Sistemas Web Desconectados

Defossé Nahuel, van Haaster Diego Marcos

10 de agosto de 2009

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

A nuestros familiares

Capítulo 1

Introducción

> Yo sólo puedo mostrarte la puerta. Tú eres quien debe atravesarla.
>
> Morfeo

1.1. Motivación

Hoy más que ayer, pero seguremente menos que mañana, Internet es "la red de redes". El alto contenido de información Hoy en día Internet supone más que un medio para obtener información, su constante expanción a convetido a esta red en un terreno muy atractivo para la implementación de sistemas de información.

Las aplicaciones web son populares debido a lo práctico que resulta el navegador web como cliente de acceso a las mismas. También resulta fácil actualizar y mantener aplicaciones web sin distribuir e instalar software a miles de usuarios potenciales. En la actualidad, existe una gran oferta de frameworks web para facilitar el desarrollo de aplicaciones web. Una ventaja significativa de las aplicaciones web es que funcionan independientemente de la versión del sistema operativo instalado en el cliente. En vez de crear clientes para los múltiples sistemas operativos, la aplicación web se escribe una vez y se ejecuta igual en todas partes. Las aplicaciones web tienen ciertas limitaciones en las funcionalidades que ofrecen al usuario. Hay funcionalidades comunes en las aplicaciones de escritorio, como dibujar en la pantalla o arrastrar y soltar, que no están soportadas por las tecnologías web estándar. Los desarrolladores web, generalmente, utilizan lenguajes interpretados o script en el lado del cliente para añadir más funcionalidades, especialmente para ofrecer una experiencia interactiva que no requiera recargar la página cada vez.Recientemente se han desarrollado tecnologías para coordinar estos lenguajes con tecnologías en el lado del servidor. Los sistemas operativos actuales de propósito general cuentan con un navegador web, con posibilidades de acceso a bases de datos y almacenamiento de código y recursos. La web, en el ámbito del software, es un medio singular por su ubicuidad y sus estándares abiertos. El conjunto de normas que rigen la forma en que se generan y transmiten los documentos a través de la web son regulados por

la W3C (Consorcio World Wide Web). La mayor parte de la web está soportada sobre sistemas operativos y software de servidor que se rigen bajo licencias OpenSource1 (Apache, BIND, Linux, OpenBSD, FreeBSD). Los lenguajes con los que son desarrolladas las aplicaciones web son generalmente OpenSource, como e PHP, Python, Ruby, Perl y Java. Los frameworks web escritos sobre estos lenguajes utilizan alguna licencia OpenSource para su distribución; incluso frameworks basados en lenguajes propietarios son liberados bajo licencias OpenSource.

1.2. Objetivos

Podemos decir que las aplicaciones tradicionales, que no hacen uso de la web, son más robustas ya que no dependen de una conexión. Por lo tanto, sería deseable poder dotar a las aplicaciones web de la capacidad de trabajar cuando no cuentan con conexión. Si bien los elementos necesarios para llevar a cabo esta tarea están disponibles actualmente, no están contemplados en los diseños de los frameworks web. Es decir, cuando una determinada aplicación web debe ser transportada al cliente, es necesario escribir el código de soporte específico para esa aplicación. Un framework no constituye un producto per sé, sino una plataforma sobre la cual construir aplicaciones. Consideramos que sería beneficioso aportar una extensión a un framework web OpenSource que brinde facilidades para transportar las aplicaciones web, basadas en éste, al cliente de manera que la aplicación que haga uso de nuestra extensión pueda ser ejecutada a posteriori en el navegador en el cual ha sido descargada. El framework web será elegido tras un estudio de las características que consideramos más importantes para el desarrollo veloz, como la calidad del mapeador de objetos (entre las características más importantes de éste buscaremos eficiencia en las consultas a la base de datos, ejecución demorada para encadenamiento de consultas, implementación de herencia, baja carga de configuración), la simplicidad para enlazar url's a funciones controladoras, extensibilidad del sistema de escritura de plantillas. Buscaremos frameworks que permitan la ejecución transversal de cierto tipo de funciones, para ejecutar tareas como compresión de salida, sustitución de patrones de texto, caché, control de acceso, etc.

La World Wide Web, o *web*, durante los últimos años ha ganado terreno como plataforma para aplicaciones del variado tipo. Diversas tecnologías fueron formuladas para convertir el escenario inical, donde la web se limitaba a ser una gran colección de documentos enlazados (*hipertexto*), para llegar a ser...

Vamos a realizar un breve análisis sobre las tecnologías que son utilizadas en la web.

Luego un análisis de las tecnologías del cliente, haciendo hincapié en ...

http://es.wikiquote.org/wiki/The_Matrix

1.3. Alcance

Aca ponemos hasta donde nos vamos a llegar.

Capítulo 2

Servidores Web

2.1. Recursos

Acá hablamos brevemente de Mozilla (cuentito de como evuoluciona Javascript), Javascript y como se perfila como estanadar de interconexión de sistemas (mashups), Python y CGI).—

2.2. Lenguajes de programación para la web

CGI es una tecnología que permite a un navegador web solicitar datos de un programa ejecutado en un servidor web. CGI establece un estandard para la transferencia de datos entre el cliente web y el servidor. El resultado de la ejecución de un CGI es un objeto MIME.

Un programa CGI (*o simmplemente CGI*) puede estar implementado en un lenguaje compilado o escrito como script de algún intérprete.

2.3. El lenguaje 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).

[1] PySWIP http://code.google.com/p/pyswip/

[2] python-logic

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 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
```

2.3.1. Desarrollo del lengauje

2.3.2. Tipos de datos y funciones

Python cuenta con los siguientes tipos de datos incorporados: *int*, *str*, *bool*, *float*, *complex*. Estos tipos de datos son denominados **inmutables**, es decir, para que su valor cambie, la instancia anterior es destruida.

Python no cuenta con el tipo de dato arreglo, sin embargo cuenta con 3 sustitutos. El tipo de dato tupla (*tuple()*) es un tipo de dato inmutable, que agrupa de manera *ordenada* un conjuntod de objetos del mismo o diferente tipo. El concepto de orden de los elementos es importante, ya que python tampoco provee la estrutura *for* tradicional de lenguajes procedurales como C o Pascal. Una tupla puede ser considreada como un arreglo fijo de valores que pueden tener diferentes tipos. Las tuplas pueden ser accedidas secuencialmente (*iteradas*) en orden ascendente o descendente, se puede acceder al n-ésimo item, pero no puede ser modificadas se generará una excepción debido a que es un tipo de datos estático.

```
a = (1, 2, 3) # Una tupla de enteros
b = (1, 1.0, 'pasta') # Una tupla con un conjuto dispar
                      # de elementos
c = ( (1, 2), (3, 4), (5, 6)) # Una tupla que tiene
                              # por elmentos otras tuplas
```

Python soprta asinganción a con la sintaxis de tupla, es decir, al colocar un conjunto de nombres separados por comas en el lado derecho de una asingación y un conjunto de valores o variables sobre la izquerda, la asingación es posicional.

Listing 2.1: Asignacion en Python

```
a, b = 1, 2     # Es equivalente a (a, b) = (1, 2)
a, b = b, a     # Operaci n de intercambio sin auxiliar
c = (1, 2, 3, 'bacalao')
w, x, y, z = c  # Asignaci n posicional
```

Una lista es una coección ordenada de objetos del mismo o diferente tipo que pueden ser modificados. Básicamente la lista se comporta como una tupla mutable.

```
 a = [1, 2, 3, 4 ]
 b[0] = 4
```

Los *bloques* en Python son conjuntos de sentencias que se encuentran a bajo un cierto nivel de inetación. Un bloque comienza con dos puntos (:) y termina cuando se pierde el nivel de indetación que lo definía. Veremos ejemplos de bloques con las sentencias de control de flujo.

Bloques y estructuras de control

La esentencia *if* ejecuta el

```
if condicion:
    bloque_si_condicion_verdadera
else:
    bloque_si_codicion_falsa
```

2.3.3. Espacio de nombres en Python

Cuando comenzamos a ejecutar el intérprete de Python de forma interactiva o cuando lo invocamos para ejecutar un script, están diponibles un conjuto de funciones, clases, clases de excepciones. El conjunto de estos nombres se conoce como ámbito de nombres __builtin__ (*incorporado*) y en este encontramos: int(), char(), str(), object, Exception, TypeError, ValueError, True, False, range, map, zip, locals, globals, entre otras. Pueden verse en el interprete interactivo mediante

```
>>> dir(__builtins__)
```

«“‘¡local
»”’Un *ámbito de nombres* en Python es un mapeo de nombres a objetos. La mayoría de los ámbitos de nombres están implementados mediante diccionarios, but that's normally not noticeable in any way (except for performance), y podría cambiar en el futuro. Otros ejemplos de ámbitos de nombre son los nombres globales en un módulo; los valores locales en la invocación de una función y en cierta forma, ======= Un *ámbito de nombres* en Python es un mapeo de nombres a objetos. La mayoría de los ámbitos de nombres están implementados mediante diccionarios, but that's normally not noticeable in any way (except for performance), y podría cambiar en el futuro. Otros ejemplos de ámbitos de nombre son los nombres globales en un módulo; los valores locales en la invocación de una función y en cierta forma, ’”»¿other los atributos de un objeto también forman un ámbito de nombres. Es importante tener en cuenta que no existe ninguna relación entre nombres de diferentes ámbitos de nobres.

Un *módulo* es un archivo con definiciones y sentencias en Python. El conjunto de todas las sentencias que no están indentadas, forman el ámbito de nombres global del modulo, las funciones, variables y clases que están suelen decirse que estan a “nivel módulo”. Los elementos que pueden ser parete del ámbito de nombres del módulo son:

- una definición de una función
- una vairable
- una expresión lambda
- el nombre de otro módulo importado

Un número, el llamado a función o el lanzado de una excepción no alteran el ámbito de nombres. Un módulo puede ser cargado desde el intérprete interactivo en el ámbito de nombres global o desde otro módulo (en su propio espacio de nombres) a través de la sentencia *import*

```
import os
```

En este caso, 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 sola 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:

```
1 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.

```
1 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 una variable **__all__** al comienzo del archivo, con una tupla en la cual se definen los símbolos que se desean exportar. Suponendo que el siguiente código se encuentra en el módulo promedio:

```
1 __all__ = ('calcular_promedio', )
2
3 def calular_promedio(numeros):
4     return suma (numeros) / len( numeros )
5 def suma(lista):
6     return float( sum( lista ) )
```

Al realizar

```
1  from promedio import *
```

El único símbolo que encontraremos en nuestro espacio de nombres será, *calcular_promedio*.

2.3.4. Sobrecarga de operadores

2.3.5. Funciones en Python

En python, una función se define de la sieguiente manera:

```
1 def funcion(argumento1, argumento2):
2   ''' Doc de la funcion '''
3   print argumento1, argumento2
```

Python soporta lo que se ha dadon en llamar *empaquetado de argumentos* tanto en la definición como en la invocación de una función.

El empaquetado más simple es el del tipo lista, que consiste en la hablidad de una función de recibir argumentos variables en formato lista. Para usar esta caractrística es necesrio utilizar una signatura especial (agregando un asterisco).

```
def promedio(*lista):
  ''' Calcula promedio '''
  return sum(lista) / float(len(lista)) # utilizando
                                        # varios builtins

# Ejemplo de utilizacion
>>> promedio(1, 2, 4, 5, 5)
3.3999999999999999
```

Esta signatura de argumentos se puede combinar con argumentos posicionales.

```
def dividir_suma_por( numero, *lista ):
  return sum(lista) / numero
```

La capiacidad de recibir listas puede ser utilizada a la inversa. Es decir, conociendo que una función recibe una cantidad de argumentos, "acomodar" en una lista los argumentos que se desean utilizar.

```
# Utilizanodo la funcion promedio anterior
>>> lista = [1, 2, 4, 6, 7]
>>> promedio( *lista )
4.0
```

2.3.6. Clases en Python

En en lengauje Python, todos los tipos de datos son objetos. No existe el concepto de tipo de dato primitivo *(como en java)*. Las clases definidas por el usuario son a su vez objetos. A partir de la versión 2.2 del lenguaje, aparecen new-style-classes que permiten utilizar descriptores (base para implemetnación de propiedades) y metaclases. «"'¡local Una metaclase es una clase que define la estructura de otra clase y es una forma dinámica de definir la estructura de una clase. Esta característica es considerada compleja, pero es utilizada en varios componentes de Django (como la generación de formularios a partir de la definición de un modelo o la generación de interfases de CRUD en la administración integrada). ======= Una metaclase es una clase que define la estructura de otra clase. Esta característica es a veces considerada compleja[3], pero es utilizada en varios componentes de Django (como la generación de formularios a partir de la definición de un modelo o la generación de interfases de CRUD en la administración integrada). '"»¿other

»"' [3]Metaclasses: a solution looking for a problem?

Instanciacion de una clase

En Python no existe el conceto de constructor como en los lenguajes C++, Objective-C o Java. La instanciación de una clase tiene dos fases: una fase de creación estructural y otra de inicialización. Estos métodos utilizan los nombres reservados __**new**__ e __**init**__.

*necesitamos saber que son *largs y **kwargs*

2.3.7. 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[4] 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.

2.4. Frameworks web

Acá tenemos que justificar por que django

2.4.1. 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[5]. 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:

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

[5] Ver apéndice sobre MIMEC

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.

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

2.4.2. 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 mod_wsgi (a module for the Apache Web server).

¿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[6] (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.

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*

2.5. Django

Django es un framework web escrito en Python[7] 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

[6] PEP 333, Python Web Server Gateway Interface v1.0
[7] Ver apartado sobre el lengauje Python 2.3

que su CMS era lo sufcientemente genérico como para curbir un ámbito más aplio de aplicaciones. Fue liberado[8] 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).

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.

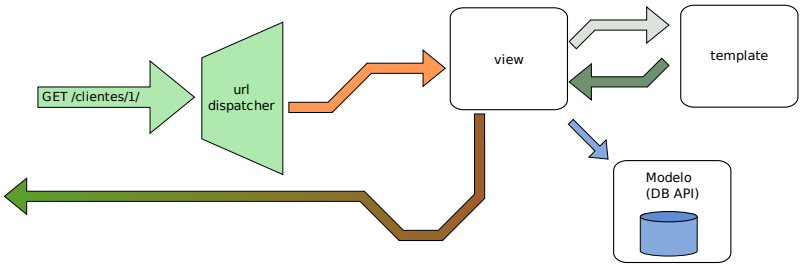

Figura 2.1: Estruccura báisca de Django

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

plantilla. 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[9] básicos:

[8] 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

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

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[10]. 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 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[11], el cual encapsula la informa-

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

[11] Documentación oficial sobre HttpRequest en djangoproject.com

ción preveniente del request, como el método (GET, POST), los elementos de la query http.

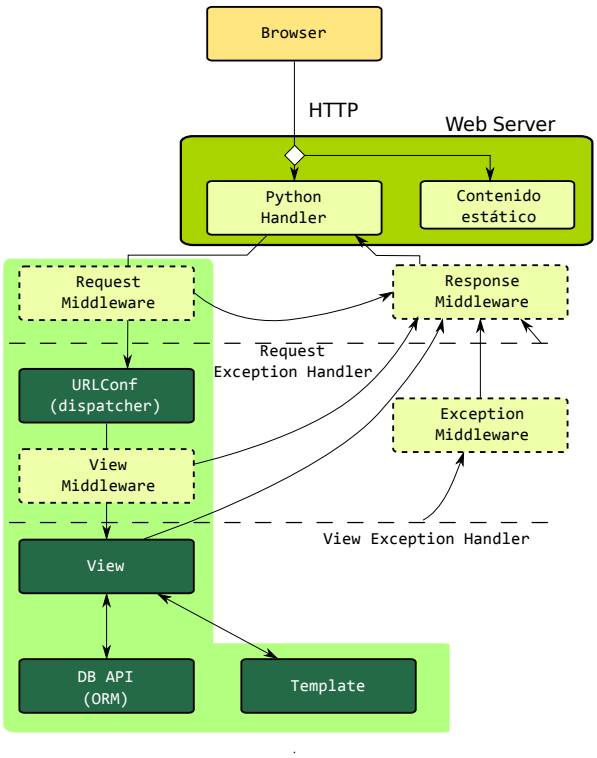

Figura 2.2: Esquema de flujo de una aplicación Django

Capítulo 3

El cliente Web

3.1. Estructura de un navegador

3.2. Google Gears

Google Gears es un plug-in para los navegadores: Mozilla Firefox, Internet Explorer y IE Mobile, Opera y Opera Mobile, Safari y Google Chrome. Gears es un proyecto de código abierto y añade 3 componentes básicos al navegador.

- **Local Server:** Permite almacenar en caché y proporcionar recursos de aplicaciones (HTML, JavaScript, imágenes, etc.) de forma local.
- **Database:** Permite almacenar datos localmente en una base de datos relacional en la que se pueden realizar búsquedas.
- **Worker Pool:** Permite realizar tareas que utilizan intensamente el CPU de forma similar a "procesos" en un sistema operativo, de manera de que el las aplicaciones tengan mejor respuesta.

Estos componentes estan enfocados en permitir al programador de aplicaciones web ejecutar sus aplicaciones cuando el navegador no está conectado al servidor.

Vale aclarar que la aplicación debe ser transferida de manera previa al cliente.

3.2.1. Componentes adicionales de Google Gears

A partir de la versión 0.4 del Gears

- API para GIS, que permite acceder a la posición geográfica del usuario.
- El API Blob, que permite gestionar bloques de datos binarios.
- Accede a archivos en el equipo cliente a través del API de Google Desktop.
- Permite enviar y recibir Blobs con el API HttpRequest.
- Localización de los cuadros de diálogo de Gears en varios idiomas.

3.3. Evolución del lenguaje en el cliente

3.3.1. JavaScript

3.4. Propuesta de extensión de Google Gears

3.5. Propuesta de extensión y aprovechamiento de los avances en cliente

Capítulo 4

La libreria

4.1. Inicio

La idea de diseñar y desarrollar un framework en que funcione en el hambiente de un navegador web, como es Firefox, deja entrever muchos aspectos que no resultan para nada triviales al momento de codificar.

- Se requieren varias lineas de codigo para implementar un framework.
- Como llega el codigo al navegador y se inicia su ejecucion.
- La cara visible o vista debe ser fasilmente manipulable por la aplicacion de usuario.
- Como los datos generados en el cliente son informados al servidor.
- El framework debe brindar soporte a la aplicacion de usuario de una forma natural y transparente.
- Se debe promover al reuso y la extension de funcionalidad del framework.
- Como se ponen en marcha los mecanimos o acciones que la aplicacion de usuario define.
- ...

En este capitulo se introducen las ideas principales que motivaron la creacion de una libreria en JavaScript, que brinde el soporte necesario al framework y a buena parte de los items expuestos.

Si bien el desarrollo de la libreria se mantuvo en paralelo a la del framework, existen aspectos basicos a los que esta brinda soporte y permiten presentarla en un apartado separado como una "Libreria JavaScript", esta constituye la base para posteriores construcciones y auna herramientas que simplifican el desarrollo client-side.

*proto*type + *py*thon = *protopy*

"La creación nace del caos", la libreria "Protopy" no escapa a esta afirmacion e inicialmente nace de la integracion de Prototype con las primeras funciones para lograr la modularizacion; con el correr de las lineas de codigo[1] el desarrollo

[1] Forma en que los informaticos miden el paso del tiempo

del framework torna el enfoque inicial poco sustentable, requiriendo este de funciones mas Python-compatibles se desecha la libreria base y se continua con un enfoque más "pythonico", persiguiendo de esta forma acercar la semántica de JavaScript 1.7 a la del lenguaje de programacion Python.

No es arbitrario que el navegador sobre el cual corre Protopy sea Firefox y mas particularmente sobre la version 1.7 de JavaScript. El proyecto mozilla esta hacercando, con cada nueva versiones del lenguaje, la semantica de JavaScript a la de Python, incluyendo en esta version generadores e iteradores los cuales son muy bien explotados por Protopy y el framework.

Protopy

Protopy es una libreria JavaScript para el desarrollo de aplicaciones web dinamicas. Aporta un enfoque modular para la inclusión de código, orientación a objetos, manejo de AJAX, DOM y eventos.

Para una referencia completa de la API de Protopy remitase al apandice A de la página 30.

4.2. Organizando el codigo

Undo de los principales inconvenientes a los que Protopy da solucion es a la inclucion dinamica de funcionalidad bajo demanda, esto se logra con los "modulos". Tradicionalmente la forma de incluir codigo JavaScript es mediante la incorporation de una etiqueta "script" con un atributo de referencia al archivo que contiene la funcionalidad. Mas tarde cuando el navegador descarga el documento y comienza su lectura al encontrar esta etiqueta solicita al servidor el archivo referenciado y lo interpreta, para continuar luego con la lectura del resto de las etiquetas. Este enfoque resulta sustentable en el desarrollo tradicional, en donde el lenguaje brinda mayormente soporte a la interaccion con el usuario y los fragmentos de codigo que se trasladan al cliente son bien conocidos por el desarrollador; para un proyecto que implica gran cantidad de codigo, como en este caso, el enfoque resulta complejo de sostener. Buscando una mejor forma de organizar y obtener el codigo es que surge el concepto de "modulo", similar a los modulos en python, cada unidad basica de de codigo define un modulo, este puede ser importado por otro fragemento de codigo y cada uno representa su propio ambito de nombres. Existen en Protopy dos tipos de modulos, los modulos integrados y los modulos organizados en archivos JavaScript. Para acceder a los modulos es necesario establecer un esquema de nombrado ... Con los modulos y un sistema de nombrado para el acceso a los mismos, la responsabilidad de la carga del codigo se deja en manos del cliente y del propio codigo que requiera determinada funcionalidad provista por un modulo.

Las funciones principales para trabajar con los modulos son "require" para cargar un modulo en el hambito de nombres local y "publish" para que los modulos publiquen o expongan la funcionalidad.

4.3. Creando tipos de objeto

En la programación basada en prototipos las "clases" no están presentes, y la re-utilización de procesos se obtiene a través de la clonación de objetos ya

existentes. Protopy agrega el concepto de clases al desarrollo, mediante un contructor de "tipos de objeto". De esta forma los objetos pueden ser de dos tipos, las clases y las instancias. Las clases definen la disposición y la funcionalidad básicas de los objetos, y las instancias son objetos "utilizables" basados en los patrones de una clase particular. ...

Como ya se menciono anteriormente Protopy explota las novedades de JavaScript 1.7, para los iteradores el constructor de tipos prevee el metodo `__iter__` con la finalidad de que los objetos generados en base al tipo sean iterables.

Los primeros tipos que surgen para la organizacion de datos dentro de la librerias con los "Sets" y los "Diccionarios", hambos aproximan su estructura a las estructuras homonimas en python, brindando una funcionalidad similar. Si bien la estructura "hasheable" nativa a JavaScript en un objeto, los diccionarios de Protopy permiten el uso de objetos como claves en lugar de solo cadenas.

4.4. Extendiendo el DOM

Si bien el **D**ocument **O**bject **M**odel (DOM) ofrece ya una **A**pplication **P**rogramming **I**nterface (API) muy completa para acceder, añadir y cambiar dinámicamente el contenido estructurado en el documento HTML.

4.5. Manejando los eventos

4.6. Envolviendo a gears

4.7. Auditando el codigo

4.8. Interactuando con el servidor

4.9. Soporte para json

Java**S**cript **O**bject **N**otation (JSON) brinda un buen soporte al intercambio de datos, resultando de fásil lectura/escritura para las personas y de un rapido interpretacion/generacion para las maquinas. Se basa en un subconjunto del lenguaje de programación JavaScript, estándar ECMA-262 3ª Edición - Diciembre de 1999. Este formato de texto es completamente independiente del lenguaje de programacion, pero utiliza convenciones que son familiares para los programadores de lenguajes de la familia "C", incluyendo C, C + +, C #, Java, JavaScript, Perl, Python y muchos otros.

JSON se basa en dos estructuras:

- Una colección de pares nombre / valor. En varios lenguajes esto se representa mediante un objeto, registro, estructura, diccionario, tabla hash, introducido lista o matriz asociativa.
- Una lista ordenada de valores. En la mayoría de los lenguajes esto se representa como un arreglo, matriz, vector, lista, o secuencia.

Estas son estructuras de datos universales. Prácticamente todos los lenguajes de programación modernos las soportan de una forma u otra. Tiene sentido que un formato de datos que es intercambiable con los lenguajes de programación también se basan en estas estructuras.

Para mas informacion sobre JSON http://www.json.org/

Mientras que un cliente se encuentre sin conexion con el servidor web, es capaz de generar y almacenar datos usando su base de datos local. Al reestablecer la conexion con el servidor web, estos datos deben ser transimitos a la base de datos central para su actulizacion y posterior sincronizacion del resto de los clinetes. La transferencia de datos involucra varios temas, uno de ellos y que compete a este apartado, es el formato de los datos que se deben pasar por la conexcion; este formato debe ser "comprendido" tanto por el cliente como por el servidor. Desde un primer momento se penso en JSON como el formato de datos a utilizar, es por esto que Protopy incluye un modulo para trabajar con el mismo.

No existe una razon concreta por la cual se deja de lado el soporte en Protopy para XML como formato de datos; aunque se puede mencionar la simplicidad de implementacion de un parser JSON contra la implementacion de uno en XML. Para el leector interesado agregar el sporte para XML en Protopy consta de escribir un modulo que realize esa tarea y agregarlo al paquete base.

El soporte para JSON se encuentra en el modulo "json.[en]tre los modulos estandar de Protopy. Este brinda soporte al pasaje de estructuras de datos JavaScript a JSON y viceversa. Los tipos base del lenguje JavaScript estan soportados y tienen su reprecentacion correspondiente, object, array, number, string, etc. pero este modulo interpreta ademas de una forma particular a aquellos objetos que implementen el metodo `__json__`, dejando de este modo en manos del desarrollador la reprecentacion en JSON de determinado objetos. La inclusion del metodo `__json__` resulta de especial impotancia a la hora de pasar a JSON los objetos creados en base a tipos definidos por el desarrollador mediante el contructor "type".

Con el soporte de datos ya establecidos en la libreria, el framework solo debe limitarce a hacer uso de él y asegurar la correcta sincronizacion de datos entre el cliente y el servidor web, este tema se retomara en el capitulo de sincronizacion.

4.10. Ejecutando codigo remoto

El RPC (del inglés Remote Procedure Call, Llamada a Procedimiento Remoto) es un protocolo que permite a un programa de ordenador ejecutar código en otra máquina remota sin tener que preocuparse por las comunicaciones entre ambos. El protocolo es un gran avance sobre los sockets usados hasta el momento. De esta manera el programador no tenía que estar pendiente de las comunicaciones, estando éstas encapsuladas dentro de las RPC.

Las RPC son muy utilizadas dentro del paradigma cliente-servidor. Siendo el cliente el que inicia el proceso solicitando al servidor que ejecute cierto procedimiento o función y enviando éste de vuelta el resultado de dicha operación al cliente.

Hay distintos tipos de RPC, muchos de ellos estandarizados como pueden ser el RPC de Sun denominado ONC RPC (RFC 1057), el RPC de OSF denominado DCE/RPC y el Modelo de Objetos de Componentes Distribuidos de Microsoft

DCOM, aunque ninguno de estos es compatible entre sí. La mayoría de ellos utilizan un lenguaje de descripción de interfaz (IDL) que define los métodos exportados por el servidor.

Hoy en día se está utilizando el XML como lenguaje para definir el IDL y el HTTP como protocolo de red, dando lugar a lo que se conoce como servicios web. Ejemplos de éstos pueden ser SOAP o XML-RPC. XML-RPC es un protocolo de llamada a procedimiento remoto que usa XML para codificar los datos y HTTP como protocolo de transmisión de mensajes.[1]

Es un protocolo muy simple ya que sólo define unos cuantos tipos de datos y comandos útiles, además de una descripción completa de corta extensión. La simplicidad del XML-RPC está en contraste con la mayoría de protocolos RPC que tiene una documentación extensa y requiere considerable soporte de software para su uso.

Fue creado por Dave Winer de la empresa UserLand Software en asociación con Microsoft en el año 1998. Al considerar Microsoft que era muy simple decidió añadirle funcionalidades, tras las cuales, después de varias etapas de desarrollo, el estándar dejó de ser sencillo y se convirtió en lo que es actualmente conocido como SOAP. Una diferencia fundamental es que en los procedimientos en SOAP los parámetros tienen nombre y no interesan su orden, no siendo así en XML-RPC.[2

Capítulo 5

El framework MTV

5.1. Percistencia

5.2. Generando HTML

5.3. Capturando datos

Capítulo 6

Soporte offline de una aplicación

6.1. Mecanismos de extension de Django

Un proyecto django es un conjunto de aplicaciones que corren en un servidor web mediante WSGI, mod-python o FastCGI, utilizando alun tipo de base de base de datos.

Para brindar soporte a las aplicaciones web en un cliente desconectado de manera organizada, integraremos un conjunto de componetes y mecanismos de django, con una libería de javascript en el cliente.

Hemos visto que existen varias características del lenguaje de porogramación Python que están ausentes en un navegador web convecional y por lo tanto han sido implementadas equivalencias en la librería Portopy.

La estructura básica del proyecto en el servidor se mantendrá de manera similar en el clinete, solo que el código de las vistas y de los modelos deberá ser reescrito.

6.1.1. Comandos Personalizados

Para poder realizar extensiones sobre la funcionalidad de Django sus desarrolladores han añadido la posibilidad de crear comandos personalizados. Un comando personalizado se define en un módulo, dentro de una aplicación y es ejecutado de manrera similar a cualquer otro comando provisto por el framework(como *syncdb*, *runserver*, *dumpdata*).

De igual manera que los comandos provistos por el framework, los comandos personalizados son ejecutados mediante el script de administración de proyecto *manage.py* o *django-admin.py*.

Un comando se define dentro de una aplicación. No existen comandos a nivel proyecto. Para que estos comandos sean detectados por el módulo de administración de poryectos deben respetar la siguiente estructura dentro del directorio de la aplicación

```
management/
    __init__.py
    commands/
```

```
__init__.py
mi_comando.py
```

Además la aplicación debe estar incluida en **INSTALLED_APPS** dentro de el módulo de configuración de proyecto **settings.py**.

Listing 6.1: Comando personalizado básico en Django

```
1
2 from django.core.management.base import NoArgsCommand
3
4 class Command(NoArgsCommand):
5   help = '''Descripci n del comando '''
6   def handle_noargs(self, **options):
7     print "Soy un comando personalizado"
```

Desde el punto de vista del lenguaje, la creación de un comando es la creación de un módulo management.commands.nombre_comando, en el cual se extiende a alguna clase base Command (que se encuentran en django.core.management.base).

Las clases base diponible son NoArgsCommand, para la imeplentación de comandos que no reciben argumentos; AppLabelCommand para la implementación de comandos que reciben como argumento el nombre de una aplicación instalada y

6.2. Creación de un sitio remoto

Para crear un sitio remoto, se importa **offfline.remote.RemoteSite** y se crea una instancia de esta clase. RemoteSite recibe un los parámetros *name*, *offline_root*, *offline_base*. El parámetro *name* debe ser único entre todas las instancias de la calase en el proyecto. El parámetro *offline_root* es una ruta sobre la cual se encuentra el sitio desconectado, es decir, el código javascript del proyecto. El parámetro *offline_base* es la url base sobre el cual se sirve el proyecto offline.

http://localhost:8000/{{ offline_base }}/{{ name }}/

Listing 6.2: Configuración básica de un sitio remoto

```
1 from offline.remote import RemoteSite
2
3 site = RemoteSiste('mi_sitio',
4                    offline_root = "/ruta/a/mi/sitio",
5                    offline_base = 'offline')
```

Un RemoteSite cumple con el objetivo de proyecto offline.

6.2.1. Registo de modelos

Para registrar un modelo en un RemoteSite, se utiliza el método *register(modelo [, remote])*

Listing 6.3: Registro de un RemoteSite

```
from mi_proyecto.mi_aplicacion.models import Persona
site.register(Persona)
```

Pero la exportación de modelos bajo un RemoteSite se puede personalizar mediante una clase RemoteModelProxy:

```
from offline.remote import RemoteModelProxy
from mi_proyecto.mi_aplicacion.models import Persona

class PersonaRemote(RemoteModelProxy):
    class Meta:
        model = Persona

site.register(Persona, PersonaRemote)
```

La responsabilidad de la clase RemoteModelProxy es realizar la serialización del los registros del modelo para envairlos al cliente, coordinado por el RemoteSite.

6.2.2. Publicación de un RemoteSite

Para publicar un RemoteSite bajo una URL

Modificación del módulo settings.py y urls.py

Definir en el módulo **settings.py** del proyecto las constantes **OFFLINE_BASE** y **OFFLINE_ROOT**. La constante **OFFLINE_BASE** es la URI relativa al proyecto offline, mientras que **OFFLINE_ROOT** es una ruta accesible en el sistema de archivos donde se vuelca el código que ser llevará al clinete. Además se debe agregar a **INSTALLED_APPS** la aplicación offline. Además de agergar estas constantes a la configuración, es necesario crear una instancia de un objeto RemoteSite. Este objeto es el encargado de realizar la transferecia al browser de la lógica y los datos, además de proveer mecanismos de sincronización. Lo recomendable es crear un modulo remote en la raíz del proyecto con el siguiente código:

Listing 6.4: Creando el sitio remoto

```

from offline.remote import RemoteSite

site = RemoteSite()
```

Creación de los RemoteProxy yo registro en el sitio remoto

Es necesario indicar al sitio remoto que modelos se desan exponer offline y como. Para esto existe una clase encargada, llamada RemoteProxy. La instancia del sitio remoto tiene un método register() similar al que poseen las insntancias de los sitios de aministración de django.contrib.admin.

Migración del poryecto mediante el comando migrate_project

Una vez modificado el archvio de configuración, se debe ejecutar el comando **manage.py migrate_project** dentro de la raíz del proyecto. Esto creara la estructura del proyecto offline.

Migración de aplicaciones

Django divide el proyecto en aplicaciones. En aplicaciones web orientadas al contenido, como son los blogs, foros, redes sociales, etc. los modelos de cada aplicación suelen estar débilmente acoplados. Generalmente hacer referencia a las clases de modelos provistos por django, como **django.cotrib.auth.User**, **django.contrib.auth.Group**, pero no existe relacion entre modelos de una aplicación y otra. Existen otros tipos de sistemas, orientados a organizaciones en las cuales se lelvan a cabo operaciones comerciales, donde los modelos suelen tener cierta relación entre si.

6.2.3. Concepto de RemoteSite

En el framework django, los proyectos agrupan aplicaciones, que son paquetes que constan de módulos de patrones de urls, vistas, modelos, módulo de administración, etc. y se comportan como una entidad independiente. Una aplicación debe estar *instalada* en un proyecto para estar activa, es decir, estar incluida en **INSTALLED_APPS** en el módulo de configuración de proyecto (**settings.py**). Una aplicación puede ser copiada o movida a otro proyecto si cumple con algunas condiciones de modularidad, como:

- Imoprtación de módulos sin base en el proyecto, es decir, si el proyecto se llama mi_proyecto, no existen sentecias como

  ```
  import mi_proyecto.models
  ```

  debido a que al mover o copiara la aplicación, las referencias con base en *mi_proyecto* quedan sin refrencia.

El equivalente a un proyecto django en el lado del servidor es un remote site del lado del cliente. En un proyecto django donde se encuentre instalada la aplicación **offline** pueden existir uno o más remote sites. El remote site tiene como responsabilidad la exportación de los modelos al clinete, mantenimiento de las versiones de código javascript en el cliente tanto del proyecto como del sistema, inicialización y sincronización de datos.

6.3. Sincronización de un RemoteSite

Durante la sincronización el cliente y el servidor intercambian regitros de la base de datos.

6.3.1. Modos de trabajo

Los distinotos modos de trabajo de la aplicacion son:

Estado online sin soporte offline

En este modo la aplicacion trabaja directamente sobre el servidor de aplicacion y depende completamente de una conexion. Al activar e implementar el soporte offline para nuestra aplicacion, en el servidor; el cliente debe ser conciente de este cambio y pasar a soportar los modos.

Estado online con soporte offline

Estado offline con soporte offline

Capítulo 7

Concluciones

Capítulo 8

Lineas futuras

8.1. Sitio de administración

Django se caracteriza por brindar una aplicación (*django.contrib.admin*) de admnistración que permite realizar CRUD (*Create*, *Retrieve Update*, *Delete*) sobre los modelos de las aplicaciones de usuario, interactuando con la aplicación *django.contrib.auth* que provee usuarios, grupos y permisos.

8.2. Historial de navegación

8.3. Workers con soporte para Javascript 1.7

Google Gears provee un mecanismo de ejecución de código en el cliente de manera concurrente llamado Worker Pool. De esta manera tareas que demandan tiempo de CPU pueden ser envaidas a segundo plano, de manera de no entorpecer el refresco de la GUI. Una caractersística de los worker pools, es que se ejecutan en un ámbito de nombres diferente al del "hilo principal". Es decir, existe encapsulamiento de su estado.

Apéndice A

Protopy

Protopy es una librería en JavaScript que simplifica el desarrollo de aplicaciones web dinámicas. Agregando un enfoque modular para la inclusión de código, orientación a objetos, soporte para AJAX, manipulación del DOM y eventos.

A.1. Módulos

Uno de los principales inconvenientes a los que Protopy da solución es a la inclusión dinámica de funcionalidad bajo demanda, esto es logrado mediante los módulos. Básicamente un modulo en un archivo con código javascript que recide en el servidor y es obtenido y ejecutado en el cliente.

Listing A.1: Estructura de un modulo

```
1  //Archivo: tests/module.js
2  require('event');
3
4  var h1 = $('titulo');
5
6  function set_texto(txt) {
7      h1.update(txt);
8  }
9
10 function get_texto() {
11     return h1.innerHTML;
12 }
13
14 event.connect($('titulo'), 'click', function(event) {
15     alert('El texto es: ' + event.target.innerHTML);
16 });
17
18 publish({
19     set_texto: set_texto,
20     get_texto: get_texto
21 });
```

```
>>> require('tests.module')
>>> module.get_texto()
"Test de modulo"
>>> module.set_texto('Un titulo')
>>> require('tests.module', 'get_texto')
>>> get_texto()
"Un titulo"
>>> require('tests.module', '*')
>>> set_texto('Hola luuu!!!')
>>> get_texto()
"Hola luuu!!!"
```

A.2. Módulos incluidos

Estos módulos están incluidos en el núcleo de Protopy, es decir que están disponibles con la sola inclusión de la librería en el documento. Los módulos que a continuación se detallan engloban las herramientas básicas requeridas para el desarrollo del lado del cliente.

A.2.1. builtin

Este modulo contiene las funciones principales de Protopy, en el se encuentran las herramientas básicas para realizar la mayoría de las tareas. No es necesario requerir este modulo en el espacio de nombres principal ("window"), ya que su funcionalidad esta disponible desde la carga de Protopy en el mismo.

Funciones

publish

```
publish(simbols: Object)
```

Publica la funcionalidad de un modulo. Para interactuar con el código definido en un modulo es necesario exponer una interfase de acceso al mismo, de esto se encarga la función publish.

require

```
require(module_name: String[, simbol: String...]) -> module: Object | simbol
```

Importa un modulo en el espacio de nombres. Al invocar a esta función un modulo es cargado desde el servidor y ejecutado en el cliente, la forma en que el modulo se presenta en el espacio de nombres depende de la invocación.

- var cntx = require('doff.template.context') Importa el modulo 'doff.template.context' y lo retorna en cntx, dejando también una referencia en el espacio de nombres llamado 'context', esta dualidad en la asociación del modulo permite importar módulos sin asociarlos a una variable, simplemente alcanza con asumir que la parte final del nombre es la referencia a usar.
- var cur = require('gears.database', 'cursor') Importa el modulo 'gears.databse' y retorna en cur el objeto publicado bajo el nombre de cursor, similar al caso anterior una referencia se define en el espacio de nombres para cursor.

- require('doff.db.models', 'model', 'query') Importa el modulo 'doff.db.models' y define en el espacio de nombres las referencias a model y query usando los mismos nombres.
- require('doff.core.urlpattern', '*') Importa del modulo 'doff.core.urlpattern' todos los objetos publicados y los publica en el espacio de nombres.

type

`type(name: String, [bases: Array ] [, class: Object ], instance: Object) -> Type`

Función encargada de la construcción de nuevos tipos de objeto o simplemente clases. Una vez definido un nuevo tipo este puede ser utilizado para la construcción de instancias mediante el operador "new". Los argumentos para la funcion "type"son, el nombre para el nuevo tipo de objeto, los tipos base de los cuales se hereda funcionalidad, opcionalmente los atributos y/o metodos de clase y los atributos y/o metodos para la instancia. La función que inicializa los objetos tiene por nombre `__init__` y es llamada en el momento de la construcción; en conjunto con otros metodos que se mencionaran a lo largo del texto estas funciones resultan de especial interés para interactuar con nuestros objetos y existen operadores en Protopy para manejarlas; esto es, no debieran ser invocadas o llamadas directamente.

Listing A.2: Definicion de tipos

```
var Animal = type('Animal', object, {
    contador: 0,
}, {
    __init__: function(especie) {
        this.especie = especie;
        this.orden = Animal.contador++;
    }
});

var Terrestre = type('Terrestre', Animal, {
    caminar: function() {
        console.log(this.especie + ' caminando');
    }
});

var Acuatico = type('Acuatico', Animal, {
    nadar: function() {
        console.log(this.especie + ' nadando');
    }
});

var Anfibio = type('Anfibio', [Terrestre, Acuatico]);

var Piton = type('Piton', Terrestre, {
    __init__: function(nombre) {
        super(Terrestre, this).__init__(this.__name__);
        this.nombre = nombre;
```

```
28      },
29      caminar: function() {
30          throw new Exception(this.especie + ' no camina');
31      },
32      reptar: function() {
33          console.log(this.nombre + ' la ' + this.especie.
                toLowerCase() + ' esta
34  reptando');
35      }
36  });
37
38  var doris = new Piton('Doris');
39  var ballena = new Acuatico('Ballena');
40  var rana = new Anfibio('Rana');
```

```
>>> doris
window.Piton especie=Piton orden=0 nombre=Doris __name__=Piton
>>> rana
window.Anfibio especie=Rana orden=2 __name__=Anfibio
>>> isinstance(rana, Terrestre)
true
>>> isinstance(doris, Animal)
true
>>> issubclass(Anfibio, Acuatico)
true
>>> issubclass(Piton, Animal)
true
>>> doris.caminar()
Exception: Piton no camina args=[1] message=Piton no camina
>>> doris.reptar()
Doris la piton esta reptando
```

$

```
$(id: String) -> HTMLElement
$(id: String[, id...]) -> [HTMLElement...]
```

Esta función recibe una cadena de texto y retorna el elemento del documento cuyo "id" se corresponda con la cadena. En conjunto con la función $$ constituyen dos herramientas muy útiles para recuperar elementos e interactuar con el árbol DOM. Si mas de un argumento es pasado, la forma de retorno es mediante un arreglo, permitiendo así la iteración sobre los mismos.

```
>>> $('content')
<div id="content">
>>> $('content body')
>>> $('content', 'body')
[div#content, div#body]
>>> $('content', 'body', 'head')
[div#content, div#body, undefined]
```

$$

```
$$(cssRule: String) -> [HTMLElement...]
```

Recupera elementos del documento, basando las reglas de seleccion en las reglas de css o hoja de estilos.

```
[div#wrap, div#top, div#content, div.header, div.breadcrumbs, div.middle, div
    ,
div.right, div#clear, div#footer, div#toolbar]
>>> $$('div#toolbar')
[div#toolbar]
>>> $$('div#toolbar li')
[li, li.panel, li.panel, li, li]
>>> $$('div#toolbar li.panel')
[li.panel, li.panel]
>>> $$('a:not([href~=google])')
[a add_post, a add_tag, a removedb, a syncdb]
>>> $$('a:not([href=google])')
[a add_post, a add_tag, a#google www.google.com, a removedb, a syncdb]
>>> $$('div:empty')
[div#logger.panel, div#dbquery.panel, div#clear, div#top]
```

extend

```
extend(destiny: Object, source: Object) -> alteredDestiny: Object
```

Extiende sobre un objeto destino todos los objetos pasados como argumentos a continuación, copiando cada uno de los atributos correspondientes, el objeto destino es retornado modificado.

```
>>> a = {perro: 4}
>>> b = {gato: 4}
>>> c = extend(a, b)
>>> c
Object perro=4 gato=4
>>> a
Object perro=4 gato=4
>>> b
Object gato=4
```

super

```
super(type: Type, instance: Object) -> boundedObject: Object
```

Enlaza un objeto con un tipo de objeto, de este modo la invocación sobre una función del tipo se realizara sobre el objeto enlazado. Normalmente esta función es utilizada para llamar a metodos de un tipo base.

isundefined

```
isundefined(object: Object) -> boolean
```

Determina si un objeto no esta definido o asociado a un valor. Retorna un valor de verdad correspondiente.

isinstance

```
isinstance(object, type | [type...]) -> boolean
```

Retorna verdadero si el objeto es una instancia del tipo, si un arreglo de tipos es pasado como segundo argumento el valor de verdad surge de preguntar por cada uno de ellos.

issubclass

```
issubclass(type1, type2 | [type...]) -> boolean
```

Retorna si type1 es una subclase de type2, cuando se pasa un arreglo en lugar de type2 la evaluación se realiza para cada una de las clases incluidas en el mismo.

Arguments

```
new Arguments(arguments) -> Arguments
```

En JavaScript El objeto para los argumentos asociativos debe ir al final de la invocación

Listing A.3: Uniformando argumentos

```
function unaFuncion(arg1, arg2, arg3) {
  var todos = new Arguments(arguments);
  print('Argumento 1: %s o %s o %s', arg1, todos[0], todos.
      arg1);
  print('Argumento 2: %s o %s o %s', arg2, todos[1], todos.
      arg2);
  print('Argumento 3: %s o %s o %s', arg3, todos[2], todos.
      arg3);
  print('Otros argumentos: %s', todos.args);
  print('Argumentos pasados por objeto: %o', todos.kwargs);
}
function otraFuncion(arg1) {
  var todos = new Arguments(arguments, {'def1': 1, 'def2':
      2});
  print('Argumento 1: %s o %s o %s', arg1, todos[0], todos.
      arg1);
  print('Otros argumentos: %s', todos.args);
  print('Argumentos pasados por objeto: %o', todos.kwargs);
}
```

```
>>> unaFuncion('uno', 2, null, 3, 4, 5, {'nombre': 'Diego', 'apellido': 'van
    Haaster'})
Argumento 1: uno o uno o uno
Argumento 2: 2 o 2 o 2
Argumento 3: null o null o null
Otros argumentos: 3,4,5
Argumentos pasados por objeto: Object nombre=Diego apellido=van Haaster
>>> unaFuncion('uno', 2, null, {'nombre': 'Diego', 'apellido': 'van Haaster
    '})
...
Otros argumentos:
Argumentos pasados por objeto: Object nombre=Diego apellido=van Haaster
>>> unaFuncion('uno', 2, null, 3, 2, 3, 4)
...
Otros argumentos: 3,2,3,4
Argumentos pasados por objeto: Object
>>> otraFuncion('uno', 2, {'nombre': 'Diego', 'apellido': 'van Haaster'})
Argumento 1: uno o uno o uno
Otros argumentos: 2
Argumentos pasados por objeto: Object def1=1 def2=2 nombre=Diego apellido=van
     Haaster
>>> otraFuncion('uno', 2, {'def1': 'Diego', 'apellido': 'van Haaster'})
Argumento 1: uno o uno o uno
```

```
Otros argumentos: 2
Argumentos pasados por objeto: Object def1=Diego def2=2 apellido=van Haaster
```

Template

```
new Template(destiny, source) -> Template
```

Dict

```
new Dict(object) -> Dict
```

```
>>> dic = new Dict({'db': 5, 'template': 2, 'core': 9})
>>> obj = {'un': 'objeto'}
>>> dic.set(obj, 10)
>>> arreglo = [1,2,3,4,obj]
>>> dic.set(arreglo, 50)
>>> dic.get('template')
2
>>> dic.get(arreglo)
50
>>> dic.get(obj)
10
>>> dic.items()
[["db", 5], ["template", 2], ["core", 9], [Object un=objeto, 10], [[1, 2, 3,
    2
more...], 50]]
>>> dic.keys()
["db", "template", "core", Object un=objeto, [1, 2, 3, 2 more...]]
>>> dic.values()
[5, 2, 9, 10, 50]
```

Set

```
new Set(array) -> Set
```

Un set es una coleccion de elementos unicos, de forma similar a los conjuntos este objeto soporta intersecciones, uniones, restas, etc.

```
>>> set = new Set([1,2,3,4,5,6,7,8,9,3,6,1,4,7])
>>> len(set)
9
>>> set.add(6)
>>> set)
9
>>> set2 = set.intersection([1,3,5,6])
>>> set2.elements
[1, 3, 5, 6]
```

hash

```
hash(string | number) -> number
```

Retorna un valor de hash para el argumento dado, para los mismos argumentos se teronran los mismos valores de hash.

id

```
id(value) -> number
```

Asigna y retorna un identificador unico para el valor pasado como argumento. Al pasar un valor que sea de tipo objeto la funcion id modificara la estructura interna agregando el atributo `__hash__` para "etiquetar" el objeto y en posteriores llamadas retornara el mismo identificador.

getattr

```
getattr(object, name, default) -> value
```

Obtiene un atributo de un objeto mediante su nombre, en caso de pasar un valor por defecto este es retornado si el atributo buscado no esta definido en el objeto, en caso contrario una excepcion es lanzada.

setattr

```
setattr(object, name, value)
```

Establece un atributo en un objeto con el nombre pasado. El valor establecido pasa a formar parte del objeto.

hasattr

```
hasattr(object, name) -> boolean
```

Retrona verdadero en caso de que el objeto tenga un atributo con el nombre correspondiente, falso en caso contrario.

assert

```
assert(boolean, mesage)
```

Chequea que el valor de verdad pasado sea verdadero en caso contrario retorna una excepcion conteniendo el mensaje pasado.

bool

```
bool(object)
```

Determina el valor de verdad de un objeto pasado, los valores de verdad son como sigue: arreglos, objetos y cadenas vacias en conjunto con los valores null y undefined son falsos; todos los demas casos son verdaderos. En el caso particular de que un objeto defina el metodo `__nonzero__` este es invocado para determinar el valor de verdad.

callable

```
callable(value) -> boolean
```

Retorna verdadero en caso de que el valor pasado sea instancia de una funcion osea pueda ser llamado, falso en caso contrario.

chr

```
chr(number) -> character
```

Retorna el caracter correspondiente al numero ordinal pasado.

ord

```
ord(character) -> number
```

Retorna un numero correspondiente al caracter pasado.

```
>>> ord(chr(65))
65
>>> chr(ord("A"))
"A"
```

bisect

```
bisect(seq, element) -> position
```

Dada una secuencia ordenada y un elemento la funcion bisect retorna un numero referenciando a la posicion en que el elemnto debe ser insertado en la secuencia, para que esta conseve su orden. Si los elementos de la secuencia definen el metodo `__cmp__` este es invocado para determinar la posicion a retornar.

```
>>> a = [1,2,3,4,5]
>>> bisect(a,6)
5
>>> bisect(a,2)
2
>>> a[bisect(a,3)] = 3
>>> a
[1, 2, 3, 3, 5]
```

equal

```
equal(object1, object2) -> boolean
```

Compara dos objetos determinando el valor de igual para los mismos, verdadero es retornado en caso de ser los dos objetos iguales. En caso de que object1 defina el metodo `__eq__` este es invocado con object2 pasado como parametro para determinar la igualdad.

nequal

```
nequal(object, object) -> boolean
```

Compara dos objetos determinando el valor de igual para los mismos, verdadero es retornado en caso de ser los dos objetos distintos. En caso de que object1 defina el metodo `__ne__` este es invocado con object2 pasado como parametro para determinar la no igualdad.

number

```
number(object) -> number
```

Convierte un objeto a su representacion numerica.

flatten

```
flatten(array) -> flattenArray
```

Aplana un arreglo de modo que el resultado sea un unico arreglo conteniendo todos los elementos que se pasaron en multiples arreglos a la funcion.

include

```
include(seq, element) -> boolean
```

Determina si un elemento esta incluido en una secuencia o coleccion de objetos, si la coleccion implementa el metodo `__contains__`, este es utilizado para determinar la pertenencia del elemento.

len

```
len(seq) -> boolean
```

Retorna un valor numerico representando la cantidad de elementos contenidos en la secuencia o coleccion, si la coleccion implementa el metodo `__len__`, este es utilizado para determinar la cantidad de elementos.

array

```
array(seq) -> [element...]
```

Genera un arreglo en base a la secuencia pasada, si la secuencia implementa el metodo `__iter__`, este es utilizado para llenar el arrgelo con los elementos.

print

```
print(text...)
```

Si la consola de firebug esta instalada este metodo imprime el texto pasado por consola.

string

```
string(object)
```

Retorna una reprecentacion en texto del objeto pasado como argumento. Si el objeto define el metodo `__str__` este es invocado para obtener la reprecentacion.

values

```
values(object) -> [value...]
```

Retorna un arreglo con los valores del objeto pasado como argumento.

keys

```
keys(object) -> [key...]
```

Retorna un arreglo con las claves del objeto pasado como argumento.

items

```
items(object) -> [[key, value]...]
```

Retorna en forma de arreglo cada pareja clave, valor de un objeto pasado como argumento.

```
>>> items({'perro': 1, 'gato': 7})
[["perro", 1], ["gato", 7]]
```

inspect

```
inspect(object) -> string
```

unique

```
unique(array) -> [element...]
```

Dado un arreglo con elementos repetidos retorna un nuevo arreglo que se compone de los elementos unicos encontrados.

range

```
range([begin = 0, ] end[, step = 1]) -> [number...]
```

Retorna un arreglo conteniendo una progresion aritmetica de numeros enteros. Los parametros son variables y en su invocacion mas simple se pasa solo el final de la secuencia de numeros a generar, asumiendo para ello un inicio en 0 y un incremento en una unidad, estos valores pueden ser modificados.

```
>>> range(10)
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> range(4, 10)
[4, 5, 6, 7, 8, 9]
>>> range(4, 10, 2)
[4, 6, 8]
```

xrange

```
xrange([begin = 0, ] end[, step = 1]) -> generator
```

Similar a range pero en lugar de retornar un arrglo retorna un objeto que generar los valores bajo demanda.

zip

```
zip(seq1 [, seq2 [...]]) -> [[seq1[0], seq2[0] ...], [...]]
```

Retorna un arreglo en donde cada secuencia contenida es el resultado de combinar cada una de las secuencias que se pasaron como argumento, la longitud de las secuencias queda acotada a la longitud de la secuencia mas corta.

```
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f','g','h'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, "e"], [6, "f"]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d'])
[[1, "a"], [2, "b"], [3, "c"], [4, "d"], [5, undefined], [6, undefined]]
>>> zip([1,2,3,4,5,6], ['a','b','c','d','e','f'], [10,11,12,13,14,15,16])
[[1, "a", 10], [2, "b", 11], [3, "c", 12], [4, "d", 13], [5, "e", 14], [6, "f
    ",
15]]
```

A.2.2. sys

Este modulo provee acceso a algunos objetos y funciones mantenidas por Protopy y que resultan de utilidad para interactuar con el ambiente.

version

Version de Protopy.

browser

Este objeto provee informacion sobre el navegador en el cual protopy se cargo.

- IE Si el navegador es Internet Explorer.
- Opera Si el navegador es Opera.
- WebKit Si el navegador es AppleWebKit.
- Geck Si el navegador es Gecko.
- MobileSafari Si el navegador es Apple Mobile Safari.
- fatures Algunas herramientas que el navegador proveea, por ejemplo XPath, un selector de css, y otras extensiones.

gears

Objeto gears para interactuar con el plugin de google gears. installed install factory

register_path

Registra una ruta en el servidor para un paquete, de este modo, las imporaciones de modulos dependientes de de ese paquete se realizaran sobre la ruta asociada.

module_url

Retorna la ruta correspondiente al nombre de modulo pasado.

modules

Un objeto para asociar los nombres de modulos con los modulos propiamente dicho que se van cargando bajo demanda.

paths

Las rutas registradas para la carga de modulos.

A.2.3. exception

Modulo que reune todas las excepciones que Protopy provee a la hora de mostrar errores, este modulo es cargado en el ambiente cuando Protopy inicia, no siendo necesario su requerimiento posteriormente.

Excepciones Exception, AssertionError, AttributeError, LoadError, KeyError, NotImplementedError, TypeError, ValueError.

A.2.4. event

Este es el modulo encargado de encapsular la logica de eventos requerida tanto por los elementos del DOM como por el usuario.

Funciones

connect

```
connect(obj: Object|null, event: String, context: Object|null, method:
String|Function, dontFix: Boolean) -> handle: Handle
```

Provee un mecanimos para conectar la ejecucion de una funcion a otra o a un evento del DOM.

disconnect

```
disconnect(handle: Handle)
```

Quita el la relacion establecida por “connect”.

subscribe

```
subscribe(topic: String, context: Object|null, method: String|Function) ->
handle: Handle
```

Subscribe una funcion a un evento de usuario expresado como un texto, cuando el evento ocurra la funcion se ejecuta.

unsubscribe

```
unsubscribe(handle: Handle)
```

Quita la relacion de la funcion con el evento.

publish

```
publish(topic: String, arguments: Array)
```

Emite el evento de usuario, provocando la ejecucion de las funciones subscriptas y pasando los argumentos correspondientes.

connectPublisher

```
connectPublisher(topic: String, obj: Object, event: String) -> handle: Handle
```

Conecta un evento a un evento de usuario asegurando que cada vez que el evento se produsca se llamara a la funcion registrada para el evento de usuario.

fixEvent

```
fixEvent(evt: Event, sender: DOMNode)
```

Normaliza las propiedades de un evento, tanto pulsasiones del teclado como posiciones x/y del raton.

stopEvent

```
stopEvent(evt: Event)
```

Detiene el evento, evitando la propagacion y la accion por defecto.

keys

Objeto que encapsula los codigos de las teclas de funcion y control.

A.2.5. timer

Funciones

delay

```
delay(function)
```

defer

```
defer(function)
```

A.2.6. ajax

Este modulo contiene funcionalidad propia del ajax, para el manejo de peticiones asincronicas al servidor.

Funciones

Request

```
new Request()
```

new Request()

Response

```
new Response()
```

new Response()

toQueryParams

```
toQueryParams(string, separator) -> object
```

toQueryString

```
toQueryString(params) -> string
```

A.2.7. dom

Este modulo brinda el soporte para el manejo del DOM de una forma simple para el usuario.

Funciones

query

```
query(cssRule) -> [HTMLElement...]
```

A.3. Extendiendo Javascript

Protopy no solo aporta modulos y funciones utiles al desarrollo, sino que tambien agrega nueva funcionalidad a los objetos de javascript.

A.3.1. String

Estos metodos amplian las herramientas para el manejo de cadenas de texto.

sub

```
string.sub(pattern, replacement[, count = 1]) -> string
```

Retorna una cadena donde la primera ocurrencia del patron pasado es reemplazado por la cadena o cada uno de los valores retornados por la funcion pasada como segundo argumento. El patron puede ser una exprecion regular o una cadena.

subs

```
string.subs(value...) -> string
```

Substitulle cada patron encontrado en la cadena por los valores correspondientes, si el primer valor es un objeto, se espera un patron del tipo clave en la cadena para su reemplazo.

format

```
string.format(f) -> string
```

Da formato a una cadena de texto, al estilo C.

inspect

```
string.inspect(use_double_quotes) -> string
```

Retorna una version de debug de la cadena, esta puede ser con comillas simples o con comillas dobles.

truncate

```
string.truncate([length = 30[, suffix = '...']]) -> string
```

Recorta una cadena recortada en la longitud indicada o 30 caracteres por defecto, si se pasa un sufijo este es utilizado para indicar el recorte, sino los "..." son utilizados.

strip

```
string.strip() -> string
```

Quita los espacios en blanco al principio y al final de una cadena.

striptags

```
string.striptags() -> string
```

Quita las etiquetas HTML de una cadena.

stripscripts

```
string.stripscripts() -> string
```

Quita todos los bloques "strips" de una cadena.

extractscripts

```
string.extractscripts() -> [ string... ]
```

Extrae todos los scripts contenidos en la cadena y los retorna en un arreglo.

evalscripts

```
string.evalscripts() -> [ value... ]
```

Evalua todos los scripts contenidos en la cadena y retorna un arreglos con los resultados de cada evaluacion.

escapeHTML

```
string.escapeHTML() -> string
```

Convierte los caracteres especiales del HTML a sus entidades equivalentes.

unescapeHTML

```
string.unescapeHTML() -> string
```

Convierte las entidades de caracteres especiales del HTML a sus respectivos simbolos.

succ

```
string.succ() -> string
```

Convierte un caracter en el caracter siguiente segun la tabla de caracteres Unicode.

times

```
string.times(count[, separator = '']) -> string
```

Concatena una cadena tantas veces como se indique, si se pasa un separador, este es utilizado para intercalar.

camelize

```
string.camelize() -> string
```

Convierte una cadena separada por guiones medios (“-”) a una nueva cadena tipo ”camello”. Por ejemplo, ’foo-bar’ pasa a ser ’fooBar’.

capitalize

```
string.capitalize() -> string
```

Pasa a mayuscula la primera letra y el resto de la cadena a minusculas.

underscore

```
string.underscore() -> string
```

Convierte una cadena tipo ”camello” a una nueva cadena separada por guiones bajos (“_”).

dasherize

```
string.dasherize() -> string
```

Remplaza cada ocurrencia de un guion bajo ("_") por un guion medio ("-").

startswith

```
string.startswith(pattern) -> boolean
```

Chequea si la cadena inicia con el patron pasado.

endswith

```
string.endswith(pattern) -> boolean
```

Chequea si la cadena termina con el patron pasado.

blank

```
string.blank() -> boolean
```

Chequea si una cadena esta en blanco, esto es si esta vacia o solo contiene espacios en blanco.

A.3.2. Number

Estos metodos agregan funcionalidad sobre los objetos numericos.

format

```
number.format(f, radix) -> string
```

Da formato a un numero en base a una cadena de texto, al estilo C.

A.3.3. Date

toISO8601

```
date.toISO8601() -> string
```

Retorna una representacion de la fecha en ISO8601.

A.3.4. Element

Extencion sobre los elementos del DOM, simplificando trabajos comunes de desarrollo.

visible

```
HTMLElement.visible() -> Boolean
```

Retorna un valor de verdad que determina si el elemento esta visible al usuario, verificando el atributo de estilo "display".

toggle

```
HTMLElement.toggle() -> HTMLElement
```

Alterna la visibilidad del elemento.

hide

```
HTMLElement.hide() -> HTMLElement
```

Oculta el elemento al usuario, modificando el atributo de estilo.

show

```
HTMLElement.show() -> HTMLElement
```

Muestra el elemento.

remove

```
HTMLElement.remove() -> HTMLElement
```

Quita el elemento del documento y lo retorna.

update

```
HTMLElement.update(content) -> HTMLElement
```

Reemplaza el contendio del elemento con el argumento pasado y terorna el elemento.

insert

```
HTMLElement.insert({ position: content }) -> HTMLElement
HTMLElement.insert(content) -> HTMLElement
```

Inserta conentido al principio, al final, sobre o debajo del elemento, para definir la poscicion de la insercion el argumento se debe pasar en forma de objeto, donde la clave es la pocicion y el valor el contenido a insertar; si el argumento es contenido a insertar este se inserta al final del elemento.

select

```
HTMLElement.select(selector) -> HTMLElement
```

Toma un numero arbitrario de selectores CSS y retorna un arreglo con los elementos que concuerden con estos y esten dentro del elemento al que se aplica la funcion.

```
>>> $('PostForm').select('input')
[input#id_title, input guardar]
>>> $('content').select('div')
[div.header, div.breadcrumbs, div.middle, div, div.right, div#clear]
>>> $('content').select('div.middle')
[div.middle]
```

A.3.5. Forms

Estos metodos decoran a los elementos del tipo formulario, agregando funcionalidad sobre los mismos y sobre los campos que contienen.

disable

```
HTMLFormElement.disable() -> HTMLFormElement
```

Deshabilita todos los campos de este formulario para el ingreso de valores.

enable

```
HTMLFormElement.enable() -> HTMLFormElement
```

Habilita todo campos del formulario para el ingreso de valores.

serialize

```
HTMLFormElement.serialize() -> object
```

Retorna un objeto conteniendo todos campos del formulario serializados con sus respectivos valores.

```
>>> $('PostForm')
<form id="PostForm" method="post" action="/blog/add_post/">
>>> $('PostForm').serialize()
Object title=Hola mundo body=Este es un post tags=[1]
```

A.3.6. Forms.Element

Los metodos que a continuación se presentan decoran a los elementos o campos de un formulario, simplificando y agilizando el trabajo con los mismos.

serialize

```
HTMLElement.serialize() -> string
```

Crea una cadena en URL-encoding representando el contenido del campo expresado como clave=valor, para su uso en una peticion AJAX por ejemplo. Este atributo trabaja sobre un unico campo, si en lugar de esto se requiere serializar todo el formulario vea Form.serialize(). Si se requiere es el valor del campo en lugar de la pareja clave=valor, vea get_value().

get_value

```
HTMLElement.get_value() -> value
```

Retorna el valor actual del campo. Una cadena de texto es retornada en la mayoria de los casos excepto en el caso de un select multiple, en que se retorna un arreglo con los valores.

set_value

```
HTMLElement.set_value(value) -> HTMLElement
```

Establese el valor de un campo.

clear

```
HTMLElement.clear() -> HTMLElement
```

Limpia un campo de texto asignando como valor la cadena vacia.

present

```
HTMLElement.present() -> boolean
```

Retorna verdadero si el campo de texto tiene un valor asignado, falso en otro caso.

activate

```
HTMLElement.activate() -> HTMLElement
```

Pone el cursor sobre el campo y selecciona el valor si el campo es del tipo texto.

disable

```
HTMLElement.disable() -> HTMLElement
```

Deshabilita el campo, impidiendo de este modo que se modifique su valor hasta que sea habilitado nuevamente. Los campos de un formulario que esten deshabilitados no se serializan.

enable

```
HTMLElement.enable() -> HTMLElement
```

Habilita un campo, previamente deshabilitado, para el ingreso de valores.

A.4. Otros modulos

A.4.1. gears

—

A.4.2. logging

—

A.4.3. json

```
>>> require('json')
>>> toJson = {'numero': 1, 'cadena': 'texto', 'arreglo': [1,2,3,4,5,6], '
    objeto': {'clave': 'valor'}, 'logico': true}
>>> toSend = json.stringify(toJson)
"{"numero": 1, "cadena": "texto", "arreglo": [1, 2, 3, 4, 5, 6], "objeto": {"
    clave": "valor"}, "logico": true}"
>>> fromJson = json.parse(toSend)
Object numero=1 cadena=texto arreglo=[6] objeto=Object
```

A.4.4. rpc

```
class MyFuncs:
    def _listMethods(self):
        # this method must be present for system.listMethods
        # to work
        return ['add', 'pow']
    def _methodHelp(self, method):
        # this method must be present for system.methodHelp
        # to work
        if method == 'add':
            return "add(2,3) => 5"
        elif method == 'pow':
            return "pow(x, y[, z]) => number"
        else:
            # By convention, return empty
            # string if no help is available
            return ""
    def _dispatch(self, method, params):
        if method == 'pow':
            return pow(*params)
        elif method == 'add':
            return params[0] + params[1]
        else:
            raise 'bad method'

server = SimpleXMLRPCServer(("localhost", 8000))
dispatcher.register_introspection_functions()
dispatcher.register_instance(MyFuncs())
#dispatcher.serve_forever()
```

```
>>> require('rpc')
>>> funcs = new rpc.ServiceProxy('rpc/test', {asynchronous: false})
>>> funcs.system.listMethods()
["add", "pow", "system.listMethods", "system.methodHelp", "system.
    methodSignature"]
>>> for each (m in funcs.system.listMethods())
                            print(m + ': ' + funcs.system.methodHelp(m));
add: add(2,3) => 5
pow: pow(x, y[, z]) => number
system.listMethods: system.listMethods() => ['add', 'subtract', 'multiple']
        Returns a list of the methods supported by the server.
system.methodHelp: system.methodHelp('add') => "Adds two integers together"
        Returns a string containing documentation for the specified method.
```

```
system.methodSignature: system.methodSignature('add') => [double, int, int]
        Returns a list describing the signature of the method. In the above
             example,
the add method takes two integers as arguments and returns a double result.
        This server does NOT support system.methodSignature.
>>> funcs.add(24,4)
28
>>> funcs.pow(2,4)
16
```

Add → `{"version": "1.1", "method": "add", "id": 1, "params":`$[24, 4]$`|` → `{"id":1,"result":28}`

Pow → `{"version": "1.1", "method": "pow", "id": 2, "params": [2,4]}` ← `{"id":2,"result":16}`

Apéndice B

Doff

B.1. Modelos

B.2. Consultas

B.3. Plantillas

B.4. URLs

B.5. Vistas

B.6. Formularios

Apéndice C

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).

C.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 D

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.
-

Bibliografía

[JavaScript] Versiones de Javascript, Jhon Reisig

[DOM] Modelo en Objetos para la representación de Documentos

[AJAX] JavaScript asíncrono y XML

Glosario

Application Programming Interface (API) Una interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción., 19

Document Object Model (DOM) EL Modelo en Objetos para la representación de Documentos o tambien Modelo de Objetos del Documento es esencialmente una interfaz de programación de aplicaciones que proporciona un conjunto estándar de objetos para representar documentos HTML y XML, un modelo estándar sobre cómo pueden combinarse dichos objetos, y una interfaz estándar para acceder a ellos y manipularlos., 19