

MASSIMO DI PIERRO

WEB2PY MANUAL

3RD EDITION ALPHA PORTUGUESE (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	Introdução	29
1.1	Princípios	32
1.2	Web Frameworks	33
1.3	Model-View-Controller	35
1.4	Por web2py	39
1.5	Segurança	40
1.6	Na caixa	44
1.7	Licença	45
1.8	Agradecimentos	47
1.9	Sobre este livro	49
1.10	Elements of Style	51

2 A linguagem Python	53
2.1 Sobre Python	53
2.2 Arranque	54
2.3 help, dir	55
2.4 Tipos	56
2.4.1 str	57
2.4.2 list	58
2.4.3 tuple	59
2.4.4 dict	60
2.5 Sobre recuo	62
2.6 for...in	62
2.7 while	64
2.8 def...return	64
2.9 if...elif...else	66
2.10 try...except...else...finally	66
3 class	69
3.1 Atributos especiais, métodos e operadores	70

3.2	Arquivo de entrada / saída	71
3.3	lambda	72
3.4	exec, eval	74
3.5	import	75
3.5.1	os	76
3.5.2	sys	77
3.5.3	datetime	77
3.5.4	time	78
3.5.5	cPickle	79
4	Visão global	81
4.1	Startup	81
4.2	Diga Olá	86
4.3	Vamos contar	92
4.4	Say My Name	93
4.5	Postbacks	95
4.6	Uma Imagem Blog	98
4.7	Adicionando CRUD	114

4.8	Adicionando autenticação	116
4.9	Configuring o Layout	118
4.10	A Wiki	118
4.11	Mais informações sobre admin	131
4.11.1	<i>site</i>	131
4.11.2	<i>about</i>	134
4.11.3	<i>edit</i>	135
4.11.4	<i>errors</i>	138
4.11.5	<i>Mercurial</i>	142
4.12	Mais informações sobre appadmin	142
5	O Core	145
5.1	Opções de linha de comando	145
5.2	Despacho	148
5.3	Bibliotecas	153
5.4	Aplicações	158
5.5	API	160
5.6	<i>request</i>	161

5.7 <code>response</code>	165
5.8 <code>session</code>	168
5.9 <code>cache</code>	169
5.10 <code>URL</code>	172
5.11 <code>HTTP e redirect</code>	174
5.12 <code>Te</code>	176
5.13 <code>Cookies</code>	179
5.14 Aplicação <code>init</code>	180
5.15 URL Rewrite	181
5.16 Rotas em caso de erro	184
5.17 <code>Cron</code>	185
5.18 Processos Antecedentes e filas de tarefas	188
5.19 Terceira Parte Módulos	190
5.20 Execution Environment	191
5.21 Cooperação	193
5.22 <code>WSGI</code>	194
5.22.1 Externas Middleware	195
5.22.2 Interno Middleware	196

5.22.3 Chamando WSGI Applications	196
6 As vistas	199
6.1 Sintaxe Básica	202
6.1.1 for...in	202
6.1.2 while	203
6.1.3 if...elif...else	203
6.1.4 try...except...else...finally	204
6.1.5 def...return	205
6.2 HTML Helpers	206
6.2.1 XML	207
6.2.2 Built-in Helpers	208
6.2.3 Helpers personalizada	221
6.3 BEAUTIFY BEAUTIFY	222
6.4 Server-side DOM e Parsing	223
6.4.1 elementos	223
6.4.2 parent	225
6.4.3 flatten	225

6.4.4	Análise	226
6.5	Page Layout	226
6.5.1	Default Layout da Página	228
6.5.2	Personalizar o layout padrão	231
6.6	Funções em Views	232
6.7	Blocos em Views	234
6.8	Usando o Modelo de sistema para gerar e-mails	235
7	A camada de abstração	237
7.1	Dependências	237
7.2	Connection Strings	239
7.2.1	Connection Pooling	240
7.2.2	Falhas de conexão	241
7.2.3	Bancos de dados replicados	241
7.3	Palavras-chave reservadas	242
7.4	DAL, Table, Field	243
7.5	Record Representação	244
7.6	Migrações	250

7.7	Fixação Migrações Broken	251
7.8	insert	252
7.9	commit e rollback	253
7.10	Raw SQL	254
7.10.1	executesql	254
7.10.2	_lastsql	255
7.11	drop	255
7.12	Índices	255
7.13	Bases de dados do Legacy e introduzidos quadros	256
7.14	Distributed Transaction	257
7.15	Manual Uploads	258
7.16	Query, Set, Rows	258
7.17	select	259
7.17.1	Atalhos	261
7.17.2	Buscando uma Row	262
7.17.3	selects recursiva	262
7.17.4	Serializing no Rows Views	263
7.17.5	orderby, groupby, limitby, distinct	265

7.17.6 Operadores lógicos	266
7.17.7 count, delete, update	267
7.17.8 Expressões	268
7.17.9 update_record	268
7.17.10 first e last	269
7.17.11 as_dict e as_list	269
7.17.12 find, exclude, sort	269
7.18 Computadorizada Campos	271
7.19 Virtual Campos	271
7.20 Um relacionamento de muitos para	274
7.20.1 Inner Joins	275
7.20.2 Left Outer Join	276
7.20.3 Agrupamento e Contagem	276
7.21 Muitos para muitos	277
7.22 Outros operadores	279
7.22.1 like, upper, lower	279
7.22.2 year, month, day, hour, minutes, seconds	280
7.22.3 belongs	280

7.23 Gerando SQL-primas	281
7.24 Exportação e importação de dados	282
7.24.1 CSV (um quadro de cada vez)	282
7.24.2 CSV (todas as tabelas de uma vez)	283
7.24.3 CSV e sincronização de banco de dados remoto	284
7.24.4 HTML / XML (uma tabela de tempo)	286
7.24.5 Data Representation	287
7.25 Seleciona Caching	288
7.26 A auto-referência e Alias	289
7.27 Quadro sucessório	291
8 Formulários e validadores	293
8.1 FORM	294
8.1.1 Campos ocultos	298
8.1.2 keepvalues	300
8.1.3 onvalidation	300
8.1.4 Formas e redirecionamento	301
8.1.5 Múltiplas formas por página	302

8.1.6	Partilha Forms	303
8.2	SQLFORM	304
8.2.1	SQLFORM e insert/update/delete	309
8.2.2	SQLFORM em HTML	311
8.2.3	SQLFORM e Uploads	312
8.2.4	Armazenar o nome original	315
8.2.5	autodelete	316
8.2.6	Links para referenciar Records	316
8.2.7	Pré-preenchimento do formulário	319
8.2.8	SQLFORM sem banco de dados IO	319
8.3	SQLFORM.factory	320
8.4	CRUD	321
8.4.1	Configurações	323
8.4.2	Mensagens	326
8.4.3	Métodos	327
8.4.4	Record Versioning	329
8.5	Os formulários personalizados	330
8.5.1	CSS Convenções	332

8.5.2	Ocultar erros	333
8.6	Validadores	334
8.6.1	Validadores	335
8.6.2	Database Validators	348
8.6.3	Custom Validators	350
8.6.4	Validadores com dependências	352
8.7	Widgets	352
8.7.1	Autocomplete Widget	354
9	Controle de Acesso	357
9.1	Autenticação	359
9.1.1	Restrições sobre o registo	363
9.1.2	Integração com o OpenID, Facebook, etc	363
9.1.3	CAPTCHA e reCAPTCHA	366
9.1.4	Personalizando Auth	367
9.1.5	Renomeando tabelas Auth	368
9.1.6	Outros métodos e formulários de login Login	369
9.2	Auth e Mail	375

9.2.1	Email Depuração	376
9.2.2	Emails formulário Google App Engine	377
9.2.3	Mais exemplos de e-mail	377
9.2.4	x509 e PGP criptografia	378
9.3	Autorização	379
9.3.1	Decoradores	381
9.3.2	Combinando Requisitos	383
9.3.3	Autorização e de CRUD	383
9.3.4	Autorização e Downloads	385
9.3.5	Autenticação e Controle de Acesso Básico	385
9.3.6	Configurações e mensagens	386
9.4	Central Authentication Service	392
10	Serviços	397
10.1	Escutar um dicionário	398
10.1.1	HTML, XML e JSON	398
10.1.2	Generic Views	399
10.1.3	Rendering linhas	400

10.1.4	Formatos personalizados	401
10.1.5	RSS	401
10.1.6	CSV	404
10.2	Chamadas de procedimento remoto	405
10.2.1	XMLRPC	407
10.2.2	JSONRPC e Pijamas	409
10.2.3	AMFRPC	413
10.2.4	SOAP	416
10.3	API de baixo nível e Outras Receitas	416
10.3.1	simplejson	417
10.3.2	PyRTF	418
10.3.3	ReportLab e PDF	419
10.4	Serviços e autenticação	420
11	Ajax Receitas	423
11.1	web2py_ajax.html	423
11.2	Efeitos jQuery	428
11.2.1	Campos condicional em formulários	432

11.2.2 Confirmação em Apagar	434
11.3 Função O ajax	435
11.3.1 alvo Eval	436
11.3.2 Auto-realização	437
11.3.3 Ajax Submission Form	440
11.3.4 Voto e Avaliação	441
12 Implantação Receitas	445
12.1 Linux / Unix	449
12.1.1 Implantação One Step Production	449
12.1.2 configuração do Apache	449
12.1.3 mod_wsgi	451
12.1.4 mod_wsgi e SSL	454
12.1.5 mod_proxy	456
12.1.6 Comece como Daemon Linux	458
12.1.7 Lighttpd	459
12.1.8 Hospedagem Compartilhada com mod_python . . .	461
12.1.9 Cherokee com FastGGI	462

12.1.10 PostgreSQL	464
12.2 Windows	465
12.2.1 Apache e mod_wsgi	466
12.2.2 Comece como serviço do Windows	469
12.3 Sessões de Proteção e admin	470
12.4 Eficiência e Escalabilidade	471
12.4.1 Eficiência Truques	472
12.4.2 Sessões no banco de dados	473
12.4.3 HAProxy um平衡ador de carga de alta disponibilidade	474
12.4.4 Limpeza Sessions	476
12.4.5 Fazer upload de arquivos no banco de dados	477
12.4.6 Coleta de ingressos	478
12.4.7 Memcache	480
12.4.8 Sessões em Memcache	481
12.4.9 Remover Aplicativos	481
12.4.10 Usando servidores de bancos de dados replicados	481
12.5 Google App Engine	482

12.5.1 Configuração	484
12.5.2 Duração e de implantação	486
12.5.3 Configurando o manipulador	487
12.5.4 Evite o Filesystem	488
12.5.5 Memcache	489
12.5.6 Questões de banco de dados	489
12.5.7 GAME e HTTPS	490
13 Outras receitas	491
13.1 Melhoramento	491
13.2 Como distribuir seus aplicativos como binários	492
13.3 Buscando uma URL externa	493
13.4 Pretty datas	493
13.5 Geocoding	493
13.6 Paginação	494
13.7 httpserver.log eo Log File Format	495
13.8 Preenchendo banco de dados com dados fictícios	496
13.9 Enviar um SMS	497

13.10 Aceitar pagamentos com cartão de crédito	498
13.11 Twitter API	500
13.12 Streaming de Arquivos Virtual	501
13.13 Jython	502
14 Componentes e Plugins	503
14.1 Componentes	504
14.2 Plugins	511
14.2.1 Componente Plugins	515
14.2.2 Plugin Manager	517
14.2.3 plugins Layout	519
14.3 plugin_wiki	520
14.3.1 sintaxe Markmin	523
14.3.2 Permissões Page	526
14.3.3 Páginas especiais	526
14.3.4 Configurando plugin_wiki	528
14.3.5 Current Widgets	529
14.3.6 Estendendo Widgets	535

Bibliography 539

Preface

Estou muito preocupado com o crescimento do fosso digital e os problemas que isso cria para o livre fluxo de informações. A habilidade de construir aplicações web de boa qualidade, rápido e eficiente é de fundamental importância para permitir que mais pessoas interagem com a Internet de uma forma não-trivial e evitar grande jogador de monopolização da informação. É por isso que eu comecei a trabalhar na web2py.

Em geral, os efeitos de qualquer framework web é tornar o desenvolvimento web mais fácil, rápido e impedir que os desenvolvedores a cometer erros (em especial nas questões relacionadas à segurança). No web2py que abordar estas questões:

Ease of use se o objetivo principal para web2py. Para nós isso significa reduzir o tempo de aprendizagem e de implantação. É por isso que nós construímos web2py como um framework full-stack sem dependências. Ele não requer instalação e não tem arquivos de configuração. Ele inclui tudo que você precisa para inicialização, incluindo um servidor de web rápido multi-threaded (embora nós apoiamos os terceiros também), uma operação de banco de dados seguro (apoiamos muitos outros terceiros também), uma poderosa camada de abstração de banco de dados e um web-based IDE que lhe dá acesso a todas as funcionalidades web2py. Também reduziu a API para um mínimo (12 objetos núcleo apenas) por isso não é pouco para você memorizar. desenvolvimento *Faster* é o objetivo secundário para web2py. Por esta razão criamos

web2py de modo que cada função tem um padrão de comportamento (que pode ser substituído). Por exemplo, se você acabou de criar um modelo (ou seja, escolher a forma como o banco de dados deve armazenar os seus dados), web2py cria automaticamente um banco de dados baseado na web interface administrativa a ele, se você adicionar as ações do controlador (a lógica da sua aplicação), eles automaticamente têm padrão vistas a apresentar a saída em HTML, XML, JSON, RSS, etc web2py também gera automaticamente criar / atualizar / excluir as formas que você pode personalizar a sua opinião (a camada de apresentação do seu aplicativo).

Security é o fim mas não menos do web2py é dele. web2py deixa muito pouca liberdade para o desenvolvedor em assuntos relacionados à segurança. A camada de abstração de banco de dados impede Injeções de SQL. O modelo de linguagem impede vulnerabilidades Cross Site Scripting. Os formulários gerados pela prestação de web2py validação de campo e bloquear Cross Site Request falsificações. Senha são sempre armazenados criptografados. As sessões são armazenadas no servidor por padrão, para evitar adulteração Cookie e cookies de sessão são uuid para evitar roubo de cookies.

Internamente, o web2py tem um design top-down vs o design de baixo para cima de outros quadros. Não é construído adicionando camada por camada. Ela é construída a partir da perspectiva do usuário e tem sido constantemente aperfeiçoado dentro, a fim de tornar mais rápido e mais simples, mantendo sempre *backward compatibility*. Estou feliz em dizer que web2py é hoje um dos quadros mais rápido da web e também um dos menores. Mais de 50 pessoas têm contribuído com código e remendos para fazer web2py que é hoje e todos nós constantemente trabalhando para torná-lo ainda melhor.

web2py é livre para você usar. Se você se beneficia com isso, nós esperamos que você vai se sentir um pouco mais como contribuir de volta para a sociedade, sob qualquer forma que você escolhe.

1

Introdução

web2py (1) é livre, framework web open-source para o desenvolvimento ágil de aplicações web seguras conduzido de banco de dados, é escrito em Python (2) e programável em Python. web2py é um framework full-stack, o que significa que contém todos os componentes necessários para construir totalmente funcional aplicações web. web2py destina-se a guia de um desenvolvedor web a seguir as boas práticas de engenharia de software, como o uso do Model View Controller (MVC). web2py separa a representação de dados (o modelo) da apresentação de dados (a vista) e também a partir da lógica da aplicação e do fluxo de trabalho (o controlador). web2py fornece bibliotecas para ajudar o projeto do desenvolvedor, implementar e testar cada uma dessas três partes separadamente, e fá-los trabalhar juntos. web2py é construído para a segurança. Isto significa que automaticamente aborda muitas das questões que podem levar a vulnerabilidades de segurança, seguindo as práticas estabelecidas bem. Por exemplo, ele valida todas as entradas (para evitar as injeções), todos os escapes de saída (para evitar cross-site scripting), renomeia arquivos enviados (para evitar ataques de passagem de diretório). Web2py folhas aplicativo para desenvolvedores pouca escolha em assuntos relacionados à segurança. web2py inclui uma camada de abstração de banco de dados (DAL), que escreve SQL (3) dinamicamente para que você, desenvolvedor, não precisa. A DAL sabe como gerar SQL transparente para

SQLite (4), MySQL (6), PostgreSQL (5), MSSQL (7), FireBird (8), Oracle (9), IBM DB2 (10), Informix (11) e Ingres (12). A DAL também podem gerar chamadas função para o Google BigTable quando em execução no Google App Engine (GAE) (13). uma ou mais tabelas do banco de dados Uma vez que são definidos, web2py também gera um plenamente funcional de base de dados da administração interface web para acessar o banco de dados e tabelas. web2py difere de outros frameworks web em que ele é o único quadro para abraçar completamente o paradigma da Web 2.0, onde a Web é o computador. Na verdade, web2py não requer instalação ou configuração, que é executado em qualquer arquitetura, que pode executar Python (Windows, Windows CE, Mac OS X, iPhone e Unix / Linux), e para o desenvolvimento, implantação e manutenção de aplicações pode ser feito através de uma interface web local ou remoto. web2py executado com CPython (a implementação C) e Jython (implementação Java), versões 2.4, 2.5 e 2.6, embora o apoio apenas 2,5 "officially" mais não podemos garantir a compatibilidade de aplicações. web2py fornece um sistema de bilhética. Se ocorrer um erro, um bilhete é emitido ao usuário, eo erro é registrada para o administrador. web2py é open source e liberado sob a licença GPL2.0, mas os aplicativos desenvolvidos com web2py não estão sujeitas a qualquer restrição de licença. Na verdade, desde que não contêm o código-fonte web2py, eles não são considerados "derivative work". web2py também permite ao desenvolvedor bytecode-compilar aplicações e distribuí-los como de código fechado, mas eles vão exigir web2py para ser executado. web2py A licença inclui uma exceção que permite que os desenvolvedores web para enviar seus produtos com o original compilado web2py binários pré-, sem o acompanhamento código-fonte.

Outra característica do web2py, é que nós, seus colaboradores, nos comprometemos a manter a compatibilidade com versões anteriores em versões futuras. Fizemo-lo desde o primeiro lançamento de web2py em outubro de 2007 As novas funcionalidades. Foram adicionados e bugs foram corrigidos, mas se um programa trabalhou com web2py 1.0, esse programa ainda vai trabalhar hoje.

Aqui estão alguns exemplos de declarações web2py que ilustram o seu poder

e simplicidade. O código a seguir:

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

cria um tabela de banco de dados chamado algo, que "person" with two fields: "name", a string; and "image" deve ser carregado (imagem real). Se a tabela já existe, mas não coincide com esta definição, é alterado de forma adequada.

Dada a tabela definida acima, o seguinte código:

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

cria uma forma de inserção deste quadro que permite aos usuários fazer upload de imagens. Ele também valida um formulário apresentado, renomeia o upload da imagem de uma forma segura, armazena a imagem em um arquivo, insere o registro correspondente no banco de dados impede a apresentação de casal e, eventualmente, modifica a própria forma, adicionando mensagens de erro se os dados apresentados pela o usuário não passar na validação.

O código a seguir:

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

impede que os visitantes acessem o `f` função a não ser que o visitante é membro de um grupo cujos membros têm permissões para "read" records of table "person". Se o visitante não está logado, ele se dirigiu a uma página de login (desde que por padrão web2py).

O código a seguir incorpora um componente da página.

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

Permite web2py para carregar em vista o conteúdo gerado pela função `other_controller` (funciona com qualquer função). Ele carrega o conteúdo através de Ajax,

incorpora-lo na página atual (com o layout atual e não o layout da função other_controller), armadilhas de todas as formas contidas no conteúdo carregado para que eles também são apresentados através de Ajax sem recarregar a página. Ele também pode carregar o conteúdo de aplicativos não-web2py.

O ajudante de carga permite que muito design modular de aplicações e é discutido com algum detalhe, no último capítulo deste livro.

1.1 *Princípios*

programação Python geralmente segue esses princípios básicos:

- Não se repita (DRY).
- Deve haver apenas uma maneira de fazer as coisas.
- Explícito é melhor que implícito.

web2py completamente abraça os dois primeiros princípios, forçando o desenvolvedor a utilizar boas práticas de engenharia de software que desencorajam a repetição de código. Guias web2py o desenvolvedor através de quase todas as tarefas comuns no desenvolvimento de aplicações web (criação e processamento de formulários, as sessões de gestão, cookies, erros, etc). web2py difere de outros quadros no que respeita ao terceiro princípio, que por vezes entra em conflito com os outros dois. Em particular, não importa web2py aplicações do usuário, mas executa-los em um contexto pré-definido. Neste contexto, expõe as palavras-chave Python, assim como as palavras-chave web2py.

Para alguns isso pode parecer mágica, mas não deveria. Na prática, apenas alguns módulos já são importados sem ter que fazer isso. Web2py está tentando evitar a característica de irritar outros quadros que força o desenvolvedor para importar os mesmos módulos no topo da cada modelo e contro-

lador. web2py, importando seus próprios módulos, economiza tempo e evita erros, seguindo assim o espírito do "don't repeat yourself" e "there should be only one way of doing things".

Se o desenvolvedor desejar utilizar Python outros módulos de terceiros ou módulos, os módulos devem ser importados de forma explícita, como em qualquer outro programa Python.

1.2 *Web Frameworks*

Em seu nível mais fundamental, uma aplicação Web consiste em um conjunto de programas (ou funções) que são executados quando o URL correspondente é visitado. A saída do programa é retornado para o visitante e processado pelo browser.

O objetivo do frameworks web é permitir que desenvolvedores criem novos app rápida, fácil e sem fazer erros. Isto é feito através da comprovação APIs e ferramentas que simplificam e reduzem a quantidade de codificação que é exigido.

As duas abordagens clássicas para o desenvolvimento de aplicativos web são:

- Gerando HTML (14; 15) programação.
- Incorporação de código em páginas HTML.

O primeiro modelo é o seguido, por exemplo, scripts CGI cedo. O segundo modelo é seguido, por exemplo, PHP (16) (onde o código está em PHP, a linguagem C-like), ASP (onde o código está em Visual Basic), e JSP (onde o código está em Java).

Aqui nós apresentamos um exemplo de um programa PHP que, quando executado, recupera dados de um banco de dados e retorna uma página

HTML que mostram os registros selecionados:

```
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>
```

O problema dessa abordagem é que o código é incorporado em HTML, mas este mesmo código também precisa de muito para gerar HTML adicionais e gerar instruções SQL para consulta o banco de dados, de enredar várias camadas de aplicação, tornando difícil de ler e manter. A situação é ainda pior para as aplicações Ajax, ea complexidade aumenta com o número de páginas (processos) que compõem o aplicativo.

A funcionalidade do exemplo acima pode ser expresso em web2py com duas linhas de código Python:

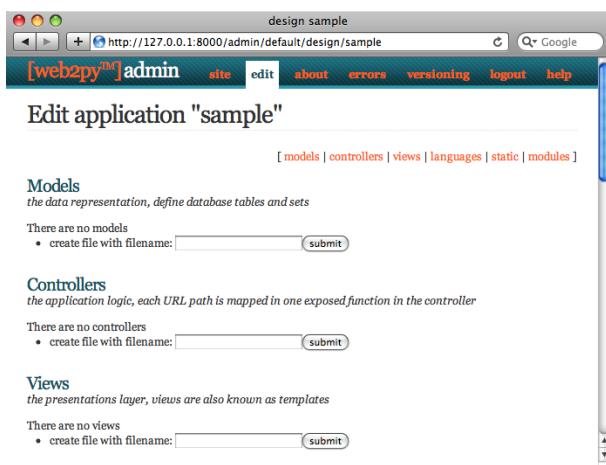
```
1 def index():
2   return HTML(BODY(H1('Records'), db().select(db.contacts.ALL)))
```

Neste exemplo simples, a estrutura da página HTML é representado programmaticamente pela, HTML, BODY e H1 objetos, o db banco de dados é consultado pelo comando select, enfim, tudo é serializada em HTML. Observe que db não é uma palavra-chave, mas um usuário variável definida. Nós vamos usar este nome sempre se referir a uma conexão de dados para evitar confusões. frameworks web são geralmente classificadas como de dois tipos: um quadro "glued" é construído pela montagem (colando) vários componentes de terceiros. Um quadro "full-stack" é construído através da criação de componentes projetados especificamente para funcionar em conjunto e ser integrado. web2py é um framework full-stack. Quase todos os seus compo-

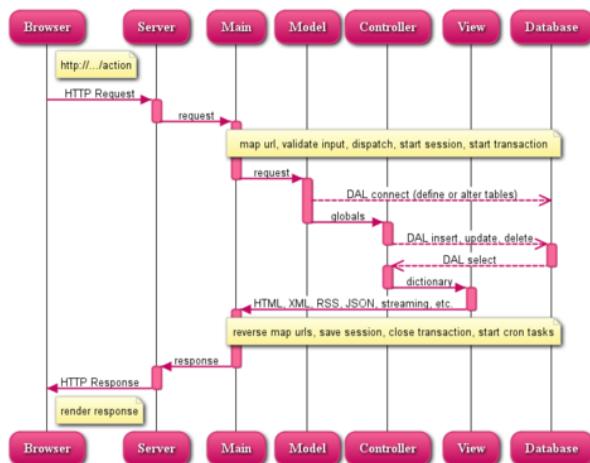
nentes são construídos a partir do zero e projetados para trabalhar juntos, mas eles funcionam tão bem fora do quadro web2py completo. Por exemplo, o banco de dados Abstraction Layer (DAL) ou a linguagem de template pode ser usado independentemente do quadro web2py importando gluon.sql ou gluon.template na sua Python próprios aplicativos. gluon é o nome da pasta que contém web2py bibliotecas do sistema. Algumas bibliotecas web2py, como a construção e processamento de formulários a partir de tabelas de banco de dados, têm dependências em outras partes do web2py. web2py também pode trabalhar com outros fabricantes Python bibliotecas terceiros, incluindo outras línguas e Dals modelo, mas eles não serão tão bem integrados como os componentes originais.

1.3 Model-View-Controller

forças web2py o desenvolvedor para representação de dados separados (o modelo), apresentação de dados (a vista) eo fluxo de trabalho do aplicativo (o controlador). Vamos considerar novamente o exemplo anterior e veja como construir uma aplicação web2py em torno dele. Aqui está um exemplo do MVC web2py interface de edição:



O fluxo de trabalho típico de um pedido web2py é descrito no diagrama abaixo:



No diagrama:

- O servidor pode ser o web2py built-in servidor Web ou um servidor de terceiros, tais como o Apache. O servidor gerencia multi-threading.
- "main" é a aplicação WSGI principal. Ele executa todas as tarefas comuns e envolve aplicações do usuário. Trata-se de cookies, sessões, transações, url de encaminhamento e roteamento reverso, a expedição. Pode servir e transmitir arquivos estáticos se o servidor web não é fazê-lo já.
- O Models, Views e componentes Controller compõem o aplicativo do usuário.
- Pode haver vários aplicativos hospedados na instância web2py mesmo.
- As setas tracejadas representam a comunicação com o mecanismo de banco de dados (ou de motores). As consultas de banco de dados podem ser escritos em SQL-primas (desanimado) ou usando o web2py Database Abstraction Layer (recomendado), de modo que o código do aplicativo web2py não é dependente do mecanismo de banco de dados específico.

- Os mapas expedidor, a URL solicitada em uma chamada de função no controlador. A saída da função pode ser uma corda ou um dicionário de símbolos (a tabela de hash). Os dados do dicionário é processado por um ponto de vista. Se os pedidos visitante de uma página HTML (o padrão), o dicionário é processado em uma página HTML. Se o visitante pede a mesma página em XML, web2py tenta encontrar uma exibição que podem tornar o dicionário em XML. O desenvolvedor pode criar vistas a tornar as páginas em qualquer um dos protocolos já apoiados (HTML, XML, JSON, RSS, CSV, RTF) ou protocolos adicionais personalizados.
- Todas as chamadas são envolvidos em uma transação, e qualquer exceção não capturada faz com que a operação de roll back. Se a solicitação for bem-sucedido, a transação é confirmada.
- web2py também lida com sessões e cookies sessão automaticamente, e quando uma transação for confirmada, a sessão também é armazenado, a menos que especificado em contrário.
- É possível registrar tarefas recorrentes (cron) para executar em horários programados e / ou após a realização de determinadas ações. Desta forma é possível a longo prazo e de computação intensiva tarefas em segundo plano, sem retardar a navegação.

Aqui está uma mínima e completa aplicação MVC consiste em três arquivos:

"db.py" é o modelo:

```

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

Ele se conecta ao banco de dados (neste exemplo um banco de dados SQLite armazenadas no arquivo storage.sqlite) e define uma tabela chamada contacts. Se a tabela não existir, cria web2py e, de forma transparente e no fundo, gera o código SQL no SQL dialeto adequadas para o mecanismo de banco de dados específico utilizado. O desenvolvedor pode ver o SQL gerado, mas não precisa alterar o código se o banco de dados back-end, que tem como

padrão SQLite, passa a ter com o MySQL, PostgreSQL, MSSQL, Firebird, Oracle, DB2, Informix, Interbase, Ingres, ou Big Google Tabelas no Google App Engine.

Depois que uma tabela está definido e criado, web2py também gera um plenamente funcional de base de dados da administração interface web para acessar o banco de dados e tabelas. É chamado **appadmin**.

"default.py" é o controlador:

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

Em web2py, URLs são mapeados para módulos Python e chamadas de função. Neste caso, o controlador possui uma função única (ou "action") `contacts` chamado. Uma ação pode retornar uma string (o site voltou) ou um dicionário Python (um conjunto de pares `key:value`). Se a função retorna um dicionário, ele é passado para uma exibição com o mesmo nome do controller / função que, por sua vez torna-la. Neste exemplo, a `contacts select` executa uma função de banco de dados e retorna os registros resultantes de um valor associado à chave `records` dicionário.

"default/contacts.html" é o ponto de vista:

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

Esta visão é chamado automaticamente pelo web2py após a função do controlador associado (ação) é executado. O objetivo desta visão é a de tornar as variáveis do dicionário `records=...` voltou em HTML. Visualização do arquivo é escrito em HTML, mas incorpora código Python delimitado pelo `{{ e }}` delimitadores especiais. Isto é muito diferente do exemplo de código PHP, pois o único código embutido dentro do HTML é um arquivo "presentation

layer" code. The "layout.html" referenciado no topo da janela é fornecido pelo web2py e constitui o layout básico para todas as aplicações web2py. O arquivo de layout facilmente pode ser modificado ou substituído.

1.4 Por web2py

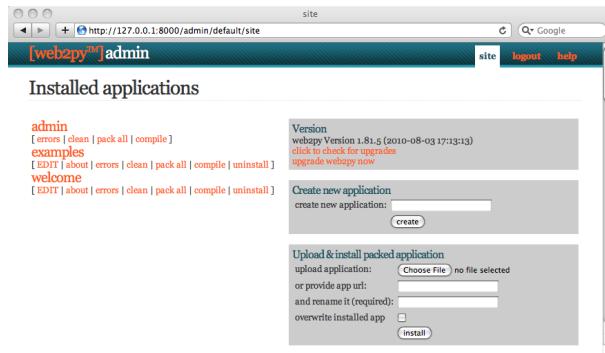
web2py é uma aplicação de muitos frameworks web, mas tem características únicas e atraentes. web2py foi originalmente desenvolvido como uma ferramenta de ensino, com a principal motivação que se segue:

- Fácil para os usuários aprendem lado do servidor para o desenvolvimento web sem comprometer a funcionalidade. Por esta razão web2py não requer instalação, sem configuração, não tem dependências (exceto para a distribuição do código fonte, que exige o Python 2.5 (ou posterior 2.x) e seus módulos da biblioteca padrão), e expõe a maioria das suas funcionalidades através de uma interface web.
- web2py tem-se mantido estável desde o primeiro dia, porque segue um design top-down, ou seja, sua API foi projetada antes de ser implementado. Mesmo com a nova funcionalidade foi adicionada, web2py nunca quebrou compatibilidade com versões anteriores, e que não irá quebrar a compatibilidade quando a funcionalidade adicional é adicionada no futuro.
- web2py proativamente aborda as questões de segurança mais importantes que afligem muitas aplicações modernas da web, como determinado pela OWASP (19) abaixo.
- web2py é leve. Seus principais bibliotecas, incluindo a camada de abstração de banco de dados, o modelo de linguagem e toda a quantidade de ajudantes 300KB. Todo o código fonte, incluindo aplicativos de exemplo e os montantes imagens 2.0MB.
- web2py tem uma pegada pequena e é muito rápido. Ele usa o servidor web Rocket (22) WSGI desenvolvido por Timothy Farrell. É 30% mais rápido que o Apache com mod_proxy. Nossos testes também indicam

que, em um PC médio, que serve uma média de página dinâmica sem acesso ao banco em cerca de 10ms. A DAL tem overhead muito baixos, geralmente inferior a 3%.

WSGI (17; 18) está para Web Server Gateway Interface. É um novo padrão de Python para comunicação entre um servidor web e aplicações Python).

Aqui está um screenshot da interface principal web2py **admin**:



1.5 Segurança

O Open Web Application Security Project (19) (OWASP) é uma comunidade mundial aberta e livre focado na melhoria da segurança do software aplicativo.

OWASP listou o top dez questões de segurança de aplicações web que colocam em risco. Esta lista é reproduzida aqui, juntamente com uma descrição de como cada tema é 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 padrão, todas as variáveis escapa prestados na vista, evitando 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." inclui uma camada de abstração de banco de dados que torna impossível a injeção de SQL. Normalmente, as instruções SQL não são escritas pelo desenvolvedor. Em vez disso, o SQL é gerado dinamicamente pelo DAL, garantindo que todos os dados inseridos corretamente escapou.

- 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." só permite que as funções expostas a ser executado, impedindo que arquivos maliciosos funções executivas Importado nunca são expostos; ações só estão expostos. web2py interface de administração baseada na web torna muito fácil de acompanhar o que está exposto e que não é.

- 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. Attack-

ers can manipulate those references to access other objects without authorization." não expor os objetos internos, além disso, web2py valida todas as URLs, evitando ataques de passagem de diretório. web2py também fornece um mecanismo simples para criar formulários, que automaticamente validar todos os 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 impede CSRF, bem como a apresentação dupla acidental de formas através da atribuição de um símbolo de um tempo aleatório para cada formulário. Além disso web2py usa UUID de cookie de sessão.

- web2py

"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." inclui um sistema de bilhética. Nenhum erro pode resultar em código que está sendo exposto aos usuários. Todos os erros são registrados e um bilhete é emitido ao usuário que permite o rastreamento de erro. Erros e código fonte é acessível somente para o 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 aplicações, inclui um poderoso

Role Based Access Control 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." usa o MD5 ou o HMAC + algoritmos de hash SHA-512 para proteger as senhas armazenadas. Outros algoritmos também estão disponíveis.

- web2py

"Insecure Communications: Applications frequently fail to encrypt network traffic when it is necessary to protect sensitive communications." inclui o SSL-enabled (21) Rocket servidor WSGI, mas também pode usar o Apache ou Lighttpd e mod_ssl para fornecer criptografia SSL de comunicações.

-

"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 solicitações de URL para módulos Python e funções. web2py fornece um mecanismo para declarar que são públicos e as funções que exigem autenticação e autorização. O incluídos Role Based Access Control API permite que os desenvolvedores para restringir o acesso a qualquer função com base em permissões de login, os membros do grupo ou grupo baseado. As permissões são muito granulares e podem ser combinadas com CRUD que permita, por exemplo, para dar acesso a tabelas específicas e / ou registros. web2py foi revisto para a segurança e você pode encontrar o resultado da revisão em ref. (20).

1.6 Na caixa

Você pode baixar web2py a partir do site oficial:

¹ <http://www.web2py.com>

web2py é composto dos seguintes componentes:

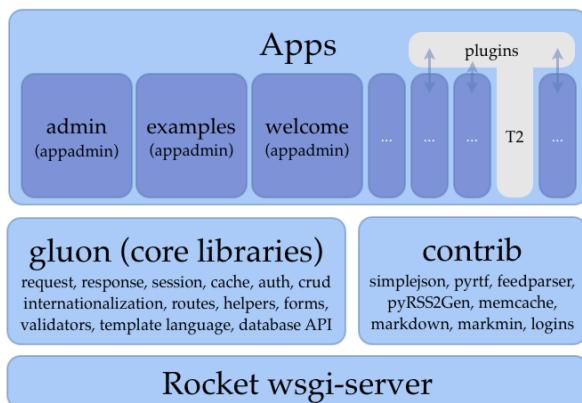
- **libraries**: fornecer funcionalidade central do web2py e são acessíveis através de programação.
- **web server**: o servidor web Rocket WSGI.
- a aplicação **admin**: usado para criar, projetar e gerenciar outras aplicações web2py. **admin** fornecer uma solução completa baseada na Web Integrated Development Environment (IDE) para construir aplicações web2py. Ele também inclui outras funcionalidades, tais como o teste baseado na web e um escudo baseado na web.
- a aplicação **examples**: contém documentação e exemplos interativos. **examples** é um clone do site oficial web2py, e inclui documentação epydoc ea Esfinge.
- a aplicação **welcome**: o plano básico de modelo para qualquer outra aplicação. Por padrão, ele inclui um menu em cascata CSS puro e autenticação de usuário (discutido no Capítulo 8).

web2py é distribuído em código fonte e binário para o Microsoft Windows e Mac OS X.

A distribuição do código-fonte pode ser usado em qualquer plataforma onde o Python executa e inclui os componentes acima mencionados. Para executar o código-fonte, você precisa Python 2.5 pré-instalado no sistema. Você também precisa de um banco de dados dos motores de suporte instalado. Para os testes e aplicações de procura de luz, você pode usar o banco de dados SQLite, incluído com o Python 2.5.

As versões binárias de web2py (para Windows e Mac OS X) incluem um interpretador Python 2.5 e banco de dados SQLite. Tecnicamente, estes dois não são componentes do web2py. Incluí-los na distribuição binária permite executar web2py fora da caixa.

A imagem seguinte mostra a estrutura geral web2py:



1.7 Licença

web2py está licenciado sob a licença GPL versão 2. O texto integral da licença se disponível em ref. (31).

A licença inclui mas não está limitado aos seguintes artigos:

1. Você pode copiar e distribuir cópias fiéis do código-fonte do Programa ao recebê-lo, em qualquer meio, desde que você conspícuo e apropriadamente publique em cada cópia um aviso de copyright apropriada e exoneração de garantia; manter intactos todos os avisos que se referem a esta Licença e à ausência de qualquer garantia e forneça a outros recebedores do Programa uma cópia desta Licença junto com o Programa.

[...]

4. Você não pode copiar, modificar, sublicenciar ou distribuir o Programa, exceto conforme expressamente estabelecido nesta Licença. Qualquer outra tentativa de copiar, modificar, sublicenciar ou distribuir o Programa é nula, e automaticamente rescindirá seus direitos sob esta Licença.

[...]

11. Porque o programa é licenciado livre de encargos, não existe qualquer garantia para o programa, na LIMITES pela legislação aplicável. Salvo disposição em contrário indicada por escrito THE detentores de copyright e / ou outras partes disponibilizam o programa "AS IS" SEM GARANTIA DE QUALQUER TIPO, EXPRESSA OU IMPLÍCITA, INCLUINDO, MAS NÃO SE LIMITANDO ÀS GARANTIAS IMPLÍCITAS DE COMERCIALIZAÇÃO E ADEQUAÇÃO PARA UM PROPÓSITO PARTICULAR. Todo o risco quanto à qualidade e PERFORMANCE DO PROGRAMA é com você. Se o programa de provar defeituoso, você assume o custo de todo serviço, reparo ou correção.

12. EM HIPÓTESE ALGUMA A menos que exigido pela lei aplicável ou acordado por escrito qualquer detentor, ou qualquer outra entidade que possa modificar e / ou redistribuir o programa conforme permitido acima, será responsável por danos, incluindo quaisquer prejuízos gerais, especiais, incidentais ou DANOS EMERGENTES decorrentes do uso ou incapacidade de usar o programa (INCLUINDO MAS NÃO SE LIMITANDO a perda de dados ou dados que sejam tornados inexatos ou perdas sustentadas por você ou por terceiros ou uma falha do programa em operar com qualquer outro programa), mesmo se tal detentor ou outra parte SIDO AVISADO DA POSSIBILIDADE DE TAIS DANOS. web2py inclui alguns códigos de terceiros (por exemplo, o interpretador Python, o servidor web Rocket, e algumas bibliotecas JavaScript). Seus respectivos autores e certificados são reconhecidos no website (1) oficiais e no próprio código.

As aplicações desenvolvidas com web2py, contanto que não incluem o código

fonte web2py, não são consideradas obras derivadas. Isso significa que eles não são obrigados pela licença GPLv2, e você pode distribuir os aplicativos desenvolvidos no âmbito de qualquer licença que você escolher, incluindo um código fechado e / ou licença comercial.

License Commercial Exception

A licença também inclui web2py uma exceção comercial:

Você pode distribuir um aplicativo desenvolvido com web2py juntamente com um funcionário sem modificações distribuição binária do web2py, como descarregado a partir do website (1) oficial, contanto que você deixar claro na carta de seu aplicativo que arquivos pertencem a aplicação e os arquivos que pertencem a web2py.

1.8 Agradecimentos

web2py foi originalmente desenvolvido por direitos autorais e por Massimo Di Pierro. A primeira versão (1.0) foi lançado em outubro de 2007. Desde então, tem sido adotado por muitos usuários, alguns dos quais também contribuíram com relatórios de bugs, testes, depuração, patches e revisão deste livro.

Alguns dos principais contribuintes são, em ordem alfabética pelo primeiro nome:

Alexey Nezhdanov, Álvaro Justen, Csipa Attila, Ferrett Bill, Boris Brian, Meredyk Manojlovic, Bruno Rocha, Haese Carsten, Chris Clark, Chris Steel, Christian Howes Foster, Christopher Smiga, Lazell CJ, Younkins Craig, Daniel Lin, 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. v. Stockhausen, Geiser Reinhart Ian, Benn Jonathan, Jonathan Lundell, Goldfoot Josh, Jose Jachuf, Yang Keith,

Kyle Smith, Limodou, Leuthi Marcel, Marcello Della Longa, Reingart Mariano, Larsen Mark, Mark Moore, Gritsch Markus, Hufsky Martin, Banach Mateusz, Willis Michael, Comitini Michele, Freeze Nathan, Sweeny Niall, Niccolo Polo, Nicolas bruxer, Ondrej Tais Pai, Phy Arkar Lwin, Bhattacharyya Robin, Ruijun Luo (Iceberg), Scott Roberts, Sergey Podlesnyi, Aina Shariff, Sriram, Durbha, Hankins Sterling, Rackham Stuart, Yusupov Telman, Burgess Thadeus, Michelsen Tim, Timothy Farrell, Eshel Yair, Yarko Tymciurak, Jo Younghyun, Zahariash.

Estou certo de que esqueci alguém, por isso peço desculpa.

Agradeço particularmente Jonathan, Iceberg, Nathan, e Thadeus de suas grandes contribuições para web2py e Alvaro, Bruno, Denes, Felipe, Graham, Jonathan, Hans, Kyle, Marcos, Michele, Richard, Robin, Roman, Scott Shane, Sharriff, Sriram, Sterling, Stuart, Thadeus e Yarko para corrigir vários capítulos deste livro. A sua contribuição foi inestimável. Se você encontrar algum erro neste livro, são exclusivamente culpa minha, provavelmente introduzido por uma edição de última hora. Agradeço, também, Ryan Steffen de Wiley Custom Learning Solutions para ajudar com a publicação da primeira edição deste livro. web2py contém o código dos seguintes autores, a quem eu gostaria de agradecer:

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

O logotipo utilizado na capa deste livro foi desenhado por Peter Kirchner em jovens designers.

Agradeço Helmut Epp (reitor da DePaul University), David Miller (decano da Faculdade de Informática e Mídia Digital da DePaul University), e Estia

Eichten (Membro do MetaCryption LLC), pela confiança e apoio contínuo.

Por último, gostaria de agradecer à minha esposa, Claudia, e meu filho, Marco, por me aturar durante as muitas horas que passei em desenvolvimento web2py, trocar e-mails com usuários e colaboradores, e escrever este livro. Este livro é dedicado a eles.

1.9 *Sobre este livro*

Este livro inclui os seguintes capítulos, além desta introdução:

- O capítulo 2 é uma introdução minimalista para Python. Ele pressupõe o conhecimento de ambos os processual e orientada a objetos conceitos de programação como loops, condições, chamadas de funções e classes, e cobre sintaxe Python básica. Cobre também exemplos de módulos Python que são usados durante todo o livro. Se você já sabe Python, você pode pular o capítulo 2.
- Capítulo 3 mostra como começar a web2py, discute a interface administrativa, e orienta o leitor através de vários exemplos de complexidade crescente: um aplicativo que retorna um string, um pedido de balcão, um blog de imagens, e um aplicativo de wiki plena soprado que permite uploads de imagens e comentários, fornece autenticação, autorização, os serviços web e um feed RSS. Ao ler este capítulo, você pode precisar consultar o Capítulo 2 para a sintaxe Python em geral e para os capítulos seguintes para uma referência mais detalhada sobre as funções que são usadas.
- Capítulo 4 aborda de forma mais sistemática a estrutura do núcleo e bibliotecas: mapeamento de URL, o pedido, a resposta, sessões cacheint, CRON, internacionalização e fluxo de trabalho em geral.
- O capítulo 5 é uma referência para o modelo de linguagem utilizada para construir pontos de vista. Ele mostra como inserir código Python no HTML, e demonstra o uso de auxiliares (objetos que podem gerar o HTML).

- Capítulo 6 cobre a camada de abstração de banco de dados, ou DAL. A sintaxe da DAL é apresentado através de uma série de exemplos.
- O Capítulo 7 aborda formas, validação de formulário e processamento de formulários. Forma é o ajudante de baixo nível para a construção de formulário. SQLFORM é o construtor do formulário de alto nível. No Capítulo 7, discutir também a criar novos / Read / Update / Delete (CRUD) API.
- Capítulo 8 cobre autenticação, autorização e extensível Role-Based Access Control mecanismo disponível no web2py. Mail configuração e CAPTCHA também são discutidas aqui, pois eles são usados por autenticação. Na terceira edição do livro, nós adicionamos uma cobertura abrangente de integração com os mecanismos de autenticação de terceiros, tais como OpenID, OAuth, Google, Facebook, Linkedin, etc
- O capítulo 9 é sobre como criar serviços Web em web2py. Nós fornecemos exemplos de integração com o Google Web Toolkit através de Pijama, e com o Adobe Flash via PyAMF.
- O capítulo 10 é sobre web2py jQuery e receitas. web2py é projetado principalmente para a programação do lado do servidor, mas inclui jQuery, uma vez que temos encontrado para ser a melhor biblioteca JavaScript open-source disponível para efeitos e Ajax. Neste capítulo, discutiremos como utilizar eficazmente com jQuery web2py.
- O capítulo 11 é sobre implantação da produção de aplicações web2py. Estamos principalmente face a três cenários de produção possíveis: em um servidor web Linux e um conjunto de servidores (que consideramos as alternativas de implantação principal), funcionando como um serviço em um ambiente Microsoft Windows, e implantação de aplicativos do Google o Engine (GAE). Neste capítulo, vamos discutir também questões de segurança e escalabilidade.
- Capítulo 12 contém uma variedade de outras receitas para resolver tarefas específicas, incluindo atualizações, geocodificação, paginação, Twitter API, e muito mais.
- Capítulo 13 foi introduzido na terceira edição do livro e discute web2py componentes e plugins, como meio para construir aplicações modulares.

Nós fornecemos um exemplo de um plugin que implementa a funcionalidade de muitos comumente utilizados, tais como gráficos, comentários, tags e páginas wiki.

Este livro abrange apenas as funcionalidades básicas web2py ea API que vem com web2py. Este livro não abrange os aparelhos web2py (ou seja, pronto pedidos apresentados).

Você pode baixar os aparelhos web2py do (34) correspondente web site.

Você pode encontrar outros temas discutidos no AlterEgo (35), o interativo web2py FAQ.

1.10 *Elements of Style*

Ref. (36) contém práticas de bom estilo durante a programação com Python. Você vai descobrir que web2py nem sempre segue essas regras. Isso não é por omissão ou negligência, é nossa convicção que os usuários de web2py devem seguir estas regras e nós incentivamos isso. Optamos por não seguir algumas dessas normas quando da definição auxiliar web2py objetos, a fim de minimizar a probabilidade de conflito de nome com objetos definidos pelo usuário.

Por exemplo, a classe que representa um `<div>` DIV é chamado, embora de acordo com a referência de estilo Python que devia ter sido chamado `Div`. Acreditamos que, para este exemplo específico que o uso de um superior "DIV" caso tudo é uma escolha mais natural. Além disso, esta abordagem deixa programadores criar uma classe chamada "Div", se optar por fazê-lo. Nosso sintaxe também mapas naturalmente na notação DOM da maioria dos navegadores (incluindo, por exemplo, o Firefox).

De acordo com o guia de estilo Python, tudo em maiúsculas strings devem ser utilizados para constantes e não variáveis. Continuando com nosso ex-

emplo, mesmo considerando que a `DIV` é uma classe, é uma classe especial que nunca deve ser modificado pelo usuário, pois isso iria quebrar outras aplicações web2py. Assim, acreditamos que isso qualifica a classe `DIV` como algo que deve ser tratado como uma constante, mas que justificam a nossa escolha de notação.

Em resumo, as seguintes convenções são seguidas:

- ajudantes HTML e validadores estão todas as letras maiúsculas para as razões discutidas acima (por exemplo, `DIV`, `A`, `FORM`, `URL`).
- O `T` é objeto tradutor maiúsculas apesar do fato de que é uma instância de uma classe e não uma classe em si. Logicamente, o objeto tradutor realiza uma ação semelhante ao HTML ajudantes, ela afeta processamento da apresentação. Além disso, precisa `T` para ser fácil de localizar o código e deve ter um nome curto.
- classes DAL seguir o guia de estilo Python (primeira letra maiúscula), por exemplo, `Table`, `Field`, `Query`, `Row`, `Rows`, etc

Nos restantes casos, acreditamos que temos seguido, tanto quanto possível, o Python Style Guide (PEP8). Por exemplo, todas as instâncias de objetos são minúsculas (`request`, `response`, `session`, `cache`), e todas as classes internas são capitalizados.

Em todos os exemplos deste livro, palavras-chave web2py são exibidas em negrito, enquanto cordas e os comentários são mostrados em itálico.

2

A linguagem Python

2.1 Sobre Python

Python é um propósito geral e de programação de nível muito elevado de línguas. Sua filosofia de design realça a produtividade do programador e legibilidade do código. Ele tem uma sintaxe núcleo minimalista, com poucos comandos básicos e semântica simples, mas também tem uma biblioteca grande e abrangente padrão, incluindo uma Application Programming Interface (API) para muitos dos subjacentes do sistema operacional (SO) funções. O código Python, enquanto a minimalist, define objetos, como listas ligadas (`list`), tuplas (`tuple`), tabelas de hash (`dict`), e números arbitrariamente longo (`long`).

Python suporta vários paradigmas de programação. Estes são orientados a objeto (`class`), imperativo (`def`) e funcional (`lambda`) de programação. Python possui um tipo de sistema de gerenciamento de memória dinâmica e automática através de contagem de referência (similar a Perl, Ruby, e Scheme).

Python foi lançada por Guido van Rossum em 1991. A linguagem tem um diálogo aberto, modelo de desenvolvimento baseado na comunidade gerido

pela organização sem fins lucrativos Python Software Foundation. Há muitos intérpretes e compiladores que implementam a linguagem Python, incluindo uma em Java (Jython), mas, neste breve resumo, nós nos referimos à implementação de referência C criado por Guido.

Você pode encontrar muitos tutoriais, a documentação oficial e referências de biblioteca da linguagem Python no oficial website (2)

Para referências adicionais Python, podemos recomendar os livros ref. (37) e ref. (38).

Você pode pular este capítulo se você já está familiarizado com a linguagem Python.

2.2 *Arranque*

A distribuição binária do web2py para o Microsoft Windows ou OS X da Apple vêm embalados com o interpretador Python embutido na distribuição de arquivos em si.

Você pode iniciá-lo no Windows com o seguinte comando (digite no prompt do DOS):

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

No OS X Apple, indicar o tipo seguinte comando em uma janela de terminal (assumindo que está na mesma pasta como web2py.app):

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

Em um sistema Linux ou outro Unix, as chances são que você tem o Python já está instalado. Se sim, em um prompt tipo concha:

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

Se você não tiver o Python 2.5 (ou posterior 2.x) já estiver instalado, você terá que fazer o download e instalá-lo antes de executar web2py.

A opção de linha de comando instrui o -S welcome web2py para executar o shell interativo, como se os comandos foram executados em um controlador para a aplicação **welcome**, o pedido de andaimes web2py. Isso expõe quase todas as classes web2py, objetos e funções para você. Esta é a única diferença entre a linha de comando web2py interativo, e na linha de comando normal Python.

A interface de administração também fornece um shell baseado na web para cada aplicação. Você pode acessar o aplicativo para o "welcome" no.

```
1 http://127.0.0.1:8000/admin/shell/index/welcome
```

Você pode experimentar todos os exemplos neste capítulo com o shell normal ou o shell web-based.

2.3 help, dir

A linguagem Python fornece dois comandos para obter a documentação sobre os objetos definidos no escopo atual, ambas built-in e definidas pelo usuário.

Podemos pedir para **help** sobre um objeto, por exemplo "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
```

56 WEB2PY MANUAL 3RD EDITION ALPHA PORTUGUESE (MACHINE TRANSLATED)

```
12 | will be returned instead.  
13 |  
14 | Methods defined here:  
15 |  
16 |     __abs__(...)  
17 |         x.__abs__() <==> abs(x)  
18 ...
```

e, desde "1" é um inteiro, podemos obter uma descrição sobre a classe `int` e todos os seus métodos. Aqui, a saída foi truncado porque é muito longo e detalhado.

Da mesma forma, podemos obter uma lista de métodos do objeto "1" `dir` com o comando:

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

2.4 Tipos

Python é uma datilografados linguagem dinâmica, o que significa que variáveis não têm um tipo e, portanto, não têm de ser declaradas. Valores, por outro lado, têm um tipo. Você pode consultar uma variável para o tipo de valor que ela contém:

```
>>> a = 3  
>>> print type(a)  
<type 'int'>  
>>> a = 3.14  
>>> print type(a)  
<type 'float'>
```

```

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

```

Python também inclui, nativamente, estruturas de dados como listas e dicionários.

2.4.1 str

Python suporta a utilização de dois diferentes tipos de seqüências: seqüências ASCII e Unicode strings. strings ASCII são delimitados por '...', "... or by "..." or """...""". Cotações Triple delimitar strings de várias linhas. strings Unicode começar com um u seguido pelo string contendo caracteres Unicode. Uma seqüência de caracteres Unicode pode ser convertida em uma seqüência de caracteres ASCII, escolhendo uma codificação, por exemplo:

```

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

```

Depois de executar estes três comandos, o a resultante é uma seqüência de caracteres ASCII armazenar caracteres codificados com UTF8. Pelo projeto, web2py usa UTF8 strings codificadas internamente.

Também é possível escrever as variáveis em seqüências de várias maneiras:

```

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

```

A última anotação é mais explícita e menos propenso a erros, e deve ser preferido.

Muitos objetos Python, por exemplo, números, pode ser serializado em strings usando o `str` ou `repr`. Estes dois comandos são muito semelhantes, mas ligeiramente diferente. Por exemplo:

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

Para as classes definidas pelo usuário, `str` e `repr` pode ser definido / redefinido usando os operadores especiais `__str__` e `__repr__`. Estes são brevemente descritos mais tarde, para mais informações, consulte o oficial Python documentation (39). `repr` tem sempre um valor padrão.

Outra característica importante de uma seqüência de Python é que, como uma lista, é um objeto iteráveis.

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

2.4.2 list

Os principais métodos de uma lista de Python são anexar, inserir e excluir:

```
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
```

As listas podem ser cortados:

```

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

e concatenadas:

```

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

A lista é iteráveis, você pode fazer um loop sobre ele:

```

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

Os elementos de uma lista não tem que ser do mesmo tipo, pois eles podem ser qualquer tipo de objeto Python.

2.4.3 tuple

Uma tupla é como uma lista, mas o seu tamanho e os elementos são imutáveis, enquanto que em uma lista que eles são mutáveis. Se um elemento de tupla é um objeto, os atributos de objeto são mutáveis. Uma tupla é delimitado por parênteses.

```

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

Assim, enquanto esta trabalha para a lista:

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

a atribuição elemento não funciona para uma 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
```

A tupla, como a lista é um objeto iteráveis. Observe que uma tupla que consiste de um elemento único deve incluir uma vírgula à direita, como mostrado abaixo:

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

Tuplas são muito úteis para um empacotamento eficiente de objetos por causa da sua imutabilidade, e os colchetes são frequentemente opcional:

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

2.4.4 dict

A Python dict-ionary é uma tabela hash que mapeia um objeto chave a um objeto de valor. Por exemplo:

```

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

```

As chaves podem ser de qualquer tipo hashable (int, string, ou qualquer objeto cuja classe implementa o método `__hash__`). Os valores podem ser de qualquer tipo. Chaves e valores diferentes no mesmo dicionário não tem que ser do mesmo tipo. Se as chaves são caracteres alfanuméricos, o dicionário também pode ser declarado com a sintaxe alternativa:

```

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

```

Métodos são úteis `has_key`, `keys`, `values` e `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)]

```

O método `items` produz uma lista de tuplas, cada uma contendo uma chave e seu valor associado.

Dicionário elementos e os elementos da lista podem ser apagados com o 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, o Python usa o operador hash para converter objetos em números inteiros, e usa esse inteiro para determinar onde armazenar o valor.

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

2.5 Sobre recuo

Python usa indentação para delimitar blocos de código. Um bloco começa com uma linha que termina em dois pontos, e continua durante todas as linhas que têm um recuo similar ou superior, a próxima linha. Por exemplo:

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

É comum usar 4 espaços para cada nível de recuo. É bom não misturar política com espaços separadores, ou você pode ter problemas.

2.6 for...in

Em Python, você pode fazer loop sobre objetos iteráveis:

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

```

5 1
6 hello
7 python

```

Um atalho é comum xrange, o que gera uma gama iteráveis sem armazenar toda a lista de elementos.

```

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

```

Isso é equivalente a C / + / C + C # / Java syntax:

```

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

```

Outro comando útil é enumerate, que conta enquanto looping:

```

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

```

Há também um range(a, b, c) palavra-chave que retorna uma lista de números inteiros começando com o a, o valor incrementado por c, e terminando com o último valor menor do que b, a padrão para o e padrões c a 1. xrange é semelhante, mas na verdade não gerar a lista, apenas um iterador sobre a lista, assim é melhor para o looping.

Você pode saltar para fora de um laço usando break

```

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

```

Você pode pular para a próxima iteração do loop sem executar o bloco de código inteiro com `continue`

```
1 >>> for i in [1, 2, 3]:  
2     print i  
3     continue  
4     print 'test'  
5 1  
6 2  
7 3
```

2.7 while

`while` O loop em Python funciona muito como acontece em muitas outras linguagens de programação, por looping um número indefinido de vezes e testar uma condição antes de cada iteração. Se a condição for `False`, o loop termina.

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

Não há construção `loop...until` em Python.

2.8 def...return

Aqui é uma função típica Python:

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

Não há necessidade (ou forma) para especificar os tipos de argumentos ou o tipo de retorno (s). argumentos da função podem ter valores padrão e pode retornar vários 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

```

argumentos da função pode ser passado explicitamente pelo nome:

```

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

```

As funções podem ter um número variável 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'}

```

Aqui argumentos não passaram por nome (3, 'Olá') são armazenados na lista de a, e argumentos passados pelo nome (c e test) são armazenados no dicionário b.

No caso oposto, uma lista ou tupla pode ser passado para uma função que requer argumentos individuais de posicionamento por descompactar eles:

```

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

```

e um dicionário pode ser descompactado para oferecer argumentos-chave:

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

2.9 if...elif...else

O uso de condicionais em Python é intuitiva:

```
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 as cláusulas `elif` e `else` são opcionais. Não pode haver mais de um `elif`, mas apenas uma declaração `else`. condições complexas podem ser criadas usando o `not`, `and` e `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 try...except...else...finally

Python pode jogar - perdão, levantar - Exceções:

```
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

```

Se a exceção é gerada, ela é capturada pela cláusula `except`, que é executado, enquanto que a cláusula não é `else`. Se nenhuma exceção é gerada, a cláusula `except` não é executado, mas o `else` é. A cláusula `finally` é sempre executado.

Pode haver várias cláusulas `except` para diferentes exceções possíveis:

```

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

```

As cláusulas `else` e `finally` são opcionais.

Aqui está uma lista de built-in exceções 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 uma descrição detalhada de cada um deles, consulte a documentação oficial do Python. web2py expõe apenas uma nova exceção, chamado **HTTP**. Quando levantou, faz com que o programa para voltar uma página de erro HTTP (para mais informações sobre este consulte o Capítulo 4).

Qualquer objeto pode ser levantado como uma exceção, mas é boa prática para levantar objetos que se estendem um dos embutidos exceções.

3

```
class
```

Porque Python é tipada dinamicamente, classes e objetos Python pode parecer estranho. Na verdade, você não precisa definir as variáveis de membro (atributos) quando declarar uma classe e diferentes instâncias da mesma classe pode ter atributos diferentes. Atributos são geralmente associados com o exemplo, não a classe (exceto quando declarada como "class attributes", which is the same as "static member variables" em C + + / Java).

Aqui está um exemplo:

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

Observe que `pass` é um comando não fazem nada. Neste caso é usado para definir um `MyClass` classe que contém nada. `MyClass()` chamadas ao construtor da classe (neste caso, o construtor padrão) e retorna um objeto, uma instância da classe. O `(object)` na definição de classe indica que nossa classe estende o built-in de classe `object`. Isso não é necessário, mas é uma boa prática.

Aqui é um complexo de classe mais:

```
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
```

Funções declaradas dentro da classe são métodos. Alguns métodos têm nomes especiais reservados. Por exemplo, a `__init__` é o construtor. Todas as variáveis são variáveis locais do método, exceto as variáveis declaradas fora de métodos. Por exemplo, a `z` é uma *class variable*, equivalente a um *static member variable* C ++ que tem o mesmo valor para todas as instâncias da classe.

Repare que tem 3 argumentos `__init__` e `add` tem um, e ainda assim chamá-los com argumentos `z` e `o`, respectivamente. O primeiro argumento representa, por convenção, o nome do local dentro do método usado para se referir ao objeto atual. Aqui usamos `self` para se referir ao objeto atual, mas poderia ter usado qualquer outro nome. `self` desempenha o mesmo papel como `*this` em C ++ ou Java no `this`, `self`, mas não é uma palavra reservada.

Esta sintaxe é necessária para evitar ambigüidade quando declara classes aninhados, como uma classe que está local para um método dentro de outra classe.

3.1 Atributos especiais, métodos e operadores

Atributos de classe, métodos e operadores de partida com um duplo sublinhado são geralmente destinados a ser privado, embora esta seja uma convenção que não é imposta pelo intérprete.

Alguns deles são palavras-chave reservadas e têm um significado especial.

Aqui, por exemplo, são três deles:

- `__len__`
- `__getitem__`
- `__setitem__` Eles podem ser usados, por exemplo, para criar um objeto de recipiente que age como uma 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]
```

Outros operadores especiais incluem `__getattr__` e `__setattr__`, que definem a obter e definir atributos para a classe, e `__sum__` e `__sub__`, que os operadores aritméticos de sobrecarga. Para o uso destes operadores que remetem o leitor para livros mais avançados sobre o tema. Já mencionamos os operadores especiais `__str__` e `__repr__`.

3.2 Arquivo de entrada / saída

Em Python você pode abrir e gravar em um arquivo com:

```

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

Da mesma forma, você pode ler de volta a partir do arquivo com:

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

Alternativamente, você pode ler em modo binário com "rb", write in binary mode with "wb", and open the file in append mode "a", usando a notação C padrão.

O comando read tem um argumento opcional, que é o número de bytes. Você também pode saltar para qualquer local em um arquivo usando o seek.

Você pode ler o arquivo de volta com read

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

e você pode fechar o arquivo com:

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

embora muitas vezes isso não é necessário, porque o arquivo é fechado automaticamente quando a variável que se refere a ela sai do escopo.

Ao utilizar web2py, você não sabe onde o diretório atual é, porque depende de como web2py está configurado. O request.folder variável contém o caminho para o aplicativo atual. Caminhos podem ser concatenado com o os.path.join comando, discutidos abaixo.

3.3 lambda

Há casos em que você pode precisar gerar dinamicamente uma função sem nome. Isso pode ser feito com a palavra-chave lambda:

```

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

```

A expressão "lambda a: [b]", literalmente, diz o "a function with arguments [a] that returns [b]". Mesmo se a função é sem nome, que pode ser armazenado em uma variável e, portanto, adquire um nome. Tecnicamente isso é diferente de usar `def`, porque é a variável referente à função que tem um nome, e não a própria função.

Quem precisa de lambdas? Na verdade, eles são muito úteis porque permitem que o fator re-função em outra função, definindo os argumentos padrão, sem definir uma nova função real, mas temporária. Por exemplo:

```

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

```

Aqui é mais complexo e mais atraente de aplicação. Suponha que você tenha uma função que verifica se seu argumento é primo:

```

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

```

Esta função é consumir obviamente tempo.

Suponha que você tenha um `cache.ram` função de caching que tenha três argumentos: a chave, uma função e um número de segundos.

```

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

```

A primeira vez que ele é chamado, ele chama a função `f()`, lojas de saída em um dicionário de memória (digamos "d"), e devolve-lo para que o valor é

```

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

```

A segunda vez que ele é chamado, se a chave existe no dicionário, e não mais do que o número de segundos especificado (60), ele retorna o valor correspondente, sem realizar a chamada de função.

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

Como é que o cache de saída do `isprime` para qualquer função de entrada? Aqui é assim:

```
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
```

A saída é sempre a mesma, mas o `cache.ram` pela primeira vez é chamado, `isprime` é chamado, pela segunda vez não é.

A existência de factoring lambda permite-re uma função existente em termos de um conjunto diferente de argumentos. `cache.ram` e `cache.disk` são web2py cache funções.

3.4 exec, eval

Ao contrário de Java, Python é uma linguagem verdadeiramente interpretado. Isso significa que ele tem a capacidade de executar comandos Python armazenados em strings. Por exemplo:

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

O que aconteceu? O `exec` função diz o intérprete para se identificar e executar o conteúdo da string passada como argumento. Também é possível executar o conteúdo de uma string dentro de um contexto definido pelos símbolos em um dicionário:

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

Aqui o intérprete, ao executar o a seqüência, vê os símbolos definidos no c (b no exemplo), mas não vê c ou a si. Isso é diferente de um ambiente restrito, uma vez que exec não limita o que o código interno pode fazer, mas apenas define o conjunto de variáveis visíveis para o código.

A função está relacionada eval, que funciona muito bem como exec, exceto que ele espera que o argumento para avaliar um valor e retorna este valor.

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

3.5 import

O verdadeiro poder do Python está na sua biblioteca de módulos. Eles fornecem um conjunto amplo e consistente de Application Programming Interfaces (APIs) para bibliotecas do sistema muitas (muitas vezes de forma independente do sistema operacional).

Por exemplo, se você precisa usar um gerador de números aleatórios, você pode fazer:

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

Isso mostra um número inteiro aleatório entre 0 e 9 (incluindo 9), 5 no exemplo. O randint função é definida no módulo random. Também é possível importar um objeto de um módulo para o namespace atual:

```
1 >>> from random import randint
```

```
2 >>> print randint(0, 9)
```

ou importar todos os objetos de um módulo para o namespace atual:

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

importação ou tudo em um espaço recém-definido:

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

No resto deste livro, vamos usar principalmente objetos definidos em módulos `os`, `sys`, `datetime`, `time` e `cPickle`.

Todos os objetos web2py são acessíveis através de um módulo chamado gluon, e que é objecto dos capítulos mais tarde. Internamente, web2py usa Python muitos módulos (por exemplo, thread), mas você raramente precisará acessá-los diretamente.

Nas subseções a seguir, considerar os módulos que são mais úteis.

3.5.1 os

Este módulo fornece uma interface para o sistema operacional API. Por exemplo:

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

— Algumas das funções `os`, `chdir`, tais como, não deve ser usado em web2py porque eles não são thread-safe. — `os.path.join` é muito útil, pois permite a concatenação de caminhos uma maneira independente de sistema operacional:

```

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

```

variáveis de ambiente do sistema pode ser acessado via:

```

1 >>> print os.environ

```

que é um dicionário só de leitura.

3.5.2 sys

O módulo `sys` contém muitas variáveis e funções, mas o que usamos é o mais `sys.path`. Ele contém uma lista de caminhos onde Python procura por módulos. Quando tenta importar um módulo, Python procura por ele em todas as pastas listadas na `sys.path`. Se você instalar módulos adicionais em algum lugar e quer Python para encontrá-los, é preciso acrescentar o caminho para o local para `sys.path`.

```

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

```

Ao executar `web2py`, Python permanece residente na memória, e só há uma `sys.path`, apesar de existirem muitas linhas de serviço a pedidos HTTP. Para evitar uma fuga de memória, é melhor verificar se o caminho já está presente antes de acrescentar:

```

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

```

3.5.3 datetime

O uso do módulo `datetime` é melhor ilustrado por alguns exemplos:

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

Ocasionalmente, você pode precisar de dados de carimbo de tempo com base na hora UTC, em oposição a hora local. Neste caso você pode usar a seguinte função:

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

Os módulos datetime contém várias classes: date, datetime tempo, e timedelta. A diferença entre os dois data / hora ou duas, ou dois objetos o tempo é um 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
```

Em web2py, data e datetime são usados para armazenar os tipos correspondentes SQL quando passaram ou voltaram a partir do banco de dados.

3.5.4 time

O módulo de tempo difere date' and datetime »porque representa o tempo em segundos desde a época (início de 1970).

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

Consulte a documentação do Python para as funções de conversão entre o tempo no segundo tempo e como um datetime.

3.5.5 cPickle

Este é um módulo muito poderoso. Ele fornece funções que podem serializar praticamente qualquer objeto Python, incluindo objetos auto-referencial. Por exemplo, vamos criar um objeto estranho:

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

e agora:

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

Neste exemplo, a b é uma representação de seqüência de a e c é uma cópia do a gerada pelo de-serializar b. cPickle também pode serializar e para de-serializar a partir de um arquivo:

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


4

Visão global

4.1 Startup

web2py vem em pacotes binários para Windows e Mac OS X. Há também uma versão de código fonte que roda em Windows, Mac, Linux e outros sistemas Unix. As versões para Windows e OS X binários incluem o interpretador Python necessário. O pacote de código fonte assume que Python já está instalado no computador. web2py não requer instalação. Para começar, descompacte o arquivo zip baixado para o seu sistema operacional específico e executar o arquivo correspondente web2py.

No Windows, execute:

```
1 web2py.exe
```

No OS X, execute:

```
1 open web2py.app
```

Em Unix e Linux, executado a partir de fonte digitando:

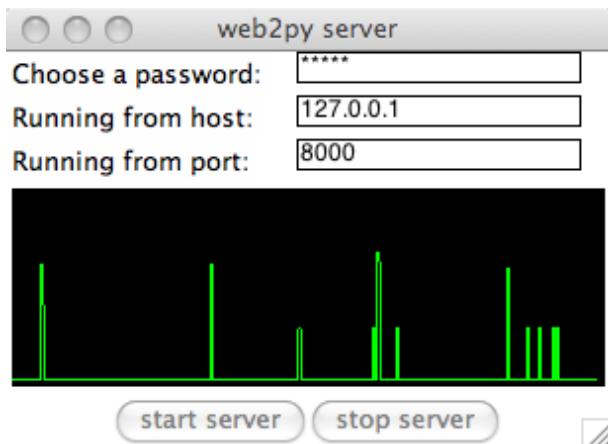
```
1 python2.5 web2py.py
```

O programa aceita web2py diversas opções de comando de linha que serão discutidas posteriormente.

Por padrão, na inicialização, web2py exibe uma janela de inicialização:



e, em seguida, exibe uma GUI widget que lhe pede para escolher uma senha de administrador de uma só vez, o endereço IP da interface de rede a ser utilizada para o servidor web e um número de porta a partir do qual a atender as solicitações. Por padrão, web2py funciona seu servidor web na porta 127.0.0.1:8000 (8000 em localhost), mas você pode executá-lo em qualquer endereço IP e porta disponível. Você pode consultar o endereço IP de sua interface de rede através da abertura de uma linha de comando e digitar ipconfig ou ifconfig no Windows no Mac OS X e Linux. A partir de agora assumimos web2py está sendo executado no localhost (127.0.0.1:8000). Usar 0.0.0.0:80 para executar web2py publicamente em qualquer uma das suas interfaces de rede.



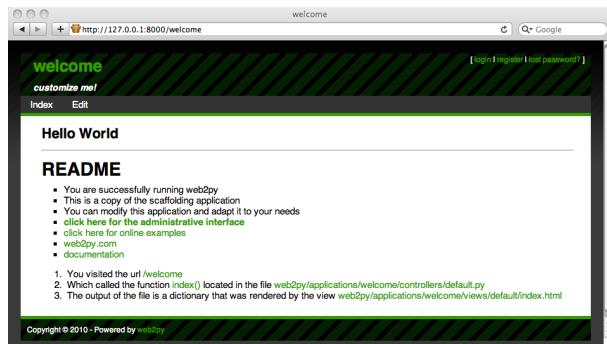
Se você não fornecer uma senha de administrador, a interface de administração está desativado. Esta é uma medida de segurança para evitar expor publicamente a interface admin.

A interface administrativa, **admin**, só é acessível a partir de localhost menos que você execute web2py atrás Apache com mod_proxy. Se detecta **admin** proxy, o cookie de sessão é definida como seguro de login e **admin** não funciona a menos que a comunicação entre o cliente eo proxy vai mais HTTPS. Esta é outra medida de segurança. Todas as comunicações entre o cliente eo **admin** deve ser sempre locais ou criptografada, caso contrário, um invasor poderia executar um man-in-ataque ou no meio de um ataque de repetição e executar um código arbitrário no servidor.

Após a senha de administração foi definida, web2py inicia o navegador da Web na página:

¹ <http://127.0.0.1:8000/>

Se o computador não tiver um navegador padrão, abra um navegador web e digite a URL.

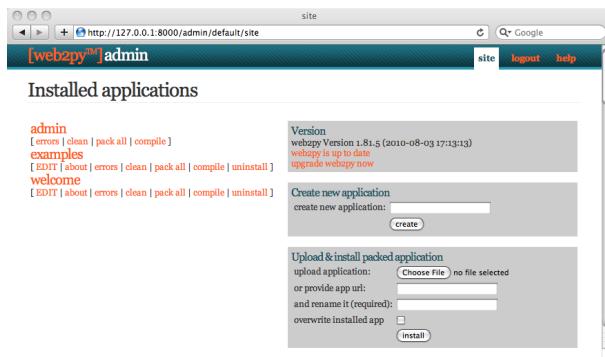


Clicando sobre "administrative interface" leva você à página de login para a interface de administração.



A senha do administrador é a mesma senha que você escolheu na inicialização. Note que existe apenas um administrador e, portanto, administrador de senha única. Por razões de segurança, o programador é convidado a escolher uma nova senha toda vez que começa a não ser que o web2py <recycle> opção é especificada. Esta é distinto do mecanismo de autenticação em aplicações web2py.

Depois que o administrador logs em web2py, o browser é redirecionado para a página "site".



Esta página lista todos os aplicativos web2py instalado e permite que o administrador para gerenciá-los. Web2py vem com três aplicações:

- Um pedido de **admin**, o que você está usando agora.
- Um aplicativo **examples**, com a documentação on-line interactiva, e uma réplica do site oficial web2py.
- A aplicação **welcome**. Este é o modelo básico para qualquer aplicação web2py outros. É referido que a aplicação de andaimes. Esta é também a aplicação que recebe um usuário na inicialização.

Ready-to-use web2py aplicações são referidos como web2py *appliances*. Você pode baixar muitos aparelhos disponíveis gratuitamente a partir de (34). web2py usuários são encorajados a apresentar novos aparelhos, tanto no código-fonte aberto ou closed-source (compilados e empacotados) formulário.

A partir da aplicação do **admin site** página, você pode realizar as seguintes operações:

- **install** um pedido através do preenchimento do formulário no canto inferior direito da página. Dê um nome para o aplicativo, selecione o arquivo que contém um aplicativo de pacote ou o URL onde o aplicativo está localizado e clique em "submit".

- **uninstall** um aplicativo clicando no botão correspondente. Existe uma página de confirmação.
- **create** um novo pedido, escolhendo um nome e clicar em "submit".
- **package** um aplicativo para distribuição, clicando no botão correspondente. A aplicação baixado é um arquivo tar com tudo, incluindo o banco de dados. Você nunca deveria descompactar esse arquivo, ele é automaticamente embalado por web2py quando se instala-lo usando **admin**.
- **clean up** um aplicativo arquivos temporários, tais como sessões, os erros e arquivos de cache.
- **EDIT** um aplicativo.

*Quando você cria um novo aplicativo usando **admin** começa como um clone do "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 os serviços. Em algumas das seguintes assumimos esses arquivos tenham sido removidos ou estão sendo substituídos.*

4.2 Diga Olá

Aqui, como exemplo, podemos criar uma aplicação web simples que exibe o "Hello from MyApp" to the user. We will call this application "myapp" mensagem. Nós também vamos adicionar um contador que conta quantas vezes o mesmo usuário visita a página.

Você pode criar um novo aplicativo simplesmente digitando o seu nome, sob a forma no canto superior direito da página no **site admin**.

Create new application

create new application:

Depois de pressionar [apresentar], o aplicativo é criado como uma cópia do built-in aplicação bem-vindos.

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 updates
 upgrade web2py now

[Create new application](#)
 create new application

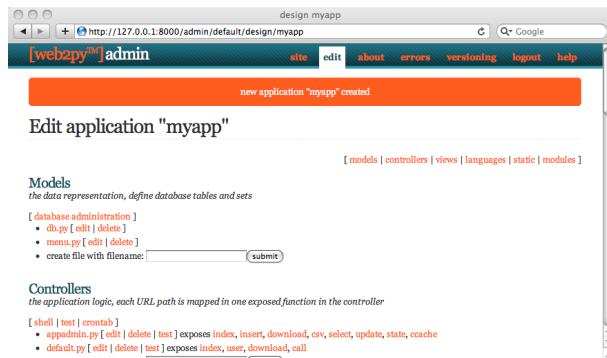
[Upload & install](#)
 upload application
 or provide app url

Para executar o novo aplicativo, visite:

¹ <http://127.0.0.1:8000/myapp>

Agora você tem uma cópia do pedido bem-vindos.

Para editar um aplicativo, clique no botão *edit* para o aplicativo recém-criado.



A página **EDIT** lhe diz o que está dentro do aplicativo. Web2py Cada pedido consiste em certos arquivos, a maioria dos quais se enquadram em uma das cinco categorias:

- **models**: descrever a representação de dados.
- **controllers**: descrever a lógica do aplicativo e fluxo de trabalho.
- **views**: descrever a apresentação dos dados.
- **languages**: descrever como traduzir a apresentação de pedido de outras línguas.
- **modules**: módulos Python que pertencem ao aplicativo.
- **static files**: imagens estáticas, files (40; 41; 42) CSS, JavaScript files (43; 44), etc

Tudo está bem organizado seguindo o padrão de projeto Model-View-Controller. Cada seção da página *edit* corresponde a uma subpasta na pasta do aplicativo.

Observe que os títulos seção irá mudar o seu conteúdo. Os nomes das pastas em arquivos estáticos também são dobráveis.

Cada arquivo listado na seção corresponde a um arquivo fisicamente localizado na subpasta. Qualquer operação executada em um arquivo através da inter-

face **admin** (*criar, editar, apagar*) pode ser feita directamente a partir do shell usando seu editor favorito.

O aplicativo contém outros tipos de arquivos (banco de dados, arquivos de sessão, os arquivos de erro, etc), mas eles não são listados na página de *edit*, porque eles não são criados ou alterados pelo administrador. Eles são criados e modificados pelo próprio aplicativo.

Os controladores contêm a lógica e workflow do pedido. Cada URL é mapeada em uma chamada para uma das funções dos controladores (ações). Existem dois controladores padrão: "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" função da seguinte forma:

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

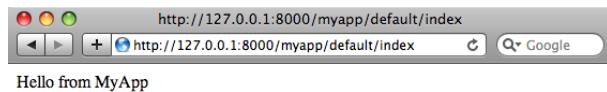
Aqui está o que o editor online é semelhante a:

Salvá-lo e voltar para a página *edit*. Clique no link do índice de visitar a página recém-criada.

Quando você visita o

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

URL a ação do índice, o controlador padrão do aplicativo myapp é chamado. Ela retorna uma seqüência que mostra o navegador para nós. Deve olhar como este:



Agora, edite a função "index" como segue:

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

Também a partir da página *edit*, editar o default view / index (o novo arquivo associado com a ação) e, neste arquivo, escreva para:

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

Agora, a ação retorna uma definição de um dicionário `message`. Quando uma ação retorna um dicionário, web2py olha para uma exibição com o nome

`[controller]/[function].[extension]` e executa-lo. Aqui] [extensão é a prorrogação solicitada. Se a prorrogação for especificado, o padrão é a "html", e é isso que vamos assumir aqui. Partindo deste pressuposto, a visão é um arquivo HTML que incorpora o código usando Python especial () tags. Em

particular, no exemplo, a `{=message}` instrui web2py para substituir o código marcado com o valor do `message` retornado pela ação. Observe que `message` aqui não é uma palavra-chave web2py mas é definida na ação. Até ao momento não foram usadas as palavras-chave web2py.

Se web2py não encontrar o ponto de vista solicitado, ele usa a visão "generic.html" que vem com cada aplicação.

Se uma extensão diferente "html" é especificada ("json" por exemplo), e a view file "[controller]/[function].json" não é encontrada, web2py procura a view "generic.json". web2py vem com generic.html, generic.json, generic.xml e generic.rss. Estes pontos de vista genérico podem ser modificados para cada aplicativo individualmente, e pontos de vista adicionais podem ser adicionados facilmente.

Leia mais sobre esse assunto no capítulo 9.

Se você voltar para a "EDIT" e clique no índice, você verá a seguinte página HTML:



4.3 Vamos contar

Vamos agora adicionar um contador para esta página que irá contar quantas vezes o mesmo visitante exibe a página. web2py forma automática e transparente os visitantes com sessões e cookies. Para cada novo visitante, ele cria uma sessão e atribui um exclusivo "session_id". A sessão é um recipiente para as variáveis que são armazenadas no servidor. A identificação original é enviado para o navegador através de um cookie. Quando solicitado pelo visitante outra página do mesmo aplicativo, o navegador envia o cookie de volta, é recuperada por web2py, ea sessão correspondente está restaurado.

Para usar a sessão, modifique o controlador padrão:

```
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)
```

Observe que counter não é uma palavra-chave web2py mas session é. Estamos pedindo web2py para verificar se há uma variável de contador na sessão e, se não, para criar um e configurá-lo para 1. Se o contador está lá, pedimos web2py para aumentar o contador de 1. Finalmente, passar o valor do contador para a exibição.

A forma mais compacto para codificar a função é essa mesmo:

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

Agora modifique a fim de adicionar uma linha que exibe o valor do contador:

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

```

6   </body>
7 </html>
```

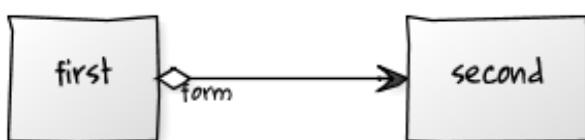
Quando você visitar a página do índice novamente (e novamente), você deve obter a seguinte página HTML:



O contador é associado a cada visitante, e é incrementado cada vez que o visitante recarregar a página. Diferentes ver os contadores de visitantes diferentes.

4.4 Say My Name

Agora crie duas páginas (primeira e segunda), onde a primeira página cria um formulário, pergunta o nome do visitante, e redireciona para a segunda página, que recebe o visitante pelo nome.



Escrever as ações correspondentes no controlador padrão:

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

Então crie um "default/first.html" para ver a primeira ação:

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:

e digite:

```
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, crie um "default/second.html" para ver a segunda ação:

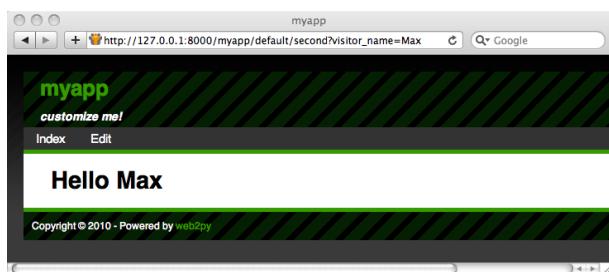
```
1 {{extend 'layout.html'}}
2 <h1>Hello {{=request.vars.visitor_name}}</h1>
```

Em ambas as visões que temos prorrogou o "layout.html" visão básica que vem com web2py. A exibição de layout mantém a aparência das duas páginas coerente. O arquivo de layout podem ser editadas e substituídos facilmente, já que principalmente contém o código HTML.

Se você visitar hoje a primeira página, digite seu nome:



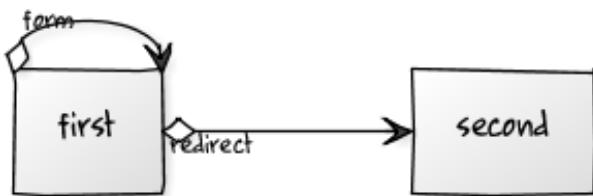
e apresentar a forma, você receberá uma saudação:



4.5 Postbacks

O mecanismo de envio de formulário que usamos antes é muito comum, mas não é boa prática de programação. Todas as entradas devem ser validadas e, no exemplo acima, a carga de validação cairia na segunda ação. Assim, a ação que realiza a validação é diferente da ação que gerou o formulário. Isso pode causar redundância no código.

A melhor forma padrão de apresentação é apresentar formas para a mesma ação que os gerou, no nosso exemplo, a "first". The "first" action should receive the variables, process them, store them server side, and redirect the visitor to the "second" página, que recupera as variáveis. Esse mecanismo é chamado de um *postback*.



Você pode modificar o controlador padrão como a seguir para implementar a auto-apresentação:

```
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()
```

Assim, é necessário modificar a visão "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>
```

ea visão "default/second.html" precisa recuperar os dados do session ao invés do request.vars:

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

Do ponto de vista do visitante, a apresentação de auto-se comporta exatamente como a aplicação anterior. Nós ainda não adicionou ainda a validação, mas agora está claro que a validação deve ser realizada pela primeira ação.

Esta abordagem é melhor também porque o nome do visitante permanece na sessão, e pode ser acessado por todas as ações e pontos de vista sobre as aplicações sem ter que ser repassados de forma explícita.

Note que, se o "second" action is ever called before a visitor name is set, it will display "Hello anonymous" porque retorna `session.visitor_name` None. Alternativamente, poderia ter acrescentado o seguinte código no controlador (dentro ou fora da função second:

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

Este é um mecanismo geral que você pode usar para aplicar a autorização de controladores, apesar de ver o capítulo 8 para um método mais poderoso.

Com web2py podemos avançar mais um passo e pedir web2py para gerar o formulário para nós, incluindo a validação. web2py fornece ajudantes (FORM, INPUT, TEXTAREA, SELECT e / OPTION) com os mesmos nomes como o equivalente tags HTML. Eles podem ser usados para construir formas, quer no tratamento ou na exibição.

Por exemplo, aqui está uma forma possível de reescrever a primeira ação:

```
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)
```

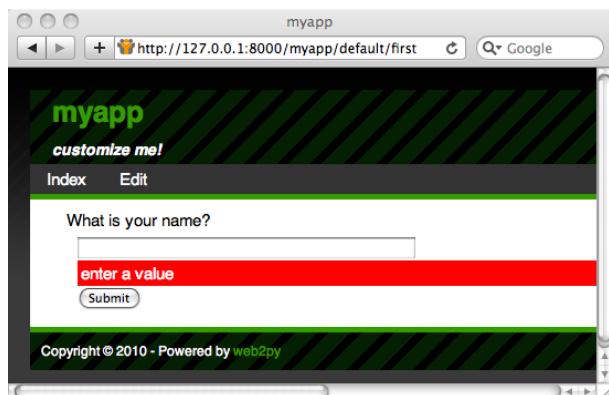
onde estamos dizendo que a tag FORM contém duas tags INPUT. Os atributos das tags de entrada são especificados pelos argumentos começando com o nome sublinhado. O argumento `requires` não é um atributo de tag (porque

não começar por sublinhar), mas estabelece um validador para o valor de visitor_name.

O objeto form pode ser facilmente serializado em HTML por inseri-la na vista "default/first.html".

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

O método aplica-se o form.accepts validadores. Se o formulário de auto-apresentado passa a validação, que armazena as variáveis na sessão e redireciona como antes. Se o formulário não passar na validação, mensagens de erro são inseridos na forma e mostrado para o usuário, mostrado a seguir:



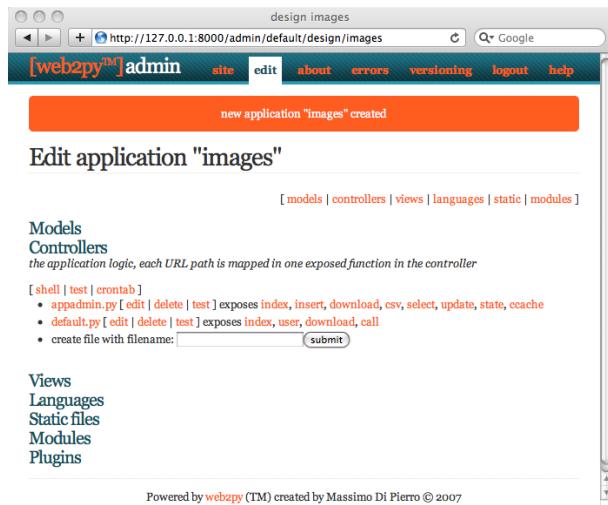
Na próxima seção iremos mostrar como as formas podem ser gerados automaticamente a partir de um modelo.

4.6 Uma Imagem Blog

Aqui, como outro exemplo, queremos criar uma aplicação web que permite ao administrador para postar imagens e dar-lhes um nome, e permite que os

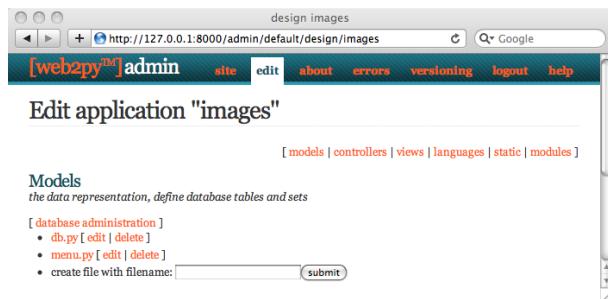
visitantes do site para visualizar as imagens e apresentar as suas observações.

Como antes, criar o novo aplicativo da página em **site admin** e navegue até a página de *edit*:



Começamos por criar um modelo, uma representação dos dados persistentes no pedido (as imagens para upload, seus nomes, e os comentários). Primeiro, você precisa para criar ou editar um arquivo de modelo que, por falta de imaginação, chamamos "db.py". We assume the code below will replace any existing code in "db.py". Modelos e controladores devem ter uma extensão de .py já que são código Python. Se a extensão não é fornecida, é anexado por web2py. Exibições em vez tem uma extensão de .html já que contêm principalmente o código HTML.

Edite o botão "db.py" file by clicking the corresponding "edit":



e digite o seguinte:

```
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 analisar esta linha por linha.

A linha 1 define uma variável global chamada `db` que representa a conexão com o banco. Neste caso, é uma conexão com um banco de dados SQLite armazenados no arquivo "applications/images/databases/storage.sqlite". No caso SQLite, se o banco não existir, ele é criado. Você pode alterar o nome do arquivo, bem como o nome do `db` variável global, mas é conveniente dar-lhes o mesmo nome, para torná-lo fácil de lembrar.

As linhas 3-5 definir um "image" tabela. `define_table` é um método do objeto `db`. O primeiro 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" é um tipo especial de área utilizada pela web2py Data Abstraction Layer (DAL) para armazenar os nomes dos arquivos enviados. web2py sabe como fazer upload de arquivos (via streaming, se forem grandes), renomeá-los com segurança, e armazená-los.

Quando uma tabela for definida, web2py leva uma das várias ações possíveis: a) se a tabela não existe, a tabela é criada; b) se a tabela existe e não correspondem à definição, a tabela é alterado em conformidade, e se um campo tem um tipo diferente, web2py tenta converter o seu conteúdo; c) se a tabela existe e corresponde à definição, web2py não faz nada.

Esse comportamento é chamado "migration". Em web2py migrações são automáticas, mas pode ser desativada para cada tabela, passando `migrate=False` como o último argumento de `define_table`.

Linhos 7-11 definir outra tabela chamada "comment". Um comentário tem um campo de referência "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" tipo que aponta para `db.image.id` através do campo.

Nas linhas 13 `db.image.title` of table "image" representa o campo. O `requires` atributo permite que você defina os requisitos / restrições que serão impostas por formas web2py. Aqui vamos exigir que o "title" é único

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

Os objetos que representam essas restrições são chamados validadores. validadores múltiplos podem ser agrupados em uma lista. Os validadores são

executadas na ordem em que aparecem. `IS_NOT_IN_DB(a, b)` é um validador especial que verifica se o valor de um `b` campo para um novo registro não é já em `a`.

Linha 14 exige que o campo está em "image_id" of table "comment" `db.image.id`. Quanto ao banco de dados está em causa, que já havia declarado isso quando definiu o "comment" da tabela. Agora estamos explicitamente dizendo o modelo que esta condição deve ser aplicada por web2py, também, a nível de processamento do formulário quando um novo comentário for publicado, para que os valores não válidos, não se propagam a partir de formas de entrada para o banco de dados. Também exigimos que a "image_id" be represented by the "title", '%(title)s', do registro correspondente.

Linha 18 indica que o "image_id" of table "comment" campo não deve ser mostrada nos formulários, `writable=False` e nem mesmo em formas `readonly`, `readable=False`.

O significado dos validadores nas linhas 15-17 deveria ser óbvio.

Observe que o

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

validador poderia ser omitido (e seria automático) é que foram para especificar um formato para representar uma imagem:

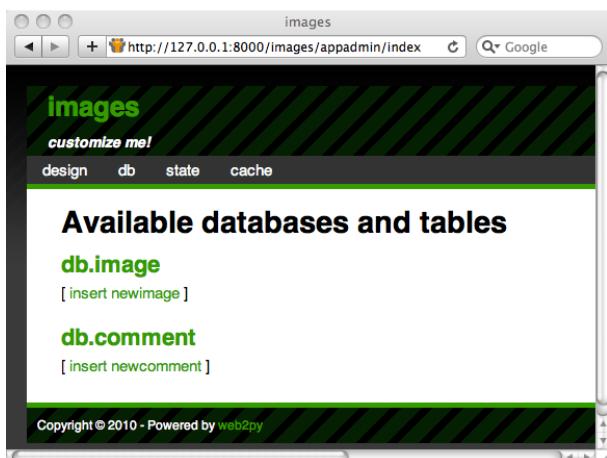
```
1 db.define_table('image', ..., format='%(title)s')
```

onde o formato pode ser uma string ou uma função que tem um registro e retorna um string.

Uma vez que um modelo é definido, se não houver erros, web2py cria uma interface de administração do aplicativo para gerenciar o banco de dados. Você acessa-lo através do link na página do "database administration" *edit* ou diretamente:

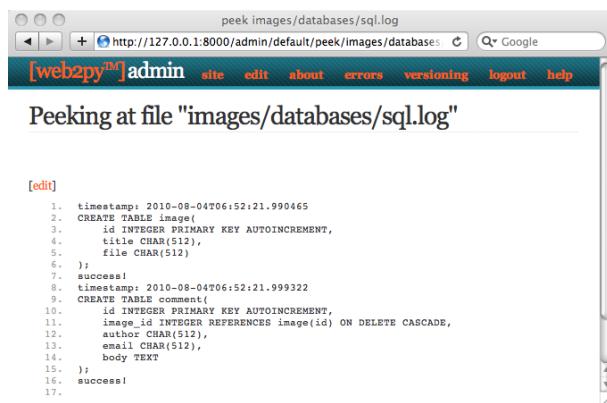
```
1 http://127.0.0.1:8000/images/appadmin
```

Aqui está um screenshot da interface do **appadmin**:



Esta interface é codificado no controlador chamado "appadmin.py" and the corresponding view "appadmin.html". De agora em diante, vamos nos referir a esta interface simplesmente como **appadmin**. Ele permite ao administrador para inserir os registros da base de novo, editar e apagar os registros existentes, tabelas procurar e executar banco junta.

O **appadmin** é acessado pela primeira vez, o modelo é executado e os quadros são criados. A DAL web2py traduz código Python em instruções SQL que são específicas para o banco de dados back-end selecionado (SQLite, neste exemplo). Você pode ver o SQL gerado a partir da página *edit* clicando no "sql.log" link under "models". Observe que o link não está presente, até as tabelas foram criadas.

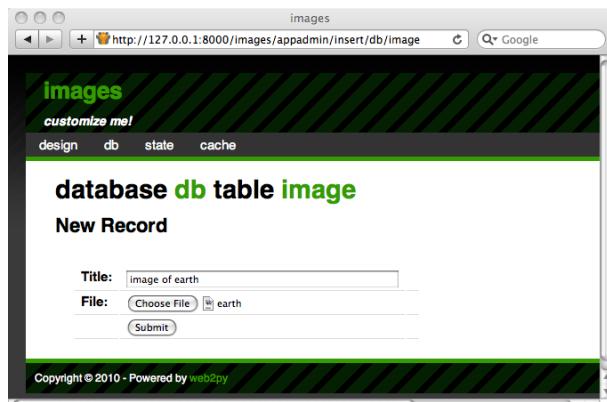


The screenshot shows a web browser window titled "peek images/databases/sql.log". The address bar contains the URL "http://127.0.0.1:8000/admin/default/peek/images/databases/sql.log". The page content displays the following SQL code:

```
[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.
```

Se você estava a editar o modelo **appadmin** e acessar novamente, web2py geraria SQL para alterar as tabelas existentes. O SQL gerado é registrado em "sql.log".

Agora volte para **appadmin** e tentar inserir um registro nova imagem:



web2py traduziu na pasta 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", eo novo nome é armazenado no campo db.image.file. Este processo foi concebido para evitar ataques de

passagem de diretório.

Observe que cada tipo de campo é processado por um *widget*. widgets padrão pode ser substituído.

Quando você clica em um nome de tabela no **appadmin**, web2py realiza uma seleção de todos os registros na tabela atual, identificada pelo

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

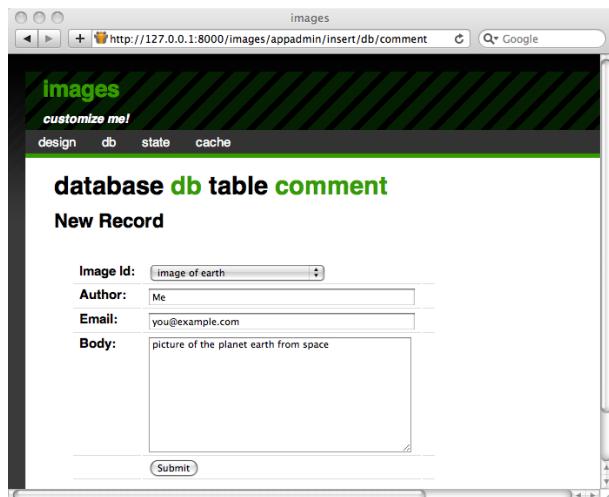
consulta DAL e processa o resultado.

The screenshot shows the web2py appadmin interface for the 'images' table. At the top, there is a query builder with fields for 'Query' (containing 'db.image.id>0'), 'Update', and 'Delete'. Below the builder, a message explains the query syntax, mentioning JOIN, AND, OR, NOT, and update/delete operations. A table titled '1 selected' displays one row with columns 'image.id', 'image.title', and 'image.file', showing values 1, 'image of earth', and 'file' respectively. 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

Você pode selecionar um conjunto diferente de registros editando a consulta SQL e pressionando "apply".

Para editar ou excluir um único registro, clique sobre o número de registro id.



Devido ao validador `IS_IN_DB`, o "image_id" campo de referência é processado por um menu drop-down. Os itens no drop-down são armazenados como chaves (`db.image.id`), mas são representadas pelos seus `db.image.title`, conforme especificado pelo validador.

Os validadores são objetos poderosos que sabem representar os campos, os valores de campo de filtro, gerar erros, e os valores formato extraído do campo.

A figura abaixo mostra o que acontece quando você enviar um formulário que não passar na validação:

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"/>

As mesmas formas que são geradas automaticamente pelo **appadmin** também pode ser gerada por meio de programação através do ajudante SQLFORM e incorporado em aplicativos do usuário. Estes formulários estão CSS-friendly, e pode ser personalizado.

Cada aplicação tem seu próprio **appadmin**, portanto, **appadmin** em si pode ser modificado sem afetar outros aplicativos.

Até agora, a aplicação sabe como armazenamento de dados e vimos como acessar o banco de dados via **appadmin**. Acesso a **appadmin** é restrita ao administrador, e não se destina a produção de uma interface web para a aplicação, portanto, a próxima parte deste passeio-through. Especificamente queremos criar:

- Uma página que lista todos os "index" imagens disponíveis classificado pelo título e links para páginas de detalhe para as imagens.
- A página "show/[id]" que mostra ao visitante a imagem solicitada e permite ao visitante ver e publicar comentários.
- A ação "download/[name]" baixar imagens carregadas.

Isso é representado esquematicamente aqui:



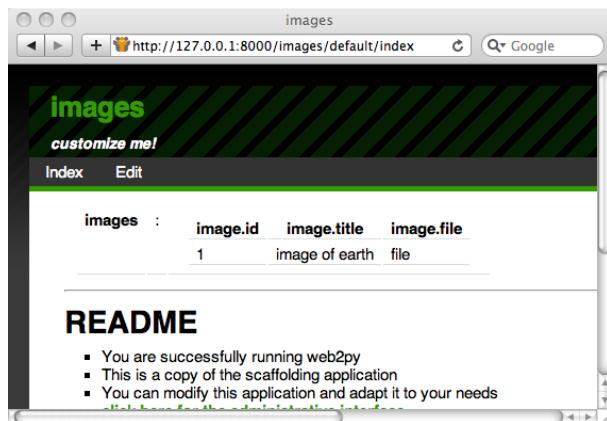
Volte para a página *edit* e "default.py" editar o controlador, substituindo o seu conteúdo com o seguinte:

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

Esta ação retorna um dicionário. As chaves dos itens do dicionário são interpretados como variáveis passadas para o ponto de vista associados à ação. Se não houver nenhum ponto de vista, a ação é processada pela visão "generic.html" que é fornecido com cada aplicação web2py.

A ação executa um índice de selecionar todos os campos (`db.image.ALL`) de imagem na mesa, ordenados por `db.image.title`. O resultado da seleção é um objeto `Rows` contendo os registros. Atribuir a uma variável local chamada `images` retornado pela ação à vista. `images` é iterável e seus elementos são as linhas selecionadas. Para cada linha das colunas podem ser acessados como dicionários: `images[0]['title']` ou equivalentemente como `images[0].title`.

Se você não escreve um ponto de vista, o dicionário é processado por "views/-generic.html" e uma chamada à ação do índice é parecido com isto:



Você ainda não criou uma exibição para esta ação, no entanto, torna assim web2py o conjunto de registros em forma tabular simples.

Continuar a criar uma visão para a ação index. Retornar para admin, editar "default/index.html" e substituir seu conteúdo com o seguinte:

```

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>
```

A primeira coisa a notar é que a visão é HTML puro, com etiquetas especiais {{...}}. O código embutido {{...}} é Python puro, com uma ressalva: o recuo é irrelevante. Blocos de começar com as linhas de código que termina em dois pontos (:) e fim em linhas que começam com a palavra-chave pass. Em alguns casos, ao final de um bloco é óbvio a partir do contexto e do uso do pass não é necessária.

Linhos de loop 07/05 sobre as linhas de imagem e para cada exibição de imagem de linha:

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

Esta é uma tag ... que contém uma etiqueta que contém o ... image.title. O valor da referência de hipertexto (atributo href) é:

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

ou seja, a URL dentro do mesmo aplicativo e controlador como a solicitação atual que chama a função chamada "show", passando por um único argumento para a função, args=image.id. LI, A, etc, são ajudantes web2py que mapeiam o HTML tags correspondentes. Seus argumentos são interpretados como anônimo objetos a ser serializado e inserido no innerHTML da etiqueta. Argumentos com nome começando com um sublinhado (por exemplo, _href) são interpretados como atributos de tags, mas sem o sublinhado. Por exemplo, é o atributo _href href, _class class é o atributo, etc

Como exemplo, a seguinte declaração:

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

é processado como:

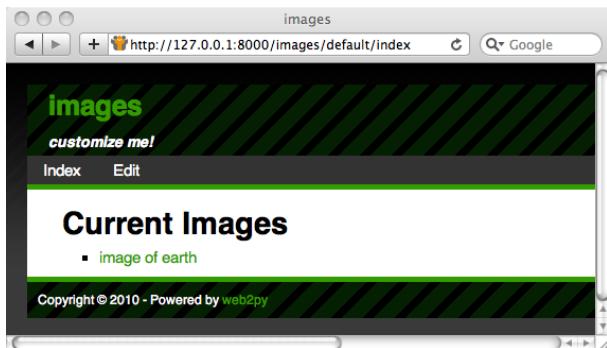
```
1 <li><a href="/images/default/show/123">something</a></li>
```

Um punhado de auxiliares (INPUT, TEXTAREA, OPTION e SELECT), também suporta alguns atributos especiais nome não começa com underscore (value e requires). Eles são importantes para a construção de formulários personalizados e será discutido mais tarde.

Volte para a página *edit*. Ele agora indica que "default.py exposes index". By clicking on "index", você poderá visitar a recém-criada página:

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

que se parece com:



Se você clicar no link o nome da imagem, você será direcionado para:

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

e isso resulta em um erro, uma vez que ainda não criou uma ação chamada "show" in controller "default.py".

Vamos editar o controlador "default.py" e substituir seu conteúdo com:

```
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)
```

O controlador possui duas ações: "show" and "download". "show" A ação seleciona a imagem com o id analisado a partir da args pedido e todos os comentários relacionados com a imagem. "show" then passes everything to the view "default/show.html".

A imagem id referenciado por:

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

no "default/index.html", pode ser acessado como: `request.args(0)` "show" da ação.

A ação "download" espera um nome de arquivo no `request.args(0)`, constrói um caminho para o local onde o arquivo é suposto ser, e envia-lo de volta para o cliente. Se o arquivo for muito grande, ele transmite os arquivos sem incorrer em qualquer sobrecarga de memória.

Observe as seguintes afirmações:

- A linha 7 cria um SQLFORM forma de inserção para a mesa `db.comment` usando apenas os campos especificados.
- A linha 8 define o valor para o campo de referência, que não faz parte do formulário de entrada porque ele não está na lista das áreas acima especificadas.
- Linha 9 processos na forma apresentada (as variáveis são apresentadas em forma `request.vars`) dentro da sessão atual (a sessão é usada para impedir o envio de casal, e para reforçar a navegação). Se as variáveis apresentadas são validadas forma, o novo comentário for inserido na tabela `db.comment`, caso contrário, o formulário é modificado para incluir mensagens de erro (por exemplo, se o endereço do autor do e-mail é inválido). Isso tudo é feito na linha 9!.
- Linha 10 só é executado quando o formulário for aceito, após o registro é inserido na tabela de banco de dados. `response.flash` é uma variável web2py que é apresentado nas vistas e utilizadas para avisar o visitante que algo aconteceu.
- Linha 11 seleciona todos os comentários de que a referência da imagem atual.

O controlador "download" action is already defined in the "default.py" da apli-

caçao de andaimes.

O "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".

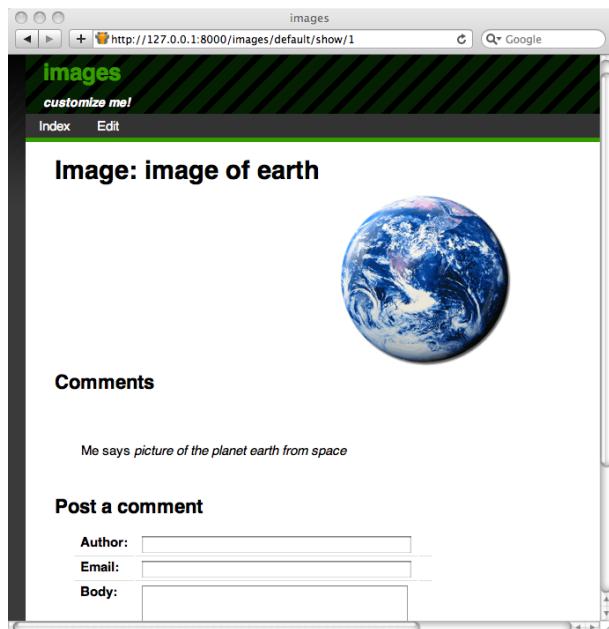
Edita esse novo arquivo e substituir seu conteúdo com o seguinte:

```

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}}>
```

Essa visão mostra o **image.file**, chamando a ação dentro de uma tag "download" . Se não houver comentários, ele faz um loop sobre eles e exibe cada um.

Aqui está como tudo irá aparecer para o visitante.



Quando um visitante envia através de um comentário nesta página, o comentário é armazenado no banco de dados e anexado na parte inferior da página.

4.7 Adicionando CRUD

web2py também fornece um CRUD (Create / Read / Update / Delete) API que simplifica as formas ainda mais. Para usar o CRUD é necessário defini-lo em algum lugar, como no "db.py" módulo:

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

Estas duas linhas estão já em aplicação andaimes.

O objeto crud fornece métodos de alto nível, por exemplo:

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

que pode ser usado para substituir o padrão de programação:

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

Aqui, vamos reescrever a ação anterior "show" CRUD e fazendo algumas melhorias mais:

```
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)
```

Antes de tudo aviso que temos usado a sintaxe

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

para buscar o registro necessário. Uma vez que nenhum retorno 'table(id)' se o registro não for encontrado, podemos usar `or redirect(...)` neste caso em uma linha.

O argumento do `next crud.create` é o URL para redirecionar para depois do formulário é aceito. O argumento `message` é o único a ser exibido após a sua aceitação. Você pode ler mais sobre o CRUD no Capítulo 7.

4.8 Adicionando autenticação

A API web2py para controle de acesso baseado em função é muito sofisticado, mas por agora vamos limitar-nos a restringir o acesso para a ação show para usuários autenticados, adiando uma discussão mais detalhada do capítulo 8.

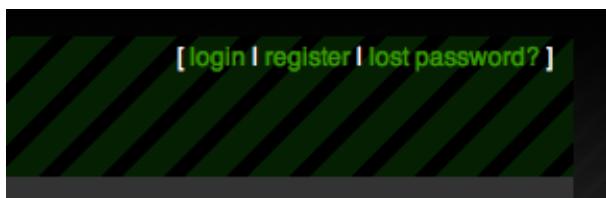
Para limitar o acesso aos usuários autenticados, é preciso preencher três etapas. Em um modelo, por exemplo, "db.py", é preciso acrescentar:

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

No nosso controlador, nós precisamos adicionar uma ação:

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

Isto é suficiente para permitir login, cadastro, logout pages, etc. O layout padrão também vai mostrar as opções para as páginas correspondentes no canto superior direito.



Agora podemos decorar as funções que deseja restringir, por exemplo:

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

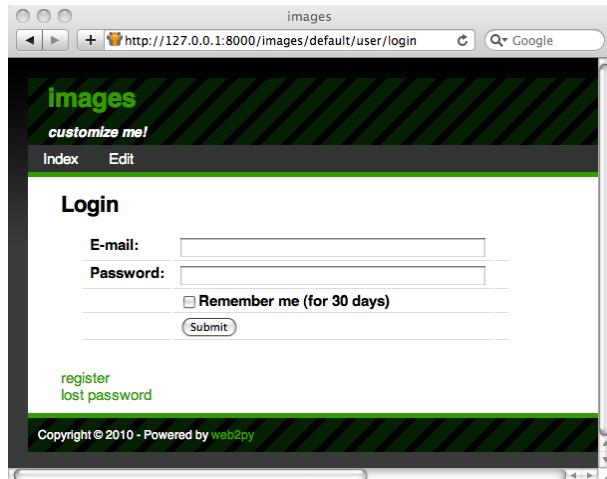
```
8     return dict(image=image, comments=comments, form=form)
```

Qualquer tentativa de acesso

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

exigirá login. Se o usuário não está conectado ele, o usuário será redirecionado para

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



A função user também expõe, entre outras, as seguintes ações:

```
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
```

Agora, pela primeira vez que um usuário precisa se registrar para poder entrar e ler / escrever comentários.

Tanto o objeto auth e user a função já estão definidas no pedido de andaimes. O objeto auth é altamente personalizável e pode lidar com a verificação de e-mail, as aprovações de registro, CAPTCHA, e métodos de login alternativo.

4.9 Configuring o Layout

Você pode configurar o layout default editando "views/layout.html" but you can also configure it without editing the HTML. In fact the "static/base.css" stylesheet is 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":

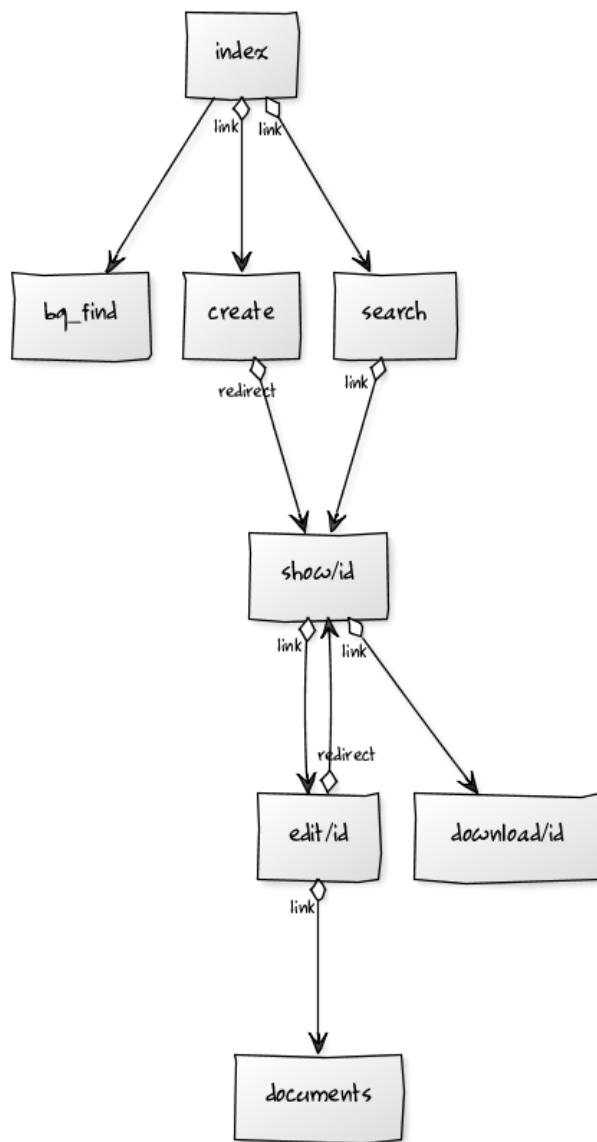
```
1 response.title = request.application
2 response.subtitle = T('customize me!')
3 response.meta.author = 'you'
4 response.meta.description = 'describe your app'
5 response.meta.keywords = 'bla bla bla'
6 response.menu = [ [ 'Index', False, URL('index') ] ]
```

4.10 A Wiki

Nesta seção, vamos construir um wiki, a partir do zero e sem utilizar a funcionalidade estendida fornecidas por plugin_wiki que é descrito no capítulo 13. O visitante poderá criar páginas, procure-los (por título), e editá-los. O visitante também vai poder postar comentários (exatamente como nas aplicações anteriores) e também postar os documentos (como anexos às páginas)

e ligá-los a partir das páginas. Por convenção, adotamos a sintaxe Markdown para a nossa sintaxe wiki. Também vamos implementar uma página de pesquisa com o Ajax, um feed RSS para as páginas, e um manipulador para pesquisar as páginas via XML-RPC (46).

O diagrama a seguir lista as ações que precisamos implementar e as ligações que pretendemos construir entre eles.



Comece criando um novo app andaimes, nomeando- "mywiki".

O modelo deve conter três tabelas: a página, comentar, e documento. Tanto

comentário e página de referência do documento, porque eles pertencem a página. O documento contém um campo de upload de arquivos do tipo como na aplicação de imagens anteriores.

Aqui é o 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 o "default.py" controlador e criar as seguintes ações:

- Índice: lista todas as páginas wiki
- criar: post outra página wiki
- Mostrar: uma página wiki e os seus comentários, e acrescentar comentários
- Editar: permite editar uma página já existente
- documentos: gerenciar os documentos anexados a uma página
- download: download de um documento (como nas imagens de exemplo)
- pesquisa: mostrar uma caixa de pesquisa e, através de uma chamada Ajax, devolver todos os títulos correspondentes, como o visitante tipos
- bg_find: a função de callback Ajax. Ele retorna o HTML que será embutido na página de pesquisa, enquanto os tipos de visitantes.

Aqui é o 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()

```

Linhas 2-6 proporcionar um comentário para a ação index. Linhas 4-5 no interior do comentário são interpretados pelo Python como o código de teste (docTest). Os testes podem ser executados através da interface de administração. Neste caso, os testes de verificar que o índice de ação é executada sem erros.

Linhas 18, 27 e 35 tentam buscar um registro com o id page no request.args(0).

Linha 13, 20 e 37 define processo e criar formas, para uma nova página e um novo comentário e um novo documento, respectivamente.

Linha 28 define processo e um formulário de atualização de uma página wiki.

Alguns mágica acontece na linha 51. O atributo do `onkeyup` "keyword" tag INPUT está definido. Toda vez que o visitante libera uma tecla, o código JavaScript dentro do atributo `onkeyup` é executado, do lado do cliente. Aqui está o código JavaScript:

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

`ajax` é uma função JavaScript definido no arquivo "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").

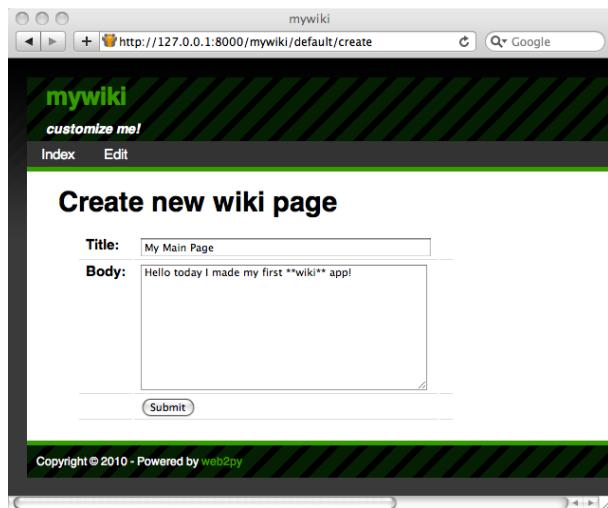
Assim que você digite algo no campo de pesquisa e soltar uma tecla, o cliente chama o servidor e envia o conteúdo da 'palavra' campo, e quando o sever responde, a resposta está embutida na própria página como o innerHTML marca o 'target'.

A "meta tag" é uma DIV definido na linha 75. Poderia ter sido definido na visão também.

Aqui está o código para o ponto de vista "default/create.html":

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

Se você visitar a página de **create**, você vê o seguinte:



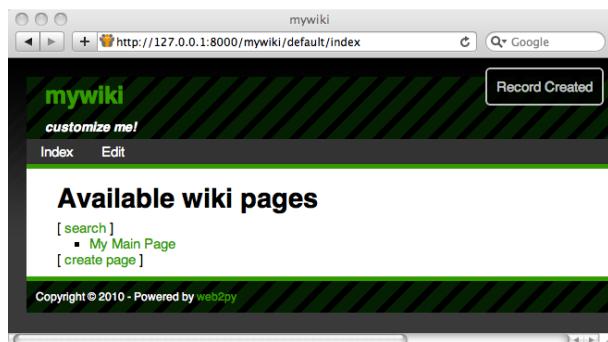
Aqui está o código para o ponto de vista "default/index.html":

```

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 {{/ul}}
7 [ {{=A('create page', _href=URL('create'))}} ]

```

Ele gera a seguinte página:



Aqui está o código para o "default/show.html" vista:

```
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}}
```

Se você quiser usar a sintaxe Markdown em vez da sintaxe markmin, de importação gluon.contrib.markdown WIKI e usá-lo em vez de auxiliar o MARKMIN. Alternativamente, você pode optar por aceitar HTML primas em vez de sintaxe markmin. Neste caso você teria que substituir:

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

com:

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

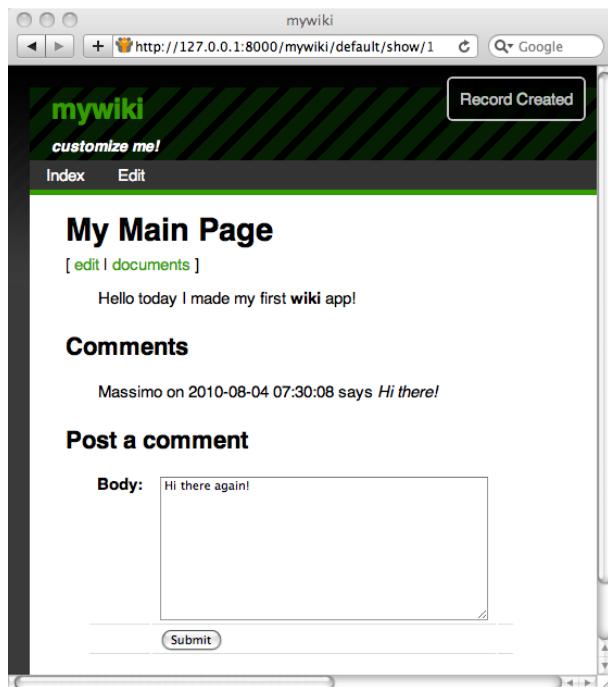
(para que o XML não consegue escapar, como por web2py comportamento padrão).

Isso pode ser feito melhor com:

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

Ao definir sanitize=True, você diz web2py escapar tags XML inseguro como "<script>", e assim evitar vulnerabilidades XSS.

Agora, se, a partir da página do índice, clicar no título de uma página, você pode ver a página que você criou:



Aqui está o código para o ponto de 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}}

```

Ele gera uma página que parece quase idêntico ao criar a página.

Aqui está o código para o ponto de vista "default/documents.html":

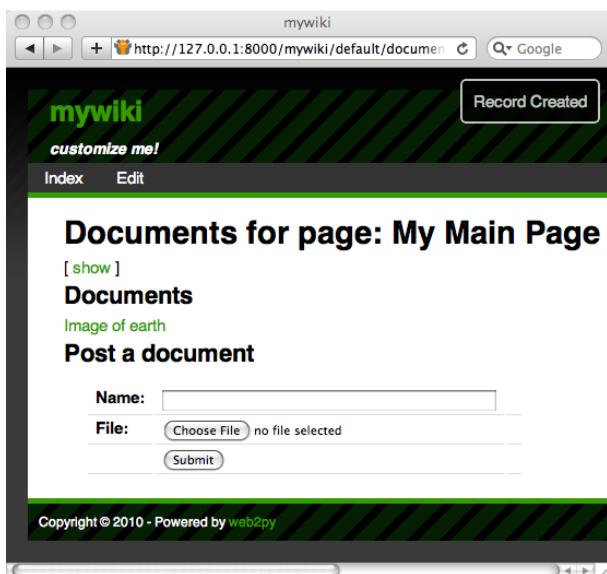
```

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 {{pass}}
9 <h2>Post a document</h2>

```

10 {{=form}}

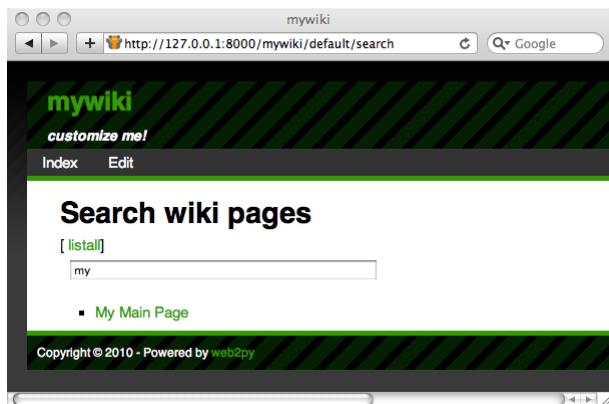
Se, a partir da página "show", você clica sobre os documentos, agora você pode gerenciar os documentos anexados à página.



Finalmente aqui está o código para o ponto de 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 gera o seguinte formulário de pesquisa Ajax:



Você também pode tentar chamar a ação callback diretamente visitando, como por exemplo, a seguinte URL:

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

Se você olhar o código fonte da página você verá o HTML retornado pelo callback:

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

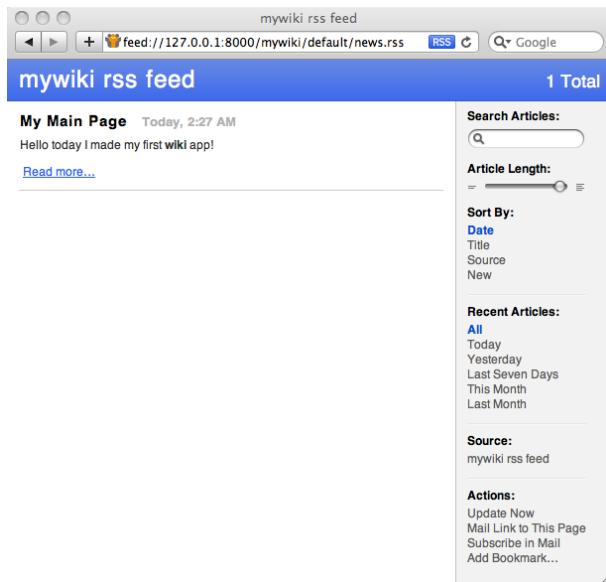
Gerando um feed RSS das páginas armazenadas usando web2py é fácil porque web2py inclui gluon.contrib.rss2. Basta acrescentar as seguintes medidas para o controlador padrão:

```
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])
```

e quando você visitar a página de

¹ <http://127.0.0.1:8000/mywiki/default/news.rss>

você vê o feed (o resultado exato depende do leitor de feeds). Observe que o dicionário é automaticamente convertido para RSS, graças à extensão rss. Na URL.



web2py também inclui feedparser para ler feeds de terceiros.

Finalmente, vamos adicionar um manipulador de XML-RPC que permite pesquisar a programação 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()

```

Aqui, a ação manipulador simplesmente publica (via XML-RPC), a funções especificadas na lista. Neste caso, o `find_by`. `find_by` não é uma ação (porque tem um argumento). Ele consulta o banco de dados com `.select()` e depois extrai os registros como uma lista com os retornos do `.response` e da lista.

Aqui está um exemplo de como acessar o manipulador de XML-RPC de um programa Python externo.

```

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

```

O manipulador pode ser acessado a partir de muitas outras linguagens de programação que compreendem XML-RPC, incluindo C, C++, C# e Java.

4.11 Mais informações sobre admin

A interface administrativa fornece funcionalidade adicional que rever brevemente aqui.

4.11.1 site

Esta página lista todos os aplicativos instalados. Existem duas formas na parte inferior.

O primeiro deles permite a criação de um novo pedido, especificando seu nome.

A segunda forma permite o upload de uma aplicação existente ou arquivo local ou uma URL remota. Quando você carregar um aplicativo, você precisa especificar um nome para ele. Este pode ser o seu nome original, mas não precisa ser. Isso permite instalar várias cópias do mesmo aplicativo. Você pode tentar, por exemplo, para carregar o KPAX sistema de gerenciamento de conteúdo:

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

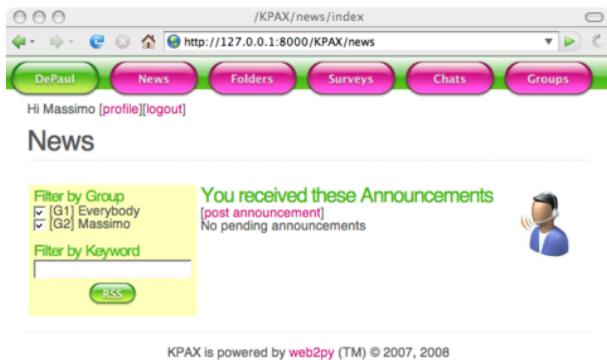
pedidos podem ser enviados arquivos .tar (Convenção de idade) e arquivos .w2p (nova convenção). Estas últimas são gzipped arquivos tar. Eles podem ser descompactado manualmente com tar zxvf [filename] embora isso nunca seja necessário.

Create new application
create new application:

Upload & install packed application
upload application: no file selected
or provide app url:
and rename it (required):
overwrite installed app

Após upload bem-sucedido, mostra web2py checksum o MD5 do arquivo carregado. Você pode usá-lo para verificar se o arquivo não foi corrompido durante o upload. O nome KPAX aparecerá na lista das aplicações instaladas.

Clique sobre o nome KPAX admin em obtê-lo em funcionamento.



Os processos — são armazenados como arquivos W2P (compactado), mas você não se destinam a piche ou untar-los manualmente; web2py faz isso por você.

Para cada aplicativo na página site permite que você:

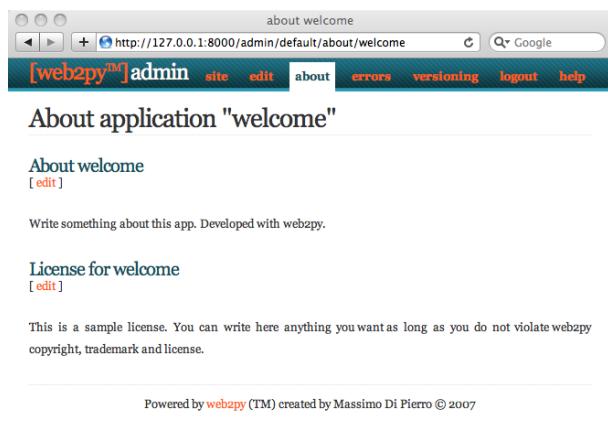
- *Desinstalar o aplicativo.*
- *Ir à página about (leia abaixo).*
- *Ir à página edit (leia abaixo).*
- *Ir à página errors (leia abaixo).*
- *Limpe os arquivos temporários (sessões, erros e arquivos cache.disk).*
- *Pack todos. Isso retorna um arquivo tar com uma cópia completa do aplicativo. Sugerimos que você limpar os arquivos temporários antes de empacotar um aplicativo.*
- *Compile o aplicativo. Se não houver erros, esta opção irá compilar bytecode, todos os modelos, controladores e vistas. Como pontos de vista podem estender e incluir outros pontos de vista em uma árvore, antes da compilação bytecode, a árvore de exibição para cada controlador é recolhido em um único arquivo. O efeito líquido é que um pedido de bytecode compilado é mais rápido, porque não há mais a análise de modelos string ou substituições que ocorrem durante a execução.*

- *Pack compilado.* Esta opção só está presente para bytecode compilado aplicações. Ela permite que a embalagem do pedido sem o código fonte para a distribuição de código fechado. Note que Python (como qualquer outra linguagem de programação) pode ser tecnicamente decompiled, portanto compilação não oferece proteção completa do código fonte. No entanto, descompilação pode ser difícil e pode ser ilegal.
- *Remover compilado.* Ele simplesmente remove o código compilado byte modelos, visões e controladores da aplicação. Se o pedido foi empacotado com o código fonte, ou concebidos a nível local, não há nenhum dano na remoção dos arquivos bytecode compilado, e a aplicação continuará a trabalhar. Se o aplicativo foi instalado formar um arquivo compactado compilado, então esta não é seguro, porque não há nenhum código-fonte para reverter, a aplicação não funcionará mais.

Todas as funcionalidades disponíveis a partir do administrador do site página web2py também é acessível através da API de programação definida na `gluon/admin.py` módulo. Basta abrir um shell python e importar este módulo.

4.11.2 *about*

A guia *about* permite a descrição de edição do pedido e sua licença. Estes são escritos, respectivamente, no Sobre e arquivos de licença na pasta do aplicativo.



Você pode usar sintaxe MARKMIN ou gluon.contrib.markdown.WIKI para esses arquivos, conforme descrito no ref. (29).

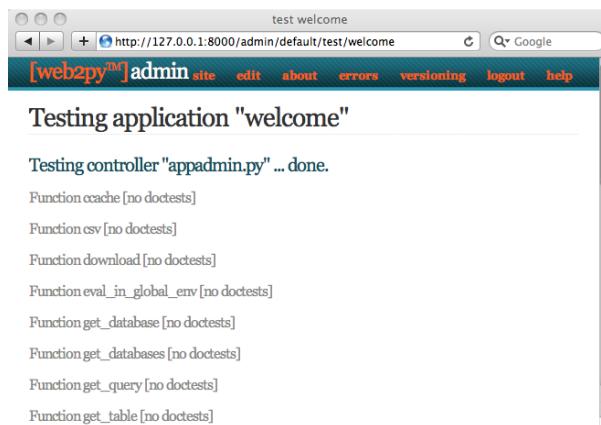
4.11.3 edit

Você usou a página de *edit* já neste capítulo. Aqui queremos apontar um pouco mais as funcionalidades da página *edit*.

- Se você clicar em qualquer nome de arquivo, você pode ver o conteúdo do arquivo com destaque de sintaxe.
- Se você clicar em editar, você pode editar o arquivo através de uma interface web.
- Se você clicar em excluir, você pode excluir o arquivo (permanente).
- Se você clicar em teste, web2py irá executar testes. Os testes são escritos pelo desenvolvedor usando doctests Python, e cada função deve ter os seus próprios testes.
- Você pode adicionar arquivos de idioma, digitalizar o aplicativo para descobrir todas as seqüências, e editar as traduções string através da interface web.

- Se os arquivos estáticos são organizados em pastas e subpastas, a hierarquia de pastas podem ser alternados clicando no nome da pasta.

A imagem abaixo mostra a saída da página de teste para a aplicação bem-vindos.



A imagem abaixo mostra a guia idiomas para a aplicação bem-vindos.

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

A imagem abaixo mostra como editar um arquivo de linguagem, neste caso, o "it" (italiano) idioma para a aplicação bem-vindos.

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

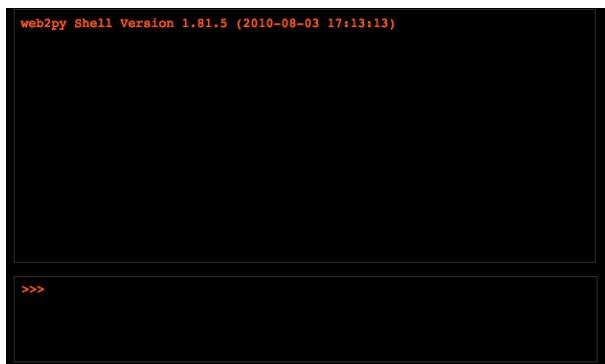
(Delete)

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

shell

Se você clicar no link "shell" no separador Controladores de *edit*, web2py irá abrir uma web shell Python e executar os modelos para o aplicativo atual.

Isto permite-lhe falar de forma interativa a sua aplicação.



crontab

Também sob a guia controladores em *edit* há um link "crontab". Ao clicar neste link você será capaz de editar o arquivo crontab web2py. Isto segue a mesma sintaxe do crontab do Unix, mas não confiam em unix. Na verdade, ele só exige web2py e funciona em Windows também. Ele permite que você registrar as ações que precisam ser executadas em segundo plano, como previsto vezes. Para mais informações sobre isso, consulte o próximo capítulo.

4.11.4 errors

Na programação web2py , você inevitavelmente cometer erros e introduzir bugs. web2py ajuda de duas maneiras: 1) que lhe permite criar testes para cada função que pode ser executado no navegador da página do *edit* e 2) quando um erro se manifesta, é emitido um bilhete para o visitante eo erro é registrado.

Propositalmente introduzir um erro na aplicação de imagens como mostrado abaixo:

```

1 def index():
2     images = db().select(db.image.ALL, orderby=db.image.title)
3     1/0
4     return dict(images=images)

```

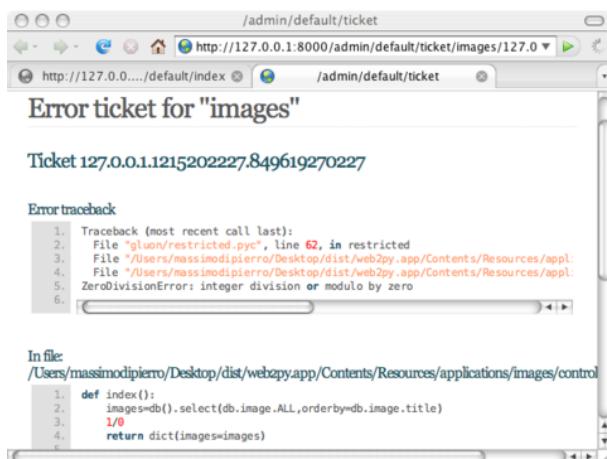
Quando você acessar a action index, você recebe o seguinte bilhete:



Internal error

Ticket issued: [images/127.0.0.1.1215202227.849619270227](http://127.0.0.1.1215202227.849619270227)

Somente o administrador pode acessar o bilhete:



O ticket mostra o traceback e o conteúdo do arquivo que causou o problema. Se o erro ocorre em um ponto de vista, web2py mostra a visão de HTML convertido em código Python. Isto permite identificar facilmente a estrutura lógica do arquivo.

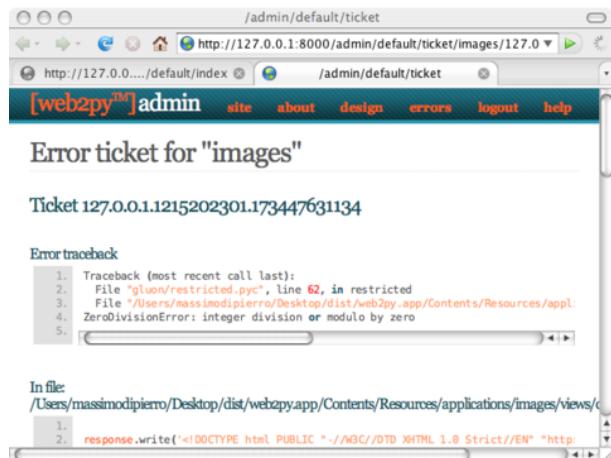
Observe que em todos os lugares **admin** mostra sintaxe destacada código (por exemplo, em relatórios de erros, palavras-chave web2py são mostrados

em laranja). Se você clicar em uma palavra-chave web2py, você será redirecionado para uma página de documentação sobre a palavra-chave.

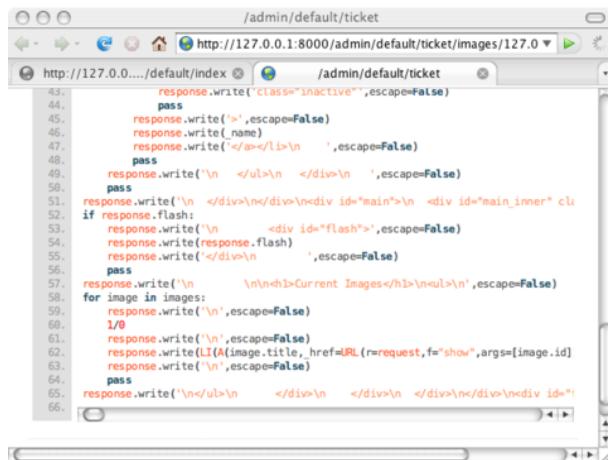
Se você reparar o / o 1 bug no processo e apresentar um índice na exibição do índice:

```
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>
```

você recebe o seguinte bilhete:



Note-se que web2py converteu o ponto de vista de HTML em um arquivo Python e, portanto, o erro descrito no bilhete se refere ao código Python gerados e não para o arquivo vista original. você recebe o seguinte bilhete:



The screenshot shows a web browser window with the URL <http://127.0.0.1:8000/admin/default/ticket/images/127.0>. The page title is "/admin/default/ticket". The content of the page is the source code of a Python script. The code uses the `response` object to write HTML and Python code to the page. It includes logic for handling a 'flash' variable and listing images. The code is color-coded with syntax highlighting.

```

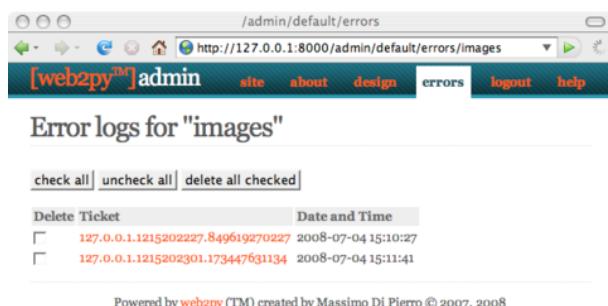
43.         response.write("class='inactive'",escape=False)
44.     pass
45.     response.write('>',escape=False)
46.     response.write(' name')
47.     response.write('</a></li>\n    ',escape=False)
48.     pass
49.     response.write('\n      </ul>\n    </div>\n  <div id="main">\n    <div id="main_inner" clu
50.   pass
51.   response.write('\n      </div>\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          Current Images</h1>\n<ul>\n',escape=False)
58.   for image in images:
59.     response.write('\n            ',escape=False)
60.     1/0
61.     response.write('\'',escape=False)
62.     response.write(LI(A(image.title,,href=URL(r=request,f="show",args=[image.id]
63.     response.write('\'',escape=False)
64.     pass
65.   response.write('\n</ul>\n      </div>\n    </div>\n  <div id="'
66.

```

Isto pode parecer confuso no início, mas na prática faz a depuração mais fácil, porque o recuo Python destaca a estrutura lógica do código que você incorporados nas vistas.

O código é mostrado na parte inferior da mesma página.

Todos os ingressos estão listados em admin na página do *errors* para cada aplicação:



The screenshot shows a web browser window with the URL <http://127.0.0.1:8000/admin/default/errors/images>. The page title is "/admin/default/errors". The navigation bar includes links for [web2py]™, admin, site, about, design, errors, logout, and help. The main content is titled "Error logs for \"images\"". Below this, there are buttons for "check all", "uncheck all", and "delete all checked". A table lists two entries under "Delete Ticket" and "Date and Time".

Delete Ticket	Date and Time
<input type="checkbox"/> 127.0.0.1.1215202227.849619270227	2008-07-04 15:10:27
<input type="checkbox"/> 127.0.0.1.1215202301.173447631134	2008-07-04 15:11:41

At the bottom, it says "Powered by web2py (TM) created by Massimo Di Pierro © 2007, 2008".

4.11.5 *Mercurial*

Se você estiver executando a partir da fonte e você tem o controle de versão bibliotecas Mercurial instalado:

```
1 easy_install mercurial
```

em seguida, a interface administrativa mostra um item mais menu chamado botão "mercurial". It automatically creates a local Mercurial repository for the application. Pressing the "commit" na página irá comprometer o aplicativo atual.

Esse recurso é experimental e vai ser melhorado no futuro.

4.12 *Mais informações sobre appadmin*

appadmin não se destina a ser exposto ao público. Ele é projetado para ajudá-lo, oferecendo um fácil acesso ao banco de dados. É constituída por apenas dois arquivos: um controlador de "appadmin.py" and a view "appadmin.html" que são utilizados por todas as ações do controlador.

O controlador **appadmin** é relativamente pequeno e de fácil leitura, que fornece um exemplo sobre a concepção de uma interface de banco de dados.

appadmin mostra que bancos de dados estão disponíveis e as tabelas que existem em cada banco de dados. Você pode inserir registros e lista todos os registros de cada tabela individualmente. saída pagina **appadmin 100** registros por vez.

Depois de um conjunto de registros é selecionada, o cabeçalho das mudanças páginas, permitindo-lhe atualizar ou excluir os registros selecionados.

Para atualizar os registros, digite um trabalho no campo SQL Query string:

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

onde os valores seqüência de caracteres deve estar entre aspas simples. Vários campos podem ser separados por vírgulas.

Para excluir um registro, clique na caixa correspondente e confirmar que você está certo.

appadmin também pode executar junta se o filtro SQL SQL contém uma condição que envolve duas ou mais tabelas. Por exemplo, tente:

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

web2py passa essa juntamente com a DAL, e entende que a consulta de duas tabelas links, daí, ambas as tabelas são selecionados com um INNER JOIN. Aqui está o resultado:

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

Se você clicar sobre o número de um campo id, você tem uma página para editar o registro com o id correspondente.

Se você clicar sobre o número de um campo de referência, você começa uma página de edição para o registro referenciado.

Você não pode atualizar ou excluir as linhas selecionadas por uma junção,

porque envolvem registros de várias tabelas e isso seria ambíguo.

5

O Core

5.1 Opções de linha de comando

É possível ignorar a GUI e começar web2py diretamente da linha de comando, digitando algo como:

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

Quando começa web2py, ele cria um arquivo chamado "parameters_8000.py" where it stores the hashed password. If you use "<ask>" como a senha, web2py pede-lhe para ela.

Para segurança adicional, você pode começar com web2py:

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

Neste caso web2py reutiliza a senha hash armazenado anteriormente. Se nenhuma senha for fornecida, ou se o arquivo "parameters_8000.py" é excluído, o interface web-based administrativa está desativado.

Em alguns / Linux sistemas Unix, se a senha for

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

web2py PAM utiliza a senha da conta do Sistema Operacional do `some_user` para autenticar o administrador, a não ser bloqueado pela configuração do PAM.

Web2py normalmente é executado com CPython (a implementação C do interpretador Python criado por Guido van Rossum), mas também pode funcionar com Jython (implementação Java do intérprete). A última possibilidade permite o uso de web2py no contexto de uma infra-estrutura J2EE. Para usar o Jython, basta substituir o "python web2py.py..." with "jython web2py.py". Detalhes sobre como instalar o Jython, zxJDBC módulos necessários para acessar os bancos de dados podem ser encontrados no capítulo 12.

O script "web2py.py" pode ter muitos argumentos de linha de comando especifica o número máximo de threads de habilitação do SSL, etc Para uma lista completa digite:

```
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
```


73 *allows multiple interfaces to be served*

opções minúsculas são usadas para configurar o servidor web. A opção -L diz web2py para ler as opções de configuração de um arquivo, instala -W web2py como um serviço do Windows, enquanto a -S, -P e -M opções de iniciar um shell interativo do Python. A opção -T verifica e executa doctests controlador em um ambiente de execução web2py. Por exemplo, o exemplo a seguir executa doctests de todos os controladores no pedido "welcome":

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

Se você executar web2py como Windows Service, -W, não é conveniente passar a configuração usando argumentos de linha de comando. Por esta razão, na pasta web2py há um exemplo de arquivo de configuração do "options_std.py" para o 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()
```

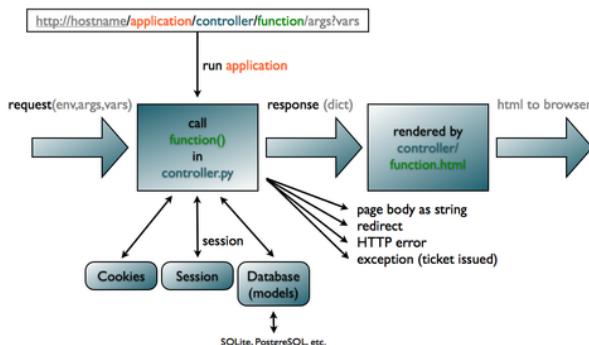
Este arquivo contém os padrões web2py. Se você editar este arquivo, você precisa importá-la explicitamente com a opção de linha de comando -L. Só funciona se você executar web2py como um serviço Windows.

5.2 Despacho

mapas web2py uma URL da seguinte forma:

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

para o `f() "c.py"` in application "a" em função do controlador. Se o `f` não está presente, o padrão web2py para a função de controlador de `index`. Se o `c` não está presente, padrões web2py ao controlador "default.py" e a se não estiver presente, padrões web2py a aplicação `init`. Se não houver nenhum pedido de `init`, web2py tenta executar o aplicativo `welcome`. Isso é mostrado esquematicamente na imagem abaixo:



Por padrão, qualquer nova solicitação também cria uma nova sessão. Além disso, um cookie de sessão é retornado ao navegador do cliente para acompanhar a sessão.

O `.html` extensão é opcional; `.html` é assumida como padrão. A extensão determina a extensão do modo de exibição que torna a saída do `f()` função do controlador. Ele permite que o mesmo conteúdo para ser servido em vários formatos (`html, xml, json, rss, etc.`)

Funções que recebem argumentos ou iniciar com um sublinhado duplo não são expostos publicamente e só pode ser chamado por outras funções.

Há uma exceção feita para URLs da forma:

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

Não há nenhum controlador chamado "static". web2py interprets this as a request for the file called "filename" in the subfolder "static" of the application "a".

Quando arquivos estáticos são baixados, web2py não cria uma sessão, nem emitir um cookie ou executar os modelos. web2py sempre cárregos arquivos estáticos em pedaços de 1MB e envia PARCIAL conteúdo quando o cliente envia uma solicitação RANGE para um subconjunto do arquivo. web2py também suporta o protocolo IF_MODIFIED_SINCE, e não envia o arquivo se ele já está armazenado no cache do browser e se o arquivo não foi alterado desde essa versão.

mapas web2py GET / POST pedidos da seguinte forma:

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

a funcionar em f "c.py" a controlador de aplicação, e armazena os parâmetros de URL na variável request da seguinte forma:

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

e:

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

e:

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

No exemplo acima, tanto a `request.args[i]` e `request.args(i)` pode ser usado para recuperar o elemento i-th do `request.args`, mas enquanto o primeiro gera uma exceção se a lista não tem um índice, Nenhum retorna neste último caso.

```
1 request.url
```

armazena a URL completa da requisição atual (não incluindo as variáveis GET).

```
1 request.ajax
```

padrões falso, mas é verdade se web2py determina que a ação foi chamado por uma solicitação do Ajax.

Se a solicitação HTTP GET é um, então `request.env.request_method` está definido para "GET", se ele é um POST, `request.env.request_method` está definido para "POST". Variáveis de consulta de URL são armazenadas no dicionário de armazenamento `request.vars`, pois eles também são armazenadas no `request.get_vars` (na sequência de uma solicitação GET) ou `request.post_vars` (na sequência de um pedido POST). Lojas web2py WSGI e variáveis de ambiente em web2py `request.env`, por exemplo:

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

e cabeçalhos HTTP em variáveis de ambiente, por exemplo:

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

Web2py Observe que valida todos os URLs para evitar ataques de passagem de diretório.

URLs são permitidas apenas para conter caracteres alfanuméricos, sublinhados, barras, o `args` pode conter pontos não consecutivos. Os espaços são substituídos por sublinhados antes da validação. Se a sintaxe URL é inválido, retorna web2py um erro HTTP 400 message (47; 48).

Se o URL corresponde a um pedido de um arquivo estático, web2py simplesmente lê e retorna (streams) o arquivo solicitado.

Se a URL não solicitar um arquivo estático web2py processa o pedido na seguinte ordem:

- Parses cookies.
- Cria um ambiente em que para executar a função.
- Inicializa `request`, `response`, cache.
- Abre o `session` existente ou cria um novo.
- Executa os modelos pertencentes à aplicação requerida.
- Executa a função de controlador de ação solicitada.
- Se a função retorna um dicionário, executa a vista associada.
- Em caso de sucesso, compromete todas as transações abertas.
- Salva a sessão.
- Retorna uma resposta HTTP.

Observe que o controlador ea visão são executadas em diversos exemplares de um mesmo ambiente, portanto, a visão não vê o controlador, mas vê os modelos e vê as variáveis retornado pela função do controlador.

Se uma exceção (além do HTTP) é gerado, web2py faz o seguinte:

- Armazena o rastreamento de erro em um arquivo e atribui um número de bilhete para ela.
- Desfaz todas as transações abertas.
- Retorna uma página de erro informando o número do bilhete.

Se a exceção é uma exceção HTTP, este é assumido como sendo o comportamento desejado (por exemplo, a HTTP redirecionamento), e todas as operações de banco de dados aberto são cometidos. O comportamento depois que é especificado pela exceção HTTP em si. A classe de exceção HTTP não é uma exceção Python padrão, é definida por web2py.

5.3 Bibliotecas

As bibliotecas web2py estão expostos às aplicações do usuário como objetos global. Por exemplo (`request`, `response`, `session`, `cache`), classes (ajudantes, validadores, DAL API) e funções (`T` e `redirect`).

Esses objetos são definidos nos arquivos de núcleo seguinte:

```

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

```

O compactado andaimes app que acompanham web2py é

```
1 welcome.w2p
```

Ele é criado durante a instalação e substituído na atualização.

A primeira vez que você iniciar web2py, duas novas pastas são criadas: depósitos e aplicações. O arquivo "welcome" app is zipped into a "welcome.w2p" para ser usado como andaime app. A pasta é usada como depósito de armazenamento temporário para instalar e desinstalar aplicativos.

web2py unidade de testes na

```
1 gluon/tests/
```

Há manipuladores para conectar-se com vários servidores Web:

```

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 foi desenvolvida por Allan Saddi)

Não existem dois arquivos de exemplo:

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

O primeiro é um arquivo de configuração opcional que pode ser passado para web2py.py com a opção -L. O segundo é um exemplo de um arquivo de mapeamento de URL. Ele é carregado automaticamente quando renomeado "routes.py".

Os arquivos

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

são arquivos de configuração necessários para a implantação no Google App Engine. Você provavelmente não é necessário modificá-los, mas você pode ler mais sobre eles nas páginas do Google Documentação.

Existem também outras bibliotecas, geralmente desenvolvidos por terceiros:

feedparser (28) por Mark Pilgrim para ler feeds RSS e Atom:

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

markdown2 (29) por Trent Mick para marcação wiki:

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

marcação **markmin**:

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

pysimplesoap é uma implementação de servidor leve SOAP criados por Mariano Reingart:

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

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

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

gql, um porto da DAL para o Google App Engine:

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

memdb, um porto da DAL em cima de memcache:

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

gae_memcache é um API para usar memcache no Google App Engine:

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

pyrtf (26) para gerar Rich Text Format (RTF) documentos, desenvolvido por Simon Cusack, e revisto 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) desenvolvido por Dalke Scientific Software, para gerar RSS feeds:

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

simplejson (25) por Bob Ippolito, a biblioteca padrão para a análise e gravação de 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) fornece API para aceitar pagamentos com cartão de crédito via rede Authorize.net

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

PAM (72) autenticação API criado por Chris Atlee:

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

Um classificador Bayesiano para preencher o banco de dados com dados fictícios para fins de teste:

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

Um arquivo que permite a interação com a barra de tarefas do Windows, quando web2py está sendo executado como um serviço:

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

login_methods opcional e **login_forms** ser usada para autenticação:

```
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 também contém uma pasta com scripts úteis:

```

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 ...

```

Os dois primeiros são particularmente úteis porque uma tentativa de instalação e configuração completa de um ambiente de produção web2py a partir do zero. Estes são discutidos no Capítulo 12, mas eles são mais ou menos auto-documentação.

Finalmente web2py inclui esses arquivos necessários para construir as distribuições binárias.

```

1 Makefile
2 setup_exe.py
3 setup_app.py

```

Estes são os scripts de configuração para o **py2exe** eo **py2app** respectivamente e eles apenas são obrigados a construir a distribuição binária do web2py. VOCÊ NUNCA DEVE NEET para executá-los.

Em resumo, as bibliotecas web2py fornecer as seguintes funcionalidades:

- Mapa URLs em chamadas de função.
- Manipular a passagem eo retorno parâmetros via HTTP.
- Realizar a validação dos parâmetros.
- Proteja os pedidos de muitos problemas de segurança.
- Manuseie a persistência de dados (banco de dados de sessão, cache, cookies).
- Traduções de string para executar vários idiomas suportados.

- Gerar HTML programaticamente (por exemplo, a partir de tabelas de banco de dados).
- Gerar SQL através de uma camada de abstração de banco de dados (DAL).
- Gerar Rich Text Format (RTF) de saída.
- Gerar Comma-Separated Value (CSV) de saída a partir de tabelas de banco de dados.
- Gerar Really Simple Syndication (RSS).
- Gera JavaScript Object Notation (JSON) cordas de serialização para o Ajax.
- Traduzir wiki markup (Markdown) para HTML.
- Exponha serviços web XML-RPC.
- Upload e download de grandes arquivos via streaming.

aplicações web2py conter arquivos adicionais, em particular as bibliotecas de terceiros JavaScript, como o jQuery, calendário e EditArea nicEdit. Seus autores são reconhecidos nos próprios arquivos.

5.4 Aplicações

Aplicativos desenvolvidos em web2py são compostas das seguintes partes:

- **models** descrever uma representação dos dados, as tabelas e as relações entre as tabelas.
- **controllers** descrever a lógica da aplicação e fluxo de trabalho.
- **views** descrever como os dados devem ser apresentados ao usuário usando HTML e JavaScript.
- **languages** descrever como traduzir strings da aplicação em vários idiomas suportados.
- **static files** não necessitam de tratamento (por exemplo, imagens, folhas de estilo CSS, etc).

- documentos **ABOUT** e **README** são auto-explicativos.
- erro loja **errors** relatórios gerados pelo aplicativo.
- **sessions** armazenar informações relacionadas a cada usuário em particular.
- **databases** armazenar bancos de dados SQLite e informações adicionais da tabela.
- **cache** armazenar em cache itens aplicação.
- **modules** são outros módulos opcionais Python.
- **private** arquivos são acessados pelos controladores, mas não diretamente pelo desenvolvedor.
- **uploads** arquivos são acessados pelos modelos, mas não diretamente pelo dono da obra (por exemplo, os arquivos enviados pelos usuários da aplicação).
- **tests** é um diretório para armazenar os scripts de teste, equipamentos e zomba.

Modelos, visões e controladores, linguagens e arquivos estáticos são acessíveis via web a administração [design] da interface. Aproximadamente, **README**, e os erros também são acessíveis através da interface de administração através dos itens de menu correspondente. Sessions, cache, os módulos e arquivos privados são acessíveis aos pedidos, mas não via a interface de administração.

Tudo está bem organizado em uma estrutura de diretórios claro que é replicado para cada aplicação web2py instalado, embora nunca o usuário precisa acessar o sistema de arquivos diretamente:

```

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

```

"**__init__.py**" é um arquivo vazio, que é necessária a fim de permitir Python (e web2py) para importar os módulos no diretório **modules**.

Observe que a aplicação **admin** simplesmente fornece uma interface web para aplicações web2py no sistema de arquivos do servidor. web2py aplicações também podem ser criadas e desenvolvidas a partir da linha de comando, você não tem que usar a interface do navegador **admin**. Um novo aplicativo pode ser criado manualmente por replicar a estrutura de diretório acima abaixo, por exemplo, "applications/newapp/" (ou simplesmente descompactar o arquivo em seu diretório welcome.w2p nova aplicação). Os arquivos do aplicativo também pode ser criada e editada a partir da linha de comando sem ter que usar a interface web **admin**.

5.5 API

Modelos, controladores e vistas são executados em um ambiente onde os seguintes objetos já são importados por nós:

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
```

Para compatibilidade com versões SQLDB=DAL e SQLField=Field. Nós encorajamos você a usar a nova sintaxe DAL e Field, em vez da sintaxe de idade.

Outros objetos e módulos são definidos nas bibliotecas, mas eles não são automaticamente importados, uma vez que não são usados com tanta frequência.

As entidades core API no ambiente de execução são web2py request, response, session, cache, URL, HTTP, redirect e T e são discutidas abaixo.

Alguns objetos e funções, incluindo **Auth**, **Crud** e **Service**, "gluon/tools.py" são definidos no e eles precisam ser importados, se necessário:

```
1 from gluon.tools import Auth, Crud, Service
```

5.6 request

O objeto request é uma instância de classe web2py onipresente que é chamado de `gluon.storage.Storage`, que estende a classe Python `dict`. É basicamente um dicionário, mas os valores do item também podem ser acessadas como atributos:

```
1 request.vars
```

é a mesma:

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

Ao contrário do dicionário, se um atributo (ou chave) não existe, ele não levanta uma exceção. Em vez disso, ele retorna None. request tem os seguintes itens / atributos, alguns dos quais são também uma instância da classe Storage:

- **request.cookies**: um objeto que contém os cookies `Cookie.SimpleCookie()` passado com a solicitação HTTP. Ele age como um dicionário de cookies. Cada "cookie" é um objeto pedaço.
- **request.env**: um objeto Storage contendo as variáveis de ambiente passado para o controlador, incluindo variáveis de cabeçalho HTTP da requisição HTTP e os parâmetros padrão WSGI. As variáveis de ambiente são convertidos para minúsculas, e os pontos são convertidos em sublinhados para facilitar a memorização.
- **request.application**: o nome do aplicativo solicitado (analisado a partir `request.env.path_info`).
- **request.controller**: o nome do controlador solicitadas (interpretados a partir do `request.env.path_info`).
- **request.function**: o nome da função solicitada (interpretados a partir do `request.env.path_info`).
- **request.extension**: a extensão da ação solicitada. O padrão para "html". Se a função de controlador retorna um dicionário e não especificar um ponto de vista, este é usado para determinar a extensão do arquivo visualizar que irá processar o dicionário (interpretados a partir do `request.env.path_info`).
- **request.folder**: o diretório do aplicativo. Por exemplo, se o pedido for "welcome", `request.folder` é definido como o "/path/to/welcome" caminho absoluto. Em seus programas, você deve sempre usar essa variável e a função `os.path.join` para construir caminhos para os arquivos que você precisa acessar. Embora web2py sempre usa caminhos absolutos, é uma boa regra nunca explicitamente alterar a pasta de trabalho atual (o que é) uma vez que esta não é uma prática segura com threads.

- **request.now:** um objeto `datetime.datetime` armazenar o carimbo de tempo da solicitação atual.
- **request.args:** Uma lista dos componentes de URL a seguir o nome da função de controle; `request.env.path_info.split('/')[3:]` equivalente a
- **request.vars:** um objeto que contém o `gluon.storage.Storage` HTTP GET e HTTP POST variáveis de consulta.
- **request.get_vars:** um objeto `gluon.storage.Storage`, contendo apenas o HTTP GET as variáveis de consulta.
- **request.post_vars:** um objeto `gluon.storage.Storage`, contendo apenas as variáveis de consulta HTTP POST.
- **request.client:** O endereço IP do cliente como determinado pela `request.env.remote_addr` ou `request.env.http_x_forwarded_for` se presente. Embora isso seja útil, não deve ser confiável, porque o `http_x_forwarded_for` pode ser falsificado.
- fluxo **request.body:** um arquivo de somente leitura que contém o corpo da solicitação HTTP. Este é automaticamente analisado para obter a `request.post_vars` e depois rebobinado. Pode ser lido com o `request.body.read()`.
- **request.wsgi** um gancho que permite chamar os aplicativos de terceiros WSGI das ações no interior.

Como exemplo, a seguinte chamada em um sistema típico:

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

resultados na tabela a seguir:

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

Que as variáveis de ambiente são definidas depende do servidor web. Aqui estamos supondo que o built-in servidor wsgi Rocket. O conjunto de var-

áveis não é muito diferente quando se utiliza o servidor web Apache.

As variáveis `request.env.http_*` são analisados a partir do cabeçalho da solicitação HTTP.

As variáveis `request.env.web2py_*`. Estes não são interpretados a partir do ambiente do servidor web, mas são criados por web2py em caso suas aplicações precisam saber sobre a localização web2py e versão, e se ele está em execução no Google App Engine (otimizações específicas, porque pode ser necessário).

Além disso, observe as variáveis `request.env.wsgi_*`. Elas são específicas para o adaptador wsgi.

5.7 response

`response` é outro exemplo de classe `Storage`. Ele contém o seguinte:

- **`response.body`**: um objeto `StringIO` em que web2py escreve o corpo da página de saída. NUNCA mudar esta variável.
- **`response.cookies`**: semelhante ao `request.cookies`, mas enquanto a segunda contém os "cookies" enviados do cliente para o servidor, o primeiro contém os cookies enviados pelo servidor para o cliente. O cookie de sessão é feita automaticamente.
- **`response.download(request, db)`**: um método usado para implementar a função de controlador que permite o download dos arquivos enviados.
- **`response.files`**: uma lista de CSS e JS exigido pela página. Eles serão automaticamente vinculado no cabeçalho do "layout.html" padrão. Para incluir um novo CSS ou JS arquivo, basta acrescentar a esta lista. Ele vai lidar com duplicatas. A ordem é significativa.
- **`response.flash`**: parâmetro opcional que pode ser incluído nas vistas. Normalmente usado para avisar o usuário sobre algo que aconteceu.

- **response.headers**: um dict para cabeçalhos de resposta HTTP.
- **response.menu**: parâmetro opcional que pode ser incluído nas exibições, normalmente usado para passar de uma árvore menu navegação à vista. Ele pode ser processado pelo ajudante MENU.
- **response.meta**: um objeto de armazenamento (como um dicionário) que contém informações de armazenamento opcional meta como `response.meta.author`, `response.meta.description`, `response.meta.keywords` e. O conteúdo da variável meta é automaticamente colocado na marca bom META pelo código "web2py_ajax.html" with is included by default by "views/layout.html".
- **response.postprocessing**: esta é uma lista de funções, vazia por padrão. Estas funções são utilizadas para filtrar o objeto de resposta na saída de uma ação, antes da saída é processado pela exibição. Ele pode ser usado para implementar o suporte para outras linguagens de template.
- **response.render(view, vars)**: um método usado para ligar para o ponto de vista explicitamente dentro do controlador. `view` é um parâmetro opcional que é o nome da exibição de arquivos, `vars` é um dicionário de valores passados para o nome de exibição.
- **response.session_file**: fluxo de arquivo que contém a sessão.
- **response.session_file_name**: o nome do arquivo onde a sessão será salva.
- **response.session_id**: o id da sessão atual. Ela é determinada automaticamente. NUNCA mudar esta variável.
- **response.session_id_name**: o nome do cookie de sessão para esta aplicação. NUNCA mudar esta variável.
- **response.status**: o código de status HTTP inteiro para ser passados para a resposta. O padrão é 200 (OK).
- **response.stream(file, chunk_size)**: quando um controlador retorna, web2py córregos o conteúdo do arquivo de volta para o cliente em blocos de tamanho `chunk_size`.
- **response.subtitle**: parâmetro opcional que pode ser incluído nas vistas. Ele deve conter o subtítulo da página.
- **response.title**: parâmetro opcional que pode ser incluído nas vistas. Ele deve conter o título da página e deve ser processado pelo título de tag HTML no cabeçalho.

- **response._vars**: esta variável é acessível apenas em uma visão, não na ação. Ele contém o valor retornado pela ação à vista.
- **response.view**: o nome do modelo de vista que deve processar a página. Isso é definido por padrão para:

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

ou, se o arquivo acima não pode ser localizado, para

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

Altere o valor dessa variável para modificar a exibição de arquivo associado com uma ação particular.

- **response.xmlrpc(request, methods)**: quando um controlador de devolve-lo, esta função expõe os métodos via XML-RPC (46). Esta função é obsoleta desde o melhor mecanismo disponível e descrito no capítulo 9.
- **response.write(text)**: um método para escrever um texto no corpo da página de saída.

Desde **response** gluon.storage.Storage é um objeto que pode ser usado para armazenar outros atributos que você pode querer passar para o ponto de vista. Embora não haja restrição técnica nossa recomendação é para armazenar apenas as variáveis que devem ser prestados por todas as páginas no layout global ("layout.html").

De qualquer forma, sugerimos para ficar com as variáveis listadas aqui:

```
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 isso irá tornar mais fácil para você substituir o arquivo padrão que vem com o "layout.html" web2py com outro arquivo de layout, que usa o mesmo conjunto de variáveis.

Versões antigas do `response.author` usuário web2py em vez de `response.meta.author` e similar para os atributos meta outros.

5.8 session

`session` é outra instância da classe `Storage`. Tudo o que é armazenado em `session`, por exemplo:

```
1 session.myvariable = "hello"
```

podem ser recuperados em um momento posterior:

```
1 a = session.myvariable
```

enquanto o código é executado dentro da mesma sessão, o usuário mesmo (desde que o usuário não tenha suprimido os cookies de sessão ea sessão não expira). Porque `session` `Storage` é um objeto, ao tentar acessar um atributo / chave que não tenha sido definida não levantar uma exceção, ele retornará ao invés `None`.

O objeto `session` tem dois métodos importantes. Um deles é **forget**:

```
1 session.forget()
```

Ele diz que não web2py para salvar a sessão. Isto deve ser utilizado nos controladores, cujas ações são chamados com frequência e não precisa controlar a atividade do usuário.

O outro método é **connect**:

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

onde db é o nome de uma conexão de banco de dados aberto (como retornado pela DAL). Diz web2py que você deseja armazenar as sessões no banco de dados e não no sistema de arquivos. web2py cria uma tabela:

```

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'))
```

e armazena cPickled sessões no campo session_data.

O masterapp=None opção, por padrão, diz web2py para tentar recuperar uma sessão existente para a aplicação com o nome no request.application, a execução da aplicação.

Se você quer que dois ou mais aplicativos para sessões partes, masterapp definir o nome do aplicativo principal.

Você pode verificar o estado do seu pedido a qualquer momento, a impressão do request, session e variáveis do sistema response. Uma maneira de fazer isso é criar uma ação específica:

```

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

5.9 cache

cache um objeto global também está disponível no ambiente de execução web2py. Ela tem dois atributos:

- **cache.ram**: o cache aplicação na memória principal.
- **cache.disk**: o cache de aplicativo no disco. cache é exigível, isto permite que ele seja usado como um decorador para o cache ações e opiniões.

Os seguintes caches exemplo, a `time.ctime()`, em função RAM:

```
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))
```

A saída do `lambda: time.ctime()` é armazenada na memória RAM para 5 segundos. O 'time' string é usado como chave de cache.

Os seguintes caches exemplo, a função `time.ctime()` em disco:

```
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))
```

A saída do `lambda: time.ctime()` é armazenada em cache no disco (usando o módulo prateleira) por 5 segundos.

O próximo caches exemplo, a `time.ctime()` função tanto de RAM e 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))
```

A saída do `lambda: time.ctime()` é armazenada em cache no disco (usando o módulo prateleira) e, em seguida, na RAM por 5 segundos. web2py olha primeiro na RAM e se não houver olha no disco. Se ele não está na RAM ou no disco, `lambda: time.ctime()` é executado eo cache é atualizado. Essa técnica é útil em um ambiente multiprocessado. As duas vezes não tem que ser o mesmo.

O exemplo a seguir é o cache em RAM a saída da função de controlador (mas não a visão):

```
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))

```

O dicionário retornado por `cache_controller_in_ram` é armazenada na memória RAM para 5 segundos. Observe que o resultado de um banco de dados selecionar não podem ser armazenadas sem primeiro ter sido serializado. A melhor maneira é o cache do banco de dados diretamente usando o método argumento `select cache`.

O exemplo a seguir é o cache de saída da função de controlador de disco (mas não a visão):

```

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))

```

O dicionário retornado por `cache_controller_on_disk` é armazenada em cache no disco por 5 segundos. Lembre-se que não pode web2py cache um dicionário que contém objetos un-pickle.

Também é possível ver o cache. O truque consiste em tornar o ponto de vista a função do controlador, para que o controlador retorna um string. Isto é feito através do envio de onde `response.render(d)` d é o dicionário que pretende passar para o modo de exibição:

O `time_expire` pode ser configurado para `0` para forçar uma atualização de cache e `None` para impedir que o conteúdo de cada vez que expirar.

Você pode limpar o cache ou mais variáveis com um

```
1 cache.ram.clear(regex='...')
```

onde `regex` é uma expressão regular matching todas as teclas que você deseja remover do cache.

O exemplo a seguir armazena em cache a saída da função de controlador de memória RAM (incluindo a visualização renderizada):

```
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)` retorna o ponto de vista apresentado como uma seqüência de caracteres que está em cache por 5 segundos. Este é o melhor e mais rápida maneira de caching.

Também é possível definir mecanismos de cache, tais como memcache. Memcache está disponível através do `gluon.contrib.memcache` e é discutido em mais detalhes no Capítulo 11.

5.10 URL

URL A função é uma das funções mais importantes em web2py. Ele gera interno caminhos URL para as ações e os arquivos estáticos.

Aqui está um exemplo:

```
1 URL('f')
```

é mapeado em

```
1 /[application]/[controller]/f
```

Observe que a saída da função URL depende do nome do aplicativo atual, o controlador de chamadas e outros parâmetros. web2py suporta mapeamento de URL e inverter o mapeamento URL. URL mapeamento permite redefinir o formato de URLs externos. Se você usa a função URL para gerar todas as

URLs internas, em seguida, aditamentos ou alterações a mapeamentos de URL irá evitar links quebrados dentro do aplicativo web2py.

Você pode passar parâmetros adicionais para a função de URL, ou seja, as condições extra no caminho do URL (args) e as variáveis de consulta de URL (vars):

```
1 URL('f', args=['x', 'y'], vars=dict(z='t'))
```

é mapeado em

```
1 /[application]/[controller]/f/x/y?z=t
```

Os atributos args são automaticamente analisados, decodificado, e, finalmente, armazenado em `request.args` por web2py. Da mesma forma, o vars são analisados, decodificado, e, em seguida, armazenados em `request.vars`. args e vars fornecer o mecanismo básico pelo qual troca informações web2py com o navegador do cliente.

Se args contém apenas um elemento, não há necessidade de passá-lo em uma lista.

Você também pode usar a função URL para gerar URLs para ações em outros controladores e outras aplicações:

```
1 URL('a', 'c', 'f', args=['x', 'y'], vars=dict(z='t'))
```

é mapeado em

```
1 /a/c/f/x/y?z=t
```

Também é possível especificar a aplicação, o controlador e a função utilizando argumentos nomeados:

```
1 URL(a='a', c='c', f='f')
```

Se o nome de um aplicativo está faltando o aplicativo atual é assumido.

```
1 URL('c', 'f')
```

Se o nome do controlador está faltando, o atual é assumido.

```
1 URL('f')
```

Em vez de passar o nome de uma função do controlador também é possível passar a própria função

```
1 URL(f)
```

Pelas razões mencionadas acima, você deve sempre utilizar a função de `URL` para gerar URLs dos arquivos estáticos para suas aplicações. Arquivos estáticos são armazenados no `static` subpasta do aplicativo (que é para onde vão quando carregou usando a interface administrativa). web2py fornece um `virtual` "estática controlador, cuja função é a de recuperar arquivos da subpasta `static`, determinar o seu tipo de conteúdo, e transmitir o arquivo para o cliente. O exemplo a seguir gera a URL para a "image.png" arquivo estático:

```
1 URL('static', 'image.png')
```

é mapeado em

```
1 /[application]/static/image.png
```

Você não precisa codificar / escapar dos argumentos `args` e `vars`, isso é feito automaticamente para você.

5.11 HTTP & redirect

web2py apenas define uma nova exceção chamada `HTTP`. Essa exceção pode ser suscitada em qualquer lugar em um modelo, um controlador, ou uma exibição com os comandos:

```
1 raise HTTP(400, "my message")
```

Isso faz com que o fluxo de controle para saltar longe usuário o código, de volta ao web2py, e retornar uma resposta 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
```

O primeiro argumento do HTTP é o código de status HTTP. O segundo argumento é a string que será retornada como o corpo da resposta. Adicional e opcional argumentos nomeados são usados para construir a resposta do cabeçalho HTTP. Por exemplo:

```
1 raise HTTP(400, 'my message', test='hello')
```

gera:

```
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
```

Se você não quer comprometer a operação de banco de dados aberto, rollback antes de levantar a exceção.

Qualquer outra exceção de causas HTTP web2py para reverter toda a transação de banco de dados aberto, o log de rastreamento de erro, emitir um bilhete para o visitante, e retornar a página de erro padrão.

Isto significa que a HTTP só pode ser usado para controle de fluxo cruzado página. Outras exceções devem ser capturadas pelo aplicativo, caso contrário são multados por web2py.

O comando:

```
1 redirect('http://www.web2py.com')
```

é simplesmente um atalho para:

```
1 raise HTTP(303,  
2     'You are being redirected <a href="%s">here</a>' % location,  
3     Location='http://www.web2py.com')
```

Os argumentos do chamado método inicializador HTTP são traduzidas em directivas cabeçalho HTTP, neste caso, o local de destino de redirecionamento. redirect tem um segundo argumento opcional, que é o código de status HTTP para o redirecionamento (303 por padrão). Alterar esse número para 307 para um redirecionamento temporário ou 301 para um redirecionamento permanente.

A maneira mais comum para o usuário é redirecionar redirecionar para outras páginas no mesmo aplicativo e (opcionalmente) passar parâmetros:

```
1 redirect(URL('index',args=(1,2,3),vars=dict(a='b')))
```

5.12 T e

Internacionalização

O T é o objeto tradutor. Constitui uma única instância global da classe web2py gluon.language.translator. Todas as constantes string (e constantes string apenas) deve ser marcado por T, por exemplo:

```
1 a = T("hello world")
```

Strings que são marcados com `T` são identificados por web2py como a necessidade de tradução de idiomas e eles vão ser traduzido quando o código (no modelo, controlador ou vista) é executado. Se o texto a ser traduzido não é uma constante, mas uma variável, que será adicionada ao arquivo de tradução em tempo de execução (exceto no jogo) para ser traduzida mais tarde.

O objeto `T` também pode conter variáveis interpoladas, por exemplo:

```
1 a = T("hello %(name)s", dict(name="Massimo"))
```

A primeira corda é traduzida de acordo com o arquivo de idioma e solicitada a variável `name` é substituído, independentemente da linguagem.

Concatenando strings de tradução não é uma boa idéia, é por isso web2py não permitem que você faça:

```
1 T("blah ") + name + T(" blah") # invalid!
```

mas não permite:

```
1 T("blah %(name)s blah", dict(name='Tim'))
```

ou o

```
1 T("blah %(name)s blah") % dict(name='Tim')
```

sintaxe alternativa

Em ambos os casos, a tradução ocorre antes do nome da variável é substituído no slot "%(name)s". A seguir, em vez não deve ser usada:

```
1 T("blah %(name)s blah" % dict(name='Tim'))
```

porque a tradução poderia ocorrer após a substituição.

A linguagem requerida é determinada pelo campo "Accept-Language" no

cabeçalho HTTP, mas esta selecção pode ser substituído de forma programática, solicitando um arquivo específico, por exemplo:

```
1 T.force('it-it')
```

que lê o arquivo de idioma "languages/it-it.py". Idioma arquivos podem ser criados e editados através da interface administrativa.

Normalmente, a tradução string é avaliada de forma tardia quando a visão se torna, assim sendo, o tradutor force método não deve ser chamado dentro de uma visão.

É possível desativar avaliação preguiçosa via

```
1 T.lazy = False
```

Desta forma, as seqüências são traduzidos imediatamente pelo operador T com base no atualmente aceita ou forçada língua.

Um problema comum é a seguinte. O pedido original está em Inglês. Suponha que exista um arquivo de tradução (por exemplo, arquivo, 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", teria usado as cordas idioma padrão (Inglês).

Há duas soluções para este problema: criar uma linguagem de tradução para Inglês, o que seria redundante e desnecessário, ou melhor, dizer que web2py línguas deve usar as seqüências de idioma padrão (as seqüências codificadas para o aplicativo). Isso pode ser feito com:

```
1 T.set_current_language('en', 'en-en')
```

Ele armazena em uma lista de idiomas que T.current_languages não precisam de tradução e as forças de uma recarga dos arquivos de linguagem.

Observe que "it" and "it-it" são línguas diferentes do ponto de vista da web2py. Para apoiar os dois, seria necessário dois arquivos de tradução, sempre em minúsculas. O mesmo é verdadeiro para todas as outras línguas.

A linguagem aceita atualmente é armazenado no

```
1 T.accepted_language
```

(...) Lembre-se que T não é apenas traduzir strings, mas também pode traduzir variáveis:

```
1 >>> a="test"
2 >>> print T(a)
```

Neste caso, o "test" mundo traduzidas, mas, se não for encontrado e se o sistema de arquivos é gravável, ele irá acrescentar à lista de palavras para ser traduzido no arquivo de linguagem.

5.13 Cookies

web2py usa cookies módulos Python para manipulação de cookies.

Os cookies do navegador está em `request.cookies` e "cookies" enviados pelo servidor está em `response.cookies`.

Você pode definir um cookie da seguinte forma:

```
1 response.cookies['mycookie'] = 'somevalue'
2 response.cookies['mycookie']['expires'] = 24 * 3600
3 response.cookies['mycookie']['path'] = '/'
```

A segunda linha informa ao navegador para manter o cookie por 24 horas. A terceira linha diz que o navegador envia o cookie de volta para qualquer aplicação (URL) no domínio atual.

O cookie pode ser protegido com:

```
1 response.cookies['mycookie']['secure'] = True
```

Um cookie seguro só é enviada de volta por HTTPS e não sobre HTTP.

O cookie pode ser recuperado com:

```
1 if request.cookies.has_key('mycookie'):
2     value = request.cookies['mycookie'].value
```

A menos que as sessões são deficientes, web2py, sob o capô, estabelece o seguinte cookie eo usa para manipular sessões:

```
1 response.cookies[response.session_id_name] = response.session_id
2 response.cookies[response.session_id_name]['path'] = "/"
```

5.14 Aplicação init

Quando você implanta web2py, você vai querer um conjunto de aplicativos padrão, ou seja, a aplicação que inicia quando há um vazio no caminho da URL, como em:

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

Por padrão, quando confrontados com um caminho vazio, web2py procura um aplicativo chamado **init**. Se não houver nenhum pedido de inicialização ele procura por um aplicativo chamado **welcome**.

Aqui estão três maneiras de definir a aplicação padrão:

- Ligue para o seu aplicativo padrão "init".
- Faça um link simbólico de "applications/init" para a pasta do seu aplicativo.
- Use URL reescrever como discutido na próxima seção.

5.15 URL Rewrite

web2py tem a capacidade de reescrever a URL de solicitações de entrada antes de chamar a ação do controlador (mapeamento de URL) e, inversamente, web2py pode reescrever o URL gerado pela função URL (reverse mapeamento URL). Uma razão para fazer isso é para manipular URLs legacy, outro é o de simplificar caminhos e torná-los mais curtos.

Para usar este recurso, crie um novo arquivo na pasta chamada "web2py" "routes.py" e definir duas listas (ou tuplas) de 2-tuplas routes_in e routes_out. Cada tupla contém dois elementos: o padrão a ser substituído ea cadeia que substitui-lo. Por exemplo:

```

1 routes_in = (
2   ('/testme', '/examples/default/index'),
3 )
4 routes_out = (
5   ('/examples/default/index', '/testme'),
6 )

```

Com estas rotas, a URL:

```
1 http://127.0.0.1:8000/testme
```

é mapeado em:

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

Para o visitante, todos os links para a URL da página parece /testme.

Os padrões têm a mesma sintaxe de expressões regulares do Python. Por exemplo:

```
1 ('.*|.php', '/init/default/index'),
```

mapeia todas as URLs terminando em ".php" para a página do índice.

Às vezes, você quer se livrar do prefixo aplicação das URLs porque você pretende expor uma única aplicação. Isto pode ser conseguido com:

```
1 routes_in = (
2     ('/(?P<any>.*)', '/init/\g<any>'),
3 )
4 routes_out = (
5     ('/init/(?P<any>.*)', '/\g<any>'),
6 )
```

Há também uma sintaxe alternativa que pode ser misturado com a anotação da expressão regular acima. Consiste em utilizar `name` ou `(?P<name>[\w_]+)` em vez de `\g<name>`. Por exemplo:

```
1 routes_in = (
2     ('/$c/$f', '/init/$c/$f'),
3 )
4
5 routes_out = (
6     ('/init/$c/$f', '/$c/$f'),
7 )
```

também eliminar o prefixo aplicação em todas as URLs "/example".

Usando a menção \$, você pode mapear automaticamente `routes_in` a `routes_out`, desde que você não use nenhum expressões regulares. Por exemplo:

```
1 routes_in = (
2     ('/$c/$f', '/init/$c/$f'),
3 )
4
5 routes_out = [(x, y) for (y, x) in routes_in]
```

Se existirem várias rotas, a primeira a corresponder a URL é executado. Se o padrão não corresponder, o caminho é deixado inalterado.

Você pode usar o `$anything` para coincidir com qualquer coisa até o fim da linha.

Aqui está um "routes.py" mínimos para manipulação de favicon e pedidos de robôs:

```

1 routes_in = (
2   ('/favicon.ico', '/examples/static/favicon.ico'),
3   ('/robots.txt', '/examples/static/robots.txt'),
4 )
5 routes_out = ()

```

Aqui está um exemplo mais complexo que expõe um "myapp" único aplicativo sem prefixos desnecessários, mas também expõe **admin**, **appadmin** e 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]]

```

A sintaxe geral para as rotas é mais complexo do que simples exemplos que temos visto até agora. Aqui está uma forma mais geral e representante exemplo:

```

1 routes_in = (
2   ('140\.191\.\d+\.\d+https://www.web2py.com:POST /(?P<any>.*)\.php',
3   '/test/default/index?vars=\g<any>'),
4 )

```

Ele mapeia solicitações https POST www.web2py.com para sediar a partir de um IP remoto correspondente ao

```

1 140\.191\.\d+\.\d+

```

expressão regular solicitando uma página correspondente do

```

1 /(?P<any>.*)\.php

```

expressão regular para

```

1 /test/default/index?vars=\g<any>

```

\g<any>, onde passa a ter a expressão regular correspondente.

A sintaxe geral é

```
1 [remote address]:[protocol]:///[host]:[method] [path]
```

A expressão inteira é acompanhada como uma expressão regular, então ":" deve ser sempre escapou e qualquer subexpressão correspondência pode ser capturado usando (?P<...>...) acordo com Python sintaxe regex.

Isto permite redirecionar os pedidos com base no endereço IP de cliente ou domínio, com base no tipo de pedido, sobre o método eo caminho. Também permite mapear as diferentes máquinas virtuais em diferentes aplicações. Qualquer subexpressão combinado pode ser usado para construir a URL de destino e, eventualmente, passado como um GET variável.

Todos os servidores web mais importantes, como o Apache eo lighttpd, também têm a capacidade de reescrever URLs. Em um ambiente de produção, sugerimos que o servidor web executar URL reescrita.

5.16 Rotas em caso de erro

Você também pode usar o "routes.py" para redirecionar o visitante para ações especiais no caso de haver um erro no servidor. Você pode especificar esse mapeamento global, para cada aplicação, para cada código de erro, para cada aplicativo e código de erro. Aqui está um exemplo:

```
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 da primeira corda é colocada contra "[appname]/[error code]". Se for encontrada uma correspondência que o usuário é redirecionado para

a URL na seqüência de segunda a tupla correspondente. No caso de um bilhete foi emitido, o bilhete é passado para a nova URL como um bilhete chamado GET variável.

Inigualável erros exibir uma página de erro padrão. Esta página de erro padrão também pode ser personalizado aqui:

```

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">%(ticket)s</a></body></html>'''
```

A primeira variável contém a mensagem de erro quando um aplicativo inválido for solicitado. A segunda variável contém a mensagem de erro quando um ticket é emitido.

5.17 Cron

O cron web2py fornece a capacidade para as aplicações para executar tarefas em horários predefinidos, em uma plataforma de forma independente.

Para cada aplicativo, funcionalidade cron é definido por um arquivo crontab "app/cron/crontab", seguindo a sintaxe definida no ref. (45) (com algumas extensões que foi web2py específico).

Isto significa que cada aplicação pode ter uma configuração separada e que cron cron config pode ser alterado a partir de dentro web2py sem afetar o próprio sistema operacional hospedeiro.

Aqui está um exemplo:

```

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
```

As duas últimas linhas neste exemplo, as extensões usar a sintaxe do cron regular para fornecer funcionalidades adicionais web2py. web2py cron tem uma sintaxe de alguns detalhes extra para apoiar a aplicação web2py.

Se a tarefa script / é prefixado com um asterisco (*) e termina com o ".py", ele será executado no ambiente web2py. Isto significa que você terá todos os controladores e modelos à sua disposição. Se você usar dois asteriscos (**), os modelos não serão executadas. Esta é a maneira recomendada de chamar, pois tem menos sobrecarga e evita potenciais problemas de travamento.

Observe que os scripts / funções executados no ambiente web2py exigem um db.commit() manual no final da função ou a transação será revertida. web2py não gera bilhetes ou tracebacks significativas no modo shell (no cron que é executado). Certifique-se de que seu código é executado sem erros web2py antes de configurá-lo como uma tarefa cron, como você provavelmente não será capaz de vê-los quando executado a partir do cron.

Além disso, tenha cuidado como você usa modelos. Embora a realização acontece em um processo separado, fechaduras banco de dados devem ser tidos em conta, a fim de evitar páginas de espera para as tarefas do cron que estar bloqueando o banco de dados. ** Use a sintaxe, se você não precisa usar o banco de dados em sua tarefa cron.

Você também pode chamar uma função do controlador. Não há necessidade de especificar um caminho. O controlador ea função será a de invocar a aplicação. Tome especial cuidado sobre as ressalvas acima. Exemplo:

```
1 */30 * * * * root *mycontroller/myfunction
```

Se você especificar @ reboot no primeiro campo no arquivo crontab, a determinada tarefa será executada apenas uma vez, na inicialização web2py. Você pode usar esse recurso se você deseja armazenar em cache de pré-seleção ou inicializar dados para um aplicativo na inicialização web2py. Observe que as tarefas cron são executadas em paralelo com a aplicação — se o pedido não está pronto para atender as solicitações até que a tarefa do cron é terminado, você deve executar verificações para refletir isso. Exemplo:

```
1 @reboot * * * * root *mycontroller/myfunction
```

Dependendo de como você está invocando web2py, existem quatro modos de operação para web2py cron.

- Soft cron: disponível em todos os modos de execução
- Hard cron: disponível se usar o built-in web server (diretamente ou via Apache mod_proxy)
- Externas cron: disponível se você tiver acesso ao serviço do sistema cron própria
- Nenhum cron

O padrão é cron difícil se você estiver usando o built-in web server, em todos os outros casos, o padrão é cron macio. cron Soft é o padrão se você estiver usando CGI, FastCGI ou WSGI. Suas tarefas serão executadas na primeira chamada (carregamento da página) para web2py após o tempo especificado no crontab (mas após o processamento da página, por isso não demora para o usuário é visível). Obviamente, há alguma incerteza exatamente quando a tarefa será executada dependendo do tráfego do site recebe. Além disso, a tarefa do cron pode ficar interrompido se o servidor web tem um tempo limite de carregamento da página definida. Se essas limitações não são aceitáveis, consulte "external cron". Soft cron é um recurso razoável passado, mas se o seu servidor web permite que os métodos cron outros, devem ter preferência sobre cron macio.

Hard cron é o padrão se você estiver usando o built-in web server (diretamente ou através de mod_proxy do Apache). cron Duro é executado em um segmento paralelo, ao contrário do cron mole não existem limitações no que diz respeito ao tempo de execução e precisão o tempo de execução. cron externo não é padrão em qualquer cenário, mas exige que você tenha acesso às instalações de cron do sistema. Ele é executado em um processo paralelo, portanto, nenhuma das limitações do cron soft aplica. Esta é a forma recomendada de usar o cron em WSGI ou FastCGI.

Exemplo de linha para adicionar o crontab do sistema (geralmente / etc / crontab):

```
1 0-59/1 * * * * web2py cd /var/www/web2py/ && python web2py.py -C -D 1 >> /tmp/cron.  
      output 2>&1
```

Se você estiver executando o cron externo, certifique-se de adicionar o parâmetro -N de linha de comando no seu script de inicialização ou web2py config que não há colisão de vários tipos de cron.

No caso de você não precisa de nenhuma funcionalidade cron dentro de um determinado processo, você pode usar o N-parâmetro de linha de comando para desativá-lo. Note que isso pode desabilitar algumas tarefas de manutenção (como a limpeza automática de dirs sessão). O uso mais comum desta função de:

- Você já criaram cron externa desencadeada a partir do sistema (mais comum com as configurações WSGI)
- Se você quiser depurar seu aplicativo cron sem interferir nem com ações ou com saída

5.18 Processos Antecedentes e filas de tarefas

Enquanto o cron é útil para executar tarefas em intervalos regulares de tempo nem sempre é a melhor solução para executar uma tarefa em segundo plano. Para este efeito web2py fornece a capacidade de executar qualquer script python como se fosse um controlador de insider:

```
1 python web2py.py -S app -M -N -R applications/app/private/myscript.py -A a b c
```

onde -S app diz web2py para executar "myscript.py" as "app", -M diz web2py para executar modelos -N diz não web2py para executar cron, e passa -A a b c argumentos opcionais da linha de comando para sys.argv=['a','b','c'] "myscript.py".

Um caso de teste típico consiste em uma fila de processamento.

Considere este modelo

```
1 db.define_table('queue',
2     Field('status'),
3     Field('email'),
4     Field('subject'),
5     Field('message'))
```

e uma aplicação que as mensagens enqueue para ser enviado por

```
1 db.queue.insert(status='pending',
2                  email='you@example.com',
3                  subject='test',
4                  message='test')
```

O script de processamento em segundo plano que envia os e-mails podem ser

```
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
```

Observe que o objeto `mail` é definido no arquivo `app db.py` no andaime, por causa da opção `-M` é visível aqui. Pode precisam ser configurados no `db.py` para funcionar corretamente. Notar também que é importante para cometer qualquer alteração o mais cedo possível a fim de não bloquear o banco de dados para outros processos simultâneos.

Este tipo de processo em segundo plano não deve ser executado via cron (exceto talvez por cron @ reboot), porque você precisa ter certeza de que

não mais de uma instância está sendo executado ao mesmo tempo. Com o cron é possível que se o processo começa no cron uma iteração e não é preenchido pelo cron iteração 2, assim cron começa novamente, e novamente, e novamente, assim, interferência do servidor de correio.

5.19 Terceira Parte Módulos

web2py é escrito em Python, por isso pode importar e usar qualquer módulo Python, incluindo módulos de terceiros. Ele só precisa ser capaz de encontrá-los.

Os módulos podem ser instalados no diretório Python oficial "site-packages" ou em qualquer lugar de sua aplicação pode encontrá-los.

Módulos na "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". Você pode instalá-lo pelo shell usando o comando `easy_install` PEAK:

```
1 easy_install py-matplotlib
```

em seguida, você pode importá-lo em qualquer modelo de controle e visualização com:

```
1 import matplotlib
```

Windows web2py distribuição binária tem um site, os pacotes na pasta de nível superior. A distribuição binária Mac possui uma pasta local, os pacotes na pasta:

`web2py.app/Contents/Resources/site-packages`

Você também pode instalar pacotes manualmente na aplicação "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" com: web2py também oferece uma função `local_import`. Aqui está um exemplo de uso:

```
: mymodule = local_import('mymodule')
```

A função procura `mymodule` na pasta app módulos local e importação, com o nome no lado esquerdo da igualdade.

Esta função recebe três argumentos: `name`, `reload` e `app`. Quando você especificar `reload=True`, ele vai voltar a importar o módulo em cada solicitação, caso contrário, seu processo de python só importar o módulo de uma vez. O padrão é `reload=False`. `app` é o nome do aplicativo a partir do qual importar o módulo, o padrão para `request.application`.

A razão para esta função é que o servidor já pode executar várias instâncias web2py você não quer adicionar seus caminhos módulos para `sys.path` mais a pesquisa poderia se tornar dependente da ordem.

5.20 Execution Environment

Apesar de tudo discutido aqui funciona bem, recomendamos em vez construir sua aplicação usando componentes, conforme descrito no capítulo 13.

modelo web2py e arquivos de controle não são módulos Python que não podem ser importados usando a instrução Python `import`. A razão para isto é que os modelos e os controladores são projetados para serem executados em um ambiente preparado que foi pré-preenchido com web2py objetos globais (`request`, `response`, `session`, `cache` e `T`) e funções auxiliares. Isso é necessário porque Python é uma forma estática (lexically língua) escopo, ao passo que o ambiente web2py é criado dinamicamente. web2py `exec_environment` oferece

a função para permitir que os modelos de acesso e os controladores diretamente. `exec_environment` cria um ambiente de execução web2py, carrega o arquivo nele e em seguida, retorna um objeto de armazenamento contendo o meio ambiente. O objeto de armazenamento também serve como um mecanismo de namespace. Qualquer arquivo Python concebido para ser executado no ambiente de execução pode ser carregado utilizando `exec_environment`. Usos para incluir `exec_environment`:

- Acessando dados (modelos) a partir de outras aplicações.
- Acessando objetos global de outros modelos ou controladores.
- Execução das funções de controlador de outros controladores.
- Carregando bibliotecas auxiliares em todo o site.

Este exemplo lê as linhas da tabela user na aplicação 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)
```

Outro exemplo: suponha que você tenha um "other.py" controlador que contém:

```
1 def some_action():
2     return dict(remote_addr=request.env.remote_addr)
```

Aqui está como você pode chamar essa ação de outro controlador (ou a partir do shell web2py):

```
1 from gluon.shell import exec_environment
2 other = exec_environment('applications/app/controllers/other.py', request=request)
3 result = other.some_action()
```

Na linha 2, `request=request` é opcional. Ele tem o efeito de passar a solicitação atual para o ambiente de "other". Sem esse argumento, o meio ambiente deverá conter um novo e vazio (para além da `request.folder`) objeto de solicitação. Também é possível passar uma resposta e um objeto de sessão para `exec_environment`. Tenha cuidado ao passar pedido, resposta e objetos

de sessão — modificação pela ação chamada ou codificação dependências da ação chamada pode levar a efeitos colaterais inesperados.

A chamada de função na linha 3 não executa o ponto de vista, ele simplesmente retorna o dicionário, a menos `response.render` é chamado explicitamente pelo "some_action".

Uma advertência final: não usar inapropriadamente `exec_environment`. Se você quer os resultados das acções em outro aplicativo, você provavelmente deve implementar uma API XML-RPC (aplicação de um XML-RPC API com web2py é quase trivial). Não use `exec_environment` como um mecanismo de redirecionamento, usar o helper `redirect`.

5.21 Cooperação

Há muitas maneiras de aplicativos podem cooperar:

- Os aplicativos podem se conectar ao mesmo banco de dados e, portanto, tabelas partes. Não é necessário que todas as tabelas no banco de dados são definidos por todas as aplicações, mas deve ser definido por aquelas aplicações que usá-los. Todos os aplicativos que usam a mesma tabela, mas, deve definir a tabela com o `migrate=False`.
- As candidaturas podem incorporar componentes de outras ações utilizando o `LOAD` ajudou (descrito no Capítulo 13).
- Sessões podem compartilhar aplicativos.
- Os aplicativos podem chamar uns aos outros as ações remotamente via XML-RPC.
- Os aplicativos podem acessar uns dos outros arquivos através do sistema de arquivos (assumindo que eles compartilham o mesmo sistema de arquivos).
- Os aplicativos podem chamar uns aos outros as ações localmente usando `exec_environment` como discutido acima.

- Os aplicativos podem importar uns dos outros módulos usando a sintaxe:
- Os aplicativos podem importar qualquer módulo no caminho de procura `PYTHONPATH`, `sys.path`.

Um aplicativo pode carregar a sessão de outro aplicativo usando o comando:

```
1 session.connect(request, response, masterapp='appname', db=db)
```

Aqui "appname" é o nome do aplicativo principal, o que define o `session_id` inicial no cookie. `db` é uma conexão de banco de dados para o banco de dados que contém a tabela de sessão (`web2py_session`). Todos os aplicativos que as sessões partes deve usar o mesmo banco de dados para armazenamento de sessão.

Um aplicativo pode carregar um módulo de outro aplicativo usando

```
1 othermodule = local_import('othermodule', app='otherapp')
```

Se um módulo de função necessita de acesso a um dos objetos centrais (request, response, session, cache e T), os objetos devem ser passados explicitamente para a função. Não deixe que o módulo de criar outra instância dos objetos centrais. Caso contrário, a função não irá se comportar conforme o esperado.

5.22 WSGI

web2py WSGI e ter um relacionamento de amor e ódio. Nossa perspectiva é que WSGI foi desenvolvido como um protocolo para conectar servidores web para aplicações web de maneira portátil e iremos utilizá-lo para essa finalidade. web2py em seu núcleo é uma aplicação WSGI: `gluon.main.wsgibase`. Alguns desenvolvedores têm empurrado WSGI aos seus limites como um protocolo de comunicações middleware e desenvolvimento de aplicações web como uma cebola com várias camadas (cada camada de middleware WSGI ser desenvolvida de forma independente em todo o quadro). web2py não

adotar essa estrutura internamente. Isto é porque nós sentimos a funcionalidade de um núcleo de quadros (erros de manipulação de cookies de sessão, transações, expedição) pode ser melhor otimizado para velocidade e segurança, se eles são tratados por uma camada única e abrangente.

Ainda web2py permite que você use aplicativos terceiros eo middleware WSGI de três maneiras (e seus combinações):

- Você pode editar o arquivo "wsgihandler.py" e incluem quaisquer terceiros middleware WSGI.
- Você pode conectar terceiros middleware WSGI a qualquer ação específica em seus aplicativos.
- Você pode chamar um terceiro app WSGI de suas ações.

A única limitação é que você não pode usar middleware de terceiros para substituir principais funções web2py.

5.22.1 Externas Middleware

Considere o "wsgibase.py" arquivo:

```

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

```

Quando o login é definida para True, `gluon.main.wsgibase` é envolvido pelo `gluon.main.appfactory` função middleware. Ele fornece o arquivo de log para "httpserver.log". De maneira semelhante você pode adicionar qualquer middleware de terceiros. Referimo-nos a documentação oficial WSGI para mais

detalhes.

5.22.2 Interno Middleware

Dado qualquer ação de seus controladores (por exemplo, `index`) e middleware qualquer aplicativo de terceiros (por exemplo, `MyMiddleware`, que converte a saída para maiúsculas, você pode usar um decorador `web2py` aplicar o middleware para essa ação. Aqui está um exemplo:

```

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'
```

Não podemos prometer que todas as middleware de terceiros irá trabalhar com esse mecanismo.

5.22.3 Chamando WSGI Applications

É fácil ligar para aplicação WSGI de uma ação `web2py`. Aqui está um exemplo:

```

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()
```

Neste caso, a ação solicita `index` `test_wsgi_app` e escapa o valor retornado antes de devolvê-lo. Observe que `index` não é em si um app WSGI e deve usar o normal web2py API (como `response.write` para gravar o socket).

6

As vistas

web2py usa Python para seus modelos, controladores e pontos de vista, embora ele usa uma sintaxe ligeiramente modificada Python em vista a permitir que o código mais legível, sem impor quaisquer restrições sobre o uso adequado Python.

O propósito de um ponto de vista é o código embed (python) em um documento HTML. Em geral, isso coloca alguns problemas:

- Como deve ser precedido de código incorporado?
- Deve basear-se em recuo Python ou regras HTML?

web2py {{ ... }} usa para escapar Python código embutido em HTML. A vantagem de usar chaves em vez de colchetes é que é transparente para todos os editores HTML comum. Isto permite ao desenvolvedor usar esses editores para criar exibições web2py.

Uma vez que o desenvolvedor está embutindo Python em código HTML, o documento deve ser recuado de acordo com as regras do HTML, e não regras Python. Portanto, nós permitimos que não endentada Python dentro das tags

`{{ ... }}`. Desde Python normalmente usa indentação para delimitar blocos de código, precisamos de uma maneira diferente de delimitá-los, é por isso que o modelo web2py língua faz uso da palavra-chave `pass` Python.

Um bloco de código começa com uma linha que termina com uma vírgula e termina com uma linha começando com `pass`. O `pass` palavra-chave não é necessário quando o fim do bloco é óbvio a partir do contexto.

Aqui está um exemplo:

```
1 {{  
2 if i == 0:  
3     response.write('i is 0')  
4 else:  
5     response.write('i is not 0')  
6 pass  
7 }}
```

Note que o `pass` é uma palavra-chave Python, não uma palavra-chave web2py. Alguns editores de Python, como o Emacs, use a palavra-chave `pass` para significar a divisão de blocos e usá-lo para o código re-indent automaticamente.

O modelo web2py língua faz exatamente o mesmo. Quando encontra algo como:

```
1 <html><body>  
2 {{for x in range(10):}} {{=x}} hello<br />{{pass}}  
3 </body></html>
```

se traduz em um 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` escreve para o `response.body`.

Quando há um erro em uma visão web2py, o relatório de erro mostra o código gerado não ver, o ponto de vista real como escrito pelo desenvolvedor. Isso ajuda o desenvolvedor depurar o código, destacando o código real que é executado (o que é algo que pode ser depurado com um editor de HTML ou o inspetor DOM do navegador).

Além disso, note que:

```
1 {{=x}}
```

gera

```
1 response.write(x)
```

Variáveis injetadas no HTML desta forma são escapadas por padrão. A fuga é ignorada se `x` XML é um objeto, mesmo que `escape` está definido para `True`.

Aqui está um exemplo que introduz o ajudante `H1`:

```
1 {{=H1(i)}}
```

que é traduzido para:

```
1 response.write(H1(i))
```

Após a avaliação, o objeto `H1` e seus componentes são recursivamente serializados, escapados e escritos ao corpo da resposta. As tags HTML geradas pelo `H1` e interior não escapamos. Este mecanismo garante que todo o texto e somente texto — exibidos na página web é sempre escapado, evitando assim as vulnerabilidades XSS. Ao mesmo tempo, o código é simples e fácil de depurar.

O `response.write(obj, escape=True)` método leva dois argumentos, o objeto a ser escrito e se ela tem que ser escapado (conjunto de `True` por padrão). Se `obj.xml()` tem um método, ele é chamado e o resultado gravado no corpo da resposta (o argumento do `escape` é ignorado). Caso contrário, ele usa o método `__str__` do objeto para serializar e, se o argumento de `escape` é `True`,

lhe escapa. Todos os objetos internos auxiliar (`H1` no exemplo) são objetos que sabe como serializar-se através do método `.xml()`.

Isso tudo é feito de forma transparente. Você nunca precisa (e não deve) chamar o método `response.write` explicitamente.

6.1 Sintaxe Básica

O modelo web2py linguagem suporta todas as estruturas de controle Python. Aqui vamos dar alguns exemplos de cada um deles. Eles podem ser aninhados de acordo com a prática de programação usual.

6.1.1 `for...in`

Nos modelos que você pode fazer um loop em cima de qualquer objeto iteráveis:

```
1 {{items = ['a', 'b', 'c']}}
2 <ul>
3 {{for item in items:}}<li>{{=item}}</li>{{pass}}
4 </ul>
```

que produz:

```
1 <ul>
2 <li>a</li>
3 <li>b</li>
4 <li>c</li>
5 </ul>
```

Aqui `item` iteráveis é qualquer objeto como uma lista de Python, Python tupla, ou linhas de objeto, ou qualquer objeto que é executado como um iterador. Os elementos apresentados são os primeiros seriados e escapou.

6.1.2 while

Você pode criar um loop usando a palavra-chave ao mesmo tempo:

```

1 {{k = 3}}
2 <ul>
3 {{while k > 0:}}<li>{{=k}}{{k = k - 1}}</li>{{pass}}
4 </ul>
```

que produz:

```

1 <ul>
2 <li>3</li>
3 <li>2</li>
4 <li>1</li>
5 </ul>
```

6.1.3 if...elif...else

Você pode usar as cláusulas condicionais:

```

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 produz:

```

1 <h2>
2 45 is odd
3 </h2>
```

Desde que, é óbvio que `else` fecha o primeiro bloco `if`, não há necessidade de uma declaração `pass`, e usando um seria incorreto. No entanto, você deve fechar explicitamente o bloco com uma `else pass`.

Lembre-se que em Python é escrito "else if" elif como no seguinte exemplo:

```
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>
```

Produz:

```
1 <h2>  
2 64 is divisible by 4  
3 </h2>
```

6.1.4 try...except...else...finally

Também é possível usar instruções try...except nas vistas com uma advertência. Considere o seguinte exemplo:

```
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}}
```

Ele vai produzir a seguinte saída:

```
1 Hello  
2 division by zero  
3 <br />
```

Este exemplo ilustra que todas as saídas geradas antes de ocorrer uma exceção é processado (incluindo a saída que precedeu a exceção) no interior de um bloco try. "Hello" é escrito, porque antecede a exceção.

6.1.5 def...return

O modelo web2py linguagem permite ao desenvolvedor definir e implementar funções que podem retornar qualquer objeto Python ou um texto / string html. Aqui vamos considerar dois exemplos:

```

1 {{def itemize1(link): return LI(A(link, _href="http://" + link))}}
2 <ul>
3 {{=itemize1('www.google.com')}}
4 </ul>
```

produz o seguinte resultado:

```

1 <ul>
2 <li><a href="http://www.google.com">www.google.com</a></li>
3 </ul>
```

A função retorna um objeto auxiliar `itemize1` que é inserido no local onde a função é chamada.

Consideremos agora o seguinte código:

```

1 {{def itemize2(link):}}
2 <li><a href="http://{{=link}}">{{=link}}</a></li>
3 {{return}}
4 <ul>
5 {{itemize2('www.google.com')}}
6 </ul>
```

Elá produz exactamente o mesmo resultado que acima. Neste caso, o `itemize2` função representa um pedaço de HTML que está indo para substituir a marca web2py onde a função é chamada. Observe que não existe '=' na frente da chamada para `itemize2`, uma vez que a função não retornar o texto, mas escreve-lo diretamente na resposta.

Há uma advertência: as funções definidas dentro de uma visão deve terminar com uma instrução de retorno, ou o recuo automático falhará.

6.2 HTML Helpers

Considere o seguinte código em uma view:

```
1 {{=DIV('this', 'is', 'a', 'test', _id='123', _class='myclass')}}
```

ele é processado como:

```
1 <div id="123" class="myclass">thisisatest</div>
```

DIV é uma classe auxiliar, ou seja, algo que pode ser usado para construir HTML programaticamente. Ela corresponde à tag HTML `<div>`. Argumentos posicionais são interpretados como objetos contidos entre as marcas de abertura e de encerramento. Nomeados argumentos que começam com um sublinhado são interpretados como atributos tag HTML (sem o sublinhado). Alguns colaboradores também têm argumentos nomeados que não começam com sublinhado, estes argumentos são de marcas específicas.

O seguinte conjunto de helpers:

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 pode ser usada para construir expressões complexas que podem ser serializados para (49) XML (50). Por exemplo:

```
1 {{=DIV(B(I("hello ", "<world>"))), _class="myclass")}}
```

é processado:

```
1 <div class="myclass"><b><i>hello &lt;world&gt;</i></b></div>
```

O mecanismo de ajudantes em web2py é mais que um sistema para gerar HTML sem concatenar strings. Ele fornece uma representação do lado do servidor do Document Object Model (DOM).

Objetos componentes podem ser referenciados através de sua posição, e atuar como ajudantes listas com relação aos seus 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 ajudantes pode ser referenciado pelo nome, e auxiliares como dicionários agir com relação aos seus 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>
```

6.2.1 XML

XML é um objeto usado para encapsular o texto que não deve ser ignorado. O texto pode ou não conter XML válido. Por exemplo, poderia conter JavaScript.

O texto neste exemplo é escaparam:

```

1 >>> print DIV("<b>hello</b>")
2 &lt;b&gt;hello&lt;/b&gt;;
```

usando o XML você pode evitar fuga:

```

1 >>> print DIV(XML("<b>hello</b>"))
2 <b>hello</b>
```

Às vezes você quer tornar HTML armazenado em uma variável, mas as tags HTML pode conter inseguro como scripts:

```
1 >>> print XML('<script>alert("unsafe!")</script>')
2 <script>alert("unsafe!")</script>
```

Un-entrada escapou executável como esta (por exemplo, entrou no corpo de um comentário em um blog) não é seguro, porque pode ser usado para gerar Cross Site Scripting (XSS) contra outros visitantes da página.

O `XML` pode auxiliar web2py sanear o nosso texto para evitar as injecções e escapar todas as marcas, exceto aquelas que você permitir. Aqui está um exemplo:

```
1 >>> print XML('<script>alert("unsafe!")</script>', sanitize=True)
2 &lt;script&gt;alert("unsafe!")&lt;/script&gt;
```

Os construtores `XML`, por omissão, considerar o conteúdo de algumas marcas e alguns de seus atributos de segurança. Você pode substituir o padrão usando os argumentos opcionais `permitted_tags` e `allowed_attributes`. Aqui estão os valores padrão dos argumentos opcionais do ajudante `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']})
```

6.2.2 Built-in Helpers

A

Esse auxiliar é utilizado para criar links.

```
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

Este assistente faz o seu conteúdo em negrito.

```
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>
```

BODY

Este assistente faz com que o corpo de uma página.

```
>>> print BODY('<hello>', XML('<b>world</b>'), _bgcolor='red')
<body bgcolor="red">&lt;hello&gt;<b>world</b></body>
```

CENTER

Esse auxiliar seus centros de conteúdo.

```
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>
```

CODE

Esse auxiliar realiza o realce de sintaxe para a linguagem Python, C, C++, HTML e código web2py, e é preferível para listagens de código PRE. CODE também possui a capacidade de criar links para a documentação da API web2py.

Aqui está um exemplo de destacar seções de código 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>
```

Aqui está um exemplo semelhante para HTML

```
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>
```

Estes são os argumentos padrão para o ajudante CODE:

```
1 CODE("print 'hello world'", language='python', link=None, counter=1, styles={})
```

Os valores suportados para o argumento language "python", "html_plain", "c", "cpp", "web2py", and "html". The "html" language interprets {{ and }} tags as "web2py" code, while "html_plain" não são.

Se um valor link é especificado, a aplicação, por exemplo "/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".

O argumento counter é usado para a numeração de linha. Ele pode ser configurado para qualquer um dos três valores diferentes. Pode ser None sem números de linha, um valor numérico que especifica o número inicial, ou

uma string. Se o contador é definido como uma string, ela é interpretada como um alerta, e não há números de linha.

DIV todos os auxiliares para além da XML são derivados do DIV e herdar seus métodos básicos.

```
1 >>> print DIV('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <div id="0" class="test">&lt;hello&gt;<b>world</b></div>
```

EM

Enfatiza o seu conteúdo.

```
1 >>> print EM('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <em id="0" class="test">&lt;hello&gt;<b>world</b></em>
```

FIELDSET

Isto é usado para criar um campo de entrada, juntamente com o seu rótulo.

```
1 >>> print FIELDSET('Height:', INPUT(_name='height'), _class='test')
2 <fieldset class="test">Height:<input name="height" /></fieldset>
```

FORM

Este é um dos auxiliares mais importantes. Na sua forma simples, apenas faz uma marca de `<form>...</form>`, mas porque são objetos auxiliares e ter conhecimento de que eles contêm, eles podem processar formulários apresentados (por exemplo, realizar a validação dos campos). Isto será discutido em detalhes no 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>
```

O "enctype" is "multipart/form-data" por padrão.

O construtor de um FORM, SQLFORM e, também pode ter um argumento especial

chamado . Quando um dicionário é passado como , seus itens são convertidos em campos de entrada "hidden". Por exemplo:

```
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

Esses ajudantes são para os títulos e subtítulos n °:

```
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>
```

HEAD

Para marcar a cabeça de uma página HTML.

```
1 >>> print HEAD(TITLE('<hello>', XML('<b>world</b>')))  
2 <head><title>&lt;hello&gt;<b>world</b></title></head>
```

HTML

Este assistente é um pouco diferente. Além de fazer as tags <html>, que antecede a marca com um 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>
```

O ajudante HTML também tem alguns argumentos opcionais adicionais, que terão o seguinte padrão:

```
1 HTML(..., lang='en', doctype='transitional')
```

doctype onde pode ser "rigoroso", "transição", conjunto de quadros »," HTML5, ou uma seqüência completa doctype.

XHTML

XHTML é semelhante ao HTML, mas ele cria um doctype XHTML vez.

```
1 XHTML(..., lang='en', doctype='transitional', xmlns='http://www.w3.org/1999/xhtml')
```

doctype onde pode ser "rigoroso", "transição", frame-set', ou uma seqüência completa doctype.

INPUT

Cria uma tag <input.../>. Uma marca de entrada não pode conter outras tags, e é fechada por /> em vez de >. A marca de entrada tem um _type atributo opcional que pode ser configurado para "text" (the default), "submit", "checkbox", or "radio".

```
1 >>> print INPUT(_name='test', _value='a')
2 <input value="a" name="test" />
```

Ele também tem um especial argumento opcional chamado "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", substitui o anterior a este último:

```
1 >>> print INPUT(_name='test', _value='a', value='b')
2 <input value="b" name="test" />
```

Para INPUT botões seletiva define o 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
```

e similarmente para checkboxes:

```
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

Este auxiliar inclui outra página da Web na página atual. A URL da página não for especificada através do atributo "_src".

```
1 >>> print IFRAME(_src='http://www.web2py.com')
2 <iframe src="http://www.web2py.com"></iframe>
```

LABEL

Ele é usado para criar uma tag LABEL para um 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

Faz um item da lista e deve ser contido em um tag UL ou 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>
```

LEGEND

Ele é usado para criar uma marca de legenda para um campo em um formulário.

```
1 >>> print LEGEND('Name', _for='somefield')
2 <legend for="somefield">Name</legend>
```

META

Para ser usado para a construção de marcas na cabeça META HTML. Por exemplo:

```
1 >>> print META(_name='security', _content='high')
2 <meta name="security" content="high" />
```

MARKMIN

Implementa o markmin sintaxe wiki. Ele converte a entrada de texto em html de saída de acordo com as regras markmin descrito no exemplo abaixo:

```
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>
```

A sintaxe markmin é descrita neste arquivo que acompanha web2py:

¹ <http://127.0.0.1:8000/examples/static/markmin.html>

e alguns exemplos são mostrados no capítulo 13, no contexto de plugin_wiki MARKMIN que utiliza extensivamente.

OBJECT

Usado para incorporar objetos (por exemplo, um jogador de flash) no 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>
```

0L

Fica para a lista ordenada. A lista deve conter tags LI. OL argumentos que não são objetos LI são automaticamente incluídos em tags

```
>>> print OL('<hello>', XML('<b>world</b>'), _class='test', _id=0)
<ol id="0" class="test"><li>&lt;hello&gt;</li><li><b>world</b></li></ol>
```

ON

Este aqui é para compatibilidade com versões anteriores e é simplesmente um alias para True. É usado exclusivamente para caixas e obsoleto desde True é mais Pythonica.

```
1 >>> print INPUT(_type='checkbox', _name='test', _checked=ON)
2 <input checked="checked" type="checkbox" name="test" />
```

OPTGROUP

Tro grupo permite-lhe várias opções em um SELECT e é útil para personalizar os 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>
```

OPTION

Isso só deve ser usado como parte de um SELECT / combinação de opções.

```
1 >>> print OPTION('<hello>', XML('<b>world</b>'), _value='a')
2 <option value="a">&lt;hello&gt;<b>world</b></option>
```

Como no caso do INPUT, web2py fazer uma distinção 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

Isto é para a marcação de um parágrafo.

```
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

Gera um <pre>...</pre> tag para exibir texto pré-formatado. O ajudante CODE

é geralmente preferível para listagens 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

Trata-se de incluir um link ou script, como JavaScript. O conteúdo entre as tags é processado como um comentário HTML, para o benefício dos antigos navegadores realmente.

```
1 >>> print SCRIPT('alert("hello world");', _language='javascript')
2 <script language="javascript"><!--
3 alert("hello world");
4 //--></script>
```

SELECT

Cria uma tag <select>...</select>. Isso é usado com o auxiliar OPTION. Estes argumentos SELECT que não são objetos OPTION são automaticamente convertidas em opções.

```
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

Semelhante ao DIV, mas usados para marcar inline (em vez de bloco) de conteúdo.

```
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>
```

STYLE

Semelhante ao script, mas utilizada para incluir ou link do código CSS. Aqui o CSS está incluído:

```
1 >>> print STYLE(XML('body {color: white;}'))  
2 <style><!--  
3 body { color: white }  
4 //--></style>
```

e aqui está vinculado:

```
1 >>> print STYLE(_src='style.css')  
2 <style src="style.css"><!--  
3 //--></style>
```

TABLE, TR, TD

Estas tags (junto com o THEAD opcional, TBODY e TF0OTER ajudantes) são utilizados para construir tabelas 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>
```

espera TR TD conteúdo, argumentos que não são objetos TD são convertidos automaticamente.

```
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>
```

É fácil converter um array Python em uma tabela HTML usando * Python função notação argumentos, que mapeia elementos da lista de argumentos da função posicional.

Aqui, vamos fazê-lo linha por linha:

```
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>
```

Aqui nós fazemos todas as linhas de uma só 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

Isto é usado para marcar as linhas contidas no corpo da tabela, ao contrário de linhas de cabeçalho ou rodapé. É 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

Este assistente faz uma tag `<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>
```

A única ressalva é que o seu "value" opcional substitui o seu conteúdo (HTML interior)

```
1 >>> print TEXTAREA(value=<hello world>, _class="test")
2 <textarea class="test" cols="40" rows="10">&lt;hello world&gt;</textarea>
```

TFOOT

Isto é usado para marcar as linhas no rodapé da tabela.

```
1 >>> print TF0OT(TR(TD('<hello>')), _class='test', _id=0)
2 <tfoot id="0" class="test"><tr><td>&lt;hello&gt;</td></tr></tfoot>
```

TH

Isso é usado em vez de TD em cabeçalhos de tabela.

```
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

Isto é usado para marcar as linhas do cabeçalho da tabela.

```
1 >>> print THEAD(TR(TD('<hello>')), _class='test', _id=0)
2 <thead id="0" class="test"><tr><td>&lt;hello&gt;*</td></tr></thead>
```

TITLE

Isto é usado para marcar o título de uma página em um cabeçalho HTML.

```
1 >>> print TITLE('<hello>', XML('<b>world</b>'))
2 <title>&lt;hello&gt;<b>world</b></title>
```

TR

Tags uma linha da tabela. Ele deve ser processado dentro de uma tabela e incluir tags `<td>...</td>`. TR argumentos que não são objetos de TD será automaticamente convertido.

```
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

Tags texto como máquina de escrever (monoespaciada) de texto.

```
1 >>> print TT('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <tt id="0" class="test">&lt;hello&gt;<b>world</b></tt>
```

UL

Significa uma lista não ordenada e deve conter itens LI. Se o seu conteúdo não é etiquetado como LI, a UL faz isso automaticamente.

```
1 >>> print UL('<hello>', XML('<b>world</b>'), _class='test', _id=0)
2 <ul id="0" class="test"><li>&lt;hello&gt;*</li><li><b>world</b></li></ul>
```

6.2.3 Helpers personalizada

Às vezes você precisa para gerar tags XML personalizadas. web2py fornece TAG, um gerador de marca universal.

```
1 {{=TAG.name('a', 'b', _c='d')}}
```

gera o seguinte XML

```
1 <name c="d">ab</name>
```

"a" and "b" and "d" Argumentos são automaticamente escapou; usar o helper XML para reprimir esse comportamento. Usando o TAG você pode gerar tags HTML / XML ainda não fornecidos pela API. Tags podem ser aninhadas e são serializados com um equivalente de sintaxe `str()`. é:

```
1 {{=TAG['name']('a', 'b', c='d')}}
```

Observe que TAG é um objeto, e TAG.name ou TAG['name'] é uma função que retorna uma classe auxiliar temporário.

MENU

O ajudante MENU leva uma lista de listas da forma de `response.menu` (como descrito no Capítulo 4) e gera uma estrutura em árvore usando listas não ordenadas representando o menu. Por exemplo:

```
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 item do menu pode ter um quarto argumento que é um submenu aninhado (e assim por diante recursivamente):

```
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>
```

O ajudante tem MENU os seguintes argumentos opcionais:

- padrão `_class`: para "web2py-menu web2py-menu-vertical" e define a classe dos elementos exteriores UL.
- padrão `ul_class`: para "web2py-menu-vertical" e define a classe dos elementos internos da UL.
- padrão `li_class`: para "web2py-menu-expand" e define a classe dos elementos internos LI.

O "base.css" da aplicação andaime compreende os seguintes tipos básicos de menus: "web2py-menu web2py-menu-vertical" and "web2py-menu web2py-menu-horizontal".

6.3 BEAUTIFY BEAUTIFY

é usada para construir representações de objetos HTML compostos, incluindo listas, tuplas e dicionários:

```
1 {{=BEAUTIFY({"a": ["hello", XML("world")], "b":(1, 2)})}}
```

BEAUTIFY retorna um objeto XML, como serializável para XML, com um belo olhar representação do seu argumento de construtor. Neste caso, a representação XML de:

```
1 {"a": ["hello", XML("world")], "b":(1, 2)}
```

processará 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>
```

6.4 Server-side DOM e Parsing

6.4.1 elementos

O ajudante DIV e todos os auxiliares de proporcionar um derivado de três métodos de pesquisa: `element` e `elements`. `element` retorna o primeiro elemento filho que combinam com uma condição especificada (ou Nenhum se não jogo). `elements` retorna uma lista de todas as crianças de correspondência.

`element` e `elements` usar a mesma sintaxe para especificar a condição de correspondência, que permite três possibilidades que podem ser misturados e combinados: as expressões do jQuery, como, por coincidir com o valor do atributo exato, jogo usando expressões regulares.

Aqui está um exemplo simples:

```

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>
```

O argumento un-named de elementos é uma cadeia que pode conter: o nome de uma marca, a identificação de uma tag precedida por um símbolo de libra, a classe precedido por um ponto, o valor de um atributo explícito entre colchetes.

Aqui estão 4 maneiras equivalentes a busca da marca anterior, 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')
```

Aqui estão quatro maneiras equivalentes à pesquisa anterior, a marca de classe:

```
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')
```

Qualquer atributo pode ser usado para localizar um elemento (não apenas `id` e `class`), incluindo múltiplos atributos (o elemento função pode ter vários argumentos nomeados), mas apenas o primeiro elemento correspondente será devolvido.

Usando o "div#target" sintaxe jQuery é possível especificar vários critérios de pesquisa separados por um espaço:

```
1 >>> a = DIV(SPAN('a', _id='t1'),div('b',_class='c2'))
2 >>> d = a.elements('span#t1', 'div#c2')
```

ou equivalentemente

```
1 >>> a = DIV(SPAN('a', _id='t1'),div('b',_class='c2'))
2 >>> d = a.elements('span#t1', 'div#c2')
```

Se o valor do atributo de pesquisa é especificado usando um argumento de nome, pode ser uma string ou uma expressão regular:

```
1 >>> a = DIV(SPAN('a', _id='test123'),div('b',_class='c2'))
2 >>> d = a.elements('span',_id=re.compile('test\d{3}'))
```

Um argumento especial chamado do DIV (e derivados) ajudantes é `find`. Ele pode ser usado para especificar um valor de pesquisa ou uma pesquisa de expressão regular no conteúdo do texto da etiqueta. Por exemplo:

```
1 >>> a = DIV(SPAN('abcde'),div('fghij'))
2 >>> d = a.elements(find='bcd')
```

```

3 >>> print d[0]
4 <span>abcde</span>

```

ou

```

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>

```

6.4.2 parent

`parent` retorna o pai do actual elemento.

```

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>

```

6.4.3 flatten

O acharar recursivo método serializa o conteúdo das crianças de um determinado elemento em texto normal (sem tags):

```

1 >>> a = DIV(SPAN('this',DIV('is',B('a'))),SPAN('test'))
2 >>> print a.flatten()
3 thisisatest

```

Flatten pode ser passado um argumento opcional, `render`, ou seja, uma função que processa / achata o conteúdo usando um protocolo diferente. Aqui está um exemplo para serializar algumas tags em Markmin sintaxe wiki:

```

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]]
```

No momento da escrita nós fornecemos `markmin_serializer` e `markdown_serializer`.

6.4.4 Análise

O objeto TAG também é um XML / parser HTML. Pode ler o texto e converter em uma estrutura de árvore de ajudantes. Isso permite a manipulação usando a API anterior:

```
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>
```

6.5 Page Layout

Exibições pode estender e incluir outros pontos de vista em uma estrutura de árvore.

Por exemplo, podemos pensar em um "index.html" extends "layout.html" and includes "body.html" vista. "layout.html" may include "header.html" and "footer.html" Ao mesmo tempo.

A raiz da árvore é o que chamamos de exibição de layout. Assim como qualquer outro modelo de arquivo HTML, você pode editá-lo usando a interface web2py administrativa. O "layout.html" nome do arquivo é apenas uma convenção.

Aqui está uma página minimalista que se estende a vista "layout.html" view and includes the "page.html":

```

1 {{extend 'layout.html'}}
2 <h1>Hello World</h1>
3 {{include 'page.html'}}

```

O arquivo de layout estendida deve conter uma diretiva {{include}}, algo como:

```

1 <html><head><title>Page Title</title></head>
2   <body>
3     {{include}}
4   </body>
5 </head>

```

Quando a exibição é chamada, o prolongado (layout) de vista é carregados, e a visão de chamada substitui a diretiva {{include}} dentro do layout. O processamento continua recursivamente até que todos os ampliam e incluem as directivas foram processadas. O modelo resultante é então traduzido em código Python.

extend e include são especiais modelo directivas não, os comandos Python.

Layouts são usados para encapsular a uniformização da página (cabeçalhos, rodapés, menus), e apesar de não serem obrigatórios, eles vão fazer a sua aplicação mais fácil de escrever e manter. Em particular, sugerimos a escrita layouts que se aproveitam das variáveis que pode ser definido no controlador. Usando essas variáveis conhecidas ajudará a fazer seus layouts intercambiáveis:

```

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

```

Exceto para menu e files estas são todas as cordas e seu significado deveria ser óbvio.

`response.menu` menu é uma lista de tuplas 3 ou 4-tuplas. Os três elementos são: o nome do link, um booleano representando se o link está ativo (é o elo atual), ea URL da página vinculada. Por exemplo:

```
1 response.menu = [('Google', False, 'http://www.google.com', []),
2                   ('Index', True, URL('index'), [])]
```

da tupla quarto elemento é opcional sub-menu.

`response.files` é uma lista de arquivos CSS e JS que são necessários para sua página.

Também recomendamos que você usa:

```
1 {{include 'web2py_ajax.html'}}
```

na cabeça HTML, uma vez que irá incluir as bibliotecas jQuery e definir algumas funções compatíveis com versões anteriores para efeitos especiais JavaScript e Ajax. "web2py_ajax.html" `response.meta` inclui as marcas no ponto de vista, base jQuery, o calendário jQuery e CSS inclui todos os necessários e JS `response.files`.

6.5.1 Default Layout da Página

Aqui está um "views/layout.html" mínimo que vem com o **welcome** web2py andaimes pedido e qualquer nova aplicação terá um layout padrão similar:

```
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   <head>
5     <title>{{=response.title or request.application}}</title>
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) }}
```

```
12     {{response.files.append(URL(request.application,'static','base.css'))}}
13     {{response.files.append(URL(request.application,'static','superfish.js'))}}
14
15     {{##### include web2py specific js code (jquery, calendar, form stuff) --}}
16     {{include 'web2py_ajax.html'}}}
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                     <h2>
31                         {{=response.subtitle or 'response.subtitle'}}
32                     </h2>
33                 </div>
34                 <div class="ez-box" id="statusbar">
35
36                     {{##### superfish menu --}}
37                     {{=MENU(response.menu,_class='sf-menu')}}}
38
39                     <script>
40                         jQuery(document).ready(function(){
41                             jQuery('ul.sf-menu').superfish({delay:400});});
42                     </script>
43                 </div>
44                 <div class="ez-wr">
45                     <div class="ez-fl ez-negmx">
46                         <div class="ez-box" id="left_sidebar">{{##### unused space}}</div>
47                     </div>
48                     <div class="ez-fl ez-negmr">
49                         <div class="ez-box" id="content">{{include}}</div>
50                     </div>
51                     <div class="ez-last ez-oh">
52                         <div class="ez-box" id="right_sidebar">{{##### unused space}}</div>
53                     </div>
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>
```

Existem algumas características desse layout padrão que torna muito fácil de usar e personalizar:

- {{#...}} são comentários especiais que não aparecem no HTML da página.
- Ele exibe tanto `response.title` e `response.subtitle`, que pode ser definido em um modelo. Se eles não estão definidas, adota o nome do aplicativo como o título
- Isso inclui o arquivo `web2py_ajax.html` no cabeçalho
- Ele requer dois arquivos explicitamente: "base.css" and "superfish.js". O primeiro contém o CSS para a página completa e é muito bem documentado e personalizável. Este último contém o JS para o padrão menu em cascata.
- O {{=auth.navbar(...)}} apresenta boas-vindas para o usuário atual e links para funções como autenticação de login logout, registrar, alterar senha, etc dependendo do contexto. É colocado em um {{try:}}...{{except:pass}} auth caso é indefinido.
- O {{=MENU(response.menu)}} apresenta a estrutura de menus como `...`.
- Existe um script explícito para ativar o menu Superfish cascata e pode ser retirado se não forem necessárias.
- {{include}} é substituído pelo conteúdo da vista que se estende quando a página é processada.
- Por padrão, ele utiliza um layout de três colunas de página e os usos os seguintes IDs DIV: "header", "left_sidebar", "content", and "right_sidebar", "footer" although the provided "base.css" define as larguras das barras laterais para zero.
- Ele usa a convenção ez.css CSS para layout de nomeação definidos no (51) ref. Em particular, usa o número de Layout 3. O ez.css minimizado está incluído no "base.css".

6.5.2 Personalizar o layout padrão

Personalizar o layout padrão sem edição é muito fácil por causa do arquivo "static/base.css" é muito bem documentado.

Em particular, é organizado nos seguintes sub-seções:

- ez.css
- reset tags comum
- escolher fontes padrão
- escolher o estilo link
- adicionar a linha de fundo da tabela as linhas
- etiquetas em negrito e, ocasionalmente, centrado
- envidar todos os campos de entrada do mesmo tamanho
- adicionar uma separação adequada entre h1-h6 e texto
- sempre recuar a primeira linha e adicionar o espaço abaixo de parágrafos
- balas e números de estilo e travessão
- forma e preenchimento da tabela
- blocos de código
- esquerdo e direito de preenchimento do texto citado
- alinhamento de layout de página, largura e espaçamento (alterar isso para os espaços)
- larguras de coluna (mudar isto para usar left_sidebar e right_sidebar)
- imagens de fundo e cores (para alterar essa cores)
- estilo de menu (para superfish.js)
- web2py específicas (. flash. erro)

Para alterar o left_sidebar, conteúdo, larguras right_sidebar, basta editar a parte do "base.css":

```
1 /***** column widths *****/
2 #left_sidebar { width: 0px; }
3 #content { width: 840px; }
4 #right_sidebar { width: 0px; }
```

Para alterar as cores e imagens de fundo, basta editar a parte que se segue:

```
1 /***** background images and colors *****/
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; }
```

O menu é construído de uma forma de cor neutra, mas você pode mudar isso também.

Claro que você também pode substituir completamente os arquivos "layout.html" and "base.css" com a sua própria.

6.6 Funções em Views

Considere esta "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>
```

e este estendendo vista

```

1 {{def mysidebar():}}
2 my new sidebar!!!
3 {{return}}
4 {{extend 'layout.html'}}
5 Hello World!!!
```

Observe a função definida antes da instrução {{extend...}}. Observe também a função está incluída no modo de exibição estendido sem o prefixo =.

O código gera o seguinte 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>
```

Observe que as funções são definidas em HTML (embora eles também podem conter código Python), para que response.write é usado para gravar seu conteúdo (as funções não retornam o conteúdo). Esta é a razão pela qual o layout chama a função view utilizando {{mysidebar()}} ao invés de {{=mysidebar()}}. Funções definidas desta forma podem ter argumentos.

6.7 Blocos em Views

Outra maneira de fazer uma abordagem mais modular é usando o `{{block...}}`s e esse mecanismo é uma alternativa ao mecanismo discutido na seção anterior.

Considere esta "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>
```

e este estendendo vista

```
1 {{extend 'layout.html'}}
2 Hello World!!!
3 {{block mysidebar}}
4 my new sidebar!!!
5 {{end}}
```

Ele gera o seguinte 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>
```

Você pode ter vários blocos e se o bloco se apresenta a visão alargada, mas não na vista que se estende, o conteúdo da exibição estendida é utilizado.

6.8 Usando o Modelo de sistema para gerar e-mails

É possível usar o modelo do sistema para gerar e-mails. Por exemplo, considere a tabela do banco de dados

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

onde você quer enviar a cada pessoa no banco de dados a seguinte mensagem, armazenados em um arquivo visualizar "message.html":

```
1 Dear {{=person.name}},
2 You have won the second prize, a set of steak knives.
```

Você pode conseguir isso da seguinte maneira

```
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)
```

A maioria dos trabalhos é feita na declaração

```
1 response.render('message.html', context)
```

Ela torna o "file.html" with the variables defined in the dictionary "context" vista, e ele retorna um string com o texto de e-mail processado. O contexto é um dicionário que contém variáveis que serão visíveis para o arquivo de modelo.

Se a mensagem começa com <html> e </html> termina com o e-mail será um e-mail HTML.

O mesmo mecanismo que é usado para gerar o texto de e-mail também pode ser usado para gerar SMS ou qualquer outro tipo de mensagem com base em um modelo.

7

A camada de abstração

7.1 Dependências

web2py vem com um banco de dados Abstraction Layer (DAL), uma API que mapeia objetos Python em objetos de banco de dados, como consultas, tabelas e registros. A DAL gera dinamicamente o SQL em tempo real usando o dialeto especificado para o banco de dados back-end, de modo que você não tem que escrever o código SQL ou aprender diferentes dialetos SQL (SQL do termo é utilizado genericamente), ea aplicação será portável entre diferentes tipos de bancos de dados. No momento da redação deste texto, os bancos de dados suportados são SQLite (que vem com Python e, portanto, web2py), PostgreSQL, MySQL, a Oracle, MSSQL, Firebird, DB2, Informix e Ingres e (parcialmente) com o Google App Engine (GAE). GAME é tratado como um caso particular no capítulo 11.

A distribuição binária do Windows funciona fora da caixa com MySQL e SQLite. A distribuição binária Mac trabalha fora da caixa com SQLite. Para usar qualquer outro banco de dados back-end, executado a partir de fonte de distribuição e instale o driver apropriado para o back-end necessário.

Depois que o driver correto está instalado, inicie web2py da fonte, e vai encontrar o driver. Aqui está uma lista de drivers:

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 as seguintes classes que compõem o DAL:

DAL representa uma conexão de banco de dados. Por exemplo:

```
1 db = DAL('sqlite://storage.db')
```

Table representa uma tabela do banco de dados. Você não diretamente instanciar mesa, em vez disso, ele instancia `DAL.define_table`.

```
1 db.define_table('mytable', Field('myfield'))
```

Os métodos mais importantes de uma tabela são:

`.insert`, `.truncate`, `.drop`, `.import_from_csv_file` e.

Field representa um campo de banco de dados. Pode ser instanciado e passou como um argumento para `DAL.define_table`.

DAL Rows é o objeto retornado por uma base de dados escolha. Pode ser pensado como uma lista de linhas `Row`:

```
1 rows = db(db.mytable.myfield!=None).select()
```

Row contém valores de campo.

```

1 for row in rows:
2     print row.myfield

```

Query é um objecto que representa um "where" cláusula SQL:

```

1 myquery = (db.mytable.myfield != None) | (db.mytable.myfield > 'A')

```

Set é um objeto que representa um conjunto de registros. Seus métodos mais importantes são `count`, `select`, `update`, `delete` e. Por exemplo:

```

1 myset = db(myquery)
2 rows = myset.select()
3 myset.update(myfield='somevalue')
4 myset.delete()

```

Expression é algo parecido com um `orderby` ou `groupby` expressões. A classe `Field` é derivado da expressão. Aqui está um exemplo.

```

1 myorder = db.mytable.myfield.upper() | db.mytable.id
2 db().select(db.table.ALL, orderby=myorder)

```

7.2 Connection Strings

A conexão com o banco de dados é estabelecida através da criação de uma instância do objeto DAL:

```

1 >>> db = DAL('sqlite://storage.db', pool_size=0)

```

`db` não é uma palavra-chave, que é uma variável local que armazena o objeto de conexão DAL. Você é livre para dar-lhe um nome diferente. O construtor de DAL requer um único argumento, a cadeia de ligação. A seqüência de conexão é o único código web2py que depende de um banco de dados de back-end específico. Aqui estão alguns exemplos de seqüências de conexão para tipos específicos de bancos de dados de suporte de back-end (em todos os casos, assumimos o banco de dados está sendo executado a partir de localhost em sua porta padrão e é nomeado "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 no banco de dados SQLite é composto de um único arquivo. Se ele não existir, ele é criado. Este arquivo está bloqueado toda vez que ela é acessada. No caso do MySQL, PostgreSQL, MSSQL, Firebird, Oracle, DB2, Informix o "test" banco de dados deve ser criado fora web2py. Uma vez que a conexão é estabelecida, web2py vai criar, alterar e mesas drop adequadamente.

Também é possível definir a seqüência de conexão para `None`. Neste caso DAL não vai se conectar a qualquer banco de dados de back-end, mas a API ainda podem ser acessados para testes. Exemplos disto será discutido no Capítulo 7.

7.2.1 *Connection Pooling*

O segundo argumento do construtor DAL é o `pool_size`, o padrão é o.

Uma vez que ele é lento para estabelecer uma conexão de banco de dados para cada novo pedido. Para evitar isso, web2py implementa um mecanismo de pool de conexão. Quando uma conexão é estabelecida, após a página ter sido cumprida e concluída a transação, a conexão não está fechado, mas ele vai para uma piscina. Quando o pedido http próximo chega, web2py tenta pegar uma conexão do pool e usar esse para uma nova transação. Se não há conexões disponíveis no pool, uma nova conexão é estabelecida.

O parâmetro `pool_size` é ignorado pelo SQLite e do jogo.

Conexões nas piscinas são compartilhados entre os tópicos em seqüência, no sentido de que eles podem ser usados por dois, mas não threads simultâneos. Há apenas um pool para cada processo web2py.

Quando começa web2py, a piscina está sempre vazia. A piscina cresce ao mínimo entre o valor do `pool_size` eo número máximo de solicitações simultâneas. Isto significa que se `pool_size=10`, mas o nosso servidor não recebe mais de 5 pedidos em simultâneo, em seguida, o tamanho do pool real só vai crescer a 5. Se o `pool_size=0`, então o pool de conexão não é utilizado.

Pool de conexão é ignorado para SQLite, uma vez que não produziria qualquer benefício.

7.2.2 Falhas de conexão

Se web2py falha ao conectar ao banco de dados que espera um segundo e tenta novamente até 5 vezes antes de declarar um fracasso. Em caso de pool de conexão, é possível que uma conexão de pool que permanece aberta, mas não utilizada há algum tempo está fechado até o final do banco de dados. Graças à repetição característica web2py tenta restabelecer essas ligações abandonadas.

7.2.3 Bancos de dados replicados

O primeiro argumento do `DAL(...)` pode ser uma lista de URIs. Neste caso web2py tenta se conectar a cada um deles. O objetivo principal desta é a de lidar com múltiplos servidores de banco de dados e distribuir a carga entre eles). Aqui está um caso de uso típico:

```
1 db=DAL(['mysql://...1','mysql://...2','mysql://...3'])
```

Neste caso, a DAL tenta ligar para o primeiro e, em caso de falha, será que o segundo eo primeiro. Isto pode também ser usado distribuir a carga em uma configuração master-slave banco de dados. Falaremos mais sobre isso no capítulo 11, no contexto de escalabilidade.

7.3 *Palavras-chave reservadas*

Existe também um outro argumento que pode ser passado para o construtor DAL para verificar os nomes de tabela e nomes de coluna contra reservados palavras-chave SQL em bancos de dados alvo de back-end.

Este argumento é `check_reserved` eo padrão é `None`.

Esta é uma lista de seqüências que contêm os nomes de banco de dados adaptador back-end.

O nome do adaptador é o mesmo utilizado na cadeia de ligação DAL. Então, se você quiser verificar contra PostgreSQL e MSSQL em seguida a seqüência de conexão ficaria da seguinte forma:

```
1 db = DAL('sqlite://storage.db',
2           check_reserved=['postgres', 'mssql'])
```

A DAL verificará as palavras-chave na mesma ordem da lista.

Existem duas opções extra "all" and "common". Se você especificar tudo, ele irá verificar contra todas as palavras-chave conhecidas SQL. Se você especificar comum, ele irá verificar apenas palavras-chave comuns contra SQL como `SELECT`, `INSERT`, `UPDATE`, etc

Para suporte back-ends você também pode especificar se você gostaria de verificar contra as palavras-chave não reservadas SQL também. Neste caso, você poderia anexar ao nome do `_nonreserved`. Por exemplo:

```
1 check_reserved=['postgres', 'postgres_nonreserved']
```

O banco de dados backends seguintes palavras reservadas apoio verificação.

PostgreSQL	postgres(_nonreserved)
MySQL	mysql
FireBird	firebird(_nonreserved)
MSSQL	mssql
Oracle	oracle

7.4 DAL, Table, Field

A melhor maneira de entender a API DAL é tentar cada função de si mesmo. Isso pode ser feito de forma interativa através da shell web2py, embora, em última instância, o código DAL vai nos modelos e controladores.

Comece por criar uma conexão. Por uma questão de exemplo, você pode usar o SQLite. Nada muda nesta discussão, quando você alterar o mecanismo de back-end.

```
1 >>> db = DAL('sqlite://storage.db')
```

O banco de dados está agora ligado e que a conexão é armazenado no db variável global.

A qualquer momento você pode recuperar a seqüência de conexão.

```
1 >>> print db._uri
2 sqlite://storage.db
```

eo nome do banco de dados

```
1 >>> print db._dbname
2 sqlite
```

A seqüência de conexão é chamado de `_uri`, porque é um exemplo de um Uniform Resource Identifier.

A DAL permite conexões múltiplas com o mesmo banco de dados ou com bases de dados diferentes, mesmo bancos de dados de diferentes tipos. Por enquanto, vamos supor a presença de um único banco de dados uma vez que esta é a situação mais comum.

O método mais importante de uma DAL é `define_table`:

```
>>> db.define_table('person', Field('name'))
```

Elá define, armazena e retorna um objeto chamado `Table "person"` containing a field (column) "name". Este objeto também pode ser acessado via `db.person`, assim você não precisa pegar o valor de retorno.

7.5 Record Representação

É opcional, mas recomendada para especificar um formato de representação de registros:

```
>>> db.define_table('person', Field('name'), format='%(name)s')
```

ou

```
>>> db.define_table('person', Field('name'), format='%(name)s %(id)s')
```

ou mesmo as mais complexas usando uma função:

```
>>> db.define_table('person', Field('name'),
2       format=lambda r: r.name or 'anonymous')
```

O atributo formato será utilizado para dois propósitos:

- Para representar registros referenciados em select / opção drop-downs.
- Para definir o atributo `db.othertable.person.represent` referência para todos os campos desta tabela. Isto significa que SQLTABLE não irá mostrar as referências feitas por id, mas usará o formato de representação preferido.

Estes são os valores padrão de um construtor 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)

```

Nem todos eles são 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` define o comprimento máximo de um campo de "string", "password" or "upload". Se o `length` não é especificado um valor padrão é usado, mas o valor padrão não é garantido para ser compatível. *To avoid unwanted migrations on upgrades, we recommend that you always specify the length for string, password and upload fields.*
- `default` define o valor padrão para o campo. O valor padrão é usado quando executar uma inserção se um valor não é especificado explicitamente. Ele também é usado para preencher formulários pré-construídos a partir da tabela utilizando SQLFORM.
- `required` diz o DAL que não deve ser permitido inserir nesta tabela se um valor para este campo não é explicitamente especificada.
- `requires` é um validador ou uma lista de validadores. Isto não é usado pela DAL, mas é usado por SQLFORM. Os validadores padrão para os tipos de dados são mostrados na tabela a seguir:

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 requer e retorna valores como objetos Decimal, tal como definido no módulo Python decimal. SQLite não suporta o tipo decimal internamente de modo que nós tratamos ele como um double. A (n, m) eo número de dígitos antes e depois do ponto decimal.

Observe que requires=... é aplicada a nível de formas, required=True é aplicada a nível da DAL (insert), enquanto a notnull, unique e ondelete são aplicadas ao nível da base. Embora, por vezes, pode parecer redundante, é importante manter a distinção durante a programação com a DAL.

- Traduz-ondelete para o "ON DELETE" SQL statement. By default "CASCADE" diz o banco de dados que, quando se exclui um registro, ele também deve excluir todos os registros que se referem a ele.
- notnull=True traduz na instrução SQL "NOT NULL". Isso impede que o banco de dados inserindo valores nulos para o campo.
- unique=True traduz na instrução SQL "UNIQUE" e garante que os valores deste campo são únicos dentro da tabela. Ela é aplicada a nível de banco de dados.
- uploadfield só se aplica aos campos da pasta "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. Se o uploadfield é definido, então o arquivo é armazenado em um campo blob na mesma

mesa eo valor do `uploadfield` é o nome do campo blob. Isto será discutido em detalhes mais adiante no contexto da SQLFORM.

- `uploadseparate` se definido como True vai fazer upload de arquivos em diferentes subpastas da pasta "upload". Esta é otimizado para evitar muitos arquivos sob a mesma pasta / subpasta. ATENÇÃO: Você não pode alterar o valor do `uploadseparate` de True para False sem quebrar o sistema. web2py ou usa as subpastas separadas ou não. Mudar o comportamento depois que os arquivos foram enviados web2py impedirá de ser capaz de recuperar esses arquivos. Se isso acontecer, é possível mover arquivos e corrigir o problema, mas isso não é descrito aqui.
- `widget` deve ser um dos objetos widget disponíveis, incluindo widgets personalizados, como por exemplo: `SQLFORM.widgets.string.widget`. A lista de widgets disponíveis serão discutidos posteriormente. Cada tipo de campo tem um widget padrão.
- `label` é uma string (ou algo que possa ser serializado para uma seqüência de caracteres) que contém o rótulo para ser usado para este campo nos formulários gerados automaticamente.
- `comment` é uma string (ou algo que possa ser serializado para uma seqüência de caracteres) que contém um comentário associado com este domínio, e será exibido à direita do campo de entrada nos formulários gerados automaticamente.
- `writable` se um campo é gravável, pode ser editado em autoprodução criar e actualizar as formas.
- `readable` se um campo é legível, que será visível na forma `readonly`. Se um campo não é lido nem gravável, não será exibido em criar e actualizar as formas.
- `update` contém o valor padrão para este campo, quando o registro é atualizado.
- `compute` é uma função opcional. Se um registro é inserido, e não há valor para um campo que tem um atributo de calcular, o valor é calculado pela passagem do registro (como um dicionário) para calcular a função.
- `authorize` pode ser usado para exigir controle de acesso no campo correspondente, para os campos de "upload" só. Será discutida mais detalhadamente no contexto de autenticação e autorização.

- autodelete determina se o arquivo correspondente do upload deve ser excluída quando a referência gravar o arquivo é excluído. Para campos "upload" só.
- represent pode ser nenhum ou pode apontar para uma função que recebe um valor de campo e retorna uma representação alternativa para o valor do campo. Exemplos:

```
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 também são especiais. Por padrão, os dados binários são codificados em base64 antes de ser armazenado no campo de banco de dados real, e é decodificado quando extraído. Isso tem o efeito negativo do uso de 25% mais espaço do que o necessário em campos blob, mas tem duas vantagens. Em média, reduz a quantidade de dados transmitidos entre web2py e o servidor de banco de dados, e faz a comunicação independente de back-end específico convenções escapar.

Você pode consultar o banco de dados para tabelas existentes:

```
1 >>> print db.tables
2 ['person']
```

Você também pode consultar uma tabela para os campos existentes:

```
1 >>> print db.person.fields
2 ['id', 'name']
```

Não declarar um campo chamado "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" é uma chave primária. (Nota: o ID a partir de 1 está de volta específicos-end. Por exemplo, isto não se aplica ao Google App Engine (GAE).)

Opcionalmente, você pode definir um campo de type='id' e web2py irá usar este campo como auto-incremento campo id. Isto não é recomendado, exceto quando as tabelas do banco de dados acessar legado. Com algumas limitações, você também pode usar chaves primárias diferentes e isso é discutido na secção "Legacy databases and keyed tables".

Você pode consultar o tipo de uma tabela:

```
1 >>> print type(db.person)
2 <class 'gluon.sql.Table'>
```

e você pode acessar uma tabela de ligação DAL usando:

```
1 >>> print type(db['person'])
2 <class 'gluon.sql.Table'>
```

Da mesma forma você pode acessar os campos de seu nome em várias 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 um campo, você pode acessar a um conjunto de atributos em sua definição:

```
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
```

incluindo a sua tabela pai, TableName e conexão parent:

```
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
```

7.6 Migrações

`define_table` se verifica ou não a tabela correspondente existir. Se não, ele gera o SQL para criá-lo e executa o SQL. Se a tabela não existe, mas difere do que está sendo definido, ele gera o SQL para alterar a tabela e executa-lo. Se um campo mudou tipo, mas não o nome, ele vai tentar converter os dados (se você não quer isso, você precisa redefinir a tabela duas vezes, a primeira vez, deixando web2py largar o campo, eliminando-o, e pela segunda vez adicionar o campo recém-definido de modo que web2py pode criá-lo.). Se a tabela existe, e coincide com a actual definição, ele vai deixá-lo sozinho. Em todos os casos ele irá criar o objeto `db.person` que representa a tabela.

Nós nos referimos a esse comportamento como uma "migration". web2py logs all migrations and migration attempts in the file "databases/sql.log".

O primeiro argumento do `define_table` é sempre o nome da tabela. Os outros argumentos sem nome são os campos (Field). A função também recebe um último argumento opcional chamado "migrate", que deve ser expressamente referido pelo nome, como em:

```
>>> db.define_table('person', Field('name'), migrate='person.table')
```

O valor de migração é o nome (no "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" arquivos têm que ser removidos manualmente. Por padrão, a migração é definida como True. Isso faz com que web2py para gerar o arquivo a partir de um hash da string de conexão. Migrar Se for definido como False, a migração não é executada, e web2py

assume que a tabela existe no armazenamento de dados e que contém (pelo menos) os campos listados na `define_table`. A melhor prática é dar um nome explícito para a migração da tabela.

Não pode haver duas tabelas no mesmo aplicativo com o mesmo nome migrar.

7.7 Fixação Migrações Broken

Existem dois problemas comuns relacionados com as migrações e existem maneiras de se recuperar a partir deles.

Um problema é específico com o SQLite. SQLite não aplicar tipos de coluna e não pode soltar colunas. Isto significa que se você tiver uma coluna do tipo string e removê-lo, não é realmente removido. Se você adicionar a coluna novamente com um tipo diferente (por exemplo, data e hora) que você acabar com uma coluna datetime que contém strings (sucata para efeitos práticos). web2py não reclama sobre isso porque não sabe o que está no banco de dados, até que ele tenta recuperar os registros e ele falhar.

Se web2py retorna um erro na função `gluon.sql.parse` quando os registros seleção, este é o problema: os dados corrompidos em uma coluna por causa da questão acima.

A solução consiste em atualizar todos os registros da tabela e atualizar os valores na coluna em questão, com nenhum.

O outro problema é mais genérico, mas típica, com MySQL. MySQL não permite mais de um ALTER TABLE em uma transação. Isto significa que deve quebrar web2py transacções complexas em pequenos (um ALTER TABLE na época) e cometer uma peça no momento. Por conseguinte, é possível que parte de uma operação complexa fica comprometida e uma parte não deixando web2py em um estado corrompido. Por que seria parte de uma

transação falhar? Porque, por exemplo, envolve a alteração de uma tabela de conversão e uma coluna de string em uma coluna datetime, web2py tenta converter os dados, mas os dados não podem ser convertidos. O que acontece com web2py? Ele fica confuso sobre o que exatamente é a estrutura da tabela realmente armazenado no banco de dados.

A solução consiste em desativar as migrações para todas as tabelas e permitindo migrações fake:

```
1 db.define_table(...,migrate=False,fake_migrate=True)
```

Isso irá recriar web2py metadados sobre a tabela de acordo com a definição da tabela. Tente várias definições de tabela para ver qual funciona (do que antes da migração falhou eo depois da migração falhou). Uma vez bem sucedido remover o atributo `fake_migrate=True`.

Antes de tentar resolver os problemas de migração é prudente fazer uma cópia de "aplicativos / YourApp / dados / table *. " arquivos.

7.8 insert

Dada uma tabela, você pode inserir registros

```
1 >>> db.person.insert(name="Alex")
2 1
3 >>> db.person.insert(name="Bob")
4 2
```

Inserir retorna o valor original "id" de cada registro inserido.

Você pode truncar a tabela, ou seja, apagar todos os registo e repor o contador do id.

```
1 >>> db.person.truncate()
```

Agora, se você inserir um registro novo, o contador começa novamente em 1 (esta é específica back-end e não se aplica ao jogo):

```
1 >>> db.person.insert(name="Alex")
2 1
```

web2py também oferece um método de E / bulk_insert ft

```
1 >>> db.person.bulk_insert([{'name':'Alex'}, {'name':'John'}, {'name':'Tim'}])
2 [3,4,5]
```

É preciso uma lista de dicionários de campos a serem inseridos e realiza várias inserções ao mesmo tempo. Ele retorna os IDs dos registros inseridos. No banco de dados suportados relacional, não há vantagem em usar esta função ao contrário do looping e executar inserções individuais, mas no Google App Engine, há uma vantagem de velocidade maior.

7.9 commit e rollback

Nenhum criar, apagar, inserir truncar, excluir ou atualizar operação é realmente confirmada até você emitir o comando commit

```
1 >>> db.commit()
```

Para verificar isso, vamos inserir um novo registro:

```
1 >>> db.person.insert(name="Bob")
2 2
```

e rolar para trás, ou seja, ignorar todas as operações desde o último commit:

```
1 >>> db.rollback()
```

Se você inserir novamente agora, o contador voltará a ser fixada em 2, desde a inserção anterior foi revertida.

```
1 >>> db.person.insert(name="Bob")
2 2
```

Código de modelos, visualizações e controladores é incluída no código web2py parecida com esta:

```
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
```

Não há necessidade de chamar cada vez `commit` ou `rollback` explicitamente web2py menos um precisa um controle mais granular.

7.10 Raw SQL

7.10.1 executesql

A DAL permite explicitamente emitir instruções SQL.

```
1 >>> print db.executesql('SELECT * FROM person; ')
2 [(1, u'Massimo'), (2, u'Massimo')]
```

Neste caso, os valores de retorno não são analisados ou transformados pela DAL, eo formato depende do driver de banco de dados específico. Esse uso com seleções normalmente não é necessário, mas é mais comum com os índices. `executesql` leva dois argumentos opcionais: `placeholders` e `as_dict`. `placeholders` é uma seqüência opcional de valores a serem substituídos ou, se suportado pelo controlador DB, um dicionário com chaves combinando espaços reservados em seu nome SQL.

Se o `as_dict` é definida como `True`, e os resultados retornados pelo cursor driver PO serão convertidos para uma seqüência de dicionários introduzidos

com os nomes dos campos db. Resultados voltou com `as_dict = True` are os mesmos devolvidos ao aplicar o `.to_list()` a uma escolha normal.

```
1 [{field1: value1, field2: value2}, {field1: value1b, field2: value2b}]
```

7.10.2 _lastsql

Se o SQL foi executada manualmente ou usando ExecuteSQL foi SQL gerado pelo DAL, você sempre pode encontrar o código SQL no `db._lastsql`. Isso é útil para fins de depuração:

```
1 >>> rows = db().select(db.person.ALL)
2 >>> print db._lastsql
3 SELECT person.id, person.name FROM person;
```

Nunca web2py gera consultas utilizando o operador “”. web2py sempre é explícito quando. campos selecção*

7.11 drop

Finalmente, você pode soltar as tabelas e todos os dados serão perdidos:

```
1 >>> db.person.drop()
```

7.12 Índices

Atualmente, a DAL API não oferece um comando para criar índices em tabelas, mas isso pode ser feito usando o comando `executesql`. Isso ocorre porque a existência de índices pode fazer as migrações complexas, e é melhor lidar com eles de forma explícita. Os índices podem ser necessários para os campos que são usados em consultas periódicas.

Aqui está um exemplo de como criar um índice utilizando SQL em 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;')
```

Outros dialetos banco de dados sintaxes muito semelhantes, mas não pode apoiar a directiva opcionais "IF NOT EXISTS".

7.13 Bases de dados do Legacy e introduzidos quadros

web2py pode se conectar a bancos de dados legados sob algumas condições.

A maneira mais fácil é quando estas condições forem preenchidas:

- Cada tabela deve ter um campo inteiro exclusivo de auto-incremento chamado "id"
- Os registros devem ser referenciados utilizando exclusivamente o campo "id".

Ao acessar uma tabela existente, ou seja, uma tabela criada não por web2py no aplicativo atual, sempre definido migrate=False.

Se a tabela legado tem um campo inteiro de auto-incremento, mas não é chamado "id", web2py ainda pode acessá-lo, mas a definição da tabela deve conter explicitamente como Field('....','id'), onde... é o nome de auto-incremento campo inteiro.

Finalmente, se a tabela legado usa uma chave primária que não é um auto-incremento campo id é possível usar um "keyed table", por exemplo:

```
1 db.define_table('account',
2     Field('accnum', 'integer'),
3     Field('acctype'),
```

```

4     Field('accdesc'),
5     primarykey=['accnum', 'acctype'],
6     migrate=False)

```

Neste exemplo, o atributo `primarykey` é uma lista de domínio que constituem a chave primária. No momento da escrita, não podemos garantir que o atributo `primarykey` trabalha com todas as tabelas legado existente e cada suporte de back-end de banco de dados. Para simplificar, recomendamos, se possível, criando uma base de dados que tem um campo id auto-incremento.

7.14 Distributed Transaction

No momento em que escrevo este recurso só é suportado pelo PostgreSQL, MySQL e Firebird, uma vez que expõem API para duas fases compromete.

Supondo que você tenha duas (ou mais) conexões para PostgreSQL banco de dados distintos, por exemplo:

```

1 db_a = DAL('postgres://... ')
2 db_b = DAL('postgres://... ')

```

Em seus modelos ou controladores, você pode cometê-los simultaneamente com:

```
1 DAL.distributed_transaction_commit(db_a, db_b)
```

Em falha, esta função desfaz e gera um `Exception`.

Nos controladores, quando uma ação retorna, se você tiver duas conexões distintas e você não chamar a função acima, web2py comete-los separadamente. Isto significa que há uma possibilidade de que um dos bem-sucedida e comete uma falha. A transação distribuída impede que isso aconteça.

7.15 Manual Uploads

Considere o seguinte modelo:

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

Normalmente, uma inserção é feita automaticamente através de um SQLFORM ou uma forma bruta (que é um SQLFORM), mas ocasionalmente você já tem o arquivo no sistema de arquivos e deseja enviá-lo programaticamente. Isso pode ser feito da seguinte maneira:

```
1 stream = open(filename,'rb')
2 db myfile.insert(image=db myfile.image.store(stream,filename))
```

O método `store` do objeto de campo de upload tem um fluxo de arquivos e um arquivo. Ele usa o nome do arquivo para determinar a extensão (tipo) do arquivo cria um nome temporário para o novo arquivo (de acordo com web2py mecanismo de upload) e carrega o conteúdo do arquivo no arquivo temp novo (na pasta uploads salvo indicação em contrário). Ela retorna o nome temp novo que está a ser armazena no campo da tabela `image` `db myfile`.

7.16 Query, Set, Rows

Vamos considerar novamente a tabela definida (e caiu) anteriormente e insira três 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
```

Você pode armazenar a tabela em uma variável. Por exemplo, com `person` variável, você pode fazer:

```
>>> person = db.person
```

Você também pode armazenar um campo em uma variável como o `name`. Por exemplo, você também pode fazer:

```
>>> name = person.name
```

Você pode até criar uma consulta (usando operadores como `==`, `!=`, `<`, `<=`, `>`, `>=`, `Como`, `pertence`) e guarde a consulta em uma variável `q`, como em:

```
>>> q = name=='Alex'
```

Quando você chamar `db` com uma consulta, você define um conjunto de registros. Você pode armazená-lo em uma variável `s` e escrever:

```
>>> s = db(q)
```

Observe que nenhuma consulta de banco de dados tenha sido realizada até agora. DAL + Query simplesmente definir um conjunto de registros neste `db` que correspondem à consulta. Web2py determina que a partir da consulta da tabela (ou tabelas) estão envolvidos e, de fato, não há necessidade de especificar isso.

7.17 select

Dado um conjunto, `s`, você pode buscar os registros com o `select` comando:

```
>>> rows = s.select()
```

Ele retorna um objeto iteráveis do `gluon.sql.Rows` classe cujos elementos são objetos `Row`. objetos `gluon.sql.Row` agir como dicionários, mas os seus elementos também podem ser acessadas como atributos, como `ex-gluon.storage.Storage.Th` diferente do último, porque seus valores são somente leitura.

As Linhas de objeto permite looping sobre o resultado da escolha e impressão os valores do campo selecionado para cada linha:

```
1 >>> for row in rows:  
2         print row.id, row.name  
3 Alex
```

Você pode fazer todas as etapas em um comunicado:

```
1 >>> for row in db(db.person.name=='Alex').select():  
2         print row.name  
3 Alex
```

O comando SELECT pode ter argumentos. Todos os argumentos sem nome são interpretados como os nomes dos campos que você deseja buscar. Por exemplo, você pode ser explícita em campo buscando "id" and field "name":

```
1 >>> for row in db().select(db.person.id, db.person.name):  
2         print row.name  
3 Alex  
4 Bob  
5 Carl
```

A tabela de atributos ALL permite especificar todos os campos:

```
1 >>> for row in db().select(db.person.ALL):  
2         print row.name  
3 Alex  
4 Bob  
5 Carl
```

Observe que não há nenhuma seqüência de consulta passou a db. web2py comprehende que se você quiser todos os campos da tabela pessoa sem informações adicionais, então, desejar que todos os registros da tabela pessoa.

Uma sintaxe alternativa equivalente, é o seguinte:

```
1 >>> for row in db(db.person.id > 0).select():  
2         print row.name  
3 Alex  
4 Bob  
5 Carl
```

web2py e comprehende que se você perguntar para todos os registros da tabela pessoa (`id> o`), sem informações adicionais, então você quer todos os campos da tabela pessoa.

7.17.1 Atalhos

A DAL suporta vários códigos simplificar atalhos. Em especial:

```
1 myrecord = db.mytable[id]
```

retorna o registro com o `id` dado, se existir. Se o `id` não existe, ele retorna `None`. A declaração acima é equivalente a

```
1 myrecord = db(db.mytable.id==id).select().first()
```

Você pode apagar registros por ID:

```
1 del db.mytable[id]
```

e isto é equivalente a

```
1 db(db.mytable.id==id).delete()
```

e exclui o registro com o `id` dado, se ele existe.

Você pode inserir os registros:

```
1 db.mytable[0] = dict(myfield='somevalue')
```

É o equivalente a

```
1 db.mytable.insert(myfield='somevalue')
```

e cria um novo registro com os valores do campo especificado pelo dicionário do lado direito.

Você pode atualizar os registros:

```
1 db.mytable[id] = dict(myfield='somevalue')
```

que é equivalente a

```
1 db(db.mytable.id==id).update(myfield='somevalue')
```

e ele atualiza um registro existente com os valores do campo especificado pelo dicionário do lado direito.

7.17.2 *Buscando uma Row*

Contudo uma outra sintaxe conveniente é a seguinte:

```
1 record = db.mytable(id)
2 record = db.mytable(db.mytable.id==id)
3 record = db.mytable(id,myfield='somevalue')
```

Aparentemente semelhante ao db.mytable[id] a sintaxe acima é mais flexível e segura. Primeiro de tudo verifica-se a id é um int (ou str(id) é um int) e retorna None se não for (nunca gera uma exceção). Também permite especificar várias condições que o registro deve satisfazer. Se não forem cumpridas, também retorna None.

7.17.3 *selects recursiva*

Considerar a pessoa tabela anterior e uma nova tabela "dog" referencing a "person":

```
1 >>> db.define_table('dog', Field('name'), Field('owner',db.person))
```

e uma simples escolha desta tabela:

```
1 >>> dogs = db(db.dog.id>0).select()
```

Para cada linha de cães é possível buscar não apenas os campos da tabela selecionada (cão), mas também de tabelas vinculadas (recursivamente):

```
1 >>> for dog in dogs: print dog.info, dog.owner.name
```

Aqui `dog.owner.name` requer um banco de dados para selecionar cada cão em cães e é, portanto, ineficiente. Sugerimos o uso de associações, sempre que possível, em vez de seleciona recursiva, isto é, no entanto, conveniente e prático quando estiver acessando registros individuais.

Você também pode fazê-lo para trás, selecionando os cães referenciados por uma pessoa:

```
1 person = db.person(id)
2 for dog in person.dog.select(orderby=db.dog.name):
3     print person.name, 'owns', dog.name
```

Neste `person.dog` expressões último é um atalho para

```
1 db(db.dog.owner==person.id)
```

ou seja, o conjunto de `dogs` referenciado pelo `person` atual. Esta sintaxe se divide a tabela de referência tem várias referenciados na tabela referenciada. Neste caso é preciso ser mais explícito e usar uma consulta completa.

7.17.4 Serializing no Rows Views

O resultado de uma escolha pode ser apresentada em uma exibição com a seguinte sintaxe:

```
1 {{extend 'layout.html'}}
2 <h1>Records</h2>
3 {{=db().select(db.person.ALL)}}
```

e ele é automaticamente convertido em uma tabela HTML com um cabeçalho contendo os nomes das colunas e uma linha por registro. As linhas são mar-

cadas como alternando "even" and class "odd" classe. Sob o capô, as linhas primeiro é convertido em um objeto SQLTABLE (não confundir com o quadro) e, em seguida, serializado. Os valores extraídos do banco de dados estão também formatado pelos validadores associado ao campo e, em seguida fugiu. (Nota: Usando um db desta forma, em uma visão geralmente não é considerada boa prática MVC).

Contudo, é possível e, por vezes conveniente chamar SQLTABLE explicitamente.

O construtor SQLTABLE leva os seguintes argumentos opcionais:

- `linkto` o URL ou uma ação a ser utilizado para interligar campos de referência (padrão de `None`)
- `upload` a URL ou o recurso de download para permitir o download de arquivos enviados (padrão de `None`)
- `headers` um campo dicionário mapeando nomes para os seus rótulos a serem utilizados como cabeçalhos (padrão para `{}`). Também pode ser uma instrução. Atualmente nós suportamos `headers='fieldname:capitalize'`.
- o número de caracteres para `truncate` truncar valores longos na tabela (o padrão é 16)
- `columns` lista de campos para ser mostrado como colunas. Aqueles que não listados não são exibidos. (Padrão para todos).
- auxiliar genérico `**attributes` atributos a serem passados para o objeto mais externo TABLE.

Aqui está um exemplo:

```
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 }}
```

7.17.5 orderby, groupby, limitby, distinct

O comando `select` leva cinco argumentos opcionais: `orderby`, `GroupBy`, `limitby`, `esquerda` e `cache`. Aqui vamos discutir os três primeiros.

Você pode buscar os registros ordenados por nome:

```
1 >>> for row in db().select(db.person.ALL, orderby=db.person.name):
2     print row.name
3 Alex
4 Bob
5 Carl
```

Você pode buscar os registros ordenados por nome em ordem inversa (note o til):

```
1 >>> for row in db().select(db.person.ALL, orderby=db.person.name):
2     print row.name
3 Carl
4 Bob
5 Alex
```

E você pode classificar os registros de acordo com vários campos, concatenando-os com um "|":

```
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` juntamente com `orderby`, você pode agrupar registros com o mesmo valor para o campo especificado (isto é backend específico, e não é do jogo):

```
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
```

Com o argumento de `distinct=True`, você pode especificar que você quer selecionar somente os registros distintos. Isto tem o mesmo efeito que o agrupamento com todos os campos especificados, exceto que ele não exige a classificação. Ao utilizar distinas é importante para não selecionar os campos todos, e em particular para não selecionar o campo "id", senão todos os registros serão sempre distintas.

Aqui está um exemplo:

```
1 >>> for row in db().select(db.person.name, distinct=True):
2         print row.name
3 Alex
4 Bob
5 Carl
```

Com `limitby`, você pode selecionar um subconjunto dos registros (neste caso, os dois primeiros a partir de zero):

```
1 >>> for row in db().select(db.person.ALL, limitby=(0, 2)):
2         print row.name
3 Alex
4 Bob
```

Atualmente, o "limitby" é apenas parcialmente compatível com o banco de dados MSSQL desde que a Microsoft não fornece um mecanismo para obter um subconjunto de registros que não começa por o.

7.17.6 Operadores lógicos

As consultas podem ser combinados usando o binário e "&" operador:

```
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
```

eo operador binário OR "|":

```
1 >>> rows = db((db.person.name=='Alex') | (db.person.id>3)).select()
```

```

1 >>> for row in rows: print row.id, row.name
2   1 Alex
3

```

Você pode negar uma consulta (ou sub-consulta), com o binário operador !=:

```

1 >>> rows = db((db.person.name != 'Alex') | (db.person.id > 3)).select()
2 >>> for row in rows: print row.id, row.name
3   2 Bob
4   3 Carl

```

ou pela negação explícita com o operador unário |:

```

1 >>> rows = db((db.person.name == 'Alex') | (db.person.id > 3)).select()
2 >>> for row in rows: print row.id, row.name
3   2 Bob
4   3 Carl

```

Devido a restrições de Python em sobrecarga de operadores "AND" and "OR", estes não podem ser usados na formação de consultas. Os operadores binários devem ser usados preferivelmente.

7.17.7 count, delete, update

Você pode contar registros em um conjunto:

```

1 >>> print db(db.person.id > 0).count()
2   3

```

Você pode excluir registros em um conjunto:

```

1 >>> db(db.person.id > 3).delete()

```

E você pode atualizar todos os registros em um conjunto de argumentos nomeados pela passagem correspondente para os campos que precisam ser atualizados:

```

1 >>> db(db.person.id > 3).update(name='Ken')

```

7.17.8 Expressões

O valor atribuído uma instrução de atualização pode ser uma expressão. Por exemplo, 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)
```

Os valores utilizados nas consultas podem também ser expressões

```
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()
```

7.17.9 update_record

web2py também permite atualizar um registro único que já está na memória usando `update_record`

```
1 >>> rows = db(db.person.id > 2).select()
2 >>> row = rows[0]
3 >>> row.update_record(name='Curt')
```

Isso não deve ser confundida com

```
1 >>> row.update(name='Curt')
```

porque, por uma única linha, o método atualiza o objeto `update` linha, mas não o registro de dados, como no caso do `update_record`.

7.17.10 first e last

Dado um objeto que contém linhas registros:

```

1 >>> rows = db(query).select()
2 >>> first_row = rows.first()
3 >>> last_row = rows.last()

```

são equivalentes a

```

1 >>> first_row = rows[0] if len(rows)>0 else None
2 >>> last_row = rows[-1] if len(rows)>0 else None

```

7.17.11 as_dict e as_list

A linha pode ser objeto serializado em um dicionário regular usando o método `as_dict()` e linhas objeto pode ser serializado em uma lista de dicionários através do método `as_list()`. Aqui estão alguns exemplos:

```

1 >>> rows = db(query).select()
2 >>> rows_list = rows.as_list()
3 >>> first_row_dict = rows.first().as_dict()

```

Estes métodos são convenientes para a passagem de linhas de pontos de vista genérico e / ou para armazenar as linhas em sessões (uma vez que objetos linhas em si não pode ser serializada desde incluir uma referência a uma conexão DB aberto):

```

1 >>> rows = db(query).select()
2 >>> session.rows = rows # not allowed!
3 >>> session.rows = rows.as_list() # allowed!

```

7.17.12 find, exclude, sort

Há momentos em que é necessário realizar dois selecione e um contém um subconjunto de uma escolha anterior. Neste caso, é inútil para acessar o

banco de dados novamente. O `find`, `exclude` e `sort` objetos permitem que você manipule um objetos Rows e gerar outro sem acessar o banco de dados. Mais especificamente:

- `find` retorna um novo conjunto de linhas filtradas por uma condição e deixa a original inalterada.
- `exclude` retorna um novo conjunto de linhas filtradas por uma condição e remove-los das linhas originais.
- `sort` retorna um novo conjunto de linhas ordenadas por uma condição e deixa o original inalterado.

Todos estes métodos de ter um único argumento, uma função que atua em cada linha individual.

Aqui está um exemplo 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
```

Eles podem ser combinados:

```
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
```

7.18 Computadorizada Campos

DAL campos podem ter um atributo `compute`. Esta deve ser uma função (ou lambda) que leva um objeto Row e retorna um valor para o campo. Quando um novo registro é inserido (ou atualizado), se um valor para o campo não for fornecido, web2py tenta calcular a partir do campo outros valores usando a função `compute`. Aqui está um exemplo:

```

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 o valor calculado é armazenado no db e não é computado na recuperação, como no caso dos campos virtuais, descritos mais adiante. Duas aplicações típicas de campos calculados são:

- em aplicações wiki, para armazenar a entrada de texto wiki processado como HTML, para evitar re-processamento em cada solicitação
- para a pesquisa, para calcular valores normalizados para um campo, a ser utilizado para a pesquisa.

7.19 Virtual Campos

domínios virtuais também são computados os campos (como na subseção anterior), mas eles diferem daquelas porque são "virtual" no sentido de que elas não são armazenadas no db e eles são computados todos os registros de tempo são extraídos da base de dados. que pode ser usado para simplificar o código do usuário sem a utilização de armazenamento adicional, mas eles não podem ser utilizados para pesquisa.

A fim de definir um ou mais campo virtual, você tem que definir uma classe de contêiner, instanciá-lo e vinculá-lo em uma tabela ou uma seleção. Por exemplo, considere a seguinte tabela:

```
1 >>> db.define_table('item',
2                     Field('unit_price','double'),
3                     Field('quantity','integer'),
```

Pode-se definir um campo virtual preço total

```
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 da classe que recebe um único argumento (self) é um novo campo virtual. self se refere a cada uma linha do select. valores de campo são referidos por caminho completo como no self.item.unit_price. A tabela está ligada aos campos virtual anexando uma instância da classe para virtualfields atributo da tabela. campos virtuais podem também acessar campos recursiva como na

```
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 o self.order_item.item.unit_price acesso recursivo campo onde self é o registro looping.

Eles podem também atuar sobre o resultado de uma junção

```
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

```

Note como neste caso, a sintaxe é diferente. Os acessos campo virtual tanto `self.item.unit_price` e `self.order_item.quantity` que pertencem ao juntar-se selecionar. O campo virtual está ligado às linhas da tabela usando o método das linhas `setvirtualfields` objeto. Este método usa um número arbitrário de argumentos com nome e pode ser usada para definir vários domínios virtuais, definidos em várias classes, e anexá-las para várias tabelas:

```

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 virtuais podem ser preguiçosos, tudo que eles precisam fazer é devolver a função e acessado por chamar a função:

```

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()

```

ou menor usando uma função lambda:

```

1 >>> class MyVirtualFields:
2     def lazy_total_price(self):
3         return lambda self=self: self.item.unit_price*self.item.quantity

```

7.20 Um relacionamento de muitos para

Para ilustrar como implementar um para muitas relações com a DAL web2py, definir outra "dog" que refers to the table "person" mesa que nós redefinir aqui:

```
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" tabela tem dois campos, o nome do cão e do proprietário do cão. Quando um tipo de campo é outra tabela, pretende-se que a referência de campo da outra tabela pelo seu id. Na verdade, você pode imprimir o valor do tipo real e obter:

```
1 >>> print db.dog.owner.type
2 reference person
```

Agora, insira três cães, dois de propriedade de Alex e um por Bob:

```
1 >>> db.dog.insert(name='Skipper', owner=1)
2
3 >>> db.dog.insert(name='Snoopy', owner=1)
4
5 >>> db.dog.insert(name='Puppy', owner=2)
6
```

Você pode escolher como você fez para qualquer outra tabela:

```
1 >>> for row in db(db.dog.owner==1).select():
2         print row.name
3 Skipper
4 Snoopy
```

Como um cão tem uma referência a uma pessoa, uma pessoa pode ter muitos cães, para um recorde de pessoa tabela agora adquire um cão novo atributo,

que é um conjunto, que define que os cães dessa pessoa. Isso permite que looping sobre todas as pessoas e buscar seus cães facilmente:

```

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

```

7.20.1 Inner Joins

Outra maneira de conseguir um resultado semelhante usando uma junção, especificamente um INNER JOIN. web2py executa junta-se automaticamente e transparente quando a consulta une duas ou mais tabelas como no exemplo a seguir:

```

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 fez uma junção, por isso as linhas agora contém dois registros, um de cada tabela, ligados entre si. Como os dois registros podem ter campos com nomes conflitantes, você precisa especificar a tabela ao extrair um valor de campo de uma linha. Isso significa que enquanto antes que você poderia fazer:

```
1 row.name
```

e era óbvio que se esse era o nome de uma pessoa ou um cão, o resultado de uma junção que você tem que ser mais explícito e dizer:

```
1 row.person.name
```

ou:

```
1 row.dog.name
```

7.20.2 *Left Outer Join*

Observe que Carl não aparecem na lista acima, pois ele não tem cães. Caso você pretenda escolher sobre as pessoas (se têm cães ou não) e seus cães (se tiver alguma), então você precisa executar um LEFT OUTER JOIN. Isso é feito usando o "left" argumento do comando select. Aqui está um exemplo:

```
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
```

onde:

```
1 left = db.dog.on(...)
```

faz a junção esquerda consulta. Aqui, o argumento do db.dog.on é a condição necessária para a adesão (o mesmo utilizado anteriormente para a junção interna). No caso de uma união da esquerda, é necessário ser explícito sobre os campos para selecionar.

7.20.3 *Agrupamento e Contagem*

Ao fazer associações, às vezes você deseja agrupar linhas de acordo com certos critérios e contá-las. Por exemplo, contar o número de cães de propriedade de cada pessoa. web2py permite isso também. Primeiro, você precisa de um operador de contagem. Segundo, você quer se juntar a tabela pessoa com o quadro do cão pelo proprietário. Terceiro, você deseja selecionar

todas as linhas (pessoa + cão), agrupá-los por pessoa, e contá-los, enquanto agrupamento:

```

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

```

Observe que o operador `count` (que é built-in) é usada como um campo. O único problema aqui está em como recuperar as informações. Cada linha contém claramente uma pessoa ea contagem, mas a contagem não é um campo de uma pessoa que nem é uma tabela. Então, para onde ela vai? Vai para o armazenamento de objetos que representam o registro com uma chave igual à expressão própria consulta.

7.21 Muitos para muitos

Nos exemplos anteriores, nós permitimos que um cão tem um dono, mas uma pessoa pode ter muitos cães. E se o capitão estava possuído por Alex e Curt? Isso requer um relacionamento de muitos para muitos, e é realizado através de uma tabela intermediária que liga uma pessoa a um cão através de uma relação de propriedade.

Aqui está como fazê-lo:

```

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))

```

as relações de propriedade existentes pode agora 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
```

Agora você pode adicionar a nova relação que Curt co-proprietária Skipper:

```
>>> db.ownership.insert(person=3, dog=1) # Curt owns Skipper too
```

Porque agora você tem uma relação de meio-três entre as tabelas, pode ser conveniente para definir um novo conjunto em que para realizar operações:

```
>>> persons_and_dogs = db((db.person.id==db.ownership.person) | (db.dog.id==db.
    ownership.dog))
```

Agora é fácil de selecionar todas as pessoas e seus cães a partir de um novo conjunto:

```
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
```

Da mesma forma, você pode procurar por todos os cães de propriedade de Alex:

```
1 >>> for row in persons_and_dogs(db.person.name=='Alex').select():
2         print row.dog.name
3 Skipper
4 Snoopy
```

e todos os proprietários de Skipper:

```
1 >>> for row in persons_and_dogs(db.dog.name=='Skipper').select():
2         print row.owner.name
3 Alex
4 Curt
```

Uma alternativa mais barata para muitos Muitos duas relações é uma marcação. Tagging é discutido no contexto do validador IS_IN_DB. Marcação de obras, mesmo em backends do banco de dados que não suporta JOINs como o Google App Engine.

7.22 Outros operadores

web2py tem outras operadoras que fornecem uma API para acesso equivalente operadores SQL. Vamos definir uma outra tabela "log" para armazenar os eventos de segurança, o seu carimbo do tempo e da gravidade, onde a gravidade é um número inteiro.

```
1 >>> db.define_table('log', Field('event'),
2                      Field('timestamp', 'datetime'),
3                      Field('severity', 'integer'))
```

Como antes, inserir alguns eventos, a "port scan", an "xss injection" and an "unauthorized login". Por causa do exemplo, você pode registrar eventos com a mesma hora, mas com diferentes gravidades (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
```

7.22.1 like, upper, lower

Campos tem um operador como a que você pode usar para combinar sequências:

```
1 >>> for row in db(db.log.event.like('port%')).select():
2         print row.event
3 port scan
```

Aqui "port%" indicates a string starting with "port". The percent sign character, "%", is a wild-card character that means "any sequence of characters".

Da mesma forma, você pode usar métodos `upper` e `lower` para converter o valor do campo para maiúsculas ou minúsculas, e você também pode

combiná-los com o operador LIKE.

```
1 >>> for row in db(db.log.event.upper().like('PORT%')).select():
2         print row.event
3
port scan
```

7.22.2 year, month, day, hour, minutes, seconds

A data e campos datetime tem dia, mês e métodos ano. Os campos de data e hora e hora a hora, minuto e segundo métodos. Aqui está um exemplo:

```
1 >>> for row in db(db.log.timestamp.year() == 2009).select():
2         print row.event
3
port scan
4 xss injection
5 unauthorized login
```

7.22.3 belongs

O SQL operador IN é realizada através do método que pertence retorna true quando o valor do campo pertence ao 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
```

A DAL também permite uma escolha aninhados como o argumento da operadora pertence. A única ressalva é que o SELECT aninhada tem de ser um _select, select, não uma, e apenas um campo tem que ser selecionado de forma explícita, o que define o 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, você usou o operador de contagem para contar os registros. Da mesma forma, você pode usar o operador de soma para adicionar (soma), os valores de um campo específico a partir de um grupo de registros. Como no caso da contagem, o resultado de uma soma é recuperada através do objeto de armazenamento:

```
1 >>> sum = db.log.severity.sum()
2 >>> print db().select(sum).first()[sum]
3 6
```

7.23 Gerando SQL-primas

Às vezes você precisa para gerar o SQL, mas não executá-lo. Isso é fácil de fazer com web2py uma vez que cada comando que executa banco de dados IO tem um comando equivalente que não, e simplesmente retorna o SQL que teria sido executado. Esses comandos têm os mesmos nomes e sintaxe como os funcionais, mas eles começam com um sublinhado:

Aqui está o `_insert`

```
1 >>> print db.person._insert(name='Alex')
2 INSERT INTO person(name) VALUES ('Alex');
```

Aqui está o `_count`

```
1 >>> print db(db.person.name=='Alex')._count()
2 SELECT count(*) FROM person WHERE person.name='Alex';
```

Aqui está o `_select`

```
1 >>> print db(db.person.name=='Alex')._select()
2 SELECT person.id, person.name FROM person WHERE person.name='Alex';
```

Aqui está o `_delete`

```
1 >>> print db(db.person.name=='Alex')._delete()
2 DELETE FROM person WHERE person.name='Alex';
```

E finalmente, aqui está `_update`

```
1 >>> print db(db.person.name=='Alex')._update()  
2 UPDATE person SET WHERE person.name='Alex';
```

Além disso, você sempre pode usar `db._lastsql` para retornar o código SQL mais recente, se foi executada manualmente ou usando `ExecuteSQL` foi SQL gerado pela DAL.

7.24 Exportação e importação de dados

7.24.1 CSV (um quadro de cada vez)

Quando um DALRows objeto é convertido para uma string é automaticamente serializado em 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
```

Você pode serializar uma única tabela em formato CSV e armazená-lo em um arquivo "test.csv":

```
1 >>> open('test.csv', 'w').write(str(db(db.person.id).select()))
```

e você pode facilmente lê-lo de volta com:

```
1 >>> db.person.import_from_csv_file(open('test.csv', 'r'))
```

Ao importar, web2py olha para os nomes de campo no cabeçalho CSV. Neste exemplo, ele encontra duas colunas: campos "person.id" and "person.name".

It ignores the "person." prefix, and it ignores the "id". Então, todos os registros são acrescentados e atribuiu novas identificações. Ambas as operações podem ser realizadas através do appadmin web interface.

7.24.2 CSV (*todas as tabelas de uma vez*)

Em web2py, você pode fazer backup / restore um banco de dados inteiro com dois comandos:

Para exportar:

```
>>> db.export_to_csv_file(open('somefile.csv', 'wb'))
```

Para importar:

```
>>> db.import_from_csv_file(open('somefile.csv', 'rb'))
```

Este mecanismo pode ser usado mesmo se o banco de dados de importação é de um tipo diferente do banco de dados de exportação. Os dados são armazenados no "somefile.csv" como um arquivo CSV, onde cada tabela começa com uma linha que indica o tablename, e outra linha com os Campos:

```
1 TABLE tablename
2 field1, field2, field3, ...
```

Duas tabelas são separadas \r\n\r\n. O arquivo termina com a linha

```
1 END
```

O arquivo não inclui os arquivos enviados, caso estes não são armazenadas no banco de dados. Em qualquer caso, é bastante fácil zip a pasta "uploads" separadamente.

Ao importar, os novos registros serão acrescentados ao banco de dados se não está vazio. Em geral, os novos registros importados não terão o mesmo

registro id como o original (guardado), mas os registros web2py irá restaurar as referências para que eles não são divididos, mesmo os valores de ID podem mudar.

Se a tabela contém um campo chamado "uuid", este campo será utilizado para identificar duplicatas. Além disso, se um registro importado tem o "uuid" mesmo que um registro existente, o recorde anterior será atualizada.

7.24.3 CSV e sincronização de banco de dados remoto

Considere o seguinte 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 é identificado por um ID e referenciado por esse ID. Se você tem duas cópias do banco de dados utilizado por diferentes instalações web2py, o ID é exclusivo somente dentro de cada banco e não através de bases de dados. Este é um problema que se fundem quando os registros de diferentes bases de dados.

A fim de fazer um registro único de identificação através de bases de dados, devem:

- ter um id exclusivo (UUID)
- tem um carimbo (para descobrir qual é o mais recente se múltiplas cópias),

- referência o UUID em vez do id.

Isto poderá ser conseguido sem modificar web2py. Aqui está o que fazer:

Alterar o modelo 1. acima em:

```

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
14 db.dog.owner.requires = IS_IN_DB(db,'person.uuid','%(name)s')
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")

```

Criar 2. a ação do controlador para exportar o banco de dados:

```

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()

```

Criar 3. a ação do controlador para importar uma cópia salva de outro banco de dados e registros de sincronização:

```

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)
```

Criar um índice 4. manualmente para fazer a busca por uuid rapidamente.

Observe que as etapas 2 e 3 trabalhos para cada modelo de banco de dados, eles não são específicas para este exemplo.

Alternativamente, você pode usar o XML-RPC para exportação / importação do arquivo.

Se a referência de registros enviados os arquivos, você também precisa exportar / importar o conteúdo da pasta de uploads. Observe que os arquivos nele já estão rotulados por UUIDs assim você não precisa se preocupar com conflitos de nomes e referências.

7.24.4 HTML / XML (*uma tabela de tempo*)

DALRows objetos também têm um método `xml` (como ajudantes) que serializa-lo para 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>
```

Se você precisar serializar o DALRows em qualquer formato XML com outras marcas personalizadas, você pode facilmente fazer isso usando o auxiliar TAG universal ea notação *:

```

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>
```

7.24.5 Data Representation

A função `export_to_csv_file` aceita um argumento nomeado chave `represent`. Quando `True` vai utilizar as colunas função `represent` ao exportar os dados em vez dos dados brutos.

A função também aceita um argumento nomeado `colnames` palavra-chave que deve conter uma lista de nomes de uma coluna deseja exportar. O padrão para todas as colunas.

Ambos aceitam `export_to_csv_file` e `import_from_csv_file` chave argumentos que dizem que o analisador no formato CSV para guardar / carregar os arquivos:

- padrão delimiter: delimitador para separar valores (',')
- quotechar: caracteres para usar para citar valores string (padrão de aspas)
- quoting: sistema de cotação (csv.QUOTE_MINIMAL padrão)

Aqui estão algumas de uso exemplo:

```
1 >>> import csv
2 >>> db.export_to_csv_file(open('/tmp/test.txt', 'w'),
3     delimiter='|',
4     quotechar='"',
5     quoting=csv.QUOTE_NONNUMERIC)
```

O que tornaria algo semelhante ao

```
1 "hello"|35|"this is the text description"|"2009-03-03"
```

Para mais informações consultar a documentação do Python oficiais `quoteall`:
`cite`

7.25 Seleciona Caching

O método de escolha também leva um argumento de cache, o padrão é `None`. Para fins de cache, ele deve ser definido como uma tupla, onde o primeiro elemento é o modelo de cache (`cache.ram`, `cache.disk`, etc), eo segundo elemento é o tempo de expiração em segundos.

No exemplo a seguir, você verá um controlador que armazena um `select` na tabela `db.log` previamente definidos. A busca selecionar os dados reais do banco de dados de back-end não mais do que uma vez a cada 60 segundos e armazena o resultado em `cache.ram`. Se a próxima chamada para este controlador ocorre em menos de 60 segundos desde o banco de dados IO passado, ele simplesmente recupera os dados anteriores de `cache.ram`.

```
1 def cache_db_select():
2     logs = db().select(db.log.ALL, cache=(cache.ram, 60))
3     return dict(logs=logs)
```

— Os resultados são de uma escolha complexa, un-pickle objetos, eles não podem ser armazenados em uma sessão e não pode ser armazenada em qualquer outra forma que o explicado aqui —.

7.26 A auto-referência e Alias

É possível definir as tabelas com os campos que se referem a si mesmos, embora a notação usual pode falhar. O código a seguir seria errado, porque ele usa uma variável db.person antes que seja definido:

```
1 db.define_table('person',
2     Field('name'),
3     Field('father_id', db.person),
4     Field('mother_id', db.person))
```

A solução consiste em utilizar uma notação alternativa

```
1 db.define_table('person',
2     Field('name'),
3     Field('father_id', 'reference person'),
4     Field('mother_id', 'reference person'))
```

Na verdade db.tablename e "reference tablename" são tipos de campo equivalente.

Se a tabela se refere a si mesmo, então não é possível realizar um JOIN para selecionar uma pessoa e dos seus pais sem o uso da palavra-chave SQL "AS". Isto é conseguido em web2py usando o `with_alias`. Aqui está um exemplo:

```
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 optámos por fazer uma distinção entre:

- "father_id": the field name used in the table "person";
- "father": o apelido que deseja usar para a tabela referenciada pelo campo acima, o que é comunicado ao banco de dados;
- "Father": a variável usada por web2py referir que alias.

A diferença é sutil, e não há nada de errado em usar o mesmo nome para os três:

```
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
```

Mas é importante ter a distinção clara, a fim de criar consultas correta.

7.27 Quadro sucessório

É possível criar uma tabela que contém todos os campos de outra tabela. É suficiente para passar a outra tabela no lugar de um campo para `define_table`. Por exemplo

```
1 db.define_table('person', Field('name'))
2
3 db.define_table('doctor', db.person, Field('specialization'))
```

Também é possível definir uma tabela fictícia que não está armazenado em um banco de dados, a fim de reutilizá-lo em vários outros lugares. Por exemplo:

```
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'))
```

Este exemplo assume que a autenticação web2py padrão é habilitado.

8

Formulários e validadores

Existem quatro maneiras distintas para construir formas de web2py:

- `FORM` proporciona uma aplicação de baixo nível em termos de ajudantes HTML. Um objeto `FORM` pode ser serializado em HTML e está ciente dos campos que ele contém. Um objeto `FORM` sabe como validar os dados submetidos pelo formulário.
- `SQLFORM` oferece uma API de alto nível para a construção de criar, atualizar e excluir as formas de uma tabela do banco de dados existente.
- `SQLFORM.factory` é uma camada de abstração em cima do `SQLFORM`, a fim de tirar proveito dos recursos de forma a geração, mesmo que não esteja presente nenhum banco de dados. Ele gera uma forma muito semelhante ao `SQLFORM` da descrição de uma tabela, mas sem a necessidade de criar a tabela banco de dados.
- métodos `CRUD`. Estes são funcionalmente equivalentes aos `SQLFORM` e baseiam-se `SQLFORM`, mas fornecem uma notação mais compacta.

Todas essas formas são auto-conhecimento e, se a entrada não passar na validação, pode-se modificar e adicionar mensagens de erro. Os formulários

podem ser consultados para as variáveis validados e as mensagens de erro que foram geradas pela validação.

Arbitraria de código HTML pode ser inserido ou extraído do formulário usando ajudantes.

8.1 FORM

Considere como exemplo uma aplicação com o **test "default.py"** controlador seguinte:

```
1 def display_form():
2     return dict()
```

eo "default/display_form.html" vista associada:

```
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)}}
```

Este é um formulário HTML normal, que pede o nome do usuário. Quando você preencher o formulário e clique no botão de envio, o formulário de auto-sustenta, eo `request.vars.name` variável e seu valor é exibido na parte inferior.

Você pode gerar a mesma forma com ajudantes. Isso pode ser feito na visão ou na ação. Desde web2py processado sob a forma da ação, que é OK para definir a forma da ação.

Aqui está o novo controlador:

```

1 def display_form():
2     form=FORM('Your name:', INPUT(_name='name'), INPUT(_type='submit'))
3     return dict(form=form)

```

eo "default/display_form.html" vista associada:

```

1 {{extend 'layout.html'}}
2 <h2>Input form</h2>
3 {{=form}}
4 <h2>Submitted variables</h2>
5 {{=BEAUTIFY(request.vars)}}

```

O código de medida é equivalente ao código anterior, porém a forma é gerada pelo {{=form}} declaração que serializa o objeto FORM.

Agora vamos adicionar um nível de complexidade, acrescentando a validação do formulário e da transformação.

Alterar o controlador do seguinte modo:

```

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)

```

eo "default/display_form.html" vista associada:

```

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)}}

```

Observe que:

- Na ação, que acrescentou o validador `requires=IS_NOT_EMPTY()` "name" para o campo de entrada.
- Na ação, nós adicionamos uma chamada para `form.accepts(...)`
- Na opinião, estamos imprimindo `form.vars` e `form.errors`, bem como a forma e `request.vars`.

Todo o trabalho é feito pelo método do objeto `accepts` `form`. Ele filtra os `request.vars` de acordo com as necessidades declaradas (expresso por validadores). `accepts` armazena as variáveis que passam pela validação em `form.vars`. Se um valor de campo não atender a um requisito, o validador não retorna um erro eo erro é armazenado em `form.errors`. Ambas as `form.vars` e `form.errors` são semelhantes aos objetos `gluon.storage.Storage` `request.vars`. O primeiro contém os valores que passaram a validação, por exemplo:

```
1 form.vars.name = "Max"
```

Este último contém erros, por exemplo:

```
1 form.errors.name = "Cannot be empty!"
```

A assinatura completa do método `accepts` é o seguinte:

```
1 form.accepts(vars, session=None, formname='default',
2                 keepvalues=False, onvalidation=None,
3                 dbio=True, hideerror=False):
```

O significado dos parâmetros é opcional explicar nas próximas sub-seções.

O `accepts True` função retorna quando o formulário for aceito e `False` em contrário. A forma não é aceite se tiver erros ou quando não tenha sido apresentado (por exemplo, a primeira vez que é mostrado).

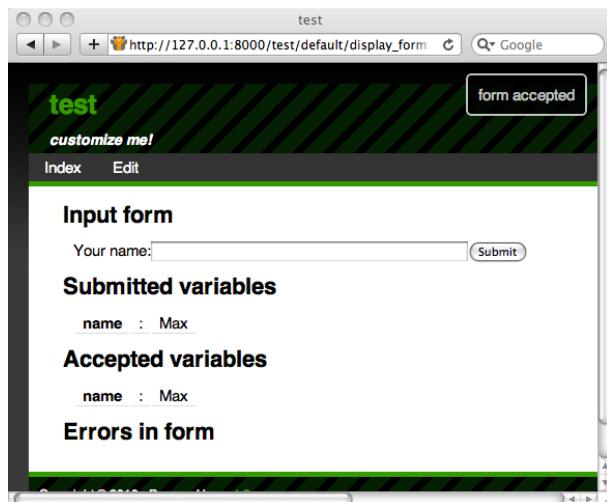
Aqui está como esta página parece a primeira vez que é exibida:

The screenshot shows a web application titled "test" with a green header bar containing "test", "customize me!", "Index", and "Edit". A message "please fill the form" is displayed in a box. The main content area has a title "Input form" and a form field "Your name: Max" with a "Submit" button. Below it, sections for "Submitted variables" and "Accepted variables" are shown. At the bottom, a "Copyright © 2010 - Powered by web2py" footer is visible.

Aqui é como parece inválido, mediante a apresentação:

The screenshot shows a web application titled "test" with a green header bar containing "test", "customize me!", "Index", and "Edit". A message "form has errors" is displayed in a box. The main content area has a title "Input form" and a form field "Your name:" with a red error message "enter a value" below it. A "Submit" button is also present. Below it, sections for "Submitted variables" and "Accepted variables" are shown. At the bottom, a "Errors in form" section is visible.

Aqui está como ele olha para uma apresentação válida:



8.1.1 Campos ocultos

Quando o objeto é serializado formulário acima de `{=form{}}`, e por causa da chamada anterior para o método `accepts`, agora fica assim:

```
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>
```

Observe a presença de dois campos ocultos: `"_formkey"` and `"_formname"`. Sua presença é desencadeada pela chamada para `accepts` e desempenham duas funções diferentes e importantes:

- O campo oculto chamado `"_formkey"` é um símbolo de um tempo que web2py usa para evitar o envio de formulários de casal. O valor desta chave é gerado quando o formulário é serializado e armazenado no `session`.

Quando o formulário é submetido este valor deve corresponder, senão retorna `accepts False` sem erros, como se o formulário não foi apresentado a todos. Isso ocorre porque web2py não pode determinar se o formulário foi enviado corretamente.

- O campo oculto chamado "`_formname`" é gerado por web2py como um nome para o formulário, mas o nome pode ser substituído. Este campo é necessário para permitir que as páginas que contêm eo processo de múltiplas formas. web2py distingue as diferentes formas apresentadas por seus nomes.
- Opcional campos ocultos especificado como `FORM(...,hidden=dict(...))`.

O papel destes campos ocultos e seu uso em formulários personalizados e páginas com múltiplas formas é discutido em detalhes mais adiante neste capítulo.

Se o formulário acima é submetido a um campo vazio "name", o formulário não passar na validação. Quando o formulário for serializado novamente ele 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>
```

Observe a presença de uma DIV de classe "error" na forma serializada. insere web2py esta mensagem de erro na forma de notificar o visitante sobre o campo que não passar na validação. O método `accepts`, mediante a apresentação, determina que o formulário é submetido, verifica se o "name" campo está vazio e se for necessário, e, eventualmente, insere a mensagem de erro do validador no formulário.

O "layout.html" view is expected to handle DIVs of class "error" base. O layout padrão utiliza efeitos jQuery para fazer aparecer os erros e deslize

para baixo com um fundo vermelho. Consulte o Capítulo 10 para obter mais detalhes.

8.1.2 keepvalues

O `keepvalues` argumento opcional web2py diz o que fazer quando uma forma é aceite e não há redirecionamento para que o mesmo formulário é exibido novamente. Por padrão, o formulário está desmarcada. Se o `keepvalues` está definido para `True`, o formulário está pré-preenchida com os valores previamente inseridos. Isso é útil quando você tem um formulário que deveria ser usado repetidamente para inserir vários registros semelhantes. Se o argumento é definido para `dbio False`, web2py não vai realizar qualquer DB inserir / atualizar depois de aceitar formulário. Se o `hideerror` está definido para `True` e do formulário contém erros, estes não serão exibidos quando o formulário é processado (será até você para mostrar-lhes de alguma forma `form.errors`). O argumento é explicado abaixo.

8.1.3 onvalidation

O argumento pode ser `onvalidation` ou `None` pode ser uma função que toma a forma e não devolve nada. Tal função seria chamada e passou a formar, logo após a validação (validação se passa) e antes de mais nada acontece. O objectivo desta função é polivalente. Ele pode ser usado, por exemplo, para executar verificações adicionais sobre a forma e, finalmente, adicionar os erros no formulário. Também pode ser usado para calcular os valores de alguns campos com base nos valores de outros campos. Ele pode ser usado para disparar alguma ação (como enviar um e-mail) antes de um registro é criado / atualizado.

Aqui está um exemplo:

```
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)

```

8.1.4 Formas e redirecionamento

A forma mais comum de usar formas é através da auto-apresentação, de modo que as variáveis de campo apresentados são processados pela mesma ação que gerou o formulário. Depois que o formulário for aceito, ele é incomum para exibir a página atual novamente (algo que estamos fazendo aqui apenas para manter as coisas simples). É mais comum para redirecionar o visitante para uma página "next".

Aqui está o novo controlador exemplo:

```

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 fim de definir um flash na página seguinte, em vez de a página atual você deve usar o `session.flash` ao invés do `response.flash`. web2py move o primeiro no segundo, após o redirecionamento. Note que `session.flash` usando exige que você não `session.forget()`.

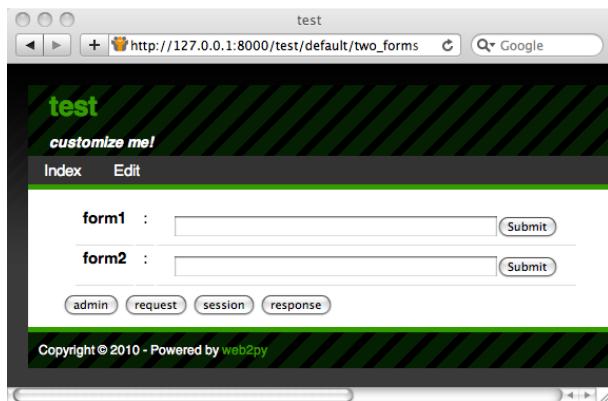
8.1.5 Múltiplas formas por página

O conteúdo desta secção é válido tanto para `FORM` e `SQLFORM` objetos.

É possível ter várias formas por página, mas você deve permitir web2py para distingui-los. Se estes são determinados pelo `SQLFORM` de tabelas diferentes, então web2py dá-lhes nomes diferentes automaticamente, caso contrário, você precisa explicitamente dar-lhes nomes de forma diferente. Além disso, quando múltiplas formas estão presentes na mesma página, o mecanismo para evitar quebras de apresentação dupla, e você deve omitir o argumento `session` ao chamar o método `accepts`. Aqui está um exemplo:

```
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)
```

e aqui é a saída que produz:



Quando o visitante envia um form1 vazio, apenas form1 exibirá um erro, se o visitante envia um Form2 vazio, apenas Form2 exibe uma mensagem de erro.

8.1.6 Partilha Forms

O conteúdo desta secção é válido tanto para FORM e SQLFORM objetos. O que nós discutimos aqui é possível, mas não é recomendado, pois é sempre bom ter as formas que se auto-apresentar. Às vezes, porém, você não tem escolha, porque a ação que envia o formulário ea ação que recebe pertencem a diferentes aplicações.

É possível gerar um formulário que submete a uma ação diferente. Isso é feito especificando o URL da ação de transformação nos atributos do objeto FORM ou SQLFORM. Por exemplo:

```

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()
```

Note que desde que o uso tanto do "page_one" and "page_two" form mesmo, temos definido apenas uma vez, colocando-o fora de todas as ações, a fim de não repetir nós mesmos. A parte comum de código no início de um controlador é executado toda vez antes de dar o controle para a ação chamada.

Desde "page_one" não chamar accepts, o formulário não tem nome e não essenciais, para que você não deve passar o session formname=None e accepts em conjunto, ou a forma não irá validar "page_two" quando recebe-lo.

8.2 SQLFORM

Passamos agora para o próximo nível, fornecendo o aplicativo com um arquivo de modelo:

```
1 db = DAL('sqlite://storage.sqlite')  
2 db.define_table('person', Field('name', requires=IS_NOT_EMPTY()))
```

Modificar o controlador da seguinte forma:

```
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)
```

A exibição não precisa ser alterado.

No novo controlador, você não precisa construir um FORM, SQLFORM desde o construtor construiu uma db.person da tabela definida no modelo. Esta nova forma, quando serializado, 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>
```

O formulário gerado automaticamente é mais complexa do que a forma anterior de baixo nível. Primeiro de tudo, ele contém uma tabela de linhas, e cada linha tem três colunas. A primeira coluna contém o campo de rótulos (como determinado a partir do db.person), a segunda coluna contém os campos de entrada (e, eventualmente, mensagens de erro), ea terceira coluna é opcional e, portanto, vazia (pode ser preenchida com os campos no construtor SQLFORM).

Todas as marcas sob a forma têm nomes derivados da tabela e nome do campo. Isto permite uma fácil personalização do formulário usando CSS e JavaScript. Esse recurso é discutido em mais detalhe no Capítulo 10.

Mais importante é que agora o método accepts funciona muito mais para você. Como no caso anterior, que realiza a validação da entrada, mas, além disso, se a entrada passa a validação, ele também executa um banco de dados de inserir o novo registro e lojas em form.vars.id a "id" exclusivo do novo registro.

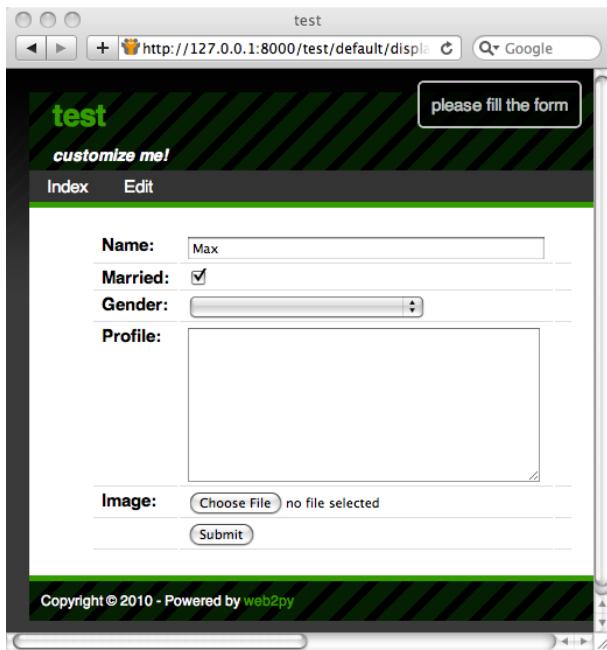
Um objeto SQLFORM também lida automaticamente com a pasta "upload" fields by saving uploaded files in the "uploads" (depois de tê-los com segurança renomeada para evitar conflitos e prevenir ataques de passagem de diretório) e armazena seus nomes (os nomes de novo) em campo apropriado na base de dados.

A mostra 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, uma vez que é suposto ser tratadas de maneira diferente, como discutido mais tarde.

Por exemplo, considere o seguinte 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'))
```

Neste caso, SQLFORM(db.person) gera o formulário abaixo:



O construtor `SQLFORM` permite personalizações diversas, como exibir apenas um subconjunto dos campos, mudando os rótulos, agregando valores para a terceira coluna opcional, ou criando formas UPDATE e DELETE, em oposição às formas INSERT como a atual. `SQLFORM` é o único objeto maior poupança de tempo em web2py.

O `SQLFORM` classe é definida em "gluon/sqlhtml.py". Ele pode ser facilmente estendido, sobrepondo o método `xml`, o método que serializa os objetos, mudar a sua saída.

A assinatura do `SQLFORM` para o construtor é o seguinte:

```

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)

```

- O segundo argumento opcional transforma a forma em uma forma de INSERT UPDATE para o registro especificado (ver ponto seguinte).
-

`deletable` Se está definido para `True`, a forma de atualização mostra uma caixa de seleção "Check to delete". O valor do rótulo, se esse campo é definido através do argumento `delete_label`.

- `submit_button` define o valor no botão enviar.
- `id_label` define o rótulo do "id" registro
- O "id" do registro não é mostrado se `showid` está definido para `False`.
- `fields` é uma lista de nomes de domínio que você deseja exibir. Se uma lista é fornecida, somente os campos na lista são exibidos. Por exemplo:

```
1 fields = ['name']
```

- `labels` é um dicionário de nomes de campo. A chave do dicionário é um nome de domínio eo valor correspondente é o que é apresentado como o seu rótulo. Se um rótulo não é fornecida, web2py deriva a etiqueta do nome do campo (que capitaliza o nome do campo e substitui sublinhados com espaços). Por exemplo:

```
1 labels = {'name': 'Your Full Name: '}
```

- `col3` é um dicionário de valores para a terceira coluna. Por exemplo:

```
1 col3 = {'name': A('what is this?',  
2 _href='http://www.google.com/search?q=define:name') }
```

- `linkto` e `upload` são URLs opcionais para os controladores definidos pelo usuário que permitem que a forma de lidar com campos de referência. Isso é discutido com mais detalhes na seção.
- `readonly`. Se definido como `True`, exibe a forma como `readonly`
- `comments`. Se definido como `False`, não exibir o `col3` comentários
- `ignore_rw`. Normalmente, um programa para criar / atualizar forma, somente os campos marcados como `writable = True` são mostrados, e as

formas readonly, os campos marcados como somente leitura = True são mostrados. Configuração faz com que essas restrições ignore_rw=True para ser ignorado, e todos os campos são exibidos. Isto é principalmente utilizado na interface appadmin para mostrar todos os campos de cada tabela, substituindo o modelo que indica.

-

`formstyle` determina o estilo a ser usado quando a serialização o formulário em html. Pode 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 a forma divs usando css simpática, para customization arbitrária). `formstyle` pode ser também uma função que recebe (`record_id`, `field_label`, `field_widget`, `field_comment`) como atributos e retorna a `TR()` do objeto.

- `attributes` Argumentos opcionais estão começando com sublinhado que você deseja passar para a tag `FORM` que torna o objeto `SQLFORM`. Exemplos são:

```
1 _action = '.'
2 _method = 'POST'
```

Não é um atributo especial . Quando um dicionário é passado como , seus itens são convertidos em campos de entrada "hidden" (veja o exemplo para o auxiliar `FORM` no capítulo 5).

8.2.1 SQLFORM e insert/update/delete

Se você passar um registro como segundo argumento opcional para o construtor `SQLFORM`, a forma torna-se um formulário de atualização para esse registro. Isto significa que quando o formulário é enviado o registro existente é atualizado e nenhum novo registro é inserido. Se você definir o argumento

`deletable=True`, a forma de atualização mostra uma caixa de seleção "check to delete". Caso marcado, o registro é excluído.

Você pode, por exemplo, modificar o controlador do exemplo anterior, de modo que quando passamos um argumento inteiro suplementar no caminho de URL, como em:

```
1 /test/default/display_form/2
```

e se existe um registro com o id correspondente, o SQLFORM gera um DELETE / UPDATE formulário para o 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)
```

Linha 3 localiza o registro, a linha 5 faz um formulário / UPDATE DELETE, a linha 7 faz um formulário de INSERT. A linha 8 faz todo o processamento de forma correspondente.

Uma forma de atualização é muito semelhante a uma forma de criar, exceto que ele é pré-preenchido com o registro atual e previews imagens dele. Por padrão deletable = True, o que significa a forma de atualização irá mostrar uma opção "delete record".

Edit formas conter também um campo hidden com o `name="id"`, que é usado para identificar o registro. Esta identificação também é armazenado no lado do servidor para segurança adicional e, se o visitante mexe com o valor deste campo, a atualização não é realizada e web2py levanta SyntaxError, "user is tampering with form".

Quando um campo é marcado com `writable=False`, o campo não é mostrada na criação de formas, e é `readonly` é mostrado em forma de atualização. Se

um campo estiver marcado como `writable=False` e `readable=False`, então o campo não é mostrado em tudo, nem mesmo em formas de atualização.

Formulários criados com o

```
1 form = SQLFORM(...,ignore_rw=True)
```

ignorar os atributos `readable` e `writable`, e sempre mostrar todos os campos. Formas de ignorar `appadmin`-los por padrão.

Formulários criados com o

```
1 form = SQLFORM(table, record_id, readonly=True)
```

sempre mostrar todos os campos no modo `readonly`, e não pode ser aceite.

8.2.2 SQLFORM em HTML

Há momentos em que você deseja usar `SQLFORM` a beneficiar da sua geração forma e tratamento, mas você precisa de um nível de personalização da forma em HTML que você não pode alcançar com os parâmetros de objeto `SQLFORM`, então você tem que criar o formulário usando HTML.

Agora, edite o controlador anterior e adicionar uma nova ação:

```
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 inserir o formulário na vista associada "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>
```

Observe que a ação não devolver o formulário, pois não precisa passá-lo à vista. A exibição contém um formulário criado manualmente em HTML. O formulário contém um campo oculto que "_formname" deve ser o mesmo FormName especificado como um argumento de accepts na ação. web2py usa o nome do formulário no caso de existirem várias formas na mesma página, para determinar qual foi submetido. Se a página contém um único formulário, você pode definir formname=None e omitir o campo oculto na exibição.

8.2.3 SQLFORM e Uploads

Campos do tipo "upload" são especiais. Eles são processados como campos de entrada de type="file". Salvo disposição em contrário, o arquivo enviado é transmitido em usar um tampão, e armazenados sob a pasta da aplicação "uploads" usando um novo nome de segurança, atribuídos automaticamente. O nome deste arquivo é salvo no campo de uploads tipo.

Como exemplo, considere o seguinte modelo:

```
1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('image', 'upload'))
```

Você pode usar o "display_form" mesma ação controlador mostrado acima.

Quando você insere um novo registro, o formulário permite que você navegue para um arquivo. Escolha, por exemplo, uma imagem jpg. O arquivo é carregado e armazenado como:

```
1 applications/test/uploads/person.image.XXXXXX.jpg
```

"XXXXXX" é um identificador para o arquivo aleatório atribuído pelo web2py.

Observe que, por padrão, o nome original de um arquivo enviado é b16encoded e utilizados para construir o novo nome para o arquivo. Este nome é recuperado pela ação padrão "download" e usado para definir o cabeçalho de disposição do conteúdo para o arquivo original.

Apenas a sua extensão está preservada. Este é um requisito de segurança desde o nome do arquivo pode conter caracteres especiais que podem permitir que um visitante para executar ataques de passagem de diretório ou outras operações maliciosas.

O novo nome do arquivo também é armazenado no `form.vars.image`.

Ao editar o registro usando um formulário de atualização, ele seria bom para mostrar um link para o arquivo existente carregado, e web2py fornece uma maneira de fazê-lo.

Se você passar uma URL para o construtor SQLFORM através do argumento de upload, web2py usa a ação em que URL para baixar o arquivo. Considere as seguintes ações:

```
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)
```

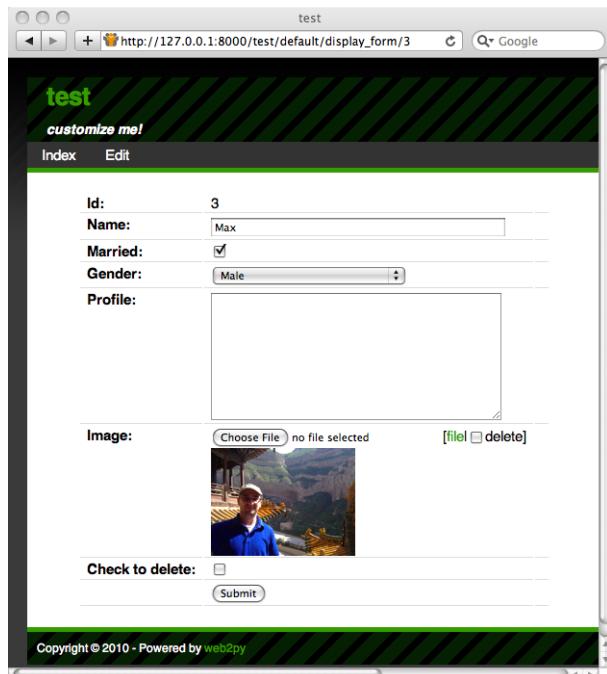
Agora, insira um novo registro na URL:

¹ http://127.0.0.1:8000/test/default/display_form

Fazer upload de uma imagem, enviar o formulário, e depois editar o registro recém-criada, visite:

¹ http://127.0.0.1:8000/test/default/display_form/3

(Aqui assumimos o último registro tem id = 3). O formulário irá exibir uma visualização da imagem como mostrado abaixo:



Este formulário, quando serializado, gera o seguinte 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 contém um link para permitir o download do arquivo carregado, e uma opção para remover o arquivo a partir do registro de dados, assim, armazenar NULL no campo "image".

Porque é que este mecanismo exposto? Por que você precisa para escrever a função de download? Porque você pode querer impor um mecanismo de autorização em função de download. Consulte o Capítulo 8 para obter um exemplo.

8.2.4 Armazenar o nome original

web2py armazena automaticamente o nome do arquivo original dentro do arquivo UUID novo e recupera quando o arquivo é baixado. Após o download, o arquivo original é armazenado no cabeçalho de disposição do conteúdo da resposta HTTP. Isso tudo é feito de forma transparente sem a necessidade de programação.

Ocasionalmente, você pode querer armazenar o nome do arquivo original em um campo de banco de dados. Neste caso, você precisará modificar o modelo e adicionar um campo para armazená-lo em:

```

1 db.define_table('person',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('image_filename'),
4     Field('image', 'upload'))

```

então você precisa modificar o controlador de lidar com ela:

```

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 o SQLFORM não exibir o campo "image_filename". "display_form" A ação move o arquivo do `request.vars.image` para o `form.vars.image_filename`, assim que é processado por `accepts` e armazenadas no banco de dados. A função de download, antes de servir o arquivo, verifica no banco de dados para o nome original e usa-lo no cabeçalho de disposição do conteúdo.

8.2.5 autodelete

O SQLFORM, a exclusão de um registro, não excluir o arquivo físico carregado (s) referenciado pelo registro. A razão é que web2py não sabe se o arquivo é utilizado o mesmo / ligados por outras tabelas ou usadas para outros fins. Se você sabe que é seguro excluir o arquivo real, quando o correspondente registro é excluído, você pode fazer o seguinte:

```
1 db.define_table('image',
2     Field('name', requires=IS_NOT_EMPTY()),
3     Field('file','upload',autodelete=True))
```

O atributo `autodelete False` é por padrão. Quando definido para `True` é torna-se o arquivo é excluído quando o registro é excluído.

8.2.6 Links para referenciar Records

Agora considere o caso de duas tabelas relacionadas por um campo de referência. Por exemplo:

```
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')

```

Uma pessoa que tem cães, cada cão pertence a um proprietário, que é uma pessoa. O proprietário de cão é necessário fazer referência a um db.person.id válido por '%(name)s'.

Vamos usar a interface **appadmin** para este aplicativo para adicionar algumas pessoas e seus cães.

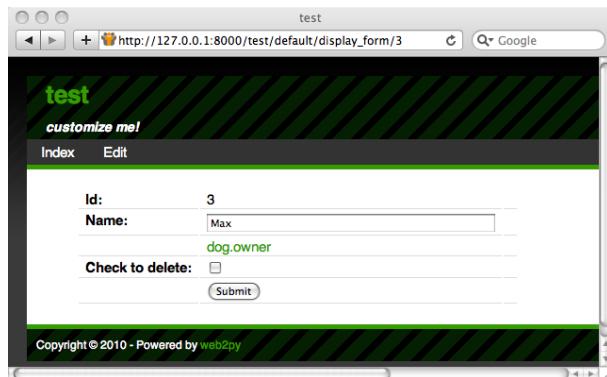
Quando a pessoa editar um já existente, a forma **appadmin** UPDATE mostra um link para uma página que lista os cães que pertencem à pessoa. Esse comportamento pode ser replicado usando o argumento do linkto SQLFORM. linkto tem que apontar para o URL de uma nova ação que recebe uma string de consulta do SQLFORM e lista os registos correspondentes. Aqui está um exemplo:

```

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)

```

Aqui está a página:



Existe um link chamado "dog.owner". O nome deste link pode ser alterado através do argumento do `labels` SQLFORM, por exemplo:

```
1 labels = {'dog.owner': "This person's dogs"}
```

Se você clicar no link você tem dirigido a:

```
1 /test/default/list_records/dog?query=dog.owner%3D5
```

"list_records" é a ação especificada, com o jogo `request.args(0)` com o nome da tabela de referência e definir `request.vars.query` para a seqüência de consulta SQL. A cadeia de consulta no URL contém o valor "dog.owner=5" adequadamente url codificado (web2py decodifica isso automaticamente quando o URL é analisado).

Você pode facilmente implementar uma ação muito geral "list_records" da seguinte forma:

```
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)
```

com o "default/list_records.html" vista associada:

```
1 {{extend 'layout.html'}}
2 {{=records}}
```

Quando um conjunto de registros é retornado por uma seleção e serializada em um ponto de vista, é convertido primeiro em um objeto SQLTABLE (não o mesmo como uma tabela) e, em seguida, serializado em uma tabela HTML, onde cada campo corresponde a uma coluna da tabela.

8.2.7 Pré-preenchimento do formulário

É sempre possível a pré-preencher um formulário usando a sintaxe:

```
1 form.vars.name = 'fieldvalue'
```

Declarações como a citada acima deve ser inserida após a declaração de forma e antes que o formulário for aceita, ou não o campo ("name" no exemplo) é explicitamente visualizado no formulário.

8.2.8 SQLFORM sem banco de dados IO

Há momentos em que você deseja gerar um formulário de uma tabela do banco de dados usando o SQLFORM e quiser validar um formulário apresentado nesse sentido, mas você não quer que qualquer inserção automática / UPDATE / DELETE no banco de dados. Este é o caso, por exemplo, quando um dos campos deve ser computado a partir do valor de outros campos de entrada. Este é também o caso quando você precisa executar uma validação adicional dos dados inseridos, que não pode ser alcançada através de validadores padrão.

Isso pode ser feito facilmente, quebrando:

```
1 form = SQLFORM(db.person)
2 if form.accepts(request.vars, session):
3     response.flash = 'record inserted'
```

em:

```
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'
```

O mesmo pode ser feito para UPDATE / DELETE formas de ruptura:

```
1 form = SQLFORM(db.person, record)
2 if form.accepts(request.vars, session):
3     response.flash = 'record updated'
```

em:

```
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'
```

Em ambos os casos trata web2py com o armazenamento e mudança de nome do arquivo carregado como se dbio=True, o cenário padrão. O nome do arquivo é carregado em:

```
1 form.vars.fieldname
```

8.3 SQLFORM.factory

Há casos em que você deseja gerar formas *as if* você tivesse uma tabela do banco de dados, mas você não quer que a tabela de banco de dados. Você simplesmente quer aproveitar a capacidade SQLFORM gerar um bom olhar CSS forma amigável e, talvez, executar o upload de arquivos e renomeação.

Isso pode ser feito através de um `form_factory`. Aqui está um exemplo onde você gera a forma, realizar a validação, upload de um arquivo e armazenar tudo no `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)

```

Aqui está o ponto de vista "default/form_from_factory.html":

```

1 {{extend 'layout.html'}}
2 {{=form}}

```

Você precisa usar um sublinhado em vez de um espaço para etiquetas de campo, ou explicitamente passar um dicionário de labels a form_factory, como você faria com um SQLFORM. Por SQLFORM.factory padrão gera o formulário usando o "id" attributes generated as if the form was generated from a table called "no_table" html. Para alterar este nome de tabela dummy, use o atributo table_name para a fábrica:

```

1 form = SQLFORM.factory(...,table_name='other_dummy_name')

```

Alterar o table_name é necessário se você precisar colocar duas fábricas geraram formas na mesma tabela e quer evitar conflitos CSS.

8.4 CRUD

Uma das recentes adições ao web2py é o Create / Read / Update / Delete (CRUD) API em cima de SQLFORM. CRUD cria um SQLFORM, mas simplifica a codificação, porque incorpora a criação do formulário, o processamento do formulário, a notificação, e à reorientação, tudo em uma única função.

A primeira coisa a notar é que difere do CRUD outras APIs web2py temos

utilizado até agora porque é que ainda não foram expostas. Deve ser importado. Também deve ser ligado a um banco de dados específico. Por exemplo:

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

O primeiro argumento do construtor é o `globals()` contexto atual de modo que CRUD pode acessar o local pedido, resposta e sessão. O segundo argumento é um objeto de conexão do banco de dados, `db`.

O objeto acima definido `crud` oferece o seguinte API:

- `crud.tables()` retorna uma lista de tabelas definidos no banco de dados.
- retorna `crud.create(db.tablename)` criar um formulário para `tablename` tabela.
- `crud.read(db.tablename, id)` retorna uma forma `readonly` para `tablename` e ID de registro.
- `crud.update(db.tablename, id)` retorna um formulário de atualização para `tablename` e ID de registro.
- `crud.delete(db.tablename, id)` exclui o registro.
- `crud.select(db.tablename, query)` retorna uma lista de registros selecionados da tabela.
- `crud.search(db.tablename)` retorna aa tupla (forma, os registros), onde a forma é uma forma de pesquisa e registros de uma lista de registros com base no formulário de pesquisa apresentados.
- `crud()` retorna um dos acima com base no `request.args()`.

Por exemplo, as seguintes ações:

```
1 def data: return dict(form=crud())
```

exporia as seguintes URLs:

```

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]
```

No entanto, as seguintes acções:

```

1 def create_tablename:
2     return dict(form=crud.create(db.tablename))
```

apenas expor o método de criação

```
1 http://.../[app]/[controller]/create_tablename
```

Enquanto as seguintes acções:

```

1 def update_tablename:
2     return dict(form=crud.update(db.tablename, request.args(0)))
```

apenas expor o método de atualização

```
1 http://.../[app]/[controller]/update_tablename/[id]
```

e assim por diante.

O comportamento do CRUD podem ser personalizados de duas maneiras: através da criação de alguns atributos do objeto crud ou passar parâmetros extra para cada um dos seus métodos.

8.4.1 Configurações

Aqui está uma lista completa de atributos do CRUD atual, seus valores padrão, e significado:

Para impor a autenticação em todas as formas requeijão:

```
1 crud.settings.auth = None
```

A utilização é explicada no capítulo 8.

Para especificar o controlador, que define a função que a data, que retorna o objeto crud

```
1 crud.settings.controller = 'default'
```

Para especificar o URL para redirecionar depois de um historial de sucesso "create":

```
1 crud.settings.create_next = URL('index')
```

Para especificar o URL para redirecionar depois de um historial de sucesso "update":

```
1 crud.settings.update_next = URL('index')
```

Para especificar o URL para redirecionar depois de um historial de sucesso "delete":

```
1 crud.settings.delete_next = URL('index')
```

Para especificar a URL a ser usado para ligar upload de arquivos:

```
1 crud.settings.download_url = URL('download')
```

Para especificar as funções extra para ser executado após os procedimentos de validação de formulários Standrad crud.create:

```
1 crud.settings.create_onvalidation = StorageList()
```

StorageList é o mesmo que um objeto Storage, ambos são definidos no arquivo de "gluon/storage.py", mas é padrão para uma [] em vez de None. Permite a seguinte sintaxe:

```
1 crud.settings.create_onvalidation.mytablename.append(lambda form:....)
```

Para especificar as funções extra para ser executado após os procedimentos de validação de formulários Standrad crud.update:

```
1 crud.settings.update_onvalidation = StorageList()
```

Para especificar as funções extra a ser executada após a conclusão dos formulários crud.create:

```
1 crud.settings.create_onaccept = StorageList()
```

Para especificar as funções extra a ser executada após a conclusão dos formulários crud.update:

```
1 crud.settings.update_onaccept = StorageList()
```

Para especificar as funções extra a ser executada após a conclusão do crud.update se o registro é excluído:

```
1 crud.settings.update_ondelete = StorageList()
```

Para especificar as funções extra a ser executada após a conclusão do crud.delete:

```
1 crud.settings.delete_onaccept = StorageList()
```

Para determinar se o botão do "update" forms should have a "delete":

```
1 crud.settings.update_deletable = True
```

Para determinar se as formas "update" deve mostrar o id do registro editado:

```
1 crud.settings.showid = False
```

Para determinar se os formulários devem manter os valores previamente inseridos ou redefinidas para padrão após a apresentação bem-sucedida:

```
1 crud.settings.keepvalues = False
```

Você pode mudar o estilo de formulário

```
1 crud.settings.formstyle = 'table3cols' or 'table2cols' or 'divs' or 'ul'
```

Você pode adicionar captcha formas, utilizando a mesma convenção expli-
cada por autoridades, com:

```
1 crud.settings.create_captcha = None  
2 crud.settings.update_captcha = None  
3 crud.settings.captcha = None
```

8.4.2 Mensagens

Aqui está uma lista de mensagens personalizáveis:

```
1 crud.messages.submit_button = 'Submit'
```

define o texto do botão "submit" para criar e actualizar as formas.

```
1 crud.messages.delete_label = 'Check to delete:'
```

define o rótulo das formas "delete" button in "update".

```
1 crud.messages.record_created = 'Record Created'
```

define a mensagem de flash para a criação de registro de sucesso.

```
1 crud.messages.record_updated = 'Record Updated'
```

define a mensagem de atualização de registro em flash com sucesso.

```
1 crud.messages.record_deleted = 'Record Deleted'
```

define a mensagem de flash em supressão registro bem-sucedido.

```
1 crud.messages.update_log = 'Record %(id)s updated'
```

define a mensagem de log on update record bem sucedido.

```
1 crud.messages.create_log = 'Record %(id)s created'
```

define a mensagem de log de criação do registro do bem-sucedido.

```
1 crud.messages.read_log = 'Record %(id)s read'
```

define a mensagem de log de registro de acesso de leitura bem-sucedida.

```
1 crud.messages.delete_log = 'Record %(id)s deleted'
```

define a mensagem de log de registro de eliminação bem-sucedida.

Observe que `crud.messages` `gluon.storage.Message` pertence à classe que é semelhante ao `gluon.storage.Storage`, mas ele automaticamente traduz os seus valores, sem a necessidade de o operador `T`. mensagens de log são usados somente se CRUD está ligado ao Auth como discutido no Capítulo 8. Os eventos são registrados no "auth_events" tabela Auth.

8.4.3 Métodos

O comportamento dos métodos CRUD também pode ser personalizado em uma base por chamada. Aqui estão suas assinaturas:

```
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` é uma tabela ou uma DAL tablename o método deve agir.
- `record` e `record_id` são o id do registro o método deve agir.
- `next` é o URL para redirecionar para depois do sucesso. Se o URL contém a substring "[id]" este será substituído pelo id do registro atualmente criado / atualizado.
- `onvalidation` tem a mesma função que SQLFORM onvalidation (...)

- onaccept é uma função para ser chamado após o envio do formulário é aceite e posta em prática, mas antes de redirecionamento.
- log é a mensagem de log. Mensagens de log em CRUD ver variáveis no dicionário como o form.vars "%(id)s".
- message é a mensagem flash, após aceitação do formulário.
- ondelete é chamado no lugar do onaccept quando um registro é eliminado através de um formulário "update".
- deletable determina se o formulário "update" deve ter uma opção de apagar.
- query é a consulta a ser utilizado para selecionar registros.
- fields é uma lista de campos a serem selecionados.
- orderby determina a ordem em que os registros devem ser seleccionados (ver Capítulo 6).
- limitby determina o intervalo de registros selecionados que devem ser exibidas (consulte o Capítulo 6).
- headers é um dicionário com os nomes de cabeçalho da tabela.
- uma lista contendo queries ['equals', 'not equal', 'contains'] como os métodos permitidos no formulário de busca.
- query_labels query_labels=dict>equals='Equals') como um dicionário de nomes que dá aos métodos de pesquisa.
- fields lista de campos a serem listados no widget de busca.
- field_labels um campo de mapeamento de dicionário de nomes em rótulos.
- padrões para zero "choose one" é usado como opção padrão para o dropdown no widget de busca.

Aqui está um exemplo do uso de uma função de controlador único:

```
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)
```

Aqui está outro controlador de função muito genérico que permite pesquisar, criar e editar os registros de qualquer tabela, onde o tablename é passado 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 o que diz a linha `table.id.represent=...` web2py para mudar a representação do campo id e exibir um link em vez disso, a página em si e passa o id como `request.args (1)` que transforma a criar a página em uma página de atualização.

8.4.4 Record Versioning

CRUD fornece um utilitário de banco de dados registros versão:

Se você tem uma tabela (`db.mytable`) que necessita de revisão completa da história você pode simplesmente fazer:

```
1 form=crud.update(db.mytable,myrecord,onaccept=crud.archive)
```

`crud.archive` define uma nova tabela chamada **db.mytable_history** (o nome é derivado da tabela acima) e uma loja a atualização, que armazena uma cópia do registro (antes da atualização), no recém-criado de mesa, incluindo uma referência para o registro atual.

Porque o registro é realmente atualizado (apenas seu estado anterior é arquivado), as referências não são quebrados.

Isso é feito sob o capô, mas você pode precisar acessar a tabela de histórico. Se você deseja acessar tabela, você precisa defini-la no modelo:

```
1 db.define_table('mytable_history',
2     Field('current_record',db.mytable),
3     db.mytable)
```

Observe a tabela se estende `db.mytable`, ou seja, inclui todos os seus campos, e tem uma referência para o `current_record`. `crud.archive` não timestamp o registro armazenado a menos que a tabela original tem o campo `timestamp`, por exemplo:

```
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),
```

Não há nada especial sobre esses campos, você pode dar qualquer nome que você gosta. Eles são preenchidos antes que o registro é arquivado e são arquivados com cada cópia do registro.

Se você quiser mudar o nome da tabela história e / ou o nome do campo de referência que você pode fazer isso da seguinte maneira:

```
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'))
```

8.5 Os formulários personalizados

Se um formulário é criado com SQLFORM, SQLFORM.factory ou CRUD, existem várias maneiras que podem ser incorporados em uma exibição que permite vários graus de personalização. Considere, por exemplo, o seguinte modelo:

```

1 db.define_table('image',
2     Field('name'),
3     Field('file', 'upload'))

```

e

```

1 def upload_image():
2     return dict(form=crud.create(db.image))

```

ação upload

A maneira mais simples para inserir o formulário na vista para upload_image é

```

1 {{=form}}

```

Isso resulta em um layout de tabela padrão. Se você quiser usar um layout diferente, você pode quebrar a forma em 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}}

```

Onde fica form.custom.widget[fieldname] serializada na widget adequada para o campo. Se o formulário é submetido e que contém erros, eles são acrescentados a seguir a widgets, como de costume.

O formulário de exemplo acima é mostrado na imagem abaixo.

The screenshot shows a web page with a form for uploading an image. The form consists of two main sections. The first section is labeled "Image name:" followed by an empty input field. The second section is labeled "Image file:" followed by a "Choose File" button and the text "no file selected". Below these sections is a link labeled "Click here to upload:" followed by a "Submit" button.

Observe que um cound resultado semelhante foram obtidos com:

```
1 crud.settings.formstyle='table2cols'
```

sem usar um formulário personalizado. Outros possíveis são `formstyles` "table3cols" (the default), "divs" and "ul".

Se você não quiser usar o widgets serializado por web2py, você pode substituí-las com HTML. Existem algumas variáveis que serão úteis para isso:

- `form.custom.label[fieldname]` contém o rótulo para o campo.
- `form.custom.dspval[fieldname]` formulário-tipo e representação de exibição de campo depende do tipo do campo.
- `form.custom.inpval[fieldname]` formulário-tipo de campo e os valores dependem do tipo a ser usado no código de campo.

É importante seguir as convenções descritas a seguir.

8.5.1 CSS Convenções

Tags em formulários gerados por SQLFORM, SQLFORM.factory e CRUD seguem um rigoroso CSS convenção de nomenclatura que pode ser usado para personalizar ainda mais as formas.

Com base em uma tabela "mytable", a field "myfield" of type "string", ele é processado por omissão de um

```
1 SQLFORM.widgets.string.widget
```

parecida com esta:

```
1 <input type="text" name="myfield" id="mytable_myfield"
2   class="string" />
```

Observe que:

- a classe da tag INPUT é o mesmo que o tipo do campo. Isto é muito importante para o código do jQuery em 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" apresentar o calendário popup.
- o id é o nome da classe mais o nome do campo, unidas por um sublinhado. Isto permite-lhe unicamente se referem ao campo, através, por exemplo, a `jQuery('#mytable_myfield')` e manipular a folha de estilo do campo ou vincular as ações associadas aos eventos de campo (foco, borrão, keyup, etc.)
- o nome é, como seria de esperar, o nome do campo.

8.5.2 Ocultar erros

Ocasionalmente, você pode querer desativar a colocação automática de erro e exibir mensagens de erro forma, em algum lugar diferente do padrão. Isso pode ser feito facilmente.

- No caso da forma ou SQLFORM, passe para o método `hiderror=True` `accepts`.
- No caso do requeijão, conjunto `crud.settings.hiderror=True`

Você também pode querer modificar os pontos de vista para mostrar o erro (uma vez que não são mais exibidos automaticamente).

Aqui está um exemplo onde os erros são exibidos acima da forma e não na 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}}
```

Os erros serão mostrados como na imagem abaixo.

Your submitted form contains the following errors:

- name error: enter a value

Name:	<input type="text"/>
File:	<input type="file"/> Choose File no file selected
	<input type="button"/> Submit

Este mecanismo também funciona para formulários personalizados.

8.6 Validadores

Os validadores são classes usadas para validar campos de entrada (incluindo as formas geradas a partir de tabelas de banco de dados).

Aqui está um exemplo de como usar um validador com um FORM:

```
1 INPUT(_name='a', requires=IS_INT_IN_RANGE(0, 10))
```

Aqui está um exemplo de como exigir a validação para um campo de tabela:

```
1 db.define_table('person', Field('name'))  
2 db.person.name.requires = IS_NOT_EMPTY()
```

Os validadores são sempre atribuídos usando o atributo `requires` de um campo. Um campo pode ter um validador único ou múltiplo validadores. Vários são os validadores fez parte de uma lista:

Os validadores são chamados pelo `accepts` função de um objeto auxiliar `FORM` ou outro HTML que contém um formulário. Eles são chamados pela ordem na qual eles são listados.

Built-in validadores ter construtores que levam o `error_message` argumento opcional, que permite substituir a mensagem de erro padrão.

Aqui está um exemplo de um validador em uma tabela do banco de dados:

```
1 db.person.name.requires = IS_NOT_EMPTY(error_message=T('fill this!'))
```

onde temos usado o operador tradução `T` para permitir a internacionalização. Observe que as mensagens de erro padrão não são traduzidos.

8.6.1 Validadores

IS_ALPHANUMERIC

Este validador verifica que o valor do campo contém caracteres só nos intervalos az, AZ ou 0-9.

```
1 requires = IS_ALPHANUMERIC(error_message=T('must be alphanumeric!'))
```

IS_DATE

Este validador verifica que um valor de campo contém uma data válida no formato especificado. É uma boa prática para especificar o formato usando o operador de tradução, a fim de suportar os formatos diferentes em locais diferentes.

```
1 requires = IS_DATE(format=T('%Y-%m-%d'),
2 error_message=T('must be YYYY-MM-DD!'))
```

Para uma descrição completa sobre as directivas% sob o olhar validador `IS_DATETIME`.

IS_DATE_IN_RANGE

Funciona muito parecido com o validador anterior, mas permite especificar um intervalo:

```
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 uma descrição completa sobre as directivas% sob o olhar validador IS_DATETIME.

IS_DATETIME

Este validador verifica que um campo contém um valor datetime válido no formato especificado. É uma boa prática para especificar o formato usando o operador de tradução, a fim de suportar os formatos diferentes em locais diferentes.

```
1 requires = IS_DATETIME(format=T('%Y-%m-%d %H:%M:%S'),  
2                         error_message=T('must be YYYY-MM-DD HH:MM:SS!'))
```

Os seguintes símbolos podem ser usados para a seqüência 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

Funciona muito parecido com o validador anterior, mas permite especificar um intervalo:

```
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 uma descrição completa sobre as directivas% sob o olhar validador IS_DATETIME.

IS_DECIMAL_IN_RANGE

O

```
1 INPUT(_type='text', _name='name', requires=IS_DECIMAL_IN_RANGE(0, 10))
```

determinar que o argumento é (ou pode ser representado como) um Decimal Python, e que se enquadra no intervalo especificado, inclusive. A comparação é feita com Python aritmética decimal.

Os limites mínimo e máximo podem ser None, que significa que nenhum limite inferior ou superior, respectivamente.

IS_EMAIL

Ele verifica que o valor do campo se parece com um endereço de e-mail. Ele não tenta enviar um email para confirmar.

```
1 requires = IS_EMAIL(error_message=T('invalid email!'))
```

IS_EQUAL_TO

Verifica se o valor validado é igual a um determinado valor (que pode ser aa variável):

```
1 requires = IS_EQUAL_TO(request.vars.password,
2                         error_message=T('passwords do not match'))
```

IS_EXPR

O primeiro argumento é uma string contendo uma expressão lógica em termos de valor variável. É valida um valor de campo se a expressão é avaliada como True. Por exemplo:

```
1 requires = IS_EXPR('int(value)%3==0',
2                     error_message=T('not divisible by 3'))
```

Um deve primeiro verificar que o valor é um número inteiro de modo que uma exceção não ocorrerá.

```
1 requires = [IS_INT_IN_RANGE(0, 100), IS_EXPR('value%3==0')]
```

IS_FLOAT_IN_RANGE

Verifica se o valor do campo é um número de ponto flutuante dentro de um intervalo definido, $0 < \text{value} < 100$ no seguinte exemplo:

```
1 requires = IS_FLOAT_IN_RANGE(0, 100,
2                             error_message=T('too small or too large!'))
```

IS_INT_IN_RANGE

Verifica se o valor do campo é um número inteiro dentro de um intervalo definido, $0 < \text{value} < 100$ no exemplo a seguir:

```
1 requires = IS_INT_IN_RANGE(0, 100,
2                             error_message=T('too small or too large!'))
```

IS_IN_SET

Verifica se os valores de campo são um conjunto:

```
1 requires = IS_IN_SET(['a', 'b', 'c'], zero=T('choose one'),
2                      error_message=T('must be a or b or c'))
```

O argumento zero é opcional e que determina o texto da opção selecionada por padrão, uma opção que não é aceito pelo validador IS_IN_SET em si. Se você não quiser uma opção "choose one", zero=False conjunto

A opção zero foi introduzida na revisão (1.67.1). Ele não quebrar compatibilidade com versões anteriores no sentido de que não quebrou aplicações, mas o fez mudar seu comportamento, pois, antes, não havia opção zero.

Os elementos do conjunto deve ser sempre cordas a menos que este validador é precedido por `IS_INT_IN_RANGE` (que converte o valor para int) ou `IS_FLOAT_IN_RANGE` (que converte o valor para float). Por exemplo:

```
1 requires = [IS_INT_IN_RANGE(0, 8), IS_IN_SET([2, 3, 5, 7],  
2     error_message='must be prime and less than 10'))]
```

`IS_IN_SET` e marcação

O validador `IS_IN_SET` `multiple=False` tem um atributo opcional. Se for definido como True, vários valores podem ser armazenados em um campo. O campo neste caso deve ser um campo string. Os valores múltiplos são armazenados separados por um "|". `multiple` referências são feitas automaticamente em criar e actualizar as formas, mas eles são transparentes para o DAL. Nós sugerimos fortemente utilizando o plugin jQuery multiselect para processar vários campos.

`IS_LENGTH`

Verifica se o comprimento do valor do campo se enquadra entre os limites dados. Funciona tanto para arquivos de texto e insumos.

Seus argumentos são:

- `maxsize`: o máximo permitido comprimento / tamanho
- `minsize`: o tamanho mínimo permitido tamanho /

Exemplos: Verificar se a string de texto é menor do que 33 caracteres:

```
1 INPUT(_type='text', _name='name', requires=IS_LENGTH(32))
```

Verifique se a seqüência de senha é mais do que 5 caracteres:

```
1 INPUT(_type='password', _name='name', requires=IS_LENGTH(minsize=6))
```

Verifique se o arquivo enviado tem tamanho entre 1KB e 1MB:

```
1 INPUT(_type='file', _name='name', requires=IS_LENGTH(1048576, 1024))
```

Para todos os tipos de campo, exceto para os arquivos, ele verifica o comprimento do valor. No caso dos arquivos, o valor é uma `cookie.FieldStorage`, assim que valida o comprimento dos dados no arquivo, que é o comportamento pode-se esperar intuitivamente.

IS_LIST_OF

Isto não é propriamente um validador. Sua utilização é permitir que as validações dos campos que retornam vários valores. É utilizado em casos raros quando um formulário contém vários campos com o mesmo nome ou uma caixa de seleção múltipla. Seu argumento é apenas outro validador, e tudo que ele faz é aplicar o validador outros a cada elemento da lista. Por exemplo, a seguinte expressão verifica que cada item em uma lista é um inteiro na faixa 0-10:

```
1 requires = IS_LIST_OF(IS_INT_IN_RANGE(0, 10))
```

Elá nunca retornará um erro e não contém uma mensagem de erro. O validador interior controla a geração de erro.

IS_LOWER

Este validador nunca retornará um erro. Ele apenas converte o valor para minúsculas.

```
1 requires = IS_LOWER()
```

IS_MATCH

Este validador corresponde ao valor contra uma expressão regular e retorna um erro se ele não corresponder. Aqui está um exemplo de uso para validar um código postal E.U.:

```
1 requires = IS_MATCH('^\d{5}(-\d{4})?$',  
2 error_message='not a zip code')
```

Aqui está um exemplo de uso para validar um endereço IPv4:

```
1 requires = IS_MATCH('^\d{1,3}(\.\d{1,3}){3}$',
2                     error_message='not an IP address')
```

Aqui está um exemplo de uso para validar um número de telefone E.U.:

```
1 requires = IS_MATCH('^1?((-)\d{3}-?)|((\d{3}))\d{3}-?\d{4}$',
2                     error_message='not a phone number')
```

Para obter mais informações sobre expressões regulares em Python, consulte a documentação oficial do Python.

IS_NOT_EMPTY

Este validador verifica que o conteúdo do valor do campo não é uma seqüência vazia.

```
1 requires = IS_NOT_EMPTY(error_message='cannot be empty!')
```

IS_TIME

Este validador verifica que um valor de campo contém uma hora válida no formato especificado.

```
1 requires = IS_TIME(error_message=T('must be HH:MM:SS!'))
```

IS_URL

Rejeita uma seqüência de URL, se alguma das seguintes for verdadeira:

- A seqüência está vazia ou Nada
- A seqüência de caracteres que usa não são permitidos em uma URL
- A seqüência de quebrar qualquer das regras sintáticas HTTP
- O esquema de URL especificado (se for especificado) não é 'http' ou 'https'
- O domínio de nível superior (se for um nome de host é especificado) não existe

(Essas regras se baseiam no RFC 2616 (65))

Esta função somente verifica a sintaxe de URL's. Ele não verifica se o URL aponta para um documento real, por exemplo, ou que fizer sentido semântico. Esta função é automaticamente prepend "http://" na frente de uma URL, no caso de uma URL abreviada (por exemplo, "google.ca").

Se o parâmetro mode = 'genérico' é usado, então esta função altera o comportamento. Em seguida, ele rejeita uma seqüência de URL, se alguma das seguintes for verdadeira:

- A seqüência está vazia ou Nada
- A seqüência de caracteres que usa não são permitidos em uma URL
- O esquema de URL especificado (se for especificado) não é válido

(Essas regras se baseiam no RFC 2396 (66))

A lista dos regimes autorizados é personalizável com o parâmetro allowed_schemes. Se você excluir nenhum da lista, em seguida abreviado URLs (por falta de um regime como o "http") serão rejeitadas.

O padrão prefixado regime é personalizável com o parâmetro prepend_scheme. Se você definir prepend_scheme para Nenhum e, em seguida prepending será desativado. URLs que exigem prepending analisar ainda será aceito, mas o valor de retorno não será modificado.

IS_URL é compatível com o nome de domínio internacionalizados (IDNs) padrão especificado no RFC 3490 (67)). Como resultado, URLs podem ser seqüências regulares ou strings unicode. Se o URL do domínio do componente (google.ca por exemplo), inclui-US-ASCII letras não, então o domínio será convertido em Punycode (definido na RFC 3492 (68)). IS_URL vai um pouco além das normas, e permite-US-ASCII não estar presente no caminho e componentes de consulta da URL também. Esses US-ASCII caracteres não serão codificados. Por exemplo, o espaço será codificada como "%20";

Os caracteres Unicode com o código hexadecimal ox4e86 será '% 4e 86%.

Exemplos:

```

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')
```

Se o check é definida como True verifica se o valor validado é uma lesma (que permite apenas caracteres alfanuméricos e os traços não-repetição).

Se o checks é definido como False (padrão) é converte o valor de entrada para uma lesma.

IS_STRONG

Reforça a requisitos de complexidade em um campo (normalmente um campo de senha)

Exemplo:

```
1 requires = IS_STRONG(min=10, special=2, upper=2)
```

onde

- min é o comprimento mínimo do valor
- especial é o número mínimo de caracteres especiais necessários caracteres especiais são algumas das !@#\$%&*(){}[]-+ followiing
- superior é o número mínimo de caracteres maiúsculos

IS_IMAGE

Este validador verifica se o arquivo carregado através da entrada arquivo foi salvo em um dos formatos de imagem selecionada e tem dimensões (largura e altura) dentro de determinados limites.

Ele não verifica o tamanho máximo do arquivo (IS_LENGTH uso para isso). Ele retorna uma falha na validação de dados se não foi carregado. Ele suporta os formatos BMP, GIF, JPEG, PNG, e não requer a Python Imaging Library.

Código peças retiradas ref. (69)

Leva os seguintes argumentos:

- extensões: iteráveis contendo permitiu extensões de arquivo de imagem em minúsculas
- maxsize: iteráveis com largura máxima e altura da imagem
- minsize: iteráveis contendo mínimo de largura e altura da imagem

Use (-1, -1), minsize para ignorar a verificação do tamanho da imagem.

Aqui estão alguns exemplos:

- Verifique se o arquivo é carregado em qualquer um dos formatos de imagem suportados:

```
1 requires = IS_IMAGE()
```

- Verifique se o arquivo enviado é ou JPEG ou PNG:

```
1 requires = IS_IMAGE(extensions=('jpeg', 'png'))
```

- Verifique se o arquivo enviado é PNG com tamanho máximo de pixels de 200x200:

```
1 requires = IS_IMAGE(extensions=('png'), maxsize=(200, 200))
```

IS_UPLOAD_FILENAME

Este validador verifica se o nome ea extensão do arquivo enviado através da entrada de arquivo corresponde dado critérios.

Ele não garante o tipo de arquivo de qualquer forma. Retorna falha na validação de dados se não foi carregado.

Seus argumentos são:

- Nome: nome do arquivo (antes do ponto) regex.
- extensão: a extensão (após o ponto) regex.
- lastdot: que ponto deve ser usado como um nome de arquivo / separador de extensão: indica True último ponto (por exemplo, "file.tar.gz" will be broken in "file.tar"+"gz"), enquanto que significa False primeiro ponto (tar.gz, por exemplo, "file.tar.gz" will be broken into "file"+").
- caso: o significa manter o caso; 1 significa transformar a string em minúsculas (padrão), 2 significa transformar a string em maiúsculas.

Se não houver nenhum ponto em apreço, verifica a extensão será feito contra a seqüência vazia e cheques filename contra valor.

Exemplos:

Verifique se o arquivo tem uma extensão pdf (case insensitive):

```
1 requires = IS_UPLOAD_FILENAME(extension='pdf')
```

Verifique se o arquivo tem uma extensão tar.gz e nome começando com backup:

```
1 requires = IS_UPLOAD_FILENAME(filename='backup.*', extension='tar.gz', lastdot=False)
```

Verifique se o arquivo não tem extensão e nome correspondente README (case sensitive):

```
1 requires = IS_UPLOAD_FILENAME(filename='^README$', extension='^$', case=0)
```

IS_IPV4

Este validador verifica se o valor de um campo é um endereço IP versão 4 em forma decimal. Pode ser definido como força de endereços de um determinado intervalo. regex IPv4 retirado ref. (70) Seus argumentos são:

- menor endereço permitiu `minip`; aceita: str, por exemplo, 192.168.0.1; iteráveis de números, por exemplo, [192, 168, 0, 1], int, por exemplo, 3232235521
- Endereço mais alto permitido `maxip`, mesmo que acima

Todos os três valores de exemplo são iguais, dado que os endereços são convertidos para inteiros, para verificação de inclusão com a seguinte função:

```
1 number = 16777216 * IP[0] + 65536 * IP[1] + 256 * IP[2] + IP[3]
```

Exemplos:

Verifique se há endereço IPv4 válido:

```
1 requires = IS_IPV4()
```

Verifique o endereço de rede IPv4 privado válidos:

```
1 requires = IS_IPV4(minip='192.168.0.1', maxip='192.168.255.255')
```

IS_LOWER

Este validador nunca retornará um erro. Ele converte o valor para minúsculas.

IS_UPPER

Este validador nunca retornará um erro. Ele converte o valor para maiúsculas.

```
1 requires = IS_UPPER()
```

IS_NULL_OR

Obsoleto, um apelido para IS_EMPTY_OR descrito abaixo.

IS_EMPTY_OR

Às vezes você precisa para permitir valores em branco em um campo junto com outras exigências. Por exemplo, um campo pode ser uma data, mas também pode ser vazio. O validador IS_EMPTY_OR permite isso:

```
1 requires = IS_NULL_OR(IS_DATE())
```

CLEANUP

Este é um filtro. Ela nunca falha. Ele apenas remove todos os caracteres cujo código ASCII decimal não estão na lista [10, 13, 32-127].

```
1 requires = CLEANUP()
```

CRYPT

Este também é um filtro. Ele executa um hash seguro na entrada e é usado para evitar que as senhas sejam passadas em claro para o banco de dados.

```
1 requires = CRYPT()
```

Se a chave não é especificada, ela usa o algoritmo MD5. Se uma chave for especificado CRYPT usa o algoritmo HMAC. A chave pode intain um prefixo que determina o algoritmo HMAC para uso com, por exemplo SHA512:

```
1 requires = CRYPT(key='sha512:thisisthekey')
```

Esta é a sintaxe recomendada. A chave tem que ser uma string única associada ao banco de dados utilizado. A chave não pode ser mudado. Se você perder a chave dos valores anteriormente hash se tornam inúteis.

8.6.2 Database Validators

IS_NOT_IN_DB

Considere o seguinte exemplo:

```
1 db.define_table('person', Field('name'))
2 db.person.name.requires = IS_NOT_IN_DB(db, 'person.name')
```

Elá exige que, quando você insere uma nova pessoa, seu nome não estiver no banco de dados, db, person.name no campo. Como com todos os outros validadores esta exigência é imposta a nível de processamento do formulário, não no nível de banco de dados. Isto significa que há uma pequena probabilidade de que, se dois visitantes tentar inserir registros ao mesmo tempo com a mesma person.name, isso resulta em uma condição de corrida e ambos os registros são aceitas. É, portanto, mais segura também para informar o banco de dados que este campo deve ter um valor único:

```
1 db.define_table('person', Field('name', unique=True))
2 db.person.name.requires = IS_NOT_IN_DB(db, 'person.name')
```

Agora, se uma condição de corrida ocorre, o banco de dados gera um OperationalError e uma das duas inserções é rejeitada.

O primeiro argumento do IS_NOT_IN_DB pode ser uma ligação de dados ou um conjunto. Neste último caso, você estaria marcando o conjunto definido pelo conjunto.

O código a seguir, por exemplo, não permitir o registo de duas pessoas com o mesmo nome dentro de 10 dias umas das outras:

```
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 as seguintes tabelas e exigência:

```

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'))
```

Ela é aplicada a nível de cão INSERT / UPDATE / DELETE formas. Ela exige que a dog.owner ser um ID válido no campo person.id db no banco de dados. Devido a este validador, o campo dog.owner é representado como um dropbox. O terceiro argumento de que o validador é uma string que descreve os elementos da dropbox. No exemplo que você quer ver a pessoa %(name)s ao invés do %(id)s pessoa. %(...)s é substituído pelo valor do campo entre parêntesis, para cada registro.

A opção zero funciona muito bem como para o validador IS_IN_SET.

Se quiser que o domínio validado, mas você não quer um dropbox, você deve colocar o validador em uma lista.

```
1 db.dog.owner.requires = [IS_IN_DB(db, 'person.id', '%(name)s')]
```

O primeiro argumento do validador pode ser uma ligação de dados ou um conjunto de DAL, como no IS_NOT_IN_DB.

Às vezes você deseja que o drop-box (assim você não quiser usar a lista de sintaxe acima), ainda assim você quiser usar validadores adicionais. Para este efeito, o validador IS_IN_DB_and tem um argumento extra que pode apontar para uma lista de outros validadores aplicada se o valor passa a validação validado IS_IN_DB. Por exemplo, para validar todos os proprietários de cães na db que não estão em um 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 e marcação

O validador `IS_IN_DB` `multiple=False` tem um atributo opcional. Se definido como verdadeiro múltiplos valores podem ser armazenados em um campo. O campo neste caso não pode ser uma referência, mas deve ser um campo string. Os valores múltiplos são armazenados separados por um "|". `multiple` referências são feitas automaticamente em criar e actualizar as formas, mas eles são transparentes para o DAL. Nós sugerimos fortemente utilizando o plugin jQuery multiselect para processar vários campos.

8.6.3 Custom Validators

Todos os validadores seguir o protótipo abaixo:

```
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)
```

Isto é, quando chamados a validar um valor, um validador retorna uma tupla (`x`, `y`). Se `o` `y` `None` é, o valor passou na validação e `x` contém um valor analisado. Por exemplo, se o validador exige que o valor seja um número inteiro, `x` é convertido para a `int(value)`. Se o valor não passar na validação, então `x` contém o valor de entrada e `y` contém um mensagem de erro que explica a falha na validação. Esta mensagem de erro é usado para relatar o erro em formulários que não validar.

O validador pode também conter um método `formatter`. É necessário realizar a conversão oposta ao que o `__call__` faz. Por exemplo, considere o código fonte do `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))

```

Em caso de sucesso, o método `__call__` lê um string da data a partir da forma e converte-lo em um `datetime.date` objeto usando a seqüência de formato especificado no construtor. O objeto `formatter` tem um objeto `datetime.date` e converte-lo para uma representação string usando o mesmo formato. O `formatter` é chamado automaticamente em formas, mas você também pode chamá-lo explicitamente para converter objetos em sua representação adequada. Por exemplo:

```

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

```

Quando validadores são necessários vários (e armazenados numa lista), são executadas na ordem e na saída de um é passado como entrada para o próximo. A cadeia de quebra quando um dos validadores falhar.

Inversamente, quando chamamos o método `formatter` de um campo, os formatadores dos validadores associados também estão acorrentados, mas em ordem inversa.

8.6.4 Validadores com dependências

Ocasionalmente, você precisará validar um campo e validador depende do valor de outro campo. Isso pode ser feito, mas exige configuração do validador no controlador, quando o valor do outro campo é conhecido. Por exemplo, aqui está uma página que gera um formulário de registro que pede login e senha duas vezes. Nenhum dos campos pode estar vazio, e as duas senhas devem corresponder:

```
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)
```

O mesmo mecanismo pode ser aplicado a objetos de formulário e SQLFORM.

8.7 Widgets

Aqui está uma lista de widgets disponíveis 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
```

Os primeiros dez deles são os padrões para os tipos de campo correspondente. O Widget "options" é usado quando um campo requer é IS_IN_SET ou IS_IN_DB com o comportamento padrão multiple=False (). O Widget "multiple" é usado quando um campo requer é IS_IN_SET ou IS_IN_DB com o multiple=True. Os widgets "radio" and "checkboxes" nunca são usados por padrão, mas pode ser configurado manualmente. O Widget de autocomplete é especial e discutido em sua própria seção.

, Por exemplo, a "string" tem um campo representado por um textarea:

```
1 Field('comment', 'string', widget=SQLFORM.widgets.text.widget)
```

Você pode criar novos widgets ou prorrogar widgets existentes.

SQLFORM.widgets[type] é uma classe e SQLFORM.widgets[type].widget é uma função de membro estático da classe correspondente. Cada função widget leva dois argumentos: o objeto de campo, eo valor atual do campo. Ele retorna uma representação do widget. Como exemplo, o widget string poderia ser recodificados da seguinte forma:

```
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)
```

Os valores de id e classe deve seguir a convenção descrita mais adiante neste capítulo. Um widget pode conter seu próprio validadores, mas é uma boa prática de associar a validação para o atributo "requires" do campo e têm o widget obtê-los de lá.

8.7.1 Autocomplete Widget

Há dois usos possíveis para o widget autocomplete: autocomplete para um campo que tem um valor de uma lista ou um campo autocomplete de referência (onde a seqüência a ser completada automaticamente é uma representação de referência que é implementado como um id).

O primeiro caso é 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)
```

Onde `limitby` instrui o widget para exibir mais de 10 sugestões, no momento, `min_length` e instrui o widget para realizar uma chamada Ajax para buscar sugestões somente após o usuário digitou pelo menos 2 caracteres na caixa de pesquisa.

O segundo caso é mais complexo:

```
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)
```

Neste caso, o valor do `id_field` diz o widget que, mesmo que o valor a ser completada automaticamente é uma `db.category.name`, o valor a ser armazenado é o `db.category.id` correspondente. Uma parâmetros opcionais é `orderby` que instrui o widget sobre como classificar as sugestões (em ordem alfabética por padrão).

Esse widget funciona através de Ajax. Onde está a chamada Ajax? Alguns magia está acontecendo neste widget. O retorno é um método do elemento próprio objeto. Como é exposto? Em web2py qualquer pedaço de código pode gerar uma resposta por levantar uma exceção HTTP. Este widget explora essa possibilidade da seguinte forma: o widget envia a chamada Ajax

para o mesmo URL que gerou o widget em primeiro lugar, e coloca um símbolo especial no `request.vars`. Ela recebe o widget instanciado novamente, ele encontra o token e gera uma exceção HTTP que responde ao pedido. Tudo isso é feito sob o capô e escondido para o desenvolvimento.

9

Controle de Acesso

web2py inclui um poderoso e personalizável Role Based Access Control mecanismo (RBAC).

Aqui está uma definição da Wikipédia:

“Role-Based Access Control (RBAC) é uma abordagem para restringir o acesso ao sistema aos utilizadores autorizados. É uma abordagem mais recente alternativa para controle de acesso obrigatório (MAC) e controle de acesso discricionário (DAC). RBAC é às vezes chamado de role-based segurança.

RBAC é uma política neutra e tecnologia de controle de acesso flexível suficientemente poderosa para simular DAC e MAC. Em contrapartida, o MAC pode simular RBAC se o gráfico de papel é restrito a uma árvore ao invés de um conjunto parcialmente ordenado.

Antes do desenvolvimento do RBAC, MAC e DAC foram considerados os únicos modelos conhecidos para controle de acesso: se o modelo não foi MAC, foi considerado um modelo de CAD, e vice-versa. Investigação na década de 1990 demonstraram que RBAC cai em nenhuma categoria.

Dentro de uma organização, os papéis são criados para várias funções de trabalho. As permissões para executar determinadas operações são atribuídas a funções específicas. Os membros do pessoal (ou outros usuários do sistema) são atribuídas funções específicas, e com aquelas atribuições de função adquirir as permissões para executar funções do sistema particular. Ao contrário do controle de acesso baseado em contexto (CBAC), RBAC não olhar para o contexto da mensagem (como uma fonte de conexão).

Como os usuários não são atribuídos permissões diretamente, mas apenas adquiri-los através de seu papel (ou papéis), gestão de direitos de usuário individual se torna uma questão de simplesmente atribuir funções apropriadas para o usuário, o que simplifica as operações comuns, tais como a adição de um usuário, ou alterar departamento de um usuário.

RBAC difere das listas de controle de acesso (ACLs) utilizadas em sistemas tradicionais de controle de acesso discricionário na medida em que atribui permissões para operações específicas com o significado da organização, ao invés de objetos de dados de baixo nível. Por exemplo, uma lista de controle de acesso podem ser utilizados para conceder ou negar acesso de gravação para um arquivo de sistema particular, mas não determinam como esse arquivo pode ser alterada.

A classe que implementa web2py RBAC é chamado **Auth**. necessidades **Auth** (e define) os seguintes quadros:

- Nome de usuário auth_user lojas, e-mail, senha e status (pendente de registo, aceite bloqueado)
- auth_group grupos lojas ou funções para os usuários em uma estrutura de muitos para muitos. Por padrão, cada usuário é em seu próprio grupo, mas um usuário pode estar em vários grupos e cada grupo pode conter vários usuários. Um grupo é identificado por um papel e uma descrição.
- auth_membership usuários e grupos de links em uma estrutura de muitos para muitos.
- auth_permission links grupos e permissões. A permissão é identificado

por um nome e, opcionalmente, uma mesa e um registro. Por exemplo, os membros de um determinado grupo pode ter permissões "update" em um registro específico de uma tabela específica.

- `auth_event` faz alterações em outras tabelas e acesso via sucesso CRUD para objetos controlado pelo RBAC.

Em princípio, não há nenhuma restrição sobre os nomes das funções e os nomes das permissões, o desenvolvedor pode criá-los para fixar os papéis e as permissões da organização. Depois de terem sido criados, web2py fornece uma API para verificar se um usuário está logado, se o usuário é membro de um determinado grupo e / ou se o usuário é membro de qualquer grupo que tenha uma permissão exigida. web2py também fornece decoradores para restringir o acesso a qualquer função com base no login, membros e permissões. web2py comprehende também algumas permissões específicas, ou seja, aqueles que têm um nome que correspondem aos métodos CRUD (criar, ler, atualizar, excluir) e pode aplicá-las automaticamente, sem a necessidade de utilizar decoradores.

Neste capítulo, vamos discutir as diferentes partes do RBAC um por um.

9.1 Autenticação

Para poder utilizar RBAC, os usuários precisam ser identificados. Isso significa que eles precisam se registrar (ou ser registrado) e log in

Auth fornece métodos login múltiplas. O padrão consiste em identificar os usuários com base na tabela `auth_user` local. Alternativamente, ele pode entrar na parte de autenticação de usuários contra os sistemas de terceiros e único sinal de fornecedores tais como Google, PAM, LDAP, Facebook, LinkedIn, OpenID, OAuth, etc.

Para começar a usar o `Auth`, você precisará de pelo menos esse código em

um arquivo de modelo, que também é fornecido com a "welcome" aplicação web2py e pressupõe um objeto de conexão db:

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

Definir `username=True` se você quiser auth nome de usuário para usuário de login, em vez de e-mail.

Para expor **Auth**, também terá a seguinte função de um controlador (por exemplo, "default.py"):

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

O objeto auth e user são as medidas já definidas no pedido de andaimes.

web2py também inclui uma amostra "default/user.html" vista a tornar esta função corretamente parecida com esta:

```
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 função simplesmente exibe uma `form` e, portanto, pode ser personalizado usando a sintaxe normal forma personalizada. A única ressalva é que a forma exibida depende do valor do `request.args(0)`, portanto, você pode precisar se declaração como esta:

```
1 {{if request.args(0)=='login':}...custom login form...{{pass}}}
```

O controlador acima expõe várias ações:

```
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 aos usuários registrar. É integrado com o CAPTCHA, embora este seja desativado por padrão.
- **login** permite que os usuários que estão registrados para log (se a inscrição é verificada ou não exigir a verificação, se foi aprovado ou não exigir a aprovação, e se não foi bloqueado).
- **logout** faz o que você poderia esperar, mas também, como os outros métodos, registra o evento e pode ser usado para acionar algum evento.
- **profile** permite aos usuários editar seu perfil, ou seja, o conteúdo da tabela auth_user. Observe que esta tabela não tem uma estrutura fixa e pode ser personalizado.
- **change_password** permite que os usuários alterem suas senhas de uma forma não-segura.
- **verify_email**. Se a verificação de e-mail está ligada, em seguida, os visitantes registro, em cima, receberá um email com um link para verificar suas informações e-mail. O link aponta para esta ação.
- **retrieve_username**. Por padrão, o e-mail usa **Auth** e senha para login, mas pode, opcionalmente, em vez de usar nome de utilizador e-mail. Neste último caso, se um usuário esquecer seu nome de utilizador, o método **retrieve_username** permite que o usuário digitar o endereço de email e recuperar o usuário por e-mail.
- **request_reset_password**. Permite que os usuários que esqueceram sua senha para solicitar uma nova senha. Eles receberão um email de confirmação apontando para **reset_password**.
- **impersonate** permite que um usuário para outro "impersonate". Isso é importante para depuração e para fins de suporte. `request.args[0]` é o id

do usuário para ser representado. Isso só é permitido se o usuário logado `has_permission('impersonate', db.auth_user, user_id)`.

- **groups** lista os grupos do atual usuário conectado é um membro do grupo.
- **not_authorized** exibe uma mensagem de erro quando o visitante tentou fazer algo que ele / ela não está autorizada a fazer
- **navbar** é um auxiliar que gera um bar com login / registro / etc. links.

Sair, perfil, change_password, representar, e os grupos exigem login.

Por padrão, todas elas são expostas, mas é possível restringir o acesso a apenas algumas dessas ações.

Todos os métodos acima podem ser estendidos ou substituídos por **Auth** subclasse.

Para restringir o acesso a funções apenas registrado nos visitantes, decorar a função como no exemplo a seguir

```
1 @auth.requires_login()  
2 def hello():  
3     return dict(message='hello %(first_name)!' % auth.user)
```

Qualquer função pode ser decorada, e não apenas expor as ações. Claro que isso ainda é apenas um exemplo muito simples de controle de acesso. Exemplos mais complexos serão discutidos posteriormente.

auth.user contém uma cópia dos registros db.auth_user para o atual usuário conectado ou None contrário. Há também um auth.user_id também, que é o mesmo que auth.user.id (ou seja, a identificação do usuário atual no logger) ou None.

9.1.1 Restrições sobre o registo

Se você quiser permitir que os visitantes se registrar, mas não para entrar até que o registro foi aprovado pelo administrador:

```
1 auth.settings.registration_requires_approval = True
```

Você pode aprovar um registo através da interface appadmin. Olhe para o auth_user da tabela. registos pendentes têm um campo definido para registration_key "pending". A inscrição é aprovado quando este campo está definido como branco.

Através do interface appadmin, você também pode bloquear um usuário de log dentro Localize o utilizador na auth_user tabela e defina o registration_key "blocked". "blocked" para os usuários não têm permissão para entrar Observe que isso irá impedir que um visitante de login mas não vai força de um visitante que já está conectado para sair.

Você também pode bloquear o acesso à página "register" completamente com esta afirmação:

```
1 auth.settings.actions_disabled.append('register')
```

Outros métodos de **Auth** pode ser restringido da mesma maneira.

9.1.2 Integração com o OpenID, Facebook, etc

Você pode usar o web2py Papel da Base Access Control e autenticar com outros serviços como o OpenID, Facebook, LinkedIn, Google, MySpace, Flickr, etc A maneira mais fácil é usar RPX (Janrain.com).

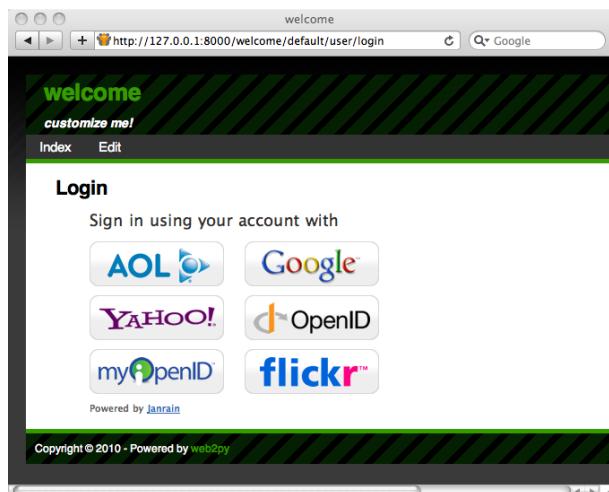
Janrain.com é um serviço que fornece autenticação middleware. Você pode se registrar com Janrain.com, registrar um domínio (o nome da sua app) e

um conjunto de URLs que você irá utilizar, e que irá fornecer-lhe com a chave API.

Agora edite o modelo do seu aplicativo web2py e em algum lugar depois de definir o objeto auth você colocar as seguintes linhas:

```
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)
```

Quando as importações a primeira linha do novo método de login, a segunda linha com deficiência registo local, a terceira linha web2py pede para usar o método login RPX. Você deve inserir o seu próprio api_key fornecido por Janrain.com, o domínio que você escolher no momento do registo e do url externa da sua página de login.



Quando um novo usuário logins pela primeira vez web2py cria um novo registro db.auth_user associado ao usuário. Vai usar o campo registration_id para armazenar um ID único para o usuário. A maioria dos métodos de

autenticação também irá fornecer um nome de usuário, e-mail, first_name e last_name, mas isso não é garantido. Quais os campos são fornecidos depende do método de login selecionado pelo usuário. Se o mesmo usuário logins duas vezes usam mecanismos de autenticação diferentes (por exemplo uma vez com OpenID e uma vez com o Facebook) JanRain pode não reconhecer o seu registration_id como o mesmo usuário e tema diferente.

Você pode personalizar o mapeamento entre os dados fornecidos pelo JanRain e os dados armazenados no db.auth_user. Aqui está um exemplo de 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"])

```

As chaves no dicionário são campos de db.auth_user e os valores são entradas de dados no perfil do objeto fornecido pela JanRain. Olhe a documentação JanRain online para obter detalhes sobre o último.

JanRain também manter estatísticas sobre o seu login dos usuários.

Este formulário de login é totalmente integrado com web2py Role Based Access Control e você ainda pode criar grupos, os usuários membros de grupos, atribuir permissões, os usuários do bloco, etc

Se você preferir não usar o JanRain e quer usar um método de login diferente (LDAP, PAM, Google, OpenID, OAuth / Facebook, LinkedIn, etc), você pode fazê-lo. A API para fazê-lo são descritos mais adiante neste capítulo.

9.1.3 CAPTCHA e reCAPTCHA

Para evitar spammers e bots registrando em seu site, você pode exigir um registo CAPTCHA. web2py suporta reCAPTCHA (71) fora da caixa. Isso ocorre porque reCAPTCHA é muito bem projetado, livre, acessível (pode ler as palavras para os visitantes), fácil de configurar e não requer a instalação de qualquer bibliotecas de terceiros.

Isto é o que você precisa fazer para usar o reCAPTCHA:

- Registre-se e obter uma reCAPTCHA (71) (PUBLIC_KEY, PRIVATE_KEY casal) para sua conta. Estas são apenas duas cordas.
- Acrescente o seguinte código ao seu modelo auth depois que o objeto está definido:

```
1 from gluon.tools import Recaptcha  
2 auth.settings.captcha = Recaptcha(request,  
3                                     'PUBLIC_KEY', 'PRIVATE_KEY')
```

reCAPTCHA pode não funcionar se você acessar o site como 'localhost' ou '127.0.0.1', porque está registrado para trabalhar com web sites publicamente visíveis apenas.

O construtor Recaptcha tem alguns argumentos opcionais:

```
1 Recaptcha(..., use_ssl=True, error_message='invalid')
```

Observe que por padrão use_ssl=False.

Se você não quiser usar o reCAPTCHA, olhar para a definição da classe no Recaptcha "gluon/tools.py", pois é fácil de usar outros sistemas CAPTCHA.

9.1.4 Personalizando Auth

A chamada para

```
1 auth.define_tables()
```

Auth define todas as tabelas que não tenham sido já definidos. Isto significa que se você quiser fazer isso, você pode definir sua própria tabela `auth_user`. Usando uma sintaxe similar à mostra abaixo, você pode personalizar qualquer outra tabela **Auth**.

Aqui é a forma correcta de definir uma tabela de usuário:

```
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()
```

Você pode adicionar qualquer campo que você deseja, mas você não pode remover os campos necessários mostrado neste exemplo.

É importante fazer campos "password", "registration_key", "reset_password_key" and "registration_id" readable=False e writable=False, uma vez que um visitante não devem ser autorizados a mexer com eles.

Se você adicionar um campo chamado "username", it will be used in place of "email" para login. Se você fizer isso, você precisará adicionar um validador, assim:

```
1 auth_table.username.requires = IS_NOT_IN_DB(db, auth_table.username)
```

9.1.5 Renomeando tabelas Auth

Os nomes reais das tabelas são armazenadas no 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'
```

Os nomes da tabela pode ser alterada através da reafectação das variáveis acima depois que o objeto auth é definido e antes dos quadros Auth estão definidas. Por exemplo:

```
1 auth = Auth(globals(),db)  
2 auth.settings.table_user_name = 'person'  
3 #...  
4 auth.define_tables()
```

As tabelas real também pode ser relacionado, independentemente dos seus nomes reais, ao

```
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
```

9.1.6 Outros métodos e formulários de login Login

Auth fornece métodos login múltiplo e ganchos para criar métodos de novo login. Cada método login suportado corresponde a um arquivo na pasta

```
1 gluon/contrib/login_methods/
```

Consulte a documentação nos arquivos próprios para cada método de login, mas aqui nós fornecemos alguns exemplos.

Primeiro de tudo precisamos fazer uma distinção entre dois tipos de métodos alternativos login:

- métodos de login que usa um formulário de login web2py (embora as credenciais são verificadas fora web2py). Um exemplo é o LDAP.
- métodos que requerem um login externo single-sign-na forma (um exemplo é o Google e Facebook).

No caso web2py nunca fica as credenciais de login, apenas um login token emitido pelo prestador do serviço. O token é armazenado no `db.auth_user.registration_token`.

Vamos considerar exemplos do primeiro caso:

Básico

Digamos que você tenha um serviço de autenticação, como por exemplo a url

```
1 https://basic.example.com
```

que aceita a autenticação de acesso básico. Isso significa que o servidor aceita pedidos HTTP com um cabeçalho do formulário:

```
1 GET /index.html HTTP/1.0
2 Host: basic.example.com
3 Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

onde a corda esta é a codificação base64 da string username: password. O serviço responde 200 OK, se o usuário está autorizado e 400, 401, 402, 403 ou 404 em contrário.

Você deseja inserir nome de usuário e senha através do formulário de login padrão Auth e verificar as credenciais contra esse serviço. Tudo que você precisa fazer é adicionar o seguinte código ao seu aplicativo

```
1 from gluon.contrib.login_methods.basic_auth import basic_auth
2 auth.settings.login_methods.append(
3     basic_auth('https://basic.example.com'))
```

Observe que auth.settings.login_methods é uma lista de métodos de autenticação que são executados seqüencialmente. Por padrão é definido para

```
1 auth.settings.login_methods = [auth]
```

Quando um método alternativo é anexado, por exemplo, basic_auth, **Auth** primeiras tentativas para entrar no visitante com base no conteúdo do auth_user, e quando isso falhar, ele tentará o próximo método na lista. Se um método for bem-sucedido no login do visitante, e se auth.settings.login_methods[0]==auth, Auth tem as seguintes ações:

- se o usuário não existe no auth_user, um novo usuário é criado eo nome de usuário / email e as senhas são armazenadas.
- se o usuário não existir no auth_user, mas a nova senha aceitos não coincidir com a senha antiga armazenados, a senha antiga é substituída por um novo aviso (que as senhas são sempre armazenados hash salvo indicação em contrário).

Se você não deseja guardar a nova senha no auth_user, então é suficiente para alterar a ordem dos métodos de login, auth ou remover da lista. Por exemplo:

```
1 from gluon.contrib.login_methods.basic_auth import basic_auth
2 auth.settings.login_methods = \
3     [basic_auth('https://basic.example.com')]
```

O mesmo se aplica para qualquer outro método login aqui descritos.

E SMTP do Gmail

Você pode verificar as credenciais de login usando um servidor SMTP remoto, por exemplo, Gmail, ou seja, você faz logon do usuário no caso do e-mail e senha que eles fornecem são credenciais válidas para acessar o servidor SMTP do Gmail (`smtp.gmail.com:587`). Tudo o que é necessário é o seguinte 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"))
```

O primeiro argumento do `email_auth` é o endereço: porta do servidor SMTP. O segundo argumento é o domínio de e-mail.

Isso funciona com qualquer servidor SMTP que requer TLS authentication.

PAM

Autenticação usando Pluggable Authentication Modules (PAM), funciona como nos casos anteriores. Permite web2py para autenticar os usuários usando as contas do sistema operacional:

```
1 from gluon.contrib.login_methods.pam_auth import pam_auth
2 auth.settings.login_methods.append(pam_auth())
```

LDAP

Autenticação LDAP usando funciona muito bem como nos casos anteriores.

Para utilizar o login LDAP com 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 o login LDAP com o Lotus Notes e Domino:

```
1 auth.settings.login_methods.append(ldap_auth(mode='domino',
2     server='my.domino.server'))
```

Para utilizar o login LDAP com OpenLDAP (com UID):

```
1 auth.settings.login_methods.append(ldap_auth(server='my.ldap.server',
2     base_dn='ou=Users,dc=domain,dc=com'))
```

Para utilizar o login LDAP com OpenLDAP (com CN):

```
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

Google usando autenticação quando em execução no Google App Engine requer pular o formulário de login web2py, sendo redirecionado para a página de login do Google, e voltar ao sucesso. Porque o comportamento é diferente do que nos exemplos anteriores, a API é um pouco diferente.

```
1 from gluon.contrib.login_methods.gae_google_login import GaeGoogleAccount
2 auth.settings.login_form = GaeGoogleAccount()
```

OpenID

Temos discutido anteriormente integração com JanRain (que tem apoio OpenID) e que é a maneira mais fácil de usar OpenID. No entanto, algumas vezes você não quer contar com um serviço de terceiros e você quiser acessar o provedor OpenID diretamente ao consumidor (o app).

Aqui está um exemplo:

```

1 from gluon.contrib.login_methods.openid_auth import OpenIDAuth
2 auth.settings.login_form = OpenIDAuth(auth)

```

OpenIDAuth requer o "python-open" instalados separadamente.

Sob o capô, este método define o login tabela a seguir

```

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 armazena os nomes openid para cada usuário. Se você deseja exibir o OpenIDs para o atual usuário conectado:

```
1 {{=auth.settings.login_form.list_user_openids()}}
```

OAuth2.0 e Facebook

Temos discutido anteriormente integração com JanRain (que tem apoio no Facebook), mas às vezes você não quer contar com um serviço de terceiros e você quiser acessar um provedor OAuth2.0 diretamente por exemplo Facebook. Veja como:

```

1 from gluon.contrib.login_methods.oauth20_account import OAuthAccount
2 auth.settings.login_form=OAuthAccount(globals(),YOUR_CLIENT_ID,YOUR_CLIENT_SECRET)

```

As coisas ficam um pouco mais complexo, se você quer usar Facebook para não oauth2.0 login login em seu próprio aplicativo, mas em um específico Facebook para acessar sua API. Aqui vamos dar um exemplo para acessar Facebook Graph API.

Primeiro de tudo você deve instalar o módulo "pyfacebook".

Segundo, você precisa o seguinte código em seu 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

Temos discutido anteriormente integração com JanRain (que tem o apoio LinkedIn) e que é a maneira mais fácil de usar OpenID. No entanto, algumas vezes você não quer contar com um serviço de terceiros e você deseja acessar LinkedIn diretamente para obter mais informações agradecer JanRain oferece.

Aqui está um exemplo:

```
1 from gluon.contrib.login_methods.linkedin_account import LinkedInAccount
2 auth.settings.login_form=LinkedInAccount(request,KEY,SECRET,RETURN_URL)
```

LinkedInAccount "python-linkedin" requer o módulo instalado separadamente.

Login formulários múltiplos

Alguns métodos de modificar o login login_form, outros não. Quando eles fazem isso, eles podem não ser capazes de coexistir. No entanto, alguns coexistem alguns formulários de login, fornecendo múltiplos na mesma página. web2py fornece uma maneira de fazê-lo. Aqui está um exemplo de mistura login normal (auth) e RPX login (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'])

```

Se os sinais são definidos e um parâmetro no pedido encontrados quaisquer sinais, ele vai retornar a ligação em vez de other_form.login_form. other_form pode lidar com algumas situações específicas, por exemplo, várias etapas do OpenID login dentro other_form.login_form. Caso contrário, ele irá processar o formulário de login normal, juntamente com o other_form.

9.2 Auth & Mail

Por padrão, a verificação de e-mail está desativado. Para habilitar-mail, acrescente as seguintes linhas no modelo auth onde está definida:

```

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://' + \
    request.env.http_host + \
11     URL(r=request,c='default',f='user',args=['verify_email']) + \
12     '/%(key)s to verify your email'
13

```

```
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'
```

É necessário substituir o mail.settings com os parâmetros adequados para o seu servidor SMTP. mail.settings.login=False Definir se o servidor SMTP não requer autenticação.

Você também precisa substituir o

```
'Click on the link ...'
```

string em auth.messages.verify_email com o URL correcto completa do verify_email ação. Isso é necessário porque web2py pode ser instalado atrás de um proxy, e ele não pode determinar o seu próprio público URLs com certeza absoluta.

Depois mail é definido, pode também ser usado para enviar e-mail via explicitamente

```
1 mail.send(to=['somebody@example.com'],
2             subject='hello',
3             message='hi there')
```

Mail retorna True se conseguir enviar o e-mail, False contrário.

9.2.1 Email Depuração

Para fins de depuração pode definir

```
1 mail.settings.server = 'logging'
```

e e-mails não serão enviados, mas em vez conectado ao console.

9.2.2 Emails formulário Google App Engine

Para enviar e-mails de conta do Google App Engine:

```
1 mail.settings.server = 'gae'
```

No momento da escrita web2py não suporta anexos e e-mails criptografados no Google App Engine.

9.2.3 Mais exemplos de e-mail

e-mail de texto simples

```
1 mail.send('you@example.com',
2   'Message subject',
3   'Plain text body of the message')
```

E-mails HTML

```
1 mail.send('you@example.com',
2   'Message subject',
3   '<html>html body</html>')
```

Se o corpo do email começa com `<html>` e termina com o `</html>` vai ser enviada como um e-mail HTML.

Combinação de texto e emails HTML

A mensagem de e-mail pode ser um texto (tuple, html):

```
1 mail.send('you@example.com',
2   'Message subject',
3   ('Plain text body', '<html>html body</html>'))
```

Emails CC e BCC

```
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']
```

Anexos

```
1 mail.send('you@example.com',
2   'Message subject',
3   '<html></html>',
4   attachments = Mail.Attachment('/path/to/photo.jpg' content_id='photo'))
```

vários anexos

```
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')])
```

9.2.4 x509 e PGP criptografia

É possível enviar x509 SMIME () criptografados e-mails usando as seguintes configurações:

```
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'
```

É possível enviar PGP e-mails criptografados usando as seguintes configurações:

```
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
```

A segunda exige que o pacote python-pyme.

9.3 Autorização

Uma vez que um novo usuário é registrado, um novo grupo é criado para conter o usuário. O papel do novo usuário que se convencionou "user_[id]" [id] é a identificação do recém-criado id. A criação do grupo pode ser desativado com

```
1 auth.settings.create_user_groups = False
```

embora não sugerem isso.

Os usuários têm participação em grupos. Cada grupo é identificado por um nome / função. Os grupos têm permissões. Os usuários têm permissões por causa dos grupos a que pertencem.

Você pode criar grupos, dão adesão e permissões via **appadmin** ou programaticamente usando os seguintes métodos:

```
1 auth.add_group('role', 'description')
```

retorna o id do grupo recém-criado.

```
1 auth.del_group(group_id)
```

exclui o grupo com **group_id**.

```
1 auth.del_group(auth.id_group('user_7'))
```

exclui o grupo com "user_7" papel, ou seja, o grupo exclusivamente associado ao número de usuários 7.

```
1 auth.user_group(user_id)
```

retorna o id do grupo exclusivamente associado ao usuário identificado pelo user_id.

```
1 auth.add_membership(group_id, user_id)
```

dá a adesão do user_id group_id grupo. Se o user_id não é especificado, então web2py assume o atual usuário conectado.

```
1 auth.del_membership(group_id, user_id)
```

Revoga a adesão do user_id group_id grupo. Se o user_id não é especificado, então web2py assume o atual usuário conectado.

```
1 auth.has_membership(group_id, user_id, role)
```

verifica se tem user_id group_id membros do grupo ou o grupo com a função especificada. Apenas group_id ou role deve ser passado para a função, mas não ambos. Se o user_id não é especificado, então web2py assume o atual usuário conectado.

```
1 auth.add_permission(group_id, 'name', 'object', record_id)
```

"name" (user defined) on the object "object" dá permissão (também definido pelo usuário) para os membros do grupo group_id. Se o "object" é um table-name então a permissão pode se referir a toda a tabela (record_id==0) ou para um registro específico (record_id>0). Ao dar permissões em tabelas, é comum usar um nome de permissão no set ('criar', 'ler', 'update', 'delete', 'select'), uma vez que estas permissões são conhecidas e podem ser aplicadas por CRUD.

```
1 auth.del_permission(group_id, 'name', 'object', record_id)
```

revoga a autorização.

```
1 auth.has_permission('name', 'object', record_id, user_id)
```

verifica se o usuário identificado por user_id tem participação em um grupo com a autorização solicitada.

```

1 rows = db(accessible_query('read', db.sometable, user_id))\
2     .select(db.mytable.ALL)

```

retorna todas as linhas da tabela que "sometable" user_id "read" usuário tenha permissão em. Se o user_id não é especificado, assume que o actual web2py logado usuário. O accessible_query(...) pode ser combinado com outras consultas para tornar mais complexos. accessible_query(...) é o único **Auth** método para exigir um JOIN, então ele não funciona no Google App Engine.

Assumindo as seguintes definições:

```

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')

```

Aqui está um exemplo:

```

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

```

9.3.1 Decoradores

A forma mais comum de verificação de permissão não é por chamadas explícitas aos métodos acima, mas por funções de decoração para que permissões são verificadas em relação ao registrado em visitante. Aqui estão alguns exemplos:

```

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)
```

Note que o acesso a todas as funções para além do primeiro é restrito com base nas permissões que o visitante pode ou não ter.

Se o visitante não está logado, então a permissão não pode ser verificada, o visitante é redirecionado para a página de login e depois voltar para a página que requer permissões.

Se o visitante não tem permissão para acessar uma determinada função, o visitante é redirecionar para a URL definida por

```
1 auth.settings.on_failed_authorization = \
2     URL('user',args='on_failed_authorization')
```

Você pode mudar esta variável e redirecionar o usuário em outro lugar.

9.3.2 Combinando Requisitos

Ocasionalmente, é necessário combinar requisitos. Isso pode ser feito através de um decorador genérico `requires`, que tem um único argumento, uma condição verdadeira ou falsa. Por exemplo, para dar acesso aos agentes, mas apenas na terça-feira:

```

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!'

```

ou equivalentemente:

```

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!'

```

9.3.3 Autorização e de CRUD

Usando decoradores e / ou verificações explícitas fornece uma maneira de implementar controle de acesso.

Outra forma de implementar o controle de acesso é sempre usar CRUD (em oposição a SQLFORM) para acessar o banco de dados e pedir CRUD para impor o controle de acesso nas tabelas do banco de dados e registros. Isto é feito ligando Auth e CRUD com a seguinte declaração:

```
1 crud.settings.auth = auth
```

Isso impedirá que o visitante de acessar qualquer uma das funções CRUD a menos que o visitante está conectado e tem acesso explícito. Por exemplo, para permitir que um visitante de postar comentários, mas apenas atualizar seus próprios comentários (assumindo crud, auth e db.comment são definidos):

```
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)
```

Você também pode selecionar registros específicos (aqueles que têm acesso de leitura 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)
```

Os nomes permissions executada por:

```
1 crud.settings.auth = auth
```

são "read", "create", "update", "delete", "select", "impersonate".

9.3.4 Autorização e Downloads

O uso de decoradores e da utilização de crud.settings.auth não cumprir autorização por arquivos baixados por download a função usual

```
1 def download(): return response.download(request, db)
```

Se alguém quiser fazer isso, deve declarar expressamente que os campos "upload" conter ficheiros que necessitam de controle de acesso a download. Por exemplo:

```
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)
```

O atributo do campo de upload authorization pode ser None (padrão) ou uma função que decide se o usuário está logado e tem permissão para 'ler' o registro atual. Neste exemplo, não há nenhuma restrição à transferência de imagens ligadas pelo domínio "small_image" field, but we require access control on images linked by the "large_image".

9.3.5 Autenticação e Controle de Acesso Básico

Ocasionalmente, pode ser necessário expor as ações que têm decoradores que requerem controle de acesso aos serviços, ou seja, para chamá-los de um programa ou script e ainda ser capaz de usar a autenticação para verificar se há autorização.

Auth permite login através de autenticação básica:

```
1 auth.settings.allow_basic_login = False
```

Com este conjunto, uma ação como

```
1 @auth.requires_login()  
2 def give_me_time():  
3     import time  
4     return time.ctime()
```

poderão ser chamados, por exemplo, a partir de um comando:

```
1 wget --user=[username] --password=[password]  
2 http://.../[app]/[controller]/give_me_time
```

login Basic é muitas vezes a única opção de serviços (descrita no capítulo seguinte), mas ele está desativado por padrão.

9.3.6 Configurações e mensagens

Aqui está uma lista de todos os parâmetros que podem ser personalizados para **Auth**

A seguir deve apontar para um objeto para permitir gluon.tools.Mail auth para enviar e-mails:

```
1 auth.settings.mailer = None
```

A seguir deve ser o nome do controlador que definiu a ação user:

```
1 auth.settings.controller = 'default'
```

O seguinte é uma configuração muito importante:

```
1 auth.settings.hmac_key = None
```

Deve ser definido para algo como o campo da tabela "sha512:a-pass-phrase" and it will be passed to the CRYPT validator for the "password" auth_user. Será o algoritmo e uma passagem frase usada para criptografar as senhas.

Para deficientes uma ação acrescentar seu nome a esta lista:

```
1 auth.settings.actions_disabled = []
```

Por exemplo:

```
1 auth.settings.actions_disabled.append('register')
```

registro será desabilitado.

Se você quiser receber um e-mail para verificar o registo este conjunto de True:

```
1 auth.settings.registration_requires_verification = False
```

Se novos inscritos devem permitir que a sessão até que seja aprovado este conjunto de True:

```
1 auth.settings.registration_requires_approval = False
```

Aprovação consiste na implantação `registration_key==`" via appadmin ou programaticamente.

Se você não quer um novo grupo para cada novo usuário definir o que se segue para False:

```
1 auth.settings.create_user_groups = True
```

As configurações a seguir determinam os métodos alternativos login e formulários de login, como foi discutido anteriormente:

```
1 auth.settings.login_methods = [auth]
2 auth.settings.login_form = auth
```

Você deseja permitir login de base?

```
1 auth.settings.allows_basic_login = False
```

O seguinte é o URL da ação login:

```
1 auth.settings.login_url = URL('user', args='login')
```

Se o usuário tentar acessar a página de cadastro, mas já está conectado, ele será redirecionado para esta URL:

```
1 auth.settings.logged_url = URL('user', args='profile')
```

Este deve apontar para o URL da ação download, caso o perfil contém imagens:

```
1 auth.settings.download_url = URL('download')
```

Estes devem apontar para o URL que você deseja redirecionar os usuários para depois das várias ações possíveis, auth (no caso, não há referrer):

```
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 é a URL que você quer ser redirecionar a autorização a falha:

```
1 auth.settings.on_failed_authorization =
2     URL('user',args='on_failed_authorization')
```

Estas são as listas de chamadas de retorno que deverá ser executada após a validação de formulário para cada uma das ações correspondentes, antes de qualquer banco de dados 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 chamada deve ser uma função que leva o objeto `form` e pode modificar os atributos do objeto de formulário antes de IO do banco de dados é realizada.

Estas são as listas de chamadas de retorno que deve ser executado após o banco de dados de E / S é executada e antes do redirecionamento:

```

1 auth.settings.login_onaccept = []
2 auth.settings.register_onaccept = []
3 auth.settings.profile_onaccept = []
4 auth.settings.verify_email_onaccept = []

```

Aqui está um exemplo:

```

1 auth.settings.register_onaccept.append(lambda form:\n2     mail.send(to='you@example.com',subject='new user',\n            message="new user email is %s'%form.vars.email))
3

```

Você pode habilitar capcha para qualquer uma das ações 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

```

Se os pontos .captcha configurações para um gluon.tools.Recaptcha, todas as formas para que a opção correspondente (como o .login_captcha) é configurado para None terá um capitão, enquanto aquelas para as quais a opção correspondente está definido para False não. Se, pelo contrário, .captcha está definido para None, apenas a forma aqueles que têm uma opção correspondente definido para um objeto gluon.tools.Recaptcha terá catcha e os outros não.

Este é o login tempo de expiração da sessão:

```

1 auth.settings.expiration = 3600 # seconds

```

Você pode mudar o nome do campo de senha (em Firebrid para "password" exemplo é a palavra-chave e não pode ser usado para nomear um campo):

```

1 auth.settings.password_field = 'password'

```

Normalmente, o formulário de login tenta validar um e-mail. Isso pode ser desativado por alterar esta configuração:

```
1 auth.settings.login_email_validate = True
```

Você quer mostrar para o registro de id na página de edição de perfil?

```
1 auth.settings.showid = False
```

Para formulários personalizados que você pode querer desativar notificação de erro automático de formulários:

```
1 auth.settings.hideerror = False
```

Também para formulários personalizados que você pode mudar o estilo:

```
1 auth.settings.formstyle = 'table3cols'
```

(Pode ser "table2cols", "divs" and "ul")

Por padrão, o formulário de login dá a opção de estender o login via opção "remember me". O tempo de expiração pode ser alterado ou desativado a opção via essas configurações:

```
1 auth.settings.long_expiration = 3600*30 # one month
2 auth.settings.remember_me_form = True
```

Você também pode personalizar as seguintes mensagens cujo uso e contexto deveria ser óbvio: a adesão

```
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'
61 auth.messages.label_registration_id = 'Registration identifier'
62 auth.messages.label_role = 'Role'
63 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 logs permitir a utilização de "%(user_id)s" and "%(group_id)s".
add|del|has logs permissão permitir a utilização de "%(user_id)s", "%(name)s",
"%(table_name)s", and "%(record_id)s".

9.4 Central Authentication Service

web2py fornece suporte para autenticação e autorização através de aparelhos. Aqui nós discutimos o aparelho para **cas** Central Authentication Service (CAS). Observe que no momento da escrita CAS é diferente e não funciona com **Auth**. Isso vai mudar no futuro.

CAS é um protocolo aberto para autenticação distribuída e funciona da seguinte maneira: Quando um visitante chega ao nosso site, nosso pedido de verificação da sessão se o usuário já está autenticado (por exemplo através de um objeto `session.token`). Se o usuário não estiver autenticado, o controlador redireciona o visitante do aparelho CAS, onde o usuário pode logar, registrar e gerenciar suas credenciais (nome, e-mail e senha). Se o usuário se registra, ele recebe um e-mail e de registro não está completo até que ele responde ao e-mail. Uma vez que o usuário tenha registrado e conectado, o aparelho CAS redireciona o usuário para a nossa aplicação, juntamente com uma chave. Nossa aplicação usa a chave para obter as credenciais do usuário através de uma solicitação HTTP em segundo plano para o servidor CAS.

Usando este mecanismo, vários aplicativos podem usar o um único sign-on através de um servidor único CAS. O servidor que fornece autenticação é

chamado de um prestador de serviços. Aplicações buscando autenticar os visitantes são chamados de consumidores de serviços.

CAS é semelhante ao OpenID, com uma diferença principal. No caso do OpenID, o visitante escolhe o fornecedor de serviços. No caso da CAS, o nosso aplicativo faz essa escolha, tornando mais seguro CAS.

Você pode executar apenas o consumidor, apenas o provedor, ou ambos (em aplicações únicas ou separadas).

Para executar o CAS, como consumidor você deve baixar o arquivo:

```
1 https://www.web2py.com/cas/static/cas.py
```

e armazena-lo como um arquivo chamado modelo "cas.py". Then you must edit the controllers that need authentication (for example "default.py") e, na parte superior, adicione o código a seguir:

```
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()
```

Você deve editar os atributos do objeto CAS acima. Por padrão, eles apontam para o prestador de CAS que funciona em "https://mdp.cti.depaul.edu". Nós fornecemos este serviço, principalmente para fins de teste. O `CAS.my_url` tem que ser a URL completa para a ação definida `login` em seu aplicativo e mostrado no código. O provedor CAS precisa redirecionar o navegador para esta ação.

Nosso fornecedor CAS retorna um token contendo uma tupla (id, email, nome), onde id é o único registro id do visitante (designados pelo banco de dados do provedor), e-mail é o endereço de e-mail do visitante (como declarado pelo visitante ao fornecedor e verificada pelo fornecedor), eo nome é o nome do visitante (que é escolhido pelo visitante, e não há garantia esse é um nome real).

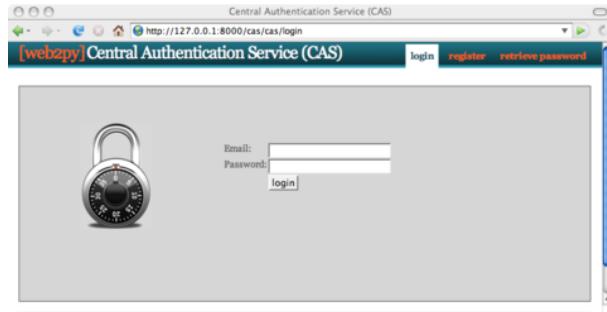
Se você visitar o local url:

```
1 /myapp/default/login
```

você for redirecionado para a página de login CAS:

```
1 https://mdp.cti.depaul.edu/cas/cas/login
```

que se parece com isto:



Você também pode usar serviços de terceiros CAS, mas pode ser necessário editar a linha 10 acima, desde fornecedores de CAS diferentes podem retornar fichas contendo valores diferentes. Verifique a documentação do serviço CAS você precisa acessar para obter mais detalhes. A maioria dos serviços só voltar (id, username).

Após um login bem-sucedido, você será redirecionado para o login ação local. O ponto de vista da ação login local é executado somente após um login bem sucedido CAS.

Você pode baixar o aparelho fornecedor de ref. (33) CAS e executá-lo sozinho. Se você optar por fazê-lo, você deve também editar as primeiras linhas do modelo "email.py" no aparelho, para que ele aponta para o servidor SMPT.

Você também pode mesclar os arquivos do prestador aparelho fornecedor CAS com aqueles de sua aplicação (modelos com modelos, etc), desde que não haja conflito nome.

10

Serviços

O W3C define um serviço web como "a software system designed to support interoperable machine-to-machine interaction over a network". Esta é uma definição ampla, e abrange um grande número de protocolos não foi concebido para comunicação máquina-a-humano, mas para a comunicação máquina-máquina, tais como XML, JSON, RSS, etc web2py prevê, fora da caixa, suporte para vários protocolos, incluindo XML, JSON, RSS, CSV, XML-RPC, JSONRPC, AMFRPC e SOAP. web2py também pode ser estendido para suportar protocolos adicionais.

Cada um destes protocolos é apoiado de várias maneiras, e nós fazemos uma distinção entre:

- Escutar a saída de uma função em um formato determinado (por exemplo XML, JSON, RSS, CSV)
- Chamadas de procedimento remoto (por exemplo XMLRPC, JSONRPC, AMFRPC)

10.1 Escutar um dicionário

10.1.1 HTML, XML e JSON

Considere as seguintes ações:

```
1 def count():
2     session.counter = (session.counter or 0) + 1
3     return dict(counter=session.counter, now=request.now)
```

Esta ação retorna um contador que é aumentada por um quando um visitante recarrega a página e, a hora do pedido da página atual.

Normalmente, esta página será solicitado através de:

```
1 http://127.0.0.1:8000/app/default/count
```

e processado em HTML. Sem escrever uma linha de código, podemos perguntar web2py para tornar esta página usando um protocolo diferente, adicionando uma extensão da URL:

```
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
```

O dicionário retornado pela ação será processado em HTML, XML e JSON, respectivamente.

Aqui é a saída XML:

```
1 <document>
2     <counter>3</counter>
3     <now>2009-08-01 13:00:00</now>
4 </document>
```

Aqui é a saída JSON:

```
1 { 'counter':3, 'now':'2009-08-01 13:00:00' }
```

Repare que data, hora, data e hora e objetos são processados como strings no formato ISO. Isto não é parte do padrão JSON, mas uma convenção web2py.

10.1.2 Generic Views

Quando, por exemplo, a ".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" são fornecidos com a aplicação de andaimes atual.

Outras extensões podem ser facilmente definidos pelo usuário.

Nada precisa ser feito para permitir isso em um app web2py. Para usá-lo em um aplicativo web2py mais velhos, talvez seja necessário copiar o "genérico.*" "arquivos de um aplicativo depois de andaimes (após a versão 1.60).

Aqui está o 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>
```

Aqui está o 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 }}
```

E aqui está o 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 dicionário pode ser processado em HTML, XML e JSON, desde que contenha apenas python tipos primitivos (int, float, string, lista, tupla, dicionário). `response._vars` contém o dicionário retornado pela ação.

Se o dicionário contém outros objetos definidos pelo usuário ou web2py específicos, devem ser prestados por uma exibição personalizada.

10.1.3 *Rendering linhas*

Se você precisa render uma série de linhas retornado por uma seleção em XML ou JSON ou outro formato, em primeiro lugar transformar a Linhas objeto em uma lista de dicionários através do método `as_list()`.

Considere, por exemplo, os seguintes modos:

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

A ação a seguir podem ser processados em HTML, mas não em XML ou JSON:

```
1 def everybody():
2     people = db().select(db.person.ALL)
3     return dict(people=people)
```

Enquanto a ação seguinte pode prestados em XML e JSON:

```
1 def everybody():
2     people = db().select(db.person.ALL).as_list()
3     return dict(people=people)
```

10.1.4 Formatos personalizados

Se, por exemplo, você quer tornar uma ação como um picles Python:

```
1 http://127.0.0.1:8000/app/default/count.pickle
```

você só precisa criar um arquivo "default/count.pickle" nova exibição que contém:

```
1 {{ 
2     import cPickle
3     response.headers['Content-Type'] = 'application/python.pickle'
4     response.write(cPickle.dumps(response._vars), escape=False)
5 }}
```

Se você quiser ser capaz de render como um arquivo escolhido qualquer ação, você só precisa salvar o arquivo acima com o nome do "generic.pickle".

Nem todos os objetos são pickle, e nem todos os objetos em conserva pode ser un-conservado. É seguro para furar a primitiva arquivos Python e suas combinações. Os objetos que não contêm referências ao arquivo de córregos ou conexões de banco de dados são geralmente pickle, mas eles só podem ser un-conservado em um ambiente onde as classes de todos os objetos em conserva já estão definidos.

10.1.5 RSS

web2py inclui uma exibição "generic.rss" que podem tornar o dicionário re-tornado pela ação como um feed RSS.

Porque os RSS feeds tem uma estrutura fixa (título, link, descrição, peças, etc), então para que isso funcione, o dicionário retornado pela ação deve ter a estrutura adequada:

```
1 {'title'      : '',
2  'link'       : '',
3  'description': '',
4  'created_on' : '',
5  'entries'    : []}
```

e cada entrada nas entradas devem ter a mesma estrutura similar:

```
1 {'title'      : '',
2  'link'       : '',
3  'description': '',
4  'created_on' : ''}
```

Por exemplo, a seguinte ação pode ser processado como um 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                 ])
```

simplesmente visitar a URL:

```
1 http://127.0.0.1:8000/app/default/feed.rss
```

Em alternativa, assumindo o seguinte modelo:

```
1 db.define_table('rss_entry',
2                 Field('title'),
3                 Field('link'),
4                 Field('created_on','datetime'),
5                 Field('description'))
```

a seguinte ação também pode ser processado como um 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())

```

O método `as_list()` de um objeto `Rows` converte as linhas em uma lista de dicionários.

Se, além de itens do dicionário são encontrados com nomes-chave não listadas aqui, eles são ignorados.

Aqui está o ponto de vista "generic.rss" fornecido por 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 mais um exemplo de um pedido de RSS, que consideramos um agregador de RSS que coleta dados do "slashdot" feed e retorna um feed web2py novo.

```

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])

```

Ele pode ser acessado no seguinte endereço:

```
1 http://127.0.0.1:8000/app/default/aggregator.rss
```

10.1.6 CSV

O Comma Separated Values (CSV) é um protocolo para representar dados tabulares.

Considere o seguinte modelo:

```
1 db.define_model('animal',
2     Field('species'),
3     Field('genus'),
4     Field('family'))
```

e as seguintes medidas:

```
1 def animals():
2     animals = db().select(db.animal.ALL)
3     return dict(animals=animals)
```

web2py não fornece um "generic.csv"; you must define a custom view "default/animals.csv" que serializa os animais em CSV. Aqui está uma possível implementação:

```
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 }}
```

Observe que para um CSV também pode definir um arquivo "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".

10.2 Chamadas de procedimento remoto

web2py fornece um mecanismo para transformar qualquer função em um serviço web. O mecanismo descrito aqui é diferente do mecanismo descrito antes, porque:

- A função pode ter argumentos
- A função pode ser definida em um modelo ou um módulo em vez do controlador
- Você pode querer especificar o tipo de método RPC deve ser apoiada
- Ela reforça uma URL mais rigorosa convenção de nomeação
- É mais esperto que os métodos anteriores, porque ele trabalha para um conjunto fixo de protocolos. Pela mesma razão, não é tão facilmente extensível.

Para usar este recurso:

Primeiro, você deve importar e instanciar um objeto de serviço.

```
1 from gluon.tools import Service
2 service = Service(globals())
```

Isso já é feito no arquivo de modelo na aplicação "db.py" andaimes.

Em segundo lugar, você deve expor o manipulador de serviço no controlador:

```
1 def call():
2     session.forget()
3     return service()
```

Isso já feita no controlador "default.py" da aplicação de andaimes. Remover session.forget() é que você planeja usar cookies de sessão com os serviços.

Em terceiro lugar, você deve decorar as funções que você deseja expor como um serviço. Aqui está uma lista de suporte decoradores atualmente:

```
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 exemplo, considere o decorado seguinte função:

```
1 @service.run
2 def concat(a,b):
3     return a+b
```

Esta função pode ser definida em um modelo ou em um controlador. Esta função pode agora ser chamado remotamente de duas maneiras:

```
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)
```

Em ambos os casos, o pedido retorna http:

```
1 helloworld
```

Se o decorador `@service.xml` é usada, a função pode ser chamada via

```
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)
```

ea saída é retornado como XML:

```
1 <document>
2   <result>helloworld</result>
3 </document>
```

Ele pode serializar a saída da função, mesmo se este é um Linhas DAL objeto. Neste caso, na verdade, ele chamará automaticamente `as_list()`.

Se o decorador `@service.json` é usada, a função pode ser chamada via

```
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)
```

ea saída retornado como JSON.

Se o decorador `@service.csv` é usado, o manipulador de serviço o exigir, como valor de retorno, um objeto iteráveis iteráveis de objetos, como uma lista de listas. Aqui está um exemplo:

```
1 @service.csv
2 def table1(a,b):
3     return [[a,b],[1,2]]
```

Este serviço pode ser chamado por visitar uma das seguintes URLs:

```
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
```

e ele retorna:

```
1 hello,world
2 1,2
```

O decorador `@service.rss` espera um valor de retorno no mesmo formato como o ponto de vista "generic.rss" discutido na seção anterior. decoradores múltiplas são permitidas para cada função.

Até agora, tudo discutido nesta seção é simplesmente uma alternativa ao método descrito na seção anterior. O poder real do objeto de serviço vem com XMLRPC, JSONRPC e AMFRPC, como discutido abaixo.

10.2.1 XMLRPC

Considere o seguinte código, por exemplo, o 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
```

Agora, em um shell python que você pode fazer

```
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
```

O módulo de Python xmlrpclib fornece um cliente para o protocolo XML-RPC. web2py atos como o servidor.

O cliente se conecta ao servidor via ServerProxy e pode remotamente ligar decorado funções no servidor. Os dados (a, b) é passada para a função (s), não via variáveis GET / POST, mas devidamente codificado no corpo da solicitação através do protocolo XMLRPC e, portanto, carrega consigo informações sobre o tipo (int ou string ou outro). O mesmo é verdadeiro para o valor de retorno (s). Além disso, qualquer exceção que acontece no servidor propaga de volta ao cliente.

Existem bibliotecas XMLRPC para muitas linguagens de programação (incluindo C, C++, Java, C#, Ruby e Perl), e eles podem interoperar uns com os outros. Este é um dos melhores métodos para criar aplicações que conversem entre si, independente de linguagem de programação.

O cliente XMLRPC também pode ser implementada dentro de uma ação web2py de modo que uma ação pode falar com outra aplicação web2py

(mesmo dentro da mesma instalação), utilizando XMLRPC. Cuidado com os impasses da sessão, neste caso. Se uma ação chamadas via XMLRPC uma função no mesmo aplicativo, o chamador deve liberar o bloqueio sessão antes da chamada:

```

1 session.forget()
2 session._unlock(response)

```

10.2.2 JSONRPC e Pijamas

JSONRPC é muito semelhante ao XMLRPC, mas usa o protocolo JSON base para codificar os dados em vez de XML. Como exemplo de aplicação aqui, discutir sua utilização com Pijama. Pijama é um porto Python do Google Web Toolkit (originalmente escrito em Java). Pijama permite escrever uma aplicação cliente em Python. Pijama traduz o código em JavaScript. web2py serve o javascript e se comunica com ele através de requisições Ajax proveniente do cliente e desencadeada por ações do usuário.

Aqui nós descrevemos como fazer o trabalho com web2py Pijama. Não requer nenhuma biblioteca adicional além web2py e Pijama.

Nós vamos construir um aplicativo simples "todo" com um cliente Pijama (todos JavaScript) que fala para o servidor exclusivamente via JSONRPC.

Aqui está como fazê-lo:

Primeiro, crie um novo aplicativo chamado "todo".

Em segundo lugar, "models/db.py", digite o seguinte 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())

```

Em terceiro lugar, "controllers/default.py", digite o seguinte 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()
```

A finalidade de cada função deve ser óbvia.

Em quarto lugar, "views/default/todoApp.html", digite o seguinte 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 ponto de vista apenas executa o código Pijama em "static/output/todoapp". Código que ainda não criado.

Em quinto lugar, em aviso ("static/TodoApp.py" é TodoApp, não todoApp!), Digite o código do cliente que se segue:

```

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()
```

Sexta, executar Pijama antes de servir o pedido:

```
1 cd /path/to/todo/static/
2 python /python/pyjamas-0.5p1/bin/pyjsbuild TodoApp.py
```

Isto irá traduzir o código Python em JavaScript para que ele possa ser executado no navegador.

Para acessar esse aplicativo, visite o URL

```
1 http://127.0.0.1:8000/todo/default/todoApp
```

Esta subseção foi criada por Chris Prinos com a ajuda de forma Luke Kenneth Casson Leighton (criadores de Pijama) e atualizado por Alexei Vinidiktov. Foi testado por Pijama 0.5p1. O exemplo foi inspirado por esta página em Django ref. (74).

10.2.3 AMFRPC

AMFRPC é o protocolo Remote Procedure Call Flash usado por clientes para se comunicar com um servidor. web2py suporta AMFRPC mas requer que você execute web2py da fonte e que pré-instalar a biblioteca PyAMF. Isto pode ser instalado a partir do Linux ou o Windows shell digitando

```
1 easy_install pyamf
```

(Consulte a documentação PyAMF para mais detalhes).

Nesta subseção vamos supor que você já está familiarizado com a programação ActionScript.

Vamos criar um simples serviço que utiliza dois valores numéricos, adiciona-los juntos, e retorna a soma. Vamos chamar nosso aplicativo web2py "pyamf_test", e vamos chamar o addNumbers serviço.

Primeiro, usando o Adobe Flash (qualquer versão a partir de MX 2004), criar o aplicativo cliente Flash, começando com um novo arquivo FLA Flash. No primeiro frame do arquivo, adicione estas linhas:

```
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();
```

Esse código permite que o cliente Flash para ligar a um serviço que corresponda a uma função chamada pasta "addNumbers" in the file "/pyamf_test/default/ga 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" ao lado do arquivo recém-criado.

Observe os argumentos do construtor Service. O primeiro argumento é a URL correspondente ao serviço que queremos que irá criar. O terceiro argumento é o domínio do serviço. Nós escolhemos chamar essa "mydomain" domínio.

Em segundo lugar, criar um campo de texto dinâmico chamado "txt_result" e colocá-lo no palco.

Em terceiro lugar, você precisa configurar um gateway web2py que pode se comunicar com o cliente Flash definidos acima.

Continuar criando um app chamado web2py novo pyamf_test que irá hospedar o novo serviço ea porta AMF para o cliente flash. Edite o controlador "default.py" e certifique-se que ele contém

```

1 @service.amfrpc3('mydomain')
2 def addNumbers(val1, val2):
3     return val1 + val2
4
5 def call(): return service()
```

Quarta, compilar e / exportação publicar o cliente flash SWF como local, o 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".

Agora você pode testar o cliente, visite:

```

1 http://127.0.0.1:8000/pyamf_test/static/pyamf_test.html
```

O gateway é posta em segundo plano quando o cliente se conecta a SomaNumeros.

Se você estiver processando AMFo em vez de AMF3 você também pode usar o decorador:

```

1 @service.amfrpc
```

em vez de:

```

1 @service.amfrpc3('mydomain')
```

Neste caso, você também precisará mudar a URL do serviço para:

```

1 http://127.0.0.1:8000/pyamf_test/default/call/amfrpc
```

10.2.4 SOAP

web2py inclui um cliente SOAP e servidor criado por Mariano Reingart. Ele pode ser usado muito como XML-RPC:

Considere o seguinte código, por exemplo, o controlador "default.py":

```
1 @service.soap('MyAdd', returns={'result':int, args={'a':int, 'b':int,}})  
2 def add(a,b):  
3     return a+b
```

Agora, em um shell python que você pode fazer

```
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)  
11 3
```

Você pode obter o WSDL para o serviço de

```
1 http://127.0.0.1:8000/app/default/call?WSDL
```

E você pode obter documentação para qualquer um dos métodos expostos:

```
1 http://127.0.0.1:8000/app/default/call?op=MyAdd
```

10.3 API de baixo nível e Outras Receitas

10.3.1 simplejson

web2py inclui gluon.contrib.simplejson, desenvolvido por Bob Ippolito. Este módulo fornece o mais padrão Python JSON encoder decoder.

Simplejson consiste em duas funções:

- codifica um objeto Python `gluon.contrib.simplejson.dumps(a)` a em JSON.
- decodifica um objeto JavaScript `gluon.contrib.simplejson.loads(b)` b em um objeto Python.

tipos de objeto que pode ser serializado incluir tipos primitivos, listas e dicionários. Os objetos compostos podem ser serializados com exceção das classes definidas pelo usuário.

Aqui está um exemplo de ação (por exemplo, controlador "default.py") que serializa o Python lista contendo a semana usando este API de baixo nível:

```

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)

```

Abaixo está um exemplo de página HTML que envia uma requisição Ajax para a ação acima, recebe a mensagem JSON, e armazena a lista em uma variável JavaScript correspondente:

```

1 {{extend 'layout.html'}}
2 <script>
3 $.getJSON('/application/default/weekdays',
4           function(data){ alert(data); });
5 </script>

```

O código usa o `$.getJSON` função jQuery, que realiza a chamada Ajax e, na resposta, armazena os nomes de dias úteis em uma variável JavaScript `data`

local e passa a variável para a função callback. No exemplo a função de callback simplesmente alerta o visitante que os dados foram recebidos.

10.3.2 PyRTF

Outra necessidade comum de sites da web é o de gerar documentos de texto do Word em formato digital. A maneira mais simples de fazer isso é usando o Rich Text Format (RTF) formato de documento. Este formato foi inventado pela Microsoft e que desde então se tornou um padrão. web2py inclui gluon.contrib.pyrtf, desenvolvido por Simon Cusack, e revisto por Grant Edwards. Este módulo permite gerar documentos RTF programação, incluindo cor de texto formatado e imagens.

No exemplo a seguir que instanciar duas classes básicas RTF, Documento e secção, anexar a este último o ex-manequim e inserir algum texto na segunda:

```
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)
```

No final, o documento é serializado pelo `q.dumps(doc)`. Observe que, antes de retornar um documento RTF que é necessário especificar o tipo de conteúdo no cabeçalho mas o browser não sabe como lidar com o arquivo.

Dependendo da configuração, o navegador pode perguntar-lhe se para salvar o arquivo ou abri-lo usando um editor de texto.

10.3.3 ReportLab e PDF

web2py também pode gerar documentos PDF, com uma biblioteca adicional chamado "ReportLab" (73).

Se você estiver executando web2py da fonte, basta ter ReportLab instalado. Se você estiver executando a distribuição binária do Windows, você precisará descompactar ReportLab "web2py/" na pasta. Se você estiver executando o binário distribuição Mac, você precisa descompactar ReportLab na pasta:

```
1 web2py/app/Contents/Resources/
```

A partir de agora assumimos ReportLab está instalado e que web2py pode encontrá-lo. Criaremos uma ação simples chamado "get_me_a_pdf", que gera um 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 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 como podemos gerar o PDF em um único arquivo temporário, `tmpfilename`, lemos o PDF gerado a partir do arquivo, então, apagou o arquivo.

Para obter mais informações sobre a API do ReportLab, consulte a documentação ReportLab. Recomendamos o uso da API do Platypus ReportLab, como Paragraph, Spacer, etc

10.4 Serviços e autenticação

No capítulo anterior discutimos o uso dos decoradores seguinte:

```
1 @auth.requires_login()  
2 @auth.requires_membership(...)  
3 @auth.requires_permission(...)
```

Para as acções normal (não como os serviços de decoração), estes decoradores podem ser usados mesmo que a saída é renderizada em um formato diferente do HTML.

Para funções definidas como serviços e decorados com os decoradores `@service...`, o `@auth...` decoradores não deve ser utilizado. Os dois tipos de decoradores não podem ser misturados. Se a autenticação deve ser realizada, é o `call` ações que precisa ser decorado:

```
1 @auth.requires_login()  
2 def call(): return service()
```

Observe que também é possível instanciar objetos múltiplos serviços, registrar as diferentes funções mesmo com eles, e expor alguns dos quais com autenticação e outros não:

```
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()
```

Isso pressupõe que o interlocutor está a passar credenciais no cabeçalho HTTP (um cookie de sessão válido ou usando autenticação básica, como discutido na seção anterior). O cliente deve apoiá-la, nem todos os clientes a fazer.

11

Ajax Receitas

Enquanto web2py é principalmente para o desenvolvimento do lado do servidor, o app andaimes **welcome** vem com o jQuery base library (32), calendários jQuery (selecionador de data, selecionador de data e hora e relógio), o menu "superfish.js", e algumas funções adicionais, baseado em JavaScript jQuery.

Nada no web2py impede você de usar outras bibliotecas Ajax (75) como Prototype, ExtJS, ou YUI, mas decidimos pacote jQuery porque nós achamos ser mais fácil de usar e mais poderoso do que qualquer outra equivalente de outras bibliotecas. Nós também achamos que capta o espírito web2py de ser funcional e concisa.

11.1 *web2py_ajax.html*

A aplicação "welcome" web2py andaimes inclui um arquivo chamado

```
views/web2py_ajax.html
```

Esse arquivo está incluído na cabeça o "layout.html" padrão e oferece os seguintes serviços:

- Inclui static/jquery.js.
- Inclui static/calendar.js e static/calendar.css, se eles existirem.
- Define uma função ajax (baseado em jQuery \$. Ajax).
- Faz qualquer DIV de slide "error" or any tag object of class "flash" classe para baixo.
- Evita digitação inteiros inválidos em campos INPUT do "integer" classe.
- Evita digitação inválido flutua em campos INPUT do "double" classe.
- Liga dos campos INPUT "date" tipo com um selecionador de data popup.
- Liga dos campos INPUT "datetime" tipo com um selecionador de data e hora popup.
- Liga dos campos INPUT "time" tipo com um selecionador de tempo popup.
- Define ferramenta web2py_ajax_component e muito importante que será descrito no capítulo 13.

Também inclui popup, collapse, fade e funções para compatibilidade com versões anteriores.

Aqui está um exemplo de como os outros efeitos joguem bem juntos.

Considere um aplicativo **test** com o seguinte modelo:

```
1 db = DAL("sqlite://db.db")
2 db.define_table('child',
3     Field('name'),
4     Field('weigth', 'double'),
5     Field('birth_date', 'date'),
6     Field('time_of_birth', 'time'))
7
8 db.child.name.requires=IS_NOT_EMPTY()
9 db.child.weigth.requires=IS_FLOAT_IN_RANGE(0,100)
10 db.child.birth_date.requires=IS_DATE()
11 db.child.time_of_birth.requires=IS_TIME()
```

com 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)

```

eo "default/index.html" a seguinte opinião:

```

1 {{extend 'layout.html'}}
2 {{=form}}

```

A ação gera "index" da seguinte forma:

Name:	<input type="text"/>
Weigth:	<input type="text"/>
Birth Date:	<input type="text"/>
Time Of Birth:	<input type="text"/>
<input type="button" value="Submit"/>	

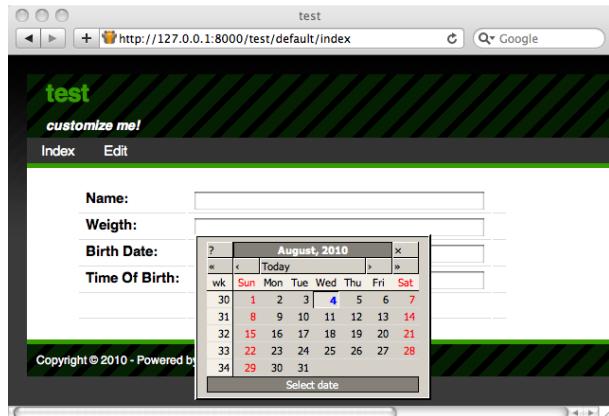
Se um formulário inválido é apresentado, o servidor retorna a página com uma forma modificada que contém mensagens de erro. As mensagens de erro são DIVs do "error" classe, e por causa do código web2py_ajax acima, o erro aparece com um efeito de slide-down:

Name:	<input type="text"/>
	<div style="background-color: red; color: white; padding: 2px;">enter a value</div>
Weigth:	<input type="text"/>
	<div style="background-color: red; color: white; padding: 2px;">enter a number between 0.0 and 100.0</div>
Birth Date:	<input type="text"/>
	<div style="background-color: red; color: white; padding: 2px;">enter date and time as 1963-08-28 14:30:59</div>
Time Of Birth:	<input type="text"/>
	<div style="background-color: red; color: white; padding: 2px;">enter time as hh:mm:ss (seconds, am, pm optional)</div>
<input type="button" value="Submit"/>	

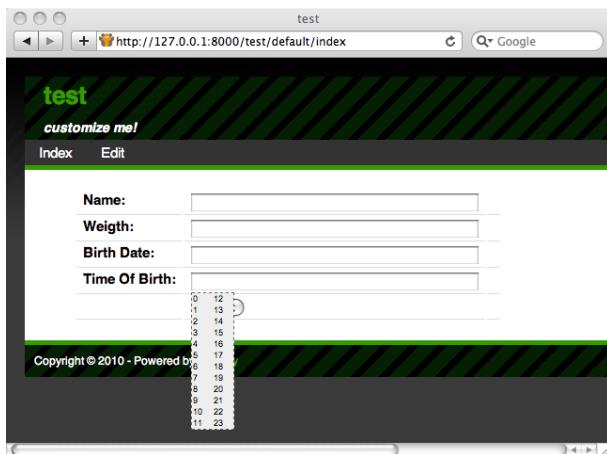
A cor dos erros é dada no código CSS no "layout.html".

O código web2py_ajax impede que você digitar um valor inválido no campo de entrada. Isso é feito antes e para além, não como um substituto para a validação do lado do servidor.

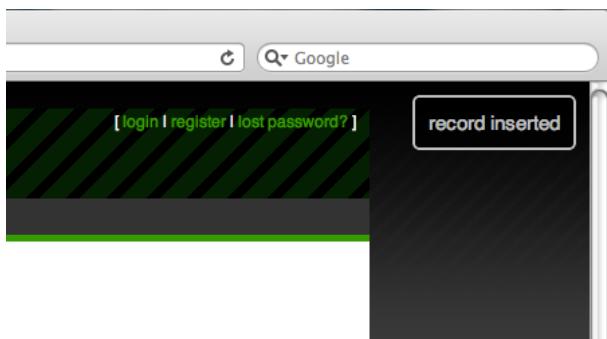
O web2py_ajax código exibe um selecionador de data quando você entra em um campo de entrada do "date", and it displays a datetime picker when you enter an INPUT field of class "datetime" classe. Aqui está um exemplo:



O código web2py_ajax também exibe o selecionador de tempo seguinte quando você tenta editar um campo de entrada da "time" classe:



Após a apresentação, a ação do controlador define o flash resposta à mensagem "record inserted". The default layout renders this message in a DIV with id="flash". O código web2py_ajax é responsável para fazer este slide DIV abaixo e fazendo desaparecer quando você clica sobre ela:



Esses e outros efeitos são acessíveis por meio de programação a vista e via ajudantes nos controladores.

11.2 Efeitos jQuery

Os efeitos básicos aqui descritos não requerem quaisquer arquivos adicionais, tudo que você precisa já está incluído para você web2py_ajax.html.

HTML / XHTML objetos podem ser identificados pelo seu tipo (por exemplo, uma DIV), suas aulas, ou o seu id. Por exemplo:

```
1 <div class="one" id="a">Hello</div>
2 <div class="two" id="b">World</div>
```

Eles pertencem à classe "one" and "two" respectivamente. They have ids equal to "a" and "b", respectivamente.

No jQuery você pode se referir à antiga com o seguinte, como um equivalente notações CSS

```
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"
```

e este, com

```
1 jQuery('.two')
2 jQuery('#b')
3 jQuery('DIV.two')
4 jQuery('DIV #b')
```

ou pode referir-se tanto com

```
1 jQuery('DIV')
```

Tag objetos estão associados a eventos, como o "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>
```

Agora, se você clicar no "Hello", "World" desaparece. If you click again, "World" reaparece. Você pode fazer uma tag ocultos por padrão, dando-lhe uma classe ocultos:

```
1 <div class="one" id="a" onclick="jQuery('.two').slideToggle()">Hello</div>
2 <div class="two hidden" id="b">World</div>
```

Você também pode vincular as ações para eventos fora da marca própria. O código anterior pode ser reescrita da seguinte forma:

```
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>
```

Efeitos de devolver o objeto de chamada, de modo que pode ser preso.

Quando o `click` define a função de callback para ser chamada no clique. Similarmente para `change`, `keyup`, `keydown`, `mouseover` etc

Uma situação comum é a necessidade de executar algum código JavaScript somente após todo o documento foi carregado. Isso geralmente é feito pelo atributo `onload` do corpo mas jQuery fornece uma maneira alternativa que não exige a edição do layout:

```
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>
```

O corpo da função anônima é executada somente quando o documento estiver pronto, depois de ter sido totalmente carregada.

Aqui está uma lista de nomes de eventos úteis:

Forma Eventos

- Onchange: Script para ser executado quando o elemento de mudanças
- onsubmit: Script para ser executado quando o formulário é enviado
- onreset: Script para ser executado quando o formulário é esvaziado
- onselect: Script para ser executado quando o elemento é selecionado
- onblur: Script para ser executado quando o elemento perde o foco
- onfocus: script para ser executado quando o elemento recebe foco

Eventos do Teclado

- Onkeydown: Script para ser executado quando a tecla é pressionada
- onkeypress: Script para ser executado quando a tecla é pressionada e solta
- onkeyup: Script para ser executado quando a tecla é liberada

Eventos do Mouse

- Onclick: Script para ser executado em um clique do mouse
- ondblclick: Script para ser executado em um clique duplo
- onmousedown: Script para ser executado quando o botão do mouse é pressionado
- OnMouseMove: Script para ser executado quando o ponteiro do mouse se move
- onmouseout: Script para ser executado quando o ponteiro do mouse se move para fora de um elemento
- onmouseover: Script para ser executado quando o mouse ponteiro se move sobre um elemento
- onmouseup: Script para ser executado quando o botão do mouse é liberado

Aqui está uma lista de efeitos útil definida pelo jQuery:

Efeitos

- `JQuery(...). attr (nome)`: Retorna o nome do valor do atributo
- `jQuery(...). attr (nome, valor)`: Define o nome do atributo de valor
- `jQuery(...). show ()`: Faz o objeto visível
- `jQuery(...). hide ()`: Faz o objeto escondido
- `jQuery(...). slideToggle velocidade (, callback)`: Faz deslizar o objeto para cima ou para baixo
- `jQuery(...). slideUp velocidade (, callback)`: Faz deslizar o objeto para cima
- `jQuery(...). slideDown velocidade (, callback)`: Faz o objeto de slide para baixo
- `jQuery(...). FadeIn velocidade (, callback)`: Faz o objeto desaparecer em
- `jQuery(...). FadeOut velocidade (, callback)`: Faz o objeto desaparecer

O argumento de velocidade normalmente é omitido ou "slow", "fast" (o padrão). O callback é uma função opcional que é chamado quando o efeito é concluída. jQuery efeitos também podem ser facilmente incorporados em auxiliares, por exemplo, em uma visão:

```
1 {{=DIV('click me!', _onclick="jQuery(this).fadeOut()")}}
```

jQuery é muito compacta e concisa biblioteca Ajax, portanto web2py não precisa de uma camada de abstração adicional no topo do jQuery (exceto para a função ajax discutido abaixo). As APIs jQuery são acessíveis e facilmente disponíveis em sua forma nativa, quando necessário.

Consulte a documentação para obter mais informações sobre estes efeitos e APIs jQuery outros.

A biblioteca jQuery também pode ser estendido usando plugins e User Interface Widgets. Este tópico não é coberto aqui, veja ref. (77) para mais detalhes.

11.2.1 Campos condicional em formulários

Uma aplicação típica de efeitos jQuery é uma forma que muda sua aparência com base no valor dos seus campos.

Isso é fácil de web2py porque o ajudante SQLFORM gera formas que são "CSS friendly". O formulário contém uma tabela com linhas. Cada linha contém um rótulo, um campo de entrada, e uma coluna opcional terceiros. Os itens têm ids derivada estritamente a partir do nome da tabela e os nomes dos campos.

A convenção é que cada campo de entrada tem um tablename_fieldname id e está contido em uma linha com tablename_fieldname_row id.

Como exemplo, criar um formulário de entrada que pede o nome de um contribuinte e para o nome do cônjuge do contribuinte, mas somente se ele é casado.

Criar um aplicativo de teste com o seguinte modelo:

```
1 db = DAL('sqlite://db.db')
2 db.define_table('taxpayer',
3     Field('name'),
4     Field('married', 'boolean'),
5     Field('spouse_name'))
```

o "default.py" controlador seguinte:

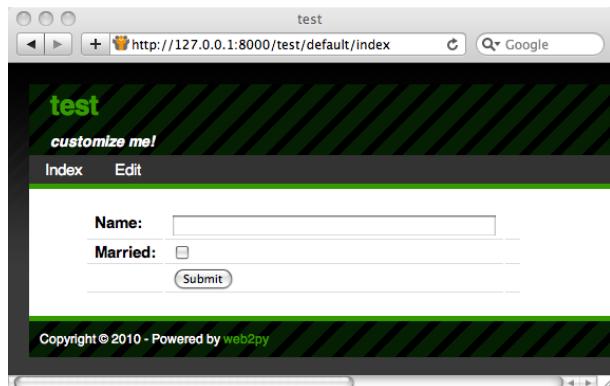
```
1 def index():
2     form = SQLFORM(db.taxpayer)
3     if form.accepts(request.vars, session):
4         response.flash = 'record inserted'
5     return dict(form=form)
```

eo "default/index.html" a seguinte opinião:

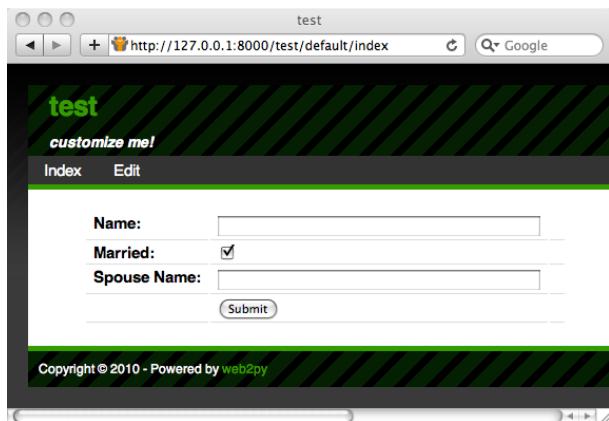
```
1 {{extend 'layout.html'}}
2 {{=form}}
3 <script>
```

```
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>
```

O script na exibição tem o efeito de esconder a linha que contém o nome do cônjuge:



Quando o contribuinte verifica a caixa de "married", o campo do nome do cônjuge reaparece:



Aqui "taxpayer_married" is the checkbox associated to the "boolean" field "married" of table "taxpayer". "taxpayer_spouse_name_row" is the row containing the input field for "spouse_name" of table "taxpayer".

11.2.2 Confirmação em Apagar

Outro aplicativo útil quando é necessária a confirmação do controlo de uma caixa "delete" como excluir a opção que aparece na edição de formulários.

Considere o exemplo acima e adicione a ação do controlador seguinte:

```
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)
```

ea vista correspondente "default/edit.html"

```
1 {{extend 'layout.html'}}
2 {{=form}}
```

O argumento `deletable=True` no construtor `SQLFORM` instrui `web2py` para exibir uma caixa "delete" no formulário de edição. É `False` por padrão.

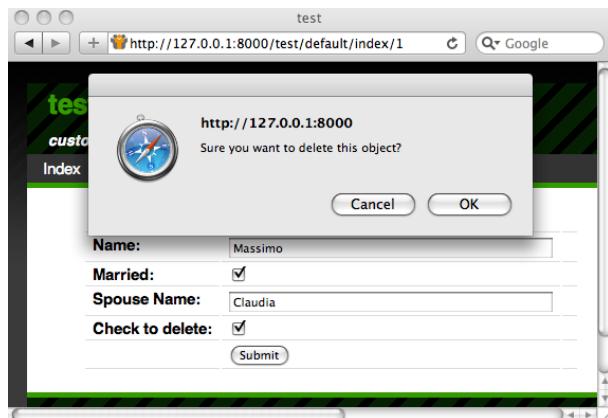
"`web2py_ajax.html`" é `web2py` inclui o seguinte código:

```

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 convenção, esta opção tem uma classe igual ao "delete". O código `jQuery` acima conecta o evento `onclick` do checkbox com uma janela de confirmação (padrão em JavaScript) e desmarca a opção, caso o contribuinte não confirme:



11.3 Função O ajax

Em `web2py_ajax.html`, `web2py` define uma função chamada `ajax` que se baseia, mas não deve ser confundido com o `$.ajax` função `jQuery`. O último é muito mais poderoso do que o anterior, e para seu uso, que encaminhá-lo para ref. (32) e ref. (76). No entanto, a função primeira é suficiente para

muitas tarefas complexas, e é fácil de usar.

A função ajax é uma função JavaScript que tem a seguinte sintaxe:

```
1 ajax(url, [id1, id2, ...], target)
```

Ele chama o argumento de forma assíncrona (url primeiro), passa os valores dos campos com o id igual a um dos ids da lista (segundo argumento), em seguida, armazena a resposta no innerHTML da marca com o id igual ao de destino (o terceiro argumento).

Aqui está um exemplo de um controlador de default:

```
1 def one():
2     return dict()
3
4 def echo():
5     return request.vars.name
```

eo "default/one.html" vista associada:

```
1 {{extend 'layout.html'}}
2 <form>
3     <input id="name" onkeyup="ajax('echo', ['name'], 'target')" />
4 </form>
5 <div id="target"></div>
```

Quando você digita algo no campo de entrada, assim você libera uma tecla (onkeyup), a função ajax é chamado, eo valor do campo id="name" é passado para o "echo" ação, que envia o texto volta para o ponto de vista. A função ajax recebe a resposta e exibe a resposta de eco no DIV "target".

11.3.1 alvo Eval

O terceiro argumento da função ajax pode ser o ":eval" string. Isto significa que a string retornada pelo servidor não serão incorporados no documento, mas ele será avaliado.

Aqui está um exemplo de um controlador de default:

```

1 def one():
2     return dict()
3
4 def echo():
5     return "jQuery('#target').html(%s);" % repr(request.vars.name)

```

eo "default/one.html" vista associada:

```

1 {{extend 'layout.html'}}
2 <form>
3     <input id="name" onkeyup="ajax('echo', ['name'], ':eval')"/>
4 </form>
5 <div id="target"></div>

```

Isso permite respostas mais articuladas do que strings simples.

11.3.2 Auto-realização

Web2py contém um built-in widgets autocomplete, foi descrito no capítulo sobre Forms. No entanto, aqui queremos construir uma mais simples a partir do zero.

Outra aplicação da função acima ajax é auto-realização. Aqui queremos criar um campo de entrada que espera um nome de mês e, quando o visitante digita um nome incompleto, realiza auto-realização através de uma solicitação do Ajax. Em resposta, uma conclusão auto-drop, aparece abaixo do campo de entrada.

Isto pode ser conseguido através do default controlador seguinte:

```

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])

```

eo "default/month_input.html" vista correspondente:

```

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>

```

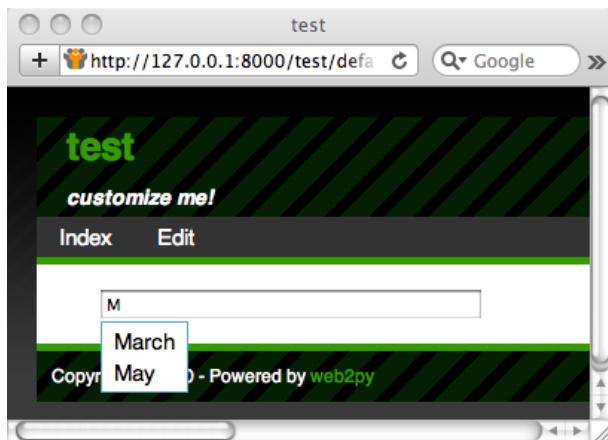
O script jQuery em vista o pedido Ajax dispara cada vez que o visitante digita algo no "month" input field. The value of the input field is submitted with the Ajax request to the "month_selector" action. This action finds a list of month names that start with the submitted text (selected), builds a list 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" a ação retorna callback:

```

1 <div onclick="jQuery('#month').val('February')"
2   onmouseout="this.style.backgroundColor='white'"
3   onmouseover="this.style.backgroundColor='yellow'">February</div>

```

Aqui é o efeito final:



Se os meses são armazenadas em uma tabela do banco de dados, tais como:

```
1 db.define_table('month', Field('name'))
```

em seguida, basta substituir a ação month_selector com:

```
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])
```

Plugin do jQuery oferece uma opção de auto-completar com funcionalidades adicionais, mas isso não é discutida aqui.

11.3.3 Ajax Submission Form

Aqui, consideramos uma página que permite ao visitante enviar mensagens usando Ajax sem recarregar a página inteira. Web2py proporciona um melhor mecanismo para fazê-lo do que o descrito aqui, usando o ajudante carga e será descrito no capítulo 13, mas aqui queremos mostrar como fazê-lo simplesmente usando jQuery.

Ele contém uma forma DIV "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".

Criar um aplicativo test com o seguinte 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()
```

Observe que cada post tem um "your_message" único campo que é necessário para ser não-vazio.

Editar default.py controlador e escrever duas ações:

```
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()])
```

A primeira ação é nada mais do que um retorno vista.

A segunda ação é a chamada Ajax. Ele espera que as variáveis de forma request.vars, os processa e retorna ao sucesso DIV("Message posted") ou TABLE de mensagens de erro em caso de falha.

Agora edite o ponto 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 como neste exemplo, o formulário é criado manualmente, usando HTML, mas ela é processada pelo SQLFORM em uma ação diferente da que exibe o formulário. O objeto SQLFORM nunca é serializado em HTML. SQLFORM.accepts neste caso não terá uma sessão de jogos e formname=None, porque optamos por não definir o nome do formulário e uma chave de formulário no formulário HTML manual.

O script na parte inferior da exibição liga o "myform" botão para apresentar uma função inline, que defende a entrada com id="your_message" usando a função web2py ajax, e exibe a resposta dentro da DIV com o id="target".

11.3.4 Voto e Avaliação

Outra aplicação Ajax é o voto ou a avaliação itens em uma página. Aqui vamos considerar uma aplicação que permite aos visitantes que votem em imagens postadas. A aplicação consiste em uma única página que exibe as imagens classificadas de acordo com o seu voto. Vamos permitir que os visitantes a votar várias vezes, embora seja fácil mudar esse comportamento se os visitantes são autenticados, por se manter a par dos votos individuais

no banco de dados e associá-los com o `request.env.remote_addr` do eleitor.

Aqui está um modelo de exemplo:

```
1 db = DAL('sqlite://images.db')
2 db.define_table('item',
3     Field('image', 'upload'),
4     Field('votes', 'integer', default=0))
```

Aqui é o 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)
```

A ação `download` é necessário para permitir a `list_items` para baixar imagens armazenadas na pasta "uploads". Os votos de acção é usada para a chamada Ajax.

Aqui está o ponto de vista "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}}
```

Quando o visitante clica no campo de entrada "[vote up]" the JavaScript code stores the item.id in the hidden "id" e sustenta esse valor para o servidor através de uma requisição Ajax. O servidor aumenta os votos contra para o registro correspondente e retorna a contagem de novo voto como uma string. Este valor é então inserido no alvo SPAN item{{=item.id}}.

Ajax callbacks — podem ser usadas para executar cálculos em segundo plano, mas recomendamos o uso **cron** ou um processo de fundo, em vez (discutido no capítulo 4), uma vez que o servidor web impõe um limite de linhas. Se o cálculo leva muito tempo, o servidor web mata-lo. Consulte o seu servidor web parâmetros para definir o valor limite. —

12

Implantação Receitas

Existem várias maneiras de implantar web2py em um ambiente de produção, os detalhes dependem da configuração e dos serviços prestados pelo anfitrião.

Neste capítulo, vamos considerar os seguintes aspectos:

- implantação de produção (Apache, Lighttpd, Cherokee)
- Segurança
- Escalabilidade
- Implantação na plataforma Google App Engine (13))

web2py vem com um servidor web SSL (21) habilitado, o wsgiserver (22) Rocket. Embora este seja um servidor de web rápido, tem capacidades de configuração limitada. Por esta razão é melhor implementar web2py atrás Apache (79), Lighttpd (86) ou Cherokee (87). Estes são livres e de código aberto servidores web que podem ser personalizados e têm provado ser fiável em ambientes de alta produção de trânsito. Eles podem ser configuradas para servir arquivos estáticos diretamente, lidar com HTTPS, e passar o controle para web2py para conteúdo dinâmico.

Até alguns anos atrás, a interface padrão para comunicação entre servidores web e aplicações web foi o Common Gateway Interface (CGI) (78). O principal problema com o CGI é que ele cria um novo processo para cada solicitação HTTP. Se a aplicação web é escrita em uma linguagem interpretada, a cada solicitação HTTP servido pelos scripts CGI inicia uma nova instância do interpretador. Isto é lento, e deve ser evitado em um ambiente de produção. Além disso, o CGI pode apenas lidar com respostas simples. Não se pode tratar, por exemplo, arquivo de streaming. `web2py modpythonhandler.py` fornece um arquivo para a interface CGI.

Uma solução para esse problema é usar o módulo `mod_python` para o Apache. Nós discutimos isso aqui, porque a sua utilização é ainda muito comum, mas o `mod_python` projeto foi oficialmente abandonado pela Apache Software Foundation. `mod_python` inicia uma instância do interpretador Python, quando inicia o Apache, e serve cada solicitação HTTP na sua própria linha sem ter que reiniciar Python cada vez. Esta é uma solução melhor do CGI, mas não é uma solução ideal, uma vez que `mod_python` usa sua própria interface de comunicação entre o servidor web ea aplicação web. Em `mod_python`, todos os aplicativos hospedados executado sob a user-id/group-id mesmo, que apresenta problemas de segurança. `web2py` fornece um arquivo de interface para `cgihandler.py mod_python`.

Nos últimos anos, a comunidade Python se reúne atrás de um novo padrão para interface de comunicação entre servidores web e aplicações web escritas em Python. Ele é chamado de Web Server Gateway Interface (WSGI) (17; 18). `web2py` foi construída sobre WSGI, e fornece manipuladores para usar outras interfaces quando WSGI não está disponível.

Apache suporta WSGI `mod_wsgi` (85) através do módulo desenvolvido por Graham Dumpleton. `web2py` fornece um arquivo para `wsgihandler.py` interface para WSGI.

Alguns serviços de hospedagem web não suportam `mod_wsgi`. Neste caso, devemos usar o Apache como um proxy e encaminhar todas as solicitações de entrada para o `web2py` built-in web server (rodando por exemplo, local-

host: 8000).

Em ambos os casos, com mod_wsgi e / ou mod_proxy, o Apache pode ser configurado para servir arquivos estáticos e lidar com a criptografia SSL diretamente, tendo a carga fora web2py.

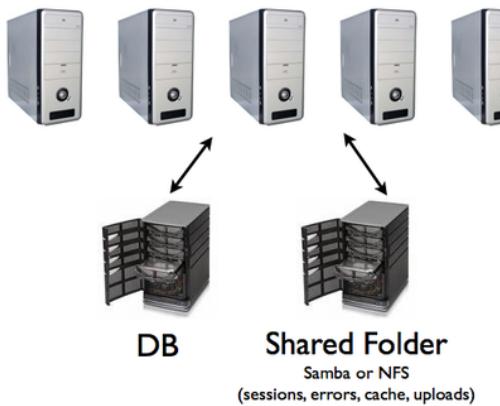
O servidor web Lighttpd atualmente não suporta a interface WSGI, mas dá suporte a interface FastCGI (88), que é uma melhoria sobre CGI. FastCGI principal objectivo é reduzir a sobrecarga associada à interface entre o servidor web e programas CGI, permitindo que um servidor para lidar com solicitações HTTP mais de uma vez.

De acordo com o site 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 com FastCGI é, de fato, mais rápido que o Apache com mod_wsgi. web2py fcgihandler.py fornece um arquivo para fazer interface com FastCGI. web2py gaehandler.py também inclui uma interface com o Google App Engine (GAE). No jogo, aplicações web rodar "in the cloud". Isso significa que o quadro abstrai completamente todos os detalhes de hardware. O aplicativo da Web é automaticamente replicada tantas vezes quanto necessário para atender todas as solicitações simultâneas. Replicação neste caso, significa mais do que vários segmentos em um único servidor, mas também significa processos múltiplos em diferentes servidores. GAME atinge esse nível de escalabilidade, bloqueando o acesso de gravação para o sistema de arquivos e todas as informações persistentes devem ser armazenados no armazenamento de dados BigTable Google ou em memcache.

Em plataformas não-GAE, a escalabilidade é um assunto que precisa ser tratada, e pode exigir alguns ajustes nas aplicações web2py. A maneira mais comum para alcançar escalabilidade é usando vários servidores web atrás de um balanceador de carga (um round robin simples, ou algo mais sofisticado, recebendo feedback pulsação dos servidores).

Mesmo que existam vários servidores web, deve haver uma, e apenas um

servidor de banco de dados. Por padrão, web2py utiliza o sistema de arquivo para armazenar as sessões, bilhetes de erro, os arquivos enviados, e do cache. Isto significa que na configuração padrão, as pastas correspondentes pastas a serem compartilhadas:



No restante do capítulo, consideramos várias receitas que podem proporcionar uma melhora esta abordagem ingênuia, incluindo:

- sessões Store no banco de dados em cache ou não armazene sessões em tudo.
- Store bilhetes em sistemas de arquivos locais e movê-los para o banco de dados em lotes.
- Use memcache em vez de cache.ram e cache.disk.
- Armazene arquivos carregados no banco de dados em vez do sistema de arquivos compartilhados.

Enquanto nós recomendamos seguir as três primeiras receitas, a receita quarto pode proporcionar uma vantagem, principalmente no caso de arquivos pequenos, mas pode ser contraproducente para arquivos grandes.

12.1 Linux / Unix

12.1.1 Implantação One Step Production

Aqui estão alguns passos para instalar o apache + python + mod_wsgi + postgresql + web2py a partir do zero.

No Ubuntu:

```
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
```

Em Fedora:

```
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 estes script deve ser executado fora da caixa, mas a cada instalação do Linux é um pouco diferente e você quiser dentro do código fonte. Assim, certifique-se de executar o código fonte destes script antes de executar. A maioria do que eles fazem é explicado a seguir no caso do Ubuntu. Eles não implementar as otimizações escalabilidade discutidos abaixo.

12.1.2 configuração do Apache

Nessa seção, nós usamos o Ubuntu 8.04 Server Edition como a plataforma de referência. Os comandos de configuração são muito semelhantes em outras distribuição Linux baseada no Debian, mas eles podem ser diferentes para os sistemas Fedora-based (que usa o yum em vez de apt-get).

Primeiro, certifique-se o necessário e Python Apache todos os pacotes são instalados por digitar os comandos de shell a seguir:

```
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
```

Em seguida, ativar o módulo SSL, o módulo de proxy, e WSGI módulo do Apache:

```
1 sudo ln -s /etc/apache2/mods-available/proxy_http.load \
           /etc/apache2/mods-enabled/proxy_http.load
2 sudo a2enmod ssl
3 sudo a2enmod proxy
4 sudo a2enmod proxy_http
5 sudo a2enmod wsgi
```

Crie a pasta SSL, e colocar os certificados SSL dentro dele:

```
1 sudo mkdir /etc/apache2/ssl
```

Você deve obter os certificados SSL de uma autoridade de certificação confiável como verisign.com, mas, para fins de teste, você pode gerar seus próprios certificados auto-assinados, seguindo as instruções em ref. (84)

Em seguida, reiniciar o servidor web:

```
1 sudo /etc/init.d/apache2 restart
```

O arquivo de configuração do Apache é:

```
1 /etc/apache2/sites-available/default
```

Os logs do Apache está em:

```
1 /var/log/apache2/
```

12.1.3 mod_wsgi

Download e fonte web2py descompactar na máquina onde você instalou o servidor web acima.

Instale web2py no /users/www-data/, por exemplo, e dar posse ao usuário www-data e grupo www-data. Essas etapas podem ser executadas com os comandos de shell a seguir:

```

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 com mod_wsgi, crie um novo arquivo de configuração do Apache:

```
/etc/apache2/sites-available/web2py
```

e incluir o seguinte 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
34 </VirtualHost>
```

Ao reiniciar o Apache, ele deve passar todos os pedidos de web2py sem passar pelo wsgiserver Rocket.

Aqui estão algumas explicações:

```
1 WSGIDaemonProcess web2py user=www-data group=www-data
 2 display-name=%{GROUP}
```

define um grupo de processos daemon no contexto do "web2py.example.com". Ao definir esta dentro da máquina virtual, somente este host virtual, incluindo qualquer máquina virtual de mesmo nome do servidor, mas em uma porta diferente, pode aceder a este usando WSGIProcessGroup. As opções do "user" and "group" deve ser definida para o usuário que tem acesso de gravação para o diretório onde foi instalado web2py. Você não precisa definir "user" e "group", se você fez o diretório de instalação web2py gravável para o usuário que o Apache é executado como por omissão. "display-name" é a opção para que o nome aparece no processo de produção como ps "(wsgi:web2py)" em vez de o nome do servidor web Apache executável. Como não "processes" ou "threads" opções especificado, o grupo de processos daemon terá um único processo com 15 threads em execução no âmbito desse processo. Isso geralmente é mais que suficiente para a maioria dos sites e deveria ser deixada como está. Se é imperiosa, não use o "processes=1" como isso irá desativar qualquer navegador em ferramentas de depuração WSGI que verificam a bandeira "wsgi.multiprocess". Isso ocorre porque qualquer uso da opção "processes" fará com que a bandeira ser definido como verdadeiro, mesmo se um único processo e ferramentas como esperar que ela

seja definida como false. Note que, se o código do aplicativo próprio ou de algum módulo de extensão de terceiros que você está usando com Python não é o segmento de seguro, em vez usar as opções de "processes=5 threads=1". Isto vai criar cinco processos no grupo de processos daemon, onde cada processo é único thread. Você pode usar o "maximum-requests=1000" se a fuga de aplicações Python os objetos através de incapacidade para que o lixo seja coletado corretamente.

```
1 WSGIProcessGroup web2py
```

delegados de todos os aplicativos em execução ao grupo WSGI processo daemon que foi configurado usando a diretiva WSGIDaemonProcess.

```
1 WSGIScriptAlias / /users/www-data/web2py/wsgihandler.py
```

monta o pedido web2py. Neste caso, é montado na raiz do site. Não se sabe como chegar web2py para montar em uma sub URL como não parece ser um bom cidadão WSGI e descobrir onde é montado a partir de valor de SCRIPT_NAME e automaticamente ajustar tudo de forma adequada sem a configuração manual do usuário ainda mais.

```
1 <Directory /users/www-data/web2py>
2 ...
3 </Directory>
```

Apache dá permissão para acessar o arquivo de script de WSGI.

```
1 <Directory /users/www-data/web2py/applications/*/*static/>
2   Order Allow,Deny
3   Allow from all
4 </Directory>
```

Encarrega o Apache para ignorar web2py ao pesquisar arquivos estáticos.

```
1 <Location /admin>
2   Deny from all
3 </Location>
```

ea

```
1 <LocationMatch ^/([^\]+)/appadmin>
2   Deny from all
3 </LocationMatch>
```

bloquear o acesso do público aos **admin** e **appadmin**

Normalmente, seria apenas permitir que a permissão para o diretório todo o arquivo de script WSGI é localizado, mas não posso fazer isso com web2py, como ele coloca o arquivo de script de WSGI em um diretório que contém o código de outra fonte, inclusive o arquivo contendo a senha interface admin. A abertura do diretório inteiro poderia causar problemas de segurança, porque tecnicamente Apache seria dada permissão para servir todos os arquivos até um usuário, se havia alguma maneira de atravessar para o diretório através de uma URL mapeada. Para evitar problemas de segurança, explicitamente negar acesso a o conteúdo do diretório, com exceção para o script de WSGI arquivo e proibir um usuário de fazer qualquer substituição a partir de um htaccess. ficheiro para ser mais seguro.

Você pode encontrar um concludo, comentou, Apache arquivo de configuração wsgi em:

```
1 scripts/web2py-wsgi.conf
```

Esta seção foi criada com a ajuda de Dumpleton Graham, desenvolvedor do mod_wsgi.

12.1.4 mod_wsgi e SSL

Para forçar algumas aplicações (como por exemplo **admin** e **appadmin**) para passar por cima de HTTPS, armazenar o certificado SSL e arquivos-chave:

```
1 /etc/apache2/ssl/server.crt
2 /etc/apache2/ssl/server.key
```

e editar o arquivo de configuração do Apache `web2py.conf` e acrescentar:

```

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>
```

Reinic peace o Apache e você deve ser capaz de acessar:

```

1 https://www.example.com/admin
2 https://www.example.com/examples/appadmin
3 http://www.example.com/examples
```

mas não:

```

1 http://www.example.com/admin
2 http://www.example.com/examples/appadmin
```

12.1.5 mod_proxy

Alguns Unix / distribuições Linux pode rodar Apache, mas não suportam mod_wsgi. Neste caso, a solução mais simples é executar o Apache como um proxy e ter Apache lidar com arquivos estáticos só.

Aqui está uma configuração Apache minimalista:

```
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>
```

O script acima expõe apenas o app "welcome" application. To expose other applications, you need to add the corresponding <Location>...</Location> with the same syntax as done for the "welcome".

O script assume que há um servidor web2py executando na porta 8000. Antes de reiniciar o Apache, certifique-se esse for o caso:

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

Você pode especificar uma senha com a opção -a ou "<recycle>" usar o parâmetro em vez de uma senha. Neste último caso, a senha previamente ar-

mazenada é reutilizada ea senha não são armazenadas no histórico de shell.

Você também pode usar o parâmetro "<ask>", a ser solicitado para uma senha.

Os comandos nohup faz se o servidor não morre quando você fechar o shell. nohup registra todas as saídas para nohup.out.

Para forçar o admin e appadmin sobre HTTPS usar o seguinte arquivo de configuração do Apache vez:

```

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>
```

A interface administrativa deve ser desativado quando web2py é executado em um servidor compartilhado com mod_proxy, ou ele vai ser exposto a outros usuários.

12.1.6 Comece como Daemon Linux

A menos que você estiver usando mod_wsgi, você deve configurar o servidor web2py para que ele possa ser iniciado / parado / reiniciado como qualquer outro servidor Linux, e para que ele possa iniciar automaticamente na fase de inicialização do computador.

O processo para configurar este é específico para várias distribuições Linux / Unix.

Na pasta web2py, existem dois scripts que podem ser usados para essa finalidade:

```
1 scripts/web2py.ubuntu.sh
2 scripts/web2py.fedora.sh
```

No Ubuntu, ou outra baseada no Debian Linux, editar "web2py.ubuntu.sh" and replace the "/usr/lib/web2py" caminho com o caminho de sua instalação web2py, em seguida, digite os comandos de shell a seguir para mover o arquivo para a pasta apropriada, registrá-lo como um serviço de inicialização e iniciá-lo:

```
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
```

No Fedora, ou quaisquer outras distribuições baseadas no Fedora, edite o caminho "web2py.fedora.sh" and replace the "/usr/lib/web2py" com o caminho de sua instalação web2py, em seguida, digite os comandos de shell a seguir para mover o arquivo para a pasta apropriada, registrá-lo como um serviço de inicialização e iniciá-lo:

```
1 sudo cp scripts/web2py.fedora.sh /etc/rc.d/init.d/web2pyd
2 sudo chkconfig --add web2pyd
3 sudo service web2py start
```

12.1.7 Lighttpd

Você pode instalar o Lighttpd em um Ubuntu ou outra distribuição baseada no Debian Linux com o seguinte comando shell:

```
1 apt-get -y install lighttpd
```

Uma vez instalado, /etc/rc.local editar e criar um fundo processo web2py fcgi

```
1 cd /var/www/web2py && sudo -u www-data nohup python fcgihandler.py &
```

Em seguida, você precisará editar o arquivo de configuração Lighttpd

```
1 /etc/lighttpd/lighttpd.conf
```

para que ele possa encontrar o socket criado pelo processo acima. No arquivo de configuração, escrever 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 server.port = 80
12 server.bind = "0.0.0.0"
13 server.event-handler = "freebsd-kqueue"
14 server.error-handler-404 = "/test.fcgi"
15 server.document-root = "/users/www-data/web2py/"
16 server.errorlog      = "/tmp/error.log"
17
18 fastcgi.server = (
19     "/handler_web2py.fcgi" => (
20         "handler_web2py" => ( #name for logs
21             "check-local" => "disable",
22             "socket" => "/tmp/fcgi.sock"
23         )
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 }
```

Agora, verificar se há erros de sintaxe:

```
1 lighttpd -t -f /etc/lighttpd/lighttpd.conf
```

e (re) iniciar o servidor web com:

```
1 /etc/init.d/lighttpd restart
```

Observe que liga o servidor FastCGI web2py para um socket Unix, não a uma tomada IP:

```
1 /tmp/fcgi.sock
```

Este é o lugar onde Lighttpd encaminha os pedidos HTTP e recebe respostas. Sockets Unix são mais leves que sockets de Internet, e esta é uma das razões Lighttpd + FastCGI + web2py é rápido. Como no caso do Apache, é possível configurar Lighttpd para lidar com arquivos estáticos diretamente, e para

forçar algumas aplicações em HTTPS. Consulte a documentação Lighttpd para obter mais detalhes.

Os exemplos nesta seção foram retirados de John post Heenan em web2pyslices.

A interface administrativa deve ser desativado quando web2py é executado em um servidor compartilhado com FastCGI, ou ele vai ser exposto para os outros usuários.

12.1.8 Hospedagem Compartilhada com mod_python

Há tempos, mais especificamente em hosts compartilhados, quando não se tem permissão para configurar o Apache config arquivos diretamente. Na hora de escrever a maioria destes hosts ainda executar mod_python mesmo se não for mantido mais a favor de mod_wsgi. Você ainda pode executar web2py. Aqui nós mostramos um exemplo de como configurá-lo.

Coloque o conteúdo de web2py para a pasta "htdocs".

Na pasta web2py, criar um ficheiro do "web2py_modpython.py" com o seguinte conteúdo:

```
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)
```

Criar / atualizar o ".htaccess" arquivo com o seguinte conteúdo:

```
1 SetHandler python-program
2 PythonHandler web2py_modpython
3 #PythonDebug On
```

Este exemplo foi fornecido por Niktar.

12.1.9 Cherokee com FastGGI

Cherokee é um servidor web muito rápido e, como web2py, ele fornece uma AJAX-enabled-based interface web para sua configuração. Sua interface web é escrita em Python. Além disso, não há reiniciar exigido para a maioria das mudanças.

Aqui estão os passos necessários para instalação com web2py Cherokee:

Download Cherokee (87)

Descompacte, construir 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
```

web2py iniciar normalmente pelo menos uma vez para certificar-se que cria a pasta "applications".

Escrever um script shell chamado "startweb2py.sh" com o seguinte código:

```
1 #!/bin/bash
2 cd /var/web2py
3 python /var/web2py/fcgihandler.py &
```

e dar o script executar privilégios e executá-lo. Isto irá iniciar web2py sob manipulador FastCGI.

Cherokee em Iniciar e admin-cherokee:

```
1 sudo nohup cherokee &
2 sudo nohup cherokee-admin &
```

Por padrão, cherokee admin somente escuta na interface local na porta 9090. Este não é um problema se você tiver acesso físico completo na máquina. Se

este não for o caso, você pode forçá-lo de se ligar a um endereço IP e a porta, usando as seguintes opções:

```
1 -b, --bind[=IP]
2 -p, --port=NUM
```

ou fazer um SSH porta para a frente (mais seguro, recomendado):

```
1 ssh -L 9090:localhost:9090 remotehost
```

Open "http://localhost:9090" em seu navegador. Se tudo estiver ok, você receberá admin-cherokee.

Em admin cherokee interface web "info sources". Choose "Local Interpreter". Write in the following code, then click "Add New", clique.

```
1 Nick: web2py
2 Connection: /tmp/fcgi.sock
3 Interpreter: /var/web2py/startweb2py.sh
```

Finalmente, execute as seguintes etapas restantes:

- Clique "Virtual Servers", then click "Default".
- Clique "Behavior", then, under that, click "default".
- Escolha "FastCGI" instead of "List and Send" na caixa de listagem.
- No fundo, selecione "web2py" as "Application Server"
- Coloque uma verificação em todas as caixas (você pode deixar Permita-x-sendfile). Se houver um aviso exibido, desativar e ativar uma das opções. (Será automaticamente re-apresentar o parâmetro do servidor de aplicativos. Às vezes isso não acontece, que é um bug).
- Aponte seu navegador para "http://yoursite", and "Welcome to web2py" irá aparecer.

12.1.10 PostgreSQL

O PostgreSQL é um banco de dados livre e de código aberto, que é utilizado em ambientes de produção, por exemplo, para armazenar o arquivo. Banco de dados nome org, e tem sido comprovada a escala bem em centenas de terabytes de dados. Tem muito rápido e sólido suporte à transação, e fornece um recurso de auto-vácuo que libera o administrador de tarefas mais manutenção de banco de dados.

Em um Ubuntu ou outra distribuição baseada em Debian Linux, é fácil de instalar o PostgreSQL e suas Python API com:

```
1 sudo apt-get -y install postgresql  
2 sudo apt-get -y install python-psycopg2
```

É aconselhável rodar o servidor web (s) eo servidor de banco de dados em diferentes máquinas. Neste caso, as máquinas que executam os servidores Web devem ser ligados a uma rede interna segura (física), ou deve estabelecer túneis SSL segura para contato com o servidor de banco de dados.

Edite o arquivo de configuração do PostgreSQL

```
1 sudo nano /etc/postgresql/8.4/main/postgresql.conf
```

e certifique-se que contém essas duas linhas

```
1 ...  
2 track_counts = on  
3 ...  
4 autovacuum = on    # Enable autovacuum subprocess? 'on'  
5 ...
```

Inicie o servidor de banco de dados com:

```
1 sudo /etc/init.d/postgresql restart
```

Quando reiniciar o servidor PostgreSQL, deve notificar a porta que está sendo executado. A menos que você tenha vários servidores de banco de dados, deve ser 5432.

O PostgreSQL logs está em:

```
1 /var/log/postgresql/
```

Depois que o servidor de banco de dados está instalado e funcionando, crie um usuário e um banco de dados para que as aplicações web2py pode usá-lo:

```
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
```

O primeiro dos comandos vai permitir acesso de superusuário para o novo usuário, chamado `myuser`. Ele pedirá uma senha.

Qualquer pedido web2py pode se conectar a esse banco de dados com o comando:

```
1 db = DAL("postgres://myuser:mypassword@localhost:5432/mydb")
```

onde `mypassword` é a senha que você digitou quando solicitado, e `5432` é a porta onde o servidor está executando.

Normalmente, você usar um banco de dados para cada aplicação, e várias instâncias do mesmo aplicativo se conectar ao mesmo banco de dados. Também é possível para diferentes aplicações para compartilhar o mesmo banco de dados.

Para mais detalhes backup do banco de dados, leia a documentação do PostgreSQL, especificamente os comandos `pg_dump` e `pg_restore`.

12.2 Windows

12.2.1 Apache e mod_wsgi

Instalando o Apache, e mod_wsgi sob o Windows requer um procedimento diferente. Aqui estão assumindo Python 2.5 é instalado, você está executando a partir da fonte e web2py está localizado na c:/web2py.

Primeiro baixe os pacotes requer:

- Apache do apache_2.2.11-win32-x86-openssl-0.9.8i.msi (80)
- mod_wsgi do (81)

Em segundo lugar, apache...msi executar e siga as telas do assistente. Na tela de informações do servidor



inserir todos os valores solicitados:

- **Network Domain:** digite o domínio DNS no qual o servidor está ou será registrado dentro Por exemplo, se o nome do seu servidor DNS completo está server.mydomain.net, você deverá digitar mydomain.net aqui

- **ServerName:** Nome do seu servidor de DNS total. A partir do exemplo acima, você deverá digitar server.mydomain.net aqui. Digite um nome de domínio totalmente qualificado ou endereço IP do web2py instalar, não é um atalho para obter mais informações, consulte o (83).
- **Administrator's Email Address.** Digite o administrador do servidor ou o endereço de e-mail webmaster aqui. Este endereço será exibido junto com mensagens de erro para o cliente, por padrão.

Continue com uma instalação típica para o final, salvo necessário

O assistente, por padrão, o Apache instalado na pasta:

```
1 C:/Program Files/Apache Software Foundation/Apache2.2/
```

De agora em diante vamos referir a esta pasta simplesmente como Apache2.2.

Em terceiro lugar, a cópia transferida mod_wsgi.so para Apache2.2/modules escrito por Chris Travers, publicado pela Open Source Software Lab da Microsoft, Dezembro de 2007.

Em quarto lugar, criar certificados server.crt e server.key (como discutido na seção anterior) e colocá-los no Apache2.2/conf pasta. Observe o arquivo cnf está na Apache2.2/conf/openssl.cnf.

Em quinto lugar, Apache2.2/conf/httpd.conf editar, remover a marca de comentário (o # caracteres) a partir da linha

```
1 LoadModule ssl_module modules/mod_ssl.so
```

adicone a seguinte linha depois de todas as outras linhas LoadModule

```
1 LoadModule wsgi_module modules/mod_wsgi.so
```

olhar para "Listen 80" e adicione esta linha após a

```
1 Listen 443
```

acrescente as seguintes linhas no final mudar letra de unidade, número de porta, ServerName acordo com seus 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>
```

Salvar e verificar a configuração usando: [Iniciar> Programas> Apache HTTP Server 2.2> Configurar Apache Server> Test Configuration]

Se não houver problemas, você verá uma tela de comando de abrir e fechar. Agora, você pode iniciar o Apache:

[Iniciar> Programas> Apache HTTP Server 2.2> Control Server> Start Apache] ou melhor ainda, iniciar o monitor da barra de tarefas

[Start > Program > Apache HTTP Server 2.2 > Control Apache Server]

Agora você pode botão direito sobre a pena como ícone da barra vermelha para abrir Apache Monitor ea partir dela iniciar, parar e reiniciar o Apache, conforme necessário.

Esta seção foi criada por Jonathan Lundell.

12.2.2 Comece como serviço do Windows

O que chama um servidor Linux, Windows solicita um serviço. Web2py O servidor pode ser facilmente instalado / iniciado / parado como um serviço Windows.

Para poder utilizar web2py como um serviço Windows, você deve criar um arquivo "options.py" com parâmetros de inicialização:

```

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()
```

Você não precisa criar "options.py" from scratch since there is already an "options_std.py" na pasta web2py que você pode usar como modelo.

Depois de criar "options.py" na pasta de instalação web2py, você pode instalar web2py como um serviço com:

```
1 python web2py.py -W install
```

e iniciar ou parar o serviço com:

```
1 python web2py.py -W start  
2 python web2py.py -W stop
```

12.3 Sessões de Proteção e admin

É muito perigoso para expor publicamente a aplicação e os controladores **admin** **appadmin**, a menos que atropelar HTTPS. Além disso, sua senha e credenciais nunca devem ser transmitidas sem criptografia. Isso é verdadeiro para web2py e qualquer aplicação web.

Em suas aplicações, se eles requerem autenticação, você deve fazer os cookies de sessão seguro com:

```
1 session.secure()
```

Uma maneira fácil de configurar um ambiente de produção em um servidor seguro é a primeira paragem web2py e remova todos os arquivos `parameters_*.py` da pasta de instalação web2py. Então comece web2py sem uma senha. Isto irá desativar completamente admin e appadmin.

Em seguida, iniciar uma segunda instância Python acessível apenas a partir de localhost:

```
1 nohup python web2py -p 8001 -i 127.0.0.1 -a '<ask>' &
```

e criar um túnel SSH a partir da máquina local (a partir do qual você deseja acessar a interface administrativa) para o servidor (a web2py onde está sendo executado, `example.com`), utilizando-se:

```
1 ssh -L 8001:127.0.0.1:8001 username@example.com
```

Agora você pode acessar a interface administrativa local através do navegador web na `localhost:8001`.

Esta configuração é seguro porque **admin** não é alcançável quando o túnel está fechado (o usuário é autenticado).

— Esta solução é segura em hosts compartilhados, se e somente se outros usuários não têm acesso de leitura para a pasta que contém web2py, caso contrário os utilizadores podem ser capazes de roubar cookies de sessão diretamente do servidor. — —

12.4 Eficiência e Escalabilidade

web2py é projetado para ser fácil de instalar e configurar. Isso não significa que compromete a eficiência ou a escalabilidade, mas isso significa que você pode precisar ajustá-lo para torná-lo escalável.

Nesta seção nós assumimos várias instalações web2py atrás de um servidor NAT que fornece local de平衡amento de carga.

Neste caso, web2py funciona out-of-the-box se algumas condições forem satisfeitas. Em particular, todas as instâncias de cada aplicação web2py deve acessar o mesmo banco de dados e servidores deve ver os mesmos arquivos. Esta última condição pode ser implementada através de pastas compartilhadas seguintes:

```

1 applications/myapp/sessions
2 applications/myapp/errors
3 applications/myapp/uploads
4 applications/myapp/cache

```

As pastas compartilhadas deve apoiar o bloqueio de ficheiros. Possíveis soluções são ZFS (ZFS foi desenvolvido pela Sun Microsystems e é a escolha preferida.), NFS (Com NFS pode ser necessário para executar the lockmgr daemon para permitir o bloqueio de arquivos.), ou Samba (SMB).

É possível, mas não uma boa idéia, para compartilhar a pasta web2py in-

teiro ou a pasta de aplicações inteiras, porque isso causaria um aumento desnecessário da banda de uso da rede.

Nós acreditamos que a configuração acima referidas ser muito escalável, pois reduz a carga de banco de dados de passar para o sistema de arquivos compartilhado os recursos que precisam ser partilhadas, mas não precisa de segurança transacional (apenas um cliente ao mesmo tempo é suposto aceder a um arquivo de sessão, cache sempre tem um bloqueio global, uploads e os erros são write once / li muitos arquivos).

Idealmente, tanto o banco de dados e armazenamento compartilhado deve ter capacidade de RAID. Não cometa o erro de armazenar o banco de dados sobre o armazenamento mesmo que as pastas compartilhadas, ou você vai criar um gargalo novo lá.

Numa base caso a caso, pode ser necessário para realizar otimizações adicionais e vamos discuti-las abaixo. Em particular, discutiremos como se livrar dessas pastas compartilhadas de um por um, e como armazenar os dados associados na base de dados em vez disso. Embora isso seja possível, não é necessariamente uma boa solução. No entanto, pode haver razões para fazê-lo. Um desses motivos é que muitas vezes não temos a liberdade para criar pastas compartilhadas.

12.4.1 Eficiência Truques

web2py código do aplicativo é executado em cada pedido que você quer minimizar essa quantidade de código. Aqui está o que você pode fazer:

- Executar uma vez com o `migrate=True` em seguida, definir todas as suas tabelas para `migrate=False`.
- Bytecode compilar seu aplicativo usando o **admin**.
- Use `cache.ram` tanto quanto você pode, mas certifique-se usar um conjunto

finito de chaves mais a quantidade de cache usado vai crescer de forma arbitrária.

- Minimize o código em modelos: não define as funções de lá, define as funções dos controladores em que deles necessitam, ou - melhor ainda - define as funções em módulos, importá-los e usar essas funções, conforme necessário.
- Não coloque muitas funções no mesmo controlador, mas utilizam vários controladores com poucas funções.
- Definir `session.forget()` em todos os controladores e / ou funções que não mudam a sessão.
- Tentar anular cron web2py, e utilizar um processo em segundo plano em seu lugar. cron web2py pode começar demasiados casos Python e fazer uso excessivo de memória.

12.4.2 Sessões no banco de dados

É possível instruir web2py para salvar sessões em um banco de dados, em vez de na pasta sessões. Isto tem que ser feito para cada aplicação individual web2py embora todos eles podem usar o mesmo banco de dados para armazenar as sessões.

Atendendo uma ligação de dados

```
1 db = DAL(...)
```

você pode armazenar as sessões neste banco de dados (db), simplesmente informando o seguinte, no arquivo de mesmo modelo que estabelece a conexão:

```
1 session.connect(request, response, db)
```

Se ele não existir, web2py cria, sob o capô, uma tabela no banco de dados chamado `web2py_session_apname` contendo os seguintes 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" é uma chave uuid usado para identificar a sessão no cookie.
"session_data" é cPickled os dados da sessão.

Para minimizar o acesso ao banco, você deve evitar o armazenamento de
sessões, quando não são necessários com:

```
1 session.forget()
```

Com este tweak a pasta "sessions" não precisa ser uma pasta compartilhada,
pois deixará de ser acessado. — Observe que, se as sessões são definidas,
não devem passar para o session e você não pode usar o form.accepts
session.flash nem CRUD. —

12.4.3 HAProxy um balanceador de carga de alta disponibilidade

Se você precisa web2py vários processos em execução em várias máquinas,
em vez de sessões de armazenar no banco ou em cache, você terá a possibilidade
de utilizar um balanceador de carga com sessões pegajosa.

Pound (89) e HAProxy (90) dois平衡adores de carga e Reverse proxies
HTTP que proporciona sessões pegajosa. Aqui vamos discutir este último
porque parece ser mais comum em VPS hospedagem comercial.

Por sessões pegajoso, entendemos que uma vez que um cookie da sessão
tenha sido emitido, o balanceador sempre encaminhar pedidos do cliente
associada à sessão, para o servidor mesmo. Isso permite que você armazene
a sessão no sistema de arquivos local, sem necessidade de um sistema de
arquivos compartilhados.

Para usar HAProxy:

Em primeiro lugar, instalá-lo na saída da máquina de teste do Ubuntu:

```
1 sudo apt-get -y install haproxy
```

Segundo editar o arquivo de configuração "/etc/haproxy.cfg" para algo como isto:

```
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    svrtimeout    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
```

A directiva diz HAProxy `listen`, que porto para aguardar conexão. `server` A directiva diz HAProxy onde encontrar os servidores proxy. O diretório `appsession` faz uma sessão pegajoso e utiliza o cookie chamado um `WEB2PYSTICKY` para esse fim.

Em terceiro lugar, permitir que este arquivo de configuração e iniciar HAProxy:

```
1 /etc/init.d/haproxy restart
```

Você pode encontrar instruções de configuração similar ao Pound na URL

```
1 http://web2pyslices.com/main/slices/take_slice/33
```

12.4.4 Limpeza Sessions

Se você optar por manter as suas sessões no sistema de arquivos, você deve estar ciente de que, em um ambiente de produção que se acumulam rapidamente. web2py fornece um script chamado:

```
1 scripts/sessions2trash.py
```

que, quando executado no fundo, periodicamente exclui todas as sessões que não foram acessados por um determinado período de tempo. Este é o conteúdo do script:

```
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)
```

Você pode executar o script com o seguinte comando:

```
1 nohup python web2py.py -S myapp -R scripts/sessions2trash.py &
```

onde myapp é o nome do seu aplicativo.

12.4.5 Fazer upload de arquivos no banco de dados

Por padrão, todos os arquivos carregados são manipulados por SQLFORMs segurança renomeados e armazenados no sistema de arquivos sob a pasta "uploads". É possível instruir web2py para armazenar arquivos no banco de dados em vez disso.

Agora, considere a seguinte tabela:

```

1 db.define_table('dog',
2     Field('name')
3     Field('image', 'upload'))
```

onde `dog.image` é de `upload` tipo. Para fazer o upload da imagem vá no registro mesmo que o nome do cão, você deve modificar a definição da tabela, adicionando um campo blob e vinculá-lo ao campo de upload:

```

1 db.define_table('dog',
2     Field('name')
3     Field('image', 'upload', uploadfield='image_data'),
4     Field('image_data', 'blob'))
```

Aqui "image_data" é apenas um nome arbitrário para campo blob novo.

Linha 3 instrui web2py com segurança renomear upload de imagens como de costume, o nome da nova loja no campo da imagem, e armazenar os dados no uploadfield chamado "image_data" ao invés de armazenar os dados no sistema de arquivos. Tudo isto é feito automaticamente pelo SQLFORMs e nenhum outro código precisa ser alterada.

Com esta emenda, a pasta "uploads" não é mais necessária.

Em arquivos do Google App Engine são armazenados por padrão no banco de dados sem a necessidade de definir um uploadfield, um é criado por padrão.

12.4.6 Coleta de ingressos

Por padrão, os bilhetes web2py lojas (erros) no sistema de arquivos local. Não faria sentido para armazenar os bilhetes diretamente na base de dados, porque a origem mais comum de erro em um ambiente de produção é a insuficiência de dados.

Armazenar bilhetes nunca é um gargalo, pois este é normalmente um evento raro, portanto, num ambiente de produção com vários servidores em simultâneo, é mais do que adequado para armazená-los em uma pasta compartilhada. No entanto, uma vez que apenas o administrador precisa recuperar os bilhetes, também é OK para armazenar os bilhetes de forma não-pasta compartilhada local "errors" e periodicamente coletá-los e / ou eliminá-los.

Uma possibilidade é deslocar periodicamente todos os bilhetes para um banco de dados local.

Para este efeito, web2py fornece o seguinte roteiro:

```
1 scripts/tickets2db.py
```

que contém:

```
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 deve ser editado. Altere a seqüência de DB_URI para que ele se conecta ao seu servidor de banco de dados e executá-lo com o comando:

```
1 nohup python web2py.py -S myapp -M -R scripts/tickets2db.py &
```

onde myapp é o nome do seu aplicativo.

Este script é executado em segundo plano e se move a cada 5 minutos todos os bilhetes para o servidor de banco de dados em uma tabela chamada "ticket" e remove os bilhetes local. Se ALLOW_DUPLICATES é definido

como False, ele só irá armazenar os bilhetes que correspondem a diferentes tipos de erros. Com esta emenda, a pasta "errors" não precisa ser uma pasta compartilhada de mais, uma vez que só será acessado localmente.

12.4.7 Memcache

Nós mostramos que web2py fornece dois tipos de cache: cache.ram e cache.disk. Ambos trabalham em um ambiente distribuído com vários servidores simultâneos, mas não funciona como esperado. Em particular, cache.ram irá armazenar apenas no nível do servidor, portanto, torna-se inútil. cache.disk também será cache no nível do servidor a menos que o "cache" pasta é uma pasta compartilhada que suporta bloqueio, portanto, em vez de acelerar as coisas, torna-se um gargalo importante.

A solução não é para usá-los, mas para usar memcache vez. web2py vem com um memcache API.

Para usar memcache, crie um arquivo novo modelo, por exemplo, 0_memcache.py e escrever neste arquivo (ou acrescentar) o seguinte 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
```

A primeira linha memcache importações. A segunda linha tem de ser uma lista de sockets memcache (servidor: porta). A terceira linha redefine cache.ram e cache.disk em termos de memcache.

Você pode optar por redefinir apenas um deles para definir uma cache totalmente novo objeto apontando para o objeto Memcache.

Com este tweak a pasta "cache" não precisa ser uma pasta compartilhada de mais, uma vez que deixará de ser acessado.

Este código requer ter memcache servidores rodando na rede local. Você deve consultar a documentação memcache para obter informações sobre como configurar os servidores.

12.4.8 Sessões em Memcache

Se você precisa fazer sessões e você não quiser usar um balanceador de carga com sessões pega-joso, você tem a opção de salvar sessões em memcache:

```
1 from gluon.contrib.memdb import MEMDB
2 session.connect(request,response, db=MEMDB(cache.memcache))
```

12.4.9 Remover Aplicativos

Em um ambiente de produção, talvez seja melhor não instalar os aplicativos padrão: **admin**, **examples** e **welcome**. Embora estas aplicações são muito pequenas, elas não são necessárias.

A remoção desses aplicativos é tão fácil como apagar as pastas correspondentes na pasta de aplicativos.

12.4.10 Usando servidores de bancos de dados replicados

Em um ambiente de alto desempenho que você pode ter um banco de dados arquitetura mestre-escravo com muitos escravos replicados, e talvez um par de servidores replicados. A DAL pode lidar com esta situação e condicionalmente se conectar a servidores diferentes, dependendo dos parâmetros pedido. A API para fazer isso foi descrito no Capítulo 6. Aqui está um exemplo:

```
1 from random import shuffle
2 db=DAL(shuffle(['mysql://...1','mysql://...2','mysql://...3']))
```

Neste caso, os pedidos HTTP diferente será servida por diferentes bases de dados de forma aleatória e cada DB será atingido mais ou menos com a mesma probabilidade.

Podemos também aplicar um simples 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'))
```

Esta é a falha de segurança no sentido de que se o servidor de banco de dados atribuído à solicitação não se conecta, DAL tentará o próximo na ordem.

Se possível também se conectar a bancos de dados diferentes, dependendo da ação solicitada ou controlador. Em uma configuração de banco de dados mestre-escravo, executa alguma ação só ler um e algumas pessoas tanto de leitura / gravação. O formato pode seguramente se conectar a um servidor db escravo, enquanto o segundo deve se conectar a um master. Então você pode fazer:

```
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 onde são replicados servidores e 3,4,5 são escravos.

12.5 Google App Engine

É possível executar código web2py no Google App Engine (GAE) (13), incluindo o código DAL, com algumas limitações. A plataforma de jogo oferece várias vantagens em relação normal soluções de hospedagem:

- Facilidade de implantação. Google abstrai completamente a arquitetura subjacente.
- Escalabilidade. Google vai replicar o app quantas vezes é preciso para servir todas as solicitações simultâneas
- BigTable. No jogo, em vez de um banco de dados relacional normal, você armazenar informações persistentes em BigTable, o armazenamento de dados do Google é famosa.

As limitações são:

- Você tem não sabe ler nem escrever o acesso ao sistema de arquivos.
- Nenhuma transação
- Não é possível executar consultas complexas sobre o armazenamento de dados, em particular não há JOIN, LIKE, IN, e operadores de DATE / DATETIME.
- Você pode usar a declaração ou entre sub-consultas que envolvem um único e mesmo campo.

Isto significa que web2py não pode armazenar as sessões, os bilhetes de erro, arquivos de cache e arquivos carregados no sistema de arquivos, eles devem ser armazenados no armazenamento de dados e não no sistema de arquivos.

Aqui nós fornecemos uma visão geral do jogo e focamos web2py questões específicas, encaminhá-lo para o online documentação oficial para maiores informações. — Atenção: Na hora de escrever jogo suporta apenas Python 2.5. Qualquer outra versão irá causar problemas. Você também deve executar a fonte web2py distribuição, e não uma distribuição binária. —

Vamos prosseguir em ordem.

12.5.1 Configuração

Há dois arquivos de configuração para estar ciente de:

```
1 web2py/app.yaml
2 web2py/index.yaml
```

"app.yaml"" tem a seguinte estrutura (que tem sido reduzido com...):

```
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 ...
```

Você deve substituir o web2py com o ID do aplicativo que você usou ao se registrar com o Google App Engine.

url: /_ha/stats.* instrui GAME para expor a URL "/stats" com estatísticas e informações de perfis.

url: /(.+)/static/(.+) instrui GAME para servir arquivos estáticos seu aplicativo diretamente, sem chamar a lógica web2py, para a velocidade.

url: /_ha/admin/.* instrui GAME para expor a interface do jogo administrativo.

url: /_ah/queue/default instrui GAME para expor uma interface para a fila de tarefas. Isto é usado por web2py a fim de manter recarregar as páginas

automaticamente e mantê-los em cache para agilizar o serviço.

`url:.* instrui web2py usar o gaehandler.py para cada solicitação de outros.`

A sessão `skip_files:` é uma lista de expressões regulares para arquivos que não precisam ser implantados no jogo. Em particular a linha:

```
1 ((admin|examples|welcome)\.tar) |
```

O jogo não conta para implantar os aplicativos padrão. Você pode adicionar aqui uma aplicação mais ser ignorado.

Salvo para aplicação id você provavelmente não precisará editar `app.yaml`.

O `index.yaml` arquivo é gerado automaticamente quando você executar o aplicativo localmente usando o appserver GAME (o servidor web que vem com o SDK do Google). Ele contém algo parecido com isto:

```
1 indexes:
2 - kind: person
  properties:
    - name: name
      direction: desc
```

Neste exemplo, ele diz jogo para criar um índice para "person" that will be used to sort by "name" em ordem alfabética inversa. Você não será capaz de procurar e classificar os registros em seu aplicativo sem índices correspondentes.

É sempre importante rodar seus aplicativos localmente com o appserver e tentar todas as funcionalidades da sua aplicação, antes da implantação. Isto será importante para testar, mas determinou também para gerar automaticamente o ficheiro "index.yaml". Às vezes você pode querer editar este arquivo e executar limpeza, tais como a remoção de entradas duplicadas.

12.5.2 Duração e de implantação

Aqui vamos supor que você tenha instalado o SDK GAME. Você pode executar seu aplicativo de dentro da pasta "web2py" usando o comando appserver:

```
1 python2.5 dev_appserver ./web2py
```

Isto irá iniciar o appserver e você pode executar o aplicativo na URL:

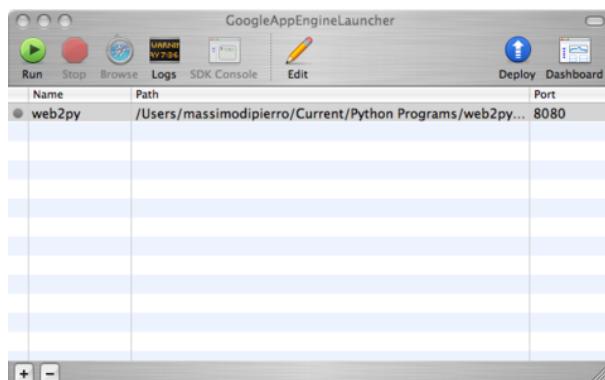
¹ <http://127.0.0.1:8080/>

Para fazer upload de sua aplicação no jogo, ter certeza de que o arquivo editado "app.yaml", como explicado antes e definir o id correcta aplicação e execute:

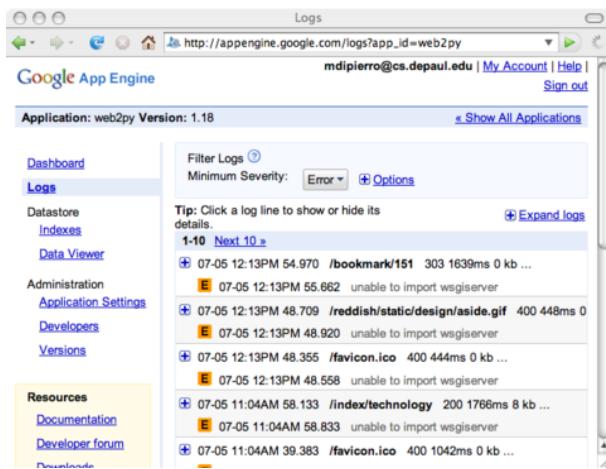
```
1 python2.5 appcfg.py update ./web2py
```

No Mac e viúvas você também pode usar o Google App Engine Launcher. Você pode baixar o software de ref. (13).

Escolha [arquivo] [Add Existing Application], defina o caminho para o caminho da pasta web2py de nível superior, e pressione o botão [Run] na barra de ferramentas. Depois de ter testado que funciona localmente, você pode implantá-lo no jogo, basta clicar sobre o botão [] Implantar na barra de ferramentas (assumindo que você tenha uma conta).



No jogo, os bilhetes web2py / erros também são registrados no console de administração do jogo, onde os registros podem ser acessados e procurado online.



12.5.3 Configurando o manipulador

O gaehandler.py arquivo é responsável por servir os arquivos no jogo e tem algumas opções. Aqui estão os valores padrão:

```

1 KEEP_CACHED = False
2 LOG_STATS = False
3 APPSTATS = True
4 DEBUG = False
5 AUTO_RETRY = True

```

KEEP_CACHED pede GAME para armazenar tarefas na fila de tarefas que ficam ligando uma ação manequim forçando GAME para o cache do seu pedido. Isto fará GAME servir os seus utilizadores mais rápido, mas ele vai consumir alguns ciclos extra.

LOG_STATS irá registrar a hora de servir as páginas do GAME logs.

APPSTATS permitirá appstats GAME que fornece estatísticas de perfis. Eles estarão disponíveis na URL:

```
1 http://localhost:8080/_ah/stats
```

DEBUG define o modo de depuração. Ele não faz diferença, na prática, menos marcada explicitamente em seu código via gluon.settings.web2py_runtime.

AUTO_RETRY instrui jogo para tentar cometer novamente em caso de falha.

12.5.4 Evite o Filesystem

No jogo você não tem acesso ao sistema de arquivos. Você não pode abrir qualquer arquivo para a escrita.

Para o efeito, no jogo, web2py armazena automaticamente todos os arquivos carregados no armazenamento de dados, com ou sem campo "upload" (s) tem um atributo uploadfield.

Você também deve armazenar as sessões e os bilhetes do banco de dados e você tem de 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')
```

O código acima verifica se está a executar no jogo, conecta-se BigTable, e instrui web2py para armazenar as sessões e bilhetes de avião para lá. Ele se conecta a um banco de dados SQLite em contrário. Esse código já está na app andaime no "db.py" arquivo.

12.5.5 Memcache

Se preferir, você pode armazenar as sessões em 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))

```

Observe que no jogo cache.ram cache.disk e não deve ser usado, por isso, torná-las ponto a cache.memcache.

12.5.6 Questões de banco de dados

A ausência de operações entidade multi-funcionalidades típicas de bancos de dados relacionais são o que define jogo além de ambiente de hospedagem outros. Este é o preço a pagar por alta escalabilidade. Jogo é uma excelente plataforma, se essas limitações são toleráveis, se não, então uma plataforma regular de hospedagem com um banco de dados relacional deveria ser considerado.

Se um aplicativo web2py não funciona no jogo, é porque de uma das limitações acima discutidas. A maioria dos problemas podem ser resolvidos removendo JOINs web2py de consultas e de-normalizar o banco de dados.

No entanto, o Google App Engine suportar alguns tipos de campos que não são mapeadas diretamente em tipos DAL, para StringListProperty exemplo. Você ainda pode usar esses tipos de web2py usando a seguinte sintaxe.

```

1 from gluon.contrib.gql import gae
2 db.define_table('myitem',
3     Field('name'),
4     Field('keywords', type=gae.StringListProperty())

```

Aqui o campo é um StringListProperty palavras-chave, portanto, seus valores devem ser as listas de strings, compatíveis com a documentação do jogo.

12.5.7 GAME e HTTPS

Se a aplicação que você tem id "myapp" é o seu domínio GAME

¹ <http://myapp.appspot.com/>

e também pode ser acessado via HTTPS

¹ <https://myapp.appspot.com/>

Neste caso, ele irá usar um certificado "appspot.com" fornecido pelo Google.

Pode registrar uma entrada de DNS e usar qualquer outro nome de domínio que você próprio para sua aplicação, mas, neste caso, você não poderá usar HTTPS. Na hora de escrever esta limitação GAME.

13

Outras receitas

13.1 Melhoramento

No botão "site" page of the administrative interface there is an "upgrade now". Caso isso não é viável ou não funcionar (por exemplo, devido a um problema de travamento de arquivo), a atualização manualmente web2py é muito fácil.

Basta descompactar a versão mais recente do web2py sobre a instalação antiga.

Isto irá atualizar todas as bibliotecas, bem como as aplicações **admin**, **examples**, **welcome**. Ele também irá criar um "NEWINSTALL". Upon restarting, web2py will delete the empty file and package the welcome app into "welcome.w2p" novo arquivo vazio que será usado como andaime novo app. web2py não atualizar qualquer arquivo em outros aplicativos existentes.

13.2 Como distribuir seus aplicativos como binários

É possível agrupar o app com a distribuição binária web2py e distribuí-los juntos. A licença permite isso, desde que você faça clara a licença do seu aplicativo que você está empacotando com web2py e adicionar um link para a web2py.com.

Aqui nós explicamos como fazê-lo para Windows:

- Criar seu app como de costume
- Usando o **admin**, bytecode compilar seu aplicativo (um clique)
- Usando o **admin**, pack seu aplicativo compilado (clique)
- Crie uma pasta "myapp"
- Baixe um windows web2py distribuição binária
- Descompacte-o em pasta "myapp" e iniciá-lo (dois cliques)
- Upload usando **admin** previamente embalados e compilados app com o nome de "init" (um clique)
- Crie um arquivo "myapp/start.bat" that contains "cd web2py; web2py.exe"
- Crie um arquivo "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 a pasta myapp em um arquivo "myapp.zip"
- Distribuir e / ou vender "myapp.zip"

Quando os usuários unzip app "myapp.zip" and click "run" they will see your app instead of the "welcome". Não há nenhuma exigência do lado do usuário, nem mesmo Python pré-instalado.

Para binários Mac o processo é o mesmo, mas não há nenhuma necessidade para o arquivo "bat".

13.3 Buscando uma URL externa

Python inclui a biblioteca `urllib` para buscar urls:

```
1 import urllib
2 page = urllib.urlopen('http://www.web2py.com').read()
```

Isso é muitas vezes bem, mas o módulo `urllib` não funciona no Google da App Engine. Google oferece uma API diferente para fazer o download de URL que funciona apenas no jogo. A fim de tornar o código portável, web2py inclui uma função de `fetch` que funciona no jogo, bem como outras instalações Python:

```
1 from google.tools import fetch
2 page = fetch('http://www.web2py.com')
```

13.4 Pretty datas

Muitas vezes, é útil para representar não um datetime como "2009-07-25 14:34:56" but as "one year ago". web2py fornece uma função de utilidade para isso:

```
1 d = datetime.datetime(2009,7,25,14,34,56)
2 from google.tools import prettydate
3 pretty_d = prettydate(d,T)
```

O segundo argumento (T) deve ser passado para permitir a internacionalização da produção.

13.5 Geocoding

Se você precisa converter um endereço (por exemplo: "243 S Wabash Ave, Chicago, IL, USA") em coordenadas geográficas (latitude e longitude), web2py

fornecendo uma função para fazer isso.

```
1 from gluon.tools import geocode
2 address = '243 S Wabash Ave, Chicago, IL, USA'
3 (latitude, longitude) = geocode(address)
```

O geocode função requer uma conexão de rede e se conectar ao serviço Google geocodificação para a geocodificação. A função retorna $(0,0)$ em caso de falha. Observe que o Google geocoding caps serviços, o número de pedidos e você deve verificar seu contrato de serviço. A função do geocode é construído em cima da função fetch e assim que funciona no jogo.

13.6 Paginação

Esta receita é um truque útil para minimizar o acesso ao banco em caso de paginação, por exemplo, quando você precisa exibir uma lista de linhas de um banco de dados, mas você quiser distribuir as linhas ao longo de várias páginas.

Comece por criar um pedido de **primes** que armazena os primeiros 1000 números primos em um banco de dados.

Aqui está o 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
```

Agora crie uma ação no `list_items` "default.py" controlador que diz assim:

```

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 esse código seleciona mais um item que é necessário, $20 + 1$. A razão é que o elemento extra narra a ver se existe uma próxima página.

Aqui está o ponto de vista "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}}

```

Desta forma obtivemos paginação com uma única escolha por ação, e que selecione apenas seleciona uma linha mais do que precisamos.

13.7 httpserver.log eo Log File Format

O servidor web web2py registra todas as solicitações para um arquivo chamado:

```
1 httpserver.log
```

no diretório raiz web2py. Um nome alternativo e localização pode ser especificada via linha de comando opções de web2py.

As novas entradas são acrescentados ao final do arquivo de cada vez que um pedido é feito. Cada linha parecida com esta:

```
1 127.0.0.1, 2008-01-12 10:41:20, GET, /admin/default/site, HTTP/1.1, 200, 0.270000
```

O formato é:

```
1 ip, timestamp, method, path, protocol, status, time_taken
```

Onde

- IP é o endereço IP do cliente que fez a solicitação
- timestamp é a data ea hora da solicitação no formato ISO 8601, AAAA-MM-DDT HH: MM: SS
- método GET ou POST
- caminho é o caminho solicitado pelo cliente
- protocolo é o protocolo HTTP usado para enviar para o cliente, geralmente HTTP/1.1
- status é um dos códigos de status HTTP (92)
- time_taken é a quantidade de tempo que o servidor teve para processar o pedido, em segundos, não incluindo o upload / download tempo.

No (34) repositório aparelhos, você encontrará um equipamento para análise de log.

Este registo é desabilitada por padrão quando usando mod_wsgi pois seria o mesmo que o log do Apache.

13.8 Preenchendo banco de dados com dados fictícios

Para fins de teste é conveniente para ser capaz de preencher as tabelas do banco com dados fictícios. web2py inclui um classificador Bayesiano já treinadas para gerar o texto fictício, mas legível para esta finalidade.

Aqui é a maneira mais simples de utilizar:

```
1 from gluon.contrib.populate import populate
2 populate(db.mytable,100)
```

Isto irá inserir 100 registros fictícios em db.mytable. Ele vai tentar fazer de forma inteligente, gerando pequeno texto para campos string, texto mais longo para campos de texto, números inteiros, duplos, datas datetimes, horários, valores booleanos, etc para os campos correspondentes. Ele vai tentar respeitar as exigências impostas pela validadores. Para os campos que contém a palavra "name" ele vai tentar gerar nomes fictícios. Para os campos de referência que irá gerar referências válidas.

Se você tiver duas tabelas (A e B), onde as referências A, B certifique-se de preencher um primeiro e segundo B.

Porque a população é feito em uma operação, não tente preencher muitos registros ao mesmo tempo, em especial se as referências estão envolvidos. Ao invés disso, preencher 100 na época, cometer loop.

```
1 for i in range(10):
2     populate(db.mytable,100)
3     db.commit()
```

Você pode usar o classificador Bayesiano para aprender algum texto e gerar o texto simulado que soa similar, mas não deve fazer 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)
```

13.9 Enviar um SMS

SMS Enviar mensagens de um aplicativo web2py requer um serviço de terceiros que podem retransmitir as mensagens para o receptor. Normalmente,

isso não é um serviço gratuito, mas difere de país para país. Tentamos alguns destes serviços, com pouco sucesso, porque as empresas de telefonia bloquear e-mails provenientes destes serviços, uma vez que são eventualmente utilizados como fonte de spam.

Uma maneira melhor é usar as empresas de telefonia se a invocar o SMS. Cada companhia telefônica tem um endereço de e-mail unicamente associado a cada número de telefone celular, o SMS pode ser enviada como um e-mail para que o número de telefone. web2py vem com um módulo para ajudar neste processo:

```
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 é um dicionário que mapeia nomes de empresas de telefonia importante para o postfix e-mail. A função sms_email tem um número de telefone (como string) e o nome de uma companhia telefônica e retorna o endereço de email do telefone.

13.10 Aceitar pagamentos com cartão de crédito

web2py oferece várias maneiras para a sua aplicação a aceitar pagamentos com cartão de crédito. Por exemplo:

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

Os dois primeiros mecanismos acima delegar o processo de autenticação do beneficiário de um serviço externo. Embora esta seja a melhor solução para a segurança (sua app não manipula todas as informações em qualquer cartão de crédito) que torna o processo moroso (o usuário precisa fazer login duas vezes, com sua aplicação, e com o Google, por exemplo) e não permite sua aplicação a lidar com os pagamentos periódicos de uma forma automatizada.

Há momentos em que você precisa de mais controle. Por esta razão, oferecemos integração fora da caixa com a API Authorize.Net (o módulo foi desenvolvido por John Conde e ligeiramente modificada). Aqui está um exemplo de fluxo de trabalho e todas as variáveis que estão expostas:

```

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()
```

Observe o código acima usa uma conta de teste simulado. Você precisa se registrar com Authorize.Net (não é um serviço gratuito) e fornecer o seu login, transkey, TestMode = Verdadeiro ou Falso para o construtor AIM.

13.11 Twitter API

Aqui estão alguns exemplos simples de como postar / receber tweets. Nenhum bibliotecas de terceiros são necessárias, uma vez que usa Twitter simples APIs RESTful.

Aqui está um exemplo de como enviar um 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())
```

Aqui está um exemplo de como receber 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 operações mais complexas, consulte a documentação da API do Twitter.

13.12 Streaming de Arquivos Virtual

É comum que os invasores mal-intencionados para escanear sites web em busca de vulnerabilidades. Eles usam scanners de segurança, como o Nessus para explorar os sítios Web de destino para scripts que são conhecidas por ter vulnerabilidades. Uma análise de registos do servidor web a partir de uma máquina digitalizada ou diretamente do banco de dados Nessus revela que a maioria das vulnerabilidades conhecidas estão em scripts PHP e scripts ASP. Uma vez que estamos executando web2py, não temos essas vulnerabilidades, mas ainda será examinado para eles. Esta irritante, por isso gostamos de como responder a esses scans de vulnerabilidade e fazer o atacante entender o seu tempo está sendo desperdiçado.

Uma possibilidade é redirecionar todas as solicitações para. Php, asp. E nada de suspeito a uma ação fictícia que vai responder ao ataque, mantendo o atacante ocupado por uma grande quantidade de tempo. Eventualmente, o atacante vai desistir e não vai digitalizar conosco novamente.

Esta receita exige duas partes.

Um aplicativo dedicado chamado **jammer** "default.py" com um controlador como segue:

```
1 class Jammer():
2     def read(self,n): return 'x'*n
3     def jam(): return response.stream(Jammer(),40000)
```

Quando essa ação é chamada, ela responde com um fluxo de dados infinito cheio de "x"-ES. 40 mil caracteres de cada vez.

O segundo ingrediente é um arquivo "route.py" que redireciona todos os pedidos que terminem em. Php. Asp, etc (tanto maiúsculas e minúsculas) para este controlador.

```
1 route_in=
2 ('.*\.(php|PHP|asp|ASP|jsp|JSP)', 'jammer/default/jam'),
3 )
```

A primeira vez que você é atacado você pode incorrer uma pequena sobre-carga, mas nossa experiência é que o atacante mesmo não vai tentar duas vezes.

13.13 Jython

web2py funciona normalmente em CPython (o interpretador Python codificado em C), mas também pode ser executado em Jython (o interpretador Python codificado em Java). Isso permite que web2py para ser executado em uma infra-estrutura Java.

Mesmo que fora executado com web2py Jython da caixa, há alguns truques envolvidos na criação de Jython e na criação de zxJDBC (o adaptador de banco de dados Jython). Aqui estão as instruções:

- Baixe o arquivo "jython_installer-2.5.0.jar" (ou 2.5.x) de Jython.org
- Instale-o:

```
1 java -jar jython_installer-2.5.0.jar
```

- Baixe e instale a partir "zxJDBC.jar" (94)
- Baixe e instale o arquivo do "sqlitejdbc-v056.jar" (95)
- Adicionar zxJDBC e sqlitejdbc ao CLASSPATH java
- web2py Comece com Jython

```
1 /path/to/jython web2py.py
```

No momento da escrita que apenas suportamos sqlite e postgres em Jython.

Componentes e Plugins

Componentes e plugins são características relativamente novo de web2py e há alguma discordância entre os desenvolvedores sobre o que é eo que deveria ser. A maior parte da confusão se origina do uso diferente destes termos em outros projetos de software e do fato de que os desenvolvedores ainda estão trabalhando para finalizar o caderno de encargos.

No entanto, eles são uma característica importante e que eu preciso para fornecer definições. Estes definição não pretende ser definitivo, mas apenas de acordo com os padrões de programação que queremos discutir neste capítulo.

Há duas questões que tentamos abordar aqui:

- Como podemos construir aplicações modulares que minimizam a carga do servidor e maximizar a reutilização de código?
- Como podemos distribuir pedaços de código em um plugin de moda-and-play, mais ou menos?

Endereço *components* primeira edição, endereço *plugins* no segundo.

14.1 Componentes

A **component** é uma parte funcionalmente autónoma de uma página web.

Um componente pode ser composto por módulos, controladores e vistas, mas não há exigência rigorosa do que outros, quando incorporado em uma página web, deve ser localizada dentro de uma tag HTML (por exemplo, um DIV, um SPAN, ou um IFRAME) e ele executa sua tarefa de forma independente do resto da página. Estamos especialmente interessados em componentes que são carregados na página e comunicar-se com a função de controlador componente via Ajax.

Um exemplo de um componente é uma "comments component" que está contida em uma DIV e mostra os comentários dos usuários e uma forma de post-comment-novo. Quando o formulário é enviado, ele é enviado para o servidor via Ajax, a lista é atualizada, o comentário do lado do servidor é armazenado no banco de dados. O conteúdo DIV é atualizada sem recarregar o resto da página.

A função LOAD web2py faz isto muito fácil de fazer sem o conhecimento explícito JavaScript Ajax / ou programação.

Nossa meta é ser capaz de desenvolver aplicações web de montagem de componentes em esquemas de página.

Considere um aplicativo simples web2py "test" that extends the default scaffolding app with a custom model in file "models/db_comments.py":

```
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
```

uma ação na "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())

```

eo "views/comments/post.html" correspondente

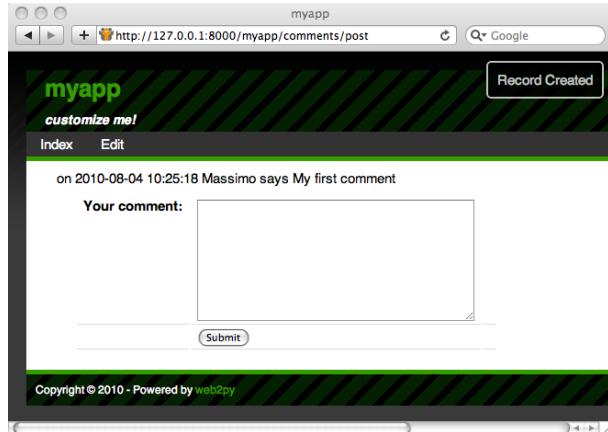
```

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}}

```

Você pode acessá-lo, como de costume em:

```
1 http://127.0.0.1:8000/test/comments/post
```



Até agora não há nada de especial nesta ação, mas podemos transformá-lo em um componente de simplesmente definir uma nova visão com ".load" extensão que não estender o layout.

Assim, criamos um "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}}
```

Nós podemos acessá-lo na URL

```
1 http://127.0.0.1:8000/test/comments/post.load
```

e ele será parecido com este:



Este é um componente e nós podemos inserir-lo em outra página simplesmente fazendo

```
1 {{=LOAD('comments','post.load',ajax=True)}}
```

Por exemplo, no "controllers/default.py" é possível editar

```
1 def index():
2     return dict()
```

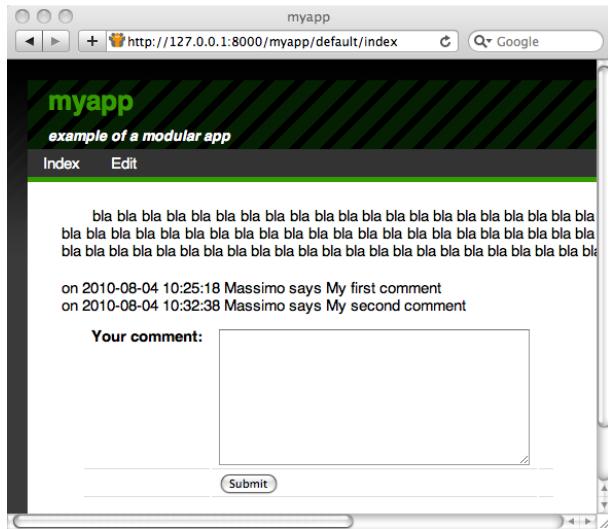
e na visão correspondente adicione o componente:

```
1 {{extend 'layout.html'}}
2 <p>{{='bla '*100}}</p>
3 {{=LOAD('comments','post.load',ajax=True)}}
```

Visitando a página

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

irá mostrar o conteúdo normal e os comentários de componentes:



O componente `{=LOAD(...)}` é processado como segue html:

```
1 <script type="text/javascript"><!--  
2 web2py_component("/test/comment/post.load","c282718984176")  
3 //--></script><div id="c282718984176">loading...</div>
```

(O verdadeiro código gerado depende das opções passadas para a função de carga).

A função é definida no `web2py_component(url,id)` e "web2py_ajax.html" executa toda a mágica: ele chama o `url` via Ajax e incorpora a resposta para a DIV com `id` correspondente; armadilhas que cada formulário de apresentação para o DIV e envia os formulários via Ajax. O alvo é sempre o Ajax DIV si.

A assinatura definitiva do ajudante de carga é a seguinte:

```
1 LOAD(c=None, f='index', args=[], vars={},
2     extension=None, target=None,
3     ajax=False, ajax_trap=False,
4     url=None):
```

Aqui:

- os dois primeiros argumentos `c` e `f` são o controlador ea função que deseja chamar, respectivamente.
- `args` e `vars` são os argumentos e as variáveis que queremos passar para a função. O primeiro é a lista, o último é um dicionário.
- `extension` é um extensão opcional. Observe que a extensão também pode ser passada uma parte da função como no `f='index.load'`.
- `target id` é o id do alvo DIV. Se não for especificado um `id` alvo aleatório é gerado.
- `ajax` deve ser ajustado para `True` se a DIV tem de ser preenchido através de Ajax e `False` se a DIV tem de ser preenchido antes de a página atual é voltar a ser (evitando assim a chamada Ajax).
- `ajax_trap=True` significa que qualquer envio de formulário no DIV deve ser capturado, apresentado via Ajax, ea resposta deve ser processado dentro do DIV. `ajax_trap=False` indica que os formulários devem ser apresentadas normalmente, assim recarregar a página inteira. `ajax_trap` é ignorada e considerada `True` `ajax=True` se.
- `url`, se especificada, substitui os valores do `c`, `f`, `args`, `vars`, `extension` e carrega o componente e no `url`. Ele é usado para carregar as páginas componentes servido por outras aplicações (que o meu ou não podem ser criadas com web2py).

Se não for especificado `.load` vista, há uma `generic.load` que torna dicionário retornado pela ação sem layout. Funciona melhor se o dicionário contém um único item.

Se você carregar um componente em conta a extensão `.load` e correspondente a função de controlador redireciona para outra ação (por exemplo, um

formulário de login), a extensão `.load` e propaga a nova URL (aquele para redirecionar demais) também é carregado com uma extensão de `.load`.

Quando a acção de um componente é chamada via Ajax, web2py passa dois cabeçalhos de HTTP com o pedido:

```
1 web2py-component-location
2 web2py-component-element
```

que pode ser acessada pela ação através das variáveis:

```
1 request.env.http_web2py_component_location
2 request.env.http_web2py_component_element
```

O primeiro contém a URL da página que chamou a ação do componente. Este último contém o `id` da DIV que conterá a resposta.

A ação componente também pode armazenar dados em duas especiais cabeçalhos de resposta HTTP que será interpretada pela página inteira na resposta. São eles:

```
1 web2py-component-flash
2 web2py-component-command
```

e que pode ser definido através de:

```
1 response.headers['web2py-component-flash']='...'
2 response.headers['web2py-component-command']='...'
```

O primeiro contém o texto que você quer ser brilhou na resposta. Este último contém o código JS que deseja ser executado na resposta. Ele não pode conter novas linhas.

Como exemplo, vamos definir um componente de formulário de contato na mensagem "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" e remover o componente da 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)
```

As primeiras quatro linhas definem a forma e a aceitá-la. O e-mail objeto usado para o envio é definido no padrão andaimes aplicação. Os últimos quatro linhas de implementar toda a lógica da componente específica, obtendo dados de cabeçalhos de solicitação HTTP e definindo os cabeçalhos de resposta HTTP.

Agora você pode inserir este formulário de contato em qualquer página via

```
1 {{=LOAD('contact','ask.load',ajax=True)}}
```

Repare que nós não definir uma visão para o nosso componente .load ask. Nós não temos porque ele retorna para um único objeto (forma) e, portanto, o "generic.load" vai fazer muito bem.

Se a exibição "generic.load" é usado você pode usar a sintaxe regular:

```
1 response.flash='...'
```

, e é equivalente a

```
1 response.headers['web2py-component-flash']='...'
```

14.2 Plugins

A **plugin** é um subconjunto dos arquivos de um aplicativo.

e que realmente significa *any*:

- Um plugin não é um módulo, não um modelo, não é um controlador, não é uma visão, ainda que possa conter módulos, modelos, controladores e / ou opiniões.
- O plugin não necessita de ser funcionalmente autônomo e pode depender de outros plugins ou código de usuário específico.
- A *plugin* não é uma *plugins system*, portanto, não tem noção de matrícula nem de isolamento, embora, vamos dar as regras para tentar conseguir algum tipo de isolamento.
- Aqui estamos falando de um plugin para seu aplicativo, e não um plugin para o web2py.

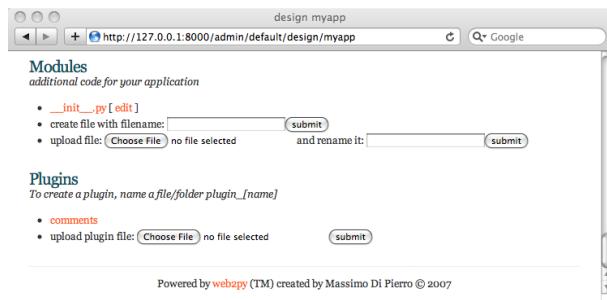
Então por que é chamado de *plugin*? Porque fornece um mecanismo para a embalagem de um subconjunto de um aplicativo e descompactá-lo em detrimento de outra aplicação (ou seja, *plug-in*). Nesta definição, qualquer arquivo em seu aplicativo pode ser tratado como plugin.

Quando o seu aplicativo distribuído seus plugins são embalados e distribuídos com ele.

Na prática, o **admin** oferece uma interface para embalar e desembalar plugins separadamente da sua app. Arquivos e pastas do seu aplicativo que têm nomes com o prefixo `plugin_name` podem ser embalados juntos em um arquivo chamado:

`web2py.plugin.name.w2p`

E distribuídos em conjunto.



Os arquivos que compõem um plugin não são tratadas por web2py diferente de qualquer outro arquivo, exceto que compreende **admin** de seus nomes que se destinam a ser distribuídos em conjunto e os exibe em uma página separada:



No entanto, como uma questão de fato, pela definição acima, estes plugins são mais gerais que as reconhecidas como tal pelo **admin**.

Na prática, só se preocupar com dois tipos de plugins:

- *Component Plugins.* Estes são os plugins que contêm componentes, tal como definido na seção anterior. Um plugin componente pode conter um ou mais componentes. Podemos pensar, por exemplo, de um `plugin_comments` que contém o componente `comments` acima propostos. Outro exemplo pode ser um `plugin_tagging` que contém um componente `tagging` e `tag-cloud` um componente que partilham algumas tabelas do banco de dados também definida pelo plugin.
- *Layout Plugins.* Estes são os plugins que contêm uma vista de esquema e os arquivos estáticos como exigido pelo layout. Quando o plug-in for aplicado, dá o app um novo olhar e sentir.

Pelas definições acima, os componentes criados na seção anterior, por exemplo, "controllers/contact.py", já são plugins. Nós podemos movê-los de um aplicativo para outro e usar os componentes que definem. No entanto, eles não são reconhecidos como tal pelo `admin`, pois não há nada que os rotula como plugins. Portanto, há dois problemas que precisamos resolver:

- Nome do plugin arquivos usando uma convenção, de modo que `admin` pode reconhecê-los como pertencentes ao mesmo plugin
- Se o plugin tem os arquivos de modelo, para estabelecer uma convenção que define os objetos que ele não poluir o namespace e não entrem em conflito uns com os outros.

Vamos supor que um plugin é chamado `name`. Aqui estão as regras que devem ser seguidas:

Rule 1: Plugin modelos e controladores devem ser chamados, respectivamente,

- `models/plugin_name.py`

- `controllers/plugin_name.py`

e pontos de vista plugin, módulos, estático e arquivos particulares devem estar em pastas denominadas, respectivamente:

- `views/plugin_name/`
- `modules/plugin_name/`
- `static/plugin_name/`
- `private/plugin_name/`

Rule 2: modelos plug-in só pode definir objetos com nomes que começam com

- `plugin_name`
- `PluginName`
- `_`

Rule 2: modelos plug-in só pode definir variáveis de sessão com nomes que começam com

- `session.plugin_name`
- `session.PluginName`

Rule 4: Plugins deve incluir licença e documentação. Estes devem ser coloados em:

- `static/plugin_name/license.html`
- `static/plugin_name/about.html`

Rule 5: O plugin só pode invocar a existência dos objetos global definido no andaime "db.py", ou seja,

- Uma conexão de dados denominado db
- uma instância chamada Auth auth
- uma instância chamada Crud crud
- uma instância chamada Service service Alguns plugins podem ser mais sofisticadas e ter um parâmetro de configuração no caso de um db exemplo mais existe.

Rule 6: Se um plugin necessidades parâmetros de configuração, estas devem ser definidas através de um PluginManager como descrito abaixo.

Seguindo as regras acima, podemos ter certeza de que:

- **admin** reconhece todos os arquivos e pastas `plugin_name` como parte de uma única entidade.
- plugins não interferem uns com os outros.

As regras acima não resolverem o problema das versões do plugin e dependências. Isso está além do nosso alcance.

14.2.1 Componente Plugins

plugins componentes são plugins que definem os componentes. Componentes normalmente acessar o banco de dados e definir com os seus próprios modelos.

Aqui voltamos a componente anterior `comments` em um `comments_plugin` usando o mesmo código exato que escreveu antes, mas vamos seguir todas as regras anteriores.

Primeiro, criamos um modelo chamado "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)
```

(observe as duas últimas linhas definir uma função que irá simplificar a incorporação do plugin)

Em segundo lugar, definir um "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())
```

Em terceiro lugar, vamos criar uma visão chamada "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}}
```

Agora podemos usar **admin** para embalar o plugin para distribuição. Admin irá salvar este plugin como:

```
1 web2py.plugin.comments.w2p
```

Podemos usar o plugin em qualquer ponto de vista simplesmente instalando o plug-in através da página na **edit admin** e somando isso aos nossos próprios pontos de vista

```
1 {{=plugin_comments()}}
```

Claro que nós podemos fazer o plugin mais sofisticado por ter componentes que precisam de parâmetros e opções de configuração. Quanto mais complexa dos componentes, mais difícil se torna para evitar colisões de nomes. O Plugin Manager descrito a seguir é projetado para evitar este problema.

14.2.2 *Plugin Manager*

O `PluginManager` é uma classe definida no `gluon.tools`. Antes de explicar como ele funciona dentro, vamos explicar como usá-lo.

Aqui vamos considerar o `comments_plugin` anterior e nós fazemo-lo melhor. Nós queremos ser capazes de personalizar:

```
1 db.plugin_comments_comment.body.label
```

sem ter que editar o código do plugin em si.

Aqui é como podemos fazê-lo:

Em primeiro lugar, reescrever o "models/plugin_comments.py" plugin da seguinte forma:

```
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 como todos os códigos, exceto a definição da tabela é encapsulado em uma única função. Além disso, observe como a função cria uma instância de um PluginManager.

Agora, em qualquer outro modelo em sua aplicação, por exemplo, "models/db.py", você pode configurar o plugin da seguinte forma:

```
1 from gluon.tools import PluginManager
2 plugins = PluginManager()
3 plugins.comments.body_label=T('Post a comment')
```

O objeto plugins já está instanciado no padrão andaime em app "models/db.py"

O objeto PluginManager é um nível de discussão, objeto de armazenamento de objetos singleton Storage. Isso significa que você pode instanciar quantos quiser dentro do mesmo aplicativo, mas (se eles têm o mesmo nome ou não) que agem como se não houvesse uma instância PluginManager único.

Em particular, cada arquivo plugin pode fazer seu próprio objeto PluginManager e registrar-se e seus parâmetros de default com ele:

```
1 plugins = PluginManager('name', param1='value', param2='value')
```

Você pode substituir esses parâmetros em outros lugares (por exemplo, "models/db.py") com o código:

```
1 plugins = PluginManager()
2 plugins.name.param1 = 'other value'
```

Você pode configurar vários plugins em um só lugar.

```
1 plugins = PluginManager()
2 plugins.name.param1 = '...'
3 plugins.name.param2 = '...'
4 plugins.name1.param3 = '...'
5 plugins.name2.param4 = '...'
6 plugins.name3.param5 = '...'
```

Lembre-se que quando o plugin é definido o PluginManager deve ter argumentos: o nome do plugin e opcional argumentos nomeados, que são os parâmetros padrão. Em vez disso, quando os plugins são configurados, o construtor deve PluginManager não têm argumentos. A configuração deve preceder a definição do plugin (isto é, deve estar em um arquivo modelo que vem em primeiro lugar por ordem alfabética).

14.2.3 plugins Layout

Layout plugins são mais simples de plugins componente porque geralmente não contêm código, mas apenas vistas e arquivos estáticos. No entanto, há algumas boas práticas que você deve seguir:

Primeiro, crie uma pasta chamada `static/plugin_layout_name":italic /` (onde nome é o nome do seu layout) e colocar todos os seus arquivos estáticos lá.

Em segundo lugar, criar um layout de arquivo chamado `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 /"

Em terceiro lugar, modificar o "views/layout.html" para que ele lê, simplesmente como:

```
1 {{include 'plugin_layout_name/layout.html'}}
```

A vantagem deste modelo é que os usuários desse plugin é possível instalar vários layouts e escolher qual deles se aplica simplesmente "views/layout.html". Moreover "views/layout.html" edição não será embalado pelo **admin**, juntamente com o plugin que não há risco de que o plugin substitui o código de usuário no anteriormente layout instalado.

14.3 plugin_wiki

AVISO: plugin_wiki ainda está muito em desenvolvimento e, portanto, não prometemos compatibilidade com versões anteriores para o mesmo nível de funções essenciais web2py.

plugin_wiki é um plugin de esteróides. O que queremos dizer é que define vários componentes úteis e que podem mudar a maneira de desenvolver as suas aplicações:

Você pode baixá-lo

¹ <http://web2py.com/examples/static/web2py.plugin.wiki.w2p>

A idéia por trás **plugin_wiki** é que a maioria dos aplicativos inclui páginas que são estáticos semi. Estas são páginas que não incluem lógica personalizada complexo. Eles contêm texto estruturado (pense em uma página de ajuda), imagens, áudio, vídeo, formas bruta, ou um conjunto de componentes padrão (comentários, tags, gráficos, mapas), etc Essas páginas podem ser públicos, exigem login, ou ter outros restrições a autorização. Essas páginas podem ser ligadas através de um menu ou apenas ser acessível através do formulário de assistente. **plugin_wiki** oferece uma maneira fácil de adicionar páginas que se encaixam nesta categoria para o seu regular web2py aplicações.

Em particular **plugin_wiki** proporciona:

- Uma interface wiki-like que permite adicionar páginas ao seu aplicativo e referenciá-los por uma lesma. Estes páginas (que chamarei de páginas wiki) têm versões e são armazenadas no banco de dados.
- Públicos e privados páginas (requer login). Se uma página de login requer também pode exigir que o usuário tem uma participação no grupo especial.

- Três níveis: 1,2,3. No nível 1 páginas só podem incluir textos, imagens, áudio e vídeo. No nível 2 páginas também pode incluir widgets (estes são os componentes, tal como definido na seção anterior, que pode ser incorporado em páginas wiki). No nível 3 páginas web2py também pode incluir código do modelo.
- A escolha de edição de páginas com a sintaxe markmin ou em HTML usando um editor WYSIWYG.
- Uma coleção de widgets: implementados como componentes são auto documentar e pode ser incorporado como componente regular em condições normais de opiniões web2py ou - usando uma sintaxe simplificada - em páginas wiki.
- Um conjunto de páginas especiais (`meta-code`, `meta-menu`, etc) que podem ser usadas para personalizar o plugin (por exemplo, definir o código do plugin deverá ser executado, personalizar o menu, etc)

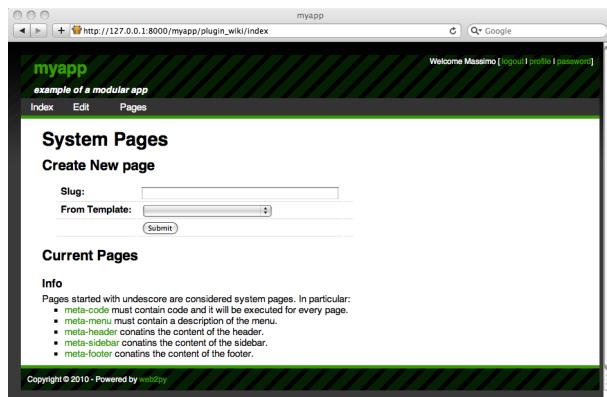
*O app **welcome** mais o **plugin_wiki** pode ser pensada como um ambiente de desenvolvimento em si, que é adequado para a construção de aplicações web simples, como um blog.*

De agora em diante, vamos assumir o **plugin_wiki** é aplicada a uma cópia do aplicativo andaimes **welcome**.

A primeira coisa que você observa após a instalação do plugin é que ele adiciona um novo item de menu chamado *pages*.

Clique no item de menu *pages*, e você será redirecionado para a ação plugin:

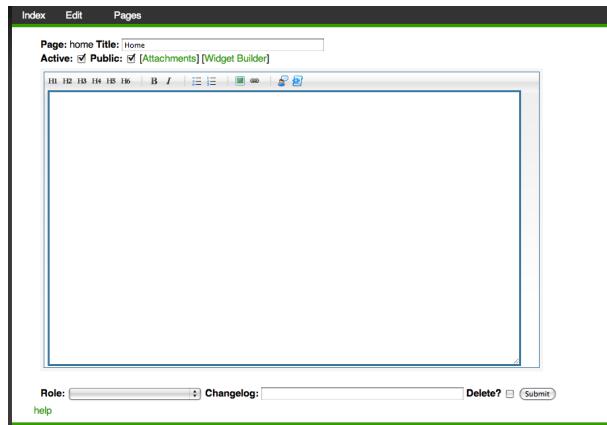
¹ http://127.0.0.1:8000/myapp/plugin_wiki/index



A página de índice plugin lista as páginas criadas utilizando o plugin em si e permite que você crie novos optando por um *slug*. Tente criar uma página home. Você será redirecionado para

http://127.0.0.1:8000/myapp/plugin_wiki/page/home

Clique no *create page* para editar seu conteúdo.



Por padrão, o plugin está no nível 3, que significa que você pode inserir widgets assim como o código nas páginas. Por padrão, ele usa a sintaxe markmin para descrever o conteúdo da página.

14.3.1 sintaxe Markmin

Aqui está uma cartilha para a sintaxe markmin:

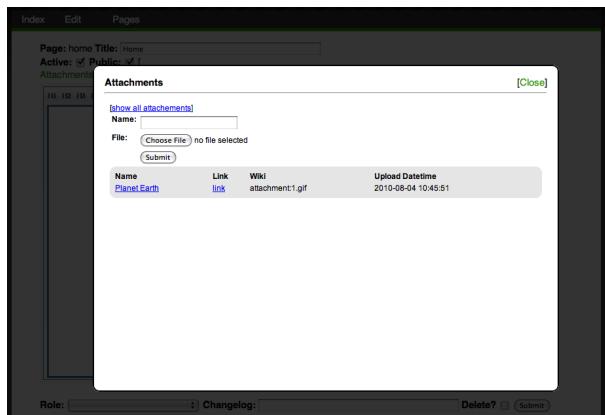
markmin	html
# title	<h1>title</h1>
## subtitle	<h2>subtitle</h2>
### subsubtitle	<h3>subsubtitle</h3>
bold	bold
"italic"	<i>italic</i>
http://...com	http:...com
[[name http://example.com]]	name
[[name http://...png left 200px]]	

Você pode adicionar link outras páginas

```
[[mylink name page:slug]]
```

E se a página não existe você será solicitado a criar uma:

A página de edição permite que você adicione anexos em páginas (ou seja, arquivos estáticos)



e você pode ligá-los como

```
1 [[mylink name attachment:3.png]]
```

ou incluí-los com

```
1 [[myimage attachment:3.png center 200px]]
```

O tamanho (200px) é opcional. center não é opcional, mas pode ser substituído por left ou right.

Você pode inserir texto com blockquoted

```
1 -----
2 this is blockquoted
3 -----
```

bem como quadros

```
1 -----
2 0 | 0 | X
3 0 | X | 0
4 X | 0 | 0
5 -----
```

e texto integral

```
1 ``
2 verbatim text
3 ``
```

Você também pode prefixar uma :class opcional para o --- ou “ final final. Para o texto blockquoted e tabelas que serão traduzidos na classe da marca, por exemplo:

```
1 -----
2 test
3 -----:abc
```

processa como

```
1 <blockquote class="abc">test</blockquote>
```

Para o texto integral da classe pode ser usado para inserir o conteúdo de diferentes tipos.

Você pode, por exemplo, o código embed com destaque de sintaxe, especificando o idioma com `:code_language`

```

1 ``
2 def index(): return 'hello world'
3 ``:code_python

```

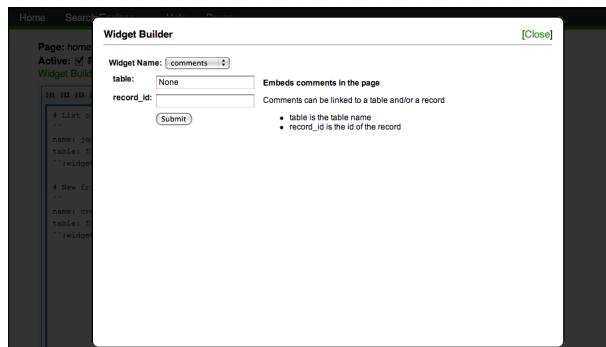
Você pode inserir widgets:

```

1 ``
2 name: widget_name
3 attribute1: value1
4 attribute2: value2
5 ``:widget

```

A partir da página de edição você pode clicar no "widget builder" widgets para inserir em uma lista, de forma interativa:



(Para uma lista de widget veja próxima subseção).

Você também pode incorporar web2py modelo de código de linguagem:

```

1 ``
2 {{for i in range(10):}}<h1>{{=i}}</h1>{{=pass}}
3 ``:template

```

14.3.2 Permissões Page

Ao editar uma página você pode encontrar os seguintes campos:

- padrão **active** (para True). Se uma página não está ativo, ele não será acessível para os visitantes (mesmo que pública).
- padrão **public** (para True). Se uma página é pública, podem ser acessados pelos visitantes sem registro dentro
- padrão **Role** (para nenhum). Se uma página tem um papel, a página pode ser acessada apenas pelos visitantes que são registrados em um são membros do grupo com o papel correspondente.

14.3.3 Páginas especiais

meta-menu contém o menu. Se esta página não existe web2py usa o `response.menu` definido no "models/menu.py". O conteúdo da página menu meta-substitui o menu. A sintaxe é a seguinte:

```
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
```

onde o recuo determinar a estrutura de submenu. Cada item é composto do texto do item de menu, seguida por um link. Um link pode ser `page:slug`. Um link pode ser `None` é o item de menu não vincular qualquer página. Espaços extras são ignorados.

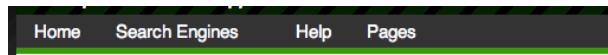
Aqui está outro exemplo:

```

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

```

Que processa da seguinte forma:



`meta-code` é uma outra página especial e deve conter código web2py. Esta é uma extensão do seu modelo e, na verdade, você pode colocar o código do modelo por aqui. Ele é executado quando o código "models/plugin_wiki.py" é executado.

Você pode definir tabelas `meta-code`.

Você pode, por exemplo, criar uma tabela simples amigos, colocando isso no `meta-code`:

```

1 db.define_table('friend', Field('name', requires=IS_NOT_EMPTY()))

```

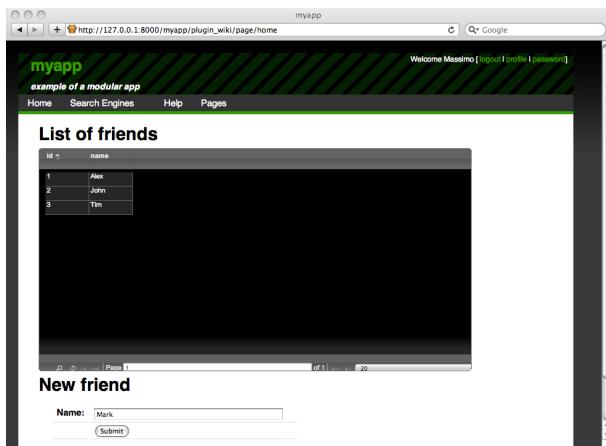
e você pode criar uma interface de gestão amigo página com a incorporação de uma página de sua escolha o seguinte código:

```

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

```

A página tem dois cabeçalhos (começando com #): "List of friends" and "New friend". A página contém dois widgets (sob os cabeçalhos correspondentes): um widget que jqgrid listas de amigos e criar um crud widget para adicionar um novo amigo.



`meta-header`, `meta-footer`, `meta-sidebar`, não são utilizados pelo layout padrão no "welcome/views/layout.html". If you want to use them edit "layout.html" **admin** usando (ou a casca) e coloque as seguintes marcas nos lugares apropriados:

```
1 {{=plugin_wiki.embed_page('meta-header') or ''}}
2 {{=plugin_wiki.embed_page('meta-sidebar') or ''}}
3 {{=plugin_wiki.embed_page('meta-footer') or ''}}
```

Desta forma, o conteúdo dessas páginas vai aparecer no cabeçalho e rodapé da barra lateral no layout.

14.3.4 Configurando plugin_wiki

Como com qualquer outros plugins no "models/db.py" você pode fazer

```
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'
```

onde

- **editor** é verdade se o atual usuário conectado está autorizado a editar páginas plugin_wiki
- **level** é o permissão: 1 - para editar páginas regular, 2 - para inserir widgets em páginas, 3 - inserir o código
- **mode** determina se deve usar um editor de "markmin" editor or a WYSIWYG "html".
- **theme** é o nome do requerido jQuery UI Theme. Por padrão apenas o "ui-darkness" cor neutra está instalado.

Você pode adicionar temas aqui:

```
1 static/plugin_wiki/ui/%(theme)s/jquery-ui-1.8.1.custom.css
```

14.3.5 Current Widgets

Cada elemento pode ser incorporado tanto em páginas plugin_wiki web2py normal e em modelos.

Por exemplo, para inserir um vídeo do YouTube em uma página plugin_wiki você pode fazer

```

1 ``
2 name: youtube
3 code: l7AWnffRc7g
4 ``:widget
```

ou para incorporar o widget mesmo em uma visão web2py você pode fazer:

```
1 {{=plugin_wiki.widget('youtube',code='l7AWnffRc7g')}}
```

Qualquer caso, a saída:



Widget argumentos que não têm um valor padrão são necessárias.

Aqui está uma lista de todos os widgets atual:

```
1 read(table,record_id=None)
```

Lê e exibe um registro

- `table` é o nome de uma tabela
- `record_id` é um número recorde

```
1 create(table,message='',next='',readonly_fields='',
2       hidden_fields='',default_fields '')
```

Exibe um formulário criar registro

- `table` é o nome de uma tabela
- `message` é uma mensagem a ser exibida após registro é criado
- `next` é onde "page/index/[id]" exemplo redirecionar
- `readonly_fields` é uma lista de campos separados por vírgula
- `hidden_fields` é uma lista de campos separados por vírgula
- `default_fields` é uma lista de valores separados por vírgulas "fieldname=value"

```
1 update(table,record_id='',message='',next='',
2        readonly_fields='',hidden_fields='',default_fields='')
```

Exibe um formulário de atualização registro

- `table` é o nome de uma tabela
- `record_id` é que ele registro a ser atualizado ou ((request.args = (-1)))
- `message` é uma mensagem a ser exibida após registro é criado
- `next` é onde "page/index/[id]" exemplo redirecionar
- `readonly_fields` é uma lista de campos separados por vírgula
- `hidden_fields` é uma lista de campos separados por vírgula
- `default_fields` é uma lista de valores separados por vírgulas "fieldname=value"

```
1 select(table,query_field='',query_value='',fields='')
```

Lista todos os registros na tabela

- `table` é o nome de uma tabela
- `query_field` e `query_value` se presentes irá filtrar registros de acordo com o `query_field == query_value` consulta
- `fields` é uma lista de campos separados por vírgulas a ser exibido

```
1 search(table,fields='')
```

A Widgets para selecionar registros

- `table` é o nome de uma tabela
- `fields` é uma lista de campos separados por vírgulas a ser exibido

```
1 jqgrid(table,fieldname=None,fieldvalue=None,col_widths=' ',  
2         _id=None,fields='',col_width=80,width=700,height=300)
```

Incorpora um plugin jqGrid

- `table` é o nome da tabela
- `fieldname`, `fieldvalue` é um filtro opcional (`fieldname == fieldValue`)
- `_id` é o "id" da DIV que contém o jqGrid
- `columns` é uma lista de nomes de colunas a ser exibido
- `col_width` é a largura de cada coluna (padrão)
- `height` é a altura do jqGrid
- `width` é a largura da jqGrid

```
1 latex(expression)
```

Google usa a API de gráficos embed LaTeX

```
1 pie_chart(data='1,2,3',names='a,b,c',width=300,height=150,align='center')
```

Incorporar um gráfico de pizza

- `data` é uma lista de valores separados por vírgula
- `names` é uma lista de etiquetas separadas por vírgula (uma para o item de dados)
- `width` é a largura da imagem
- `height` é a altura da imagem
- `align` determina o alinhamento da imagem

```
1 bar_chart(data='1,2,3',names='a,b,c',width=300,height=150,align='center')
```

Google usa API de gráficos para inserir um gráfico de barras

- `data` é uma lista de valores separados por vírgula
- `names` é uma lista de etiquetas separadas por vírgula (uma para o item de dados)
- `width` é a largura da imagem
- `height` é a altura da imagem
- `align` determina o alinhamento da imagem

```
1 slideshow(table,field='image',transition='fade',width=200,height=200)
```

Incorpora um slideshow. Ela recebe as imagens de uma tabela.

- `table` é o nome da tabela
- `field` é o campo de upload na tabela que contém imagens
- `transition` determina o tipo de transição, por exemplo, fade, etc
- `width` é a largura da imagem
- `height` é a altura da imagem

```
1 youtube(code,width=400,height=250)
```

Incorpora um vídeo do YouTube (por código)

- `code` é o código do vídeo
- `width` é a largura da imagem
- `height` é a altura da imagem

```
1 vimeo(code,width=400,height=250)
```

Incorpora um vídeo viemo (por código)

- `code` é o código do vídeo
- `width` é a largura da imagem
- `height` é a altura da imagem

```
1 mediaplayer(src,width=400,height=250)
```

Incorpora um arquivo de mídia (como vídeo ou um arquivo flash mp3)

- `src` é o `src` do vídeo
- `width` é a largura da imagem
- `height` é a altura da imagem

```
1 comments(table='None',record_id=None)
```

Incorpora comentários na página de comentários pode estar ligado a uma mesa e / ou um registro

- `table` é o nome da tabela
- `record_id` é o id do registro

```
1 tags(table='None',record_id=None)
```

Incorpora tags na página de tags podem ser ligados a uma mesa e / ou um registro

- `table` é o nome da tabela
- `record_id` é o id do registro

```
1 tag_cloud()
```

Incorpora uma nuvem de tags

```
1 map(key='....', table='auth_user', width=400, height=200)
```

incorpora um mapa do Google ganha pontos no mapa de uma tabela

- key é a api do google map-chave (padrão para obras 127.0.0.1)
- table é o nome da tabela
- width é a largura do mapa
- height é a 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)
```

Incorpora uma página em um <iframe></iframe>

```
1 load_url(src)
```

Carrega o conteúdo da url usando a função LOAD

```
1 load_action(action, controller='', ajax=True)
```

Carrega o teor de URL (request.application, controlador, ação) usando a função LOAD

14.3.6 Estendendo Widgets

Adicionando widgets para plugin_wiki pode ser feito através da criação de um novo modelo de arquivo chamado "models/plugin_wiki_" *name name* onde é arbitrária e que o arquivo que contém 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"
```

A primeira linha declara que você está estendendo a lista de widgets. Dentro da classe, você pode definir como a função que for necessário. Cada função estática é um novo widget, exceto as funções que começar por destacar. A função pode ter um número arbitrário de argumentos que podem ou não ter valores padrão. O docstring da função deve documentar a função utilizando a sintaxe markmin si.

Quando os elementos estão embutidos em páginas plugin_wiki, os argumentos serão passados para o widget como strings. Isto significa que a função widget deve ser capaz de aceitar cadeias de todos os argumentos e, eventualmente, transformá-la em uma representação necessária. Você pode decidir que a representação de seqüência de caracteres deve ser, apenas tenha certeza que isto está documentado no docstring.

O widget pode retornar uma string ou ajudantes web2py. Neste último caso, eles serão serializados usando .xml().

Observe como o novo widget pode acessar qualquer variável declarada no escopo global.

Como exemplo, vamos criar um novo widget que mostram o "contact/ask" form created at the beginning of this chapter. This can be done my creating a file "models/plugin_wiki_contact" que contém:

```

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* não são prestados por uma visão menos que a função `response.render(...)` é chamado explicitamente pelo widget.

Index

- `_init__.py`, 159
- `_and`, 349
- `_count`, 281
- `_dbname`, 243
- `_delete`, 281
- `_insert`, 281
- `_lastdb`, 255
- `_select`, 281
- `_update`, 282
- `_uri`, 243

- A, 208
- about, 134, 159
- accepts, 97, 111, 294
- Access Control, 357
- access restriction, 43
- Active Directory, 372
- admin, 85, 131, 470
- admin.py, 134
- Adobe Flash, 413
- Ajax, 118, 423, 504
- ALL, 260
- and, 266
- Apache, 445
- appadmin, 102, 142
- appliances, 85
- as_dict, 269

- B, 209
- bar chart, 533
- belongs, 280
- blob, 248
- block, 234
- BODY, 209
- bulk_insert, 253

- cache, 159, 160, 169
- cache clear, 171
- cache controller, 170
- cache select, 288
- cache view, 172
- cache.disk, 169
- cache.ram, 169
- CAPTCHA, 366

- as_list, 269, 400
- ASCII, 57
- ASP, 33
- asynchronous, 440
- Auth, 357
- auth.user, 362
- auth.user_id, 362
- Authentication, 392, 420
- autocomplete, 354
- autodelete, 316

- CAS, 392
- CENTER, 209
- CGL, 445
- checkbox, 213
- Cherokee, 462
- class, 69
- CLEANUP, 347
- CODE, 209
- command line, 146
- comments, 534
- commit, 253
- component, 503
- component plugin, 515
- compute, 271
- confirmation, 434
- connection pooling, 240
- connection strings, 239
- content-disposition, 313
- controllers, 159
- cookies, 179
- cooperation, 193
- count, 267
- cPickle, 79
- cron, 185
- cross site scripting, 40
- CRUD, 321, 527
- crud.create, 321, 322

crud.delete, 321, 322
crud.read, 322
crud.search, 321
crud.select, 321, 322
crud.tables, 321, 322
crud.update, 321, 322
CRYPT, 347
cryptographic store, 43
CSRF, 42
CSV, 404
csv, 282
custom validator, 350

DAC, 357
DAL, 161, 237, 243
DAL shortcuts, 261
database drivers, 237
databases, 159
date, 77, 279
datetime, 77, 279
day, 280
DB2, 243
def, 64, 205
default, 244
define_table, 238, 244
deletable, 435
delete, 267
delete_label, 308
dict, 60
dir, 55
dispatching, 148
distinct, 266
distributed transactions, 257
DIV, 211
Document Object Model (DOM), 207
Domino, 372
drop, 255
EDIT, 135

effects, 428
element, 223
elements, 223
elif, 66, 203
else, 66, 203, 204
EM, 211
email attachments, 377
email from GAE, 377
email html, 377
email logging, 376
emails, 235
encode, 57
env, 163
errors, 138, 159
escape, 201
eval, 74
examples, 85
except, 66, 204
Exception, 66
exclude, 269
exec, 74
exec_environment, 191
executesql, 254
export, 282
Expression, 239
extent, 226
ez.css, 228

Facebook, 363, 373
fake_migrate, 251
FastCGI, 459, 462
favicon, 182
fcgihandler, 459
fetch, 493
Field, 238, 244, 259
Field constructor, 245
fields, 248, 307
FIELDSET, 211
file.read, 71
file.seek, 72
file.write, 71

finally, 66, 204
find, 269
FireBird, 243
first, 269
flash mediaplayer, 534
Flickr, 363
for, 62, 202
FORM, 97, 211
form, 93, 294
format, 102
formname, 294
formstyle, 309

GAE login, 372
geocode, 493
GET, 150
Gmail, 371
Google, 363
Google App Engine, 482
Google map, 534
grouping, 277

H1, 212
HAProxy, 474
HEAD, 212
help, 55
helpers, 160, 206
hideerror, 333
hour, 280
HTML, 212, 286
HTTP, 160, 174
httpserver.log, 495

id_label, 308
if, 66, 203
IF_MODIFIED_SINCE, 150
IFRAME, 214
import, 75, 190, 282
improper error handling, 42

include, 226
 index, 86
 information leakage, 42
 Informix, 243
 inheritance, 291
 init, 180
 injection flaws, 41
 inner join, 275
 INPUT, 97, 213
 insecure object reference, 41
 insert, 252
 internationalization, 160, 176
 IS_ALPHANUMERIC, 335
 IS_DATE, 335
 IS_DATE_IN_RANGE, 336
 IS_DATETIME, 336
 IS_DATETIME_IN_RANGE, 336
 IS_DECIMAL_IN_RANGE, 337
 IS_EMAIL, 100, 337
 IS_EMPTY_OR, 347
 IS_EQUEL_TO, 337
 IS_EXPR, 337
 IS_FLOAT_IN_RANGE, 338
 IS_IMAGE, 344
 IS_IN_DB, 100, 349
 IS_IN_SET, 338
 IS_INT_IN_RANGE, 338
 IS_IPV4, 346
 IS_LENGTH, 339
 IS_LIST_OF, 340
 IS_LOWER, 340, 346
 IS_MATCH, 340
 IS_NOT_EMPTY, 97, 100, 341
 IS_NOT_IN_DB, 348
 IS_NULL_OR, 347
 IS_SLUG, 343
 IS_STRONG, 343
 IS_TIME, 341
 IS_UPLOAD_FILENAME,
 345
 IS_UPPER, 346
 IS_URL, 341
 Janrain, 363
 join, 275
 jqGrid, 527, 532
 JSON, 398, 417
 JSONRPC, 409
 JSP, 33
 Jython, 502
 keepvalues, 300
 KPAX, 131
 LABEL, 214
 labels, 307
 lambda, 72
 languages, 159
 last, 269
 latex, 532
 layout, 94
 layout plugin, 519
 layout.html, 226
 LDAP, 369, 371
 left outer join, 276
 LEGEND, 214
 length, 244
 LI, 214
 license, 45, 134, 159
 Lighttpd, 459
 like, 279
 limitby, 266
 LinkedIn, 363, 374
 list, 58
 LOAD, 504
 load, 504
 local_import, 190
 Lotus Notes, 372
 lower, 280
 MAC, 357
 mail.send, 376
 malicious file execution, 41
 many-to-many, 277
 markdown, 126
 MARKIN, 215
 markmin, 126
 markmin syntax, 523
 memcache, 480
 MENU, 221
 menu, 227
 menu and plugin_wiki,
 526
 Mercurial, 142
 META, 214
 meta, 227
 meta-header, 527
 meta-menu, 527
 meta-sidebar, 527
 migrate, 244
 migrations, 250
 minutes, 280
 mod_proxy, 445
 mod_python, 445
 mod_wsgi, 445
 Model-View-Controller, 35
 models, 159
 modules, 159
 month, 280
 MSSQL, 243
 MySpace, 363
 MySQL, 243
 named id field, 248
 nested select, 280
 not, 266
 notnull, 244

OAuth, 373
OBJECT, 215
OL, 215
ON, 215
onDelete, 244
one to many, 274
onvalidation, 296
OpenID, 363, 372
OpenLDAP, 372
OPTGROUP, 216
OPTION, 216
or, 266
Oracle, 243
orderby, 265
os, 76
os.path.join, 76
os.unlink, 76
outer join, 276

P, 216
page layout, 226
pagination, 494
PAM, 145, 369, 371
PARTIAL CONTENT, 150
password, 145
PDF, 419
PGP, 378
PHP, 33
pie chart, 532
PIL, 366
plugin, 503
plugin_wiki, 520
PluginManager, 515
POST, 150
postback, 95
PostgreSQL, 243
PRE, 216
prettydate, 493
private, 159
PyAMF, 413
Pyjamas, 409

PyRTF, 418
Python, 53
Query, 239, 259
radio, 213
random, 75
raw SQL, 281
RBAC, 357
reCAPTCHA, 366
recursive selects, 262
redirect, 95, 160, 174
referencing, 275
removing application, 481
ReportLab, 419
request, 32, 160, 161, 163
request.ajax, 151
request.application, 150
request.args, 111, 150
request.controller, 150
request.cookies, 161
request.function, 150
request.get_vars, 151
request.post_vars, 151
request.url, 150
request.vars, 93, 151
required, 244
requires, 97, 244
reserved Keywords, 242
response, 160, 165
response.body, 165
response.cookies, 165
response.download, 165
response.files, 165
response.flash, 111, 165
response.headers, 165
response.menu, 165, 227
response.meta, 165, 227
response.postprocessing,
 165
response.render, 165

response.status, 165
response.stream, 111, 165
response.subtitle, 165
response.title, 165
response.view, 165
response.write, 165, 201
return, 64, 205
robots, 182
rollback, 253
routes_in, 181
routes_on_error, 184
routes_out, 181
Row, 238, 259
Rows, 238, 259, 275, 276
RPC, 405
RSS, 118, 401
rss, 129
RTF, 418

sanitize, 126, 208
scaffolding, 85
scalability, 471
SCRIPT, 217
seconds, 280
secure communications, 43
security, 40, 470
SELECT, 217
select, 108, 259
selected, 216
session, 92, 159, 160, 168
session.connect, 168
session.forget, 168
session.secure, 168
Set, 239, 259
shell, 54
showid, 308
simplejson, 417
site, 131
slideshow, 533
SMS, 497
SMTP, 371

SOAP, 416
sort, 269
SPAN, 217
sql.log, 244
SQLFORM, 111
SQLite, 243
SQLTABLE, 263
static, 159
static files, 149
Storage, 161
str, 57
streaming, 501
STYLE, 217
sub-menu, 228
submit_button, 308
sum, 281
superfish, 228
sys, 77
sys.path, 77

T, 160, 176
TABLE, 218
Table, 249, 258
tables, 248
TAG, 221
tag could, 534
tags, 534
TBODY, 219

TD, 218
template language, 199
tests, 159
TEXTAREA, 219
TFOOT, 219
TH, 219
THEAD, 219
time, 77, 78, 279
TITLE, 220
TLS, 371
TR, 218, 220
truncate, 252
try, 66, 204
TT, 220
tuple, 59
type, 56, 244

UL, 220
Unicode, 57
unique, 244
update, 267
update_record, 268
upgrades, 491
upload, 98
uploadfield, 244
uploads, 159
uploadseparate, 244
upper, 280

URL, 95, 172
url mapping, 148
url rewrite, 181
UTF8, 57

validators, 161, 334
views, 159, 199
Vimeo, 533
virtualfields, 271

Web Services, 397
welcome, 85
while, 64, 203
widget in plugin_wiki, 520
wiki, 118, 520
Windows service, 469
WSGI, 194, 445
WYSIWYG, 529

x509, 378
XHTML, 212
XML, 207, 287, 398
XMLRPC, 118, 130, 407

year, 280
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/>