the original Hacker

creado por EUGENIA BAHIT

Jugando con la inteligencia

Número 1

Woman Eyes creado por Mourad Mokrane - Silueta de Mujer creado por Leonardo B. Cunha

THE ORIGINAL HACKER

SOFTWARE LIBRE, HACKING y PROGRAMACIÓN, EN UN PROYECTO DE

EUGENIA BAHIT

@eugeniabahit

GLAMP HACKER Y PROGRAMADORA EXTREMA

Hacker especializada en Programación eXtrema e Ingeniería Inversa de código sobre GNU/Linux, Apache, MySQL, Python y PHP. eugeniabahit.com

Docente e instructora de tecnologías GLAMP cursos.eugeniabahit.com cursosdeprogramacionadistancia.com

Miembro de la Free Software Foundation fsf.org, The Linux Foundation linuxfoundation.org e integrante del equipo de Debian Hackers debianhackers.net.

Creadora de python-printr, Europio Engine, JackTheStripper. Vim contributor. Fundadora de Hackers n' Developers Magazine y responsable editorial hasta octubre '13.

Buenos Aires, 5 de Noviembre de 2013

ÍNDICE DE LA EDICIÓN NRO1

Shell Scripting: Análisis de argumentos enviados por línea de comandos mediante Python con argparse

argparse es un módulo de la librería estándar de Python, que reemplazando a optparse desde la versión 2.7 del lenguaje, se ha convertido en el módulo por excelencia para analizar los argumentos enviados a través de la línea de comandos.

El módulo `argparse` forma parte de la librería de módulos estándar de Python y su finalidad es la de analizar los argumentos enviados al programa mediante línea de comandos, facilitando las mismas funcionalidades que el obsoleto `optparse` pero incorporando ciertas características con las que éste no contaba.

Se trata de un módulo muy simple de utilizar y no necesariamente será implementado solo por especialistas en Python: también es una excelente alternativa para crear de forma rápida un *script* principal (*main*) para cualquier tipo de aplicación de consola, incluso, aquellas programadas con bash u otros lenguajes que no cuenten con tanta facilidad para, por ejemplo, el análisis de argumentos y/o la generación de ayudas en pantalla.

Con solo agregar una lista de argumentos a `ArgumentParser()`, el módulo se encargará de poner a disposición del usuario de la aplicación, los argumentos `-h` y `--help` y generar de forma automática, textos de ayuda similares al siguiente:

```
eugenia@cococha-gnucita:~/HDMagazine/12$ python newhost --help
usage: newhost [-h] [-v] -d DOMAIN [-a [ALIAS [ALIAS ...]]]
               [-l [{static,python,php}]] [-u USERNAME] [-p PATH]
               [-lp LOGPATH] [--send-email] [-e EMAIL]

Prepara el ambiente necesario para hospedar un nuevo dominio en Ubuntu Server
12.04 LTS o versiones posteriores

optional arguments:
  -h, --help            show this help message and exit
  -v, --version         show program's version number and exit
```

```
  -d DOMAIN, --domain DOMAIN
                        Nombre del dominio a configurar
  -a [ALIAS [ALIAS ...]], --alias [ALIAS [ALIAS ...]]
                        Alias de dominio
  -l [{static,python,php}], --language [{static,python,php}]
                        Lenguaje predeterminado del sitio Web
  -u USERNAME, --user USERNAME
                        Usuario del dominio
  -p PATH, --path PATH  Directorio raíz de archivos Web
  -lp LOGPATH, --log-path LOGPATH
                        Directorio en el que serán almacenados los logs de
                        Apache
  --send-email          Si se indica, enviará un e-mail con los datos del
                        nuevo dominio.
  -e EMAIL, --email EMAIL
                        Válido si --send-email se ha indicado.
```

Como se puede observar en el bloque anterior, dos argumentos por defecto, son descriptos al comienzo: `help` y `version`. Ambos son facilitados por `argparse` para mostrar la ayuda y versión del programa, respectivamente. Incluso, la ayuda de uso será mostrada si los argumentos recibidos no son los esperados:

```
eugenia@cococha-gnucita:~/HDMagazine/12$ python newhost
usage: newhost [-h] [-v] -d DOMAIN [-a [ALIAS [ALIAS ...]]]
               [-l [{static,python,php}]] [-u USERNAME] [-p PATH]
               [-lp LOGPATH] [--send-email] [-e EMAIL]
newhost: error: argument -d/--domain is required
```

Todo esto es lo que `argparse` pondrá a disposición del usuario, con unas pocas líneas de código fuente.

Introducción

Importación del módulo:

Para comenzar a utilizar `argparse`, bastará con importar la clase `ArgumentParser()`:

```
from argparse import ArgumentParser
```

Construcción de un objeto ArgumentParser:

Construir un objeto `ArgumentParser`, es una forma de inicializar los datos principales de la aplicación. El método constructor del objeto ArgumentParser (función __init__), si bien puede ser invocado sin argumentos, permite (entre otros), los siguientes parámetros:

prog	por defecto es el nombre del archivo aunque en circunstancias muy puntuales, podría modificarse
description	una descripción del programa que será mostrada al inicio de la ayuda
epilog	texto que será mostrado al final de la ayuda

```
version       número de versión del programa
```

Todos estos parámetros son opcionales pero sin embargo, si se indicase el parámetro `version`, automáticamente dispondríamos de la opción `-v` y `–version`:

```
#!/usr/bin/env python

from argparse import  ArgumentParser

argp = ArgumentParser(
    version='1.0',
    description='Descripción breve del programa',
    epilog='Copyright 2013 Autor bajo licencia GPL v3.0'
)
```

Otros **parámetros admitidos por `ArgumentParser.__init__()`** pueden verse en la siguiente URL del manual oficial: http://docs.python.org/2/library/argparse.html#argumentparser-objects

Agregando argumentos con ArgumentParser.add_argument

Cuando se crea un objeto `ArgumentParser`, éste dispone de un método `add_argument()` que como su nombre lo indica, tiene por finalidad agregar argumentos de a uno por vez. Este método, puede recibir como parámetro, un nombre de argumento o una lista de banderas (*flags*). Por ejemplo:

```
argp.add_argument('directorio', '-d', '--directorio')
argp.add_argument('dominio')
argp.add_argument('-l')
argp.add_argument('--listar')
argp.add_argument('-p', '--printer')
```

Además del argumento en sí mismo, `add_argument()` puede recibir muchos otros parámetros. Entre los más frecuentes, podemos encontrar los siguientes:

action
Descripción: Acción que se deberá realizar con el valor del argumento.
Valores posibles:

```
store        almacena el valor (acción predeterminada)
store_const  si el argumento es pasado, almacenará el valor definido en el parámetro
             const (ver más abajo). Es útil cuando se requiere recibir un flag pero
             sin valor asociado

store_true / store_false
             Igual que store_const pero no necesita definir el valor de const
```

	ya que almacenarán True o False respectivamente en caso que el argumento sea pasado
append	almacena los valores del argumento en una lista. Es útil cuando un mismo argumento puede indicarse varias veces con diferentes valores Por ejemplo: --argumento valor1 --argumento valor2 generararía argumento = ['valor1', 'valor2']
append_const	almacena el valor de const en una lista. Especialmente útil cuando el valor de diferentes argumentos es una constante y se los necesita de forma unificada (ver ejemplo para mejor comprensión). Requiere que el parámetro dest (ver más abajo) posea el mismo valor en los diferentes argumentos

Valor por defecto: store
Obligatorio: NO
Ejemplo:

```
argp.add_argument('--table', action='store')
--table foo genera: table = 'foo'
argp.add_argument('--table', action='store_const', const='users')
--table genera: table = 'users'
argp.add_argument('--table', action='append')
--table foo --table bar genera: table = ['foo', 'bar']
argp.add_argument('--php', dest='lenguajes',
                  action='append_const', const='php')
argp.add_argument('--python', dest='lenguajes',
                  action='append_const', const='python')
--php --python genera: lenguajes = ['php', 'python']
```

nargs
Descripción: Cantidad de valores que pueden recibirse para el argumento en cuestión.
Valores posibles: el literal de un entero (incluso cuando sea 1, retornará una lista), o sino:

?	uno o ninguno
+	uno o más
*	cero o más
REMAINDER	todos los argumentos se recogen en una lista. Esto es especialmente útil, cuando los comandos recibidos se requieren pasar a otra aplicación.

Valor por defecto: uno solo
Obligatorio: NO
Ejemplo:

```
argp.add_argument('--table', nargs='+')
argp.add_argument('--rango', nargs=2)
```

default

Descripción: Un valor por defecto para el argumento.
Valores posibles: cualquiera
Valor por defecto: ninguno
Obligatorio: NO
Ejemplo:

```
argp.add_argument('--host', default='localhost')
```

type

Descripción: El tipo de datos
Valores posibles: str, int, etc.
Valor por defecto: None
Obligatorio: NO
Ejemplo:

```
argp.add_argument('-n, --nombre', type=str)
argp.add_argument('--edad', type=int)
```

choices

Descripción: Una lista de opciones con valores posibles
Valores posibles: una lista
Valor por defecto: None
Obligatorio: NO
Ejemplo:

```
argp.add_argument('-l, --language', choices=['php', 'bash', 'ruby'])
```

required

Descripción: Indica si el argumento es o no obligatorio
Valores posibles:

```
True        Argumento obligatorio
False       Argumento no obligatorio
```

Valor por defecto: False
Obligatorio: NO
Ejemplo:

```
argp.add_argument('--obligatorio', required=True)
argp.add_argument('--opcional', required=False)
```

help

Descripción: Texto a mostrar en la ayuda del argumento.
Valores posibles: cadena de texto

Valor por defecto: None
Obligatorio: NO (aunque es muy recomendado indicarlo)
Ejemplo:

```
argp.add_argument('--list', help='Retorna la lista de tablas en la DB')
```

dest
Descripción: Nombre que se utilizará para la variable que almacenará el valor del argumento.
Valores posibles: string con nombre de variable válido
Valor por defecto: el nombre argumento o flag
Obligatorio: NO
Ejemplo:

```
argp.add_argument('-p', dest='path')
argp.add_argument('-h', dest='hostname')
```

Generar el análisis de los argumentos con ArgumentParser.parse_args

Finalmente, se necesitará indicar a `ArgumentParser` que analice los argumentos:

```
argumentos = argp.parse_args()
```

`parse_args` retornará cada argumento indicado por línea de comandos, según su configuración, como propiedades del objeto generado:

```
argumentos = argp.parse_args()
suponiendo un argumento cuyo destino sea llamado foo, obtendríamos su valor con:
print argumentos.foo
```

A continuación, el código que generó el texto de ayuda del ejemplo al comienzo del artículo:

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
from argparse import ArgumentParser

argp = ArgumentParser(
    prog='newhost',
    description="""Prepara el ambiente necesario para hospedar un nuevo
    dominio en Ubuntu Server 12.04 LTS o versiones posteriores""",
    epilog='Copyright 2013 Eugenia Bahit - GPL v3.0',
    version='New WebSite Hosting beta 1.0'
)

argp.add_argument( '-d', '--domain', action='store', required=True,
                   help='Nombre del dominio a configurar', dest='domain')
```

```
argp.add_argument( '-a', '--alias', action='store', required=False, nargs='*',
                   help='Alias de dominio', dest='alias' )

argp.add_argument( '-l', '--language', choices=['static', 'python', 'php'],
                   action='store', required=False, default='static', nargs='?',
                   help='Lenguaje predeterminado del sitio Web', dest='language' )

argp.add_argument( '-u', '--user', action='store', default='www-data', required=False,
                   help='Usuario del dominio', dest='username' )

argp.add_argument( '-p', '--path', action='store', default='/srv/websites/',
                   required=False, help='Directorio raíz de archivos Web', dest='path' )

argp.add_argument( '-lp', '--log-path', action='store', default='/srv/websites/logs/',
                   required=False,
                   help='Directorio en el que serán almacenados los logs de Apache',
                   dest='logpath' )

argp.add_argument( '--send-email', action='store_true', required=False,
                   help='Si se indica, enviará un e-mail con los datos del nuevo dominio.',
                   dest='sendemail' )

argp.add_argument( '-e', '--email', action='store', required=False,
                   help='Válido si --send-email se ha indicado.', dest='email' )

args = argp.parse_args()
print vars(args)

# ***** FIN CÓDIGO SCRIPT PYTHON *****
# ejecución del script

eugenia@cococha-gnucita:~/HDMagazine/12$ python newhost -v
New WebSite Hosting beta 1.0

eugenia@cococha-gnucita:~/HDMagazine/12$ python newhost -d eugeniabahit.com -a
www.eugeniabahit.com -l php -u juanito
{'username': 'juanito', 'domain': 'eugeniabahit.com', 'language': 'php', 'sendemail': False,
'logpath': '/srv/websites/logs/', 'alias': ['www.eugeniabahit.com'], 'path':
'/srv/websites/', 'email': None}
```

Procesos de Razonamiento Inverso: Patrón de Diseño Adapter en Python y PHP, los "cómo" y los "para qué"

Tienes un enchufe de tres patas pero un toma corriente de dos. Necesitas insertar la clavija en el toma sin modificar ninguna de las dos. Solución: utilizas un adaptador. Bienvenidas/os al patrón de diseño Adapter.

Necesitaba un ejemplo sencillo que me permitiese demostrar visualmente, cómo funciona el patrón de diseño Adapter y el ejemplo del enchufe, me vino «*como anillo al dedo*». Pues no podría ser más claro: es la imagen perfecta e inequívoca para comprender este maravilloso patrón de diseño.

En la programación orientada a objetos, un **patrón de diseño** representa la **forma en la cual se puede resolver el comportamiento de un objeto o la relación entre ellos** y el patrón de diseño *Adapter* (adaptador) resuelve la necesidad de conectar dos (o más) objetos entre sí, dónde al menos uno de ellos necesitaría ser modificado.

Pero para poder explicarlo sin riesgo a confusiones, me voy a valer de un caso real. Para ello, les pido que imaginemos que tenemos un típico objeto Usuario:

```
# La clase Usuario en PHP

class Usuario {

    function __construct() {
          $this->usuario_id = 0;
          $this->nombre = '';
    }

}


# La clase Usuario en Python

class Usuario(object):
```

```
    def __init__(self):
        self.usuario_id = 0
        self.nombre = ''
```

Y nos encontramos con que debemos implementar las siguientes **Historias de Usuario**:

- Cómo usuario puede enviar mensajes a otro usuario del sistema
- Cómo usuario puedo ver una lista de los mensajes que he recibido desde otros usuarios

Hablamos de un requerimiento típico de una gran parte de las aplicaciones tanto Web como de escritorio.

Si simplemente pensáramos en DATOS pero NO EN OBJETOS, nos encontraríamos con una tabla de mensajes, con un campo remitente_id, otro destinatario_id, y otros más destinados a almacenar la fecha, el asunto y el cuerpo del mensaje, por ejemplo. Y lo anterior, arrojaría como resultado una clase con propiedades simples y complicados y errados métodos no estándar, que realicen consultas cruzadas en la base de datos:

```
SELECT     mensaje.mensaje_id,
           mensaje.asunto,
           mensaje.cuerpo,
           mensaje.fecha,
           mensaje.estado,
           usuario.nombre AS remitente,
           usuario.nombre AS destinatario

FROM       mensaje INNER JOIN usuario
           ON mensaje.remitente_id = usuario.usuario_id
           OR mensaje.destinatario_id = usuario.usuario_id

etc.
```

Y luego la clase Mensaje derivaría del diseño previo de la base de datos:

```
class Mensaje(object):

    def __init__(self):
        self.mensaje_id = 0
        self.asunto = ''
        # …
        self.remitente_id = 0  # Usuario.usuario_id
        self.destinatario_id = 0  # también sería Usuario.usuario_id
```

Lo anterior, sería un ejemplo sobre cómo las “malas costumbres” nos llevan a **pensar en datos**. Pero sin embargo, **podríamos pensar estrictamente en objetos** y en este caso, nos encontraríamos con la siguiente definición inicial del objeto mensaje:

- El mensaje *tiene* 1 *remitente*
- El mensaje *tiene* 1 *destinatario*
- …

Esto, nos haría deducir, correctamente, que Remitente y Destinatario, serían dos tipos de objetos diferentes:

```
# En PHP
class Remitente { }

class Destinatario { }


# En Python
class Remitente(object):
    pass


class Destinatario(object):
    pass
```

Y a la vez, Mensaje, tendría dos propiedades compuestas por los objetos Remitente y Destinatario respectivamente:

```
# En PHP
class Mensaje {

    function __construct(Remitente $rtte=NULL, Destinatario $dest=NULL) {
          # ...
          $this->remitente = $rtte;
          $this->destinatario = $dest;
          # ...
    }
}


# En Python
class Mensaje(object):

    def __init__(self, remitente=None, destinatario=None):
          # ...
          self.remitente = compose(Remitente, remitente)
          self.destinatario = compose(Destinatario, remitente)
# notar que he utilizado una función llamada compose(ObjetoRequerido, objeto_compositor)
# cuyo fin sería el de componer a la propiedad
# para más información, se puede leer sobre el patrón de diseño compuesto en Python
# en http://www.bubok.es/libros/219288/Teoria-sintacticogramatical-de-objetos
```

Sin embargo, no necesitamos siquiera, analizar a los objetos Remitente y Destinatario en profundidad, para saber que en realidad estamos hablando de objetos Usuario:

```
# En PHP
class Remitente extends Usuario { }

class Destinatario extends Usuario  { }


# En Python
class Remitente(Usuario):
    pass
```

```
class Destinatario(Usuario):
    pass
```

Hasta aquí, nada parece complicado. Sin embargo, los objetos Remitente, Destinatario y Usuario, son exactamente iguales. Entonces **¿por qué no componer Mensaje directamente con el objeto Usuario?** La razón es sencilla y es que en la orientación objetos bien implementada, **las propiedades compuestas llevan el nombre del objeto que las compone** y eso permite:

- Obtener objetos más reales, relacionalmente lógicos y estructuralmente coherentes;
- Lograr una mayor legibilidad y facilidad de comprensión;
- No ensuciar los métodos constructores con funciones destinadas a manejar el tipo de datos de las propiedades;
- Utilizar mapeadores relacionales de objetos (ORM) basados en objetos reales y no en "parches" de datos colocados a los objetos.

Entonces, repasando, hasta aquí tendríamos los siguientes objetos y relaciones:

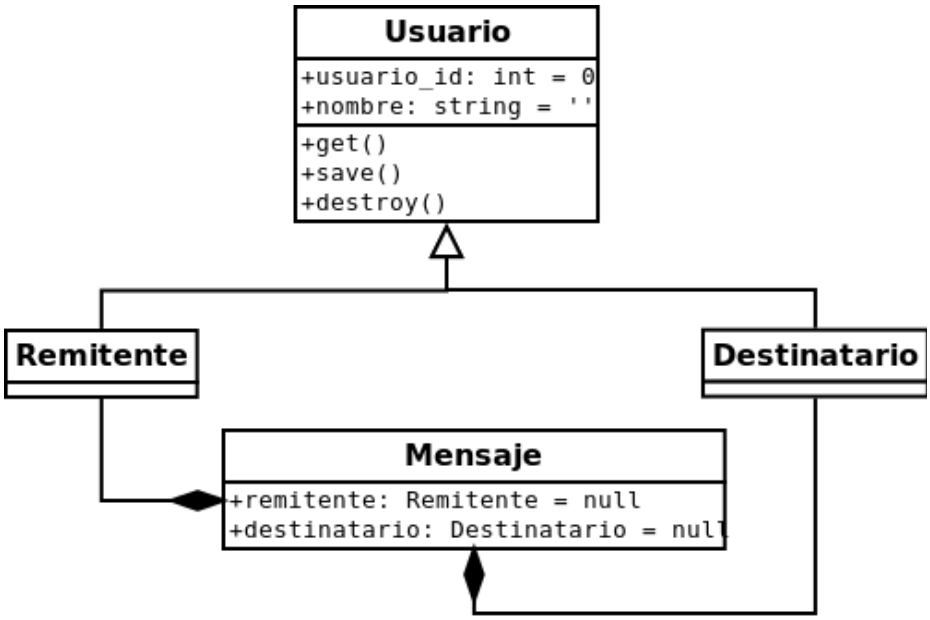


Diagrama UML realizado con Dia https://projects.gnome.org/dia/ (Software Libre)

Pero lo cierto es, que tanto Remitente como Destinatario son **objetos simulados**, creados para satisfacer las exigencias del objeto Mensaje. Y en realidad, cuando el método `get()` de Mensaje sea invocado y éste, intente componer sus dos propiedades (`Mensaje.remitente` y `Mensaje.destinatario`), el método `get()` real al que debería invocarse, tendría que ser el método `get()` de Usuario, ya que a decir verdad, es el verdadero compositor.

> *Mensaje debería recurrir al método get() de Usuario que es el verdadero compositor*

Aquí es dónde la "adaptación" real sale a la luz. Remitente y Destinatario al fin deberán "adaptar" las "clavijas" de `Usuario` para ser insertadas en el "toma corriente" de `Mensaje`. Es entonces cuando la función transformadora de ambos objetos, pasa a ser la adaptación del método `get()` de `Usuario`:

Cuando Mensaje **llame al método** get **de** Remitente **y** Destinatario, **en realidad deberá ser el método** get **de** Usuario **quien deba ejecutarse y luego, transformar a** Usuario **en un** Remitente **y en un** Destinatario **respectivamente.**

`Remitente` y `Destinatario`, deberán entonces, sobrescribir el método `get` heredado de `Usuario`, a fin de que `Usuario.get()` sea llamado, cada vez que `Remitente.get()` y `Destinatario.get()` sean invocados:

```
# En PHP
function get() {
    $usuario = new Usuario();
    $usuario->usuario_id = $this->usuario_id;
    $usuario->get();
    foreach($usuario as $property=>$value) {
        $this->$property = $value;
    }
}


# En Python
def get(self):
    usuario = Usuario()
    usuario.usuario_id = self.usuario_id
    usuario.get()
    for property, value in vars(usuario).iteritems():
        setattr(self, property, value)
```

Incluyendo esta sobre-escritura del método `get()` de Usuario, Remitente y Destinatario se transforman en dos adaptadores. Y sería genial, si no existiese la redundancia, pues el método get() se sobrescribe igual en Remitente que en Destinatario y uno de los principios del Refactoring indica que:

> *Si dos clases que heredan de la misma clase tienen métodos idénticos, se extrae el método repetido y se lo lleva a la clase Madre*

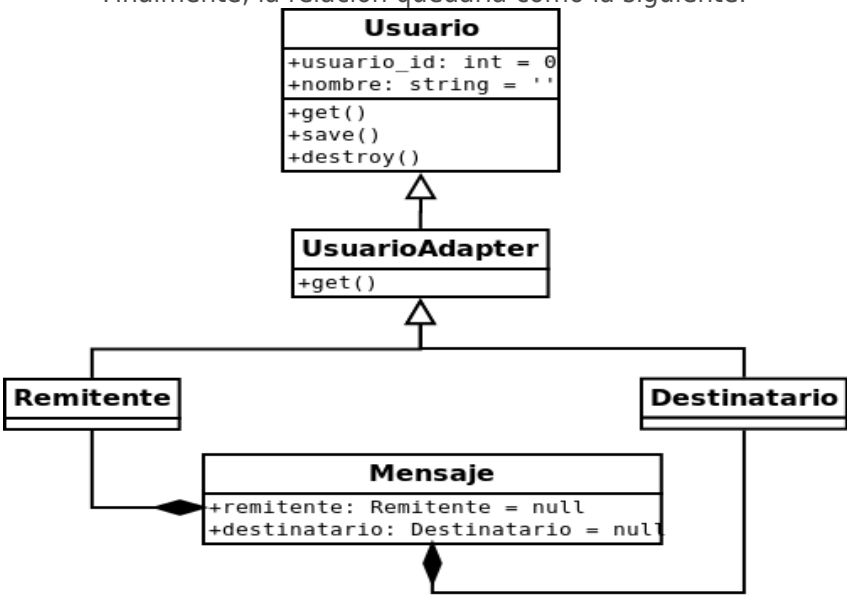
Pero ¿qué sucedería si este método fuese llevado a la clase madre? Claramente, estaríamos eliminando el verdadero método `get()` de Usuario. Entonces, la solución es extraer el método hacia una nueva clase y hacer heredar a Remitente y destinatario de esta nueva clase. Y ésta, a la vez, sería quien herede de Usuario:

```
# En PHP
class UsuarioAdapter extends Usuario {
```

```
    function get() {
          $usuario = new Usuario();
          $usuario->usuario_id = $this->usuario_id;
          $usuario->get();
          foreach($usuario as $property=>$value) {
                $this->$property = $value;
          }
    }

}


class Remitente extends UsuarioAdapter { }
class Destinatario extends UsuarioAdapter { }


# En Python
class UsuarioAdapter(Usuario):

    def get(self):
          usuario = Usuario()
          usuario.usuario_id = self.usuario_id
          usuario.get()
          for property, value in vars(usuario).iteritems():
                setattr(self, property, value)


class Remitente(UsuarioAdapter):
    pass


class Destinatario(UsuarioAdapter):
    pass
```

AVISO: notar que en los ejemplos no se ha utilizado el método constructor en las clases, no porque no deba utilizarse /al contrario/, sino porque no es de relevancia para el ejemplo.

Finalmente, la relación quedaría como la siguiente:



Patrón de Diseño Adapter en Europio Engine

Si estás utilizando **Europio Engine** (www.europio.org), habrá que tener en cuenta que al momento de invocar al método `get()` de `Mensaje`, cuando éste recurra al `get()` de `Remitente` y `Destinatario` para componer sus dos propiedades homónimas, el ORM considerará las propiedades remitente_id y destinatario_id, como las propiedades identificadoras de Remitente y Destinatario respectivamente. Lo propio sucederá con la propiedad id de UsuarioAdapter.

Por lo tanto, habrá que agregar a `Remitente`, `Destinatario` y `UsuarioAdapter`, sus respectivas propiedades ID y encargarse de asignarles el valor necesario:

```
class Remitente extends UsusarioAdapter {

    function __construct() {
        parent::__construct();
        $this->remitente_id = 0;
    }

    function get() {
        $this->usuario_id = $this->remitente_id;
        parent::get();
    }
}


class Destinatario extends UsusarioAdapter {

    function __construct() {
        parent::__construct();
        $this->destinatario_id = 0;
    }

    function get() {
        $this->usuario_id = $this->destinatario_id;
        parent::get();
    }
}


class UsuarioAdapter extends Usuario {

    function __construct() {
        $this->usuarioadapter_id = $this->usuario_id;
    }

    function get() {
        $usuario = new Usuario();
        $usuario->usuario_id = $this->usuario_id;
        $usuario->get();
        foreach($usuario as $property=>$value) {
            $this->$property = $value;
        }
    }

}
```

Ingeniería de Software: archivos pre y post instalación/desinstalación en Los paquetes Debian

En la edición Nro. 11 de la revista Hackers & Developers Magazine /www.hdmagazine.org/, escribí un artículo introductorio sobre la creación de paquetes .deb, avanzando muy superficialmente sobre el directorio DEBIAN. En este paper, la idea es comenzar a insertarnos más a fondo en dicho directorio, tomando como punto de partida, los archivos de instalación y desinstalación propios, que pueden ejecutarse mediante dpkg pero que exceden su responsabilidad.

Crear un paquete Debian de forma estándar con los elementos básicos, no siempre puede resultar ser la solución más acertada. Muchas veces, contar con un paquete que solo "distribuya" archivos por los diversos directorios, nos queda "chico" y se hace inevitable la necesidad de ejecutar tareas de instalación más allá de la responsabilidad de dpkg. Para esto, la solución es incorporar *scripts* de instalación propia, dentro del directorio DEBIAN .

Los archivos de instalación propios, pueden ser *scripts* que se ejecuten antes de la instalación o después de ésta. Incluso, pueden coexistir ambos *scripts* (pre y post instalación). Estos archivos deben crearse dentro del directorio DEBIAN y se los llamará **preinst** (para ejecutarse **antes** de la instalación) y **postinst** (para ser ejecutado **después** de la instalación).

Pero la mejor noticia es que **los archivos pre y post-instalación** para dpkg, **pueden ser escritos en cualquier lenguaje** de programación, pudiendo ser **desde *Shell Scripts* en bash hasta pequeños programas en PHP, Python, Ruby** o en cualquier otro lenguaje soportado.

Consideraciones previas

Antes de crear y utilizar los archivos DEBIAN/postinst y/o DEBIAN/preinst es necesario tener en cuenta los siguientes factores:

1. **Tanto** postinst **como** preinst **necesitan permisos de ejecución**

Los archivos de pre y post instalación, deben poder "ser ejecutados" por dpkg, motivo por el cual, será

necesario **otorgar permisos de ejecución a los archivos DEBIAN/postinst y DEBIAN/preinst** antes de proceder a su empaquetado.

Para dar permisos de ejecución al archivo, recuerda ejecutar:

```
chmod +x DEBIAN/postinst
chmod +x DEBIAN/preinst
```

2. dpkg NO elimina archivos ni directorios creados mediante postinst o preinst

Si tienes pensado hacer que tu pre o post instalador cree archivos y/o directorios, antes de decidirlo, ten en cuenta que éstos no serán desinstalados cuando `dpkg -r` *paquete* o `dpkg -P` *paquete* sean ejecutados. Una solución a este inconveniente, puede ser la siguiente:

3. crea un archivo DEBIAN/postrm o DEBIAN/prerm para que archivos y/o directorios creados mediante DEBIAN/postinst y DEBIAN/preinst sean desinstalados al llamar a dpkg -r o dpkg -P

Los archivos prerm y postrm se ejecutan antes y después de que dpkg sea llamado con los argumentos -r (remove) o -P (purge). En estos *scripts*, puedes colocar instrucciones para que, por ejemplo, archivos y/o directorios creados por `postinst` o `preinst` sean desinstalados mediante dpkg.

Al igual que los archivos de pre y post instalación, **prerm y postrm necesitan ser creados con permisos de ejecución** para que dpkg pueda ejecutarlos.

Argumentos enviados por dpkg

Cuando dpkg llama a `preinst`, el primer argumento que le pasa es `install`, es decir, que el archivo `preinst` es llamado por dpkg de la siguiente forma:

```
./preinst install
```

Al llamar a `postinst`, en cambio, el primer argumento será `configure`:

```
./postinst configure
```

Estos argumentos, deben ser tenidos en cuenta al momento de desarrollar la lógica pre y post instalación, para el paquete que se esté armando.

```
#!/bin/sh

set -e


if [ '$1' == 'configure' ]; then
    # acciones de configuración
    ruta_apache='/etc/apache2/sites-available'
    echo -n "Ingrese nombre de dominio: "; read domain_name
    sed s/DOMAIN_NAME/$domain_name/g $ruta_apache/myapp.conf $ruta_apache/$DOMAIN_NAME.conf
    rm $ruta_apache/myapp.conf
    a2ensite $DOMAIN_NAME.conf
    service apache2 restart
else
    echo 'dpkg no me ha llamado con configure así que no ejecuto nada'
fi


#DEBHELPER#

exit 0
```

Ejemplo #1: Shell Script que comprueba en el archivo **DEBIAN/postinst** que `configure` haya sido enviado por dpkg para ejecutar las acciones de configuración correspondientes. Si `configure` no es recibido, imprime un mensaje de error.

Ten en cuenta que postinst **no será ejecutado si** dpkg **es invocado con el argumento** --unpack

Cuando el argumento `--unpack` es pasado a dpkg para la instalación, justamente, el usuario solo está pidiendo el desempaquetado del .deb pero no su configuración. Dado que el archivo postinst está destinado a acciones de configuración, no es llamado si dpkg fue invocado mediante –unpack.

Europio Engine Lab: Formularios Web y Tablas HTML en solo unos pocos pasos

Hace apenas menos de un mes, lancé una nueva versión estable de Europio Engine: la 3.2.5. Esto me motivó a terminar dos complementos que tenía pendientes: un generador de formularios Web y otro, de listados en tablas HTML. En este artículo veremos como en unos pocos pasos, podemos crear formularios y tablas con estos dos complementos.

Uno de los fines de Europio Engine, como motor MVC, es abogar por la independencia absoluta de la GUI. Es entonces, que se hace necesario crear los archivos HTML que serán implementados en las vistas de cada recurso. Y esto incluye, por lo menos, a los métodos agregar, editar y listar.

Pero desde ahora, estos tres métodos podrán generar sus GUI de forma dinámica, gracias a los dos nuevos complementos que dan origen a este artículo. Se trata de dos *plug-ins* que conservan el estilo independentista de Europio Engine.

Ambos complementos pueden descargarse desde www.europio.org/downloads/plugins/

Y supongo que a estas alturas, demás estará decir, que al igual que Europio Engine, ambos complementos son **software libre**, distribuidos bajo los términos de la licencia **GNU GPL v 3.0**.

Europio WebForm: generador de formularios Web

Se trata de una clase para las vistas, que permite ir agregando campos de formulario configurables y así generar el formulario que realmente se necesite.

Instalación:

Tras descargar el paquete **Europio_WebForm_1_1_0_stable.tar.gz** desde el sitio Web de descargas de Europio Engine[1], descomprimir y colocar la carpeta `webform` dentro del directorio `./common/plugins/` de la aplicación.

1 http://www.europio.org/downloads/plugins/

Para **habilitar el complemento** y que éste sea importado de forma automática por el motor de Europio, bastará con agregarlo en la variable `$enabled_apps` del archivo `settings.php`:

```
$enabled_apps = array('webform');
```

Si no se lo habilita desde el `settings`, puede importarse luego de forma manual, desde el `user_imports.php` o desde la vista en la que se lo desee implementar.

Uso:
El primer paso, consiste en crear un objeto `WebForm`:

```
$action = '/modulo/modelo/guardar';
$method = 'POST';
$form = new WebForm($action, $method);
```

El constructor de la clase, puede recibir dos parámetros: el *action* y *method* del formulario HTML, pero ambos son opcionales.

A continuación, será necesario agregar los campos de formulario -de a uno por vez-, utilizando los métodos "add_*" disponibles. Existe un método "add_" por cada tipo de campo estándar de formulario: `hidden`, `text`, `textarea`, `password`, `file`, `select`, `radio`, `checkbox` y `submit`. Finalmente, para obtener el formulario, se utiliza el método `show`.

En resumen, la clase `WebForm` se compone de:

```
     Clase: WebForm

Parámetros:
    string $action      (opcional)   valor por defecto: '.'
    string $method      (opcional)   valor por defecto: 'POST'

   Métodos:
    add_hidden
    add_text
    add_textarea
    add_password
    add_file
    add_select
    add_radio
    add_checkbox
    add_submit
    show
```

Para implementar los métodos add_* listados anteriormente, los siguientes parámetros deberán ser tenidos en cuenta:

```
    Método: void add_hiden
Parámetros:
    string $name        nombre del campo
    string $value       valor por defecto         (opcional)

================================================================================

    Método: void add_text
Parámetros:
    string $name        nombre del campo
    string $value       valor por defecto         (opcional)
    string $text        Texto a mostrar en la etiqueta del campo
                                                  (opcional)   valor por defecto: $name

================================================================================

    Método: void add_textarea
Parámetros:
    string $name        nombre del campo
    string $value       valor por defecto         (opcional)
    string $text        Texto a mostrar en la etiqueta del campo
                                                  (opcional)   valor por defecto: 'Mensaje'

================================================================================

    Método: void add_password
Parámetros:
    string $name        nombre del campo
    string $text        Texto a mostrar en la etiqueta del campo
                                                  (opcional)   valor por defecto: $name

================================================================================

    Método: void add_file
Parámetros:
    string $name        nombre del campo
    string $text        Texto a mostrar en la etiqueta del campo
                                                  (opcional)   valor por defecto: $name

================================================================================

    Método: void add_select
Parámetros:
    string $name        nombre del campo
    array  $options     colección de objetos o de arrays asociativos para las opciones
    string $text        Texto a mostrar en la etiqueta del campo
                                                  (opcional)   valor por defecto: $name

================================================================================

    Método: void add_radio, void add_checkbox
Parámetros:
    string $groupname   nombre del grupo de campos
    array  $options     colección de objetos o de arrays asociativos para cada opción
    string $text        Texto a mostrar como título del grupo
                                                  (opcional)
                        valor por defecto: 'Seleccione las opciones de su preferencia'

================================================================================

    Método: void add_submit
Parámetros:
    string $value       texto del botón           (opcional)   valor por defecto: 'Enviar'
```

Los **métodos destinados a generar múltiples opciones** como `add_select`, `add_checkbox` y `add_radio`, reciben como segundo parámetro un array en el cual, cada uno de los elementos de éste, pueden ser objetos o ***arrays asociativos***.

Sean objetos o diccionarios, las propiedades o parámetros que deberán incluir, son:

```
value      valor de cada opción
text       texto a mostrar para cada opción
extras     si la opción irá seleccionada o marcada por defecto, indicar ' selected' o
           ' checked' según corresponda. Caso contrario, un valor nulo.
```

Un ***ejemplo con uso de array de opciones***, podría verse como el siguiente:

```
# se definen (a modo de ejemplo) las opciones para dos radio buttons
$opcion1 = array('value'=>1, 'texto'=>'Suscribirme', 'extras'=>' selected');
$opcion1 = array('value'=>2, 'texto'=>'Darme de baja', 'extras'=>'');

# Se las agrega a un array que se pasará finalmente a add_radio
$opciones = array($opcion1, $opcion2);

# Creo un WebForm
$form = new WebForm('/modulo/modelo/recurso');
# Agrego un campo para que el usuario escriba su e-mail
$form->add_text('email', NULL, 'E-mail');

# Agrego el grupo de radio buttons (envío mi array de opciones)
$form->add_radio('newsletter', $opciones, '¿Qué desea hacer?');

# Agrego un botón de envío
$form->add_submit();
# Almaceno el formulario en una variable
$html = $form->show();

# Muestro el formulario dentro de la plantilla general del sitio Web
$plantilla = file_get_contents(APP_DIR . '/static/html/webtemplate.html');
print Template('Newsletter')->show($html);
```

Hack para convertir colecciones de objetos en diccionarios para los formularios Web generados con el complemento WebForm

Imaginemos que tenemos una colección de objetos de tipo `Pais` cuyas propiedades son `pais_id` y `denominacion`. Necesitamos un `select` en un formulario, para que el usuario elija el país de su residencia.

Con muy poco, podemos generar diccionarios aptos para `add_select`:

```
function agregar($coleccion_paises=array()) {
    foreach($coleccion_paises as &$obj) {
          $obj->value = $obj->pais_id;
          $obj->text = $obj->denominacion;
    }
    # ...
```

```
}
```

Por favor, notar que la variable `$obj` **está siendo modificada por referencia. Caso contrario, los cambios se perderían al finalizar el bucle** `foreach`**.**

CollectorViewer: visualizando colecciones de objetos en una tabla HTML

Mostrar una colección de objetos en una tabla HTML, con botones ver, editar y eliminar, con filtro de búsqueda, paginado y ordenamiento de resultados por columnas, no requiere más de 4 líneas de código:

```
function listar($coleccion) {
    $tabla = CollectorViewer($coleccion, 'modulo', 'modelo')->get_table();
    print Template('Título de la Tabla')->show($tabla);
}
```

CollectorViewer generará una tabla HTML utilizando **jQuery DataTables** (librería JavaScript incluida en el paquete del *plug-in* CollectorViewer) para el paginado, filtro y orden de resultados por columnas.

Para instalarlo, basta con descargar el paquete **Europio_HTMLGrid_1_0_1_beta.tar.gz** desde el sitio Web de descargas de Europio Engine[2].

Luego de descomprimir el paquete, colocar la carpeta `collectorviewer` dentro del directorio `./common/plugins/` de la aplicación.

Para **habilitar el complemento** y que éste sea importado de forma automática por el motor de Europio, bastará con agregarlo en la variable `$enabled_apps` del archivo `settings.php`:

```
$enabled_apps = array('webform', 'collectorviewer');
```

Si no se lo habilita desde el `settings`, puede importarse luego de forma manual, desde el `user_imports.php` o desde la vista en la que se lo desee implementar, tal cual se ha explicado para el complemento anteriormente descrito.

Para finalizar, se debe copiar el contenido de la carpeta `./common/plugins/collectorviwer/static` en el directorio de archivos estáticos (por defecto: `./static`).

2 http://www.europio.org/downloads/plugins/

Ocultar los botones ver, editar y eliminar o mostrar solo alguno de ellos

Es posible que no desees mostrar alguno de los botones que `CollectorViewer` agrega por defecto a las tablas HTML generadas.

En ese caso, cuando se llame a `CollectorViewer`, se le deberá indicar `False` o `True` para ocultar o mostrar los botones ver, editar y eliminar respectivamente:

```
# Mostrar el botón Ver/descargar y ocultar el de editar y eliminar
$tabla = CollectorViewer($coleccion, 'modulo', 'modelo', True, False, False)->get_table();
```

Para ver la captura de pantalla de una **tabla generada con Europio Engine + CollectorViewer** ingresa en https://docs.google.com/file/d/0B-WmXm2YQ4sXRmJBUGItNUhZSUE/edit?usp=drivesdk

Seguridad Informática: Emulación de Tokens de Seguridad temporales como mecanismo de verificación en el registro de usuarios

Asegurarse de que el usuario que solicita el registro es humano y de que el e-mail indicado es realmente de su propiedad, es tan importante como mantener al resguardo la base de datos de usuarios ya confirmados. La emulación temporal de Tokens de Seguridad es el mecanismo más simple y eficiente para poder lograrlo, que se ha utilizado tradicionalmente.

Hace ya unas dos o tres semanas atrás, **Guillermo Montero** -uno de mis alumnos del curso de Análisis en PHP- me preguntaba sobre cómo implementar un **sistema de *Tokens* de Seguridad temporales en bases de datos** y de allí mi promesa de escribir este *paper* y dedicárselo, pues el tema, fue una idea suya, que seguramente resulte de interés a muchos programadores.

En el mundo de la Ingeniería de Software suele ser muy frecuente hablar de "*tokens*" como mecanismos de seguridad para efectuar validaciones de diversos tipos. Es importante aclarar que **un *token* de programación, es en realidad una palabra clave o identificador del lenguaje** mientras que **los *Tokens* de Seguridad, son dispositivos físicos** que electrónica o digitalmente almacenan información, que sirve para identificar al usuario que lo porta y son utilizados como medio de autenticación. **Cuando en programación, se habla de *Tokens* de Seguridad,** se lo hace en analogía a estos dispositivos y **usualmente se lo utiliza como sinónimo de "valor *hash*".**

Un valor *hash* es una cadena de longitud fija obtenida por una función H que efectúa un resumen (o compactación) de una cadena M, codificándola de manera irreversible. Se podría concluir entonces, en que el **proceso de emulación de un Token de Seguridad en programación**, es el resultado de una aplicación criptográfica, **H(M) = *valor hash.***

Generación del Token de Seguridad o valor hash

El valor Hash puede ser obtenido mediante un conjunto M a libre elección del programador. Por ejemplo, M podría ser la sumatoria del e-mail del usuario y el tiempo POSIX (*timestamp*). Mientras tanto, la función H podría ser MD5, SHA-512 o cualquier otra función *hash* segura:

```
# M = email + timestamp
$m = $email . Time();
$valor_hash = hash('sha512', $m);
```

Flujo de Procedimientos: ¿en qué consiste el mecanismo de verificación?

1. El flujo de procedimientos comienza con el usuario completando sus **datos de registro** mediante un formulario;
2. Cuando este formulario es enviado, se genera el ***Token* de Seguridad** y se lo almacena junto a los datos de registros en una **tabla de registro temporal**;
3. A continuación, se envía un **e-mail** al usuario con el *token* generado;
4. El usuario deberá ingresar en la aplicación (ya sea por GET o POST) el token recibido por e-mail, el cuál deberá ser **comparado con el *token* almacenado en la tabla de registro temporal**. En caso de coincidir, **se inserta el registro en la tabla de usuarios confirmados** y **se elimina el mismo, de la tabla temporal**.

Tabla de registro temporal

La tabla de registro temporal **debe ser un clon exacto de la tabla de usuarios confirmados**. Es importante que la tabla de usuarios confirmados cuente con un campo "token" y un *timestamp*, ya que incluso, estando confirmado un usuario, un nuevo token podría ser de utilidad para efectuar una restauración de contraseña.

Una vez creada la tabla principal (la de usuarios confirmados), puede obtenerse una copia exacta de la misma (un clon) con la siguiente sentencia SQL:

```
CREATE TABLE tmp_users LIKE users;
```

Comparación de Tokens e intercambio de registros

La comparación de Tokens e intercambio de registro, puede hacerse en un único paso, insertando el registro en tabla final, mediante una selección del mismo en la tabla temporal:

```
INSERT INTO users SELECT * FROM tmp_users WHERE token = ?
```

Si el resultado de la ejecución anterior es verdadero, entonces simplemente, se elimina el registro de la tabla temporal:

```
DELETE FROM tmp_users WHERE token = ?
```

Optimizando recursos

La optimización no solo de recursos sino también de procesos, forma parte de una buena política de seguridad:

> *cuanto más optimizada se encuentra una aplicación, más simple se hace la detección de errores y más se reducen los riesgos de fallos y de fugas de información*

Es imprescindible que la tabla destinada al registro temporal, sea verificada de forma periódica a fin de evitar la sobre-indexación de registros que pudiesen ser producto de intentos masivos y automatizados, de registros falsos. Limpiar la tabla temporal de falsos registros y optimizar sus índices y el espacio libre, previene la saturación de recursos y su consiguiente falta de disponibilidad.

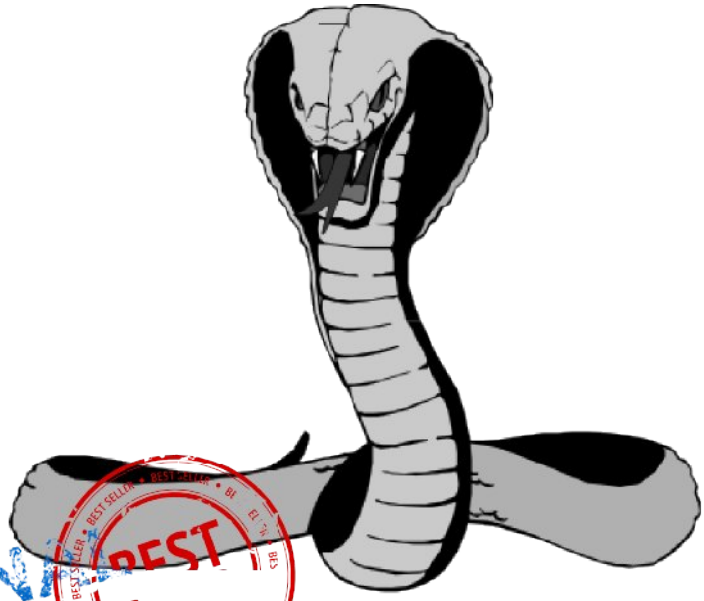
Se puede crear entonces, una tarea programada con `crontab`[3] que ejecute un archivo encargado de recorrer la tabla temporal, en busca de registros donde el *timestamp* del registro + 12 horas, por ejemplo, sea inferior al *timestamp* actual y a continuación, eliminar dichos registros obsoletos y optimizar la tabla:

```
#!/bin/bash

QUERY_ELIMINAR='DELETE FROM tmp_users WHERE TIMESTAMP(timestamp, '12:00:00') < NOW();'
QUERY_OPTIMIZAR='OPTIMIZE TABLE tmp_users;'

mysql -u usuario -pclave -e "$QUERY_ELIMINAR $QUERY_OPTIMIZAR"
```

Y finalmente:

```
# Agregar permisos de ejecución al archivo
chmod +x /ruta/a/script.sh

# Y agregar la tarea programada al cron con crontab
crontab -e
@daily /ruta/a/scipt.sh
```

3 Recomiendo leer el artículo "Actualizando tus aplicaciones con Cron" que escribí para la revista Hackers & Developers Magazine Nº6: http://www.eugeniabahit.com/static/pdf/LINUXSYSADMIN%20-%20Cron.pdf



The Original Hacker #1

SafeCreative Work: 1310258856007

safecreative.org/work/1310258856007