Sistemas Web Desconectados

Defossé Nahuel, van Haaster Diego Marcos

20 de abril de 2009

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

A Adrian Holovaty, Simon Willison, Jacob Kaplan-Moss y Wilson Miner por haber desarrollado un excelente framework simple y así mismo muy podroso para nuestro lenguaje preferido.

A Michael Trier por su mantenernos actualizados sobre lo que ocurre en la comunidad con su *podcast* **This Week in Django**

Resumen

Capítulo 1

Introducción

Capítulo 2

Aplicación del lado del servidor

2.1. El lenguaje de programación Python

El lenguaje de programación Python fue creado por Guido van Rossum en el año 1991. Python es un lenguaje de programación multiparadigma. Es decir, permite al programador utilizar diferentes formas de resolución de problemas

- programación orientada a objetos
- programación estructurada
- programación funcional

Otros paradigmas (como programación lógica) están soportados mediante el uso de extensiones [1] [2].

Python usa tipo de dato dinámico y reference counting para el manejo de memoria. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Nuevos módulos se pueden escribir fácilmente en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. Aunque el diseño de Python es de alguna manera hostil a la programación funcional tradicional del Lisp, existen bastantes analogías entre Python y los lenguajes minimalistas de la familia Lisp como puede ser Scheme.

Filosofía del lenguaje

Los usuarios de Python se refieren a menudo a la Filosofía Python que es bastante análoga a la filosofía de Unix. El código que sigue los principios de Python de legibilidad y transparencia se dice que es "pythonico". Contrariamente, el

[1] PySWIP http://code.google.com/p/pyswip/
[2] python-logic

código opaco u ofuscado es bautizado como "no pythonico"(ünpythonic.[en] inglés). Estos principios fueron famosamente descritos por el desarrollador de Python Tim Peters en *El Zen de Python*

1. Bello es mejor que feo.
2. Explícito es mejor que implícito.
3. Simple es mejor que complejo.
4. Complejo es mejor que complicado.
5. Plano es mejor que anidado.
6. Ralo es mejor que denso.
7. La legibilidad cuenta.
8. Los casos especiales no son tan especiales como para quebrantar las reglas.

   Aunque lo práctico gana a la pureza.
9. Los errores nunca deberían dejarse pasar silenciosamente.

   A menos que hayan sido silenciados explícitamente.
10. Frente a la ambigüedad, rechaza la tentación de adivinar.
11. Debería haber una -y preferiblemente sólo una- manera obvia de hacerlo.

    Aunque esa manera puede no ser obvia al principio a menos que usted sea Holandés
12. Ahora es mejor que nunca

    Aunque nunca es a menudo mejor que ya
13. Si la implementación es dificil de explicar, es una mala idea.
14. Si la implementacion es fácil de explicar, puede que sea una buena idea
15. Los espacios de nombres (namespaces) son una gran idea ¡Hagamos más de esas cosas!

Desde la versión 2.1.2, Python incluye estos puntos (en su versión original en inglés) como un huevo de pascua que se muestra al ejecutar import this.

2.1.1. Espacio de nombres en Python

Un módulo es un archivo con definiciones y sentencias en Python. En un módulo se define un conjunto de símbolos. Un símbolo puede ser

- Definición de una función
- Definición de una variable
- Definición de una expresión lambda
- Importación de otro módulo

Un módulo puede ser cargado desde el intérprete interactivo o desde otro módulo a través de la sentencia *import*

```
import os
```

Al ejecutrse esta sentencia, el intérprete buscará secuencialmente en el PYTHONPATH un módulo llamado os, en caso de encontrarlo, lo cargará y en el ámbito de nombres actual generará una referencia con el nombre "os". A esta actividad se la llama importación y el interprete garantiza que un módulo se cargará de unicamente una vez, en otras palabras, existe una única instancia de un módulo. En este caso, si varios módulos importan a "os", solo la primer ocurrencia realiza la importación efetiva. El resto de las sentencias import solamente generan referencias a la primera "instancia" de un módulo.

La sentencia import permite a su vez hacer una importación selectiva de símbolos de un módulo, por ejemplo:

```
from os import path
```

En este caso, import crea la instancia del módulo en la máquina virtual, pero no expone todo el contenido al ámbito de nombres local, sino que únicamente genera una referencia al símbolo *path*.

El comodín * permite importar todos los símbolos definidos en un módulo al espacio de nombres local.

```
from os import *
```

Está técnica debe ser utilizada con cuidado, debido a que "contamina" el ámbito de nombres local. Para mitigar este problma, puede definirse el símbolo __all__ al comienzo del módulo, con una tupla en la cual se definen los símbolos que se desean exportar. Por ejemplo:

```
__all__ = ('calcular_promedio', )

def calular_promedio(numeros):
    return suma (numeros) / len( numeros )
def suma(lista):
    return float( sum( lista ) )
```

2.1.2. Expresiones regulares en Python

Las expresiones reuglares proveen un medio conciso y flexible de identificar cadenas en un texto, como caracteres en particular, palabras o patrones de caracteres. Las expresiones regulares (a menudo abreviadas **regex**, o **regexp**, o en plural **regexes**, **regexps**, o incluso **regexen**) son escritas en un lenguaje formal que puede ser interpretado por un procesador de expresiones regulares. Las expresiones regulares se utilizan en muchos editores de texto, utilitarios[3] y lenguajes de programación.

POSIX Prel-like

El módulo *re* en python provee un motor de análisis de expresiones regulares. La sintaxis de definición de expresiones con captura nombrada de gurpos.

[3] Como las utilidades de consola UNIX *sed*, *grep*, etc.

2.2. Frameworks web basados en Python

Acá tenemos que justificar por que django

2.2.1. CGI

Interfaz de entrada común (en inglés Common Gateway Interface, abreviado CGI) es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web. CGI especifica un estándar para transferir datos entre el cliente y el programa. Es un mecanismo de comunicación entre el servidor web y una aplicación externa cuyo resultado final de la ejecución son objetos MIME[4]. Las aplicaciones que se ejecutan en el servidor reciben el nombre de CGIs.

Las aplicaciones CGI fueron una de las primeras maneras prácticas de crear contenido dinámico para las páginas web. En una aplicación CGI, el servidor web pasa las solicitudes del cliente a un programa externo. Este programa puede estar hecho en cualquier lenguaje que soporte el servidor, aunque por razones de portabilidad se suelen usar lenguajes de script. La salida de dicho programa es enviada al cliente en lugar del archivo estático tradicional.

CGI ha hecho posible la implementación de funciones nuevas y variadas en las páginas web, de tal manera que esta interfaz rápidamente se volvió un estándar, siendo implementada en todo tipo de servidores web.

Ejecución de CGI

A continuación se describe la forma de actuación de un CGI de forma esquemática:

1. En primera instancia, el servidor recibe una petición (el cliente ha activado un URL que contiene el CGI), y comprueba si se trata de una invocación de un CGI.

2. Posteriormente, el servidor prepara el entorno para ejecutar la aplicación. Esta información procede mayoritariamente del cliente.

3. Seguidamente, el servidor ejecuta la aplicación, capturando su salida estándar.

4. A continuación, la aplicación realiza su función: como consecuencia de su actividad se va generando un objeto MIME que la aplicación escribe en su salida estándar.

5. Finalmente, cuando la aplicación finaliza, el servidor envía la información producida, junto con información propia, al cliente, que se encontraba en estado de espera. Es responsabilidad de la aplicación anunciar el tipo de objeto MIME que se genera (campo CONTENT_TYPE), pero el servidor calculará el tamaño del objeto producido.

[4]Ver apéndice sobre MIMEA

Intercambio de información: Variable de Entorno

Variables de entorno que se intercambian de cliente a CGI:

QUERY_STRING Es la cadena de entrada del CGI cuando se utiliza el método GET sustituyendo algunos símbolos especiales por otros. Cada elemento se envía como una pareja Variable=Valor. Si se utiliza el método POST esta variable de entorno está vacía

CONTENT_TYPE Tipo MIME de los datos enviados al CGI mediante POST. Con GET está vacía. Un valor típico para esta variable es: Application/X-www-form-urlencoded

CONTENT_LENGTH Longitud en bytes de los datos enviados al CGI utilizando el método POST. Con GET está vacía

PATH_INFO Información adicional de la ruta (el "path") tal y como llega al servidor en el URL

REQUEST_METHOD Nombre del método (GET o POST) utilizado para invocar al CGI

SCRIPT_NAME Nombre del CGI invocado

SERVER_PORT Puerto por el que el servidor recibe la conexión

Variables de entorno que se intercambian de servidor a CGI:

SERVER_SOFTWARE Nombre y versión del software servidor de www

SERVER_NAME Nombre del servidor

GATEWAY_INTERFACE Nombre y versión de la interfaz de comunicación entre servidor y aplicaciones CGI/1.12

2.2.2. WSGI

The Web Server Gateway Interface defines a simple and universal interface between web servers and web applications or frameworks for the Python programming language. The latest version 3.0 of Python, released in December 2008, is already supported by $\mathrm{mod}_{wsgi}(a module for the Apache Webserver)$.

¿Por qué WSGI?

Historically Python web application frameworks have been a problem for new Python users because, generally speaking, the choice of web framework would limit the choice of usable web servers, and vice versa. Python applications were often designed for either CGI, FastCGI, mod_{python} or even custom API interfaces of specific web-servers.

WSGI[5] (sometimes pronounced 'whiskey' or 'wiz-gee') was created as a low-level interface between web servers and web applications or frameworks to promote common ground for portable web application development. WSGI is based on the existing CGI standard.

[5] PEP 333, Python Web Server Gateway Interface v1.0

Descripción general de la especificación

The WSGI has two sides: the "server.ºr "gateway"side, and the .ªpplication.ºr "framework"side. The server side invokes[clarification needed] a callable object (usually a function or a method) that is provided by the application side. Additionally WSGI provides middleware; WSGI middleware implements both sides of the API, so that it can be inserted "between.ª WSGI server and a WSGI application – the middleware will act as an application from the server's point of view, and as a server from the application's point of view.

Un módulo middleware puede realizar opreaciones como:

- Routing a request to different application objects based on the target URL, after changing the environment variables accordingly.
- Allowing multiple applications or frameworks to run side-by-side in the same process
- Load balancing and remote processing, by forwarding requests and responses over a network
- Perform content postprocessing, such as applying XSLT stylesheets

Aplicación de ejemplo

A WSGI compatible "Hello World.ªpplication in Python syntax: *continuar*

2.3. Django

Django es un framework web escrito en Python[6] el cual sigue vagamente el concepto de Modelo Vista Controlador. Ideado inicialmente como un adminsitrador de contenido para varios sitios de noticias, los desarrolladores encontraron que su CMS era lo sufcientemente genérico como para curbir un ámbito más aplio de aplicaciones. Fue liberado[7] bajo la licencia BSD en Julio del 2005 como Django Web Framework en honor a Django Reinhart. En junio del 2008 fue anuncidada la cereación de la Django Software Fundation, la cual se hace cargo hasta la fecha del desarrollo y mantenimiento.

Los orígenes de Django en la administración de páginas de noticias son evidentes en su diseño, ya que proporciona una serie de características que facilitan el desarrollo rápido de páginas orientadas a contenidos. Por ejemplo, en lugar de requerir que los desarrolladores escriban controladores y vistas para las áreas de administración de la página, Django proporciona una aplicación incorporada para administrar los contenidos (**django.contrib.admin**), que puede incluirse como parte de cualquier página hecha con Django y que puede administrar varias páginas hechas con Django a partir de una misma instalación; la aplicación administrativa permite la creación, actualización y eliminación de objetos de contenido, llevando un registro de todas las acciones realizadas sobre cada uno, y proporciona una interfaz para administrar los usuarios y los grupos de usuarios (incluyendo una asignación detallada de permisos).

[6] Ver apartado sobre el lengauje Python 2.1

[7] En el ámbito del software libre, la liberación es la fecha en la cual se pone a disposición de la comunidad del software en cuestión

La distribución principal de Django también aglutina aplicaciones que proporcionan un sistema de comentarios, herramientas para sindicar contenido via RSS y/o Atom, "páginas planas"que permiten gestionar páginas de contenido sin necesidad de escribir controladores o vistas para esas páginas, y un sistema de redirección de URLs.

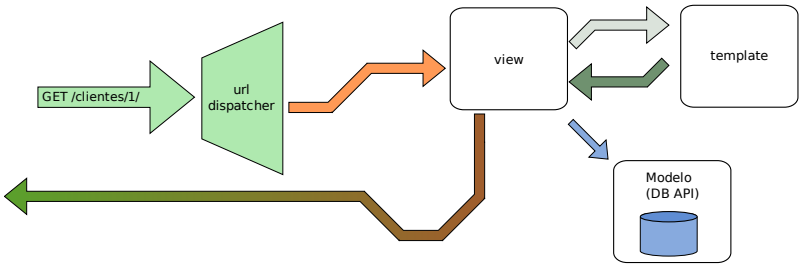

Figura 2.1: Estruccura báisca de Django

Django como framework de desarrollo consiste en un conjunto de utilidades de consola que permiten crear y manipular proyectos y aplicaciones.

Estructura de un proyecto

Un proyecto funciona como un contenedor de aplicaciones que ser rigen bajo la misma base de datos, los mismos templates y las mismas clases de middleware.

Una aplicación es un paquete que contiene al menos los módulos **models.py** y **views.py**, y generalmente suele agregarse un módulo **urls.py**.

Un proyecto Django consite en 3 módulos[8] básicos:

Módulo manage.py

Esta es la interfase con el framework. Este módulo permite crear aplicaciones, testear que los modelos de una aplicación estén bien definidos (validación), iniciar el servidor de desarrollo, crar volcados de la base de datos y restaurarlos restaurarlos (*fixtures*, utilizados en casos de pruebas y para precarga de datos conocidos).

Módulo settings.py

El módulo *settings* define la configuración transversal a las aplicaciones de usuario. En este módulo no se suelen definir más que constantes. Dentro de estas constantes encontramos la base de datos sobre la cual trabaja el ORM, el(los) directorio(s) de las plantillas, las clases middleware, ubicación de los medios estáticos[9]. En este módulo se defnien la lista de aplicaciones instaladas.

Módulo urls.py

Este módulo define las asociacines entre las URL y las funciones (vistas) que las atienden. Para generar código más modular, django permite delegar urls que

[8] Un módulo en Python, es un archivo con extensión .py

[9] Un medio estático es todo contenido que no se genera dinámicamente, como imágenes, liberías de javascript, contenido para embeber como archivos multimedia u elementos

cumplan con cierto patrón a un otro módulo. Típicamente este módulo se llama también urls y es parte de una aplicación (Ej: tratar todo lo lo que comience con **/clientes/** con el módulo **mi_proyecto.mi_aplicacion.urls**).

Una *expresión regular* es una forma de definir un patrón en una cadena. Mediante ésta técnica se realizan validaciones y búsquedas de elementos en cadenas. En python se define además una forma de otorgarle un alias a los elementos buscaods (en contraposición a la forma tradicional que utiliza un índice numérico). Esta particularidad de las expresiones regulares ha sido explotada para la asociación de las URLs a las funciones que las atienden. Cuando el cliente realiza accede a una URL dentro de un proyecto django, esta es checkeada contra cada patrón definido como url, en caso de éxito, se ejecuta la función asociada o vista. Si la expresión regualar tiene definido gurpos nombrados, cada subcadena pasa a ser argumento

pasan a ser argumentos de la vista, es decir, cada grupo nombrado, pasa a ser argumento de la función asociada.

Elementos de una aplicación Django

Una aplicación consiste en 2 módulos fundamentales.

Módulo models.py

En este módulo se definien los modelos.

Módulo views.py

En este módulo se definen las viastas. Una vista es una función que recibe como primer argumento un objeto HttpRequest[10], el cual encapsula la información preveniente del request, como el método (GET, POST), los elementos de la query http.

2.4. Aplicación del lado del cliente

2.5. Modelo de ejecución en el cliente

2.6. Liberías de Javascript

Breve explicación de la existencia de tantas librerías de javascript, necesidad de utilización de estas librerías para consistencia, modularidad y portabilidad del código

2.6.1. Elementos de una librería de javascript

Manejo unificado de DOM

Problemas con las implementaciones de DOM de cada navegador, sacar material de la charla que dio Jhon Reisig (está en Google Techtalks)

[10] Documentación oficial sobre HttpRequest en djangoproject.com

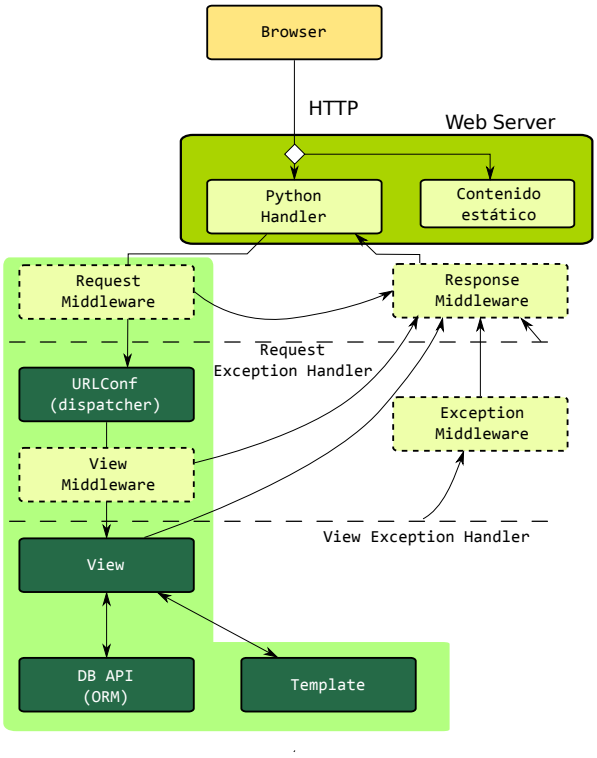

Figura 2.2: Esquema de flujo de una aplicación Django

Enfoque funcinal (encadenamiento)

Tipos de datos

Eventos

Ajax

Widgets

2.7. Ejemplos de librerías

2.7.1. jQuery

2.7.2. Prototype

2.7.3. YUI (Yahoo User Interface)

2.8. Javascript 1.7

Explicación de Javascript 2.0.

2.8.1. Protopy

Javascript 1.7 + modularidad + ejecución en contexto

Capítulo 3

Aplicación web desconectada

Doff, Django implementado sobre protopy.

3.0.2. Problemas de la traducción del framework a Javascript

3.0.3. Soporte en Django

Apéndice A

MIME

MIME (***M**ultipurpose **I**nternet **M**ail **E**xtensions*), (Extensiones de Correo de Internet Multipropósito), son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos. En sentido general las extensiones de MIME van encaminadas a soportar:

- texto en conjuntos de caracteres distintos de US-ASCII
- adjuntos que no son de tipo texto
- cuerpos de mensajes con múltiples partes (multi-part)
- información de encabezados con conjuntos de caracteres distintos de ASCII.

Prácticamente todos los mensajes de correo electrónico escritos por personas en Internet y una proporción considerable de estos mensajes generados automáticamente son transmitidos en formato MIME a través de SMTP. Los mensajes de correo electrónico en Internet están tan cercanamente asociados con el SMTP y MIME que usualmente se les llama mensaje SMTP/MIME.[1]

En 1991 la IETF (Internet Engineering Task Force) comenzó a desarrollar esta norma y desde 1994 todas las extensiones MIME están especificadas de forma detallada en diversos documentos oficiales disponibles en Internet.

MIME está especificado en seis RFCs (acrónimo inglés de Request For Comments) : RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 y RFC 2077.

Los tipos de contenido definidos por el estándar MIME tienen gran importancia también fuera del contexto de los mensajes electrónicos. Ejemplo de esto son algunos protocolos de red tales como HTTP de la Web. HTTP requiere que los datos sean transmitidos en un contexto de mensajes tipo e-mail aunque los datos pueden no ser un e-mail propiamente dicho.

En la actualidad ningún programa de correo electrónico o navegador de Internet puede considerarse completo si no acepta MIME en sus diferentes facetas (texto y formatos de archivo).

A.1. Introducción

El protocolo básico de transmisión de mensajes electrónicos de Internet soporta solo caracteres ASCII de 7 bit (véase también 8BITMIME). Esto limita los mensajes de correo electrónico, ya que incluyen solo caracteres suficientes para escribir en un número reducido de lenguajes, principalmente Inglés. Otros lenguajes basados en el Alfabeto latino es adicionalmente un componente fundamental en protocolos de comunicación como HTTP, el que requiere que los datos sean transmitidos como un e-mail aunque los datos pueden no ser un e-mail propiamente dicho. Los clientes de correo y los servidores de correo convierten automáticamente desde y a formato MIME cuando envían o reciben (SMTP/MIME) e-mails.

Encabezados MIME

MIME-Version

La presencia de este encabezado indica que el mensaje utiliza el formato MIME. Su valor es típicamente igual a "1.0"por lo que este encabezado aparece como:

```
MIME-Version: 1.0
```

Debe señalarse que los implementadores han intentado cambiar el número de versión en el pasado y el cambio ha tenido resultados imprevistos. En una reunión de IETF realizada en Julio 2007 se decidió mantener el número de versión en *"1.0"* aunque se han realizado muchas actualizaciones a la versión de MIME.

Content-Type

Este encabezado indica el tipo de medio que representa el contenido del mensaje, consiste en un tipo: type y un subtipo: subtype, por ejemplo:

```
Content-Type: text/plain
```

A través del uso del tipo multiparte (multipart), MIME da la posibilidad de crear mensajes que tengan partes y subpartes organizadas en una estructura arbórea en la que los nodos hoja pueden ser cualquier tipo de contenido no multiparte y los nodos que no son hojas pueden ser de cualquiera de las variedades de tipos multiparte. Este mecanismo soporta:

- mensajes de texto plano usando text/plain (este es el valor implícito para el encabezado Çontent-type:")
-
- texto más archivos adjuntos (multipart/mixed con una parte text/plain y otras partes que no son de texto, por ejemplo: application/pdf para documentos pdf, application/vnd.oasis.opendocument.text para OpenDocument text). Un mensaje MIME que incluye un archivo adjunto generalmente indica el nombre original del archivo con un encabezado Çontent-disposition:.º por un atributo name de Content-Type, por lo que el tipo o

formato del archivo se indica usando tanto el encabezado MIME content-type y la extensión del archivo (usualmente dependiente del SO).

```
Content-Type: application/vnd.oasis.opendocument.text;
    name="Carta.odt"
Content-Disposition: inline;
    filename="Carta.odt"
```

- reenviar con el mensaje original adjunto (multipart/mixed con una parte text/plain y el mensaje original como una parte message/rfc822)
- contenido alternativo, un mensaje que contiene el texto tanto en texto plano como en otro formato, usualmente HTML (multipart/alternative con el mismo contenido en forma de text/plain y text/html)
- muchas otras construcciones de mensaje

Apéndice B

Referencia Protopy

B.1. Módulos

B.1.1. event

event.connect Concetar

B.2. Plataforma Mozilla

- Porque desarrollaron e implementaron Javascript 1.7
- Porque javascript 1.7 tomo semántica (y sintaxis???) de Python
- Porque es código abierto
- Porque es extensible mediante plugins
  - Tiene firebug
  - Gears y firebug = muy compardor para el desarrollador.
-

Protopy persigue acercar la semántica del Javascript 1.7 a la del lenguaje Python. Las funcionalidades principales son las siguientes:

- Ámbito de nombres
- Semántica de objetos, dentro de la cual se hace una adaptación de
  - Iteradores
  - Generadores
- Iteradores
- Generadores
- Tipos básicos de la librería estándar de python, entre los que se encuentran: *bool*,

type Type sirve para defninir clases.

Bibliografía

[1] Versiones de Javascript, Jhon Reisig