de la carpeta de archivos. Observe que los archivos en él ya están etiquetados por UUID por lo que no necesita preocuparse por conflictos de nombres y referencias.

HTML / XML (un cuadro a la vez)

DALRows objetos también tienen un método de `xml` (como ayudantes) que serializa a XML / HTML:

```

1 >>> rows = db(db.person.id > 0).select()
2 >>> print rows.xml()
3 <table>
4   <thead>
5     <tr>
6       <th>person.id</th>
7       <th>person.name</th>
8       <th>dog.id</th>
9       <th>dog.name</th>
10      <th>dog.owner</th>
11    </tr>
12  </thead>
13  <tbody>
14    <tr class="even">
15      <td>1</td>
16      <td>Alex</td>
17      <td>1</td>
18      <td>Skipper</td>
19      <td>1</td>
20    </tr>
21    ...
22  </tbody>
23 </table>
```

Si necesita serializar el DALRows en cualquier otro formato XML con etiquetas personalizadas, usted puede hacer fácilmente que el uso del Asistente de etiquetas universal y la notación *:

```

1 >>> rows = db(db.person.id > 0).select()
2 >>> print TAG.result(*[TAG.row(*[TAG.field(r[f], _name=f) \
3           for f in db.person.fields]) for r in rows])
4 <result>
5   <row>
6     <field name="id">1</field>
```

```

7   <field name="name">Alex</field>
8 </row>
9 ...
10</result>
```

Representación de datos

La función `export_to_csv_file` acepta un argumento de palabra clave de la llamada `represent`. Cuando `True` se utilizará la función de columnas `represent`, mientras que exportar los datos en lugar de los datos en bruto.

La función también acepta un argumento de palabra clave llamado `colnames` que debería contener una lista de nombres de columna uno desea exportar. El valor por defecto a todas las columnas.

Ambos aceptan `export_to_csv_file` y `import_from_csv_file` argumentos clave que le indican al analizador CSV el formato para guardar / cargar los archivos:

- predeterminado `delimiter`: delimitador para separar los valores (',')
- `quotechar`: carácter que se utiliza para citar a los valores de cadena (por defecto para las comillas dobles)
- `quoting`: sistema de cotizaciones (`csv.QUOTE_MINIMAL` por defecto)

He aquí algunos ejemplo de uso:

```

1 >>> import csv
2 >>> db.export_to_csv_file(open('/tmp/test.txt', 'w'),
3     delimiter='|',
4     quotechar='"',
5     quoting=csv.QUOTE_NONNUMERIC)
```

¿Qué haría algo similar a

```
1 "hello"|35|"this is the text description"|"2009-03-03"
```

Para más información consultar la documentación oficial de Python `quoteall::cite`

6.25 Selecciona el almacenamiento en caché

El método de selección también tiene un argumento de caché, que por defecto es Ninguno. Para efectos de almacenamiento en caché, se debe establecer en una tupla donde el primer elemento es el modelo de caché (cache.ram, cache.disk, etc), y el segundo elemento es el tiempo de expiración en segundos.

En el ejemplo siguiente, se ve un controlador que almacena una selección sobre la mesa previamente db.log definido. El real se vende seleccionar datos de la base de datos de back-end con más frecuencia de una vez cada 60 segundos y almacena el resultado en cache.ram. Si la siguiente llamada a este controlador se produce en menos de 60 segundos desde la última base de datos de E / S, sino que simplemente recupera los datos previos de cache.ram.

```
1 def cache_db_select():
2     logs = db().select(db.log.ALL, cache=(cache.ram, 60))
3     return dict(logs=logs)
```

Los resultados de una selección son complejos, pickleable objetos-sin, no pueden ser almacenados en una sesión y no puede ser almacenado en caché en modo alguno que no sea la que se explicó aquí.

6.26 Auto-Referencia y alias

Es posible definir tablas con campos que se refieren a sí mismos, aunque la notación habitual puede fallar. El código siguiente sería un error, ya que utiliza una variable antes de db.person se define:

```
1 db.define_table('person',
2     Field('name'),
3     Field('father_id', db.person),
4     Field('mother_id', db.person))
```

La solución consiste en utilizar una notación alternativa

```

1 db.define_table('person',
2     Field('name'),
3     Field('father_id', 'reference person'),
4     Field('mother_id', 'reference person'))

```

De hecho `db.tablename` y "reference tablename" son tipos equivalentes campo.

Si el cuadro se refiere a sí mismo, entonces no es posible ejecutar un JOIN para seleccionar una persona y de sus padres sin el uso de la palabra clave de SQL "AS". Esto se logra en web2py utilizando el `with_alias`. Aquí está un ejemplo:

```

1 >>> Father = db.person.with_alias('father')
2 >>> Mother = db.person.with_alias('mother')
3 >>> db.person.insert(name='Massimo')
4 1
5 >>> db.person.insert(name='Claudia')
6 2
7 >>> db.person.insert(name='Marco', father_id=1, mother_id=2)
8 3
9 >>> rows = db().select(db.person.name, Father.name, Mother.name,
10      left=(Father.on(Father.id==db.person.father_id),
11          Mother.on(Mother.id==db.person.mother_id)))
12 >>> for row in rows:
13     print row.person.name, row.father.name, row.mother.name
14 Massimo None None
15 Claudia None None
16 Marco Massimo Claudia

```

Observe que hemos optado por hacer una distinción entre:

- "father_id": the field name used in the table "person";
- "father": el alias que desea utilizar para la tabla que hace referencia el campo de arriba, lo que se comunica a la base de datos;
- "Father": la variable utilizada por web2py para referirse a ese alias.

La diferencia es útil, y no hay nada malo en utilizar el mismo nombre para los tres de ellos:

```

1 db.define_table('person',
2     Field('name'),
3     Field('father', 'reference person'),
4     Field('mother', 'reference person'))
5 >>> father = db.person.with_alias('father')
6 >>> mother = db.person.with_alias('mother')
7 >>> db.person.insert(name='Massimo')
8 1
9 >>> db.person.insert(name='Claudia')
10 2
11 >>> db.person.insert(name='Marco', father=1, mother=2)
12 3
13 >>> rows = db().select(db.person.name, father.name, mother.name,
14     left=(father.on(father.id==db.person.father),
15         mother.on(mother.id==db.person.mother)))
16 >>> for row in rows:
17     print row.person.name, row.father.name, row.mother.name
18 Massimo None None
19 Claudia None None
20 Marco Massimo Claudia

```

Pero es importante tener clara la distinción con el fin de generar consultas correctas.

6.27 Cuadro Herencia

Es posible crear una tabla que contiene todos los campos de otra tabla. Basta con pasar la otra tabla, en lugar de una campo para `define_table`. Por ejemplo

```

1 db.define_table('person', Field('name'))
2
3 db.define_table('doctor', db.person, Field('specialization'))

```

También es posible definir un cuadro ficticio que no se almacena en una base de datos con el fin de reutilizarlo en múltiples otros lugares. Por ejemplo:

```

1 current_user_id = (auth.user and auth.user.id) or 0
2
3 timestamp = db.Table(db, 'timestamp_table',
4     Field('created_on', 'datetime', default=request.now),
5     Field('created_by', db.auth_user, default=current_user_id),

```

```
6     Field('updated_on', 'datetime', default=request.now),
7     Field('updated_by', db.auth_user, update=current_user_id))
8
9 db.define_table('payment', timestamp, Field('amount', 'double'))
```

En este ejemplo se asume que la autenticación web2py estándar está activado.

7

Los formularios y validadores

Hay cuatro maneras distintas de construir formularios en web2py:

- `FORM` proporciona una implementación de bajo nivel en términos de ayudantes HTML. Un objeto `FORM` se puede serializar en HTML y es consciente de los campos que contiene. Un objeto `FORM` sabe cómo validar los valores presentados formulario.
- `SQLFORM` proporciona una API de alto nivel para la creación de crear, actualizar y eliminar las formas de una tabla de base de datos existente.
- `SQLFORM.factory` es una capa de abstracción en la parte superior de la `SQLFORM` con el fin de aprovechar la generación de formas características, aunque no se presente una base de datos. Se genera una forma muy similar a la `SQLFORM` de la descripción de una mesa, pero sin la necesidad de crear la tabla de base de datos.
- métodos de `CRUD`. Estos son funcionalmente equivalentes a `SQLFORM` y se basan en `SQLFORM`, pero proporcionan una notación más compacta.

Todas estas formas son conscientes de sí mismos y, si la entrada no pasa la validación, se pueden modificar y agregar mensajes de error. Los formularios

pueden ser consultados para las variables y validado para los mensajes de error que se han generado por la validación.

Arbitraria de código HTML puede ser insertado en otro extraído de la forma con ayudantes.

7.1 FORMA

Consideremos como ejemplo una aplicación con el controlador **test** "default.py" siguientes:

```
1 def display_form():
2     return dict()
```

y la vista asociada "default/display_form.html":

```
1 {{extend 'layout.html'}}
2 <h2>Input form</h2>
3 <form enctype="multipart/form-data"
4     action="{{=URL()}}" method="post">
5 Your name:
6 <input name="name" />
7 <input type="submit" />
8 </form>
9 <h2>Submitted variables</h2>
10 {{=BEAUTIFY(request.vars)}}
```

Se trata de un formulario HTML que regularmente pide el nombre del usuario. Cuando llene el formulario y haga clic en el botón de enviar, el formulario de auto-estima, y el `request.vars.name` variable y su valor se muestra en la parte inferior.

Puede generar la misma forma con ayudantes. Esto se puede hacer en la vista o en la acción. Desde web2py procesado la forma en la acción, que está bien para definir la forma en la acción.

Aquí está el nuevo controlador:

```

1 def display_form():
2     form=FORM('Your name:', INPUT(_name='name'), INPUT(_type='submit'))
3     return dict(form=form)

```

y la vista asociada "default/display_form.html":

```

1 {{extend 'layout.html'}}
2 <h2>Input form</h2>
3 {{=form}}
4 <h2>Submitted variables</h2>
5 {{=BEAUTIFY(request.vars)}}

```

El código de la medida es equivalente al código anterior, pero la forma es generada por el {{=form}} declaración que serializa el objeto FORM.

Ahora añadimos un nivel de complejidad mediante la adición de la validación de formularios y procesamiento.

Cambiar el dispositivo de la siguiente manera:

```

1 def display_form():
2     form=FORM('Your name:',
3               INPUT(_name='name', requires=IS_NOT_EMPTY()),
4               INPUT(_type='submit'))
5     if form.accepts(request.vars, session):
6         response.flash = 'form accepted'
7     elif form.errors:
8         response.flash = 'form has errors'
9     else:
10        response.flash = 'please fill the form'
11     return dict(form=form)

```

y la vista asociada "default/display_form.html":

```

1 {{extend 'layout.html'}}
2 <h2>Input form</h2>
3 {{=form}}
4 <h2>Submitted variables</h2>
5 {{=BEAUTIFY(request.vars)}}
6 <h2>Accepted variables</h2>
7 {{=BEAUTIFY(form.vars)}}
8 <h2>Errors in form</h2>
9 {{=BEAUTIFY(form.errors)}}

```

Tenga en cuenta que:

- En la acción, hemos añadido el validador para la `requires=IS_NOT_EMPTY()` "name" campo de entrada.
- En la acción, hemos añadido una llamada a `form.accepts(...)`
- En la vista, se está imprimiendo `form.vars` y `form.errors`, así como la forma y el `request.vars`.

Todo el trabajo se realiza por el método del objeto `accepts form`. Se filtra el `request.vars` de acuerdo a los requisitos declarados (expresadas por los validadores). Tiendas `accepts` aquellas variables que se han validado en `form.vars`. Si un valor de campo no cumple con un requisito, el validador no devuelve un error y el error se almacena en `form.errors`. Ambos `form.vars` y `form.errors` gluon.storage.Storage son objetos similares a `request.vars`. El primero contiene los valores que la validación pasado, por ejemplo:

```
1 form.vars.name = "Max"
```

Este último contiene los errores, por ejemplo:

```
1 form.errors.name = "Cannot be empty!"
```

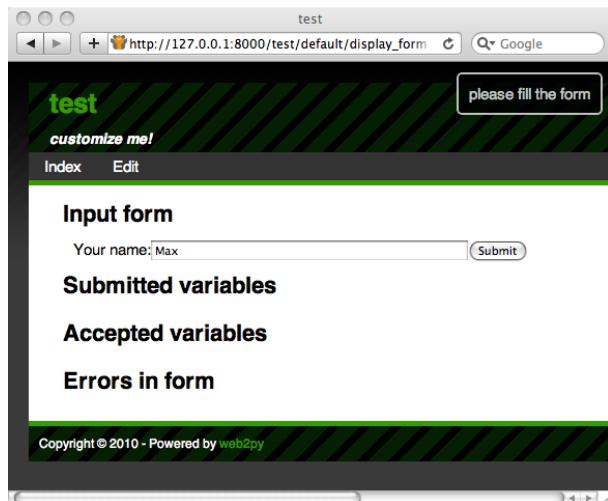
La firma definitiva del método de `accepts` es el siguiente:

```
1 form.accepts(vars, session=None, formname='default',
2               keepvalues=False, onvalidation=None,
3               dbio=True, hideerror=False):
```

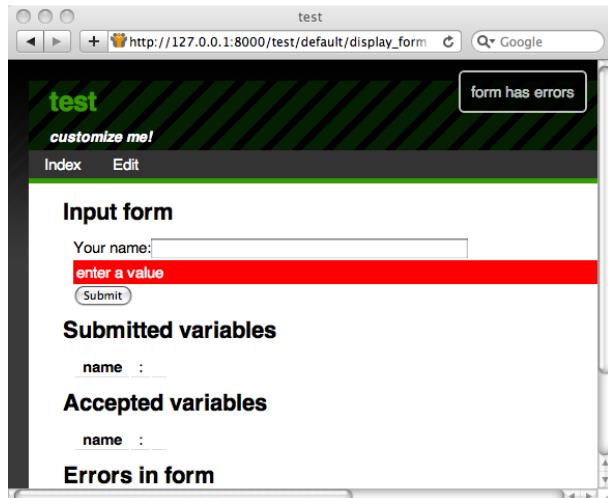
El significado de los parámetros opcionales se explica en la siguiente subsecciónes.

La función devuelve `accepts` True si el formulario es aceptado y lo contrario False. Un formulario no será aceptado si tiene errores o cuando no se ha presentado (por ejemplo, la primera vez que se muestra).

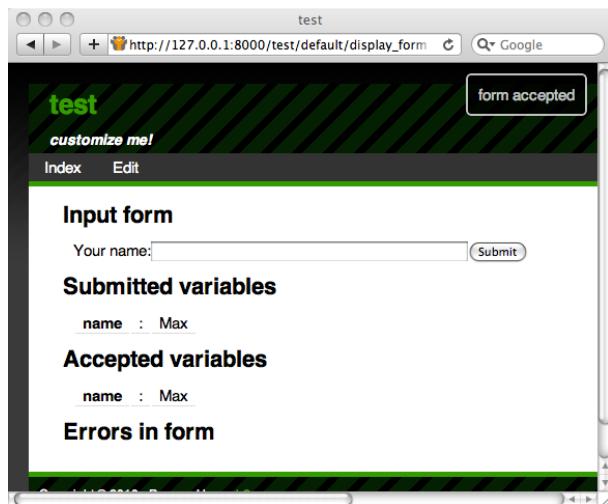
Así es como esta página es la primera vez que aparece:



Aquí está cómo se ve a la presentación nulas:



Aquí es como se ve en una sumisión válidas:



7.1.1 Ocultos campos

Cuando el objeto se serializa formulario de arriba de `{{=form}}`, y debido a la llamada anterior al método `accepts`, ahora se ve así:

```

1 <form enctype="multipart/form-data" action="" method="post">
2 your name:
3 <input name="name" />
4 <input type="submit" />
5 <input value="783531473471" type="hidden" name="_formkey" />
6 <input value="default" type="hidden" name="_formname" />
7 </form>
```

Nótese la presencia de dos campos ocultos: `"_formkey"` and `"_formname"`. Su presencia se ha disparado por la llamada a `accepts` y juegan dos roles diferentes e importantes:

- El campo oculto llamado `"_formkey"` es una muestra de una sola vez que web2py utiliza para prevenir la doble presentación de los formularios. El valor de esta clave se genera cuando el formulario se serializa y se almacena en la `session`. Cuando se envía el formulario debe coincidir con

este valor, o de lo contrario devuelve `accepts False` sin errores, como si el formulario no se presentó a todos. Esto se debe a web2py no puede determinar si el formulario se envió correctamente.

- `"_formname"` El campo oculto llamado es generado por web2py como un nombre para el formulario, pero el nombre se puede anular. Este campo es necesario para que las páginas que contienen y el proceso de múltiples formas. web2py distingue las diferentes formas presentadas por sus nombres.
- Opcional campos ocultos especificado como `FORM(...,hidden=dict(...))`.

El papel de estos campos ocultos y su utilización en los formularios personalizados y las páginas con distintas formas se analiza con más detalle más adelante en este capítulo.

Si lo anterior se envía el formulario con un campo vacío "name", la forma no pasa la validación. Cuando el formulario se serializa una vez más aparece como:

```

1 <form enctype="multipart/form-data" action="" method="post">
2 your name:
3 <input value="" name="name" />
4 <div class="error">cannot be empty!</div>
5 <input type="submit" />
6 <input value="783531473471" type="hidden" name="_formkey" />
7 <input value="default" type="hidden" name="_formname" />
8 </form>
```

Nótese la presencia de un DIV de "error" clase en la forma serializada. web2py inserta este mensaje de error en la forma de notificar al visitante sobre el campo que no pasó la validación. El método de `accepts`, a la entrega, determina que se envía el formulario, comprueba si el campo está vacío "name" y si es necesario, y finalmente inserta el mensaje de error del validador en el formulario.

El "layout.html" view is expected to handle DIVs of class "error" base. El diseño por defecto usa efectos jQuery para hacer aparecer errores y deslice hacia abajo con un fondo rojo. Consulte el Capítulo 10 para más detalles.

7.1.2 *keepvalues*

El `keepvalues` argumento opcional web2py dice qué hacer cuando una forma es aceptada y no hay cambio de dirección, por lo que la misma forma se muestra de nuevo. Por defecto está desactivada la forma. Si se establece en `keepvalues True`, el formulario se llenará con los valores previamente insertado. Esto es útil cuando se tiene un formulario que se supone que debe usarse repetidamente para insertar varios registros similares. Si el argumento `dbio` se establece en `False`, web2py no realizará ninguna parte móvil DB / actualizar después de aceptar la forma. Si se establece en `hideerror True` y el formulario contiene errores, éstos no se mostrarán cuando el formulario se representa (se depende de usted para mostrarlos `form.errors` de alguna manera. `onvalidation` El argumento se explica a continuación.

7.1.3 *onvalidation*

El argumento puede ser `onvalidation None` o puede ser una función que toma la forma y no devuelve nada. Tal función se llama y se pasa a la forma, inmediatamente después de la validación (si pasa la validación) y antes que nada sucede. El propósito de esta función es múltiple. Se puede utilizar, por ejemplo, para llevar a cabo controles adicionales sobre la forma y finalmente añadir los errores en el formulario. También se puede utilizar para calcular los valores de algunos campos basada en los valores de otros campos. Se puede utilizar para desencadenar alguna acción (como enviar un correo electrónico) antes de se crea un registro / actualización.

Aquí está un ejemplo:

```

1 db.define_table('numbers',
2     Field('a', 'integer'),
3     Field('b', 'integer'),
4     Field('c', 'integer', readable=False, writable=False))
5
6 def my_form_processing(form):
7     c = form.vars.a * form.vars.b
8     if c < 0:

```

```

9      form.errors.b = 'a*b cannot be negative'
10     else:
11         form.vars.c = c
12
13 def insert_numbers():
14     form = SQLFORM(db.numbers)
15     if form.accepts(request.vars, session,
16                     onvalidation=my_form_processing)
17         session.flash = 'record inserted'
18         redirect(URL())
19     return dict(form=form)

```

7.1.4 Formularios y redirección

La forma más común el uso de las formas es a través de auto-presentación, de modo que las variables de campo presentados son procesados por la misma acción que generó el formulario. Una vez que el formulario es aceptado, no es habitual para mostrar la página otra vez (algo que estamos haciendo aquí sólo para mantener las cosas simples). Es más común para redirigir al usuario a una página "next".

Aquí está el controlador nuevo ejemplo:

```

1 def display_form():
2     form = FORM('Your name:',
3                 INPUT(_name='name', requires=IS_NOT_EMPTY()),
4                 INPUT(_type='submit'))
5     if form.accepts(request.vars, session):
6         session.flash = 'form accepted'
7         redirect(URL('next'))
8     elif form.errors:
9         response.flash = 'form has errors'
10    else:
11        response.flash = 'please fill the form'
12    return dict(form=form)
13
14 def next():
15     return dict()

```

A fin de establecer un flash en la página siguiente en lugar de la página actual

se debe utilizar en vez de `session.flash` `response.flash`. web2py mueve la primera a la segunda después de la redirección. Tenga en cuenta que el uso `session.flash` requiere que usted no `session.forget()`.

7.1.5 Múltiples formas por página

El contenido de esta sección se aplica tanto a los `FORM` y `SQLFORM` objetos.

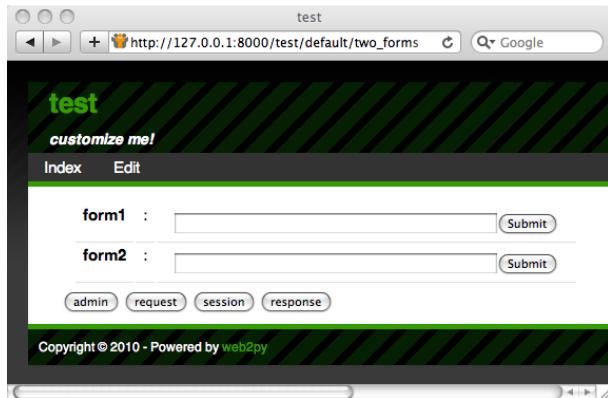
Es posible tener múltiples formas por página, pero debe permitir web2py para distinguirlos. Si estos se obtienen `SQLFORM` de diferentes tablas, entonces web2py les da diferentes nombres de forma automática, de lo contrario tiene que explícitamente les dan diferentes nombres formulario. Además, cuando están presentes las múltiples formas en la misma página, el mecanismo para la prevención de roturas en la doble presentación, y debe omitir el argumento `session` al llamar el método de `accepts`. Aquí está un ejemplo:

```

1 def two_forms():
2     form1 = FORM(INPUT(_name='name', requires=IS_NOT_EMPTY()),
3                   INPUT(_type='submit'))
4     form2 = FORM(INPUT(_name='name', requires=IS_NOT_EMPTY()),
5                   INPUT(_type='submit'))
6     if form1.accepts(request.vars, formname='form_one'):
7         response.flash = 'form one accepted'
8     if form2.accepts(request.vars, formname='form_two'):
9         response.flash = 'form two accepted'
10    return dict(form1=form1, form2=form2)

```

y aquí está la salida que produce:



Cuando el usuario envía un vacío form1, form1 sólo muestra un error, si el usuario envía un vacío form2, sólo Form2 muestra un mensaje de error.

7.1.6 Compartiendo las formas

El contenido de esta sección se aplica tanto a los FORM y SQLFORM objetos. Lo que discutimos aquí es posible pero no es recomendable, ya que siempre es buena práctica tener formas que se auto-envía. A veces, sin embargo, usted no tiene otra opción, porque la acción que envía el formulario y la acción que la recibe pertenecen a diferentes aplicaciones.

Es posible generar un formulario que se somete a una acción diferente. Esto se hace mediante la especificación de la dirección de la acción de transformación en los atributos del objeto FORM o SQLFORM. Por ejemplo:

```

1 form = FORM(INPUT(_name='name', requires=IS_NOT_EMPTY()),
2             INPUT(_type='submit'), _action=URL('page_two'))
3
4 def page_one():
5     return dict(form=form)
6
7 def page_two():
8     if form.accepts(request.vars, formname=None):
9         response.flash = 'form accepted'
10    else:

```

```

11     response.flash = 'there was an error in the form'
12     return dict()

```

Tenga en cuenta que ya que tanto el uso de la "page_one" and "page_two" form mismo, hemos definido una sola vez, colocándolo fuera de todas las acciones, a fin de no repetirnos. La parte común de código al principio de una controladora es ejecutado cada vez que antes de dar el control a la acción requerida.

Dado que no llama "page_one" accepts, la forma no tiene nombre y la clave no, por lo que no debe pasar el session formname=None y poner en accepts, o el formulario no validará cuando "page_two" que recibe.

7.2 SQLFORM

Pasamos ahora al siguiente nivel, proporcionando la aplicación con un archivo de modelos:

```

1 db = DAL('sqlite://storage.sqlite')
2 db.define_table('person', Field('name', requires=IS_NOT_EMPTY()))

```

Modificar el controlador de la siguiente manera:

```

1 def display_form():
2     form = SQLFORM(db.person)
3     if form.accepts(request.vars, session):
4         response.flash = 'form accepted'
5     elif form.errors:
6         response.flash = 'form has errors'
7     else:
8         response.flash = 'please fill out the form'
9     return dict(form=form)

```

La vista no tiene por qué ser cambiado.

En el nuevo controlador, no es necesario para construir una FORM, SQLFORM desde el constructor construyó uno de los db.person tabla definida en el mod-

elo. Esta nueva forma, cuando en serie, aparece como:

```

1 <form enctype="multipart/form-data" action="" method="post">
2   <table>
3     <tr id="person_name__row">
4       <td><label id="person_name__label"
5           for="person_name">Your name: </label></td>
6       <td><input type="text" class="string"
7           name="name" value="" id="person_name" /></td>
8       <td></td>
9     </tr>
10    <tr id="submit_record__row">
11      <td></td>
12      <td><input value="Submit" type="submit" /></td>
13      <td></td>
14    </tr>
15  </table>
16  <input value="9038845529" type="hidden" name="_formkey" />
17  <input value="person" type="hidden" name="_formname" />
18 </form>
```

El formulario generado automáticamente es más compleja que la anterior forma de bajo nivel. En primer lugar, contiene una tabla de filas y cada fila tiene tres columnas. La primera columna contiene el campo de las etiquetas (como se determina a partir de la db.person), la segunda columna contiene los campos de entrada (y, eventualmente, mensajes de error), y la tercera columna es opcional y por lo tanto vacío (que se pueden llenar con los campos en el constructor de SQLFORM).

Todas las etiquetas en forma tienen nombres derivados de la tabla y el nombre del campo. Esto permite una fácil personalización de la forma usando CSS y JavaScript. Esta capacidad se analiza con más detalle en el capítulo 10.

Más importante es que ahora el método `accepts` tiene mucho más trabajo para usted. Al igual que en el caso anterior, se realiza la validación de la entrada, sino que además, si la entrada supera la validación, sino que también realiza una inserción de base de datos del nuevo registro y tiendas en la `form.vars.id` "id" única del nuevo registro.

Un objeto SQLFORM también se ocupa automáticamente en la carpeta "upload" fields by saving uploaded files in the "uploads" (después de haberles cambiado el nombre para evitar conflictos con seguridad y prevenir los ataques de salto de directorio) y tiendas de sus nombres (sus nombres nuevos) en el campo apropiado en la base de datos.

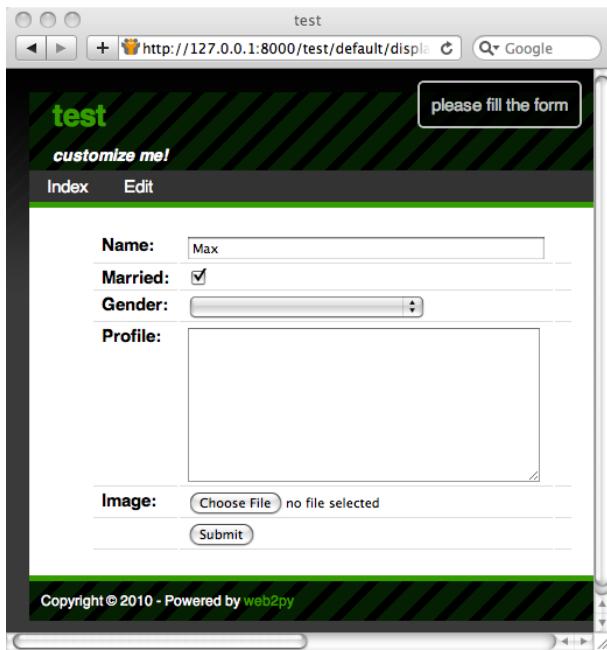
Una muestra SQLFORM "boolean" values with checkboxes, "text" values with textareas, values required to be in a definite set or a database with drop-boxes, and "upload" fields with links that allow users to download the uploaded files. It hides "blob" campos, ya que se supone que deben ser tratados de manera diferente, como se comenta más adelante.

Por ejemplo, consideremos el siguiente modelo:

```

1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('married', 'boolean'),
4     Field('gender', requires=IS_IN_SET(['Male', 'Female', 'Other'])),
5     Field('profile', 'text'),
6     Field('image', 'upload'))
```

En este caso, SQLFORM(db.person) genera la forma se muestra a continuación:



El constructor SQLFORM permite personalizaciones diferentes, como mostrar sólo un subconjunto de los campos, el cambio de las etiquetas, añadiendo valores a la tercera columna opcional, o la creación de formas y DELETE UPDATE, por oposición a las formas INSERT de la actual. SQLFORM es el único mayor ahorro de tiempo objeto en web2py.

El SQLFORM clase se define en "gluon/sqlhtml.py". Se puede ampliar fácilmente por la sobrecarga de su método xml, el método que serializa los objetos, cambiar su producción.

La firma para el constructor SQLFORM es la siguiente:

```

1 SQLFORM(table, record=None, deletable=False,
2         linkto=None, upload=None, fields=None, labels=None, col3={},
3         submit_button='Submit', delete_label='Check to delete:',
4         id_label='Record id: ', showid=True,
5         readonly=False, comments=True, keepopts=[],
6         ignore_rw=False, formstyle='table3cols', **attributes)

```

- El segundo argumento opcional convierte la forma INSERT en un formulario de Actualización para el registro especificado (véase la subsección siguiente).
- `deletable` Si se establece en `True`, el formulario de actualización muestra una casilla de verificación "Check to delete". El valor de la etiqueta, si este campo se establece a través del argumento de `delete_label`.
- `submit_button` establece el valor del botón de envío.
- `id_label` establece la etiqueta del registro de "id"
- El "id" del registro no aparece si se establece en `showid False`.
- `fields` es una lista opcional de nombres de campo que desea mostrar. Si se proporciona una lista, sólo campos de la lista se muestran. Por ejemplo:

```
1 fields = ['name']
```

- `labels` es un diccionario de las etiquetas de campo. La clave de diccionario es un nombre de campo y el valor correspondiente es lo que se muestra como su etiqueta. Si la etiqueta no se incluya el web2py deriva de la etiqueta el nombre del campo (que aprovecha el nombre del campo y sustituye subraya con espacios). Por ejemplo:

```
1 labels = {'name': 'Your Full Name: '}
```

- `col3` es un diccionario de valores para la tercera columna. Por ejemplo:

```
1 col3 = {'name': A('what is this?',
2 _href='http://www.google.com/search?q=define:name') }
```

- `linkto` y `upload` son direcciones URL opcional a los controladores definidos por el usuario que permiten a la forma para hacer frente a los campos de referencia. Esto se discute en más detalle más adelante en la sección.
- `readonly`. Si se establece en `True`, muestra la forma como sólo lectura
- `comments`. Si se establece en `False`, no muestra el COL3 comentarios
- `ignore_rw`. Normalmente, para un crear / actualizar la forma, sólo los campos marcados como modificables = true se muestra, y para las formas de sólo lectura, sólo los campos marcados como lectura = true se muestra. Establecer las causas `ignore_rw=True` las limitaciones que se han ignorado, y todos los campos se muestran. Casi siempre se usa en la interfaz de

appadmin para mostrar todos los campos para cada tabla, superando a lo que indica el modelo.

- `formstyle` determina el estilo que se utilizará cuando se serializa el formulario en html. Puede ser "table3cols" (default), "table2cols" (one row for label and comment, and one row for input), "ul" (makes an unordered list of input fields), "divs" (representa la forma usando CSS divs de usar, por customization arbitraria). `formstyle` también puede ser una función que toma (`record_id`, `field_label`, `field_widget`, `field_comment`) como atributos y devuelve un TR () del objeto.
- `attributes` argumentos son opcionales a partir de subrayar que desea pasar a la etiqueta FORM que hace que el objeto SQLFORM. Algunos ejemplos son:

```
1 _action = '.'
2 _method = 'POST'
```

No es un atributo especial . Cuando el diccionario se pasa como , sus elementos se convierten en campos de entrada "hidden" (véase el ejemplo de la ayuda en el capítulo 5 FORM).

7.2.1 SQLFORM e insertar, actualizar o borrar

Si pasa un registro como segundo argumento opcional para el constructor de SQLFORM, la forma se convierte en un formulario de Actualización para ese registro. Esto significa que cuando se envía el formulario del registro existente se actualiza y no se inserta el nuevo registro. Si establece el argumento de `deletable=True`, el formulario de actualización muestra una casilla de verificación "check to delete". Si se selecciona, se elimina el registro.

Puede, por ejemplo, modificar el controlador del ejemplo anterior, para que cuando se pasa un argumento entero adicional en la ruta URL, como en:

```
1 /test/default/display_form/2
```

y si hay un registro con el identificador correspondiente, el SQLFORM genera un UPDATE / DELETE formulario para el registro:

```

1 def display_form():
2     record = db.person(request.args(0)) or redirect(URL('index'))
3     form = SQLFORM(db.person, record)
4     if form.accepts(request.vars, session):
5         response.flash = 'form accepted'
6     elif form.errors:
7         response.flash = 'form has errors'
8     return dict(form=form)

```

La línea 3 se encuentra el registro, la línea 5 hace un UPDATE / DELETE forma, y la línea 7 hace un formulario de inserción. La línea 8 hace todo el procesamiento del formulario correspondiente.

Un formulario de actualización es muy similar a una forma de crear excepto que se rellenará con el registro actual y las imágenes previstas. Por defecto deletable = True que significa el formulario de actualización aparecerá una opción de "delete record".

Editar formularios contienen un campo de entrada escondida con name="id", que se utiliza para identificar el registro. Esta identidad también se almacena en el servidor de seguridad adicional y, si el visitante altere el valor de este campo, la actualización no se realiza y web2py plantea una SyntaxError, "user is tampering with form".

Cuando un campo se marca con writable=False, el campo no se muestra en la creación de formas, y es de sólo lectura se muestra en las formas de actualización. Si un campo se marca como writable=False y readable=False, entonces el campo no se muestra en absoluto, ni siquiera en las formas de actualización.

Los formularios creados con

```

1 form = SQLFORM(...,ignore_rw=True)

```

ignoran los atributos `readable` y `writable` y siempre muestran todos los campos. Formas de ignorar `appadmin` ellos de forma predeterminada.

Los formularios creados con

```
1 form = SQLFORM(table, record_id, readonly=True)
```

siempre muestran todos los campos en modo solo lectura, y no puede ser aceptada.

7.2.2 SQLFORM en HTML

Hay momentos en que desea utilizar `SQLFORM` para beneficiarse de su generación de la forma y el procesamiento, pero se necesita un nivel de personalización de la forma en HTML que no se puede lograr con los parámetros del objeto `SQLFORM`, así que hay que diseñar el formulario con HTML.

Ahora, edita el controlador anterior y añadir una nueva acción:

```
1 def display_manual_form():
2     form = SQLFORM(db.person)
3     if form.accepts(request.vars, formname='test'):
4         response.flash = 'form accepted'
5     elif form.errors:
6         response.flash = 'form has errors'
7     else:
8         response.flash = 'please fill the form'
9     return dict()
```

e insertar el formulario en la vista asociada "default/display_manual_form.html":

```
1 {{extend 'layout.html'}}
2 <form>
3 <ul>
4   <li>Your name is <input name="name" /></li>
5 </ul>
6   <input type="submit" />
7   <input type="hidden" name="_formname" value="test" />
8 </form>
```

Tenga en cuenta que la acción no devuelve el formulario, ya que no tiene por qué pasar a la vista. La vista contiene un formulario creado de forma manual en HTML. El formulario contiene una "_formname" campo oculto que debe ser la misma FormName especificado como un argumento de accepts en la acción. web2py utiliza el nombre del formulario en caso de que haya múltiples formas en la misma página, para determinar cuál de ellas se presentó. Si la página contiene una sola forma, puede establecer formname=None y omitir el campo oculto en la vista.

7.2.3 SQLFORM y Cargas

Los campos de tipo "upload" son especiales. Ellos se representan como campos de entrada del type="file". A menos que se especifique lo contrario, el archivo subido es escuchados en el uso de un tampón, y se almacena en la carpeta "uploads" de la aplicación que utiliza un nombre nuevo y seguro, asignado de forma automática. El nombre de este archivo se guarda en el campo de las subidas de tipos.

Como ejemplo, considere el siguiente modelo:

```
1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('image', 'upload')
```

Usted puede utilizar el controlador de "display_form" misma acción se muestra arriba.

Cuando se inserta un nuevo registro, el formulario le permite buscar un archivo. Elija, por ejemplo, una imagen jpg. El archivo se carga y se almacena como:

```
1 applications/test/uploads/person.image.XXXXXX.jpg
```

"XXXXXX" es un identificador aleatorio para el archivo asignado por web2py.

Tenga en cuenta que, por defecto, el nombre del archivo original de un archivo subido es `b16encoded` y se utiliza para construir el nuevo nombre para el archivo. Este nombre se recupera por la acción `download` y se utiliza para establecer el encabezado de disposición de contenido al nombre del archivo original.

Sólo se conserva su extensión. Este es un requisito de seguridad desde el nombre del archivo puede contener caracteres especiales que podría permitir a un usuario para realizar ataques de directorio transversal u otras operaciones maliciosas.

El nuevo nombre de archivo se almacena también en `form.vars.image`.

Al editar el registro utilizando un formulario de Actualización, que sería bueno para mostrar un enlace al archivo subido existentes, y web2py proporciona una manera de hacerlo.

Si pasa una dirección URL para el constructor `SQLFORM` mediante el argumento de carga, web2py utiliza la acción en esa dirección URL para descargar el archivo. Considere las siguientes acciones:

```

1 def display_form():
2     record = db.person(request.args(0)) or redirect(URL('index'))
3     form = SQLFORM(db.person, record, deletable=True,
4                     upload=URL('download'))
5     if form.accepts(request.vars, session):
6         response.flash = 'form accepted'
7     elif form.errors:
8         response.flash = 'form has errors'
9     return dict(form=form)
10
11 def download():
12     return response.download(request, db)
```

Ahora, inserte un nuevo registro en la URL:

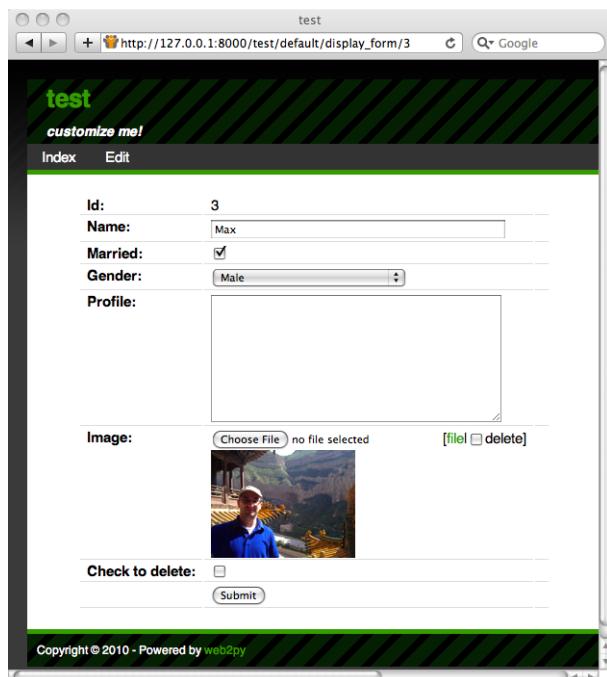
```
1 http://127.0.0.1:8000/test/default/display_form
```

Subir una foto, envía el formulario, y luego modificar el registro recién

creado, visite:

```
1 http://127.0.0.1:8000/test/default/display_form/3
```

(En este caso suponemos que el último disco ha id = 3). El formulario mostrará una previsualización de la imagen como se muestra a continuación:



Esta forma, cuando serializado, genera el siguiente código HTML:

```
1 <td><label id="person_image_label" for="person_image">Image: </label></td>
2 <td><div><input type="file" id="person_image" class="upload" name="image"
3 >[<a href="/test/default/download/person.image.0246683463831.jpg">file</a>|
4 <input type="checkbox" name="image_delete" />delete]</div></td><td></td></tr>
5 <tr id="delete_record_row"><td><label id="delete_record_label" for="delete_record"
6 >">Check to delete:</label></td><td><input type="checkbox" id="delete_record"
7 class="delete" name="delete_this_record" /></td>
```

que contiene un enlace para permitir la descarga del archivo cargado, y una

casilla de verificación para eliminar el archivo de registro de la base, por lo tanto almacenar NULL en el campo de "image".

¿Por qué este mecanismo expuesto? ¿Por qué tiene que escribir la función de descarga? Porque usted lo desea, puede hacer valer algún mecanismo de autorización en la función de descarga. Vea el Capítulo 8 para un ejemplo.

7.2.4 Almacenar el nombre de archivo original

web2py almacena automáticamente el nombre del archivo original que se encuentra el nuevo nombre de archivo UUID y recupera cuando el archivo se descarga. Al realizar la descarga, el nombre original del archivo se almacena en el encabezado de disposición de contenidos de la respuesta HTTP. Todo esto se hace de forma transparente sin necesidad de programación.

En ocasiones es posible que desee guardar el nombre del archivo original en un campo de base de datos. En este caso, tiene que modificar el modelo y agregar un campo para almacenarlo en:

```

1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('image_filename'),
4     Field('image', 'upload'))
```

entonces usted necesita para modificar el controlador para manejarlo:

```

1 def display_form():
2     record = db.person(request.args(0)) or redirect(URL('index'))
3     url = URL('download')
4     form = SQLFORM(db.person, record, deletable=True,
5                   upload=url, fields=['name', 'image'])
6     if request.vars.image:
7         form.vars.image_filename = request.vars.image.filename
8     if form.accepts(request.vars, session):
9         response.flash = 'form accepted'
10    elif form.errors:
11        response.flash = 'form has errors'
12    return dict(form=form)
```

Observe que el SQLFORM no muestra el campo de "image_filename". La acción "display_form" se mueve el nombre de archivo de la `request.vars.image` en el `form.vars.image_filename`, de manera que se procesa por `accepts` y se almacena en la base de datos. La función de descarga, antes de servir el archivo, revisa en la base de datos para el nombre del archivo original y lo utiliza en el encabezado de disposición de contenidos.

7.2.5 *autodelete*

El SQLFORM, al eliminar un registro, no se elimina el archivo físico subido (s) de referencia en el registro. La razón es que web2py no sabe si el mismo archivo se usa / unidas por otras tablas o utilizados para otro fin. Si usted sabe que es seguro eliminar el archivo real cuando el registro correspondiente se suprime, puede hacer lo siguiente:

```

1 db.define_table('image',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('file','upload',autodelete=True))
```

El atributo `autodelete False` es de forma predeterminada. Cuando se establece en `True` se asegura de que el archivo se elimina cuando se elimina el registro.

7.2.6 *Enlaces a los expedientes de referencia a*

Consideremos ahora el caso de dos tablas unidas por un campo de referencia. Por ejemplo:

```

1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()))
3 db.define_table('dog',
4     Field('owner', db.person),
5     Field('name', requires=IS_NOT_EMPTY()))
6 db.dog.owner.requires = IS_IN_DB(db,db.person.id,'%(name)s')
```

Una persona tiene perros, y cada perro pertenece a un propietario, que es una persona. El dueño del perro es necesario hacer referencia a una db.person.id válido por '%(name)s'.

Vamos a usar la interfaz de **appadmin** para esta aplicación para agregar unas pocas personas y sus perros.

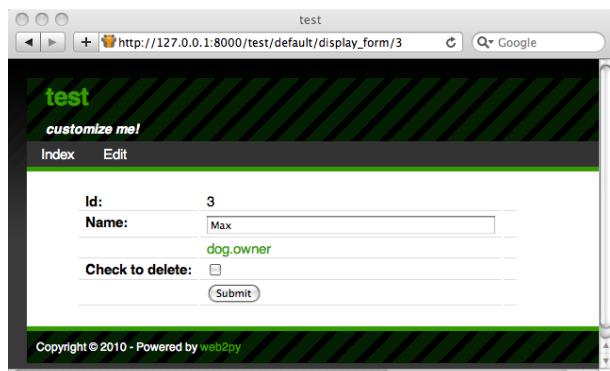
Al editar una persona existente, la forma UPDATE **appadmin** muestra un enlace a una página que enumera los perros que pertenecen a la persona. Este comportamiento se puede replicar con el argumento de la `linkto` SQLFORM. `linkto` tiene que apuntar a la dirección URL de una nueva acción que recibe una cadena de consulta de la SQLFORM y las listas de los registros correspondientes. He aquí un ejemplo:

```

1 def display_form():
2     record = db.person(request.args(0)) or redirect(URL('index'))
3     url = URL('download')
4     link = URL('list_records')
5     form = SQLFORM(db.person, records, deletable=True,
6                     upload=url, linkto=link)
7     if form.accepts(request.vars, session):
8         response.flash = 'form accepted'
9     elif form.errors:
10         response.flash = 'form has errors'
11     return dict(form=form)

```

Aquí está la página:



No hay un enlace llamado "dog.owner". El nombre de este enlace se puede cambiar mediante el argumento de la `labels` SQLFORM, por ejemplo:

```
1 labels = {'dog.owner': "This person's dogs"}
```

Si hace clic en el enlace que se canalizan en:

```
1 /test/default/list_records/dog?query=dog.owner%3D5
```

"list_records" es la acción especificada, con juego de `request.args(0)` en el nombre de la tabla de referencia y establecer `request.vars.query` a la cadena de consulta SQL. La cadena de consulta en la URL contiene el valor "dog.owner=5" adecuadamente con codificación URL (web2py decodifica automáticamente cuando la dirección se analiza).

Usted puede aplicar una acción de carácter general muy "list_records" de la siguiente manera:

```
1 def list_records():
2     table = request.args(0)
3     query = request.vars.query
4     records = db(query).select(db[table].ALL)
5     return dict(records=records)
```

con la visión asociados "default/list_records.html":

```
1 {{extend 'layout.html'}}
2 {{=records}}
```

Cuando un conjunto de registros devuelto por un selecto y serializado en un punto de vista, se convierte primero en un objeto SQLTABLE (no el mismo como una tabla) y luego serializado en una tabla HTML, donde cada campo corresponde a una columna de la tabla.

7.2.7 Pre-llenar el formulario

Siempre es posible llenar previamente un formulario utilizando la sintaxis:

```
1 form.vars.name = 'fieldvalue'
```

Declaraciones como la de arriba debe ser insertado después de la declaración antes de la forma y el formulario es aceptado, si el campo ("name" en el ejemplo) se visualizan de forma explícita en el formulario.

7.2.8 SQLFORM sin base de datos de E / S

Hay momentos en que desea generar un formulario de una tabla de base de datos utilizando SQLFORM y desea validar un formulario presentado en consecuencia, pero no desea que el almacenamiento automático INSERT / UPDATE / DELETE en la base de datos. Este es el caso, por ejemplo, cuando uno de los campos debe ser computado a partir del valor de los campos de entrada. Este es también el caso cuando se necesita para realizar la validación adicional sobre los datos insertados que no se puede lograr a través de los validadores estándar.

Esto puede hacerse fácilmente por rotura:

```
1 form = SQLFORM(db.person)
2 if form.accepts(request.vars, session):
3     response.flash = 'record inserted'
```

en:

```
1 form = SQLFORM(db.person)
2 if form.accepts(request.vars, session, dbio=False):
3     ### deal with uploads explicitly
4     form.vars.id = db.person.insert(**dict(form.vars))
5     response.flash = 'record inserted'
```

Lo mismo se puede hacer para UPDATE / DELETE, rompiendo las formas:

```
1 form = SQLFORM(db.person, record)
2 if form.accepts(request.vars, session):
3     response.flash = 'record updated'
```

en:

```

1 form = SQLFORM(db.person, record)
2 if form.accepts(request.vars, session, dbio=False):
3     if form.vars.get('delete_this_record', False):
4         db(db.person.id==record.id).delete()
5     else:
6         record.update_record(**dict(form.vars))
7     response.flash = 'record updated'

```

En ambos casos se ocupa web2py con el almacenamiento y el cambio de nombre del archivo cargado, como si dbio=True, el escenario por defecto. El nombre del fichero subido es en:

```
1 form.vars.fieldname
```

7.3 SQLFORMfactory

Hay casos en los que desea generar formas *as if* que había una tabla de base de datos pero usted no desea que la tabla de base de datos. Usted simplemente quieren tomar ventaja de la capacidad para generar SQLFORM una bonita forma de usar CSS y tal vez realizar archivo de carga y el cambio de nombre.

Esto se puede hacer a través de una `form_factory`. Aquí está un ejemplo donde se genera el formulario, realizar la validación, cargar un archivo y todo lo guarde en el `session`:

```

1 def form_from_factory():
2     form = SQLFORM.factory(
3         Field('your_name', requires=IS_NOT_EMPTY()),
4         Field('your_image', 'upload'))
5     if form.accepts(request.vars, session):
6         response.flash = 'form accepted'
7         session.your_name = form.vars.your_name
8         session.filename = form.vars.your_image
9     elif form.errors:
10         response.flash = 'form has errors'
11     return dict(form=form)

```

Esta es la opinión "default/form_from_factory.html":

```
1 {{extend 'layout.html'}}
2 {{=form}}
```

Tiene que usar un guión en lugar de un espacio para las etiquetas de campo, o explícitamente pasar un diccionario de `labels` a `form_factory`, como lo haría para un `SQLFORM`. Por defecto `SQLFORM.factory` genera el formulario usando `html "id"` attributes generated as if the form was generated from a table called "no_table". Para cambiar este nombre de la tabla ficticia, utilice el atributo de la fábrica `table_name`:

```
1 form = SQLFORM.factory(...,table_name='other_dummy_name')
```

Cambio de la `table_name` es necesario si usted necesita colocar dos formas de fábrica generados en la misma tabla y quiere evitar conflictos CSS.

7.4 CRUD

Una de las adiciones recientes a `web2py` es el crear / Leer / Actualizar / Eliminar (CRUD) de la API en la parte superior de `SQLFORM`. CRUD crea un `SQLFORM`, pero simplifica la codificación, ya que incorpora la creación de la forma, el tratamiento de la forma, la notificación, y el cambio, todos en una sola función.

La primera cosa a notar es que CRUD difiere de la API de otros `web2py` hemos utilizado hasta ahora porque ya no está expuesto. Tiene que ser importados. También debe estar vinculada a una base de datos específica. Por ejemplo:

```
1 from gluon.tools import Crud
2 crud = Crud(globals(), db)
```

El primer argumento del constructor es el `globals()` contexto actual para que CRUD puede acceder a la solicitud de locales, la respuesta, y la sesión. El

segundo argumento es un objeto de conexión de base de datos, db.

El objeto definido anteriormente crud ofrece las siguientes API:

- crud.tables() devuelve una lista de tablas definidas en la base de datos.
- crud.create(db.tablename) devuelve un nombre de tabla para crear la forma de tabla.
- crud.read(db.tablename, id) vuelve una forma de sólo lectura de nombre de tabla y el identificador de registro.
- crud.update(db.tablename, id) devuelve un formulario de actualización de nombre de tabla y el identificador de registro.
- crud.delete(db.tablename, id) elimina el registro.
- crud.select(db.tablename, query) devuelve una lista de registros seleccionados de la tabla.
- crud.search(db.tablename) vuelve aa tupla (forma, los registros) donde la forma es una forma de búsqueda y los registros es una lista de registros basado en el formulario de búsqueda presentadas.
- crud() vuelve uno de los anteriores basados en la request.args().

Por ejemplo, las siguientes actuaciones:

```
1 def data: return dict(form=crud())
```

expondría a las siguientes URL:

```
1 http://.../[app]/[controller]/data/tables
2 http://.../[app]/[controller]/data/create/[tablename]
3 http://.../[app]/[controller]/data/read/[tablename]/[id]
4 http://.../[app]/[controller]/data/update/[tablename]/[id]
5 http://.../[app]/[controller]/data/delete/[tablename]/[id]
6 http://.../[app]/[controller]/data/select/[tablename]
7 http://.../[app]/[controller]/data/search/[tablename]
```

Sin embargo, las siguientes actuaciones:

```

1 def create_tablename:
2     return dict(form=crud.create(db.tablename))

```

sólo exponer el método create

```
1 http://.../[app]/[controller]/create_tablename
```

Si bien las siguientes medidas:

```

1 def update_tablename:
2     return dict(form=crud.update(db.tablename, request.args(0)))

```

sólo exponer el método de actualización

```
1 http://.../[app]/[controller]/update_tablename/[id]
```

y así sucesivamente.

El comportamiento de CRUD se pueden personalizar de dos formas: mediante el establecimiento de algunos atributos del objeto crud o pasando parámetros extra a cada uno de sus métodos.

7.4.1 Configuración

Aquí está una lista completa de los atributos actuales CRUD, sus valores por defecto, y el significado:

Para hacer cumplir todas las formas de autenticación en requesón:

```
1 crud.settings.auth = None
```

El uso se explica en el capítulo 8.

Para especificar el controlador que define la función de la data, que devuelve el objeto crud

```
1 crud.settings.controller = 'default'
```

Para especificar la dirección URL para redirigir a un registro después de "create" con éxito:

```
1 crud.settings.create_next = URL('index')
```

Para especificar la dirección URL para redirigir a un registro después de "update" con éxito:

```
1 crud.settings.update_next = URL('index')
```

Para especificar la dirección URL para redirigir a un registro después de "delete" con éxito:

```
1 crud.settings.delete_next = URL('index')
```

Para especificar la URL que se utiliza para vincular los archivos subidos:

```
1 crud.settings.download_url = URL('download')
```

Para especificar funciones adicionales que se ejecutan después de los procedimientos de validación standrad formas crud.create:

```
1 crud.settings.create_onvalidation = StorageList()
```

StorageList es el mismo como un objeto Storage, ambos son definidos en el archivo de "gluon/storage.py", pero por defecto a una [] en lugar de None. Permite que la siguiente sintaxis:

```
1 crud.settings.create_onvalidation.mytablename.append(lambda form:....)
```

Para especificar funciones adicionales que se ejecutan después de los procedimientos de validación standrad formas crud.update:

```
1 crud.settings.update_onvalidation = StorageList()
```

Para especificar las funciones extra para ser ejecutado después de la finalización de las formas crud.create:

```
1 crud.settings.create_onaccept = StorageList()
```

Para especificar las funciones extra para ser ejecutado después de la finalización de las formas crud.update:

```
1 crud.settings.update_onaccept = StorageList()
```

Para especificar las funciones extra para ser ejecutado después de la finalización de la crud.update si el registro se elimina:

```
1 crud.settings.update_ondelete = StorageList()
```

Para especificar las funciones extra para ser ejecutado después de la finalización de la crud.delete:

```
1 crud.settings.delete_onaccept = StorageList()
```

Para determinar si el botón "update" forms should have a "delete":

```
1 crud.settings.update_deletable = True
```

Para determinar si las formas "update" debe mostrar el id del registro editado:

```
1 crud.settings.showid = False
```

Para determinar si las formas deben mantener los valores previamente insertado o reiniciarán por defecto después de su presentación con éxito:

```
1 crud.settings.keepvalues = False
```

Puede cambiar el estilo de formulario

```
1 crud.settings.formstyle = 'table3cols' or 'table2cols' or 'divs' or 'ul'
```

Puede agregar captcha a las formas, utilizando la misma convención, explicó a autoridades, con:

```
1 crud.settings.create_captcha = None
2 crud.settings.update_captcha = None
3 crud.settings.captcha = None
```

7.4.2 Mensajes

Aquí está una lista de mensajes personalizables:

```
1 crud.messages.submit_button = 'Submit'
```

establece el texto del botón "submit", tanto para crear y actualizar las formas.

```
1 crud.messages.delete_label = 'Check to delete:'
```

establece la etiqueta de las formas "delete" button in "update".

```
1 crud.messages.record_created = 'Record Created'
```

establece el mensaje flash en la creación del registro exitoso.

```
1 crud.messages.record_updated = 'Record Updated'
```

establece el mensaje flash en actualizar el registro de éxito.

```
1 crud.messages.record_deleted = 'Record Deleted'
```

establece el mensaje flash sobre la supresión de registro con éxito.

```
1 crud.messages.update_log = 'Record %(id)s updated'
```

establece el mensaje de registro en actualizar el registro de éxito.

```
1 crud.messages.create_log = 'Record %(id)s created'
```

establece el mensaje de registro en la creación de registro con éxito.

```
1 crud.messages.read_log = 'Record %(id)s read'
```

establece el mensaje de registro en registro con éxito el acceso de lectura.

```
1 crud.messages.delete_log = 'Record %(id)s deleted'
```

establece el mensaje de registro en registro borrado con éxito.

Tenga en cuenta que pertenece a la `crud.messages gluon.storage.Message` clase que es similar a la `gluon.storage.Storage`, pero sus valores se traduce de forma automática, sin necesidad de que el operador de T.

Los mensajes de registro se utilizan si y sólo si está conectado a CRUD de autenticación como se explica en el capítulo 8. Los sucesos se registran en el cuadro "auth_events" Auth.

7.4.3 Métodos

El comportamiento de los métodos CRUD también se puede personalizar en una base por llamada. Aquí están sus firmas:

```

1 crud.tables()
2 crud.create(table, next, onvalidation, onaccept, log, message)
3 crud.read(table, record)
4 crud.update(table, record, next, onvalidation, onaccept, ondelete, log, message,
     deletable)
5 crud.delete(table, record_id, next, message)
6 crud.select(table, query, fields, orderby, limitby, headers, **attr)
7 crud.search(table, query, queries, query_labels, fields, field_labels, zero)

```

- `table` es una tabla o un nombre de tabla DAL el método debe actuar.
- `record` y `record_id` son el identificador del registro el método debe actuar.
- `next` es la URL para redirigir a después del éxito. Si la URL contiene la subcadena "[id]" este será sustituido por el identificador del registro actualmente creado / actualizado.
- `onvalidation` tiene la misma función que SQLFORM `onvalidation (...)`
- `onaccept` es una función que se llamará después del envío del formulario es aceptado y actuar en consecuencia, pero antes de la redirección.
- `log` es el mensaje de registro. Entra mensajes en CRUD ver variables en el diccionario como `form.vars "%(id)s"`.
- `message` es el mensaje flash después de la aceptación formulario.

- `onDelete` se llama en lugar de `onaccept`, cuando se elimina un registro a través de un formulario de "update".
- `deletable` determina si el formulario "update" debe tener una opción de borrado.
- `query` es la consulta que se utilizará para seleccionar registros.
- `fields` es una lista de campos a ser seleccionado.
- `orderby` determina el orden en que los datos deberán ser seleccionados (véase el capítulo 6).
- `limitby` determina el rango de registros seleccionados que se debe mostrar (véase el capítulo 6).
- `headers` es un diccionario con los nombres de encabezado de tabla.
- `queries` ['`equals`', '`not equal`', '`contains`'] como una lista que contiene los métodos permitidos en el buscador.
- `query_labels` `query_labels=dict>equals='Equals'` un diccionario como nombres dando a los métodos de búsqueda.
- `fields` una lista de campos que se enumeran en el widget de búsqueda.
- `field_labels` un campo de la cartografía diccionario de nombres en las etiquetas.
- valores predeterminados para `zero` "choose one" se utiliza como opción predeterminada para el menú desplegable en el widget de búsqueda.

He aquí un ejemplo de uso en una función solo controlador:

```

1 # assuming db.define_table('person', Field('name'))
2 def people():
3     form = crud.create(db.person, next=URL('index'),
4                         message=T("record created"))
5     persons = crud.select(db.person, fields=['name'],
6                           headers={'person.name': 'Name'})
7     return dict(form=form, persons=persons)

```

Aquí hay otro controlador de función genérica mismo de que te permite buscar, crear y editar los registros de cualquier tabla donde el nombre de tabla se pasa `request.args(0)`:

```

1 def manage():
2     table=db[request.args(0)]
3     form = crud.update(table,request.args(1))
4     table.id.represent = lambda id: \
5         A('edit:',id,_href=URL(args=(request.args(0),id)))
6     search, rows = crud.select(table)
7     return dict(form=form,search=search,rows=rows)

```

Observe el `table.id.represent=...` línea que le indicará web2py para cambiar la representación del campo id y mostrar un vínculo en lugar de a la misma página y pasa el id como `request.args(1)` que convierte la creación de página en una página de actualización.

7.4.4 Registro de control de versiones

CRUD proporciona una utilidad a los registros de base de datos de versión:

Si usted tiene una tabla (`db.mytable`) que necesita la historia revisión total sólo puede hacer:

```
1 form=crud.update(db.mytable,myrecord,onaccept=crud.archive)
```

`crud.archive` define una nueva tabla llamada **db.mytable_history** (el nombre se deriva de la tabla de arriba) y una tienda cuando se actualiza, almacena una copia de la registro (antes de la actualización) en la tabla recién creada, que incluye una referencia al registro actual.

Debido a que el registro es en realidad actualización (sólo su estado anterior está archivado), las referencias no están rotos.

Esto se hace bajo el capó, pero puede que tenga que acceder a la tabla de la historia. Si desea tener acceso a mesa de este tipo se necesitan para definir en el modelo:

```

1 db.define_table('mytable_history',
2     Field('current_record',db.mytable),
3     db.mytable)

```

Notificación de la tabla se extiende db.mytable, es decir, incluye todos sus campos, y tiene una referencia a la current_record. crud.archive no marca de tiempo del registro almacenado a menos que su tabla original tiene campo de marca de tiempo, por ejemplo:

```

1 db.define_table('mytable',
2     Field('saved_on','datetime',
3           default=request.now,update=request.now,writable=False),
4     Field('saved_by',auth.user,
5           default=auth.user_id,update=auth.user_id,writable=False),

```

No hay nada especial acerca de estos campos, puede darles el nombre que deseé. Están llenos antes de que el expediente se archiva y se archivan con cada copia del acta.

Si desea cambiar el nombre de la tabla historia y / o el nombre del campo de referencia se puede hacer de esta manera:

```

1 db.define_table('myhistory',
2     Field('parent_record',db.mytable),
3     db.mytable)
4 # ...
5 form = crud.update(db.mytable,myrecord,
6                     onaccept=lambda form:crud.archive(form,
7                                         archive_table=db.myhistory,
8                                         current_record='parent_record'))

```

7.5 Formularios personalizados

Si un formulario se crea con SQLFORM, SQLFORM.factory o un mantenimiento, hay múltiples maneras que se pueden incrustar en una vista que permite varios grados de personalización. Consideremos por ejemplo el siguiente modelo:

```

1 db.define_table('image',
2     Field('name'),
3     Field('file', 'upload'))

```

```

1 def upload_image():
2     return dict(form=crud.create(db.image))

```

y la acción de carga

La forma más sencilla de integrar el formulario en la vista para la `upload_image` es

```
1 {{=form}}
```

Esto se traduce en un diseño de tabla estándar. Si desea utilizar un diseño diferente, puede romper la forma en componentes

```
1 {{=form.custom.begin}}
2 Image name: <div>{{=form.custom.widget.name}}</div>
3 Image file: <div>{{=form.custom.widget.file}}</div>
4 Click here to upload: {{=form.custom.submit}}
5 {{=form.custom.end}}
```

Cuando `form.custom.widget[fieldname]` se serializa en el widget adecuada para el campo. Si el formulario es enviado y que contiene errores, se anexan a continuación los widgets, como de costumbre.

El formulario de ejemplo anterior es muestra en la imagen debajo.

The screenshot shows a web page with a form. The first field is labeled "Image name:" with an empty text input field. The second field is labeled "Image file:" with a file input field showing "no file selected" and a "Choose File" button. Below these fields is a button labeled "Click here to upload:" followed by a "Submit" button.

Tenga en cuenta que un resultado similar Cound se han obtenido con:

```
1 crud.settings.formstyle='table2cols'
```

sin necesidad de utilizar un formulario personalizado. Otros posibles son `formstyles` "table3cols" (the default), "divs" and "ul".

Si usted no desea utilizar la widgets serializado por `web2py`, puede reemplazarlas con HTML. Hay algunas variables que serán de utilidad para esto:

- `form.custom.label[fieldname]` contiene la etiqueta para el campo.
- `form.custom.dspval[fieldname]` forma de modelo y la representación depende del tipo de campo de visualización del campo.
- `form.custom.inpval[fieldname]` forma de modelo y los valores en función del tipo de campo que se utilizará en un código de campo.

Es importante seguir las convenciones se describen a continuación.

7.5.1 CSS convenios

Tags en formularios generados por SQLFORM, SQLFORM.factory CRUD y seguir una estricta CSS convención de nomenclatura que se pueden utilizar para personalizar aún más las formas.

Dada una tabla de "mytable", a field "myfield" of type "string", es dictada en rebeldía por un

```
1 SQLFORM.widgets.string.widget
```

que tiene este aspecto:

```
1 <input type="text" name="myfield" id="mytable_myfield"
2   class="string" />
```

Tenga en cuenta que:

- la clase de la etiqueta de entrada es el mismo que el tipo del campo. Esto es muy importante para el código jQuery en campos "web2py_ajax.html" to work. It makes sure that you can only have numbers in "integer" and "double" fields, and that "time", "date" and "datetime" exhibirá el calendario emergente.
- el identificador es el nombre de la clase más el nombre del campo, unidos por un guión bajo. Esto permite hacer referencia únicamente al campo

a través, por ejemplo, `jquery('#mytable_myfield')` y manipular la hoja de estilo del campo o de obligar a las acciones asociadas a los eventos de campo (enfoque, desenfoque, KeyUp, etc.)

- el nombre es, como era de esperar, el nombre del campo.

7.5.2 Ocultar los errores

De vez en cuando, es posible que desee desactivar la colocación automática de errores y mostrar mensajes de error la forma en algún lugar que no sea el predeterminado. Eso se puede hacer fácilmente.

- En el caso de la forma o SQLFORM, pase al método `hiderror=True` `accepts`.
- En el caso de la cuajada, sistema `crud.settings.hiderror=True`

También puede modificar los puntos de vista para mostrar el error (ya que ya no se muestran de forma automática).

Aquí está un ejemplo donde los errores se muestran por encima de la forma y no en la forma.

```

1 {{if form.errors:}}
2   Your submitted form contains the following errors:
3   <ul>
4     {{for fieldname in form.errors:}}
5       <li>{{=fieldname}} error: {{=form.errors[fieldname]}}</li>
6     {{pass}}
7   </ul>
8   {{form.errors.clear()}}
9 {{pass}}
10 {{=form}}
```

Los errores se mostrarán como en la imagen de abajo.

Your submitted form contains the following errors:

- name error: enter a value

Name:	<input type="text"/>
File:	<input type="button" value="Choose File"/> no file selected
<input type="button" value="Submit"/>	

Este mecanismo también funciona para los formularios personalizados.

7.6 Validadores

Los validadores son clases que se utilizan para validar los campos de entrada (incluyendo formularios generados a partir de las tablas de base de datos).

He aquí un ejemplo del uso de un validador con una FORM:

```
1 INPUT(_name='a', requires=IS_INT_IN_RANGE(0, 10))
```

Aquí está un ejemplo de cómo exigir un validador de un campo de tabla:

```
1 db.define_table('person', Field('name'))
2 db.person.name.requires = IS_NOT_EMPTY()
```

Validadores siempre se asignan mediante el atributo de un campo de `requires`. Un campo puede tener un validador de uno o varios validadores. validadores múltiples pasan a ser parte de una lista:

```
1 db.person.name.requires = [IS_NOT_EMPTY(),
2                             IS_NOT_IN_DB(db, 'person.name')]
```

Validadores son llamados por el `accepts` función en un objeto de ayuda FORM o HTML que contiene un formulario. Ellos son llamados en el orden en que aparecen.

Construido en los validadores tienen constructores que toman la `error_message` argumento opcional, que le permite anular el mensaje de error por defecto.

He aquí un ejemplo de un validador en una mesa de base de datos:

```
1 db.person.name.requires = IS_NOT_EMPTY(error_message=T('fill this!'))
```

donde hemos utilizado el operador de traducción `T` para permitir la internacionalización. Tenga en cuenta que los mensajes de error por defecto no se traducen.

7.6.1 Validadores

IS_ALPHANUMERIC

Este validador comprueba que un valor de campo contiene caracteres sólo en los rangos az, AZ, o 0-9.

```
1 requires = IS_ALPHANUMERIC(error_message=T('must be alphanumeric!'))
```

IS_DATE

Este validador comprueba que un valor de campo contiene una fecha válida en el formato especificado. Es una buena práctica para especificar el formato mediante el operador de la traducción, a fin de apoyar diferentes formatos en distintos lugares.

```
1 requires = IS_DATE(format=T('%Y-%m-%d'),
2                      error_message=T('must be YYYY-MM-DD!'))
```

Para la descripción completa de las directivas% busque en el validador IS_DATETIME.

IS_DATE_IN_RANGE

Obras muy parecida a la anterior, pero validador permite especificar un rango:

```

1 requires = IS_DATE(format=T('%Y-%m-%d'),
2                     minimum=datetime.date(2008,1,1),
3                     maximum=datetime.date(2009,12,31),
4                     error_message=T('must be YYYY-MM-DD!'))

```

Para la descripción completa de las directivas% busque en el validador IS_DATETIME.

IS_DATETIME

Este validador comprueba que un valor de campo contiene una fecha y hora válida en el formato especificado. Es una buena práctica para especificar el formato mediante el operador de la traducción, a fin de apoyar diferentes formatos en distintos lugares.

```

1 requires = IS_DATETIME(format=T('%Y-%m-%d %H:%M:%S'),
2                         error_message=T('must be YYYY-MM-DD HH:MM:SS!'))

```

Los siguientes símbolos se pueden utilizar para la cadena de formato:

```

1 as a decimal number [00,53]. All days in a new year preceding
2 the first Sunday are considered to be in week 0.
3 as a decimal number [00,53]. All days in a new year preceding
4 the first Monday are considered to be in week 0.

```

IS_DATETIME_IN_RANGE

Obras muy parecida a la anterior, pero validador permite especificar un rango:

```

1 requires = IS_DATETIME(format=T('%Y-%m-%d %H:%M:%S'),
2                         minimum=datetime.datetime(2008,1,1,10,30),
3                         maximum=datetime.datetime(2009,12,31,11,45),
4                         error_message=T('must be YYYY-MM-DD HH:MM:SS!'))

```

Para la descripción completa de las directivas% busque en el validador IS_DATETIME.

IS_DECIMAL_IN_RANGE

```

1 INPUT(_type='text', _name='name', requires=IS_DECIMAL_IN_RANGE(0, 10))

```

Se determina que el argumento es (o puede ser representado como) un decimal de Python, y que está comprendida en el rango inclusivo especificado. La comparación se hace con Python aritmética decimal.

Los límites mínimos y máximos pueden ser None, significa que no hay límite inferior o superior, respectivamente. ##### IS_EMAIL

Comprueba que el valor del campo se parece a una dirección de correo electrónico. No se trata de enviar correo electrónico para confirmar.

```
1 requires = IS_EMAIL(error_message=T('invalid email!'))
```

IS_EQUAL_TO

Comprueba si el valor convalidadas es igual a un valor dado (que puede ser variable a bis):

```
1 requires = IS_EQUAL_TO(request.vars.password,
2                         error_message=T('passwords do not match'))
```

IS_EXPR

Su primer argumento es una cadena que contiene una expresión lógica en términos de valor de la variable. Valida un valor de campo si la expresión se evalúa como True. Por ejemplo:

```
1 requires = IS_EXPR('int(value)%3==0',
2                     error_message=T('not divisible by 3'))
```

Uno primero debe comprobar que el valor es un número entero de modo que una excepción no tendrá lugar.

```
1 requires = [IS_INT_IN_RANGE(0, 100), IS_EXPR('value%3==0')]
```

IS_FLOAT_IN_RANGE

Comprueba que el valor del campo es un número de punto flotante dentro de un rango definido, $0 < \text{value} < 100$ en el ejemplo siguiente:

```

1 requires = IS_FLOAT_IN_RANGE(0, 100,
2     error_message=T('too small or too large!'))

```

IS_INT_IN_RANGE

Comprueba que el valor del campo es un número entero dentro de un rango definido, $0 < \text{value} < 100$ en el ejemplo siguiente:

```

1 requires = IS_INT_IN_RANGE(0, 100,
2     error_message=T('too small or too large!'))

```

IS_IN_SET

Comprueba que los valores de campo están en un conjunto:

```

1 requires = IS_IN_SET(['a', 'b', 'c'], zero=T('choose one'),
2     error_message=T('must be a or b or c'))

```

El argumento cero es opcional y determina el texto de la opción seleccionada por defecto, una opción que no es aceptado por el validador `IS_IN_SET` sí mismo. Si usted no desea una opción de "choose one", `zero=False` conjunto

La opción `zero` se introdujo en la revisión (1.67.1). No se rompió la compatibilidad hacia atrás en el sentido de que no le quebraron las aplicaciones, pero lo hizo cambiar su comportamiento ya que, antes, no había otra opción `zero`.

Los elementos del conjunto siempre debe ser cadenas a menos que este validador está precedida por `IS_INT_IN_RANGE` (que convierte el valor a int) o `IS_FLOAT_IN_RANGE` (que convierte el valor a flotar). Por ejemplo:

```

1 requires = [IS_INT_IN_RANGE(0, 8), IS_IN_SET([2, 3, 5, 7],
2     error_message=T('must be prime and less than 10'))]

```

IS_IN_SET y Etiquetado

El validador de `IS_IN_SET` `multiple=False` tiene un atributo opcional. Si se establece en `True`, varios valores se pueden almacenar en un campo. El

campo en este caso debe ser un campo de cadena. Los valores se almacenan múltiples separados por una "|". multiple referencias se manejan de forma automática en crear y actualizar las formas, pero son transparentes para el DAL. Le sugerimos usar el plugin de jQuery para hacer multiselect varios campos.

IS_LENGTH

Comprueba si la longitud del valor del campo se ajusta entre los límites dado. Obras de texto y archivo de las entradas.

Sus argumentos son:

- maxsize: el máximo permitido longitud / tamaño
- MinSize: la longitud mínima permitida / tamaño

Ejemplos: Comprobar si la cadena de texto es más corto de 33 caracteres:

```
1 INPUT(_type='text', _name='name', requires=IS_LENGTH(32))
```

Compruebe si la cadena es la contraseña más de 5 caracteres:

```
1 INPUT(_type='password', _name='name', requires=IS_LENGTH(minsize=6))
```

Compruebe si el archivo tiene un tamaño subido entre 1KB y 1 MB:

```
1 INPUT(_type='file', _name='name', requires=IS_LENGTH(1048576, 1024))
```

Para todos los tipos de campo, salvo para los archivos, comprueba la longitud del valor. En el caso de los archivos, el valor es una cookie.FieldStorage, por lo que valida la longitud de los datos en el archivo, que es el comportamiento que intuitivamente se podría esperar.

IS_LIST_OF

Esto no es propiamente un validador. Su uso está pensado para permitir la

validación de los campos que devuelven varios valores. Se utiliza en los raros casos en que un formulario contiene varios campos con el mismo nombre o un cuadro de selección múltiple. Su único argumento es otro validador, y todo lo que hace es aplicar el validador de otros a cada elemento de la lista. Por ejemplo, la expresión siguiente comprueba que cada elemento de una lista es un número entero en el rango 0-10:

```
1 requires = IS_LIST_OF(IS_INT_IN_RANGE(0, 10))
```

Nunca devuelve un error y no contiene un mensaje de error. El validador interior controla la generación de errores.

IS_LOWER

Este validador no devuelve un error. Sólo convierte el valor en minúsculas.

```
1 requires = IS_LOWER()
```

IS_MATCH

Este validador coincide con el valor con una expresión regular y devuelve un error si no coinciden. He aquí un ejemplo de uso para validar un código postal de EE.UU.:

```
1 requires = IS_MATCH('^\d{5}(-\d{4})?$',  
2 error_message='not a zip code')
```

He aquí un ejemplo de uso para validar una dirección IPv4:

```
1 requires = IS_MATCH('^\d{1,3}(\.\d{1,3}){3}$',  
2 error_message='not an IP address')
```

He aquí un ejemplo de uso para validar un número de teléfono de EE.UU.:

```
1 requires = IS_MATCH('^1?\((-)\d{3}-?|(\d{3}\|)\)\d{3}-?\d{4}$',  
2 error_message='not a phone number')
```

Para obtener más información sobre las expresiones regulares de Python, consulte la documentación oficial de Python.

IS_NOT_EMPTY

Este validador comprueba que el contenido del valor del campo no es una cadena vacía.

```
1 requires = IS_NOT_EMPTY(error_message='cannot be empty!')
```

IS_TIME

Este validador comprueba que un valor de campo contiene una hora válida en el formato especificado.

```
1 requires = IS_TIME(error_message=T('must be HH:MM:SS!'))
```

IS_URL

Rechaza una cadena URL si alguno de los siguientes:

- La cadena está vacía o Ninguno
- La cadena utiliza caracteres no permitidos en una dirección URL
- La cadena no cumple alguna de las reglas sintácticas HTTP
- El esquema de URL especificado (si se especifica) no es "http" o "https";
- El dominio de nivel superior (si es un nombre de host se especifica) no existe

(Estas normas se basan en 2616 (65) RFC)

Esta función sólo comprueba la sintaxis de la URL. No comprueba que la dirección URL señala a un documento real, por ejemplo, o que de otro modo tiene sentido semántico. Esta función realiza automáticamente anteponer 'http://' delante de una URL en el caso de una URL abreviada (por ejemplo, «google.ca»).

Si el modo de parámetro = 'genérico' se utiliza, a continuación, cambia el comportamiento de esta función. A continuación, rechaza una cadena URL si alguno de los siguientes:

- La cadena está vacía o Ninguno
- La cadena utiliza caracteres no permitidos en una dirección URL
- El esquema de URL especificado (si se especifica) no es válido

(Estas normas se basan en 2396 (66) RFC)

La lista de regímenes permitido es personalizable con el parámetro allowed_schemes. Si se excluye ninguno de la lista, luego abreviado URL (que carecen de un régimen como "http") será rechazada.

El valor predeterminado antepone régimen es personalizable con el parámetro prepend_scheme. Si establece prepend_scheme en Ninguno, a continuación, anteponiendo se desactivará. URL que requieren anteponiendo para analizar será aceptado, pero el valor de retorno no se modificará.

IS_URL es compatible con el nombre de dominio internacionalizados (IDN) estándar especificado en el RFC 3490 (67)). Como resultado, las direcciones URL pueden ser cadenas regulares o cadenas Unicode. Si la URL de los componentes de dominio de la (google.ca por ejemplo) contiene-US-ASCII letras no, entonces el dominio se convierte en Punycode (definida en el RFC 3492 (68)). IS_URL va un poco más allá de las normas, y permite-US-ASCII caracteres no estar presente en la ruta de acceso y consulta de los componentes de la dirección también. Estos US-ASCII a caracteres no será codificado. Por ejemplo, el espacio se codifican como '% 20'. El carácter Unicode con el código hexadecimal ox4e86 se convertirá en '% 4e% 86'.

Ejemplos:

```

1 requires = IS_URL()
2 requires = IS_URL(mode='generic')
3 requires = IS_URL(allowed_schemes=['https'])
4 requires = IS_URL(prepend_scheme='https')
```

```

5 requires = IS_URL(mode='generic',
6                     allowed_schemes=['ftps', 'https'],
7                     prepend_scheme='https')
```

IS_SLUG

```
1 requires = IS_SLUG(maxlen=80, check=False, error_message='must be slug')
```

Si `check` se establece en los controles Verdadero si el valor validado es una babosa (permitiendo sólo caracteres alfanuméricos y guiones no repetidas).

Si `checks` se establece en `False` (por defecto) se convierte el valor de entrada a una babosa.

IS_STRONG

Vigila el cumplimiento de los requisitos de complejidad en un campo (normalmente un campo de contraseña)

Ejemplo:

```
1 requires = IS_STRONG(min=10, special=2, upper=2)
```

donde

- `min` es la longitud mínima del valor
- `especial` es el número mínimo requerido de caracteres especiales son caracteres especiales cualquiera de los following !@#\$%&*(){}[]-+
- `superior` es el número mínimo de caracteres en mayúsculas

IS_IMAGE

Este validador comprueba si el archivo subido a través de la entrada del archivo se guardó en uno de los formatos de imagen seleccionada, y sus dimensiones (anchura y altura) dentro de los límites señalados.

No comprueba el tamaño máximo de archivo (IS_LENGTH uso para eso). Devuelve un error de validación de datos si no se ha subido. Soporta los formatos de archivo BMP, GIF, JPEG, PNG, y no requiere la colección de imágenes de Python.

Código partes tomadas de ref. (69)

Toma los siguientes argumentos:

- extensiones: iterable que contiene la imagen permite extensiones de archivo en minúsculas
- maxsize: iterable contengan ancho máximo y la altura de la imagen
- MinSize: iterable que contenga como mínimo la anchura y altura de la imagen

Utilice (-1, -1) como MinSize de pasar por alto la comprobación de la imagen a tamaño.

Éstos son algunos ejemplos:

- Compruebe si el archivo subido es en cualquiera de los formatos de imagen soportados:

```
1 requires = IS_IMAGE()
```

- Compruebe si el archivo subido es en formato JPEG o PNG:

```
1 requires = IS_IMAGE(extensions=('jpeg', 'png'))
```

- Comprobar si el archivo subido es PNG con un tamaño máximo de 200x200 pixeles:

```
1 requires = IS_IMAGE(extensions=('png'), maxsize=(200, 200))
```

IS_UPLOAD_FILENAME

Este validador comprueba si el nombre y la extensión de archivo subido por entrada de archivo coincide con los criterios necesarios.

No se garantiza el tipo de archivo de ninguna manera. Devuelve error de validación de datos si no se ha subido.

Sus argumentos son:

- Nombre de archivo: nombre de archivo (antes del punto) regex.
- extensión: la extensión (después de punto) regex.
- lastdot: que punto se debe utilizar como un nombre de archivo / separador de la extensión: indica `True` último punto (por ejemplo, "file.tar.gz" will be broken in "file.tar"+"gz"), mientras que los medios `False` primer punto (por ejemplo, `file.tar.gz` will be broken into "file"+.gz).
- caso: 0 significa mantener el caso; 1 significa transformar la cadena en minúsculas (por defecto); 2 significa transformar la cadena en mayúsculas.

Si no hay punto de autos, los controles de extensión se hará contra la cadena vacía y los controles contra el nombre de archivo de valor.

Ejemplos:

Compruebe si el archivo tiene una extensión pdf (mayúsculas y minúsculas):

```
1 requires = IS_UPLOAD_FILENAME(extension='pdf')
```

Compruebe si el archivo tiene una extensión tar.gz y nombre a partir de copia de seguridad:

```
1 requires = IS_UPLOAD_FILENAME(filename='backup.*', extension='tar.gz', lastdot=False)
```

Compruebe si el archivo no tiene extensión y el nombre correspondiente README (mayúsculas y minúsculas):

```
1 requires = IS_UPLOAD_FILENAME(filename='^README$', extension='^$', case=0)
```

IS_IPV4

Este validador comprueba si el valor de un campo es una dirección IP versión 4 en forma decimal. Se puede establecer en vigor direcciones de un cierto rango. regex IPv4 tomado de ref. (70) Sus argumentos son:

- dirección `minip` menor la más baja; acepta: str, por ejemplo, `192.168.0.1`; iterable de los números, por ejemplo, `[192, 168, 0, 1]`; int, por ejemplo, `3232235521`
- dirección `maxip` nivel más alto permitido; mismo que el anterior

Los tres valores de ejemplo son iguales, ya que las direcciones se convierten en enteros para comprobar su inclusión con la siguiente función:

```
1 number = 16777216 * IP[0] + 65536 * IP[1] + 256 * IP[2] + IP[3]
```

Ejemplos:

Compruebe la dirección IPv4 válida:

```
1 requires = IS_IPV4()
```

Verificar la validez de direcciones de red IPv4 privadas:

```
1 requires = IS_IPV4(minip='192.168.0.1', maxip='192.168.255.255')
```

IS_LOWER

Este validador no devuelve un error. Se convierte el valor en minúsculas.

IS_UPPER

Este validador no devuelve un error. Se convierte el valor a mayúsculas.

```
1 requires = IS_UPPER()
```

IS_NULL_OR

Obsoletas, un alias para IS_EMPTY_OR se describe a continuación.

IS_EMPTY_OR

A veces es necesario permitir que los valores vacíos en un campo junto con otros requisitos. Por ejemplo, un campo puede ser una fecha, pero que también puede estar vacío. El validador IS_EMPTY_OR permite esto:

```
1 requires = IS_NULL_OR(IS_DATE())
```

LIMPIEZA

Se trata de un filtro. Nunca falla. Simplemente elimina todos los caracteres cuyos códigos decimales ASCII no están en la lista [10, 13, 32-127].

```
1 requires = CLEANUP()
```

CRIPTA

Esto también es un filtro. Se realiza un control seguro sobre la entrada y se utiliza para prevenir que las contraseñas de ser aprobada en el claro de la base de datos.

```
1 requires = CRYPT()
```

Si una clave no se especifica, se utiliza el algoritmo MD5. Si una clave se especifica CRIPTA utiliza el algoritmo HMAC. La clave puede tener un prefijo que determina el algoritmo para el uso de HMAC, por ejemplo SHA512:

```
1 requires = CRYPT(key='sha512:thisisthekey')
```

Esta es la sintaxis recomendada. La clave tiene que ser una cadena única asociada a la base de datos utilizada. La clave no puede ser cambiado. Si pierde la clave hash previamente los valores se vuelven inútiles.

7.6.2 Base de datos de los validadores

IS_NOT_IN_DB

Consideremos el siguiente ejemplo:

```
1 db.define_table('person', Field('name'))
2 db.person.name.requires = IS_NOT_IN_DB(db, 'person.name')
```

Se requiere que cuando se inserta una nueva persona, su nombre no está ya la base de datos, db, person.name en el campo. Como con todos los otros validadores se hace cumplir este requisito en el nivel de procesamiento de formularios, no a nivel de base de datos. Esto significa que hay una pequeña probabilidad de que, si dos usuarios intentan insertar registros simultáneamente con el person.name mismo, esto resulta en una condición de carrera y los registros de ambos son aceptados. Por lo tanto, más seguros de informar también a la base de datos que este campo debe tener un valor único:

```
1 db.define_table('person', Field('name', unique=True))
2 db.person.name.requires = IS_NOT_IN_DB(db, 'person.name')
```

Ahora bien, si se produce una condición de carrera, la base de datos plantea un OperationalError y uno de los dos insertos se rechaza.

El primer argumento de IS_NOT_IN_DB puede ser una conexión de base de datos o un conjunto. En este último caso, sería la comprobación sólo el conjunto definido por el conjunto.

El siguiente código, por ejemplo, no permite el registro de dos personas con el mismo nombre dentro de los 10 días de separación:

```
1 import datetime
2 now = datetime.datetime.today()
3 db.define_table('person',
4     Field('name'),
5     Field('registration_stamp', 'datetime', default=now))
6 recent = db(db.person.registration_stamp>now-datetime.timedelta(10))
7 db.person.name.requires = IS_NOT_IN_DB(recent, 'person.name')
```

IS_IN_DB

Considere las siguientes tablas y exigencia:

```

1 db.define_table('person', Field('name', unique=True))
2 db.define_table('dog', Field('name'), Field('owner', db.person)
3 db.dog.owner.requires = IS_IN_DB(db, 'person.id', '%(name)s',
4 zero=T('choose one'))

```

Se aplican a nivel de INSERT perro / UPDATE / DELETE formas. Se requiere que un ser dog.owner un ID válido en el campo de la person.id db base de datos. Debido a este validador, el campo dog.owner se presenta como una caja de selección. El tercer argumento del validador es una cadena que describe los elementos de la caja de selección. En el ejemplo que usted desea ver el %(name)s persona en lugar del %(id)s persona. %(...)s se sustituye por el valor del campo entre corchetes para cada registro.

La opción zero funciona muy parecido para el validador IS_IN_SET.

Si desea que el campo validado, pero no desea una dropbox, usted debe poner el validador en una lista.

```

1 db.dog.owner.requires = [IS_IN_DB(db, 'person.id', '%(name)s')]

```

El primer argumento del validador puede ser una conexión de base de datos o un conjunto de DAL, como en IS_NOT_IN_DB.

De vez en cuando desea que el cuadro desplegable (para que no desea utilizar la lista de la sintaxis más arriba), pero que desea utilizar validadores adicionales. Para ello, el validador IS_IN_DB tiene un argumento extra que _and puede apuntar a una lista de validadores de otra índole aplicadas si el valor validado pasa la validación IS_IN_DB. Por ejemplo, para validar todos los dueños de perros en db que no están en un subconjunto:

```

1 subset=db(db.person.id>100)
2 db.dog.owner.requires = IS_IN_DB(db, 'person.id', '%(name)s',
3 _and=IS_NOT_IN_DB(subset,'person.id'))

```

IS_IN_DB y Etiquetado

El validador de `IS_IN_DB` `multiple=False` tiene un atributo opcional. Si es asignado true varios valores se pueden almacenar en un campo. El campo en este caso no puede ser una referencia, pero debe ser un campo de cadena. Los valores se almacenan múltiples separados por una "|". `multiple` referencias se manejan de forma automática en crear y actualizar las formas, pero son transparentes para el DAL. Le sugerimos usar el plugin de jQuery para hacer multiselect varios campos.

7.6.3 Validadores personalizados

Todos los validadores seguir el prototipo a continuación:

```

1 class sample_validator:
2     def __init__(self, *a, error_message='error'):
3         self.a = a
4         self.e = error_message
5     def __call__(value):
6         if validate(value):
7             return (parsed(value), None)
8         return (value, self.e)
9     def formatter(self, value):
10        return format(value)

```

por ejemplo, cuando llama para validar un valor, un validador devuelve una tupla (`x`, `y`). Si es `y` `None`, entonces el valor que se pasa la validación y la `x` contiene un valor analizado. Por ejemplo, si el validador requiere que el valor de ser un número entero, se convierte en `x int(value)`. Si el valor no se han validado, a continuación, `x` contiene el valor de entrada y `y` contiene un mensaje de error que explica la validación ha fallado. Este mensaje de error se utiliza para informar del error en formas que no validan.

El validador puede contener un método de `formatter`. Se debe realizar la conversión opuesta a la que el `__call__` hace. Por ejemplo, considere el código fuente para `IS_DATE`:

```

1 class IS_DATE(object):
2     def __init__(self, format='%Y-%m-%d', error_message='must be YYYY-MM-DD!'):
3         self.format = format
4         self.error_message = error_message
5     def __call__(self, value):
6         try:
7             y, m, d, hh, mm, ss, t0, t1, t2 = time.strptime(value, str(self.format)
8                     )
9             value = datetime.date(y, m, d)
10            return (value, None)
11        except:
12            return (value, self.error_message)
13    def formatter(self, value):
14        return value.strftime(str(self.format))

```

En caso de éxito, el método `__call__` lee un texto de tiempo de la forma y la convierte en un objeto utilizando `datetime.date` la cadena de formato especificado en el constructor. El objeto `formatter` tiene un objeto `datetime.date` y la convierte en una representación de cadena usando el mismo formato. El `formatter` es llamado automáticamente los formularios, pero también se puede llamar explícitamente para convertir los objetos en su representación adecuada. Por ejemplo:

```

1 >>> db = DAL()
2 >>> db.define_table('atable',
3                     Field('birth', 'date', requires=IS_DATE('%m/%d/%Y')))
4 >>> id = db.atable.insert(birth=datetime.date(2008, 1, 1))
5 >>> row = db.atable[id]
6 >>> print db.atable.formatter(row.birth)
7 01/01/2008

```

Cuando los validadores se requieren múltiples (y se almacena en una lista), que se ejecutan en orden y la salida de una se pasa como entrada a otra. La cadena se rompe cuando uno de los validadores no.

Por el contrario, cuando llamamos al método `formatter` de un campo, los formateadores de los validadores asociados también están encadenados, pero en orden inverso.

7.6.4 Validadores con dependencias

En ocasiones, es necesario para validar un campo y el validador depende del valor de otro campo. Esto se puede hacer, pero se requiere establecer el validador en el controlador, cuando el valor de otro campo que se conoce. Por ejemplo, aquí hay una página que genera un formulario de registro que pide nombre de usuario y contraseña dos veces. Ninguno de los campos pueden estar vacíos, y las dos contraseñas que deben ajustarse:

```

1 def index():
2     form = SQLFORM.factory(
3         Field('username', requires=IS_NOT_EMPTY()),
4         Field('password', requires=IS_NOT_EMPTY()),
5         Field('password_again',
6             requires=IS_SAME_AS(request.vars.password)))
7     if form.accepts(request.vars, session):
8         pass # or take some action
9     return dict(form=form)

```

El mismo mecanismo se puede aplicar a los objetos y SQLFORM FORMA.

7.7 Reproductores

Aquí está una lista de los widgets web2py:

```

1 SQLFORM.widgets.string.widget
2 SQLFORM.widgets.text.widget
3 SQLFORM.widgets.password.widget
4 SQLFORM.widgets.integer.widget
5 SQLFORM.widgets.double.widget
6 SQLFORM.widgets.time.widget
7 SQLFORM.widgets.date.widget
8 SQLFORM.widgets.datetime.widget
9 SQLFORM.widgets.upload.widget
10 SQLFORM.widgets.boolean.widget
11 SQLFORM.widgets.options.widget
12 SQLFORM.widgets.multiple.widget
13 SQLFORM.widgets.radio.widget
14 SQLFORM.widgets.checkboxes.widget
15 SQLFORM.widgets.autocomplete

```

Los primeros diez de ellos son los valores predeterminados para los tipos de campo correspondientes. Los widget "options" se utiliza cuando un campo se requiere `IS_IN_SET` o `IS_IN_DB` con el comportamiento por defecto `multiple=False` (). Los widget "multiple" se utiliza cuando un campo se requiere `IS_IN_SET` o `IS_IN_DB` con `multiple=True`. Los widgets "radio" and "checkboxes" nunca se usan por defecto, pero se puede configurar manualmente. Los widget autocomplete es especial y debatido en su propia sección.

Por ejemplo, para tener un campo "string" representada por un área de texto:

```
1 Field('comment', 'string', widget=SQLFORM.widgets.text.widget)
```

Puede crear nuevos widgets o ampliar reproductores existentes.

`SQLFORM.widgets[type]` es una clase y `SQLFORM.widgets[type].widget` es una función miembro estática de la clase correspondiente. Cada función widget toma dos argumentos: el objeto de campo, y el valor actual de ese campo. Devuelve una representación del artílugo. Como ejemplo, el widget de cadena puede ser recodificado de la siguiente manera:

```
1 def my_string_widget(field, value):
2     return INPUT(_name=field.name,
3                 _id="%s_%s" % (field._tablename_, field.name),
4                 _class=field.type,
5                 _value=value,
6                 requires=field.requires)
7
8 Field('comment', 'string', widget=my_string_widget)
```

Los valores de id y class debe seguir la convención se describe más adelante en este capítulo. Un widget puede contener su validadores propios, pero es una buena práctica de asociar los validadores para el atributo "requires" del campo y tener el widget obtener desde allí.

7.7.1 Autocompletar Widget

Hay dos posibles usos para el widget de autocompletar: para autocompletar un campo que toma un valor de lista o para autocompletar un campo de referencia (donde la cadena a Javascript se es una representación de la referencia que se implementa como un id).

El primer caso es fácil:

```

1 db.define_table('category',Field('name'))
2 db.define_table('product',Field('name'),Field('category'))
3 db.product.category.widget = SQLHTML.widgets.autocomplete(
4     request, db.category.name, limitby=(0,10), min_length=2)

```

Cuando `limitby` instruye el widget para mostrar no más de 10 sugerencias en ese momento, y `min_length` instruye el widget para realizar una devolución de llamada Ajax en busca de sugerencias sólo después de que el usuario ha escrito al menos 2 caracteres en el cuadro de búsqueda.

El segundo caso es más complejo:

```

1 db.define_table('category',Field('name'))
2 db.define_table('product',Field('name'),Field('category'))
3 db.product.category.widget = SQLHTML.widgets.autocomplete(
4     request, db.category.name, id_field=db.category.id)

```

En este caso el valor de `id_field` dice el widget que incluso si el valor que se Javascript se es un `db.category.name`, el valor que se almacena es el `db.category.id` correspondiente. Un parámetros opcionales que instruye `orderby` es el widget en la forma de ordenar las sugerencias (en orden alfabético por defecto).

Este widget funciona a través de Ajax. ¿Dónde está la devolución de llamada Ajax? Algunos magia que está pasando en este widget. La devolución de llamada es un método para el widget propio objeto. ¿Cómo se exponen? En web2py cualquier pieza de código se puede generar una respuesta al elevar una excepción HTTP. Este widget explota esta posibilidad de la siguiente manera: el widget envía la llamada Ajax a la misma URL que ha generado

el widget en el primer lugar y pone un especial de token en el request.vars. Ella el widget se crea una instancia más, que encuentra el token y lanza una excepción HTTP que responde a la solicitud. Todo esto se hace bajo el capó y ocultos a los países desarrollados.

8

Control de Acceso

web2py incluye un potente y personalizable mecanismo de acceso basado en roles de Control (RBAC).

He aquí una definición de Wikipedia:

La función control de acceso basado en roles (RBAC) es una aproximación al sistema de restricción de acceso a usuarios autorizados. Se trata de un enfoque alternativo más reciente para el control de acceso obligatorio (MAC) y control de acceso discrecional (DAC). RBAC se refiere a veces como basada en roles la seguridad.

RBAC es una política neutral y flexible tecnología de control de acceso lo suficientemente poderosa para simular DAC y MAC. Por el contrario, el MAC puede simular RBAC si el gráfico de papel se limita a un árbol en lugar de un conjunto parcialmente ordenado.

Antes del desarrollo de RBAC, MAC y el CAD se consideraban los únicos modelos conocidos de control de acceso: si un modelo no fue CMA, se consideró como un modelo CAD, y viceversa. La investigación en la década de 1990 demostraron que RBAC cae en ninguna categoría.

Dentro de una organización, las funciones se crean para diversas funciones laborales. Los permisos para realizar determinadas operaciones se les asignan funciones específicas. Los miembros del personal (u otros usuarios del sistema) se asignan funciones específicas, ya través de las asignaciones de funciones adquirir los permisos necesarios para realizar determinadas funciones del sistema. A diferencia del control de acceso basado en el contexto (CBAC), RBAC no toma en cuenta el contexto del mensaje (tales como fuente de una conexión).

Puesto que los usuarios no se asignan los permisos directamente, sino sólo a adquirir a través de su papel (o papeles), la gestión de derechos de usuario individual se convierte en una cuestión de simplemente asignar roles apropiados para el usuario, lo que simplifica las operaciones comunes, como agregar un usuario, o cambiar un usuario del departamento.

RBAC se diferencia de las listas de control de acceso (ACL) utilizados en los sistemas tradicionales de control de acceso discrecional al atribuir los permisos a las operaciones específicas de significado en la organización, en lugar de datos de bajo nivel de los objetos. Por ejemplo, una lista de control de acceso podría ser utilizada para conceder o denegar acceso de escritura a un archivo de sistema en particular, pero no dictan cómo el expediente podría ser cambiado. "

La clase que implementa web2py RBAC se llama **Auth**. necesidades **Auth** (y define) los siguientes cuadros:

- nombre de los usuarios tiendas auth_user, dirección de correo electrónico, contraseña, y el estado (registro pendiente, aceptado, bloqueada)
- grupos de tiendas auth_group o funciones para los usuarios en una estructura de muchos a muchos. De forma predeterminada, cada usuario está en su propio grupo, pero un usuario puede estar en varios grupos y cada grupo puede contener varios usuarios. Un grupo se identifica por un papel y una descripción.
- usuarios enlaces auth_membership y grupos con una estructura de muchos

a muchos.

- grupos de enlaces auth_permission y permisos. Un permiso se identifica por un nombre y, opcionalmente, una mesa y grabar un archivo. Por ejemplo, los miembros de un determinado grupo puede tener permisos "update" en un registro específico de una tabla específica.
- auth_event registros de los cambios en los otros cuadros y el acceso a través de CRUD con éxito a los objetos controlados por el RBAC.

En principio, no hay ninguna restricción sobre los nombres de las funciones y los nombres de los permisos, el desarrollador puede crear para arreglar los papeles y permisos en la organización. Una vez que se haya creado, web2py proporciona una API para comprobar si un usuario está conectado, si un usuario es miembro de un grupo dado, y / o si el usuario es miembro de ningún grupo que tenga un permiso dado requerido. web2py también proporciona decoradores para restringir el acceso a cualquier función sobre la base de entrada, la composición y permisos. web2py también comprende algunos permisos específicos, es decir, aquellos que tienen un nombre que se corresponden con los métodos CRUD (crear, leer, actualizar, eliminar) y puede hacer valer de forma automática sin la necesidad de utilizar los decoradores.

En este capítulo, vamos a discutir las diferentes partes de RBAC uno por uno.

8.1 Autenticación

Para poder usar RBAC, los usuarios deben ser identificados. Esto significa que tienen que registrarse (o estar registrado) y entrar al sistema

Auth proporciona múltiples métodos de entrada. El defecto consiste en la identificación de usuarios basado en la tabla auth_user local. Alternativamente, se puede identificar a los usuarios contra los sistemas de autenticación de tercero y solo signo a los proveedores como Google, PAM, LDAP, Facebook,

LinkedIn, OpenID, OAuth, etc.

Para empezar a utilizar Auth, necesita al menos el código en un archivo de modelo, que también se proporciona con la aplicación "welcome" web2py y asume un objeto de conexión db:

```
1 from gluon.tools import Auth
2 auth = Auth(globals(), db)
3 auth.define_tables(username=False)
```

Establecer username=True si quieres autoridades a nombre de usuario para inicio de sesión en lugar del correo electrónico.

Para exponer **Auth**, también necesita la siguiente función en un controlador (por ejemplo en "default.py"):

```
1 def user(): return dict(form=auth())
```

El objeto auth y user la acción ya han sido definidas en la solicitud de andamios.

web2py también incluye una vista de ejemplo "default/user.html" para que esta funcione correctamente que tiene este aspecto:

```
1 {{extend 'layout.html'}}
2 <h2>{{=request.args(0)}}</h2>
3 {{=form}}
4 {{if request.args(0)=='login':}}
5 <a href="{{=URL(args='register')}}" >register</a><br />
6 <a href="{{=URL(args='request_reset_password')}}" >lost password</a><br />
7 {{pass}}
```

Observe que esta función simplemente muestra una form, por lo que se puede personalizar mediante la sintaxis normal de formulario personalizado. La única salvedad es que el formulario que se muestra depende del valor de la request.args(0) por lo tanto es posible que necesite si sentencia como esta:

```
1 {{if request.args(0)=='login':}...custom login form...{{pass}}}
```

El controlador anterior expone varias acciones:

```
1 http://.../[app]/default/user/register  
2 http://.../[app]/default/user/login  
3 http://.../[app]/default/user/logout  
4 http://.../[app]/default/user/profile  
5 http://.../[app]/default/user/change_password  
6 http://.../[app]/default/user/verify_email  
7 http://.../[app]/default/user/retrieve_username  
8 http://.../[app]/default/user/request_reset_password  
9 http://.../[app]/default/user/reset_password  
10 http://.../[app]/default/user/impersonate  
11 http://.../[app]/default/user/groups  
12 http://.../[app]/default/user/not_authorized
```

- **register** permite a los usuarios a registrarse. Se integra con el CAPTCHA, aunque esto está desactivado por defecto.
- **login** permite a los usuarios que estén inscritos para iniciar sesión (si el registro se verifica o no tenga necesidad de verificación, si ha sido aprobado o no requiere la aprobación, y si no ha sido bloqueado).
- **logout** hace lo que se puede esperar sino también, como los otros métodos, registra el evento y se puede utilizar para activar algún evento.
- **profile** permite a los usuarios para editar su perfil, es decir, el contenido de la tabla de auth_user. Observe que esta tabla no tiene una estructura fija y se puede personalizar.
- **change_password** permite a los usuarios cambiar sus contraseñas de un modo a prueba de fallos.
- **verify_email**. Si la verificación de correo electrónico está habilitada, los visitantes, registro en el momento, recibir un correo electrónico con un enlace para verificar su información de correo electrónico. El enlace apunta a esta acción.
- **retrieve_username**. De forma predeterminada, el correo electrónico utiliza **Auth** y la contraseña de entrada, pero se puede, opcionalmente, utilizar en lugar del nombre de usuario de correo electrónico. En este último caso, si un usuario olvida su nombre de usuario, el método **retrieve_username** permite que el usuario escriba la dirección de correo electrónico y recuperar el nombre de usuario por correo electrónico.

- **request_reset_password**. Permite a los usuarios que se olvidó de su contraseña para solicitar una nueva contraseña. Ellos recibirán un email de confirmación señalando **reset_password**.
- **impersonate** permite a un usuario "impersonate" otro usuario. Esto es importante para la depuración y para fines de apoyo. `request.args[0]` es el id del usuario a ser suplantado. Esto sólo se permite si el usuario se conectó `has_permission('impersonate', db.auth_user, user_id)`.
- **groups** listas de los grupos de la actual sesión de usuario es miembro de.
- **not_authorized** muestra un mensaje de error cuando el visitante trató de hacer algo que él / ella no está autorizada a hacer
- **navbar** es un auxiliar que genera un bar con nombre de usuario / registro / etc. enlaces.

Cerrar sesión, el perfil, `change_password`, suplantar, y los grupos requieren entrada.

Por defecto están todos expuestos, pero es posible restringir el acceso sólo a una parte de estas acciones.

Todos los métodos anteriores puede ser prorrogado o sustituido por **Auth** subclases.

Para restringir el acceso a las funciones que sólo se conectó visitantes, decorar la función como en el ejemplo siguiente

```
1 @auth.requires_login()
2 def hello():
3     return dict(message='hello %(first_name)!' % auth.user)
```

Cualquier función puede ser decorado, no sólo expone las acciones. Por supuesto, esto sigue siendo sólo un ejemplo muy simple de control de acceso. Ejemplos más complejos se discutirá más adelante.

`auth.user` contiene una copia de los registros `db.auth_user` para el actual usuario registrado o no `None`. También existe también una `auth.user_id`, que es

lo mismo que auth.user.id (es decir, la identificación de la actual registrador de usuario) o None.

8.1.1 Restricciones en materia de registro

Si desea permitir que los visitantes se registren, pero no de diálogo hasta que el registro haya sido aprobado por el administrador:

```
auth.settings.registration_requires_approval = True
```

Puede aprobar una solicitud de registro a través de la interfaz appadmin. Mira en el auth_user mesa. En espera de los registros tienen un campo registration_key establece en "pending". La solicitud de registro es aprobado cuando este campo está en blanco.

A través de la interfaz de appadmin, también puede bloquear a un usuario la tala pulg Localiza al usuario en la tabla de auth_user y registration_key para establecer el "blocked". "blocked" usuarios no se les permite iniciar sesión pulg Tenga en cuenta que esto evitará que un visitante de la tala en pero no será obligar a un visitante que ya está conectado a la sesión.

También puede bloquear el acceso a la página "register" completamente con esta afirmación:

```
auth.settings.actions_disabled.append('register')
```

Otros métodos de **Auth** puede ser restringido de la misma manera.

8.1.2 Integración con OpenID, Facebook, etc

Puede utilizar el papel web2py Base de control de acceso y autenticación con otros servicios como OpenID, Facebook, LinkedIn, Google, MySpace, Flickr, etc La forma más sencilla es utilizar RPX (Janrain.com).

Janrain.com es un servicio que proporciona autenticación de middleware. Puede registrarse en Janrain.com, registrar un dominio (el nombre de su aplicación) y un conjunto de direcciones URL que va a utilizar, y le proporcionará la clave del API.

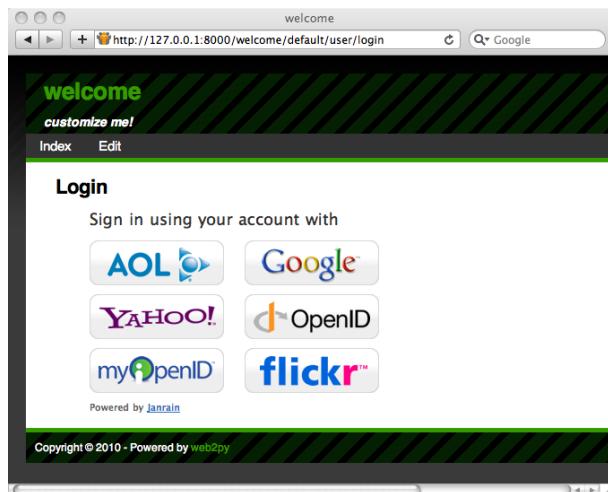
Ahora edite el modelo de su aplicación web2py y en algún lugar después de definir el objeto auth se coloca en los siguientes términos:

```

1 from gluon.contrib.login_methods.rpx_account import RPXAccount
2 auth.settings.actions_disabled=['register','change_password',
3     'request_reset_password']
4 auth.settings.login_form = RPXAccount(request,
5     api_key='...',
6     domain='...',
6 url = "http://localhost:8000/%s/default/user/login" % request.application)

```

Cuando el registro local de las importaciones de primera línea del método nuevo inicio de sesión, la segunda línea con discapacidad, la tercera línea web2py pide utilizar el método entrada RPX. Usted debe insertar su propia api_key proporciona Janrain.com, el dominio que usted elija el momento del registro y la url externa de la página de inicio de sesión.



Cuando un nuevo usuario inicia sesion por primera vez web2py crea

un nuevo registro db.auth_user asociada al usuario. Se utiliza el campo registration_id para almacenar un identificador único para el usuario. La mayoría de los métodos de autenticación también proporcionará un nombre de usuario, correo electrónico, nombre apellido y last_name pero que no está garantizada. ¿Qué campos se proporcionan depende del método de entrada seleccionado por el usuario. Si el mismo usuario el doble de inicios de sesión de uso de diferentes mecanismos de autenticación (por ejemplo una vez con OpenID y una vez con Facebook) JanRain no reconozca su registration_id como el mismo usuario y cuestión diferente.

Puede personalizar la correspondencia entre los datos facilitados por JanRain y los datos almacenados en db.auth_user. He aquí un ejemplo en Facebook:

```

1 auth.settings.login_form.mappings.Facebook = lambda profile:\n2     dict(registration_id = profile["identifier"],\n3          username = profile["preferredUsername"],\n4          email = profile["email"],\n5          first_name = profile["name"]["givenName"],\n6          last_name = profile["name"]["familyName"])

```

Las claves están en el diccionario de campos en db.auth_user y los valores son las entradas de datos en el perfil del objeto proporcionado por JanRain. Mira la documentación JanRain línea para obtener detalles sobre estos últimos.

JanRain también llevar estadísticas sobre entrada de los usuarios.

Esta forma de conexión es completamente integrado con web2py acceso basado en roles de control y todavía se pueden crear grupos, hacer que los miembros de los grupos de usuarios, asignar permisos, bloquear usuarios, etc

Si no desea utilizar el JanRain y desea utilizar un método de acceso diferente (LDAP, PAM, Google, OpenID, OAuth / Facebook, LinkedIn, etc) puede hacerlo. La API para hacerlo se describen más adelante en este capítulo.

8.1.3 CAPTCHA y reCAPTCHA

Para evitar que los spammers y bots registrarse en su sitio, es posible que necesiten un registro CAPTCHA. web2py apoya reCAPTCHA (71) fuera de la caja. Esto se debe a reCAPTCHA está muy bien diseñado, libre y accesible (se puede leer las palabras a los visitantes), fácil de configurar, y no requiere instalar ningún bibliotecas de terceros.

Esto es lo que hay que hacer para usar reCAPTCHA:

- Registrarse en reCAPTCHA (71) y obtener un (PUBLIC_KEY, private_key) Pareja de su cuenta. Estos son sólo dos cuerdas.
- Anexe el código siguiente para su modelo auth después que el objeto se define:

```
1 from gluon.tools import Recaptcha
2 auth.settings.captcha = Recaptcha(request,
3     'PUBLIC_KEY', 'PRIVATE_KEY')
```

reCAPTCHA puede no funcionar si usted accede al sitio web como 'localhost' o "127.0.0.1, porque está registrada para trabajar con sitios web públicamente visibles solamente.

El constructor Recaptcha tiene algunos argumentos opcionales:

```
1 Recaptcha(..., use_ssl=True, error_message='invalid')
```

Tenga en cuenta que use_ssl=False de forma predeterminada.

Si usted no desea usar reCAPTCHA, mira en la definición de la clase en Recaptcha "gluon/tools.py", ya que es fácil de usar otros sistemas de CAPTCHA.

8.1.4 Personalización de la autenticación

El llamado a la

```
1 auth.define_tables()
```

define todas las tablas **Auth** que no se han definido ya. Esto significa que si usted desea hacerlo, puede definir su propia tabla de auth_user. Usando una sintaxis similar a la muestra que aparece a continuación, puede personalizar cualquier tabla **Auth** otros.

Esta es la manera apropiada de definir una tabla de usuario:

```
1 # after
2 # auth = Auth(globals(),db)
3
4 db.define_table(
5     auth.settings.table_user_name,
6     Field('first_name', length=128, default=''),
7     Field('last_name', length=128, default=''),
8     Field('email', length=128, default='', unique=True),
9     Field('password', 'password', length=512,
10           readable=False, label='Password'),
11     Field('registration_key', length=512,
12           writable=False, readable=False, default=''),
13     Field('reset_password_key', length=512,
14           writable=False, readable=False, default=''),
15     Field('registration_id', length=512,
16           writable=False, readable=False, default=''))
17
18 auth_table.first_name.requires = \
19     IS_NOT_EMPTY(error_message=auth.messages.is_empty)
20 auth_table.last_name.requires = \
21     IS_NOT_EMPTY(error_message=auth.messages.is_empty)
22 auth_table.password.requires = [IS_STRONG(), CRYPT()]
23 auth_table.email.requires = [
24     IS_EMAIL(error_message=auth.messages.invalid_email),
25     IS_NOT_IN_DB(db, auth_table.email)]
26 auth.settings.table_user = auth_table
27
28 # before
29 # auth.define_tables()
```

Usted puede agregar cualquier campo que desee, pero no puede quitar los campos obligatorios se muestran en este ejemplo.

Es importante hacer campos "password", "registration_key", "reset_password_key" and "registration_id" `readable=False` y `writable=False`, ya que un visitante no se debe permitir que manipularlos.

Si agrega un campo denominado "username", it will be used in place of "email" de entrada. De ser así, tendrá que añadir un validador así:

```
1 auth_table.username.requires = IS_NOT_IN_DB(db, auth_table.username)
```

8.1.5 Cambiar el nombre de autenticación tablas

Los nombres reales de las tablas se almacenan en Auth

```
1 auth.settings.table_user_name = 'auth_user'
2 auth.settings.table_group_name = 'auth_group'
3 auth.settings.table_membership_name = 'auth_membership'
4 auth.settings.table_permission_name = 'auth_permission'
5 auth.settings.table_event_name = 'auth_event'
```

Los nombres de la tabla pueden ser modificados por la reasignación de las variables mencionados, a partir del objeto auth se define y antes de que la autenticación se definen. Por ejemplo:

```
1 auth = Auth(globals(),db)
2 auth.settings.table_user_name = 'person'
3 ...
4 auth.define_tables()
```

Las tablas reales también se puede hacer referencia, independientemente de sus nombres reales, por

```
1 auth.settings.table_user
2 auth.settings.table_group
3 auth.settings.table_membership
4 auth.settings.table_permission
5 auth.settings.table_event
```

8.1.6 Otros métodos sesión y formularios de ingreso

Proporciona múltiples métodos de autenticación de usuario y ganchos para crear nuevos métodos de entrada. Cada método de entrada el apoyo corresponde a un archivo en la carpeta de

¹ `gluon/contrib/login_methods/`

Consulte la documentación en los propios archivos para cada método de entrada, pero aquí le ofrecemos algunos ejemplos.

En primer lugar tenemos que hacer una distinción entre dos tipos de métodos de entrada alternativo:

- métodos de entrada que utilizan un formulario de acceso web2py (aunque las credenciales son verificadas fuera web2py). Un ejemplo es LDAP.
- métodos que requieren una entrada externa inicio de sesión único en la forma (un ejemplo es Google y Facebook).

En el último caso web2py nunca se las credenciales de inicio de sesión, sólo una entrada token emitido por el proveedor de servicios. El símbolo se almacena en `db.auth_user.registration_id`.

Veamos ejemplos del primer caso:

Básico

Digamos que usted tiene un servicio de autenticación, por ejemplo en la url

¹ `https://basic.example.com`

que acepte la autenticación de acceso básico. Eso significa que el servidor acepta peticiones HTTP con un encabezado de la forma:

¹ `GET /index.html HTTP/1.0`

² `Host: basic.example.com`

`3 Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

donde la cadena este último es la codificación base64 del usuario cadena: contraseña. El servicio responde 200 OK si el usuario está autorizado y 400, 401, 402, 403 o 404 de otro modo.

¿Quieres un nombre de usuario y contraseña en el formulario normalizado entrada Auth y verificar las credenciales en contra de tal servicio. Todo lo que necesitas hacer es agregar el siguiente código a su aplicación

```
1 from gluon.contrib.login_methods.basic_auth import basic_auth
2 auth.settings.login_methods.append(
3     basic_auth('https://basic.example.com'))
```

Tenga en cuenta que `auth.settings.login_methods` es una lista de métodos de autenticación que se ejecutan en forma secuencial. Por defecto está configurado para

```
1 auth.settings.login_methods = [auth]
```

Cuando un método alternativo se anexa, por ejemplo, `basic_auth`, Auth primeros intentos para ingresar en el usuario en función del contenido de `auth_user`, y cuando esto falla, intenta el método siguiente de la lista. Si un método tiene éxito en la explotación forestal en el visitante, y si `auth.settings.login_methods[0]==auth`, Auth realiza las siguientes acciones:

- si el usuario no existe en la `auth_user`, un nuevo usuario se crea y nombre de usuario el correo electrónico / y las contraseñas se almacenan.
- si el usuario existe en `auth_user`, pero aceptó la contraseña nueva no coincide con la antigua contraseña almacenada, la contraseña anterior se sustituye por uno nuevo ("el anuncio de que las contraseñas se almacenan siempre hash menos que se especifique lo contrario).

Si no desea almacenar la nueva contraseña en `auth_user`, entonces es suficiente para cambiar el orden de los métodos de entrada, o eliminar `auth` de la lista. Por ejemplo:

```

1 from gluon.contrib.login_methods.basic_auth import basic_auth
2 auth.settings.login_methods = \
3     [basic_auth('https://basic.example.com')]

```

Lo mismo se aplica para cualquier método de entrada de otros descritos aquí.

SMTP y Gmail

Usted puede comprobar las credenciales de inicio de sesión mediante un servidor SMTP remoto, por ejemplo Gmail, es decir, que se conecte el usuario en si el correo electrónico y la contraseña que proporcionamos son credenciales válidas para acceder al servidor SMTP de Gmail (`smtp.gmail.com:587`). Todo lo que se necesita es el siguiente código:

```

1 from gluon.contrib.login_methods.email_auth import email_auth
2 auth.settings.login_methods.append(
3     email_auth("smtp.gmail.com:587", "@gmail.com"))

```

El primer argumento de la `email_auth` es la dirección: puerto del servidor SMTP. El segundo argumento es el dominio de correo electrónico.

Esto funciona con cualquier servidor SMTP que requiere TLS authentication.

PAM

Autenticación mediante Pluggable Authentication Modules (PAM) funciona como en los casos anteriores. Permite web2py para autenticar a los usuarios utilizar las cuentas de sistema operativo:

```

1 from gluon.contrib.login_methods.pam_auth import pam_auth
2 auth.settings.login_methods.append(pam_auth())

```

LDAP

Autenticación mediante LDAP funciona muy parecido a como en los casos anteriores.

Para utilizar inicio de sesión LDAP con MS Active Directory:

```

1 from gluon.contrib.login_methods.ldap_auth import ldap_auth
2 auth.settings.login_methods.append(ldap_auth(mode='ad',
3     server='my.domain.controller',
4     base_dn='ou=Users,dc=domain,dc=com'))
```

Para utilizar inicio de sesión LDAP con Lotus Notes y Domino:

```

1 auth.settings.login_methods.append(ldap_auth(mode='domino',
2     server='my.domino.server'))
```

Para utilizar inicio de sesión LDAP con OpenLDAP (con UID):

```

1 auth.settings.login_methods.append(ldap_auth(server='my.ldap.server',
2     base_dn='ou=Users,dc=domain,dc=com'))
```

Para utilizar inicio de sesión LDAP con OpenLDAP (con NC):

```

1 auth.settings.login_methods.append(ldap_auth(mode='cn',
2     server='my.ldap.server', base_dn='ou=Users,dc=domain,dc=com'))
```

Google App Engine

Autenticación de Google a través de cuando se ejecuta en Google App Engine requiere saltarse el formulario de acceso web2py, siendo redireccionado a la página de acceso a Google, y de regreso al éxito. Debido a que el comportamiento es diferente que en los ejemplos anteriores, el API es un poco diferente.

```

1 from gluon.contrib.login_methods.gae_google_login import GaeGoogleAccount
2 auth.settings.login_form = GaeGoogleAccount()
```

OpenID

Ya hemos discutido la integración con JanRain (que tiene OpenID apoyo) y que es la forma más fácil de utilizar OpenID. Sin embargo, en algún momento usted no desea contar con un servicio de terceros y desea acceder al proveedor de OpenID directamente en el consumidor (su aplicación).

Aquí está un ejemplo:

```
1 from gluon.contrib.login_methods.openid_auth import OpenIDAuth
2 auth.settings.login_form = OpenIDAuth(auth)
```

OpenIDAuth "python-open" requiere la instalación por separado.

Bajo el capó este método login define la siguiente tabla

```
1 db.define_table('alt_logins',
2     Field('username', length=512, default=''),
3     Field('type', length =128, default='openid', readable=False),
4     Field('user', self.table_user, readable=False))
```

que almacena los nombres de usuario de OpenID para cada usuario. Si desea mostrar el OpenID de la actual sesión de usuario:

```
1 {{=auth.settings.login_form.list_user_openids()}}
```

OAuth2.0 y Facebook

Ya hemos discutido la integración con JanRain (que cuenta con el apoyo de Facebook), pero en algún momento usted no desea contar con un servicio de terceros y desea tener acceso a un proveedor de OAuth2.0 directamente, por ejemplo, Facebook. He aquí cómo:

```
1 from gluon.contrib.login_methods.oauth20_account import OAuthAccount
2 auth.settings.login_form=OAuthAccount(globals(),YOUR_CLIENT_ID,YOUR_CLIENT_SECRET)
```

Las cosas se ponen un poco más complejo si desea utilizar facebook oauth2.0 entrada a no acceder a su propia aplicación, sino en un Facebook específico para acceder a su API. Aquí ofrecemos un ejemplo para acceder a Facebook Gráfico API.

En primer lugar debe instalar el módulo de "pyfacebook".

En segundo lugar, se necesita el siguiente código en su modelo:

```
1 # import required modules
```

```

2 from facebook import GraphAPI
3 from gluon.contrib.login_methods.oauth20_account import OAuthAccount
4 # extend the OAuthAccount class
5 class FaceBookAccount(OAuthAccount):
6     """OAuth impl for FaceBook"""
7     AUTH_URL="https://graph.facebook.com/oauth/authorize"
8     TOKEN_URL="https://graph.facebook.com/oauth/access_token"
9     def __init__(self, g):
10         OAuthAccount.__init__(self, g,
11                               YOUR_CLIENT_ID,
12                               YOUR_CLIENT_SECRET,
13                               self.AUTH_URL,
14                               self.TOKEN_URL)
15         self.graph = None
16     # override function that fetches user info
17     def get_user(self):
18         "Returns the user using the Graph API"
19         if not self.accessToken():
20             return None
21         if not self.graph:
22             self.graph = GraphAPI((self.accessToken()))
23         try:
24             user = self.graph.get_object("me")
25             return dict(first_name = user['first_name'],
26                         last_name = user['last_name'],
27                         username = user['id'])
28         except GraphAPIError:
29             self.session.token = None
30             self.graph = None
31             return None
32     # use the above class to build a new login form
33     auth.settings.login_form=FaceBookAccount(globals())

```

LinkedIn

Ya hemos discutido la integración con JanRain (que cuenta con el apoyo LinkedIn) y que es la forma más fácil de utilizar OpenID. Sin embargo, en algún momento usted no desea contar con un servicio de terceros y desea tener acceso a LinkedIn directamente para obtener más información gracias JanRain proporciona.

Aquí está un ejemplo:

```

1 from gluon.contrib.login_methods.linkedin_account import LinkedInAccount
2 auth.settings.login_form=LinkedInAccount(request,KEY,SECRET,RETURN_URL)

```

LinkedInAccount "python-linkedin" requiere el módulo instalado por separado.

Varias Formas Login

Algunos métodos de entrada modificar el login_form, otros no. Cuando lo hacen, puede que no sean capaces de coexistir. Sin embargo, algunos conviven algunos proporcionando múltiples formas de acceso en la misma página. web2py proporciona una manera de hacerlo. Aquí está un ejemplo mezclar entrada normal (autenticación) y entrada RPX (janrain.com):

```

1 from gluon.contrib.login_methods.extended_login_form import ExtendedLoginForm
2 other_form = RPXAccount(request, api_key='...', domain='...', url='...')
3 auth.settings.login_form = ExtendedLoginForm(request,
4     auth, other_form, signals=['token'])

```

Si las señales se fijen y un parámetro en la solicitud coincide con ninguna señal, se le devolverá la llamada de la other_form.login_form en su lugar. other_form puede manejar algunas situaciones particulares, por ejemplo, varios pasos de la entrada en el interior OpenID other_form.login_form. De lo contrario va a proporcionar la forma de entrada normal, así como el other_form.

8.2 Auth y correo electrónico

De forma predeterminada, la verificación de correo electrónico está deshabilitada. Para habilitar el correo electrónico, añadir las siguientes líneas en el modelo auth, donde se define:

```

1 from gluon.tools import Mail
2 mail = Mail(globals())
3 mail.settings.server = 'smtp.example.com:25'
4 mail.settings.sender = 'you@example.com'
5 mail.settings.login = 'username:password'
6 auth.settings.mailer = mail
7 auth.settings.registration_requires_verification = False
8 auth.settings.registration_requires_approval = False

```

```

9 auth.settings.reset_password_requires_verification = True
10 auth.messages.verify_email = 'Click on the link http://' + \
11     request.env.http_host + \
12     URL(r=request,c='default',f='user',args=['verify_email']) + \
13     '/%(key)s to verify your email'
14 auth.messages.reset_password = 'Click on the link http://' + \
15     request.env.http_host + \
16     URL(r=request,c='default',f='user',args=['reset_password']) + \
17     '/%(key)s to reset your password'
```

Es necesario sustituir el mail.settings con los parámetros adecuados para su servidor SMTP. Establecer `mail.settings.login=False` si el servidor SMTP no requiere autenticación.

También es necesario reemplazar la cadena

```
1 'Click on the link ...'
```

en `auth.messages.verify_email` con la dirección correcta completa de la `verify_email` acción. Esto es necesario porque web2py se puede instalar detrás de un proxy, y no se puede determinar su propia URL pública con absoluta certeza.

Una vez `mail` está definida, también se puede utilizar para enviar correo electrónico a través explícitamente

```

1 mail.send(to=['somebody@example.com'],
2           subject='hello',
3           message='hi there')
```

Mail devuelve True si tiene éxito en el envío del correo electrónico, de lo contrario falso.

8.2.1 Correo electrónico de depuración

Para propósitos de depuración puede establecer

```
1 mail.settings.server = 'logging'
```

y mensajes de correo electrónico no se enviará a la consola, pero registra su lugar.

8.2.2 Mensajes de correo electrónico forma Google App Engine

Para envío de correos electrónicos de Google App Engine en cuenta:

```
1 mail.settings.server = 'gae'
```

En el momento de redactar web2py no es compatible con archivos adjuntos y mensajes de correo electrónico cifrado en Google App Engine.

8.2.3 Más ejemplos de correo electrónico

Simple correo electrónico de texto

```
1 mail.send('you@example.com',
2   'Message subject',
3   'Plain text body of the message')
```

Correo electrónico HTML

```
1 mail.send('you@example.com',
2   'Message subject',
3   '<html>html body</html>')
```

Si el cuerpo del mensaje comienza con `<html>` y termina con la `</html>` que se enviará en un correo electrónico HTML.

La combinación de texto y mensajes de correo electrónico HTML

El mensaje de correo electrónico puede ser una tupla (texto, html):

```
1 mail.send('you@example.com',
2   'Message subject',
3   ('Plain text body', '<html>html body</html>'))
```

CC y BCC mensajes de correo electrónico

```

1 mail.send('you@example.com',
2   'Message subject',
3   'Plain text body',
4   cc=['other1@example.com', 'other2@example.com'],
5   bcc=['other3@example.com', 'other4@example.com'])

```

Adjuntos

```

1 mail.send('you@example.com',
2   'Message subject',
3   '<html></html>',
4   attachments = Mail.Attachment('/path/to/photo.jpg' content_id='photo'))

```

Múltiples accesorios

```

1 mail.send('you@example.com',
2   'Message subject',
3   'Message body',
4   attachments = [Mail.Attachment('/path/to/fist.file'),
5                 Mail.Attachment('/path/to/second.file')])

```

8.2.4 x509 y el cifrado PGP

Es posible enviar x509 (SMIME) cifrado mensajes de correo electrónico usando los ajustes siguientes:

```

1 mail.settings.cipher_type = 'x509'
2 mail.settings.sign = True
3 mail.settings.sign_passphrase = 'your passphrase'
4 mail.settings.encrypt = True
5 mail.settings.x509_sign_keyfile = 'filename.key'
6 mail.settings.x509_sign_certfile = 'filename.cert'
7 mail.settings.x509_crypt_certfiles = 'filename.cert'

```

Es posible enviar mensajes de correo electrónico PGP cifrados usando los siguientes ajustes:

```

1 from gpgme import pgp
2 mail.settings.cipher_type = 'gpg'

```

```

3 mail.settings.sign = True
4 mail.settings.sign_passphrase = 'your passphrase'
5 mail.settings.encrypt = True

```

Esto último requiere el paquete python-pyme.

8.3 Autorización

Una vez que un nuevo usuario se registra, un nuevo grupo es creado para contener el usuario. El papel del nuevo usuario es convencionalmente "user_[id]" donde [id] es el identificador de la recién creada id. La creación del grupo se puede desactivar con

```

1 auth.settings.create_user_groups = False

```

si bien no sugerimos hacerlo.

Los usuarios tienen la pertenencia a grupos. Cada grupo se identifica por un nombre o un papel. Los grupos tienen permisos. Los usuarios tienen permisos porque de los grupos que pertenecen.

Puede crear grupos, dar permisos a través de la adhesión y **appadmin** o mediante programación usando los siguientes métodos:

```

1 auth.add_group('role', 'description')

```

devuelve el identificador del grupo recién creado.

```

1 auth.del_group(group_id)

```

elimina el grupo con **group_id**.

```

1 auth.del_group(auth.id_group('user_7'))

```

elimina el grupo con "user_7" papel, es decir, el grupo asociado únicamente a 7 el número de usuario.

```
1 auth.user_group(user_id)
```

devuelve el id del grupo asociado únicamente al usuario identificado por user_id.

```
1 auth.add_membership(group_id, user_id)
```

da la adhesión de la user_id group_id grupo. Si la user_id no se especifica, entonces web2py asume el actual usuario registrado.

```
1 auth.del_membership(group_id, user_id)
```

user_id revoca la adhesión de la group_id grupo. Si la user_id no se especifica, entonces web2py asume el actual usuario registrado.

```
1 auth.has_membership(group_id, user_id, role)
```

comprueba si ha user_id de miembros del grupo de group_id o el grupo con la función especificada. Sólo group_id o role se pasa a la función, pero no ambos. Si la user_id no se especifica, entonces web2py asume el actual usuario registrado.

```
1 auth.add_permission(group_id, 'name', 'object', record_id)
```

da permiso "name" (user defined) on the object "object" (también definida por el usuario) a los miembros del grupo de group_id. Si "object" es un nombre de tabla a continuación, el permiso puede referirse a toda la tabla (record_id==0) o a un registro específico (record_id>0). Cuando los permisos que da sobre las tablas, es común el uso de un nombre de permiso en el conjunto ('crear', 'read', la 'actualización', 'borrar', 'select'), ya que estos permisos se entienden y pueden imponerse por CRUD.

```
1 auth.del_permission(group_id, 'name', 'object', record_id)
```

revoca el permiso.

```
1 auth.has_permission('name', 'object', record_id, user_id)
```

comprueba si el usuario identificado por `user_id` tiene la pertenencia a un grupo con el permiso solicitado.

```
1 rows = db(accessible_query('read', db.sometable, user_id))\
2     .select(db.mytable.ALL)
```

devuelve todas las filas de la tabla que "sometable" `user_id` "read" usuario tiene el permiso de. Si la `user_id` no es especificado, entonces web2py asume el actual usuario registrado. El `accessible_query(...)` se puede combinar con otros para hacer consultas más complejas. `accessible_query(...)` es el único **Auth** método que requiere una combinación, por lo que no funciona en el motor de Google App.

Asumiendo las siguientes definiciones:

```
1 >>> from gluon.tools import Auth
2 >>> auth = Auth(globals(), db)
3 >>> auth.define_tables()
4 >>> secrets = db.define_table('document', Field('body'))
5 >>> james_bond = db.auth_user.insert(first_name='James',
6                                         last_name='Bond')
```

Aquí está un ejemplo:

```
1 >>> doc_id = db.document.insert(body = 'top secret')
2 >>> agents = auth.add_group(role = 'Secret Agent')
3 >>> auth.add_membership(agents, james_bond)
4 >>> auth.add_permission(agents, 'read', secrets)
5 >>> print auth.has_permission('read', secrets, doc_id, james_bond)
6 True
7 >>> print auth.has_permission('update', secrets, doc_id, james_bond)
8 False
```

8.3.1 Decoradores

La manera más común para verificar el permiso no es por llamadas explícitas a los métodos anteriores, sino por las funciones de la decoración para que los permisos se comprueban en relación con el registrado en el visitante. He aquí algunos ejemplos:

```

1 def function_one():
2     return 'this is a public function'
3
4 @auth.requires_login()
5 def function_two():
6     return 'this requires login'
7
8 @auth.requires_membership(agents)
9 def function_three():
10    return 'you are a secret agent'
11
12 @auth.requires_permission('read', secrets)
13 def function_four():
14    return 'you can read secret documents'
15
16 @auth.requires_permission('delete', 'any file')
17 def function_five():
18     import os
19     for file in os.listdir('./'):
20         os.unlink(file)
21     return 'all files deleted'
22
23 @auth.requires(auth.user_id==1 or request.client=='127.0.0.1')
24 def function_six():
25    return 'you can read secret documents'
26
27 @auth.requires_permission('add', 'number')
28 def add(a, b):
29    return a + b
30
31 def function_six():
32    return add(3, 4)

```

Tenga en cuenta que el acceso a todas las funciones no sólo en el primero se limita función de los permisos que el usuario puede o no tener.

Si el visitante no está conectado, entonces el permiso no se puede comprobar, el visitante es redirigido a la página de registro y luego de vuelta a la página que requiere permisos.

Si el visitante no tiene permiso para acceder a una determinada función, el visitante se redirigen a la dirección definida por

```

1 auth.settings.on_failed_authorization = \
2     URL('user',args='on_failed_authorization')

```

Puede cambiar esta variable y redirigir al usuario a otra parte.

8.3.2 La combinación de Requerimientos

En ocasiones, es necesario combinar los requisitos. Esto se puede hacer a través de un decorador de genéricos `requires` que tiene un solo argumento, una condición verdadera o falsa. Por ejemplo, para dar acceso a los agentes, pero sólo el martes:

```

1 @auth.requires(auth.has_membership(group_id=agents) \
2                 and request.now.weekday()==1)
3 def function_seven():
4     return 'Hello agent, it must be Tuesday!'

```

o equivalente:

```

1 @auth.requires(auth.has_membership(role='Secret Agent') \
2                 and request.now.weekday()==1)
3 def function_seven():
4     return 'Hello agent, it must be Tuesday!'

```

8.3.3 Autorización y CRUD

Uso de los decoradores y / o controles explícitos proporciona una manera de implementar el control de acceso.

Otra forma de implementar el control de acceso es utilizar siempre CRUD (a diferencia de SQLFORM) para acceder a la base de datos y pedir CRUD para hacer cumplir el control de acceso en las tablas de bases de datos y registros. Esto se hace mediante la vinculación de Auth y CRUD con la siguiente afirmación:

```

1 crud.settings.auth = auth

```

Esto evitara que el visitante el acceso a cualquiera de las funciones CRUD menos que el visitante se registra y tiene acceso explícito. Por ejemplo, para permitir a un usuario para enviar comentarios, pero sólo actualizar sus propios comentarios (suponiendo porquería, autoridades y db.comment se definen):

```

1 def give_create_permission(form):
2     group_id = auth.id_group('user_%s' % auth.user.id)
3     auth.add_permission(group_id, 'read', db.comment)
4     auth.add_permission(group_id, 'create', db.comment)
5     auth.add_permission(group_id, 'select', db.comment)
6
7 def give_update_permission(form):
8     comment_id = form.vars.id
9     group_id = auth.id_group('user_%s' % auth.user.id)
10    auth.add_permission(group_id, 'update', db.comment, comment_id)
11    auth.add_permission(group_id, 'delete', db.comment, comment_id)
12
13 auth.settings.register_onaccept = give_create_permission
14 crud.settings.auth = auth
15
16 def post_comment():
17     form = crud.create(db.comment, onaccept=give_update_permission)
18     comments = db(db.comment.id>0).select()
19     return dict(form=form, comments=comments)
20
21 def update_comment():
22     form = crud.update(db.comment, request.args(0))
23     return dict(form=form)

```

También puede seleccionar registros específicos (los que ha 'leído' el acceso a):

```

1 def post_comment():
2     form = crud.create(db.comment, onaccept=give_update_permission)
3     query = auth.accessible_query('read', db.comment, auth.user.id)
4     comments = db(query).select(db.comment.ALL)
5     return dict(form=form, comments=comments)

```

Los nombres permisiones forzada por:

```
1 crud.settings.auth = auth
```

son "read", "create", "update", "delete", "select", "impersonate".

8.3.4 Autorización y descargas

El uso de los decoradores y el uso de crud.settings.auth no hacen cumplir la autorización de los archivos descargados por el

```
1 def download(): return response.download(request, db)
```

descarga habitual función

Si uno lo desea, se debe declarar explícitamente qué campos "upload" contienen los archivos que necesitan de control de acceso a la descarga. Por ejemplo:

```
1 db.define_table('dog',
2     Field('small_image', 'upload')
3     Field('large_image', 'upload'))
4
5 db.dog.large_image.authorization = lambda record: \
6     auth.is_logged_in() and \
7     auth.has_permission('read', db.dog, record.id, auth.user.id)
```

El atributo de campo authorization carga puede ser None (por defecto) o una función que decide si el usuario está conectado y tiene permiso para "leer" el registro actual. En este ejemplo, no hay ninguna restricción a la descarga de imágenes vinculadas por el campo "small_image" field, but we require access control on images linked by the "large_image".

8.3.5 Control de acceso y autenticación básica

En ocasiones, puede ser necesario para exponer las acciones que han decoradores que requieren control de acceso a los servicios, es decir, para llamar desde un programa o script y seguir siendo capaz de utilizar la autenticación para verificar la autorización.

Auth permite entrada a través de la autenticación básica:

```
1 auth.settings.allow_basic_login = False
```

Con este conjunto, una acción como

```

1 @auth.requires_login()
2 def give_me_time():
3     import time
4     return time.ctime()
```

se puede llamar, por ejemplo, de una orden de shell:

```

1 wget --user=[username] --password=[password]
2 http://.../[app]/[controller]/give_me_time
```

entrada básica es a menudo la única opción para los servicios (que se describe en el capítulo siguiente), pero está desactivado por defecto.

8.3.6 Configuración y mensajes

Aquí está una lista de todos los parámetros que se pueden personalizar para **Auth**

A continuación debe apuntar a un objeto para permitir gluon.tools.Mail auth para enviar mensajes de correo electrónico:

```
1 auth.settings.mailer = None
```

Lo siguiente debe ser el nombre del controlador, que define la acción user:

```
1 auth.settings.controller = 'default'
```

El siguiente es un escenario muy importante:

```
1 auth.settings.hmac_key = None
```

Se debe establecer en algo así como campo de la tabla "sha512:a-pass-phrase" and it will be passed to the CRYPT validator for the "password" auth_user. Será el algoritmo y una frase de paso-utilizados para el hash de la contraseña.

Para agregar una acción con discapacidad su nombre a esta lista:

```
1 auth.settings.actions_disabled = []
```

Por ejemplo:

```
1 auth.settings.actions_disabled.append('register')
```

se registro deshabilitar.

Si desea recibir un correo electrónico para verificar el registro de establecer esta True:

```
1 auth.settings.registration_requires_verification = False
```

Si los nuevos registrados se debe permitir a la conexión hasta que sea aprobado establecer este a True:

```
1 auth.settings.registration_requires_approval = False
```

Aprobación consiste en establecer a través de appadmin registration_key==" o mediante programación.

Si usted no quiere un nuevo grupo para cada usuario nuevo conjunto de los siguientes campos para False:

```
1 auth.settings.create_user_groups = True
```

Las siguientes opciones determinan los métodos alternativos de acceso y formularios de ingreso, como se discutió previamente:

```
1 auth.settings.login_methods = [auth]
2 auth.settings.login_form = auth
```

¿Quieres permitir el acceso de base?

```
1 auth.settings.allows_basic_login = False
```

La siguiente es la dirección de la acción login:

```
1 auth.settings.login_url = URL('user', args='login')
```

Si el usuario ha intentado acceder a la página de registro, pero se ha identificado, el que será remitido a la siguiente dirección URL:

```
1 auth.settings.logged_url = URL('user', args='profile')
```

Esto debe apuntar a la dirección de la acción de descarga, en caso de que el perfil contiene imágenes:

```
1 auth.settings.download_url = URL('download')
```

Estos deben apuntar a la URL que desea redirigir a los usuarios que después de las diversas acciones posibles auth (en caso de que no hay referencia):

```
1 auth.settings.login_next = URL('index')
2 auth.settings.logout_next = URL('index')
3 auth.settings.profile_next = URL('index')
4 auth.settings.register_next = URL('user', args='login')
5 auth.settings.retrieve_username_next = URL('index')
6 auth.settings.retrieve_password_next = URL('index')
7 auth.settings.change_password_next = URL('index')
8 auth.settings.request_reset_password_next = URL('user', args='login')
9 auth.settings.reset_password_next = URL('user', args='login')
10 auth.settings.verify_email_next = URL('user', args='login')
```

Esta es la URL que desea que se redirigir a la autorización de error:

```
1 auth.settings.on_failed_authorization =
2     URL('user',args='on_failed_authorization')
```

Estas son listas de devoluciones de llamada que debe ser ejecutado después la validación de formularios para cada una de las acciones correspondientes ante cualquier base de datos de E / S:

```
1 auth.settings.login_onvalidation = []
2 auth.settings.register_onvalidation = []
3 auth.settings.profile_onvalidation = []
4 auth.settings.retrieve_password_onvalidation = []
5 auth.settings.reset_password_onvalidation = []
```

Cada devolución de llamada debe ser una función que toma el objeto `form` y puede modificar los atributos del objeto de formulario antes de base de datos de E / S se lleva a cabo.

Estas son listas de devoluciones de llamada que debe ser ejecutado después de la base de datos de E / S se lleva a cabo y antes de redirección:

```

1 auth.settings.login_onaccept = []
2 auth.settings.register_onaccept = []
3 auth.settings.profile_onaccept = []
4 auth.settings.verify_email_onaccept = []

```

Aquí está un ejemplo:

```

1 auth.settings.register_onaccept.append(lambda form:\n2     mail.send(to='you@example.com',subject='new user',\n3     message="new user email is %s"%form.vars.email))

```

Puede habilitar Capcha para cualquiera de las acciones auth:

```

1 auth.settings.captcha = None
2 auth.settings.login_captcha = None
3 auth.settings.register_captcha = None
4 auth.settings.retrieve_username_captcha = None
5 auth.settings.retrieve_password_captcha = None

```

Si la configuración de puntos .captcha gluon.tools.Recaptcha a una, todas las formas para las que la opción correspondiente (por ejemplo, .login_captcha) se establece en `None` tendrá un capitán, mientras que aquellos para los que la opción correspondiente se establece que no se `False`. En cambio, si se establece en .captcha `None`, sólo aquellos que tienen una forma correspondiente opción establecida en un objeto gluon.tools.Recaptcha tendrá Catcha y las otras no lo harán.

Esta es la sesión de inicio fecha de caducidad:

```

1 auth.settings.expiration = 3600 # seconds

```

Puede cambiar el nombre del campo de la contraseña (en Firebrid para "password" ejemplo es una palabra clave y no se puede utilizar para nombrar un campo):

```

1 auth.settings.password_field = 'password'

```

Normalmente, el formulario de acceso intenta validar un correo electrónico. Esto se puede desactivar cambiando esta configuración:

```
1 auth.settings.login_email_validate = True
```

¿Desea mostrar los ID de registro en la página de edición de perfil?

```
1 auth.settings.showid = False
```

Para los formularios personalizados que lo desea, puede desactivar la notificación automática de errores en las formas:

```
1 auth.settings.hideerror = False
```

También para los formularios personalizados que puede cambiar el estilo:

```
1 auth.settings.formstyle = 'table3cols'
```

(Puede ser "table2cols", "divs" and "ul")

Por defecto, el formulario de acceso da la opción de ampliar el inicio de sesión mediante la opción "remember me". La fecha de caducidad se puede cambiar ni la opción de desactivar a través de los siguientes valores:

```
1 auth.settings.long_expiration = 3600*30 # one month
2 auth.settings.remember_me_form = True
```

También puede personalizar los siguientes mensajes cuyo uso y el contexto debería ser obvio: la adhesión

```
1 auth.messages.submit_button = 'Submit'
2 auth.messages.verify_password = 'Verify Password'
3 auth.messages.delete_label = 'Check to delete:'
4 auth.messages.function_disabled = 'Function disabled'
5 auth.messages.access_denied = 'Insufficient privileges'
6 auth.messages.registration_verifying = 'Registration needs verification'
7 auth.messages.registration_pending = 'Registration is pending approval'
8 auth.messages.login_disabled = 'Login disabled by administrator'
9 auth.messages.logged_in = 'Logged in'
10 auth.messages.email_sent = 'Email sent'
11 auth.messages.unable_to_send_email = 'Unable to send email'
12 auth.messages.email_verified = 'Email verified'
```

```
13 auth.messages.logged_out = 'Logged out'
14 auth.messages.registration_successful = 'Registration successful'
15 auth.messages.invalid_email = 'Invalid email'
16 auth.messages.unable_send_email = 'Unable to send email'
17 auth.messages.invalid_login = 'Invalid login'
18 auth.messages.invalid_user = 'Invalid user'
19 auth.messages.is_empty = "Cannot be empty"
20 auth.messages.mismatched_password = "Password fields don't match"
21 auth.messages.verify_email = ...
22 auth.messages.verify_email_subject = 'Password verify'
23 auth.messages.username_sent = 'Your username was emailed to you'
24 auth.messages.new_password_sent = 'A new password was emailed to you'
25 auth.messages.password_changed = 'Password changed'
26 auth.messages.retrieve_username = 'Your username is: %(username)s'
27 auth.messages.retrieve_username_subject = 'Username retrieve'
28 auth.messages.retrieve_password = 'Your password is: %(password)s'
29 auth.messages.retrieve_password_subject = 'Password retrieve'
30 auth.messages.reset_password = ...
31 auth.messages.reset_password_subject = 'Password reset'
32 auth.messages.invalid_reset_password = 'Invalid reset password'
33 auth.messages.profile_updated = 'Profile updated'
34 auth.messages.new_password = 'New password'
35 auth.messages.old_password = 'Old password'
36 auth.messages.group_description = \
    'Group uniquely assigned to user %(id)s'
37 auth.messages.register_log = 'User %(id)s Registered'
38 auth.messages.login_log = 'User %(id)s Logged-in'
39 auth.messages.logout_log = 'User %(id)s Logged-out'
40 auth.messages.profile_log = 'User %(id)s Profile updated'
41 auth.messages.verify_email_log = 'User %(id)s Verification email sent'
42 auth.messages.retrieve_username_log = 'User %(id)s Username retrieved'
43 auth.messages.retrieve_password_log = 'User %(id)s Password retrieved'
44 auth.messages.reset_password_log = 'User %(id)s Password reset'
45 auth.messages.change_password_log = 'User %(id)s Password changed'
46 auth.messages.add_group_log = 'Group %(group_id)s created'
47 auth.messages.del_group_log = 'Group %(group_id)s deleted'
48 auth.messages.add_membership_log = None
49 auth.messages.del_membership_log = None
50 auth.messages.has_membership_log = None
51 auth.messages.add_permission_log = None
52 auth.messages.del_permission_log = None
53 auth.messages.has_permission_log = None
54 auth.messages.label_first_name = 'First name'
55 auth.messages.label_last_name = 'Last name'
56 auth.messages.label_username = 'Username'
57 auth.messages.label_email = 'E-mail'
58 auth.messages.label_password = 'Password'
59 auth.messages.label_registration_key = 'Registration key'
60 auth.messages.label_reset_password_key = 'Reset Password key'
```

```

62 auth.messages.label_registration_id = 'Registration identifier'
63 auth.messages.label_role = 'Role'
64 auth.messages.label_description = 'Description'
65 auth.messages.label_user_id = 'User ID'
66 auth.messages.label_group_id = 'Group ID'
67 auth.messages.label_name = 'Name'
68 auth.messages.label_table_name = 'Table name'
69 auth.messages.label_record_id = 'Record ID'
70 auth.messages.label_time_stamp = 'Timestamp'
71 auth.messages.label_client_ip = 'Client IP'
72 auth.messages.label_origin = 'Origin'
73 auth.messages.label_remember_me = "Remember me (for 30 days)"

```

add|del|has registros de permitir el uso de "%(user_id)s" and "%(group_id)s". add|del|has registros permiso permite el uso de "%(user_id)s", "%(name)s", "%(table_name)s", and "%(record_id)s".

8.4 Servicio Central de autenticación

web2py proporciona soporte para la autenticación y autorización a través de aparatos. Aquí discutimos el aparato para **cas** Central Authentication Service (CAS). Observe que en el momento de escribir CAS es distinto y no funciona con **Auth**. Esto cambiará en el futuro.

CAS es un protocolo abierto para la autenticación distribuida y funciona de la siguiente manera: Cuando un visitante llega a nuestro sitio web, nuestro cheque aplicación en el período de sesiones si el usuario ya está autenticado (por ejemplo a través de un objeto session.token). Si el usuario no está autenticado, el controlador vuelve a dirigir al visitante desde el aparato del CAS, donde el usuario puede acceder, registrar y administrar sus credenciales (nombre, correo electrónico y contraseña). Si el usuario se registra, recibe un correo electrónico, y el registro no está completo hasta que se responde al correo electrónico. Una vez que el usuario ha registrado y conectado, el aparato CAS redirige al usuario a nuestra aplicación junto con una clave. Nuestra aplicación utiliza la clave para obtener las credenciales del usuario a través de una solicitud HTTP en segundo plano al servidor CAS.

El uso de este mecanismo, varias aplicaciones pueden utilizar el un inicio de sesión único a través de un único servidor CAS. El servidor proporciona autenticación se llama un proveedor de servicios. Las solicitudes para autenticar la búsqueda de los visitantes son llamados consumidores de servicios.

CAS es similar a OpenID, con una diferencia principal. En el caso de OpenID, el visitante elige el proveedor de servicios. En el caso de CAS, nuestra aplicación hace esta elección, haciendo más seguro CAS.

Puede ejecutar sólo el consumidor, el proveedor, o ambos (en una sola aplicación o separadas).

Para ejecutar CAS como los consumidores se debe descargar el archivo:

[1 https://www.web2py.com/cas/static/cas.py](https://www.web2py.com/cas/static/cas.py)

y guardarla como un archivo llamado modelo "cas.py". Then you must edit the controllers that need authentication (for example "default.py") y, en la parte superior, añada el siguiente código:

```

1 CAS.login_url='https://www.web2py.com/cas/cas/login'
2 CAS.check_url='https://www.web2py.com/cas/cas/check'
3 CAS.logout_url='https://www.web2py.com/cas/cas/logout'
4 CAS.my_url='http://127.0.0.1:8000/myapp/default/login'

5
6 if not session.token and not request.function=='login':
7     redirect(URL('login'))
8 def login():
9     session.token=CAS.login(request)
10    id,email,name=session.token
11    return dict()
12 def logout():
13    session.token=None
14    CAS.logout()
```

Debe editar los atributos del objeto por encima de CAS. De forma predeterminada, apuntan al proveedor de CAS que se ejecuta en "https://mdp.cti.depaul.edu". Ofrecemos este servicio principalmente para propósitos de prueba. El `CAS.my_url` tiene que ser la URL completa de la acción de acceso definidos en su aplicación y se muestra en el código. El proveedor de las necesidades de CAS

para redirigir el navegador a esta acción.

Nuestro proveedor de CAS devuelve una que contiene un token de tupla (id, correo electrónico, nombre), donde id es el registro único de identificación del visitante (según sean asignados por la base de datos del proveedor), el correo electrónico es la dirección de correo electrónico del visitante (según lo declarado por el visitante al proveedor y verificada por el proveedor), y el nombre es el nombre del visitante (que es elegida por el visitante y no hay garantía de que es un nombre real).

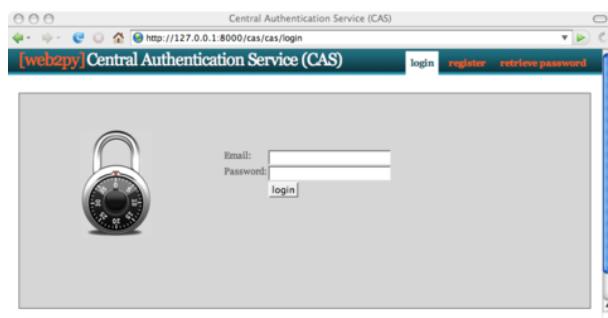
Si usted visita la URL local:

```
1 /myapp/default/login
```

te redirige a la página de acceso del CAS:

```
1 https://mdp.cti.depaul.edu/cas/cas/login
```

que se parece a esto:



También puede utilizar los servicios de CAS de terceros, pero usted puede necesitar para editar la línea 10, desde diferentes proveedores de CAS puede devolver fichas que contienen valores diferentes. Consulte la documentación del servicio CAS necesita tener acceso a los detalles. La mayoría de servicios sólo devuelven (id, nombre de usuario).

Después de una conexión exitosa, se le redirige a la acción de acceso local. El punto de vista de la acción de acceso local se ejecuta sólo después de iniciar sesión correctamente CAS.

Puede descargar el aparato proveedor de CAS a ref. (33) y ejecutar por sí mismo. Si decide hacerlo, también debe modificar las primeras líneas del modelo "email.py" en el aparato, de manera que apunte a su servidor SMPT.

También puede combinar los archivos del proveedor de equipo CAS proveedor con los de su aplicación (modelos bajo modelos, etc) siempre y no hay conflicto nombre de archivo.

9

Servicios

El W3C define un servicio web como "a software system designed to support interoperable machine-to-machine interaction over a network". Esta es una definición amplia, y abarca un gran número de protocolos que no están diseñados para la comunicación de máquina a humanos, pero para la comunicación de máquina a máquina, como XML, JSON, RSS, etc web2py establece, fuera de la caja, el apoyo a los protocolos, incluyendo XML, JSON, RSS, CSV, XMLRPC, JSONRPC, AMFRPC y SOAP. web2py también puede ser extendido para soportar protocolos adicionales.

Cada uno de estos protocolos con el apoyo de múltiples maneras, y hacemos una distinción entre:

- Representación de la salida de una función en un formato determinado (por ejemplo, XML, JSON, RSS, CSV)
- Remote Procedure Calls (por ejemplo, XMLRPC, JSONRPC, AMFRPC)

9.1 Representación de un diccionario

9.1.1 HTML, XML y JSON

Considere las siguientes actuaciones:

```

1 def count():
2     session.counter = (session.counter or 0) + 1
3     return dict(counter=session.counter, now=request.now)

```

Esta acción devuelve un contador que se incrementa en uno cuando un visitante vuelve a cargar la página y la fecha y hora de la solicitud de página actual.

Normalmente esta página se solicitará a través de:

```
1 http://127.0.0.1:8000/app/default/count
```

y prestados en HTML. Sin escribir una sola línea de código, podemos pre-guntar web2py para que esta página utilizando una protocolos diferentes, añadiendo una extensión a la dirección:

```

1 http://127.0.0.1:8000/app/default/count.html
2 http://127.0.0.1:8000/app/default/count.xml
3 http://127.0.0.1:8000/app/default/count.json

```

El diccionario devuelto por la acción se hizo en HTML, XML y JSON, respec-tivamente.

Aquí está la salida XML:

```

1 <document>
2     <counter>3</counter>
3     <now>2009-08-01 13:00:00</now>
4 </document>

```

Aquí está la salida JSON:

```
1 { 'counter':3, 'now':'2009-08-01 13:00:00' }
```

Tenga en cuenta que la fecha, hora, fecha y hora y los objetos se representan como cadenas en formato ISO. Esto no es parte del estándar de JSON, sino una convención web2py.

9.1.2 Genérico Vistas

Cuando, por ejemplo, la ".xml" extension is called, web2py looks for a template file called "default/count.xml", and if it does not find it, web2py looks for a template called "generic.xml". The files "generic.html", "generic.xml", "generic.json" se proporcionan con la aplicación de scaffolding actual.

Otras extensiones pueden ser fácilmente definidos por el usuario.

Nada de lo que hay que hacer para que esto en una aplicación web2py. Para usarlo en una aplicación web2py mayores de esa edad, puede que tenga que copiar el "genérico.*" los archivos de una aplicación más tarde andamios (después de la versión 1.60).

Aquí está el código para "generic.html"

```

1 {{extend 'layout.html'}}
2
3 {{=BEAUTIFY(response._vars)}}
4
5 <button onclick="document.location='{{=URL("admin","default","design",
6 args=request.application)}}'">admin</button>
7 <button onclick="jQuery('#request').slideToggle()">request</button>
8 <div class="hidden" id="request"><h2>request</h2>{{=BEAUTIFY(request)}}</div>
9 <button onclick="jQuery('#session').slideToggle()">session</button>
10 <div class="hidden" id="session"><h2>session</h2>{{=BEAUTIFY(session)}}</div>
11 <button onclick="jQuery('#response').slideToggle()">response</button>
12 <div class="hidden" id="response"><h2>response</h2>{{=BEAUTIFY(response)}}</div>
13 <script>jQuery('.hidden').hide();</script>
```

Aquí está el código para "generic.xml"

```

1 {{{
2     try:
3         from gluon.serializers import xml
```

```

4     response.write(xml(response._vars), escape=False)
5     response.headers['Content-Type'] = 'text/xml'
6 except:
7     raise HTTP(405, 'no xml')
8 }}
```

Y aquí está el código para "generic.json"

```

1 {{{
2 try:
3     from gluon.serializers import json
4     response.write(json(response._vars), escape=False)
5     response.headers['Content-Type'] = 'text/json'
6 except:
7     raise HTTP(405, 'no json')
8 }}}
```

Cada diccionario puede ser dictada en HTML, XML y JSON, siempre y cuando sólo contiene python tipos primitivos (int, float cadena, lista, tupla, diccionario). `response._vars` contiene el diccionario devuelto por la acción.

Si el diccionario contiene otros objetos definidos por el usuario o web2py específico, deben ser prestados por una vista personalizada.

9.1.3 Representación de Filas

Si usted necesita para hacer un conjunto de filas devuelto por un selecto en XML o JSON u otro formato, en primer lugar transformar el objeto en las filas de una lista de diccionarios utilizando el método de `as_list()`.

Consideremos por ejemplo el modo siguiente:

```
1 db.define_table('person', Field('name'))
```

La siguiente acción se puede representar en HTML, pero no en XML o JSON:

```

1 def everybody():
2     people = db().select(db.person.ALL)
3     return dict(people=people)
```

Si bien las siguientes medidas pueden dictada en XML y JSON:

```

1 def everybody():
2     people = db().select(db.person.ALL).as_list()
3     return dict(people=people)

```

9.1.4 Formatos personalizados

Si, por ejemplo, usted quiere hacer una acción como pickle de Python:

```
1 http://127.0.0.1:8000/app/default/count.pickle
```

sólo tiene que crear una vista "default/count.pickle" nuevo archivo que contiene:

```

1 {{{
2     import cPickle
3     response.headers['Content-Type'] = 'application/python.pickle'
4     response.write(cPickle.dumps(response._vars), escape=False)
5 }}}

```

Si usted quiere ser capaz de hacer como un archivo de escoger cualquier acción, sólo tiene que guardar el archivo anterior con el nombre de "generic.pickle".

No todos los objetos son pickleable, y no todos los objetos en vinagre puede ser un-en vinagre. Es seguro que se adhieren a archivos primitiva de Python y combinaciones de ellos. Los objetos que no contienen referencias a los flujos de archivos o conexiones de base de datos son generalmente pickleable, pero sólo puede ser un-sumergido en un ambiente donde las clases de todos los objetos en escabeche ya están definidos.

9.1.5 RSS

web2py "generic.rss" incluye una vista que puede hacer que el diccionario devuelto por la acción como un feed RSS.

Debido a los feeds RSS tienen una estructura fija (título, enlace, descripción, artículos, etc), entonces para que esto funcione, el diccionario devuelto por la acción deberá tener la estructura adecuada:

```

1 {'title'      : '',
2  'link'       : '',
3  'description': '',
4  'created_on' : '',
5  'entries'    : []}
```

y cada entrada en las entradas deben tener la misma estructura similar:

```

1 {'title'      : '',
2  'link'       : '',
3  'description': '',
4  'created_on' : ''}
```

Por ejemplo, la siguiente acción se puede representar como un feed RSS:

```

1 def feed():
2     return dict(title="my feed",
3                 link="http://feed.example.com",
4                 description="my first feed",
5                 entries=[
6                     dict(title="my feed",
7                         link="http://feed.example.com",
8                         description="my first feed")
9                 ])
```

simplemente visitando la dirección:

```
1 http://127.0.0.1:8000/app/default/feed.rss
```

O, si el siguiente modelo:

```

1 db.define_table('rss_entry',
2                 Field('title'),
3                 Field('link'),
4                 Field('created_on','datetime'),
5                 Field('description'))
```

la siguiente acción también puede ser traducido como un feed RSS:

```

1 def feed():
2     return dict(title="my feed",
3                 link="http://feed.example.com",
4                 description="my first feed",
5                 entries=db().select(db.rss_entry.ALL).as_list())

```

El método de un objeto `as_list()` Filas convierte las filas en una lista de los diccionarios.

Si, además de artículos de diccionario se encuentran con los nombres clave no explícitamente figuran en esta lista, se les ignora.

Esta es la opinión "generic.rss" proporciona web2py:

```

1 {{{
2 try:
3     from gluon.serializers import rss
4     response.write(rss(response._vars),escape=False)
5     response.headers['Content-Type']='application/rss+xml'
6 except:
7     raise HTTP(405,'no rss')
8 }}}

```

Como un ejemplo más de una aplicación de RSS, consideramos un agregador RSS que recoge datos de los piensos "slashdot" y devuelve un feed web2py nuevo.

```

1 def aggregator():
2     import gluon.contrib.feedparser as feedparser
3     d = feedparser.parse(
4         "http://rss.slashdot.org/Slashdot/slashdot/to")
5     return dict(title=d.channel.title,
6                 link = d.channel.link,
7                 description = d.channel.description,
8                 created_on = request.now,
9                 entries = [
10                     dict(title = entry.title,
11                         link = entry.link,
12                         description = entry.description,
13                         created_on = request.now) for entry in d.entries])
14

```

Se puede acceder en:

```
1 http://127.0.0.1:8000/app/default/aggregator.rss
```

9.1.6 CSV

Los valores separados por comas (CSV) es un protocolo para representar datos tabulares.

Considere el siguiente modelo:

```
1 db.define_model('animal',
2     Field('species'),
3     Field('genus'),
4     Field('family'))
```

y las siguientes actuaciones:

```
1 def animals():
2     animals = db().select(db.animal.ALL)
3     return dict(animals=animals)
```

web2py no proporciona una "generic.csv"; you must define a custom view "default/animals.csv" que serializa los animales en CSV. He aquí una posible implementación:

```
1 {{%
2 import cStringIO
3 stream=cStringIO.StringIO()
4 animals.export_to_csv_file(stream)
5 response.headers['Content-Type']='application/vnd.ms-excel'
6 response.write(stream.getvalue(), escape=False)
7 }}
```

Tenga en cuenta que para CSV también se podría definir un archivo de "generic.csv" file, but one would have to specify the name of the object to be serialized ("animals" in the example). This is why we do not provide a "generic.csv".

9.2 Pide a procedimiento remoto

web2py proporciona un mecanismo para activar cualquier función en un servicio web. El mecanismo descrito aquí difiere del mecanismo descrito antes, porque:

- La función puede tener argumentos
- La función se puede definir en un modelo o un módulo en lugar del controlador de
- Es posible que desee especificar en detalle el método de RPC debe ser apoyada
- Impone una URL más estricta convención de nomenclatura
- Es más inteligente a continuación, los métodos anteriores, ya que trabaja para un conjunto fijo de protocolos. Por la misma razón no es tan fácilmente extensible.

Para utilizar esta función:

En primer lugar, debe importar y crear instancias de un objeto de servicio.

```
1 from gluon.tools import Service
2 service = Service(globals())
```

Este ya se hace en el archivo del modelo "db.py" en la solicitud de andamios.

En segundo lugar, debe exponer el controlador de servicio en el controlador:

```
1 def call():
2     session.forget()
3     return service()
```

Este hecho ya en el controlador de la aplicación "default.py" andamios. session.forget() quita es que planea usar cookies de sesión con los servicios.

En tercer lugar, hay que decorar las funciones que deseé para exponer como un servicio. Aquí está una lista de los decoradores de la actualidad:

```

1 @service.run
2 @service.xml
3 @service.json
4 @service.rss
5 @service.csv
6 @service.xmlrpc
7 @service.jsonrpc
8 @service.amfrpc3('domain')
```

Como ejemplo, consideremos la siguiente función decoradas:

```

1 @service.run
2 def concat(a,b):
3     return a+b
```

Esta función se puede definir en un modelo o en un controlador. Esta función ahora se puede llamar de forma remota de dos maneras:

```

1 http://127.0.0.1:8000/app/default/call/run(concat?a=hello&b=world)
2 http://127.0.0.1:8000/app/default/call/run/concat/hello/world
```

En ambos casos los rendimientos petición http:

```
1 helloworld
```

Si el decorador `@service.xml` se utiliza, la función puede ser llamada a través de

```

1 http://127.0.0.1:8000/app/default/call/xml(concat?a=hello&b=world)
2 http://127.0.0.1:8000/app/default/call/xml(concat/hello/world
```

y la salida se devuelve como XML:

```

1 <document>
2   <result>helloworld</result>
3 </document>
```

Puede serializar la salida de la función, incluso si se trata de un Filas DAL objeto. En este caso, de hecho, se llamará al `as_list()` automáticamente.

Si el decorador `@service.json` se utiliza, la función puede ser llamada a través de

```
1 http://127.0.0.1:8000/app/default/call/json/concat?a=hello&b=world
2 http://127.0.0.1:8000/app/default/call/json/concat/hello/world
```

y la salida que se devuelve como JSON.

Si el decorador `@service.csv` se utiliza, el controlador de servicio requiere, como valor de retorno, un objeto iterable de objetos iterable, como una lista de listas. Aquí está un ejemplo:

```
1 @service.csv
2 def table1(a,b):
3     return [[a,b],[1,2]]
```

Este servicio se puede llamar al visitar una de las siguientes URL:

```
1 http://127.0.0.1:8000/app/default/call/csv/table1?a=hello&b=world
2 http://127.0.0.1:8000/app/default/call/csv/table1/hello/world
```

y lo devuelve:

```
1 hello,world
2 1,2
```

El decorador `@service.rss` espera un valor devuelto en el mismo formato que la vista "generic.rss" discutidos en la sección anterior. decoradores múltiples son permitidos para cada función.

Hasta ahora, todo lo debatido en esta sección es simplemente una alternativa al método descrito en la sección anterior. El poder real del objeto de servicio viene con XMLRPC, JSONRPC AMFRPC y, como veremos más adelante.

9.2.1 XMLRPC

Considere el siguiente código, por ejemplo, en el controlador "default.py":

```

1 @service.xmlrpc
2 def add(a,b):
3     return a+b
4
5 @service.xmlrpc
6 def div(a,b):
7     return a+b

```

Ahora, en una cáscara python que puede hacer

```

1 >>> from xmlrpclib import ServerProxy
2 >>> server = ServerProxy(
3         'http://127.0.0.1:8000/app/default/call/xmlrpc')
4 >>> print server.add(3,4)
5 7
6 >>> print server.add('hello','world')
7 'helloworld'
8 >>> print server.div(12,4)
9 3
10 >>> print server.div(1,0)
11 ZeroDivisionError: integer division or modulo by zero

```

El módulo de Python xmlrpclib proporciona a un cliente para el protocolo XMLRPC. web2py actos como el servidor.

El cliente se conecta al servidor a través ServerProxy y remotamente puede llamar decoradas funciones en el servidor. Los datos (a, b) se pasa a la función (s), no a través de GET / POST variables, pero con la codificación correcta en el cuerpo de la petición mediante el protocolo XMLRPC, por lo que lleva consigo misma información de tipo (int o una cadena u otro). Lo mismo es cierto para el valor devuelto (s). Por otra parte, cualquier excepción que ocurre en el servidor se propaga de vuelta al cliente.

Hay bibliotecas XMLRPC para muchos lenguajes de programación (incluyendo C, C + +, Java, C #, Ruby y Perl), y pueden interoperar entre sí. Este es uno de los mejores métodos para crear aplicaciones que se comuniquen entre sí, independientemente del lenguaje de programación.

El cliente XMLRPC también puede ser aplicado dentro de una acción web2py de manera que una acción puede hablar con otra aplicación web2py (incluso

dentro de la misma instalación) con XMLRPC. Tenga cuidado con los callejones sin salida período de sesiones en este caso. Si una acción llamadas a través de XMLRPC una función en la misma aplicación, la persona que llama debe liberar el bloqueo de sesión antes de la llamada:

```

1 session.forget()
2 session._unlock(response)

```

9.2.2 JSONRPC y Pijamas

JSONRPC es muy similar al XMLRPC, pero utiliza el protocolo JSON base para codificar los datos en lugar de XML. Como ejemplo de aplicación aquí, se discute su uso con pijama. Pijama Python es un puerto de la Guía Web de Google (originalmente escrito en Java). Pijama permite escribir una aplicación cliente en Python. Pijama traduce este código en JavaScript. web2py sirve el javascript y se comunica con él a través de peticiones AJAX originarios del cliente y provocados por acciones del usuario.

A continuación se describe cómo hacer el trabajo con el pijama de web2py. No requiere ninguna librería adicional que no sea web2py y Pijamas.

Vamos a crear una aplicación simple "todo" con un cliente Pijamas (todos los navegadores) que comunica con el servidor exclusivamente a través de JSONRPC.

Aquí está cómo hacerlo:

Primero, cree una nueva aplicación llamada "todo".

En segundo lugar, "models/db.py", introduzca el siguiente código:

```

1 db=SQLDB('sqlite://storage.sqlite')
2 db.define_table('todo', Field('task'))
3
4 from gluon.tools import Service      # import rpc services
5 service = Service(globals())

```

En tercer lugar, "controllers/default.py", introduzca el siguiente código:

```

1  def index():
2      redirect(URL('todoApp'))
3
4      @service.jsonrpc
5      def getTasks():
6          todos = db(db.todo.id>0).select()
7          return [(todo.task,todo.id) for todo in todos]
8
9      @service.jsonrpc
10     def addTask(taskFromJson):
11         db.todo.insert(task= taskFromJson)
12         return getTasks()
13
14     @service.jsonrpc
15     def deleteTask (idFromJson):
16         del db.todo[idFromJson]
17         return getTasks()
18
19     def call():
20         session.forget()
21         return service()
22
23     def todoApp():
24         return dict()
```

El propósito de cada función debe ser obvia.

En cuarto lugar, "views/default/todoApp.html", introduzca el siguiente código:

```

1 <html>
2     <head>
3         <meta name="pygwt:module"
4             content="{{=URL('static','output/todoapp')}}" />
5         <title>
6             simple todo application
7         </title>
8     </head>
9     <body bgcolor="white">
10        <h1>
11            simple todo application
12        </h1>
13        <i>
14            type a new task to insert in db,
15            click on existing task to delete it
16        </i>
17        <script language="javascript"
```

```
18     src="{{=URL('static','output/pygwt.js')}}">
19   </script>
20 </body>
21 </html>
```

Este punto de vista sólo se ejecuta el código de Pijamas en "static/output/todoapp". El código que aún no hemos creado.

En quinto lugar, en la "static/TodoApp.py" (el anuncio es TodoApp, no todoApp!), Introduzca el código de cliente siguientes:

```
1 from pyjamas.ui.RootPanel import RootPanel
2 from pyjamas.ui.Label import Label
3 from pyjamas.ui.VerticalPanel import VerticalPanel
4 from pyjamas.ui.TextBox import TextBox
5 import pyjamas.ui.KeyboardListener
6 from pyjamas.ui.ListBox import ListBox
7 from pyjamas.ui.HTML import HTML
8 from pyjamas.JSONService import JSONProxy
9
10 class TodoApp:
11     def onModuleLoad(self):
12         self.remote = DataService()
13         panel = VerticalPanel()
14
15         self.todoTextBox = TextBox()
16         self.todoTextBox.addKeyboardListener(self)
17
18         self.todoList = ListBox()
19         self.todoList.setVisibleItemCount(7)
20         self.todoList.setWidth("200px")
21         self.todoList.addClickListener(self)
22         self.Status = Label("")
23
24         panel.add(Label("Add New Todo:"))
25         panel.add(self.todoTextBox)
26         panel.add(Label("Click to Remove:"))
27         panel.add(self.todolist)
28         panel.add(self.Status)
29         self.remote.getTasks(self)
30
31         RootPanel().add(panel)
32
33     def onKeyUp(self, sender, keyCode, modifiers):
34         pass
35
```

```

36     def onKeyDown(self, sender, keyCode, modifiers):
37         pass
38
39     def onKeyPress(self, sender, keyCode, modifiers):
40         """
41             This function handles the onKeyPress event, and will add the
42             item in the text box to the list when the user presses the
43             enter key. In the future, this method will also handle the
44             auto complete feature.
45         """
46         if keyCode == KeyboardListener.KEY_ENTER and \
47             sender == self.todoTextBox:
48             id = self.remote.addTask(sender.getText(),self)
49             sender.setText("")
50             if id<0:
51                 RootPanel().add(HTML("Server Error or Invalid Response"))
52
53     def onClick(self, sender):
54         id = self.remote.deleteTask(
55             sender.getValue(sender.getSelectedIndex()),self)
56         if id<0:
57             RootPanel().add(
58                 HTML("Server Error or Invalid Response"))
59
60     def onRemoteResponse(self, response, request_info):
61         self.todoList.clear()
62         for task in response:
63             self.todoList.addItem(task[0])
64             self.todoList.setValue(self.todoList.getItemCount()-1,
65                                 task[1])
66
67     def onRemoteError(self, code, message, request_info):
68         self.Status.setText("Server Error or Invalid Response: " \
69                             + "ERROR " + code + " - " + message)
70
71 class DataService(JSONProxy):
72     def __init__(self):
73         JSONProxy.__init__(self, "../../../../default/call/jsonrpc",
74                            ["getTasks", "addTask","deleteTask"])
75
76 if __name__ == '__main__':
77     app = TodoApp()
78     app.onModuleLoad()

```

En sexto lugar, ejecute Pijama antes de servir la demanda:

```

1 cd /path/to/todo/static/
2 python /python/pyjamas-0.5pl/bin/pyjsbuild TodoApp.py

```

Esto se traducirá el código Python en JavaScript para que pueda ser ejecutado en el navegador.

Para acceder a esta aplicación, visite la dirección

```
1 http://127.0.0.1:8000/todo/default/todoApp
```

Este apartado fue creado por Chris Prinos con la ayuda de forma Luke Kenneth Casson Leighton (creadores de Pijamas) y actualizada por Alexei Vinidiktov. Ha sido probado por el Pijama 0.5p1. En el ejemplo se inspiró en esta página Django en ref. (74).

9.2.3 AMFRPC

AMFRPC es el protocolo de llamada a procedimiento remoto que utilizan los clientes de Flash se comunique con un servidor. web2py apoya AMFRPC pero requiere que se ejecuta desde el origen y web2py que preinstalar la biblioteca PyAMF. Esto se puede instalar desde el shell de Windows o Linux, escriba

```
1 easy_install pyamf
```

(Por favor, consulte la documentación PyAMF para más detalles).

En este apartado se supone que ya está familiarizado con la programación ActionScript.

Vamos a crear un servicio simple que toma dos valores numéricos, los agrega juntos, y devuelve la suma. Nosotros llamamos el "pyamf_test" aplicación web2py, y vamos a llamar al servicio de addNumbers.

En primer lugar, con Adobe Flash (cualquier versión a partir del MX 2004), crear la aplicación cliente Flash empezando con un nuevo archivo FLA de Flash. En el primer fotograma del archivo, agregue estas líneas:

```

1 import mx.remoting.Service;
2 import mx.rpc.RelayResponder;
3 import mx.rpc.FaultEvent;
4 import mx.rpc.ResultEvent;
5 import mx.remoting.PendingCall;
6
7 var val1 = 23;
8 var val2 = 86;
9
10 service = new Service(
11     "http://127.0.0.1:8000/pyamf_test/default/call/amfrpc3",
12     null, "mydomain", null, null);
13
14 var pc:PendingCall = service.addNumbers(val1, val2);
15 pc.responder = new RelayResponder(this, "onResult", "onFault");
16
17 function onResult(re:ResultEvent):Void {
18     trace("Result : " + re.result);
19     txt_result.text = re.result;
20 }
21
22 function onFault(fault:FaultEvent):Void {
23     trace("Fault: " + fault.fault.faultstring);
24 }
25
26 stop();

```

Este código permite que el cliente Flash para conectarse a un servicio que corresponde a una función llamada "addNumbers" in the file "/pyamf_test/default/gateway". You must also import ActionScript version 2 MX remoting classes to enable Remoting in Flash. Add the path to these classes to the classpath settings in the Adobe Flash IDE, or just place the "mx" carpeta junto al archivo recién creado.

Notificación de los argumentos del constructor de Servicio. El primer argumento es la URL correspondiente al servicio que queremos que se cree. El tercer argumento es el dominio del servicio. Decidimos llamar a este "mydomain" dominio.

En segundo lugar, crear un campo de texto dinámico llamado "txt_result" y colocarlo en el escenario.

En tercer lugar, es necesario establecer una puerta de enlace web2py que pueden comunicarse con el cliente Flash definido anteriormente.

Proceder mediante la creación de una aplicación llamada web2py nuevo pyamf_test que albergará el nuevo servicio y la pasarela AMF para el cliente flash. Edite el controlador de "default.py" y asegúrese de que contenga

```

1 @service.amfrpc3('mydomain')
2 def addNumbers(val1, val2):
3     return val1 + val2
4
5 def call(): return service()
```

En cuarto lugar, recopilar y exportación / publicar el cliente Flash SWF como lugar pyamf_test.swf, "pyamf_test.amf", "pyamf_test.html", "AC_RunActiveContent.js", and "crossdomain.xml" files in the "static" folder of the newly created application that is hosting the gateway, "pyamf_test" el.

Ahora puede probar el cliente, visite:

```
1 http://127.0.0.1:8000/pyamf_test/static/pyamf_test.html
```

La puerta de enlace se denomina en segundo plano cuando el cliente se conecta a addNumbers.

Si usted está demandando AMF0 lugar de AMF3 también puede utilizar la decoradora:

```
1 @service.amfrpc
```

en lugar de:

```
1 @service.amfrpc3('mydomain')
```

En este caso también tiene que cambiar la dirección de servicios a:

```
1 http://127.0.0.1:8000/pyamf_test/default/call/amfrpc
```

9.2.4 SOAP

web2py incluye un cliente y el servidor SOAP creado por Mariano Reingart. Puede ser utilizado en gran medida como XML-RPC:

Considere el siguiente código, por ejemplo, en el controlador "default.py":

```
1 @service.soap('MyAdd', returns={'result':int, args={'a':int, 'b':int,}})
2 def add(a,b):
3     return a+b
```

Ahora, en una cáscara python que puede hacer

```
1 >>> from gluon.contrib.pysimplesoap.client import SoapClient
2 >>> client = SoapClient(
3     location = "http://localhost:8000/app/default/call/soap",
4     action = 'http://example.com/', # SOAPAction
5     namespace = "http://example.com/sample.wsdl",
6     soap_ns='soap', # classic soap 1.1 dialect
7     trace = True, # print http/xml request and response
8     ns = False) # do not add target namespace prefix
9
10>>> print client.MyAdd(a=1,b=2)
113
```

Puede obtener el WSDL para el servicio en

```
1 http://127.0.0.1:8000/app/default/call?WSDL
```

Y usted puede obtener la documentación para cualquiera de los métodos expuestos:

```
1 http://127.0.0.1:8000/app/default/call?op=MyAdd
```

9.3 Bajo nivel de API y otras recetas

9.3.1 simplejson

web2py incluye gluon.contrib.simplejson, desarrollado por Bob Ippolito. Este módulo proporciona el codificador más estándar Python-JSON-decodificador.

Simplejson consiste en dos funciones:

- codifica un objeto Python `gluon.contrib.simplejson.dumps(a)` a en JSON.
- decodifica un objeto JavaScript `gluon.contrib.simplejson.loads(b)` b en un objeto Python.

Los tipos de objeto que se puede serializar incluye tipos primitivos, listas y diccionarios. Objetos de composición se puede serializar con la excepción de las clases definidas por el usuario.

He aquí una muestra de la acción (por ejemplo en el controlador "default.py") que serializa la lista de Python que contiene los días de semana con este bajo nivel API:

```
1 def weekdays():
2     names=['Sunday', 'Monday', 'Tuesday', 'Wednesday',
3           'Thursday', 'Friday', 'Saturday']
4     import gluon.contrib.simplejson
5     return gluon.contrib.simplejson.dumps(names)
```

A continuación se muestra una página HTML de ejemplo que envía una petición Ajax a la acción anterior, recibe el mensaje de JSON, y almacena la lista en una variable correspondiente JavaScript:

```
1 {{extend 'layout.html'}}
2 <script>
3 $.getJSON('/application/default/weekdays',
4           function(data){ alert(data); });
5 </script>
```

El código utiliza la función de jQuery `$.getJSON`, que realiza la llamada Ajax y, en respuesta, almacena los nombres de los días de semana en una variable

local data JavaScript y pasa la variable a la función de llamada. En el ejemplo de la función de devolución de llamada, simplemente advierte al visitante que los datos han sido recibidos.

9.3.2 PyRTF

Otra necesidad común de los sitios web es que la generación de documentos de texto de Word en formato electrónico. La forma más sencilla de hacerlo es utilizando el formato de texto enriquecido (RTF) formato de documento. Este formato fue inventado por Microsoft y desde entonces se ha convertido en un estándar. web2py incluye gluon.contrib.pyrtf, desarrollado por Simon Cusack y revisado por el subsidio de Edwards. Este módulo le permite generar documentos RTF programación incluyendo color de texto con formato e imágenes.

En el ejemplo siguiente instancia de dos clases básicas de RTF, y en la sección de documentos, añadir la segunda a la primera e insertaremos un texto de relleno en este último:

```

1 def makertf():
2     import gluon.contrib.pyrtf as q
3     doc=q.Document()
4     section=q.Section()
5     doc.Sections.append(section)
6     section.append('Section Title')
7     section.append('web2py is great. '*100)
8     response.headers['Content-Type']='text/rtf'
9     return q.dumps(doc)

```

Al final, el documento se serializa por `q.dumps(doc)`. Tenga en cuenta que antes de devolver un documento RTF que es necesario especificar el tipo de contenido en la cabecera de lo contrario el navegador no sabe cómo manejar el archivo.

Dependiendo de la configuración, el navegador le pedirá si desea guardar el archivo o abrirlo con un editor de texto.

9.3.3 ReportLab y PDF

web2py también pueden generar documentos PDF, con una biblioteca adicional denominado "ReportLab" (73).

Si está ejecutando web2py de la fuente, es suficiente con tener instalado ReportLab. Si está ejecutando la distribución de Windows binarios, es necesario descomprimir la carpeta en ReportLab "web2py/". Si está ejecutando el binario de distribución de Mac, es necesario descomprimir ReportLab en la carpeta:

```
1 web2py/app/Contents/Resources/
```

A partir de ahora suponemos ReportLab está instalado y que web2py lo puede encontrar. Vamos a crear una acción simple llamada "get_me_a_pdf" que genera un documento PDF.

```
1 from reportlab.platypus import *
2 from reportlab.lib.styles import getSampleStyleSheet
3 from reportlab.rl_config import defaultPageSize
4 from reportlab.lib.units import inch, mm
5 from reportlab.lib.enums import TA_LEFT, TA_RIGHT, TA_CENTER, TA_JUSTIFY
6 from reportlab.lib.colors import colors
7 from uuid import uuid4
8 from cgi import escape
9 import os
10
11 def get_me_a_pdf():
12     title = "This The Doc Title"
13     heading = "First Paragraph"
14     text = 'bla ' * 10000
15
16     styles = getSampleStyleSheet()
17     tmpfilename=os.path.join(request.folder,'private',str(uuid4()))
18     doc = SimpleDocTemplate(tmpfilename)
19     story = []
20     story.append(Paragraph(escape(title),styles["Title"]))
21     story.append(Paragraph(escape(heading),styles["Heading2"]))
22     story.append(Paragraph(escape(text),styles["Normal"]))
23     story.append(Spacer(1,2*inch))
24     doc.build(story)
25     data = open(tmpfilename,"rb").read()
26     os.unlink(tmpfilename)
```

```

27     response.headers['Content-Type']='application/pdf'
28     return data

```

Observe cómo se genera el PDF en un único archivo temporal, `tmpfilename`, se lee el PDF generado a partir del archivo, a continuación, hemos borrado el archivo.

Para obtener más información acerca de la API ReportLab, consulte la documentación ReportLab. Recomendamos encarecidamente que utilice la API de ReportLab Platypus, como `Paragraph`, `Spacer`, etc

9.4 Servicios y autenticación

En el capítulo anterior hemos analizado el uso de los decoradores siguientes:

```

1 @auth.requires_login()
2 @auth.requires_membership(...)
3 @auth.requires_permission(...)

```

Para las acciones normales (no como los servicios de decoración), estos decoradores se puede utilizar incluso si la salida se representa en un formato diferente al HTML.

Para las funciones definidas como servicios y decorada con la decoradores `@service...`, `@auth...` los decoradores no se debe utilizar. Los dos tipos de decoradores no se pueden mezclar. Si la autenticación se va a realizar, son las acciones `call` que necesita ser decorada:

```

1 @auth.requires_login()
2 def call(): return service()

```

Tenga en cuenta que también es posible crear una instancia de servicio de varios objetos, registrar las diferentes funciones con ellos mismos, y exponer algunos de ellos con la autenticación y otros no:

```
1 public_services=Service(globals())
2 private_services=Service(globals())
3
4 @public_service.jsonrpc
5 @private_service.jsonrpc
6 def f(): return 'public'
7
8 @private_service.jsonrpc
9 def g(): return 'private'
10
11 def public_call(): return public_service()
12
13 @auth.requires_login()
14 def private_call(): return private_service()
```

Esto supone que la persona que llama es pasar las credenciales en la cabecera HTTP (una cookie de sesión válido o mediante la autenticación básica, como se discutió en la sección anterior). El cliente debe apoyarlo, no todos los clientes de hacer.

10

Ajax Recetas

Mientras web2py es principalmente para el desarrollo del lado del servidor, la aplicación andamios **welcome** viene con la base de jQuery library (32), calendarios jQuery (selector de fechas, el selector de fecha y hora y el reloj), el menú de "superfish.js", y algunas funciones adicionales basadas en Javascript jQuery.

Ninguna de las web2py no le permite usar otras bibliotecas Ajax (75) como Prototype, ExtJS o YUI pero decidimos paquete de jQuery porque encontramos que sea más fácil de usar y más potente que ninguna biblioteca equivalente. También nos parece captura el espíritu web2py de ser funcional y conciso.

10.1 web2py_ajax.html

La aplicación de scaffolding web2py "welcome" incluye un archivo denominado

¹ `views/web2py_ajax.html`

Este archivo se incluye en el HEAD de la "layout.html" por defecto y proporciona los siguientes servicios:

- Incluye static/jquery.js.
- Incluye static/calendar.js y static/calendar.css, si es que existen.
- Define una función ajax (basado en jQuery \$. Ajax).
- Hace que cualquier DIV de diapositivas "error" or any tag object of class "flash" clase abajo.
- Evita escribir números enteros no válidos en los campos de entrada del "integer" clase.
- Evita escribir válida flota en los campos de entrada del "double" clase.
- Conecta la entrada de los campos "date" tipo con un selector de fecha que aparece.
- Conecta la entrada ámbitos de la "datetime" con un selector de tipo de fecha y hora que aparece.
- Conecta la entrada de los campos "time" tipo con un selector horario emergente.
- Define web2py_ajax_component y muy importante herramienta que se describe en el capítulo 13.

También incluye popup, collapse, fade y las funciones de compatibilidad con versiones anteriores.

He aquí un ejemplo de cómo una a los demás efectos jueguen bien juntos.

Considere una aplicación **test** con el siguiente modelo:

```

1 db = DAL("sqlite://db.db")
2 db.define_table('child',
3     Field('name'),
4     Field('weight', 'double'),
5     Field('birth_date', 'date'),
6     Field('time_of_birth', 'time'))
7
8 db.child.name.requires=IS_NOT_EMPTY()

```

```

9 db.child.weight.requires=IS_FLOAT_IN_RANGE(0,100)
10 db.child.birth_date.requires=IS_DATE()
11 db.child.time_of_birth.requires=IS_TIME()

```

con este controlador "default.py":

```

1 def index():
2     form = SQLFORM(db.child)
3     if form.accepts(request.vars, session):
4         response.flash = 'record inserted'
5     return dict(form=form)

```

y la vista "default/index.html" siguientes:

```

1 {{extend 'layout.html'}}
2 {{=form}}

```

La acción "index" genera la forma siguiente:

Name:	<input type="text"/>
Weigth:	<input type="text"/>
Birth Date:	<input type="text"/>
Time Of Birth:	<input type="text"/>
<input type="button" value="Submit"/>	

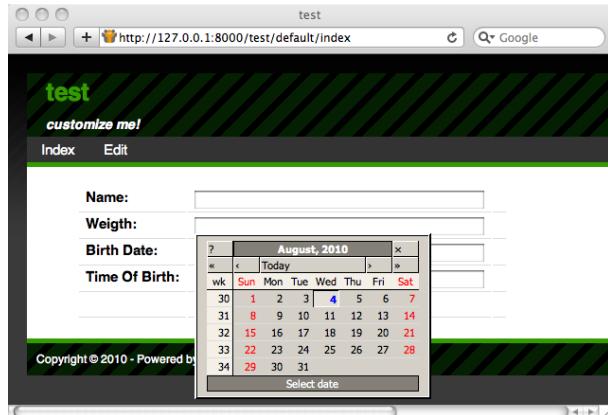
Si una forma inválida se presenta, el servidor devuelve la página con una forma modificada que contiene mensajes de error. Los mensajes de error son de DIVs "error" clase, y debido al código web2py_ajax anteriormente, los errores aparece con un efecto de deslizamiento hacia abajo:

Name:	<input type="text"/>
	enter a value
Weigth:	<input type="text"/>
	enter a number between 0.0 and 100.0
Birth Date:	<input type="text"/>
	enter date and time as 1963-08-28 14:30:59
Time Of Birth:	<input type="text"/>
	enter time as hh:mm:ss (seconds, am, pm optional)
<input type="button" value="Submit"/>	

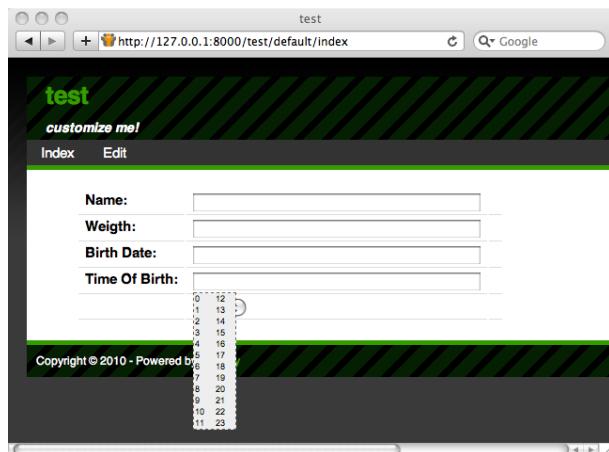
El color de los errores, se indican en el código CSS en "layout.html".

El código web2py_ajax no le permite escribir un valor no válido en el campo de entrada. Esto se hace antes y además, no como un sustituto de la validación del lado del servidor.

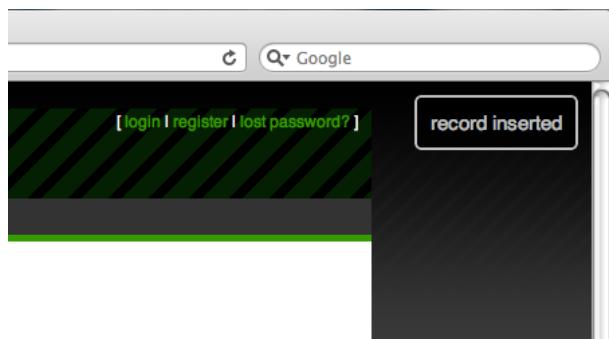
El código se muestra web2py_ajax un selector de fecha cuando entre un campo de entrada de "date", and it displays a datetime picker when you enter an INPUT field of class "datetime" clase. Aquí está un ejemplo:



El código web2py_ajax también muestra el selector horario siguiente cuando intenta modificar un campo de entrada de la clase "time":



Tras la presentación, la acción del controlador establece el flash respuesta a la "record inserted". The default layout renders this message in a DIV with id="flash" mensaje. El código web2py_ajax es responsable de tomar esta diapositiva DIV abajo y hacerlo desaparecer cuando se hace clic en ella:



Estos y otros efectos son accesibles mediante programación en los puntos de vista ya través de ayudantes con los controladores.

10.2 Efectos jQuery

Los efectos básicos descritos en esta sección no requiere archivos adicionales, todo lo que necesitas ya está incluida por usted por web2py_ajax.html.

HTML / XHTML objetos pueden ser identificados por su tipo (por ejemplo, un DIV), sus clases, o su id. Por ejemplo:

```
1 <div class="one" id="a">Hello</div>
2 <div class="two" id="b">World</div>
```

Pertenecen a la clase "one" and "two" respectivamente. They have ids equal to "a" and "b" respectivamente.

En jQuery se puede hacer referencia a la antigua con las siguientes notaciones equivalentes como una

```
1 jquery('.one')      // address object by class "one"
2 jquery('#a')        // address object by id "a"
3 jquery('DIV.one')  // address by object of type "DIV" with class "one"
4 jquery('DIV #a')   // address by object of type "DIV" with id "a"
```

en CSS y al segundo con

```
1 jquery('.two')
2 jquery('#b')
3 jquery('DIV.two')
4 jquery('DIV #b')
```

o puede referirse tanto a

```
1 jquery('DIV')
```

Tag objetos están asociados a eventos, tales como "onclick". jQuery allows linking these events to effects, for example "slideToggle":

```
1 <div class="one" id="a" onclick="jquery('.two').slideToggle()">Hello</div>
2 <div class="two" id="b">World</div>
```

Ahora bien, si haces clic en "Hello", "World" disappears. If you click again, "World" vuelve a aparecer. Usted puede hacer una etiqueta oculta de manera predeterminada, dándole una clase oculta:

```

1 <div class="one" id="a" onclick="jQuery('.two').slideToggle()">Hello</div>
2 <div class="two hidden" id="b">World</div>
```

También puede vincular las acciones a eventos fuera de la propia etiqueta. El código anterior se puede reescribir de la siguiente manera:

```

1 <div class="one" id="a">Hello</div>
2 <div class="two" id="b">World</div>
3 <script>
4 jQuery('.one').click(function(){jQuery('.two').slideToggle()});
5 </script>
```

Efectos de devolver el objeto de llamar, por lo que se pueden encadenar.

Cuando el `click` establece la función callback que se llama al hacer clic. Lo mismo ocurre con `change`, `keyup`, `keydown`, `mouseover`, etc

Una situación común es la necesidad de ejecutar algún código JavaScript sólo después de todo el documento se haya cargado. Esto se hace generalmente por el atributo `onload` de BODY jQuery, pero proporciona una forma alternativa que no requiere editar el formato:

```

1 <div class="one" id="a">Hello</div>
2 <div class="two" id="b">World</div>
3 <script>
4 jQuery(document).ready(function(){
5   jQuery('.one').click(function(){jQuery('.two').slideToggle()});
6 });
7 </script>
```

El cuerpo de la función sin nombre sólo se ejecuta cuando el documento esté listo, después de que haya sido totalmente cargado.

Aquí está una lista de nombres de evento útil:

Formulario de Eventos

- Onchange: Script para ejecutarse cuando el elemento de cambios
- onsubmit: Script para ejecutar cuando se envía el formulario
- onreset: Script para ejecutarse cuando el formulario se restablece
- onselect: Script para ejecutarse cuando el elemento está seleccionado
- onblur: Script para ejecutarse cuando el elemento pierde el foco
- onfocus: Script para ejecutarse cuando el elemento recibe el foco

Eventos del teclado

- Onkeydown: Script para ejecutar cuando se pulsa la tecla
- onkeypress: Script para ejecutarse cuando la tecla se presiona y se suelta
- onkeyup: Script para ejecutarse cuando la tecla se libera

Eventos del mouse

- OnClick: Script para ejecutarse en un clic del ratón
- ondblclick: Script para ejecutarse en un doble clic del ratón
- onmousedown: Script para ejecutar cuando se pulsa el botón del ratón
- onmousemove: Script para ejecutarse cuando el cursor del ratón se mueve
- onmouseout: Script para ejecutarse cuando el cursor del ratón se mueve fuera de un elemento
- onmouseover: Script para ejecutarse cuando se mueve el puntero sobre un elemento
- onmouseup: Script para ejecutarse cuando el botón del ratón es liberado

Aquí está una lista de efectos útiles definidas por jQuery:

Efectos

- JQuery (...). Attr (nombre): Devuelve el nombre del valor del atributo

- jQuery (...). Attr (nombre, valor): Establece el nombre de atributo en el valor
- jQuery (...). show (): Hace que el objeto visible
- jQuery (...). hide (): Hace que el objeto escondido
- jQuery (...). slideToggle (velocidad, de devolución de llamada): Hace que la diapositiva objeto hacia arriba o hacia abajo
- jQuery (...). slideUp (velocidad, de devolución de llamada): Hace que la diapositiva objeto hacia arriba
- jQuery (...). slideDown (velocidad, de devolución de llamada): Hace que la diapositiva objeto en el suelo
- jQuery (...). FadeIn (velocidad, de devolución de llamada): Hace que el objeto se desvanecen en
- jQuery (...). FadeOut (velocidad, de devolución de llamada): Hace que el objeto se desvanecen

El argumento de la velocidad es por lo general "slow", "fast" u omitido (por defecto). La devolución de llamada es una función opcional que se llama cuando el efecto se ha completado. jQuery efectos también pueden ser fácilmente incorporados en los ayudantes, por ejemplo, en una vista:

```
1 {{=DIV('click me!', _onclick="jQuery(this).fadeOut()")}}
```

jQuery es una librería Ajax muy compacta y concisa, por lo que web2py no necesita una capa de abstracción adicional en la parte superior de jQuery (con excepción de la función ajax discutido más adelante). Las API de jQuery son accesibles y están disponibles en su forma nativa cuando sea necesario.

Consulte la documentación para obtener más información acerca de estos efectos y otras API de jQuery.

La biblioteca jQuery también puede ser extendida a través de plug-ins y widgets de interfaz de usuario. En este tema no se describe aquí, ver ref. (77) para más detalles.

10.2.1 Campos condicional en las formas

Una aplicación típica de los efectos jQuery es una forma que cambia su aspecto en función del valor de sus campos.

Esto es fácil en web2py porque el ayudante SQLFORM genera formas que son "CSS friendly". El formulario contiene una tabla con filas. Cada fila contiene una etiqueta, un campo de entrada, y una tercera columna opcional. Los elementos identificadores han derivado estrictamente del nombre de la tabla y los nombres de los campos.

La convención es que todos los campos de entrada tiene un identificador de tablename_fieldname y está contenida en una fila con id tablename_fieldname_row.

A modo de ejemplo, crear un formulario de entrada que solicita el nombre de un contribuyente y para el nombre del cónyuge del contribuyente, pero sólo si se casó con él / ella es.

Crear una aplicación de prueba con el siguiente modelo:

```

1 db = DAL('sqlite://db.db')
2 db.define_table('taxpayer',
3     Field('name'),
4     Field('married', 'boolean'),
5     Field('spouse_name'))
```

el controlador de "default.py" siguientes:

```

1 def index():
2     form = SQLFORM(db.taxpayer)
3     if form.accepts(request.vars, session):
4         response.flash = 'record inserted'
5     return dict(form=form)
```

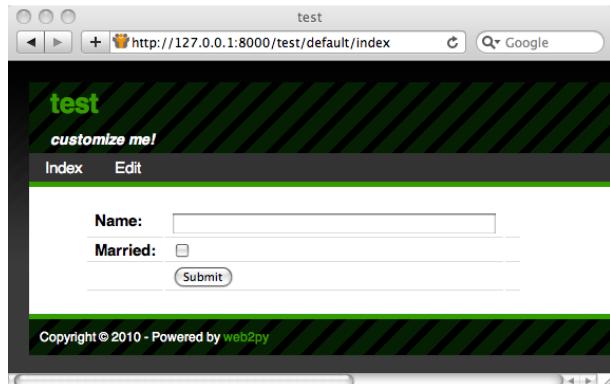
y la vista "default/index.html" siguientes:

```

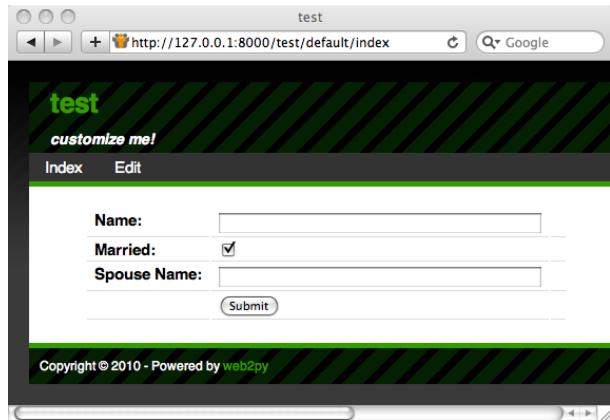
1 {{extend 'layout.html'}}
2 {{=form}}
3 <script>
```

```
4  jQuery(document).ready(function(){
5      jQuery('#taxpayer_spouse_name_row').hide();
6      jQuery('#taxpayer_married').change(function(){
7          if(jQuery('#taxpayer_married').attr('checked'))
8              jQuery('#taxpayer_spouse_name_row').show();
9          else jQuery('#taxpayer_spouse_name_row').hide();});
10 });
11 </script>
```

La secuencia de comandos en la vista tiene el efecto de ocultar la fila que contiene el nombre del cónyuge:



Cuando el contribuyente comprueba la casilla de verificación "married", el campo vuelve a aparecer el nombre del cónyuge:



Aquí "taxpayer_married" is the checkbox associated to the "boolean" field "married" of table "taxpayer". "taxpayer_spouse_name__row" it the row containing the input field for "spouse_name" of table "taxpayer".

10.2.2 Confirmación en Suprimir

Otra aplicación útil es exigir la confirmación en la comprobación de una casilla "delete" como eliminar la casilla de verificación que aparece en los formularios de edición.

Consideremos el ejemplo de arriba y agregar el controlador de la acción siguiente:

```

1 def edit():
2     row = db.taxpayer[request.args(0)]
3     form = SQLFORM(db.taxpayer, row, deletable=True)
4     if form.accepts(request.vars, session):
5         response.flash = 'record updated'
6     return dict(form=form)

```

y la vista correspondiente "default/edit.html"

```

1 {{extend 'layout.html'}}
2 {{=form}}

```

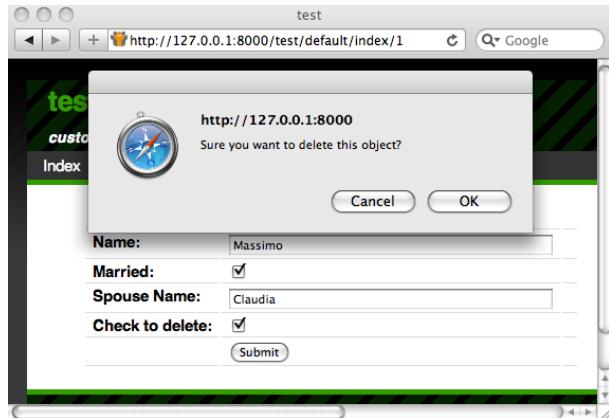
El argumento `deletable=True` en el constructor `SQLFORM` instruye `web2py` para mostrar una casilla de verificación "delete" en el formulario de edición. Es `False` de forma predeterminada. De "web2py_ajax.html" `web2py` incluye el siguiente código:

```

1  jQuery(document).ready(function(){
2      jQuery('input.delete').attr('onclick',
3          'if(this.checked) if(!confirm(
4              "{{{=T("Sure you want to delete this object?")}}}}"))
5          this.checked=false;');
6  });

```

Por convención, esta opción tiene una clase igual a "delete". El código `jQuery` anterior conecta el evento `onclick` de esta casilla de verificación con un cuadro de diálogo de confirmación (de serie en JavaScript) y desactiva la casilla de verificación si el contribuyente no lo confirma:



10.3 La función de ajax

En `web2py_ajax.html`, `web2py` define una función llamada `ajax` que se basa en, pero no se debe confundir con el `$.ajax` función de `jQuery`. Este último es mucho más potente que el anterior, y para su uso, le remitimos a ref. (32)

y ref. (76). Sin embargo, la primera función es suficiente para muchas tareas complejas, y es más fácil de usar.

La función ajax es una función de JavaScript que tiene la siguiente sintaxis:

```
1 ajax(url, [id1, id2, ...], target)
```

Es de forma asincrónica llama a la url (primer argumento), pasa el valor de los campos con la igualdad de id a una de las identificaciones en la lista (segundo argumento), a continuación, almacena la respuesta en la innerHTML de la etiqueta con el id iguales al objetivo (el tercer argumento).

He aquí un ejemplo de un controlador de default:

```
1 def one():
2     return dict()
3
4 def echo():
5     return request.vars.name
```

y la vista asociada "default/one.html":

```
1 {{extend 'layout.html'}}
2 <form>
3     <input id="name" onkeyup="ajax('echo', ['name'], 'target')" />
4 </form>
5 <div id="target"></div>
```

Cuando usted escriba algo en el campo de entrada, tan pronto como se suelta una tecla (onkeyup), la función se llama ajax, y el valor del campo id="name" se pasa a la acción de "echo", que envía el texto de nuevo a la vista. La función ajax recibe la respuesta y muestra el eco de la respuesta en la "target" DIV.

10.3.1 Eval objetivo

El tercer argumento de la función puede ser la ajax ":eval" cadena. Esto significa que la cadena devuelta por el servidor no se incrustará en el documento pero se evaluará en su lugar.

He aquí un ejemplo de un controlador de default:

```

1 def one():
2     return dict()
3
4 def echo():
5     return "jQuery('#target').html(%s);" % repr(request.vars.name)

```

y la vista asociada "default/one.html":

```

1 {{extend 'layout.html'}}
2 <form>
3     <input id="name" onkeyup="ajax('echo', ['name'], ':eval')"/>
4 </form>
5 <div id="target"></div>

```

Esto permite respuestas más articuladas que simples cadenas de texto.

10.3.2 Auto-realización

Web2py contiene un built-in Widgets de autocompletado, se describió en el capítulo dedicado a las formas. Sin embargo, aquí queremos construir uno más sencillo desde el principio.

Otra aplicación de la función por encima de ajax es auto-realización. Aquí queremos crear un campo de entrada que espera un nombre de mes y, cuando el usuario escribe un nombre incompleto, realiza auto-realización a través de una petición Ajax. En respuesta, una caída de finalización automática de la caja aparece debajo del campo de entrada.

Esto se puede lograr a través del controlador default siguientes:

```

1 def month_input():
2     return dict()
3
4 def month_selector():
5     if not request.vars.month: return ''
6     months = ['January', 'February', 'March', 'April', 'May',
7               'June', 'July', 'August', 'September', 'October'],

```

```

8           'November', 'December']
9 month_start = request.vars.month.capitalize()
10 selected = [m for m in months if m.startswith(month_start)]
11 return DIV(*[DIV(k,
12     _onclick="jQuery('#month').val('%s')" % k,
13     _onmouseover="this.style.backgroundColor='yellow'",
14     _onmouseout="this.style.backgroundColor='white'"
15     ) for k in selected)])

```

y la vista correspondiente "default/month_input.html":

```

1 {{extend 'layout.html'}}
2 <style>
3 #suggestions { position: relative; }
4 .suggestions { background: white; border: solid 1px #55A6C8; }
5 .suggestions DIV { padding: 2px 4px 2px 4px; }
6 </style>
7
8 <form>
9   <input type="text" id="month" name="month" style="width: 250px" /><br />
10  <div style="position: absolute;" id="suggestions"
11    class="suggestions"></div>
12 </form>
13 <script>
14   jQuery("#month").keyup(function(){
15     ajax('month_selector', ['month'], 'suggestions'));
16 </script>

```

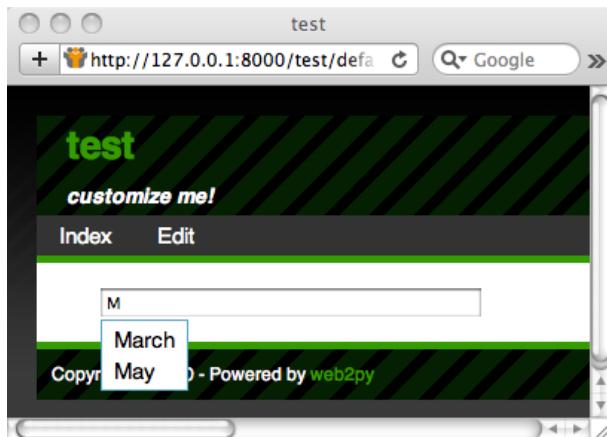
El script de jQuery en la vista activa la petición Ajax cada vez que algo los tipos de visitantes en el "month" input field. The value of the input field is submitted with the Ajax request to the "month_selector" action. This action finds a list of month names that start with the submitted text (selected), builds a list of DIVs (each one containing a suggested month name), and returns a string with the serialized DIVs. The view displays the response HTML in the "suggestions" DIV. The "month_selector" action generates both the suggestions and the JavaScript code embedded in the DIVs that must be executed when the visitor clicks on each suggestion. For example when the visitor types "Ma" llega la acción de devolución de llamada:

```

1 <div onclick="jQuery('#month').val('February')"
2   onmouseout="this.style.backgroundColor='white'"
3   onmouseover="this.style.backgroundColor='yellow'">February</div>

```

Este es el efecto final:



Si los meses se almacenan en una tabla de base de datos como:

```
1 db.define_table('month', Field('name'))
```

entonces simplemente reemplazar la acción con `month_selector`:

```
1 def month_input():
2     return dict()
3
4 def month_selector():
5     if not request.vars.month:
6         return ""
7     pattern = request.vars.month.capitalize() + '%'
8     selected = [row.name for row in db(db.month.name.like(pattern)).select()]
9     return ".join([DIV(k,
10         _onclick="jQuery('#month').val('%s')" % k,
11         _onmouseover="this.style.backgroundColor='yellow'",
12         _onmouseout="this.style.backgroundColor='white'"
13         ).xml() for k in selected])"
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
307
308
309
309
310
311
312
313
314
315
315
316
317
317
318
319
319
320
321
322
323
323
324
325
325
326
327
327
328
329
329
330
331
331
332
333
333
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523

```

10.3.3 Formulario de Ajax

Aquí consideramos una página que permite al visitante enviar mensajes usando Ajax sin recargar la página entera. Web2py proporciona un mecanismo mejor para hacerlo que se describe aquí utilizando el ayudante CARGA y se describe en el capítulo 13, sin embargo, aquí queremos mostrar cómo hacerlo simplemente usando jQuery.

Contains a form "myform" and a "target" DIV. When the form is submitted, the server may accept it (and perform a database insert) or reject it (because it did not pass validation). The corresponding notification is returned with the Ajax response and displayed in the "target" DIV.

Construir una aplicación test con el siguiente modelo:

```
1 db = DAL('sqlite://db.db')
2 db.define_table('post', Field('your_message', 'text'))
3 db.post.your_message.requires = IS_NOT_EMPTY()
```

Tenga en cuenta que los temas tienen una "your_message" solo campo que se requiere para ser no-vacío.

Editar el controlador de default.py y escribir dos acciones:

```
1 def index():
2     return dict()
3
4 def new_post():
5     form = SQLFORM(db.post)
6     if form.accepts(request.vars, formname=None):
7         return DIV("Message posted")
8     elif form.errors:
9         return TABLE(*[TR(k, v) for k, v in form.errors.items()])
```

La primera acción no hace otra cosa que el retorno de una vista.

La segunda acción es la devolución de llamada Ajax. Se espera que las variables de formulario en el request.vars, los procesa y devuelve DIV("Message posted") en el éxito o una TABLE de mensajes de error en caso de fallo.

Ahora edite el punto de vista "default/index.html":

```

1 {{extend 'layout.html'}}
2
3 <div id="target"></div>
4
5 <form id="myform">
6   <input name="your_message" id="your_message" />
7   <input type="submit" />
8 </form>
9
10 <script>
11   jQuery('#myform').submit(function() {
12     ajax('{{=URL('new_post')}}',
13           ['your_message'], 'target');
14     return false;
15   });
16 </script>
```

Observe cómo en este ejemplo se crea el formulario manualmente usando el HTML, pero es procesado por el SQLFORM en una acción distinta a la que muestra el formulario. El objeto se serializa SQLFORM nunca en HTML. SQLFORM.accepts en este caso no hace falta ser un período de sesiones y establece formname=None, ya que optó por no establecer el nombre del formulario y una forma clave en el manual de formulario HTML.

La secuencia de comandos en la parte inferior de la vista conecta el presente "myform" botón para una función en línea que presenta la entrada con id="your_message" con la función web2py ajax, y muestra la respuesta dentro del DIV con id="target".

10.3.4 Voto y Valoración

Otra aplicación es el Ajax de voto o del número de elementos de una página. Aquí consideramos una aplicación que permite a los visitantes a votar sobre las imágenes mostradas. La aplicación consta de una sola página que muestra las imágenes ordenadas de acuerdo a su votación. Nos permitirá a los visitantes votar varias veces, aunque es fácil cambiar este comportamiento si

los usuarios se autentican, llevando un registro de los votos individuales en la base de datos y los asocian con el `request.env.remote_addr` del votante.

Aquí es un modelo de ejemplo:

```

1 db = DAL('sqlite://images.db')
2 db.define_table('item',
3     Field('image', 'upload'),
4     Field('votes', 'integer', default=0))
```

Aquí está el controlador de default:

```

1 def list_items():
2     items = db().select(db.item.ALL, orderby=db.item.votes)
3     return dict(items=items)
4
5 def download():
6     return response.download(request, db)
7
8 def vote():
9     item = db.item[request.vars.id]
10    new_votes = item.votes + 1
11    item.update_record(votes=new_votes)
12    return str(new_votes)
```

La acción de descarga es necesario para que el `list_items` fin de descargar imágenes almacenadas en la carpeta de "uploads". Los votos de acción se utiliza para la devolución de llamada Ajax.

Esta es la opinión "default/list_items.html":

```

1 {{extend 'layout.html'}}
2
3 <form><input type="hidden" id="id" value="" /></form>
4 {{for item in items:}}
5 <p>
6 
8 <br />
9 Votes=<span id="item{{=item.id}}">{{=item.votes}}</span>
10 [<span onclick="jQuery('#id').val('{{=item.id}}');"
11      ajax('vote', ['id'], 'item{{=item.id}}');">vote up</span>]
12 </p>
13 {{pass}}
```

Cuando el usuario hace clic en campo INPUT "[vote up]" the JavaScript code stores the item.id in the hidden "id" y presenta este valor en el servidor a través de una petición Ajax. El servidor incrementa los votos contador para el registro correspondiente y devuelve el nuevo conteo de votos como una cadena. Este valor se inserta en el objetivo item{{=item.id}} SPAN.

Devoluciones de llamada de Ajax se puede utilizar para realizar cálculos en un segundo plano, pero le recomendamos que utilice cron o un proceso en segundo plano en su lugar (discutió en el capítulo 4), ya que el servidor web de cumplir un tiempo de espera en las roscas. Si el cálculo toma demasiado tiempo, el servidor web, lo mata. Consulte a su servidor web los parámetros para establecer el valor de tiempo de espera.

11

Recetas de implementación

Hay varias formas de implementar web2py en un entorno de producción, y los detalles dependen de la configuración y los servicios prestados por el anfitrión.

En este capítulo se consideran las siguientes cuestiones:

- Producción de despliegue (Apache, Lighttpd, Cherokee)
- Seguridad
- Escalabilidad
- Implementación en la plataforma de Google App Engine (GAE (13))

web2py viene con un servidor web SSL (21) habilitado, el wsgiserver (22) Rocket. Si bien este es un servidor web rápido, tiene una capacidad limitada de configuración. Por esta razón, es mejor para desplegar web2py detrás Apache (79), Lighttpd (86) o Cherokee (87). Estos son libres y de código abierto servidores web que se pueden personalizar y han demostrado su fiabilidad en entornos de alto tráfico de la producción. Pueden ser configurados para servir archivos estáticos directamente, tratar con HTTPS, y pasar el control a web2py para el contenido dinámico.

Hasta hace unos años, la interfaz estándar para la comunicación entre los servidores web y aplicaciones web era el Common Gateway Interface (CGI) (78). El principal problema con CGI es que crea un nuevo proceso para cada solicitud HTTP. Si la aplicación web está escrita en un lenguaje interpretado, cada solicitud HTTP servido por los scripts CGI se inicia una nueva instancia del intérprete. Esto es lento, y debe ser evitada en un entorno de producción. Por otra parte, CGI sólo puede manejar respuestas simples. No puede manejar, por ejemplo, archivo de secuencias. `web2py modpythonhandler.py` proporciona un archivo a la interfaz a CGI.

Una solución a este problema es utilizar el módulo `mod_python` para Apache. Lo discutiremos aquí porque su uso es todavía muy común, pero el proyecto `mod_python` ha sido oficialmente abandonado por la Apache Software Foundation. `mod_python` se inicia una instancia del intérprete de Python en Apache se inicia, y sirve cada solicitud HTTP en su propio hilo sin tener que reiniciar cada vez que Python. Esta es una solución mejor que la CGI, pero no es una solución ideal, ya que `mod_python` utiliza su propia interfaz para la comunicación entre el servidor web y la aplicación web. En `mod_python`, todos presentados aplicaciones se ejecutan en el marco del user-id/group-id mismo, que presenta problemas de seguridad. `web2py cghandler.py` proporciona un archivo de interfaz para `mod_python`.

En los últimos años, la comunidad de Python se ha unido detrás de una nueva interfaz estándar para la comunicación entre los servidores web y aplicaciones web escritas en Python. Se llama Web Server Gateway Interface (WSGI) (17; 18). `web2py` fue construido en WSGI, y proporciona controladores para el uso de otras interfaces WSGI cuando no está disponible.

Apache es compatible con WSGI a través del módulo `mod_wsgi` (85) desarrollado por Graham Dumpleton. `web2py wsgihandler.py` proporciona un archivo de interfaz para WSGI.

Algunos servicios de alojamiento web no admiten `mod_wsgi`. En este caso, debemos utilizar Apache como un proxy y transmita todas las solicitudes entrantes al `web2py` incorporada en el servidor web (por ejemplo ejecutando

en localhost: 8000).

En ambos casos, con mod_wsgi y / o mod_proxy, Apache puede ser configurado para servir archivos estáticos y tratar directamente con el cifrado SSL, tomando el peso de encima web2py.

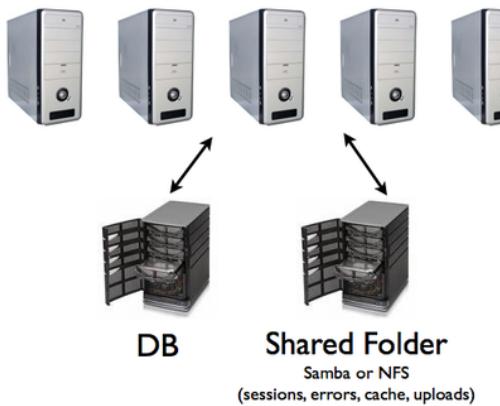
El servidor web Lighttpd no es compatible con la interfaz WSGI, pero es compatible con la interfaz FastCGI (88), que es una mejora con respecto a CGI. FastCGI objetivo principal es reducir la sobrecarga asociada a la interfaz del servidor web y los programas CGI, lo que permite un servidor para manejar más solicitudes de HTTP a la vez.

Según el sitio web Lighttpd, "Lighttpd powers several popular Web 2.0 sites such as YouTube and Wikipedia. Its high speed IO-infrastructure allows them to scale several times better with the same hardware than with alternative web-servers". Lighttpd con FastCGI es, de hecho, más rápido que Apache con mod_wsgi. web2py fcgihandler.py proporciona un archivo a la interfaz de FastCGI. web2py también incluye una gaehandler.py para interactuar con el Google App Engine (GAE). En GAME, aplicaciones Web se ejecuten "in the cloud". Esto significa que el marco completo resúmenes detalles de hardware. La aplicación web se replica automáticamente tantas veces como sea necesario para atender todas las solicitudes concurrentes. Replicación, en este caso significa algo más que varios subprocessos en un único servidor, sino que también significa varios procesos en diferentes servidores. JUEGO alcanza este nivel de escalabilidad al bloquear el acceso de escritura al sistema de archivos y toda la información persistente se debe almacenar en el almacén de datos o en Google BigTable memcache.

En plataformas no-GAE, la escalabilidad es un problema que debe abordarse, y puede requerir algunos ajustes en las solicitudes de web2py. La manera más común para lograr cobertura es mediante el uso de múltiples servidores Web detrás de un equilibrador de carga (un round robin sencillo, o algo más sofisticado, de la retroalimentación latidos del corazón de los servidores).

Aunque hay varios servidores web, debe haber una, y sólo uno, servidor

de base de datos. De forma predeterminada, web2py utiliza el sistema de archivos para almacenar sesiones, entradas de error, los archivos subidos, y la memoria caché. Esto significa que en la configuración predeterminada, las carpetas de los correspondientes carpetas para compartir:



En el resto del capítulo, analizaremos las distintas recetas que puedan brindar una mejora con respecto a este enfoque ingenuo, incluyendo:

- Tienda sesiones en la base de datos en la memoria caché o hacer sesiones no almacenar a todos.
- entradas de la tienda en sistemas de archivos locales y colocarlos en la base de datos en lotes.
- Utilice memcache lugar de cache.ram y cache.disk.
- Guarde los archivos subidos en la base de datos en lugar del sistema de archivos compartidos.

Si bien se recomienda después de los primeros tres recetas, la receta cuarto puede proporcionar una ventaja sobre todo en el caso de archivos pequeños, pero puede ser contraproducente para archivos de gran tamaño.

11.1 Linux / Unix

11.1.1 Un paso de implementación de Producción

Éstos son algunos pasos para instalar apache + python + + + postgresql mod_wsgi web2py desde cero.

En Ubuntu: En Fedora

```

1 wget http://web2py.googlecode.com/svn/trunk/scripts/setup-web2py-ubuntu.sh
2 chmod +x setup-web2py-ubuntu.sh
3 sudo ./setup-web2py-ubuntu.sh
:
```

```

1 wget http://web2py.googlecode.com/svn/trunk/scripts/setup-web2py-fedora.sh
2 chmod +x setup-web2py-fedora.sh
3 sudo ./setup-web2py-fedora.sh
```

Ambos secuencia de comandos debe ejecutarse fuera de la caja sino que cada instalación de Linux es un poco diferente y desea que en el código fuente. Así que asegúrese de ejecutar el código fuente de estas secuencias de comandos antes de ejecutar. La mayor parte de lo que hacen se explica a continuación en el caso de ubuntu. No aplicar las optimizaciones escalabilidad discuten a continuación.

11.1.2 Apache configuración

En esta sección, usamos Ubuntu 8.04 Server Edition como plataforma de referencia. Los comandos de configuración son muy similares en otra distribución de Linux basada en Debian, pero puede variar para los sistemas basados en Fedora (que utiliza en lugar de `yum apt-get`).

En primer lugar, asegúrese de que todas las medidas necesarias Python y

Apache paquetes están instalados escribiendo los comandos de shell siguientes:

```

1 sudo apt-get update
2 sudo apt-get -y upgrade
3 sudo apt-get -y install openssh-server
4 sudo apt-get -y install python
5 sudo apt-get -y install python-dev
6 sudo apt-get -y install apache2
7 sudo apt-get -y install libapache2-mod-wsgi
8 sudo apt-get -y install libapache2-mod-proxy-html

```

A continuación, activar el módulo SSL, el módulo de proxy, y el módulo en Apache WSGI:

```

1 sudo ln -s /etc/apache2/mods-available/proxy_http.load \
2           /etc/apache2/mods-enabled/proxy_http.load
3 sudo a2enmod ssl
4 sudo a2enmod proxy
5 sudo a2enmod proxy_http
6 sudo a2enmod wsgi

```

Crear la carpeta de SSL, y poner los certificados SSL en su interior:

```

1 sudo mkdir /etc/apache2/ssl

```

Usted debe obtener sus certificados SSL de una entidad emisora de certificados de confianza como verisign.com, pero, para fines de comprobación, puede generar sus propios certificados con firma automática siguiendo las instrucciones de ref. (84)

A continuación, reinicie el servidor web:

```

1 sudo /etc/init.d/apache2 restart

```

El archivo de configuración de Apache es:

```

1 /etc/apache2/sites-available/default

```

Los logs de Apache está en:

```

1 /var/log/apache2/

```

11.1.3 mod_wsgi

Descargar y descomprimir web2py fuente en el equipo donde ha instalado el servidor web más arriba.

Instale web2py en `/users/www-data/`, por ejemplo, y se sientan partícipes a www-data de usuario y datos del grupo www-. Estos pasos se pueden realizar con los siguientes comandos de la shell:

```

1 cd /users/www-data/
2 sudo wget http://web2py.com/examples/static/web2py_src.zip
3 sudo unzip web2py_src.zip
4 sudo chown -R www-data:www-data /user/www-data/web2py

```

Para configurar web2py con mod_wsgi, crear una nueva configuración de Apache:

```
1 /etc/apache2/sites-available/web2py
```

e incluir el siguiente código:

```

1 <VirtualHost *:80>
2   ServerName web2py.example.com
3   WSGIDaemonProcess web2py user=www-data group=www-data \
4                           display-name=%{GROUP}
5   WSGIProcessGroup web2py
6   WSGIScriptAlias / /users/www-data/web2py/wsgihandler.py
7
8   <Directory /users/www-data/web2py>
9     AllowOverride None
10    Order Allow,Deny
11    Deny from all
12    <Files wsgihandler.py>
13      Allow from all
14    </Files>
15  </Directory>
16
17  AliasMatch ^/([^\/]+)/static/(.*) \
18    /users/www-data/web2py/applications/$1/static/$2
19  <Directory /users/www-data/web2py/applications/*/*static/>
20    Order Allow,Deny
21    Allow from all
22  </Directory>

```

```

23 <Location /admin>
24 Deny from all
25 </Location>
26
27 <LocationMatch ^/([^\/]+)/appadmin>
28 Deny from all
29 </LocationMatch>
30
31 CustomLog /private/var/log/apache2/access.log common
32 ErrorLog /private/var/log/apache2/error.log
33 </VirtualHost>
34

```

Al reiniciar Apache, hay que aprobar todas las solicitudes de web2py sin pasar por el wsgiserver Rocket.

He aquí algunas explicaciones:

```

1 WSGIDaemonProcess web2py user=www-data group=www-data
2 display-name=%{GROUP}

```

define un grupo de proceso del demonio en el contexto del "web2py.example.com". Al definir este interior de la máquina virtual, sólo que esta máquina virtual, incluyendo cualquier máquina virtual para el mismo nombre de servidor, pero en un puerto diferente, puede acceder a esta utilizando WSGIProcess-Group. Las opciones "user" and "group" se debería establecer en el usuario que tiene acceso de escritura al directorio donde web2py fue instalado. No es necesario configurar "user" y "group", si usted hizo el directorio de instalación web2py de escritura para el usuario que Apache se ejecuta como por defecto. La opción "display-name" es para que nombre del proceso aparece en la salida como ps "(wsgi:web2py)" y no como nombre del servidor web Apache ejecutable. Dado que no se "processes" o "threads" opciones especificado, el grupo de procesos demonio tendrá un solo proceso con 15 subprocesos que se ejecutan dentro de ese proceso. Esto suele ser más que suficiente para la mayoría de los sitios y debe dejarse como está. Si imperiosa, no utilice "processes=1" ya que al hacerlo se desactivará cualquier navegador en herramientas de depuración WSGI que comprueban la bandera "wsgi.multiprocess". Esto es porque cualquier uso de la opción "processes" hará que esa bandera que se establece en true, incluso si un solo proceso y

herramientas tales esperar que se establece en false. Tenga en cuenta que si su propio código de aplicación o algún módulo de extensión tercera parte que está utilizando con Python no es hilo seguro, en lugar utilizar las opciones de "processes=5 threads=1". Esto creará cinco procesos en el grupo de proceso del demonio en el que cada proceso es de un solo subproceso. Le recomendamos que utilice "maximum-requests=1000" si su solicitud de fugas de objetos de Python a través de la incapacidad para que sean recogidos de basura correctamente.

```
1 WSGIProcessGroup web2py
```

delegados de funcionamiento de todas las aplicaciones WSGI al grupo de proceso demonio que se configuró usando la directiva WSGIDaemonProcess.

```
1 WSGIScriptAlias / /users/www-data/web2py/wsgihandler.py
```

monta la aplicación web2py. En este caso, se monta en la raíz del sitio web. No se sabe cómo llegar web2py para montar en un sub URL, no parece ser un buen ciudadano y WSGI punto donde está montado a partir del valor de SCRIPT_NAME y automáticamente ajustar todas las cosas adecuadamente sin configuración manual del usuario.

```
1 <Directory /users/www-data/web2py>
2   ...
3 </Directory>
```

Apache da permiso para tener acceso al archivo de comandos WSGI.

```
1 <Directory /users/www-data/web2py/applications/*/*static/>
2   Order Allow,Deny
3   Allow from all
4 </Directory>
```

Encarga a Apache para eludir web2py la hora de buscar los archivos estáticos.

```
1 <Location /admin>
2   Deny from all
3 </Location>
```

y

```

1 <LocationMatch ^/([^\+]+)/appadmin>
2   Deny from all
3 </LocationMatch>
```

bloquear el acceso del público a los **admin** y **appadmin**

Normalmente sólo permitiría el permiso para el directorio completo script se encuentra en el expediente de la WSGI, pero no puedo hacer eso con web2py, ya que coloca el archivo de comandos WSGI en un directorio que contiene el código fuente, incluyendo el archivo que contiene la contraseña de la interfaz de administración. La apertura de todo el directorio podría causar problemas de seguridad, porque técnicamente Apache se le daría permiso para servir a todos los archivos a un usuario si había alguna forma de atravesar a ese directorio a través de una dirección asignada. Para evitar problemas de seguridad, de manera explícita denegar el acceso a el contenido del directorio, a excepción de la secuencia de comandos WSGI archivo y prohibir a un usuario de hacer algo anula de un htaccess. archivo a ser más seguro.

Usted puede encontrar una completa, comentó, WSGI archivo de configuración de Apache en:

```
1 scripts/web2py-wsgi.conf
```

Esta sección fue creada con la ayuda de Graham Dumpleton, desarrollador de mod_wsgi.

11.1.4 mod_wsgi y SSL

Para obligar a que algunas aplicaciones (por ejemplo, **admin** y **appadmin**) para ir a través de HTTPS, almacenar el certificado SSL y los archivos de clave:

```

1 /etc/apache2/ssl/server.crt
2 /etc/apache2/ssl/server.key
```

y editar el archivo de configuración de Apache y añadir web2py.conf:

```

1 <VirtualHost *:443>
2   ServerName web2py.example.com
3   SSLEngine on
4   SSLCertificateFile /etc/apache2/ssl/server.crt
5   SSLCertificateKeyFile /etc/apache2/ssl/server.key
6
7   WSGIProcessGroup web2py
8
9   WSGIScriptAlias / /users/www-data/web2py/wsgihandler.py
10
11  <Directory /users/www-data/web2py>
12    AllowOverride None
13    Order Allow,Deny
14    Deny from all
15    <Files wsgihandler.py>
16      Allow from all
17    </Files>
18  </Directory>
19
20  AliasMatch ^/([^\/]+)/static/(.*) \
21    /users/www-data/web2py/applications/$1/static/$2
22
23  <Directory /users/www-data/web2py/applications/*/*static/>
24    Order Allow,Deny
25    Allow from all
26  </Directory>
27
28  CustomLog /private/var/log/apache2/access.log common
29  ErrorLog /private/var/log/apache2/error.log
30
31 </VirtualHost>
```

Reiniciar Apache y usted debe poder tener acceso a:

```

1 https://www.example.com/admin
2 https://www.example.com/examples/appadmin
3 http://www.example.com/examples
```

pero no:

```

1 http://www.example.com/admin
2 http://www.example.com/examples/appadmin
```

11.1.5 mod_proxy

Algunos Unix / Linux se puede ejecutar Apache, pero no son compatibles mod_wsgi. En este caso, la solución más simple es ejecutar Apache como un proxy y tienen mucho Apache con los archivos estáticos.

Aquí está un minimalista de configuración de Apache:

```

1 NameVirtualHost *:80
2 ### deal with requests on port 80
3 <VirtualHost *:80>
4   Alias / /users/www-data/web2py/applications
5   ### serve static files directly
6   <LocationMatch "^/welcome/static/.*">
7     Order Allow, Deny
8     Allow from all
9   </LocationMatch>
10  ### proxy all the other requests
11  <Location "/welcome">
12    Order deny,allow
13    Allow from all
14    ProxyRequests off
15    ProxyPass http://localhost:8000/welcome
16    ProxyPassReverse http://localhost:8000/
17    ProxyHTMLURLMap http://127.0.0.1:8000/welcome/ /welcome
18  </Location>
19  LogFormat "%h %l %u %t \"%r\" %>s %b" common
20  CustomLog /var/log/apache2/access.log common
21 </VirtualHost>
```

El script anterior sólo expone la aplicación "welcome" application. To expose other applications, you need to add the corresponding <Location>...</Location> with the same syntax as done for the "welcome".

El script asume que hay un servidor web2py en el puerto 8000. Antes de reiniciar Apache, asegúrese de que éste es el caso:

```
1 nohup python web2py.py -a '<recycle>' -i 127.0.0.1 -p 8000 &
```

Puede especificar una contraseña con la opción -a o "<recycle>" utiliza el parámetro en lugar de una contraseña. En este último caso, la contraseña

previamente almacenada se reutiliza y la contraseña no se almacena en la historia de shell.

También puede utilizar el parámetro de "<ask>", que se le pide una contraseña.

Los comandos `nohup` hace que el servidor no se muere cuando se cierra la concha. `nohup` registra toda la salida en `nohup.out`.

Para forzar admin y appadmin a través de HTTPS utiliza el siguiente archivo de configuración de Apache en su lugar:

```
1 NameVirtualHost *:80
2 NameVirtualHost *:443
3 ### deal with requests on port 80
4 <VirtualHost *:80>
5     Alias / /usres/www-data/web2py/applications
6     ### admin requires SSL
7     <LocationMatch "^/admin">
8         SSLRequireSSL
9     </LocationMatch>
10    ### appadmin requires SSL
11    <LocationMatch "^/welcome/appadmin/.*">
12        SSLRequireSSL
13    </LocationMatch>
14    ### serve static files directly
15    <LocationMatch "^/welcome/static/.*">
16        Order Allow,Deny
17        Allow from all
18    </LocationMatch>
19    ### proxy all the other requests
20    <Location "/welcome">
21        Order deny,allow
22        Allow from all
23        ProxyPass http://localhost:8000/welcome
24        ProxyPassReverse http://localhost:8000/
25    </Location>
26    LogFormat "%h %l %u %t \"%r\" %>s %b" common
27    CustomLog /var/log/apache2/access.log common
28 </VirtualHost>
29 <VirtualHost *:443>
30     SSLEngine On
31     SSLCertificateFile /etc/apache2/ssl/server.crt
32     SSLCertificateKeyFile /etc/apache2/ssl/server.key
33     <Location "/">
```

```

34     Order deny,allow
35     Allow from all
36     ProxyPass http://localhost:8000/
37     ProxyPassReverse http://localhost:8000/
38   </Location>
39   LogFormat "%h %l %u %t \"%r\" %>s %b" common
40   CustomLog /var/log/apache2/access.log common
41 </VirtualHost>
```

La interfaz administrativa debe desactivarse cuando web2py se ejecuta en un servidor compartido con mod_proxy, o pueda estar expuesto a otros usuarios.

11.1.6 Inicio como Daemon Linux

A menos que utilice mod_wsgi, debe configurar el servidor web2py para que pueda ser iniciado / parado / reiniciar el demonio como cualquier otro Linux, y para que pueda iniciarse automáticamente en la fase de arranque del ordenador.

El proceso para realizar esta configuración es específica para diferentes distribuciones Linux / Unix.

En la carpeta web2py, hay dos secuencias de comandos que pueden utilizarse para este propósito:

```

1 scripts/web2py.ubuntu.sh
2 scripts/web2py.fedora.sh
```

En Ubuntu, Debian u otra distribución de Linux basada, editar ruta "web2py.ubuntu.sh" and replace the "/usr/lib/web2py" con la ruta de su instalación web2py, a continuación, escriba los comandos shell siguiente para mover el archivo a la carpeta adecuada, registrarla como un servicio de inicio, y ponerlo en marcha:

```

1 sudo cp scripts/web2py.ubuntu.sh /etc/init.d/web2py
2 sudo update-rc.d web2py defaults
3 sudo /etc/init.d/web2py start
```

En Fedora, o cualquier otra distribución basada en Fedora, edite ruta "web2py.fedora.sh" and replace the "/usr/lib/web2py" con la ruta de su instalación web2py, a continuación, escriba los comandos shell siguiente para mover el archivo a la carpeta adecuada, registrarla como el inicio del servicio y ponerlo en marcha:

```

1 sudo cp scripts/web2py.fedora.sh /etc/rc.d/init.d/web2pyd
2 sudo chkconfig --add web2pyd
3 sudo service web2py start

```

11.1.7 Lighttpd

Puede instalar Lighttpd en un Ubuntu u otra distribución de Linux basada en Debian, con el comando siguiente:

```
1 apt-get -y install lighttpd
```

Una vez instalado, editar y crear una /etc/rc.local

```
1 cd /var/www/web2py && sudo -u www-data nohup python fcgihandler.py &
```

web2py fondo fcgi proceso

Entonces, es necesario modificar la configuración de Lighttpd archivo

```
1 /etc/lighttpd/lighttpd.conf
```

para que pueda encontrar el socket creado por el proceso anterior. En el archivo de configuración, escribe algo como:

```

1 server.modules          =
2   "mod_access",
3   "mod_alias",
4   "mod_compress",
5   "mod_rewrite",
6   "mod_fastcgi",
7   "mod_redirect",
8   "mod_accesslog",
9   "mod_status",
10 )

```

```

11
12 server.port = 80
13 server.bind = "0.0.0.0"
14 server.event-handler = "freebsd-kqueue"
15 server.error-handler-404 = "/test.fcgi"
16 server.document-root = "/users/www-data/web2py/"
17 server.errorlog      = "/tmp/error.log"
18
19 fastcgi.server = (
20     "/handler_web2py.fcgi" => (
21         "handler_web2py" => ( #name for logs
22             "check-local" => "disable",
23             "socket" => "/tmp/fcgi.sock"
24         )
25     ),
26 )
27
28 $HTTP["host"] = "(^|\.example\.com$" {
29     server.document-root="/var/www/web2py"
30     url.rewrite-once = (
31         "^^(.+/static/.+)$" => "/applications$1",
32         "(^|/.*$" => "/handler_web2py.fcgi$1",
33     )
34 }
```

Ahora comprobar errores de sintaxis:

```
1 lighttpd -t -f /etc/lighttpd/lighttpd.conf
```

y (re) iniciar el servidor web con:

```
1 /etc/init.d/lighttpd restart
```

Tenga en cuenta que se une el servidor FastCGI web2py a una toma de Unix, no a una toma de propiedad intelectual:

```
1 /tmp/fcgi.sock
```

Aquí es donde Lighttpd reenvía las peticiones HTTP y recibe respuestas de. sockets Unix son más ligeros que tomas de internet, y esta es una de las razones Lighttpd + FastCGI + web2py es rápido. Como en el caso de Apache, es posible configurar Lighttpd para tratar directamente los archivos estáticos, y

obligar a algunas aplicaciones a través de HTTPS. Consulte la documentación de Lighttpd para más detalles.

Los ejemplos en esta sección fueron tomadas de John Heenan puesto en [web2pyslices](#).

La interfaz administrativa debe desactivarse cuando web2py se ejecuta en un servidor compartido con FastCGI, o pueda estar expuesta a los otros usuarios.

11.1.8 Alojamiento Compartido con mod_python

Hay ocasiones, concretamente en el intercambio de los ejércitos, cuando uno no tiene el permiso para configurar la configuración de Apache archivos directamente. En el momento de escribir la mayoría de estas máquinas siguen ejecutando mod_python incluso si no se mantiene más en favor de mod_wsgi. Aún puede ejecutar web2py. Aquí mostramos un ejemplo de cómo configurarlo.

Ponga el contenido de la carpeta web2py en "htdocs".

En la carpeta web2py, crear un Archivo "web2py_modpython.py" con el siguiente contenido:

```
1 from mod_python import apache
2 import modpythonhandler
3
4 def handler(req):
5     req.subprocess_env['PATH_INFO'] = req.subprocess_env['SCRIPT_URL']
6     return modpythonhandler.handler(req)
```

Crear o actualizar el archivo ".htaccess" con el siguiente contenido:

```
1 SetHandler python-program
2 PythonHandler web2py_modpython
3 #PythonDebug On
```

Este ejemplo fue proporcionado por Niktar.

11.1.9 Cherokee con FastGGI

Cherokee es un servidor web muy rápido y, como web2py, proporciona una habilitada para AJAX basado en interfaz web para su configuración. Su interfaz web está escrita en Python. Además, existe no requiere reiniciar para la mayoría de los cambios.

Éstos son los pasos necesarios para la instalación con web2py Cherokee:

Descargar Cherokee (87)

Descomprimir, compilar e instalar:

```

1 tar -xzf cherokee-0.9.4.tar.gz
2 cd cherokee-0.9.4
3 ./configure --enable-fcgi && make
4 make install

```

Inicio web2py lo menos una vez para asegurarse de que crea la carpeta "applications".

Escriba un script de shell llamado "startweb2py.sh" con el siguiente código:

```

1 #!/bin/bash
2 cd /var/web2py
3 python /var/web2py/fcgihandler.py &

```

y dar la secuencia de comandos ejecutar privilegios y ejecutarlo. Esto iniciará web2py en controlador de FastCGI.

Inicio Cherokee y Cherokee-admin:

```

1 sudo nohup cherokee &
2 sudo nohup cherokee-admin &

```

De forma predeterminada, cherokee-admin sólo escucha en la interfaz local en el puerto 9090. Esto no es un problema si usted tiene acceso completo, físico de esa máquina. Si este no es el caso, puede obligarlo a asociarse a una dirección IP y el puerto mediante el uso de las siguientes opciones:

```
1 -b, --bind[=IP]
2 -p, --port=NUM
```

o hacer un puerto SSH-forward (más seguro, recomendado):

```
1 ssh -L 9090:localhost:9090 remotehost
```

Abrir "http://localhost:9090" en su navegador. Si todo está correcto, usted conseguirá cherokee-admin.

En la interfaz web cherokee-admin, haga clic en "info sources". Choose "Local Interpreter". Write in the following code, then click "Add New".

```
1 Nick: web2py
2 Connection: /tmp/fcgi.sock
3 Interpreter: /var/web2py/startweb2py.sh
```

Por último, realice los siguientes pasos restantes:

- Haga clic en "Virtual Servers", then click "Default".
- Haga clic en "Behavior", then, under that, click "default".
- Elija "FastCGI" instead of "List and Send" en el cuadro de lista.
- En la parte inferior, seleccione "web2py" as "Application Server"
- Ponga una marca en todas las casillas de verificación (se puede dejar Permitir-x-sendfile). Si hay una advertencia que se muestra, desactivar y activar una de las casillas de verificación. (Automáticamente se vuelve a presentar el parámetro de servidor de aplicaciones. A veces no, que es un error) lo hace.
- Apunte su navegador a "http://yoursite", and "Welcome to web2py" aparecerá.

11.1.10 PostgreSQL

PostgreSQL es una base de datos libre y de código abierto que se utiliza en entornos de producción, por ejemplo, para almacenar el archivo. Es la base de datos de nombres de dominio, y se ha demostrado que escala bien en cientos de terabytes de datos. Tiene muy rápido y sólido soporte de transacciones, y proporciona una característica de auto-vacío que libera al administrador de base de datos de la mayoría de las tareas de mantenimiento.

En una Ubuntu u otra distribución de Linux basada en Debian-, es fácil de instalar PostgreSQL y su API con Python:

```
1 sudo apt-get -y install postgresql
2 sudo apt-get -y install python-psycopg2
```

Es una buena idea para ejecutar el servidor web (s) y el servidor de base de datos en diferentes máquinas. En este caso, los equipos que ejecutan los servidores web deben estar conectados con un seguro interno (física) de la red, o debe establecer túneles SSL para conectarse de forma segura con el servidor de base de datos.

Editar la configuración de PostgreSQL archivo

```
1 sudo nano /etc/postgresql/8.4/main/postgresql.conf
```

y asegúrese de que contenga estas dos líneas

```
1 ...
2 track_counts = on
3 ...
4 autovacuum = on    # Enable autovacuum subprocess? 'on'
5 ...
```

Inicie el servidor de base de datos con:

```
1 sudo /etc/init.d/postgresql restart
```

Al volver a arrancar el servidor PostgreSQL, lo notificará a qué puerto se está

ejecutando. A menos que tenga varios servidores de base de datos, debe ser 5432.

Los registros de PostgreSQL en:

```
1 /var/log/postgresql/
```

Una vez que el servidor de base de datos está en marcha, crear un usuario y una base de datos para que las aplicaciones web2py puede utilizarlo:

```
1 sudo -u postgres createuser -PE -s myuser
2 postgresql> createdb -O myself -E UTF8 mydb
3 postgresql> echo 'The following databases have been created:'
4 postgresql> psql -l
5 postgresql> psql mydb
```

El primero de los comandos de superusuario otorgará acceso al nuevo usuario, llamado `myuser`. Se le solicitará una contraseña.

Toda solicitud web2py pueden conectarse a esta base de datos con el comando:

```
1 db = DAL("postgres://myuser:mypassword@localhost:5432/mydb")
```

donde `mypassword` es la contraseña que ha entrado cuando se le solicite, y 5432 es el puerto donde el servidor de base de datos se está ejecutando.

Normalmente se utiliza una base de datos para cada aplicación, y varias instancias de la misma aplicación conectarse a la misma base de datos. También es posible que diferentes aplicaciones para compartir la misma base de datos.

Para obtener más información de copia de seguridad de base de datos, lea la documentación de PostgreSQL, y específicamente sobre los comandos de `pg_dump` y `pg_restore`.

11.2 Windows

11.2.1 Apache y mod_wsgi

Instalación de Apache, y mod_wsgi en Windows requiere un procedimiento diferente. Aquí están asumiendo Python 2.5 está instalado, se ejecuta desde el origen y web2py está situado en la c:/web2py.

En primer lugar descargar el paquete requiere:

- Apache apache_2.2.11-win32-x86-openssl-0.9.8i.msi de (80)
- mod_wsgi de (81)

En segundo lugar, apache...msi ejecutar y seguir las pantallas del asistente. En la pantalla de información del servidor



ingresar todos los valores solicitados:

- **Network Domain:** entrar en el dominio DNS en el que su servidor esté o vaya a ser registrado pulg Por ejemplo, si el nombre completo de su servidor DNS es server.mydomain.net, debería escribir aquí mydomain.net

- **ServerName:** nombre completo de su servidor DNS. En el ejemplo anterior, debería escribir server.mydomain.net aquí. Escriba un nombre de dominio completo o la dirección IP de la instalación no web2py, un acceso directo, para más información ver (83).
- **Administrator's Email Address.** Introduzca el administrador del servidor o la dirección del webmaster e-mail aquí. Esta dirección se mostrará junto con los mensajes de error al cliente de forma predeterminada.

Continuar con una instalación típica hasta el final a no ser que lo exijan

El asistente, por defecto, Apache instalado en la carpeta:

```
1 C:/Program Files/Apache Software Foundation/Apache2.2/
```

A partir de ahora nos referiremos a esta carpeta simplemente como Apache2.2.

En tercer lugar, copie el mod_wsgi.so descargado para Apache2.2/modules escrito por Chris Travers, publicado por el Open Source Lab Software de Microsoft, diciembre de 2007.

En cuarto lugar, crear certificados server.crt y server.key (como se explica en la sección anterior) y colocarlos en la carpeta de Apache2.2/conf. Observe el archivo CNF está en Apache2.2/conf/openssl.cnf.

En quinto lugar, editar Apache2.2/conf/httpd.conf, quite la marca de comentario (el # carácter) desde la línea de

```
1 LoadModule ssl_module modules/mod_ssl.so
```

añadir la siguiente línea después de todas las otras líneas LoadModule

```
1 LoadModule wsgi_module modules/mod_wsgi.so
```

buscar "Listen 80" y añadir la línea después de que

```
1 Listen 443
```

agregar las siguientes líneas al final cambiar letra de unidad, número de puerto, ServerName de acuerdo a sus valores

```

1 NameVirtualHost *:443
2 <VirtualHost *:443>
3   DocumentRoot "C:/web2py/applications"
4   ServerName server1
5
6   <Directory "C:/web2py">
7     Order allow,deny
8     Deny from all
9   </Directory>
10
11  <Location "/">
12    Order deny,allow
13    Allow from all
14  </Location>
15
16  <LocationMatch "^(/[\w_]*static/.*)">
17    Order Allow,Deny
18    Allow from all
19  </LocationMatch>
20
21 WSGIScriptAlias / "C:/web2py/wsgihandler.py"
22
23 SSLEngine On
24 SSLCertificateFile conf/server.crt
25 SSLCertificateKeyFile conf/server.key
26
27 LogFormat "%h %l %u %t \"%r\" %>s %b" common
28 CustomLog logs/access.log common
29 </VirtualHost>
```

Guardar y comprobar la configuración utilizando: [Inicio> Programas> Apache HTTP Server 2.2> Configurar Apache Server> Configuración de las pruebas]

Si no hay problemas, verá una pantalla de mando de apertura y cierre. Ahora puede comenzar a Apache:

[Inicio> Programas> Servidor HTTP Apache 2.2> Control> Servidor Apache Inicio] o mejor aún, iniciar la barra de tareas de seguimiento

[Start > Program > Apache HTTP Server 2.2 > Control Apache Server]

Ahora puedes hacer clic derecho sobre el icono de pluma roja como la barra de tareas para abrir el Apache Monitor y desde ella iniciar, parar y reiniciar Apache según sea necesario.

Esta sección fue creada por Jonathan Lundell.

11.2.2 Inicio como servicio de Windows

Lo que llama a un demonio de Linux, Windows llama a un servicio. Web2py El servidor puede ser fácilmente instalado / marcha / parado como servicio de Windows.

Para poder utilizar web2py como servicio de Windows, debe crear un archivo con "options.py" parámetros de inicio:

```

1 import socket, os
2 ip = socket.gethostname()
3 port = 80
4 password = '<recycle>'
5 pid_filename = 'httpserver.pid'
6 log_filename = 'httpserver.log'
7 ssl_certificate =
8 ssl_private_key =
9 numthreads = 10
10 server_name = socket.gethostname()
11 request_queue_size = 5
12 timeout = 10
13 shutdown_timeout = 5
14 folder = os.getcwd()
```

No es necesario crear en la carpeta "options.py" from scratch since there is already an "options_std.py" web2py que puede utilizar como modelo.

Después de crear "options.py" en la carpeta de instalación web2py, puede instalar como un servicio web2py con:

```
1 python web2py.py -W install
```

e iniciar / detener el servicio con:

```
1 python web2py.py -W start
2 python web2py.py -W stop
```

11.3 Asegurar Sesiones y admin

Es muy peligroso para exponer públicamente la aplicación **admin** y **appadmin** menos que los controladores se ejecutan a través de HTTPS. Por otra parte, su contraseña y credenciales nunca deben ser transmitidos sin protección. Esto es cierto para web2py y cualquier otra aplicación Web.

En sus aplicaciones, si se requiere autenticación, debe hacer las galletas sesión segura con:

```
1 session.secure()
```

Una manera fácil de configurar un entorno de producción segura en un servidor es la primera parada web2py ya continuación, quite todos los archivos `parameters_*.py` de la carpeta de instalación web2py. A continuación, iniciar web2py sin contraseña. Esto deshabilitará completamente admin y appadmin.

A continuación, iniciar una segunda instancia Python, accesible exclusivamente desde localhost:

```
1 nohup python web2py -p 8001 -i 127.0.0.1 -a '<ask>' &
```

y crear un túnel SSH de la máquina local (la una de la que desea acceder a la interfaz de administración) al servidor (en la que se está ejecutando web2py, example.com), utilizando:

```
1 ssh -L 8001:127.0.0.1:8001 username@example.com
```

Ahora puedes acceder a la interfaz de administración a nivel local a través del navegador web en `localhost:8001`.

Esta configuración es seguro porque **admin** no es accesible cuando el túnel está cerrado (el usuario está desconectado).

— Esta solución es segura en los servidores compartidos si y sólo si otros usuarios no tienen acceso de lectura a la carpeta que contiene web2py, de lo contrario los usuarios puede ser capaz de robar las cookies de sesión directamente desde el servidor. — —

11.4 Eficiencia y Escalabilidad

web2py está diseñado para ser fácil de instalar y de configurar. Esto no significa que se compromete en la eficiencia o capacidad de ampliación, pero significa que usted puede necesitar ajustar para que sea escalable.

En esta sección se asume múltiples instalaciones web2py detrás de un servidor NAT que proporciona locales de equilibrio de carga.

En este caso, web2py trabaja fuera de la caja, si se cumplen determinadas condiciones. En particular, todas las instancias de cada aplicación web2py debe acceder a la misma base de datos y servidores que ver los mismos archivos. Esta última condición puede ser aplicado por hacer las siguientes carpetas compartidas:

```

1 applications/myapp/sessions
2 applications/myapp/errors
3 applications/myapp/uploads
4 applications/myapp/cache

```

Las carpetas para compartir, debe ser compatible con el bloqueo de archivos. Las posibles soluciones son de ZFS (ZFS fue desarrollado por Sun Microsystems y es la opción preferida.), NFS (Con NFS puede ser necesario para ejecutar `thenlockmgr` demonio para permitir el bloqueo de archivos.) o Samba (SMB).

Es posible, pero no una buena idea, para compartir la carpeta web2py entero o la carpeta de aplicaciones completas, porque esto causaría un aumento innecesario de uso de la red de ancho de banda.

Creemos que la configuración discutido anteriormente es muy escalable, ya que reduce la carga de base de datos de trasladarse a los sistemas de archivos para compartir los recursos que deben ser compartidas, pero no necesitan la seguridad de transacciones (sólo cliente uno a la vez se supone que debe tener acceso a un archivo de sesión, cache siempre necesita un bloqueo global, las cargas y los errores son una escritura / lectura de muchos archivos).

Lo ideal sería que tanto la base de datos y el almacenamiento compartido debería tener la capacidad de RAID. No cometa el error de guardar la base de datos en el mismo almacenamiento como las carpetas compartidas, o va a crear un cuello de botella de nuevo allí.

Sobre una base caso por caso, es posible que necesite para llevar a cabo optimizaciones adicionales y las discutiremos más adelante. En particular, vamos a discutir la manera de deshacerse de estas carpetas para compartir, uno por uno, y la forma de almacenar los datos asociados en la base de datos en su lugar. Aunque esto es posible, no es necesariamente una buena solución. Sin embargo, puede haber razones para hacerlo. Uno de estos motivos es que a veces no tenemos la libertad de creación de carpetas compartidas.

11.4.1 Eficiencia trucos

código de la aplicación se ejecuta web2py en cada petición por lo que desea reducir al mínimo esta cantidad de código. Esto es lo que puede hacer:

- Ejecutar una vez con `migrate=True` continuación, establecer que todas sus tablas `migrate=False`.
- Bytecode compilar su aplicación con `admin`.

- Use todo lo que puede, pero asegúrese de usar un conjunto finito de teclas más la cantidad de caché utilizada crecerá arbitrariamente `cache.ram`.
- Minimizar el código en los modelos: no definen las funciones de allí, definir funciones en los controladores que necesita o - mejor aún - definir funciones en módulos, la importación y utilizar esas funciones como sea necesario.
- No ponga muchas funciones en el mismo controlador pero el uso de los controladores de muchos de ellos con pocas funciones.
- `session.forget()` Situado en todos los controladores y / o funciones que no cambian la sesión.
- Trate de vacío web2py cron, y el uso de un proceso de fondo. cron web2py puede comenzar Python demasiados casos y causar el uso excesivo de memoria.

11.4.2 Sesiones en base de datos

Es posible dar instrucciones web2py para almacenar las sesiones en una base de datos en lugar de en la carpeta de sesiones. Esto tiene que hacerse para cada aplicación individual web2py a pesar de que todos puedan utilizar la misma base de datos a las sesiones de la tienda.

Teniendo en cuenta una conexión de base

```
1 db = DAL(...)
```

puedes guardar las sesiones en esta base de datos (`db`) simplemente señalando lo siguiente, en el archivo mismo modelo que establece la conexión:

```
1 session.connect(request, response, db)
```

Si no existe ya, web2py crea, bajo la campana, una tabla en la base de datos llamada `web2py_session_appname` contiene los siguientes campos:

```

1 Field('locked', 'boolean', default=False),
2 Field('client_ip'),
3 Field('created_datetime', 'datetime', default=now),
4 Field('modified_datetime', 'datetime'),
5 Field('unique_key'),
6 Field('session_data', 'text')

```

"unique_key" es una clave uuid utilizada para identificar el período de sesiones en la cookie. "session_data" es el cPickled datos de la sesión.

Para minimizar el acceso de base de datos, se debe evitar el almacenamiento de sesiones cuando no se necesitan con:

```
1 session.forget()
```

Con este truco la carpeta "sessions" no tiene por qué ser una carpeta compartida, ya que dejará de ser visitada. — Tenga en cuenta que, si las sesiones son discapacitados, no debe pasar el `session` y no puede utilizar `form.accepts` `session.flash` ni `CRUD`. —

11.4.3 HAProxy una alta disponibilidad de equilibrador de carga

Si necesita web2py múltiples procesos que se ejecutan en varios equipos, en lugar de las sesiones de almacenamiento en la base de datos o en la memoria caché, usted tiene la opción de utilizar un equilibrador de carga con sesiones pegajosa.

Pound (89) HAProxy (90) y dos balanceadores de carga HTTP y proxy inverso que proporciona sesiones pegajosa. Aquí hablamos de la segunda porque parece ser más común en VPS alojamiento comercial.

Mediante sesiones pegajosa, nos referimos a que una vez que una cookie de sesión se ha publicado, el balanceador de carga siempre enrutar las solicitudes del cliente asociado a la sesión, en el mismo servidor. Esto le permite guardar la sesión en el sistema de ficheros local, sin necesidad de un sistema

de archivos compartidos.

Para utilizar HAProxy:

En primer lugar, instalar, en Ubuntu máquina de prueba de salida:

```
1 sudo apt-get -y install haproxy
```

En segundo lugar editar el fichero de configuración "/etc/haproxy.cfg" a algo como esto:

```
1 # this config needs haproxy-1.1.28 or haproxy-1.2.1
2
3 global
4     log 127.0.0.1    local0
5     maxconn 1024
6     daemon
7
8 defaults
9     log      global
10    mode     http
11    option   httplog
12    option   httpchk
13    option   httpclose
14    retries 3
15    option  redispatch
16    contimeout    5000
17    clitimeout    50000
18    srvtimeout    50000
19
20 listen 0.0.0.0:80
21     balance url_param WEB2PYSTICKY
22     balance roundrobin
23     server L1_1 10.211.55.1:7003 check
24     server L1_2 10.211.55.2:7004 check
25     server L1_3 10.211.55.3:7004 check
26     appsession WEB2PYSTICKY len 52 timeout 1h
```

La directiva dice HAProxy `listen`, el puerto que se espera de conexión desde el. `server` La directiva dice HAProxy dónde encontrar los servidores proxy. El directorio `appsession` hace una sesión pegajoso y utiliza la cookie de una llamada `WEB2PYSTICKY` para este fin.

En tercer lugar, permitir a este archivo de configuración y comenzar HAProxy:

```
1 /etc/init.d/haproxy restart
```

Puede encontrar instrucciones de configuración similar a la libra en la URL

```
1 http://web2pyslices.com/main/slices/take_slice/33
```

11.4.4 Limpieza de Sesiones

Si opta por conservar sus sesiones en el sistema de archivos, debe tener en cuenta que en un entorno de producción que se acumulan rápidamente. web2py proporciona un script llamado:

```
1 scripts/sessions2trash.py
```

que cuando se ejecuta en segundo plano, borra periódicamente todas las sesiones que no han sido visitada por una cierta cantidad de tiempo. Este es el contenido del guión:

```
1 SLEEP_MINUTES = 5
2 EXPIRATION_MINUTES = 60
3 import os, time, stat
4 path = os.path.join(request.folder, 'sessions')
5 while 1:
6     now = time.time()
7     for file in os.listdir(path):
8         filename = os.path.join(path, file)
9         t = os.stat(filename)[stat.ST_MTIME]
10        if now - t > EXPIRATION_MINUTES * 60:
11            os.unlink(filename)
12        time.sleep(SLEEP_MINUTES * 60)
```

Usted puede ejecutar el script con el siguiente comando:

```
1 nohup python web2py.py -S myapp -R scripts/sessions2trash.py &
```

miaplicacion donde es el nombre de su aplicación.

11.4.5 Cargar archivos en la base de datos

Por defecto, todos los archivos subidos a cargo de SQLFORMs de forma segura nombre y se almacena en el sistema de archivos en la carpeta de "uploads". Es posible dar instrucciones web2py para almacenar los archivos cargados en la base de datos en su lugar.

Ahora, considere la siguiente tabla:

```

1 db.define_table('dog',
2     Field('name')
3     Field('image', 'upload'))
```

donde dog.image es de subida de tipos. Para hacer que la imagen cargada ir en el mismo registro que el nombre del perro, debe modificar la definición de tabla mediante la adición de un campo blob y vincularlo a la opción adjuntar:

```

1 db.define_table('dog',
2     Field('name')
3     Field('image', 'upload', uploadfield='image_data'),
4     Field('image_data', 'blob'))
```

Aquí "image_data" es sólo un nombre arbitrario para el campo blob nuevo.

Línea 3 instruye web2py para cambiar el nombre con seguridad las imágenes subidas como de costumbre, guardar el nuevo nombre en el campo de la imagen, y almacenar los datos en el uploadfield llamada "image_data" en lugar de almacenar los datos en el sistema de archivos. Todo esto se hará automáticamente por SQLFORMs y ningún otro tipo de código debe ser cambiado.

Con este truco, la carpeta "uploads" ya no es necesario.

El Google App Engine archivos se almacenan por defecto en la base de datos sin necesidad de definir una uploadfield, uno se crea de forma predeterminada.

11.4.6 Recogida de entradas

De forma predeterminada, los boletos tiendas web2py (errores) en el sistema de archivos local. No tendría sentido para almacenar los boletos directamente en la base de datos, ya que el origen más común de errores en un entorno de producción es la falta de bases de datos.

Almacenamiento de los boletos no es nunca un cuello de botella, porque esto es normalmente un evento raro, por tanto, en un entorno de producción con varios servidores simultáneamente, es más que suficiente para que los guarde en un carpeta compartida. Sin embargo, dado que sólo el administrador tiene que recuperar los boletos, también es OK para guardar los boletos en una carpeta compartida no "errors" local y periódicamente su recopilación y / o borrarlos.

Una posibilidad es pasar periódicamente todos los billetes locales para una base de datos.

Con este fin, web2py proporciona el siguiente script:

```
1 scripts/tickets2db.py
```

que contiene:

```
1 import sys
2 import os
3 import time
4 import stat
5 import datetime
6
7 from gluon.utils import md5_hash
8 from gluon.restricted import RestrictedError
9
10 SLEEP_MINUTES = 5
11 DB_URI = 'sqlite://tickets.db'
12 ALLOW_DUPLICATES = True
13
14 path = os.path.join(request.folder, 'errors')
15
16 db = SQLDB(DB_URI)
```

```

17 db.define_table('ticket', SQLField('app'), SQLField('name'),
18                 SQLField('date_saved', 'datetime'), SQLField('layer'),
19                 SQLField('traceback', 'text'), SQLField('code', 'text'))
20
21 hashes = {}
22
23 while 1:
24     for file in os.listdir(path):
25         filename = os.path.join(path, file)
26
27     if not ALLOW_DUPLICATES:
28         file_data = open(filename, 'r').read()
29         key = md5_hash(file_data)
30
31     if key in hashes:
32         continue
33
34     hashes[key] = 1
35
36     error = RestrictedError()
37     error.load(request, request.application, filename)
38
39     modified_time = os.stat(filename)[stat.ST_MTIME]
40     modified_time = datetime.datetime.fromtimestamp(modified_time)
41
42     db.ticket.insert(app=request.application,
43                      date_saved=modified_time,
44                      name=file,
45                      layer=error.layer,
46                      traceback=error.traceback,
47                      code=error.code)
48
49     os.unlink(filename)
50
51     db.commit()
52     time.sleep(SLEEP_MINUTES * 60)

```

Este script debe ser editado. Cambie la cadena de DB_URI de manera que se conecta al servidor de base de datos y ejecutarlo con el comando:

```
1 nohup python web2py.py -S miapp -M -R scripts/tickets2db.py &
```

miaplicacion donde es el nombre de su aplicación.

Este script se ejecuta en segundo plano y cada 5 minutos se mueve todos

los boletos para el servidor de base de datos en una tabla llamada "ticket" y elimina las entradas locales. Si ALLOW_DUPLICATES se establece en False, sólo guardará los billetes que corresponden a diferentes tipos de errores. Con este truco, la carpeta "errors" no tiene por qué ser una carpeta compartida más, ya que sólo se tendrá acceso a nivel local.

11.4.7 Memcache

Hemos demostrado que web2py proporciona dos tipos de caché: cache.ram y cache.disk. Ambos trabajan en un entorno distribuido con varios servidores simultáneamente, pero no funcionan como se espera. En particular, cache.ram sólo caché a nivel del servidor, por lo que se convierte en inútil. cache.disk también caché a nivel de servidor a menos que la carpeta "cache" es una carpeta compartida que soporta bloqueo, por lo que, en lugar de acelerar las cosas, se convierte en un importante cuello de botella.

La solución no va a usarlos, pero en lugar de usar memcache. web2py viene con un memcache API.

Para usar memcache, cree un archivo nuevo modelo, por ejemplo, 0_memcache.py, y en este archivo escribir (o añadir) el siguiente código:

```
1 from gluon.contrib.memcache import MemcacheClient
2 memcache_servers = ['127.0.0.1:11211']
3 cache.memcache = MemcacheClient(request, memcache_servers)
4 cache.ram = cache.disk = cache.memcache
```

La primera línea memcache las importaciones. La segunda línea tiene que ser una lista de sockets memcache (servidor: puerto). La tercera línea cache.ram y cache.disk redefine en términos de memcache.

Usted puede optar por redefinir sólo uno de ellos para definir un nuevo objeto de caché totalmente apuntando al objeto Memcache.

Con este truco la carpeta "cache" no tiene por qué ser una carpeta compartida

más, ya que dejará de ser visitada.

Este código requiere que se ejecuta en servidores memcache la red local. Usted debe consultar la documentación memcache para obtener información sobre cómo configurar estos servidores.

11.4.8 Sesiones en Memcache

Si lo hace sesiones de necesidad y usted no desea utilizar un equilibrador de carga con sesiones pegajosa, tiene la opción de almacenar las sesiones en memcache:

```
1 from gluon.contrib.memdb import MEMDB
2 session.connect(request,response,db=MEMDB(cache.memcache))
```

11.4.9 Eliminación de aplicaciones

En un entorno de producción, puede ser mejor no instalar las aplicaciones por defecto: **admin**, **examples** y **welcome**. Aunque estas aplicaciones son muy pequeñas, no son necesarias.

La eliminación de estas aplicaciones es tan fácil como borrar las carpetas correspondientes en la carpeta de aplicaciones.

11.4.10 Uso de bases de datos replicadas Servidores

En un entorno de alto rendimiento que puede tener una base de datos esclava de la arquitectura principal con muchos esclavos y quizás replicar un par de servidores replicados. El DAL puede manejar esta situación y de forma condicionada conectarse a distintos servidores en función de los parámetros

de petición. El API para hacer esto fue descrito en el capítulo 6. Aquí está un ejemplo:

```
1 from random import shuffle
2 db=DAL(shuffle(['mysql://...1','mysql://...2','mysql://...3']))
```

En este caso diferentes peticiones HTTP serán atendidos por diferentes bases de datos al azar y cada uno de DB se verán afectadas más o menos con la misma probabilidad.

También podemos aplicar un simple Round-Robin

```
1 def fail_safe_round_robin(*uris):
2     i = cache.ram('round-robin', lambda: 0, None)
3     uris = uris[i:]+uris[:i] # rotate the list of uris
4     cache.ram('round-robin', lambda: (i+1)%len(uris), 0)
5     return uris
6 db = DAL(fail_safe_round_robin('mysql://...1','mysql://...2','mysql://...3'))
```

Esto es prueba de fallos en el sentido de que si el servidor de base de datos asignado a la solicitud no se puede conectar, DAL tratará el siguiente en el orden.

Si también es posible conectarse a diferentes bases de datos en función de la acción solicitada o controlador. En una configuración de base de datos master-esclavo, una acción sólo realiza una persona y leer algunas de lectura / escritura. El formato de seguridad puede conectarse a un servidor db de esclavos, mientras que la segunda debe conectarse a un maestro. Así que usted puede hacer:

```
1 if request.action in read_only_actions:
2     db=DAL(shuffle(['mysql://...1','mysql://...2','mysql://...3']))
3 else:
4     db=DAL(shuffle(['mysql://...3','mysql://...4','mysql://...5']))
```

1,2,3 donde se replican en los servidores y 3,4,5 son esclavos.

11.5 Google App Engine

Es posible ejecutar código web2py en Google App Engine (GAE) (13), incluyendo el código de DAL, con algunas limitaciones. La plataforma de juego ofrece varias ventajas sobre las soluciones de hosting normal:

- Facilidad de implementación. Google resúmenes completamente la arquitectura subyacente.
- Escalabilidad. Google va a replicar su aplicación tantas veces como sea necesario para atender todas las solicitudes concurrentes
- BigTable. El juego, en lugar de una base de datos relacional normal, se almacena la información persistente en BigTable, el almacén de datos de Google es famoso.

Las limitaciones son las siguientes:

- No tienes ni leer ni escribir el acceso al sistema de archivos.
- Las operaciones a plazo
- No es posible realizar consultas complejas en el almacén de datos, en particular, no existen JOIN, LIKE, IN, y la fecha y los operadores DATETIME.
- Usted puede utilizar la instrucción OR entre sub-consultas que implican una misma superficie.

Esto significa que no puede almacenar web2py sesiones, entradas de error, los archivos caché y los archivos cargados en el sistema de archivos, que estén almacenados en el almacén de datos y no en el sistema de archivos.

Aquí proporcionamos una breve descripción del juego y nos centramos en web2py cuestiones específicas, lo remitimos a la documentación en línea del Juego oficial para más detalles. — Atención: Al momento de escribir este juego es compatible con sólo Python 2.5. Cualquier otra versión causará

problemas. También debe ejecutar la distribución web2py fuente, no una distribución binaria. —

Vamos a proceder en orden.

11.5.1 Configuración

Hay dos archivos de configuración para tener en cuenta:

```
1 web2py/app.yaml
2 web2py/index.yaml
```

"app.yaml"" tiene la siguiente estructura (se ha reducido el uso...):

```
1 application: web2py
2 version: 1
3 api_version: 1
4 runtime: python
5 handlers:
6 - url: /_ha/stats.*
7   ...
8 - url: /(?P<a>.+?)/static/(?P<b>.+)
9   ...
10 - url: /_ha/admin/.*
11   ...
12 - url: /_ah/queue/default
13   ...
14 - url: .*
15   ...
16 skip_files:
17 ...
```

Debe reemplazar web2py con la solicitud de identificación que utilizó al registrarse en Google App Engine.

`url: /_ha/stats.*` instruye JUEGO para exponer la dirección "/stats" con las estadísticas y perfiles de información.

`url: /(.+?)/static/(.+)` instruye GAME para servir a sus archivos estáticos aplicación directa, sin llamar la lógica web2py, para la velocidad.

`url: /_ha/admin/.*` instruye JUEGO para exponer la interfaz administrativa JUEGO.

`url: /_ah/queue/default` instruye JUEGO para exponer una interfaz a la cola de tareas. Esto es usado por web2py el fin de mantener a cargar de nuevo las páginas de forma automática y mantenerlos en caché para agilizar el servicio.

`url: .*` instruye web2py utilizar el `gaehandler.py` para todas las peticiones de otros.

La sesión `skip_files`: está la lista de expresiones regulares para los archivos que no necesita desplegar en GAME. En particular, la línea:

```
1 ((admin|examples|welcome)\.tar) |
```

JUEGO dice no desplegar las aplicaciones por defecto. Usted puede agregar aquí más solicitudes para ser ignorado.

A excepción de la aplicación que id probablemente no necesita editar `app.yaml`.

El `index.yaml` de archivo se genera automáticamente al ejecutar la aplicación localmente utilizando el appserver GAME (el servidor web que viene con el SDK de GOOGLE). Contiene algo como esto:

```
1 indexes:
2 - kind: person
3   properties:
4     - name: name
5       direction: desc
```

En este ejemplo se le dice a juego para crear un índice para "person" that will be used to sort by "name" tabla en orden alfabético inverso. Usted no será capaz de buscar y ordenar registros en su aplicación sin índices correspondientes.

Es importante ejecutar siempre sus aplicaciones a nivel local con el appserver y probar todas las funcionalidades de su aplicación, antes del despliegue. Esto será importante para el ensayo propuesto, sino también para generar automáticamente el archivo de "index.yaml". En ocasiones es posible que desee editar este archivo y realizar la limpieza, tales como la eliminación de la duplicación de entradas.

11.5.2 Ejecución e implementación

Aquí se supone que ha instalado el SDK de juego. Usted puede ejecutar la aplicación desde dentro de la carpeta "web2py" con el comando appserver:

```
1 python2.5 dev_appserver ../web2py
```

Esto iniciará el appserver y se puede ejecutar la aplicación en la URL:

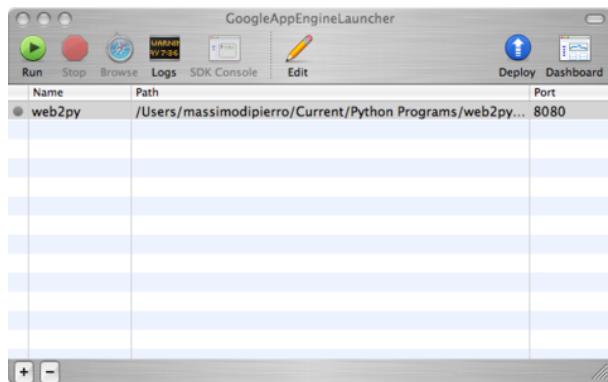
```
1 http://127.0.0.1:8080/
```

Con el fin de subir su aplicación en el juego, asegúrese de editar el fichero de "app.yaml" como se ha explicado antes y configurar el ID de la correcta aplicación, a continuación, ejecute:

```
1 python2.5 appcfg.py update ../web2py
```

En Mac y viudas también se puede usar el Google App Engine Launcher. Puede descargar el software desde ref. (13).

Seleccione [File] [Aregar existentes solicitud], establezca la ruta a la ruta de la carpeta web2py de nivel superior, y presione el botón [Ejecutar] botón en la barra de herramientas. Después de haber probado que funciona a nivel local, se puede implementar en el juego con un simple clic en el botón [Desplegar] en la barra de herramientas (asumiendo que usted tiene una cuenta).



En GAME, las entradas web2py / errores también se registran en la administración de consola de juegos donde los troncos se puede acceder y buscar en línea.

Date	Time	Message
07-05	12:13PM	54.970 /bookmark/151 303 1639ms 0 kb ...
07-05	12:13PM	55.662 unable to import wsgiserver
07-05	12:13PM	48.709 /redislib/static/design/aside.gif 400 448ms 0
07-05	12:13PM	48.920 unable to import wsgiserver
07-05	12:13PM	48.355 /favicon.ico 400 444ms 0 kb ...
07-05	12:13PM	48.558 unable to import wsgiserver
07-05	11:04AM	58.133 /index/technology 200 1766ms 8 kb ...
07-05	11:04AM	58.833 unable to import wsgiserver
07-05	11:04AM	39.383 /favicon.ico 400 1042ms 0 kb ...

11.5.3 Configuración del controlador

El gaehandler.py fichero es responsabilidad de servir archivos de juego y tiene unas cuantas opciones. He aquí sus valores por defecto:

```

1 KEEP_CACHED = False
2 LOG_STATS = False
3 APPSTATS = True
4 DEBUG = False
5 AUTO_RETRY = True

```

`KEEP_CACHED` pide JUEGO para almacenar las tareas en la cola de tareas que seguir llamando a una acción ficticia lo que obliga a GAME para el caché de su aplicación. Esto hará que JUEGO servir a sus usuarios más rápido pero consume algo de ciclos adicionales.

`LOG_STATS` registrará el tiempo para servir páginas en el acceso al juego.

`APPSTATS` permitirá appstats juego que proporciona estadísticas de perfiles. Estarán disponibles en la URL:

```
1 http://localhost:8080/_ah/stats
```

`DEBUG` establece el modo de depuración. Se hacen ninguna diferencia en la práctica a menos marcada de forma explícita en el código a través de `gluon.settings.web2py_runtime`.

`AUTO_RETRY` instruye juego para intentar cometer de nuevo en caso de avería.

11.5.4 Evite el sistema de archivos

El JUEGO que no tienen acceso al sistema de archivos. No se puede abrir cualquier archivo para escribir.

A tal efecto, en el partido, web2py almacena automáticamente todos los archivos cargados en el almacén de datos, sea o no "upload" Campo (s) tiene un atributo `uploadfield`.

También debe guardar sesiones y entradas en la base de datos y tiene que ser explícito:

```

1 if request.env.web2py_runtime_gae
2     db = DAL('gae')
3     session.connect(request,response,db)
4 else:
5     db = DAL('sqlite://storage.sqlite')
```

El código anterior comprueba si se está ejecutando en el juego, se conecta a BigTable, y encarga a web2py para almacenar sesiones y los boletos en ese país. Se conecta a una base de datos SQLite de otra manera. Este código ya está en la aplicación andamio en el archivo de "db.py".

11.5.5 Memcache

Si lo prefieres, puedes guardar las sesiones en memcache:

```

1 from gluon.contrib.gae_memcache import MemcacheClient
2 from gluon.contrib.memdb import MEMDB
3 cache.memcache = MemcacheClient(request)
4 cache.ram = cache.disk = cache.memcache
5
6 db = DAL('gae')
7 session.connect(request,response,MEMDB(cache.memcache))
```

Tenga en cuenta que en cache.ram GAME y cache.disk no debe ser utilizado, por lo que hacen punto de cache.memcache.

11.5.6 Problemas de base de datos

La ausencia de transacciones multi-entidad y funcionalidades típicas de bases de datos relacionales son lo que diferencia a GAME, aparte de otro entorno de alojamiento. Este es el precio a pagar por una alta escalabilidad. GAME es una plataforma excelente si estas limitaciones son tolerables, si no, entonces una plataforma regular de acogida con una base de datos relacional y no debe ser considerada.

Si una aplicación web2py no se ejecuta en juego, es debido a una de las limitaciones discutidas anteriormente. La mayoría de problemas se pueden resolver mediante la eliminación de las consultas JOIN web2py y de la normalización de la base de datos.

Sin embargo, Google App Engine soporta a algunos tipos de campo que no lo hacen directamente en el mapa tipos DAL, por ejemplo `StringListProperty`. Puede seguir utilizando este tipo con web2py con la siguiente sintaxis.

```

1 from gluon.contrib.gql import gae
2 db.define_table('myitem',
3     Field('name'),
4     Field('keywords', type=gae.StringListProperty())

```

Aquí el campo de palabras clave es una `StringListProperty` tanto, sus valores deben ser listas de cadenas, de forma compatible con la documentación del juego.

11.5.7 JUEGO y HTTPS

Si tu solicitud ha id "myapp" es su dominio JUEGO

```

1 http://myapp.appspot.com/

```

y también se puede acceder a través de HTTPS

```

1 https://myapp.appspot.com/

```

En este caso se utilizará un certificado "appspot.com" proporcionada por Google.

Usted puede registrar una entrada de DNS y utilizar cualquier otro nombre de dominio que usted posee para su aplicación, pero en este caso, no será capaz de utilizar HTTPS. En el momento de escribir esto una limitación JUEGO.

12

Otras recetas

12.1 Actualizar

En el botón de "site" page of the administrative interface there is an "upgrade now". En caso de que esto no es posible o no funciona (por ejemplo debido a un problema de bloqueo de archivos), el mejoramiento web2py manualmente es muy fácil.

Simplemente descomprimir la última versión de web2py sobre la instalación de edad.

Esto actualizará todas las bibliotecas, así como las aplicaciones **admin**, **examples**, **welcome**. También creará una nueva "NEWINSTALL". Upon restarting, web2py will delete the empty file and package the welcome app into "welcome.w2p" archivo vacío que se utilizarán como andamios nueva aplicación. web2py no actualiza ningún archivo en otras aplicaciones existentes.

12.2 Cómo distribuir sus aplicaciones como Binarios

Es posible que envuelva a su aplicación con la distribución binaria web2py y distribuirlos juntos. La licencia permite que este, siempre que dejar claro en la licencia de su aplicación que está empaquetado con web2py y añadir un enlace a la web2py.com.

Aquí le explicamos cómo hacerlo para Windows:

- Crea tu aplicación como de costumbre
- Uso de **admin**, bytecode compilar su aplicación (un clic)
- Uso de **admin**, el paquete de su aplicación compilada (otro clic)
- Crear una carpeta de "myapp"
- Descargar una ventana web2py distribución binaria
- Descomprimir en carpeta "myapp" y ponerlo en marcha (dos clics)
- Cargar con la **admin** embalado y aplicación compilada con el nombre de "init" (un clic)
- Cree un archivo de "myapp/start.bat" that contains "cd web2py; web2py.exe"
- Cree un archivo de "myapp/license" that contains a license for your app and make sure it states that it is being "distributed with an unmodified copy of web2py from web2py.com"
- Zip la carpeta miaplicacion en un archivo de "myapp.zip"
- Distribuir y / o venta de "myapp.zip"

Cuando los usuarios descomprimirá aplicación "myapp.zip" and click "run" they will see your app instead of the "welcome". No existe un requisito por parte del usuario, ni siquiera pre-instalado Python.

Por binarios Mac el proceso es el mismo pero no es necesario para el archivo de "bat".

12.3 Obtención de una dirección URL externa

Python incluye la biblioteca de ir a buscar urls `urllib`:

```
1 import urllib
2 page = urllib.urlopen('http://www.web2py.com').read()
```

Esto es a menudo muy bien, pero el módulo `urllib` no funciona en el motor de Google App. Google ofrece una API diferente para la descarga de URL que funciona en juego solamente. Con el fin de hacer que su código portable, `web2py fetch` incluye una función que funciona en juego, así como otras instalaciones de Python:

```
1 from google.tools import fetch
2 page = fetch('http://www.web2py.com')
```

12.4 Bastante fechas

A menudo es útil para representar una fecha y hora no como "2009-07-25 14:34:56" but as "one year ago". `web2py` proporciona una función de utilidad para esto:

```
1 d = datetime.datetime(2009,7,25,14,34,56)
2 from google.tools import prettydate
3 pretty_d = prettydate(d,T)
```

El segundo argumento (T) debe ser promulgada para permitir la internacionalización de la producción.

12.5 Geocodificación

Si usted necesita para convertir una dirección (por ejemplo: "243 S Wabash Ave, Chicago, IL, USA") en coordenadas geográficas (latitud y longitud),

web2py proporciona una función para ello.

```

1 from gluon.tools import geocode
2 address = '243 S Wabash Ave, Chicago, IL, USA'
3 (latitude, longitude) = geocode(address)

```

El geocode función requiere una conexión de red y conectarse al servicio de geocodificación de Google para la geocodificación. La función devuelve $(0,0)$ en caso de fallo. Observe que el servicio de Google geocodificación tapas del número de solicitudes y debe revisar su contrato de servicio. La función geocode está construida encima de la función de fetch y por lo tanto funciona en GAME.

12.6 Paginación

Esta receta es un truco útil para minimizar el acceso de base de datos en el caso de la paginación, por ejemplo, cuando se necesita mostrar una lista de filas de una base de datos pero desea distribuir las filas durante varias páginas.

Comience por crear una aplicación que almacena **primes** los primeros 1000 números primos en una base de datos.

Aquí está el modelo db.py:

```

1 db=DAL('sqlite://primes.db')
2 db.define_table('prime',Field('value','integer'))
3 def isprime(p):
4     for i in range(2,p):
5         if p%i==0: return False
6     return True
7 if len(db().select(db.prime.id))==0:
8     p=2
9     for i in range(1000):
10        while not isprime(p): p+=1
11        db.prime.insert(value=p)
12        p+=1

```

Ahora crea una `list_items` acción en el controlador "default.py" que dice así:

```

1 def list_items():
2     if len(request.args): page=int(request.args[0])
3     else: page=0
4     items_per_page=20
5     limitby=(page*items_per_page,(page+1)*items_per_page+1)
6     rows=db().select(db.prime.ALL,limitby=limitby)
7     return dict(rows=rows,page=page,items_per_page=items_per_page)

```

Observe que este código selecciona un tema más de lo necesario, `20 +1`. La razón es que el elemento adicional cuenta la opinión de si hay una página siguiente.

Esta es la opinión "default/list_items.html":

```

1 {{extend 'layout.html'}}
2
3 {{for i,row in enumerate(rows):}}
4 {{if i==items_per_page: break}}
5 {{=row.value}}<br />
6 {{pass}}
7
8 {{if page:}}
9 <a href="{{=URL(args=[page-1])}}>previous</a>
10 {{pass}}
11
12 {{if len(rows)>items_per_page:}}
13 <a href="{{=URL(args=[page+1])}}>next</a>
14 {{pass}}

```

De esta manera hemos obtenido la paginación con una selección única por acción, y que uno seleccione sólo selecciona una fila más de lo que necesitamos.

12.7 httpserver.log y el formato de archivo de registro

El servidor web web2py registra los registros de todas las peticiones en un archivo llamado:

```
1 httpserver.log
```

en el directorio raíz web2py. Un nombre de archivo y la ubicación alternativa se puede especificar a través de opciones de línea de comandos-web2py.

Las nuevas entradas se anexan al final del archivo cada vez que se realiza una solicitud. Cada línea tiene el siguiente aspecto:

```
1 127.0.0.1, 2008-01-12 10:41:20, GET, /admin/default/site, HTTP/1.1, 200, 0.270000
```

El formato es:

```
1 ip, timestamp, method, path, protocol, status, time_taken
```

Donde

- ip es la dirección IP del cliente que hizo la solicitud
- timestamp es la fecha y hora de la solicitud en formato ISO 8601, AAAA-MM-DDT HH: MM: SS
- método es bien GET o POST
- ruta es la ruta solicitada por el cliente
- protocolo es el protocolo HTTP utilizado para enviar al cliente, por lo general HTTP/1.1
- estatus es el de los códigos de estado HTTP (92)
- time_taken es la cantidad de tiempo que el servidor se procese la solicitud, en segundos, sin incluir la carga / tiempo de descarga.

En el (34) repositorio de aplicaciones, usted encontrará un aparato de análisis de registros.

Este registro está deshabilitado de forma predeterminada cuando se utiliza mod_wsgi ya que sería el mismo que el registro de Apache.

12.8 Poblar la base de datos con un maniquí de datos

Para propósitos de prueba es conveniente para poder llenar las tablas de base de datos con datos ficticios. web2py incluye un clasificador bayesiano ya formados para generar texto de relleno pero de fácil lectura para este fin.

Esta es la forma más sencilla de usarlo:

```
1 from gluon.contrib.populate import populate
2 populate(db.mytable,100)
```

Se le insertará 100 registros ficticios en db.mytable. Se trata de hacer de forma inteligente mediante la generación de texto breve para los campos de cadena, ya de texto para los campos de texto, números enteros, dobles, fechas, datetimes, los tiempos, booleanos, etc de los campos correspondientes. Se tratará de respetar las condiciones impuestas por los validadores. Para los campos que contienen la palabra "name" que tratará de generar nombres ficticios. Para los campos de referencia que vaya a generar referencias válidas.

Si dispone de dos tablas (A y B), donde las referencias a, b, asegúrese de llenar una primera y segunda B.

Puesto que la población se lleva a cabo en una transacción, no trate de llenar demasiados registros a la vez, en particular, si las referencias están involucrados. En su lugar, llenar 100 en el momento, completar bucle.

```
1 for i in range(10):
2     populate(db.mytable,100)
3     db.commit()
```

Usted puede utilizar el clasificador bayesiano para aprender un poco de texto y generar texto de relleno que suena similar pero no debe tener sentido:

```
1 from gluon.contrib.populate import Learner, IUP
2 ell=Learner()
3 ell.learn('some very long input text ...')
4 print ell.generate(1000,prefix=None)
```

12.9 Enviar un SMS

Enviar mensajes SMS desde una aplicación web2py requiere un servicio de terceros que puede reenviar los mensajes al receptor. Por lo general esto no es un servicio gratuito, pero difiere de un país a otro. Hemos tratado algunos de estos servicios con poco éxito debido a impedir que las empresas telefónicas correos electrónicos procedentes de estos servicios, ya que finalmente se utiliza como una fuente de spam.

Una mejor manera es utilizar las empresas de telefonía mismos que depender del SMS. Cada compañía telefónica tiene una dirección de correo electrónico asociado únicamente a cada número de teléfono celular el SMS se pueden enviar como un correo electrónico a ese número de teléfono. web2py viene con un módulo para ayudar en este proceso:

```
1 from gluon.contrib.sms_utils import SMSCODES, sms_email
2 email = sms_email('1 (111) 111-1111','T-Mobile USA (tmail)')
3 mail.send(to=email,subject='test',message='test')
```

SMSCODES es un diccionario que mapea los nombres de las compañías telefónicas importantes en el postfix dirección de correo electrónico. La función sms_email tiene un número de teléfono (en forma de cadena) y el nombre de una compañía telefónica y devuelve la dirección de correo electrónico del teléfono.

12.10 A aceptar pagos con tarjeta de crédito

web2py existen múltiples formas para su aplicación a aceptar pagos con tarjeta de crédito. Por ejemplo:

Google Checkout Plugin

```
1 http://web2py.com/plugins/static/web2py.plugin.google_checkout.w2p
```

PayPal

```
1 http://www.web2pyslices.com/main/slices/take_slice/9
```

Authorize.Net

Los primeros dos mecanismos anteriores delegar el proceso de autenticación del beneficiario a un servicio externo. Si bien esta es la mejor solución para la seguridad (su aplicación no controla en absoluto ninguna información de tarjeta de crédito) que hace el proceso engoroso (el usuario debe ingresar dos veces, con su aplicación, y en Google por ejemplo) y no permite su aplicación a gestionar los pagos recurrentes de forma automática.

Hay momentos en que necesita más control. Por esta razón, una integración de la caja con la API de Authorize.Net (el módulo fue desarrollado por John Conde y modificada ligeramente). He aquí un ejemplo de flujo de trabajo y todas las variables que están expuestos:

```
1 from gluon.contrib.AuthorizeNet import AIM
2 payment = AIM(login='cnpdev4289',
3                 transkey='SR2P8g4jdEn7vFLQ',
4                 testmod=True)
5 payment.setTransaction(creditcard, expiration, total, cvv, tax, invoice)
6 payment.setParameter('x_duplicate_window', 180) # three minutes duplicate windows
7 payment.setParameter('x_cust_id', '1324')      # customer ID
8 payment.setParameter('x_first_name', 'Agent')
9 payment.setParameter('x_last_name', 'Smith')
10 payment.setParameter('x_company', 'Test Company')
11 payment.setParameter('x_address', '1234 Main Street')
12 payment.setParameter('x_city', 'Townsville')
13 payment.setParameter('x_state', 'NJ')
14 payment.setParameter('x_zip', '12345')
15 payment.setParameter('x_country', 'US')
16 payment.setParameter('x_phone', '800-555-1234')
17 payment.setParameter('x_description', 'Test Transaction')
18 payment.setParameter('x_customer_ip', socket.gethostbyname(socket.gethostname()))
19 payment.setParameter('x_email', 'you@example.com')
20 payment.setParameter('x_email_customer', False)
21
22 payment.process()
23 if payment.isApproved():
24     print 'Response Code: ', payment.response.ResponseCode
25     print 'Response Text: ', payment.response.ResponseText
26     print 'Response: ', payment.getResultResponseFull()
```

```

27     print 'Transaction ID: ', payment.response.TransactionID
28     print 'CVV Result: ', payment.response.CVVResponse
29     print 'Approval Code: ', payment.response.AuthCode
30     print 'AVS Result: ', payment.response.AVSResponse
31 elif payment.isDeclined():
32     print 'Your credit card was declined by your bank'
33 elif payment.isError():
34     print 'It did not work'
35 print 'approved',payment.isApproved()
36 print 'declined',payment.isDeclined()
37 print 'error',payment.isError()

```

Aviso sobre el código utiliza una cuenta de prueba simulada. Es necesario registrarse con Authorize.Net (no es un servicio gratuito), y proporcionar su propia entrada, transkey, testmode = Verdadero o Falso al constructor AIM.

12.11 Twitter API

Aquí hay algunos ejemplos rápidos de como poner / obtener tweets. No hay bibliotecas de terceros están obligados, ya que Twitter API REST utiliza simples.

He aquí un ejemplo de cómo publicar un tweet:

```

1 def post_tweet(username,password,message):
2     import urllib, urllib2, base64
3     import gluon.contrib.simplejson as sj
4     args= urllib.urlencode([('status',message)])
5     headers={}
6     headers['Authorization'] = 'Basic ' +base64.b64encode(username+':'+password)
7     request = urllib2.Request('http://twitter.com/statuses/update.json', args,
8                               headers)
9     return sj.loads(urllib2.urlopen(req).read())

```

He aquí un ejemplo de cómo recibir tweets:

```

1 def get_tweets():
2     user='web2py'
3     import urllib
4     import gluon.contrib.simplejson as sj

```

```

5 page = urllib.urlopen('http://twitter.com/%s?format=json' % user).read()
6 tweets=XML(sj.loads(page)['#timeline'])
7 return dict(tweets=tweets)

```

Para operaciones más complejas, consulte la documentación de la API de Twitter.

12.12 Streaming de archivos virtual

Es común que los atacantes maliciosos para escanear páginas web en busca de vulnerabilidades. Ellos utilizan escáneres de seguridad como Nessus para explorar los sitios web de destino para los scripts que se sabe que tienen vulnerabilidades. Un análisis de los registros del servidor web de una máquina escaneada o directamente de la base de datos de Nessus revela que la mayoría de las vulnerabilidades conocidas en las secuencias de comandos y scripts PHP ASP. Ya que estamos ejecutando web2py, no tenemos esas vulnerabilidades, pero aún se analizan en busca de ellos. Este molesto, por lo que nos gusta como para responder a los escaneos de vulnerabilidades y hacer que el atacante entender su tiempo está siendo desperdiciado.

Una posibilidad es redirigir todas las solicitudes de. Php,. Asp, y cualquier cosa sospechosa a una acción ficticia que responder al ataque, manteniendo el atacante ocupado por una gran cantidad de tiempo. Finalmente, el atacante se rendirá y no nos vuelva a explorar.

Esta receta requiere de dos partes.

Una aplicación dedicada llamada **jammer** "default.py" con un controlador de la siguiente manera:

```

1 class Jammer():
2     def read(self,n): return 'x'*n
3     def jam(): return response.stream(Jammer(),40000)

```

Cuando esta acción se le llama, responde con una infinita secuencia de datos completa de "x"-es. 40000 personajes a la vez.

El segundo ingrediente es un archivo de "route.py" que redirige cualquier petición que terminan en. Php,. Asp, etc (tanto mayúsculas y minúsculas) de este controlador.

```
1 route_in=(  
2   ('.*\.(php|PHP|asp|ASP|jsp|JSP)', 'jammer/default/jam'),  
3 )
```

La primera vez que son atacados puede ocurrir en una pequeña sobrecarga, pero nuestra experiencia es que el mismo atacante no intentará dos veces.

12.13 Jython

web2py normalmente funciona en CPython (el intérprete de Python codificado en C), pero también se puede ejecutar en Jython (el intérprete de Python codificado en Java). Esto permite web2py para ejecutar en una infraestructura de Java.

A pesar de que las carreras web2py con Jython de la caja, hay algo de engaño conlleva la creación de Jython y en la creación de zxJDBC (el adaptador de base de datos de Jython). Aquí están las instrucciones:

- Descargue el archivo de "jython_installer-2.5.0.jar" (o 2.5.x) de Jython.org
- Instálelo:

```
1 java -jar jython_installer-2.5.0.jar
```

- Descargue e instale "zxJDBC.jar" de (94)
- Descargar e instalar el fichero de la "sqlitejdbc-v056.jar" (95)
- Añadir zxJDBC y sqlitejdbc al CLASSPATH java
- web2py Comience con Jython

```
1 /path/to/jython web2py.py
```

En el momento de la escritura sólo se da soporte `sqlite` y `postgres` en Jython.

13

Componentes y Plugins

Los componentes y los plugins son características relativamente nuevo de web2py y hay cierto desacuerdo entre los desarrolladores acerca de lo que son y lo que debería ser. La mayor parte de la confusión se origina en la diferente utilización de estos términos en otros proyectos de software y del hecho de que los desarrolladores siguen trabajando en la finalización de las especificaciones.

Sin embargo, son una característica importante y tengo que dar definiciones. Estos definición no pretende ser definitiva, sino sólo en consonancia con las pautas de programación que queremos discutir en este capítulo.

Hay dos cuestiones que tratamos de dirección aquí:

- ¿Cómo podemos construir aplicaciones modulares que reducen al mínimo la carga del servidor y maximizar la reutilización de código?
- ¿Cómo podemos distribuir piezas de código de una manera más o menos plugin-and-play?

dirección *components* la primera edición, dirección *plugins* el segundo.

13.1 Componentes

A **component** es una parte autónoma funcionalmente de una página web.

Un componente puede estar compuesto de los módulos, controladores y puntos de vista pero no hay ningún requisito estricto que no se incrusta en una página web, deberán estar localizados en una etiqueta html (por ejemplo, un DIV, un SPAN o un IFRAME) y Realiza su tarea de forma independiente del resto de la página. Estamos especialmente interesados en los componentes que se cargan en la página y comunicarse con la función de controlador de componentes a través de Ajax.

Un ejemplo de un componente es un "comments component" que figura en un DIV y muestra los comentarios de los usuarios y un formulario de envío-new-comentario. Cuando se envía el formulario, se envía al servidor a través de Ajax, la lista se actualiza, el comentario es almacenado en el servidor de la base de datos. El contenido se actualiza DIV sin recargar el resto de la página.

El web2py función de carga hace que este muy fácil de hacer explícita sin JavaScript / Ajax o el conocimiento de programación.

Nuestro objetivo es ser capaces de desarrollar aplicaciones web mediante el ensamblaje de componentes en los diseños de página.

Considere una aplicación web2py "test" that extends the default scaffolding app with a custom model in file "models/db_comments.py" simple:

```

1 db.define_table('comment',
2     Field('body','text',label='Your comment'),
3     Field('posted_on','datetime',default=request.now),
4     Field('posted_by',db.auth_user,default=auth.user_id))
5 db.comment.posted_on.writable=db.comment.posted_on.readable=False
6 db.comment.posted_by.writable=db.comment.posted_by.readable=False

```

una acción en "controllers/comments.py"

```

1 @auth.requires_login()
2 def post():
3     return dict(form=crud.create(db.comment),
4                 comments=db(db.comment.id>0).select())

```

y la correspondiente "views/comments/post.html"

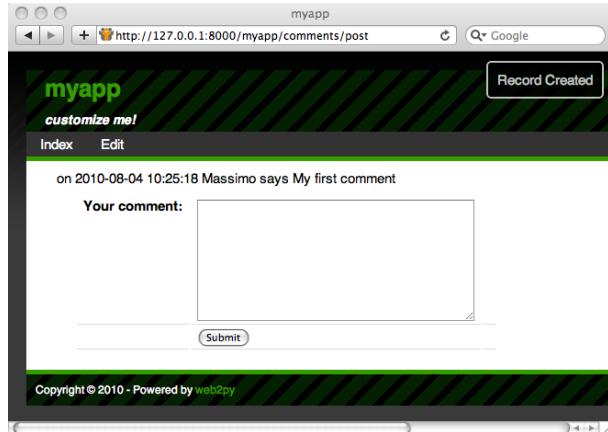
```

1 {{extend 'layout.html'}}
2 {{for comment in comments:}}
3 <div class="comment">
4     on {{=comment.posted_on}} {{=comment.posted_by.first_name}}
5     says <span class="comment_body">{{=comment.body}}</span>
6 </div>
7 {{pass}}
8 {{=form}}

```

Puedes acceder a él, como de costumbre en:

```
1 http://127.0.0.1:8000/test/comments/post
```



Hasta el momento no hay nada especial en esta acción, pero lo podemos convertir en un componente simplemente definiendo un nuevo punto de vista con ".load" extensión que no se extiende a la disposición.

Por lo tanto creamos un "views/comments/post.load":

```

1 {{#extend 'layout.html' <- notice this is commented out!}}
2 {{for comment in comments:}}
3 <div class="comment">
4   on {{=comment.posted_on}} {{=comment.posted_by.first_name}}
5   says <span class="comment_body">{{=comment.body}}</span>
6 </div>
7 {{/pass}}
8 {{=form}}

```

Podemos acceder a él en la dirección

```
1 http://127.0.0.1:8000/test/comments/post.load
```

y tendrá el siguiente aspecto:



Se trata de un componente y se puede incrustar en cualquier otra página simplemente haciendo

```
1 {{=LOAD('comments','post.load',ajax=True)}}
```

Por ejemplo, en "controllers/default.py" podemos editar

```

1 def index():
2     return dict()

```

y en la vista correspondiente agregar el componente:

```

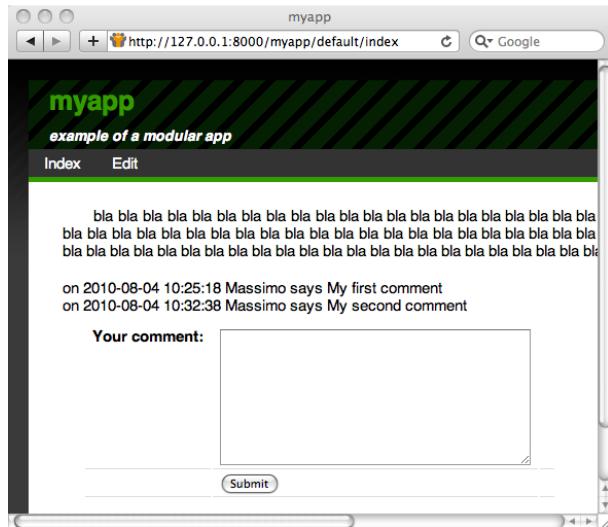
1 {{extend 'layout.html'}}
2 <p>{{='bla '*100}}</p>
3 {{=LOAD('comments','post.load',ajax=True)}}

```

Visitar la página de

```
1 http://127.0.0.1:8000/test/default/index
```

mostrará el contenido normal y el componente de observaciones:



El componente de `{{=LOAD(...)}}` se presenta como sigue html:

```
1 <script type="text/javascript"><!--
2 web2py_component("/test/comment/post.load","c282718984176")
3 <!--&gt;&lt;/script&gt;&lt;div id="c282718984176"&gt;loading...&lt;/div&gt;</pre>

```

(El código generado real depende de las opciones de pasar a la función CARGA).

La función se define en el `web2py_component(url,id)` "web2py_ajax.html" y realiza toda la magia: llama a la `url` por medio de Ajax e incrusta la respuesta en el DIV con sus correspondientes `id`; atrapa cada envío de formularios en el DIV y presenta las formas a través de Ajax. El objetivo es siempre el Ajax se DIV.

La firma definitiva del ayudante de carga es el siguiente:

```

1 LOAD(c=None, f='index', args=[], vars={},
2     extension=None, target=None,
3     ajax=False, ajax_trap=False,
4     url=None):

```

Aquí:

- los dos primeros argumentos `c` y `f` son responsable del tratamiento y la función que queremos llamar, respectivamente.
- `args` y `vars` son los argumentos y las variables que queremos pasar a la función. La primera es una lista, el segundo es un diccionario.
- `extension` es una ampliación opcional. Tenga en cuenta que la ampliación también se puede transmitir una parte de la función como en `f='index.load'`.
- `target id` es el ID de la meta DIV. Si no se especifica un objetivo al azar `id` se genera.
- `ajax` se debería establecer en `True` si el DIV tiene que ser llenado a través de Ajax y `False` para el DIV si hay que rellenar antes de la página actual es el retorno de ser (evitando así la llamada Ajax).
- `ajax_trap=True` significa que cualquier envío de formularios en el DIV debe ser capturado, presentado a través de Ajax, y la respuesta tiene que ser devuelta dentro del DIV. `ajax_trap=False` indica que los formularios deben presentarse normalmente por lo tanto volver a cargar la página entera. `ajax_trap` se ignora y supone que si `True` `ajax=True`.
- `url`, si se especifica, reemplaza los valores de la `c`, `f`, `args`, `vars`, `extension` y las cargas y el componente en la `url`. Se utiliza para cargar las páginas servidas por componentes de otras aplicaciones (que mi o no puede ser creada con web2py).

Si no se especifica vista `.load`, `generic.load` hay un diccionario que hace devuelto por la acción sin diseño. Funciona mejor si el diccionario contiene un único elemento.

Si carga un componente que tiene la extensión `.load` y la función de controlador correspondiente redirecciona a otra acción (por ejemplo, un formulario

de acceso), se propaga la extensión `.load` y la nueva dirección (la de redirigir también) también está cargado con una extensión `.load`.

Cuando la acción de un componente se le llama a través de Ajax, web2py pasa dos cabeceras HTTP con la petición:

```
1 web2py-component-location
2 web2py-component-element
```

que se puede acceder por la acción a través de las variables:

```
1 request.env.http_web2py_component_location
2 request.env.http_web2py_component_element
```

El primero contiene la dirección URL de la página que llamó a la acción de los componentes. Este último contiene la `id` del DIV que contiene la respuesta.

La acción componente también puede almacenar datos en dos cabeceras especiales respuesta HTTP que será interpretado por la página completa de la respuesta. Ellos son:

```
1 web2py-component-flash
2 web2py-component-command
```

y que se pueden establecer a través de:

```
1 response.headers['web2py-component-flash']='...'
2 response.headers['web2py-component-command']='...'
```

El primero contiene el texto que desea ser flasheada en la respuesta. Este último contiene el código JS que desea ser ejecutado en la respuesta. No puede contener saltos de línea.

Como ejemplo vamos a definir un componente de formulario de contacto en el mensaje de "controllers/contact/ask.py" that allows the user to ask a question and the component will emails the question to the system administrator, flash a "thank you" y eliminar el componente de la página:

```
1 def ask():
```

```

2     form=SQLFORM.factory(
3         Field('your_email',requires=IS_EMAIL()),
4         Field('question',requires=IS_NOT_EMPTY()))
5     if form.accepts(request.vars,session):
6         if mail.send(to='admin@example.com',
7             subject='from %s' % form.vars.your_email,
8             message = form.vars.question):
9             div_id = request.env.http_web2py_component_element
10            command="jQuery('#%s').hide()" % div_id
11            response.headers['web2py-component-command']=command
12            response.headers['web2py-component-flash']='thanks you'
13        else:
14            form.errors.your_email="Unable to send the email"
15    return dict(form=form)

```

Las primeras cuatro líneas definen la forma y aceptarlo. El correo de objetos utilizados para el envío se define por defecto en el andamiaje de la aplicación. Las últimas cuatro líneas de aplicar toda la lógica específica de los componentes por la obtención de datos de las cabeceras de petición HTTP y el establecimiento de los encabezados de respuesta HTTP.

Ahora usted puede integrar este formulario de contacto en cualquier página a través de

```
1 {{=LOAD('contact','ask.load',ajax=True)}}
```

Observe que no se ha fijado una vista para nuestro componente .load ask. No tenemos porque devuelve a un solo objeto (forma) y por tanto la "generic.load" no tendrán ningún problema.

Si la vista se utiliza "generic.load" puede utilizar la sintaxis regular:

```
1 response.flash='...'
```

y es equivalente a

```
1 response.headers['web2py-component-flash']='...'
```

13.2 Plugins

El plugin es un subconjunto de los archivos de una aplicación.

y que realmente queremos decir *any*:

- Un plugin no es un módulo, no un modelo, no es un controlador, no es un punto de vista, sin embargo, puede contener módulos, modelos, controladores y / o puntos de vista.
- Un plugin no tiene que ser funcionalmente autónomo y que puede depender de otros plugins o código de usuario específico.
- Una *plugin* no es una *plugins system* por lo tanto no tiene ningún concepto de registro ni de aislamiento, aunque vamos a dar reglas para tratar de lograr cierto aislamiento.
- Aquí estamos hablando de un plugin para su aplicación, no un plugin para web2py.

¿Por qué se llama un *plugin*? Debido a que proporciona un mecanismo para el embalaje de un subconjunto de una aplicación y desempaquetarlo en otra aplicación (es decir, *plug-in*). Bajo esta definición cualquier archivo en su aplicación puede ser tratado como plugin.

Cuando la aplicación de su distribuyó sus plugins son envasados y distribuidos de la misma.

En la práctica el **admin** proporciona una interfaz para el embalaje y desembalaje plugins por separado de su aplicación. Archivos y carpetas de la aplicación que tienen nombres con la `plugin_name` prefijo puede agruparse en un archivo llamado:

`web2py.plugin.name.w2p`

Y distribuidos juntos.

Modules
additional code for your application

- [__init__.py \[edit\]](#)
- create file with filename: [submit](#)
- upload file: [Choose File](#) no file selected and rename it: [submit](#)

Plugins
To create a plugin, name a file/folder plugin_[name]

- [comments](#)
- upload plugin file: [Choose File](#) no file selected [submit](#)

Powered by web2py (TM) created by Massimo Di Pierro © 2007

Los archivos que componen un plugin no son tratados por web2py diferente que otros archivos excepto que entiende **admin** de sus nombres que están destinados a ser distribuidos juntos y los muestra en una hoja aparte:

Plugin "comments" in application "myapp"

[back | delete plugin | pack plugin] [models | controllers | views | static | modules]

Models
the data representation, define database tables and sets

[database administration | sql.log]

- [plugin_comments.py \[edit | delete\]](#) defines tables comment

Controllers
the application logic, each URL path is mapped in one exposed function in the controller

[shell | test | crontab]

- [plugin_comments.py \[edit | delete | test\]](#) exposes post

Views
the presentation layer, views are also known as templates

- [plugin_comments/post.html \[edit | htmledit | delete\]](#) extends layout.html
- [plugin_comments/post.load \[edit | htmledit | delete\]](#)
- create file with filename: [submit](#)

Static files
these files are served without processing, your images go here

There are no static files

- create file with filename: [submit](#)
- upload file: [Choose File](#) no file selected and rename it: [submit](#)

Powered by web2py (TM) created by Massimo Di Pierro © 2007

Sin embargo, como cuestión de hecho, según la definición anterior, estos plugins son más generales que los reconocidos como tales por **admin**.

En la práctica, sólo se refiere a dos tipos de plugins:

- *Component Plugins.* Estos son los plugins que contienen componentes tal como se define en la sección anterior. Un plugin componente puede tener uno o más componentes. Podemos pensar, por ejemplo de una `plugin_comments` que contiene el componente `comments` propuesto anteriormente. Otro ejemplo podría ser un `plugin_tagging` que contiene un componente `tagging` y `tag-cloud` un componente que comparten algunas tablas de bases de datos también se define por el plugin.
- *Layout Plugins.* Estos son los plugins que contiene una vista de diseño y los archivos estáticos requeridos por dicha disposición. Cuando se aplica el plugin le da la aplicación un nuevo look.

En las definiciones anteriores, los componentes creados en la sección anterior, por ejemplo, "controllers/contact.py", ya están plugins. Podemos pasar de una aplicación a otra y utilizar los componentes que definen. Sin embargo, no son reconocidos como tales por `admin` porque no hay nada de lo que les califica de plugins. Así que hay dos problemas que debemos resolver:

- Nombre del plugin de archivos mediante un convenio, de modo que `admin` puede reconocer como pertenecientes al mismo plugin
- En caso de que el plugin tiene archivos de modelos, establecer un convenio para que los objetos se define no contaminan el espacio de nombres y no están en conflicto unas con otras.

Supongamos que un plugin se llama `name`. Estas son las reglas que deben seguirse:

Plugin modelos: **Rule 1** y los controladores deben ser llamados, respectivamente

- `models/plugin_name.py`
- `controllers/plugin_name.py`

y puntos de vista de plugins, módulos, electricidad estática y los archivos privados deben estar en carpetas llamadas, respectivamente:

- `views/plugin_name /`
- `modules/plugin_name /`
- `static/plugin_name /`
- `private/plugin_name /`

Plugin modelos: **Rule 2** sólo puede definir los objetos con nombres que empiezan con

- `plugin_name`
- `PluginName`
- `_`

Plugin modelos: **Rule 2** sólo puede definir las variables de sesión con nombres que empiezan con

- `session.plugin_name`
- `session.PluginName`

Rule 4: Plugins debe incluir licencia y documentación. Estos deben ser colocados en:

- `static/plugin_name /license.html`
- `static/plugin_name /about.html`

Rule 5: El plugin sólo puede basarse en la existencia de los objetos globales definidas en andamios "db.py", es decir,

- Una conexión de base de datos llamada `db`

- una instancia llamada `Auth auth`
- una instancia llamada `Crud crud`
- una instancia llamada `Service service` Algunos plugins pueden ser más sofisticados y tienen un parámetro de configuración en caso de que más de una db ejemplo existe.

Rule 6: Si un plugin requiere de parámetros de configuración, estos deben definirse a través de un `PluginManager` como se describe a continuación.

Siguiendo las reglas anteriores podemos asegurarnos de que:

- `admin` reconoce todos los archivos y carpetas `plugin_name` como parte de una sola entidad.
- plugins no interfieran entre sí.

Las reglas anteriores no resuelven el problema de versiones de los plugins y dependencias. Eso está fuera de nuestro alcance.

13.2.1 Componente Plugins

plugins son los plugins de componentes que definen los componentes. Componentes de acceso por lo general la base de datos y definir sus propios modelos.

Giramos a la componente `comments` anterior en una `comments_plugin` con el mismo código que escribió antes, pero vamos a seguir todas las reglas anteriores.

En primer lugar, creamos un modelo denominado "models/plugin_comments.py":

```
1 db.define_table('plugin_comments_comment',
2     Field('body','text',label='Your comment'),
3     Field('posted_on','datetime',default=request.now),
```

```

4     Field('posted_by',db.auth_user,default=auth.user_id))
5 db.plugin_comments_comment.posted_on.writable=False
6 db.plugin_comments_comment.posted_on.readable=False
7 db.plugin_comments_comment.posted_by.writable=False
8 db.plugin_comments_comment.posted_by.readable=False
9
10 def plugin_comments():
11     return LOAD('plugin_comments','post',ajax=True)

```

(nótese las dos últimas líneas definir una función que va a simplificar la incorporación del plugin)

En segundo lugar, se define un "controllers/plugin_comments.py"

```

1 @auth.requires_login()
2 def post():
3     comments = db.plugin_comments_comment
4     return dict(form=crud.create(comment),
5                 comments=db(comment.id>0).select())

```

En tercer lugar, creamos una vista llamada "views/plugin_comments/post.load":

```

1 {{for comment in comments:}}
2 <div class="comment">
3     on {{=comment.posted_on}} {{=comment.posted_by.first_name}}
4     says <span class="comment_body">{{=comment.body}}</span>
5 </div>
6 {{pass}}
7 {{=form}}

```

Ahora podemos usar **admin** para empacar el plugin para su distribución. Admin salvará a este plugin como:

```
1 web2py.plugin.comments.w2p
```

Podemos utilizar el plug-in en cualquier punto de vista, simplemente instalando el plugin desde la página en **edit admin** y sumando esto a nuestros propios puntos de vista

```
1 {{=plugin_comments()}}
```

Por supuesto, podemos hacer que el plugin más sofisticados al tener componentes que toman parámetros y opciones de configuración. Cuanto más complejo de los componentes más difícil se hace para evitar conflictos de nombres. El Administrador de plug-in se describe a continuación está diseñado para evitar este problema.

13.2.2 Plugin Manager

El `PluginManager` es una clase definida en `gluon.tools`. Antes de explicar cómo funciona el interior vamos a explicar cómo usarlo.

Aquí consideramos que la `comments_plugin` anterior y lo hacemos mejor. Queremos ser capaces de personalizar:

```
1 db.plugin_comments_comment.body.label
```

sin tener que editar el código del plugin en sí.

Aquí es cómo podemos hacerlo:

En primer lugar, volver a escribir el "models/plugin_comments.py" plugin de esta manera:

```
1 db.define_table('plugin_comments_comment',
2     Field('body','text',label=plugin_comments.comments.body_label),
3     Field('posted_on','datetime',default=request.now),
4     Field('posted_by',db.auth_user,default=auth.user_id))
5
6 def plugin_comments():
7     from gluon.tools import PluginManager
8     plugins = PluginManager('comments',body_label='Your comment')
9
10    comment = db.plugin_comments_comment
11    comment.label=plugins.comments.body_label
12    comment.posted_on.writable=False
13    comment.posted_on.readable=False
14    comment.posted_by.writable=False
15    comment.posted_by.readable=False
16    return LOAD('plugin_comments','post',ajax=True)
```

Observe cómo todo el código, salvo la definición de tabla se resume en una sola función. Observe también cómo la función crea una instancia de una PluginManager.

Ahora, en cualquier otro modelo en su aplicación, por ejemplo en "models/db.py", puede configurar este plugin de la siguiente manera:

```
1 from gluon.tools import PluginManager
2 plugins = PluginManager()
3 plugins.comments.body_label=T('Post a comment')
```

El objeto plugins ya está en la instancia por defecto andamios aplicación en "models/db.py"

El objeto es un singleton PluginManager hilos a nivel de objeto de almacenamiento de objetos de almacenamiento. Eso significa que usted puede crear una instancia tantos como desee dentro de la misma aplicación pero (si tienen el mismo nombre o no) que actúan como si no hubiera una instancia PluginManager único.

En particular, el expediente de cada plugin puede hacer que su objeto PluginManager poseer y registrar mismo y con sus parámetros por defecto con él:

```
1 plugins = PluginManager('name', param1='value', param2='value')
```

Puede reemplazar estos parámetros en otro lugar (por ejemplo, en "models/db.py") con el código:

```
1 plugins = PluginManager()
2 plugins.name.param1 = 'other value'
```

Puede configurar varios plugins en un solo lugar.

```
1 plugins = PluginManager()
2 plugins.name.param1 = '...'
3 plugins.name.param2 = '...'
4 plugins.name1.param3 = '...'
```

```

5 plugins.name2.param4 = '...'
6 plugins.name3.param5 = '...'

```

Importe de que cuando el plugin se define el PluginManager debe tener argumentos: plugin y nombre opcional llamado las alegaciones de la que son los parámetros por defecto. En cambio, cuando se configuran los plugins, el constructor debe tener PluginManager sin argumentos. La configuración debe preceder a la definición de la extensión (es decir, debe estar en un archivo de modelo que es lo primero en orden alfabético).

13.2.3 Diseño de plugins

Diseño de plugins son más simples que los plugins de componentes porque por lo general no contienen código, sino sólo puntos de vista y los archivos estáticos. Sin embargo, hay algunas buenas prácticas que deben seguir:

Primero, cree una carpeta llamada `static/plugin_layout_name":italic /` (donde nombre es el nombre de su diseño) y colocar todos sus archivos estáticos en ese país.

En segundo lugar, crear un diseño de archivo llamado `views/plugin_layout_name:italic /layout.html` that contains your layout and links the images, CSS and JS files in "static/plugin_layout_name":italic /"

En tercer lugar, modificar la "views/layout.html" para que se lea, simplemente como:

```

1 {{include 'plugin_layout_name/layout.html'}}

```

La ventaja de este diseño es que los usuarios de este plugin se puede instalar múltiples diseños y elegir cuál aplicar simplemente "views/layout.html". Moreover "views/layout.html" edición no será embalado por el **admin**, junto

con el plugin de lo que no hay riesgo de que el plugin reemplaza el código del usuario en el anteriormente diseño instalado.

13.3 plugin_wiki

AVISO LEGAL: plugin_wiki es todavía muy bajo desarrollo y por lo tanto no nos comprometemos compatibilidad con versiones anteriores para el mismo nivel que a las funciones básicas web2py.

plugin_wiki es un plugin en esteroides. Lo que queremos decir es que define varios componentes útiles y puede cambiar la manera de desarrollar sus aplicaciones:

Puede descargarlo gratuitamente desde

: <http://web2py.com/examples/static/web2py.plugin.wiki.w2p>

La idea detrás de **plugin_wiki** es que la mayoría de aplicaciones incluyen páginas que son semi estática. Se trata de páginas que no incluyen la lógica personalizada complejo. Contienen texto estructurado (piense en una página de ayuda), imágenes, audio, vídeo, formularios de porquería, o un conjunto de componentes estándar (comentarios, etiquetas, gráficos, mapas), etc Estas páginas pueden ser públicas, requieren una entrada o tiene otros restricciones de autorización. Estas páginas pueden estar vinculados por un menú o sólo ser accesible a través del formulario asistente. **plugin_wiki** proporciona una forma fácil de añadir páginas que caben en esta categoría para las aplicaciones web2py regular.

En particular, **plugin_wiki** dispone:

- Una interfaz wiki, que permite el añadir páginas a su aplicación y haga referencia a ellas por una babosa. Estas páginas (que se hará referencia

a como las páginas wiki) tienen versiones y se almacenan en la base de datos.

- Pública y privada páginas (requiere login). Si una página requiere de entrada también puede requerir que el usuario tiene una pertenencia a un grupo en particular.
- Tres niveles: 1,2,3. En el nivel 1 sólo las páginas que pueden incluir texto, imágenes, audio y video. En el nivel 2 páginas también pueden incluir widgets (estos son los componentes tal como se define en la sección anterior, que puede ser embebido en páginas wiki). En el nivel 3 páginas también pueden incluir código de la plantilla web2py.
- Una opción de edición de páginas con la sintaxis markmin o en HTML mediante un editor WYSIWYG.
- Una colección de widgets: implementado como componentes que son independientes documentan y se puede integrar como componentes normales de visitas regulares en web2py o - utilizando una sintaxis simplificada - en las páginas wiki.
- Un conjunto de páginas especiales (`meta-code`, `meta-menu`, etc) que se pueden utilizar para personalizar el plugin (por ejemplo, definir el código del plugin debe funcionar, personalizar el menú, etc)

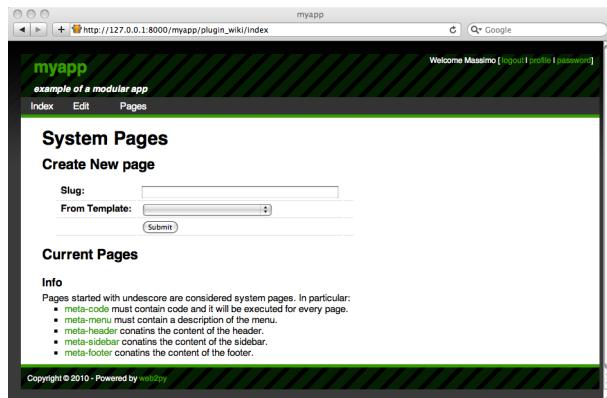
*La aplicación más el **welcome plugin_wiki** puede ser pensado como un entorno de desarrollo en sí mismo que es adecuado para la construcción de aplicaciones web sencillas, como un blog.*

A partir de ahora asumiremos que el **plugin_wiki** se aplica a una copia de la aplicación andamios **welcome**.

Lo primero que se nota después de instalar el plugin es que añade un nuevo elemento de menú llamado *pages*.

Haga clic en el elemento de menú *pages* y usted será redirigido a la acción **plugin**:

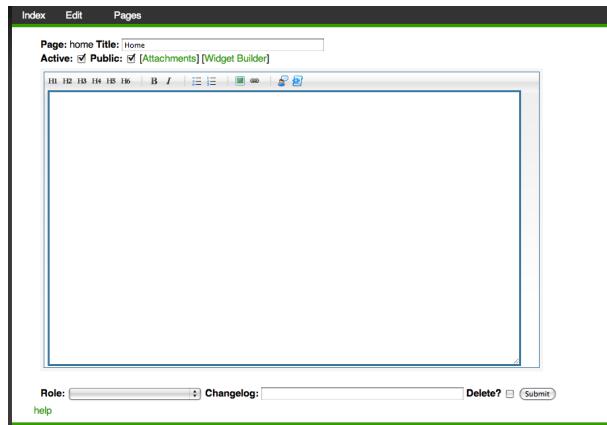
¹ http://127.0.0.1:8000/myapp/plugin_wiki/index



La página de índice plugin lista de las páginas creadas con el propio plugin y le permite crear otros nuevos, eligiendo una *slug*. Trate de crear una página de home. Usted será redirigido a

http://127.0.0.1:8000/myapp/plugin_wiki/page/home

Haga clic en *create page* para editar su contenido.



Por defecto, el plugin está en el nivel 3 lo que significa que los widgets se pueden insertar así como el código en las páginas. Por defecto se usa la sintaxis markmin para describir el contenido de la página.

13.3.1 Markmin sintaxis

Aquí está una cartilla para la sintaxis markmin:

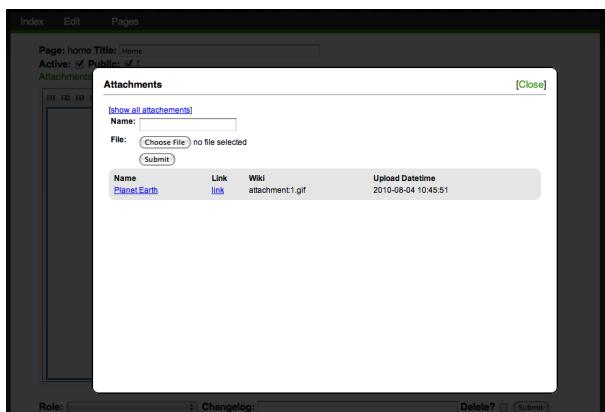
markmin	html
# title	<h1>title</h1>
## subtitle	<h2>subtitle</h2>
### subsubtitle	<h3>subsubtitle</h3>
bold	bold
"italic"	<i>italic</i>
http://...com	http:...com
[[name http://example.com]]	name
[[name http://...png left 200px]]	

Puedes añadir enlaces otras páginas

```
[[mylink name page:slug]]
```

Y si la página no existe, se le pedirá crear una:

La página de edición le permite agregar datos adjuntos a las páginas (es decir, los archivos estáticos)



y se puede vincular como

```
1 [[mylink name attachment:3.png]]
```

o incrustarlos con

```
1 [[myimage attachment:3.png center 200px]]
```

El tamaño (200px) es opcional. center no es opcional pero puede ser sustituido por left o right.

Puedes integrar blockquoted texto con

```
1 -----
2 this is blockquoted
3 -----
```

así como las tablas

```
1 -----
2 0 | 0 | X
3 0 | X | 0
4 X | 0 | 0
5 -----
```

y el texto literal

```
1 ``
2 verbatim text
3 ``
```

También puede anteponer un facultativo de la :class --- " final o el final. Para el texto blockquoted y tablas que se traducirá en la clase de la etiqueta, por ejemplo:

```
1 -----
2 test
3 -----:abc
```

se muestra como

```
1 <blockquote class="abc">test</blockquote>
```

Para el texto literal puede ser la clase utilizada para embeber contenido de diferentes tipos.

Puede, por ejemplo, código de inserción con resaltado de sintaxis, especificando el idioma con :code_language

```

1 `````
2 def index(): return 'hello world'
3 ```:code_python

```

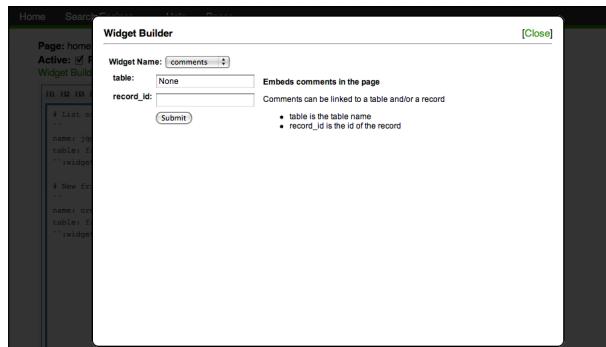
Puede incrustar widgets:

```

1 `````
2 name: widget_name
3 attribute1: value1
4 attribute2: value2
5 ```:widget

```

Desde la página de edición puede hacer clic en "widget builder" a los widgets de inserción de una lista, de forma interactiva:



(Para una lista de widget véase la siguiente subsección).

También puede incrustar código de la plantilla web2py idioma:

```

1 `````
2 {{for i in range(10):}}<h1>{{=i}}</h1>{{=pass}}
3 ```:template

```

13.3.2 Page Permisos

Cuando se edita una página usted puede encontrar los siguientes campos:

- incumplimientos **active** (para True). Si una página no está activa, no será accesible a los visitantes (aunque público).
- incumplimientos **public** (para True). Si una página es público se puede acceder por los visitantes sin entrar
- incumplimientos **Role** (en Ninguno). Si una página tiene un papel, la página se puede acceder sólo por los visitantes que se registran en un son miembros del grupo con el papel que le corresponde.

13.3.3 Páginas especiales

meta-menu contiene el menú. Si esta página no existe web2py utiliza el `response.menu` se define en "models/menu.py". El contenido de la página meta-menú reemplaza el menú. La sintaxis es la siguiente:

```

1 Item 1 Name http://link1.com
2   Submenu Item 11 Name http://link11.com
3     Submenu Item 12 Name http://link12.com
4       Submenu Item 13 Name http://link13.com
5 Item 2 Name http://link1.com
6   Submenu Item 21 Name http://link21.com
7     Submenu Item 211 Name http://link211.com
8       Submenu Item 212 Name http://link212.com
9     Submenu Item 22 Name http://link22.com
10    Submenu Item 23 Name http://link23.com

```

donde la sangría determinar la estructura submenú. Cada elemento está compuesto por el texto del elemento de menú seguido por un enlace. Un enlace puede ser `page:slug`. Un enlace puede ser `None` es el elemento de menú no se vincula cualquier página. espacios extra son ignorados.

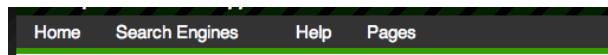
He aquí otro ejemplo:

```

1 Home      page:home
2 Search Engines None
3   Yahoo    http://yahoo.com
4   Google   http://google.com
5   Bing     http://bing.com
6 Help      page:help

```

Eso hace de la siguiente manera:



`meta-code` `meta-footer` `meta-code` es otra página especial y debe contener código web2py. Este de esta una extensión de sus modelos y, de hecho, puede colocar el código modelo en aquí. Se ejecuta cuando el código se ejecuta "models/plugin_wiki.py".

Puede definir tablas en `meta-code`.

Por ejemplo, puede crear una tabla sencilla amigos poniendo esto en `meta-code`:

```
1 db.define_table('friend',Field('name',requires=IS_NOT_EMPTY()))
```

y usted puede crear una interfaz de gestión de la página con el amigo por la incrustación en una página de su elección el siguiente código:

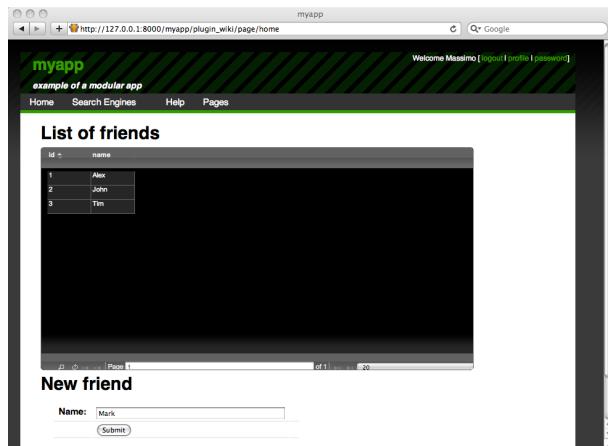
```

1 # List of friends
```
2
3 name: jqgrid
4 table: friend
5 ``:widget
6
7 # New friend
```
8
9 name: create
10 table: friend
11 ``:widget

```

La página tiene dos cabeceras (empezando por el n °): "List of friends" and "New friend". La página contiene dos widgets (en las cabeceras correspondientes).

entes): un widget que muestra jqgrid amigos y crear una porquería widget para añadir un nuevo amigo.



`meta-header`, `meta-footer`, `meta-sidebar` no son utilizados por el diseño predefinido en "welcome/views/layout.html". If you want to use them edit "layout.html" **admin** utiliza (o la cáscara) y coloque las siguientes etiquetas en los lugares adecuados:

```

1 {{=plugin_wiki.embed_page('meta-header') or ''}}
2 {{=plugin_wiki.embed_page('meta-sidebar') or ''}}
3 {{=plugin_wiki.embed_page('meta-footer') or ''}}
```

De esta manera el contenido de esas páginas aparecerá en la cabecera, barra lateral y pie de página en el diseño.

13.3.4 Configuración `plugin_wiki`

Al igual que con cualquier otra extensión en el "models/db.py" que usted puede hacer

```

1 from gluon.tools import PluginManager
2 plugins = PluginManager()
```

```

3 plugins.wiki.editor = auth.user.email==mail.settings.sender
4 plugins.wiki.level = 3
5 plugins.wiki.mode = 'markmin' or 'html'
6 plugins.wiki.theme = 'ui-darkness'
```

donde

- **editor** es cierto si la corriente se conectó el usuario tiene autorización para editar páginas plugin_wiki
- **level** es el permiso: 1 - para editar artículos regulares, 2 - a los widgets incrustar en páginas, 3 - el código embed
- **mode** determina si se debe usar un editor de "markmin" editor or a WYSIWYG "html".
- **theme** es el nombre de la necesaria jQuery UI tema. Por defecto sólo el "ui-darkness" de color neutro está instalado.

Usted puede agregar temas aquí:

```
1 static/plugin_wiki/ui/%(theme)s/jquery-ui-1.8.1.custom.css
```

13.3.5 Reproductores actual

Cada widget puede ser embebido en páginas plugin_wiki tanto y en web2py normal plantillas.

Por ejemplo, para incrustar un vídeo de YouTube en una página plugin_wiki que usted puede hacer

```

1 ``
2 name: youtube
3 code: l7AWnffFRc7g
4 `` :widget
```

o para integrar el mismo artilugio en una vista web2py que puede hacer:

```
1 {{=plugin_wiki.widget('youtube',code='l7AWnfFRc7g')}}
```

Cualquiera de los casos esta salida el:



Widget argumentos que no tienen un valor por defecto son obligatorios.

Aquí está una lista de todos los reproductores actuales:

```
1 read(table,record_id=None)
```

Lee y muestra un registro

- `table` es el nombre de una tabla
- `record_id` es un número de registro

```
1 create(table,message='',next='',readonly_fields='',
2       hidden_fields='',default_fields '')
```

Muestra un registro de crear el formulario

- `table` es el nombre de una tabla
- `message` es un mensaje que se muestra después de registro es creado
- `next "page/index/[id]"` es dónde redirigir ejemplo,
- `readonly_fields` es una lista de campos separados por comas
- `hidden_fields` es una lista de campos separados por comas
- `default_fields` es una lista separada por comas de "fieldname=value"

```
1 update(table,record_id='',message='',
2        readonly_fields='',hidden_fields='',default_fields='')
```

Muestra un formulario actualizar el registro

- `table` es el nombre de una tabla
- `record_id` es aquel registro que se actualiza o ((request.args = (-1)))
- `message` es un mensaje que se muestra después de registro es creado
- `next "page/index/[id]"` es dónde redirigir ejemplo,
- `readonly_fields` es una lista de campos separados por comas
- `hidden_fields` es una lista de campos separados por comas
- `default_fields` es una lista separada por comas de "fieldname=value"

```
1 select(table,query_field='',query_value='',fields='')
```

Enumera todos los registros de la tabla

- `table` es el nombre de una tabla
- `query_field` y `query_value` si está presente filtrará los registros por consulta de acuerdo con el `query_field == query_value`
- `fields` es una lista de campos separados por comas para que se muestre

```
1 search(table,fields='')
```

Un Widgets de selección de registros

- `table` es el nombre de una tabla
- `fields` es una lista de campos separados por comas para que se muestre

```
1 jqgrid(table,fieldname=None,fieldvalue=None,col_widths='',
2         _id=None,fields='',col_width=80,width=700,height=300)
```

Incrusta un plugin jqGrid

- `table` es el nombre de la tabla
- `fieldname`, `fieldvalue` son un filtro opcional (`nombredcampo == fieldvalue`)
- `_id` es el "id" de la DIV que contiene el jqGrid
- `columns` es una lista de nombres de columnas que se mostrarán
- `col_width` es el ancho de cada columna (por defecto)
- `height` es la altura de la jqGrid
- `width` es el ancho de la jqGrid

```
1 latex(expression)
```

Utiliza cartografía Google API para integrar LaTeX

```
1 pie_chart(data='1,2,3',names='a,b,c',width=300,height=150,align='center')
```

Insertar un gráfico circular

- `data` es una lista de valores separados por comas
- `names` es una lista de etiquetas separadas por comas (una para cada elemento de datos)
- `width` es el ancho de la imagen
- `height` es la altura de la imagen

- `align` determina la alineación de la imagen

```
1 bar_chart(data='1,2,3',names='a,b,c',width=300,height=150,align='center')
```

Utiliza Google API de gráficos para incrustar un gráfico de barras

- `data` es una lista de valores separados por comas
- `names` es una lista de etiquetas separadas por comas (una para cada elemento de datos)
- `width` es el ancho de la imagen
- `height` es la altura de la imagen
- `align` determina la alineación de la imagen

```
1 slideshow(table,field='image',transition='fade',width=200,height=200)
```

Incrusta un pase de diapositivas. Se pone las imágenes de una tabla.

- `table` es el nombre de la tabla
- `field` es el campo de la carga en la tabla que contiene las imágenes
- `transition` determina el tipo de transición, por ejemplo, decoloración, etc
- `width` es el ancho de la imagen
- `height` es la altura de la imagen

```
1 youtube(code,width=400,height=250)
```

Incrusta un video de youtube (por código)

- `code` es el código del vídeo
- `width` es el ancho de la imagen
- `height` es la altura de la imagen

```
1 vimeo(code,width=400,height=250)
```

Incrusta un vídeo viemo (por código)

- `code` es el código del vídeo
- `width` es el ancho de la imagen
- `height` es la altura de la imagen

```
1 mediaplayer(src,width=400,height=250)
```

Incrusta un archivo multimedia (como flash video o un archivo mp3)

- `src` es el `src` del video
- `width` es el ancho de la imagen
- `height` es la altura de la imagen

```
1 comments(table='None',record_id=None)
```

Incrusta comentarios en la página de comentarios puede estar vinculado a una tabla y / o grabar un

- `table` es el nombre de la tabla
- `record_id` es el identificador del registro

```
1 tags(table='None',record_id=None)
```

Incrusta etiquetas en las etiquetas de la página puede estar vinculado a una tabla y / o grabar un

- `table` es el nombre de la tabla
- `record_id` es el identificador del registro

```
1 tag_cloud()
```

Incrusta una nube de etiquetas

Incrusta

```
1 map(key='....', table='auth_user', width=400, height=200)
```

un mapa de Google obtiene puntos en el mapa de una tabla

- key es la API de google map clave (por defecto para las obras 127.0.0.1)
- table es el nombre de la tabla
- width es el ancho del mapa
- height es la altura mapa

```
1 The table must have columns: latitude, longitude and map_popup.  
2 When clicking on a dot, the map_popup message will appear.
```

```
1 iframe(src, width=400, height=300)
```

Incrusta una página en una <iframe></iframe>

```
1 load_url(src)
```

Carga el contenido de la URL con la función CARGA

```
1 load_action(action, controller='', ajax=True)
```

Carga el contenido de la URL (request.application, controlador, la acción) con la función CARGA

13.3.6 Ampliación de Widgets

Adición de widgets para plugin_wiki se puede hacer mediante la creación de un nuevo modelo "models/plugin_wiki_" archivo llamado *name name* donde es arbitraria y que el documento que contiene algo como:

```

1 class PluginWikiWidgets(PluginWikiWidgets):
2     @staticmethod
3     def my_new_widget(arg1, arg2='value', arg3='value'):
4         """
5             document the widget
6         """
7         return "body of the widget"

```

La primera línea indica que se amplía la lista de widgets. Dentro de la clase, se puede definir como la función que sea necesario. Cada función estática es un nuevo widget, excepto las funciones que comienzan por guión bajo. La función puede tomar un número arbitrario de argumentos que pueden o no tener valores predeterminados. El docstring de la función debe documentar la función que utiliza la sintaxis markmin sí mismo.

Cuando widgets están incrustados en páginas plugin_wiki, los argumentos serán pasados al widget como cadenas. Esto significa que la función widget debe ser capaz de aceptar las cadenas para todos los argumentos y, finalmente, convertirlo en la representación requerida. Usted puede decidir lo que la representación de cadena debe ser, simplemente asegúrese de que esto está documentado en el docstring.

El widget puede devolver una cadena o ayudantes web2py. En este último caso, se serializará usando .xml().

Observe cómo el nuevo widget puede acceder a cualquier variable declarada en el espacio de nombres global.

Como ejemplo vamos a crear un nuevo widget que muestren el "contact/ask" form created at the beginning of this chapter. This can be done by creating a file "models/plugin_wiki_contact" que contenga:

```

1 class PluginWikiWidgets(PluginWikiWidgets):
2     @staticmethod
3     def ask(email_label='Your email', question_label='question'):
4         """
5             This plugin will display a contact us form that allows
6             the visitor to ask a question.
7             The question will be emailed to you and the widget will

```

```
8      disappear from the page.
9      The arguments are
10
11      - email_label: the label of the visitor email field
12      - question_label: the label of the question field
13
14      """
15      form=SQLFORM.factory(
16          Field('your_email',requires=IS_EMAIL(),label=email_label),
17          Field('question',requires=IS_NOT_EMPTY()),label=question_label)
18      if form.accepts(request.vars,session):
19          if mail.send(to='admin@example.com',
20                      subject='from %s' % form.vars.your_email,
21                      message = form.vars.question):
22              div_id = request.env.http_web2py_component_element
23              command="jQuery('#%s').hide()" % div_id
24              response.headers['web2py-component-command']=command
25              response.headers['web2py-component-flash']='thanks you'
26          else:
27              form.errors.your_email="Unable to send the email"
28      return form.xml()
```

Widgets plugin_wiki no se representan por una visión menos que la función response.render(...) se llama explícitamente por el widget.

Index

- `_init__.py`, 157
- `_and`, 347
- `_count`, 279
- `_dbname`, 241
- `_delete`, 279
- `_insert`, 279
- `_lastdb`, 253
- `_select`, 279
- `_update`, 279
- `_uri`, 241
- A, 207
- about, 132, 157
- accepts, 95, 109, 292
- Access Control, 355
- access restriction, 41
- Active Directory, 370
- admin, 83, 129, 468
- admin.py, 132
- Adobe Flash, 411
- Ajax, 116, 421, 504
- ALL, 258
- and, 264
- Apache, 443
- appadmin, 100, 140
- appliances, 83
- as_dict, 266
- as_list, 266, 398
- ASCII, 55
- ASP, 31
- asynchronous, 438
- Auth, 355
- auth.user, 360
- auth.user_id, 360
- Authentication, 390, 418
- autocomplete, 352
- autodelete, 314
- B, 207
- bar chart, 533
- belongs, 278
- blob, 246
- block, 232
- BODY, 207
- bulk_insert, 251
- cache, 157, 158, 169
- cache clear, 171
- cache controller, 170
- cache select, 286
- cache view, 171
- cache.disk, 169
- cache.ram, 169
- CAPTCHA, 364
- CAS, 390
- CENTER, 207
- CGL, 443
- checkbox, 212
- Cherokee, 460
- class, 67
- CLEANUP, 345
- CODE, 207
- command line, 144
- comments, 534
- commit, 251
- component, 503
- component plugin, 515
- compute, 268
- confirmation, 432
- connection pooling, 238
- connection strings, 237
- content-disposition, 310
- controllers, 157
- cookies, 178
- cooperation, 193
- count, 265
- cPickle, 77
- cron, 184
- cross site scripting, 39
- CRUD, 319, 527
- crud.create, 319, 320

crud.delete, 319, 320
 crud.read, 320
 crud.search, 319
 crud.select, 319, 320
 crud.tables, 319, 320
 crud.update, 319, 320
 CRYPT, 345
 cryptographic store, 41
 CSRF, 40
 CSV, 402
 csv, 280
 custom validator, 348
 DAC, 355
 DAL, 159, 235, 241
 DAL shortcuts, 259
 database drivers, 235
 databases, 157
 date, 75, 276
 datetime, 75, 276
 day, 277
 DB2, 241
 def, 62, 203
 default, 242
 define_table, 236, 242
 deletable, 433
 delete, 265
 delete_label, 306
 dict, 58
 dir, 53
 dispatching, 147
 distinct, 263
 distributed transactions, 255
 DIV, 209
 Document Object Model (DOM), 205
 Domino, 370
 drop, 253
 EDIT, 133
 effects, 426
 element, 221
 elements, 221
 elif, 64, 201
 else, 64, 201, 202
 EM, 209
 email attachments, 375
 email from GAE, 375
 email html, 375
 email logging, 374
 emails, 233
 encode, 55
 env, 162
 errors, 136, 157
 escape, 199
 eval, 72
 examples, 83
 except, 64, 202
 Exception, 64
 exclude, 267
 exec, 72
 exec_environment, 191
 executesql, 252
 export, 280
 Expression, 237
 extent, 224
 ez.css, 226
 Facebook, 361, 371
 fake_migrate, 249
 FastCGI, 457, 460
 favicon, 182
 fcgihandler, 457
 fetch, 491
 Field, 236, 242, 257
 Field constructor, 243
 fields, 246, 305
 FIELDSET, 209
 file.read, 69
 file.seek, 69
 file.write, 69
 finally, 64, 202
 find, 267
 FireBird, 241
 first, 266
 flash mediaplayer, 534
 Flickr, 361
 for, 60, 200
 FORM, 95, 209
 form, 91, 292
 format, 100
 formname, 292
 formstyle, 307
 GAE login, 370
 geocode, 491
 GET, 148
 Gmail, 369
 Google, 361
 Google App Engine, 481
 Google map, 535
 grouping, 274
 H1, 210
 HAProxy, 472
 HEAD, 210
 help, 53
 helpers, 159, 204
 hideerror, 331
 hour, 277
 HTML, 210, 284
 HTTP, 158, 174
 httpserver.log, 493
 id_label, 306
 if, 64, 201
 IF_MODIFIED_SINCE, 148
 IFRAME, 212
 import, 73, 189, 280
 improper error handling, 40

include, 224
 index, 84
 information leakage, 40
 Informix, 241
 inheritance, 288
 init, 179
 injection flaws, 39
 inner join, 273
 INPUT, 95, 211
 insecure object reference, 40
 insert, 250
 internationalization, 158, 176
 IS_ALPHANUMERIC, 333
 IS_DATE, 333
 IS_DATE_IN_RANGE, 333
 IS_DATETIME, 334
 IS_DATETIME_IN_RANGE, 334
 IS_DECIMAL_IN_RANGE, 334
 IS_EMAIL, 98, 335
 IS_EMPTY_OR, 345
 IS_EQUEL_TO, 335
 IS_EXPR, 335
 IS_FLOAT_IN_RANGE, 335
 IS_IMAGE, 341
 IS_IN_DB, 98, 347
 IS_IN_SET, 336
 IS_INT_IN_RANGE, 336
 IS_IPV4, 344
 IS_LENGTH, 337
 IS_LIST_OF, 337
 IS_LOWER, 338, 344
 IS_MATCH, 338
 IS_NOT_EMPTY, 95, 98, 339
 IS_NOT_IN_DB, 346
 IS_NULL_OR, 345
 IS_SLUG, 341
 IS_STRONG, 341
 IS_TIME, 339
 IS_UPLOAD_FILENAME, 342
 IS_UPPER, 344
 IS_URL, 339
 Janrain, 361
 join, 273
 jqGrid, 527, 532
 JSON, 396, 415
 JSONRPC, 407
 JSP, 31
 Jython, 500
 keepvalues, 298
 KPAX, 130
 LABEL, 212
 labels, 305
 lambda, 70
 languages, 157
 last, 266
 latex, 532
 layout, 92
 layout plugin, 519
 layout.html, 224
 LDAP, 367, 369
 left outer join, 273
 LEGEND, 213
 length, 242
 LI, 212
 license, 43, 132, 157
 Lighttpd, 457
 like, 277
 limitby, 264
 LinkedIn, 361, 372
 list, 56
 LOAD, 504
 load, 504
 local_import, 189
 Lotus Notes, 370
 lower, 277
 MAC, 355
 mail.send, 374
 malicious file execution, 39
 many-to-many, 275
 markdown, 124
 MARKIN, 213
 markmin, 124
 markmin syntax, 523
 memcache, 478
 MENU, 219
 menu, 225
 menu and plugin_wiki, 526
 Mercurial, 140
 META, 213
 meta, 225
 meta-header, 527
 meta-menu, 527
 meta-sidebar, 527
 migrate, 242
 migrations, 248
 minutes, 277
 mod_proxy, 443
 mod_python, 443
 mod_wsgi, 443
 Model-View-Controller, 33
 models, 157
 modules, 157
 month, 277
 MSSQL, 241
 MySpace, 361
 MySQL, 241
 named id field, 246
 nested select, 278
 not, 264
 notnull, 242

OAuth, 371
 OBJECT, 213
 OL, 214
 ON, 214
 ondelete, 242
 one to many, 271
 onvalidation, 294
 OpenID, 361, 370
 OpenLDAP, 370
 OPTGROUP, 214
 OPTION, 214
 or, 264
 Oracle, 241
 orderby, 263
 os, 74
 os.path.join, 74
 os.unlink, 74
 outer join, 273

P, 215
 page layout, 224
 pagination, 492
 PAM, 143, 367, 369
 PARTIAL CONTENT, 148
 password, 143
 PDF, 417
 PGP, 376
 PHP, 31
 pie chart, 532
 PIL, 364
 plugin, 503
 plugin_wiki, 520
 PluginManager, 515
 POST, 148
 postback, 93
 PostgresSQL, 241
 PRE, 215
 prettydate, 491
 private, 157
 PyAMF, 411
 Pyjamas, 407

PyRTF, 416
 Python, 51

Query, 237, 257

radio, 212
 random, 73
 raw SQL, 279
 RBAC, 355
 reCAPTCHA, 364
 recursive selects, 260
 redirect, 93, 158, 174
 referencing, 272
 removing application, 479
 ReportLab, 417
 request, 30, 158, 160, 162
 request.ajax, 149
 request.application, 148
 request.args, 109, 148
 request.controller, 148
 request.cookies, 160
 request.function, 148
 request.get_vars, 149
 request.post_vars, 149
 request.url, 149
 request.vars, 91, 149
 required, 242
 requires, 95, 242
 reserved Keywords, 240
 response, 158, 164
 response.body, 164
 response.cookies, 164
 response.download, 164
 response.files, 164
 response.flash, 109, 164
 response.headers, 164
 response.menu, 164, 225
 response.meta, 164, 225
 response.postprocessing,
 164
 response.render, 164

response.status, 164
 response.stream, 109, 164
 response.subtitle, 164
 response.title, 164
 response.view, 164
 response.write, 164, 199
 return, 62, 203
 robots, 182
 rollback, 251
 routes_in, 180
 routes_on_error, 184
 routes_out, 180
 Row, 236, 258
 Rows, 236, 257, 273
 RPC, 403
 RSS, 116, 399
 rss, 127
 RTF, 416

sanitize, 124, 206
 scaffolding, 83
 scalability, 469
 SCRIPT, 215
 seconds, 277
 secure communications, 41
 security, 39, 468
 SELECT, 215
 select, 106, 257
 selected, 215
 session, 89, 157, 158, 167
 session.connect, 167
 session.forget, 167
 session.secure, 167
 Set, 237, 257
 shell, 52
 showid, 306
 simplejson, 415
 site, 129
 slideshow, 533
 SMS, 496
 SMTP, 369

SOAP, 414
sort, 267
SPAN, 216
sql.log, 242
SQLFORM, 109
SQLite, 241
SQLTABLE, 261
static, 157
static files, 148
Storage, 160
str, 55
streaming, 499
STYLE, 216
sub-menu, 226
submit_button, 306
sum, 278
superfish, 226
sys, 75
sys.path, 75

T, 158, 176
TABLE, 216
Table, 247, 257
tables, 246
TAG, 219
tag could, 534
tags, 534
TBODY, 217

TD, 216
template language, 197
tests, 157
TEXTAREA, 217
TFOOT, 218
TH, 218
THEAD, 218
time, 75, 76, 276
TITLE, 218
TLS, 369
TR, 216, 218
truncate, 250
try, 64, 202
TT, 219
tuple, 57
type, 54, 242

UL, 219
Unicode, 55
unique, 242
update, 265
update_record, 266
upgrades, 489
upload, 96
uploadfield, 242
uploads, 157
uploadseparate, 242
upper, 277

URL, 93, 171
url mapping, 147
url rewrite, 180
UTF8, 55

validators, 159, 332
views, 157, 197
Vimeo, 533
virtualfields, 269

Web Services, 395
welcome, 83
while, 62, 201
widget in plugin_wiki, 520
wiki, 116, 520
Windows service, 467
WSGI, 194, 443
WYSIWYG, 529

x509, 376
XHTML, 210, 211
XML, 205, 284, 396
XMLRPC, 116, 128, 405

year, 277
YouTube, 533

Bibliography

- [1] <http://www.web2py.com>
- [2] <http://www.python.org>
- [3] <http://en.wikipedia.org/wiki/SQL>
- [4] <http://www.sqlite.org/>
- [5] <http://www.postgresql.org/>
- [6] <http://www.mysql.com/>
- [7] <http://www.microsoft.com/sqlserver>
- [8] <http://www.firebirdsql.org/>
- [9] <http://www.oracle.com/database/index.html>
- [10] <http://www-01.ibm.com/software/data/db2/>
- [11] <http://www-01.ibm.com/software/data/informix/>
- [12] <http://www.ingres.com/>
- [13] <http://code.google.com/appengine/>
- [14] <http://en.wikipedia.org/wiki/HTML>
- [15] <http://www.w3.org/TR/REC-html40/>

- [16] <http://www.php.net/>
- [17] http://en.wikipedia.org/wiki/Web_Server_Gateway_Interface
- [18] <http://www.python.org/dev/peps/pep-0333/>
- [19] <http://www.owasp.org>
- [20] <http://www.pythonsecurity.org>
- [21] http://en.wikipedia.org/wiki/Secure_Sockets_Layer
- [22] <https://launchpad.net/rocket>
- [23] <http://www.cdolivet.net/editarea/>
- [24] <http://nicedit.com/>
- [25] <http://pypi.python.org/pypi/simplejson>
- [26] <http://pyrtf.sourceforge.net/>
- [27] <http://www.dalkescientific.com/Python/PyRSS2Gen.html>
- [28] <http://www.feedparser.org/>
- [29] <http://code.google.com/p/python-markdown2/>
- [30] <http://www.tummy.com/Community/software/python-memcached/>
- [31] <http://www.fsf.org/licensing/licenses/info/GPLv2.html>
- [32] <http://jquery.com/>
- [33] <https://www.web2py.com/cas>
- [34] <http://www.web2py.com/appliances>
- [35] <http://www.web2py.com/AlterEgo>
- [36] <http://www.python.org/dev/peps/pep-0008/>
- [37] <http://www.network-theory.co.uk/docs/pytut/>
- [38] <http://oreilly.com/catalog/9780596158071>

- [39] <http://www.python.org/doc/>
- [40] http://en.wikipedia.org/wiki/Cascading_Style_Sheets
- [41] <http://www.w3.org/Style/CSS/>
- [42] <http://www.w3schools.com/css/>
- [43] <http://en.wikipedia.org/wiki/JavaScript>
- [44] <http://www.amazon.com/dp/0596000480>
- [45] http://en.wikipedia.org/wiki/Cron#crontab_syntax
- [46] <http://www.xmlrpc.com/>
- [47] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol
- [48] <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
- [49] <http://en.wikipedia.org/wiki/XML>
- [50] <http://www.w3.org/XML/>
- [51] <http://www.ez-css.org>
- [52] <http://en.wikipedia.org/wiki/XHTML>
- [53] <http://www.w3.org/TR/xhtml1/>
- [54] <http://www.w3schools.com/xhtml/>
- [55] <http://www.web2py.com/layouts>
- [56] <http://sourceforge.net/projects/zxjdbc/>
- [57] <http://pypi.python.org/pypi/psycopg2>
- [58] <http://sourceforge.net/projects/mysql-python>
- [59] http://python.net/crew/atuining/cx_Oracle/
- [60] <http://pyodbc.sourceforge.net/>
- [61] <http://kinterbasdb.sourceforge.net/>

- [62] <http://informixdb.sourceforge.net/>
- [63] <http://pypi.python.org/simple/ingresdbi/>
- [64] http://docs.python.org/library/csv.html#csv.QUOTE_ALL
- [65] <http://www.faqs.org/rfcs/rfc2616.html>
- [66] <http://www.faqs.org/rfcs/rfc2396.html>
- [67] <http://tools.ietf.org/html/rfc3490>
- [68] <http://tools.ietf.org/html/rfc3492>
- [69] <http://mail.python.org/pipermail/python-list/2007-June/617126.html>
- [70] <http://mail.python.org/pipermail/python-list/2007-June/617126.html>
- [71] <http://www.recaptcha.net>
- [72] http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules
- [73] <http://www.reportlab.org>
- [74] <http://gdwarner.blogspot.com/2008/10/brief-pyjamas-django-tutorial.html>
- [75] <http://en.wikipedia.org/wiki/AJAX>
- [76] <http://www.learningjquery.com/>
- [77] <http://ui.jquery.com/>
- [78] http://en.wikipedia.org/wiki/Common_Gateway_Interface
- [79] <http://www.apache.org/>
- [80] <http://httpd.apache.org/download.cgi>
- [81] http://adal.chiriliuc.com/mod_wsgi/revision_1018_2.3/mod_wsgi_py25_apache22/mod_wsgi.so

- [82] http://httpd.apache.org/docs/2.0/mod/mod_proxy.html
- [83] <http://httpd.apache.org/docs/2.2/mod/core.html>
- [84] <http://sial.org/howto/openssl/self-signed>
- [85] <http://code.google.com/p/modwsgi/>
- [86] <http://www.lighttpd.net/>
- [87] <http://www.cherokee-project.com/download/>
- [88] <http://www.fastcgi.com/>
- [89] <http://www.apsis.ch/pound/>
- [90] <http://haproxy.lwt.eu/>
- [91] <http://pyamf.org/>
- [92] http://en.wikipedia.org/wiki/List_of_HTTP_status_codes
- [93] <http://www.authorize.net/>
- [94] <http://sourceforge.net/projects/zxjdbc/>
- [95] <http://www.zentus.com/sqlitejdbc/>