

La biblioteca estándar de Ruby

Claudio Bustos

28 de julio de 2008

1. Introducción

Uno de los grandes recursos que tiene Ruby es su gran biblioteca estándar, que reúne gran cantidad de módulos y clases tanto en extensiones en C como en Ruby, para múltiples necesidades. Si bien el libro "Programming Ruby: The Pragmatic Programmer's Guide" (<http://www.ruby-doc.org/docs/ProgrammingRuby/>) trae una excelente guía sobre los paquetes más usados, me pareció interesante hacer una rápida revisión de los archivos presentes en `/usr/lib/ruby/1.8/` (la dirección de la biblioteca estándar en Gentoo), y hacer una breve descripción de la utilidad de cada uno

2. Nivel principal: /

2.1. abbrev.rb

Objetivo: Calcula el conjunto de abreviaciones únicas para un conjunto dado de strings.

Utilidad: Para usos específicos. Podría ser útil para hacer un sistema de búsqueda incremental, por ejemplo.

Uso:

```
require 'abbrev'
require 'pp'
pp Abbrev.new(abbrev(['ruby', 'rules']).sort
  ==>[["rub", "ruby"],
      ["ruby", "ruby"],
      ["rul", "rules"],
      ["rule", "rules"],
      ["rules", "rules"]])
```

2.2. auto-reload.rb

Objetivo: Al incluirse, reemplaza el método `Kernel#require` para que se recarguen las bibliotecas modificadas automáticamente.

Utilidad: Muy útil si estás trabajando en entornos donde el script sólo se carga una vez, al iniciarse al proceso, como en webrick o modruby.

Uso:

```
# Incluir al inicio del archivo
require 'auto-reload'
```

2.3. base64.rb

Objetivo: El módulo `Base64` permite codificar y decodificar datos binarios en Base64. La ventaja de usar Base64 es que sólo utiliza caracteres imprimibles, por lo que es usado con frecuencia para enviar información por e-mail.

Utilidad: Muy útil si estas creando una clase para enviar adjuntos por e-mail o si quieres un método seguro de envío de información por un canal que posea dificultades para manejar algo que no sea ascii

Uso:

```
require 'base64'
enc = Base64.encode64("Esto es un mensaje en base64")
==>"RXN0byBlcyB1biBtZW5zYWplIGVuIGJhc2U2NA=="
plano = Base64.decode64(enc)
==>"Esto es un mensaje en base64"
```

2.4. benchmark.rb

Objetivo: El modulo *Benchmark* provee de métodos para medir la rapidez del código Ruby, dando un detallado reporte del tiempo ocupado en cada tarea.

Utilidad: Uno de los módulos indispensables, si quieres crear aplicaciones para producción.

Uso:

```
# Medir el tiempo que dura una operación específica
puts Benchmark.measure { "a"*1_000_000 }
```

```
0.030000  0.000000  0.030000 ( 0.022438)
```

```
# Comparar la rapidez de 3 métodos distintos. En estos casos, iterar de 1 a 50000
n = 50000
```

```
Benchmark.bm do |x|
x.report { for i in 1..n; a = "1"; end }
x.report { n.times do ; a = "1"; end }
x.report { 1.upto(n) do ; a = "1"; end }
end
```

```
      user      system      total      real
0.920000  0.010000  0.930000 ( 0.950809)
1.000000  0.010000  1.010000 ( 1.065786)
0.980000  0.010000  0.990000 ( 1.087133)
```

2.5. cgi-lib.rb

Obsoleto: Usar `cgi`

2.6. cgi.rb

Objetivo: Provee de soporte para CGI a Ruby, lo que le permite incorporar scripts ruby a un servidor HTTP, sin necesidad de tener un módulo incorporado, como `mod_ruby`.

Utilidad: Si no utilizas Ruby on Rails u otro framework para trabajo en Web, es de uso indispensable.

Más información: <http://www.ruby-doc.org/docs/ProgrammingRuby/html/web.html>

Uso:

```
require 'cgi'
# ver los valores enviados por un formulario en los campos 'player' y 'year'
cgi = CGI.new
cgi['player'] ==> ["Miles Davis"]
cgi['year'] ==> ["1958"]
```

2.7. complex.rb

Objetivo: Implementa la clase *Complex*, para trabajar con números complejos. Además, agrega métodos a las clases relacionadas con *Numeric* para hacer más fácil el trabajo con números complejos.

Utilidad: Si no trabajas en matemáticas, ninguna.

Más información: http://www.ruby-doc.org/docs/ProgrammingRuby/html/lib_standard.html#Complex.new

Uso:

```

require 'complex'
# Raiz cuadrada de -1
Math.sqrt(-1)
  ==>Complex(0, 1.0)
# Sumemos el complejo 1i con 5
a=Complex.new(0,1)+5
  ==>Complex(5, 1)
# Se puede ocupar el método Numeric.im para crear un imaginario a partir de un real
a+(2.im)
  ==>Complex(5, 3)

```

2.8. csv.rb

Objetivo: Permite leer y escribir archivos, strings y streams en formato CSV (archivos con valores separados por comas). Es el formato estándar para intercambiar planillas de cálculo u otros datos tabulados.

Utilidad: Extremadamente útil para leer o escribir información para Excel.

Uso:

```

CSV.open('csvfile.csv', 'r') do |row|
  p row
end

```

2.9. date2.rb

Obsoleto: date fue reemplazado por date2 y este a su vez fue renombrado a date

2.10. data.rb

Objetivo: Provee de dos clases para trabajar con fechas. La primera clase, *Date*, representa fechas desde años hasta días. La segunda clase, *DateTime*, extiende Date desde horas hasta fracciones de segundo

Utilidad: Casi todas las aplicaciones empresariales fechas, así que es muy probable que te topes con ella

Mas información: http://www.ruby-doc.org/docs/ProgrammingRuby/html/lib_standard.html#Date.exist2_qm

Uso:

```

require 'date'
d = Date.new(2000, 3, 31)
  ==>#<Date: 4903269/2,0,2299161>
[d.year, d.yday, d.wday]
  ==>[2000, 91, 5]
[d.month, d.mday]
  ==>[3, 31]

```

2.11. debug.rb

Objetivo: Provee las funcionalidades de 'debugueo' para Ruby. Generalmente, se activa desde la línea de comandos usando `ruby -r debug NOMBRE_SCRIPT.rb`

Utilidad: El debugger es una herramienta imprescindible en el set de herramientas de un programador, así que no te lo puedes perder

Mas información: <http://www.ruby-doc.org/docs/ProgrammingRuby/html/trouble.html>

2.12. delegate.rb

Objetivo: La clase *Delegator* y sus derivadas implementan el patrón Delegación ([http://en.wikipedia.org/wiki/Delegation_\(programming\)](http://en.wikipedia.org/wiki/Delegation_(programming))). En términos sencillos, se ocupa delegación cuando queremos que el comportamiento de un objeto cambie en tiempo de ejecución, a través del uso de los métodos de otro objeto

Utilidad: Es útil en aplicaciones donde determinados objetos, en función de una o más condiciones, deba cambiar completamente su comportamiento. Por ejemplo, si muchos de los métodos de nuestro objeto tienen secuencias del tipo

```
def accion
  if(estado==1) then puts 'estoy bien' end
  if(estado==2) then puts 'estoy mas o menos' end
end
```

es recomendable evaluar si este patrón puede ser útil

Mas información: http://www.ruby-doc.org/docs/ProgrammingRuby/html/lib_patterns.html

Uso:

```
require 'delegate'
```

Definimos dos clases, que presentan distinto comportamiento

```
class Sano
  def caminar
    puts "Camina muy rapido"
  end
  def dormir
    puts "Duerme 8 horas"
  end
end
class Enfermo
  def caminar
    puts "Camina apenas"
  end
  def dormir
    puts "Duerme 16 horas"
  end
end
```

Creamos una clase llamada *Persona*, que ocupa la clase *SimpleDelegator*. Al utilizar el método `__setobj__`, el objeto instanciado de esta clase se comporta para todos los efectos como si fuera la clase delegada

```
class Persona < SimpleDelegator
  def initialize
    @sano=Sano.new
    @enfermo=Enfermo.new
    super(@sano)
  end
  def enfermar
    __setobj__(@enfermo)
  end
  def sanar
    __setobj__(@sano)
  end
end
```

A partir de ahora, creamos un objeto de clase *Persona* y lo hacemos enfermar y sanarse, de acuerdo a nuestras necesidades

```
persona = Persona.new
=> #<Sano:0xb77e5074>
persona.caminar
Camina muy rapido
persona.dormir
Duerme 8 horas
persona.enfermar
=> #<Enfermo:0xb77e5060>
persona.caminar
Camina apenas
persona.dormir
Duerme 16 horas
```

2.13. digest.rb

Objetivo: El módulo *Digest* funciona como un façade de las distintas módulos que proveen de funciones de hash en Ruby. Entre los algoritmos de hash más usados tenemos MD5 y SHA2

Utilidad: Si necesitas verificar que un archivo o cadena es distinta a otra y necesitas guardar este dato, los hash son una herramienta muy poderosa

Uso:

```
require 'digest'
Digest::MD5
  ==>nil
Digest::MD5
  ==>Digest::MD5
Digest::SHA2
  ==>Digest::SHA2
Digest::NOEXISTE
LoadError: library not found for class Digest::NOEXISTE -- digest/noexiste
  from (irb):12
# Hash MD5 de 16 bytes
Digest::MD5.digest("hola")
  ==>"M\030c!\301\247\360\363T\262\227\350\221J\262@"
# Hash MD5 expresado en hexadecimal (32 bytes)
Digest::MD5.hexdigest("hola")
  ==>"4d186321c1a7f0f354b297e8914ab240"
```

2.14. drb.rb

Objetivo: El módulo *DRb* provee de las funcionalidades necesarias para implementar un sistema de objetos distribuido. Esto permite que un computador funcione como servidor de objetos para el uso de otro computador

Utilidad: Si realizas computación distribuida, parte por acá!

Más información: <http://www.ruby-doc.org/docs/ProgrammingRuby/html/ospace.html>

2.15. e2mmap.rb

Objetivo: No tengo idea....

2.16. English.rb

Objetivo: Define alias en inglés para las variables locales más utilizadas.

Utilidad: Si quieres que tu script se vea como PHP y no como Perl, puedes hacerlo fácilmente!

Uso:

```
$\ = ' -- '
  "waterbuffalo" =~ /buff/
  print $, $', $$, "\n"
```

Con el módulo English:
require "English"

```
$OUTPUT_FIELD_SEPARATOR = ' -- '
"waterbuffalo" =~ /buff/
print $LOADED_FEATURES, $POSTMATCH, $PID, "\n"
```

2.17. Env.rb

Objetivo: Importa las variables del ambiente como variables globales.

Utilidad: Si trabajas en línea de comandos, te puede ser útil

2.18. erb.rb

Objetivo: El módulo *ERB* provee de un sistema de plantillas para ruby.

Utilidad: Provee de variadas utilidades. Entre las más básicas, proveer de un sistema de plantillas para un sitio web, así como incorporar variables determinadas dentro de un archivo como un RTF o un PDF

Uso:

```
require 'erb'
a=5
b=5
plantilla=ERB.new("El valor de a es <%=a %> y de b es <%=b %>")
==>#<ERB:0xb77381bc @safe_level=nil, @filename=nil, @src="_erbout = ''"; _erbout.conca
puts plantilla.result(binding)
El valor de a es 5 y de b es 5
```

2.19. expect.rb

Objetivo: Provee del método *expect* a la clase *IO*. Los parámetros de este método son un string o expresión regular y un tiempo máximo de espera. Al correr este método, se ingresa en un ciclo infinito hasta que el stream ofrezca la cadena o expresión regular deseada

Utilidad: Puede ser bastante útil cuando se establece un sistema cliente servidor.

Uso: En mi directorio tengo un archivo llamado `last.fm`, que contiene la cadena 'last'

```
require 'expect'
fp=File.open("lastfm.rb", "r")
==>#<File:lastfm.rb>
fp.expect("last")
==>["#!/bin/env ruby\n# \n# Graficador de artistas escuchados, registrados por last"]
```

2.20. fileutils.rb

Objetivo: Provee del módulo *FileUtils*, con variadas funcionalidades para copiar, modificar y remover archivos y directorios

Utilidad: Indispensable para cualquier trabajo serio con archivos

Listado de funciones: Las funciones que provee FileUtils son las siguientes:

- `cd(dir, options)`
- `cd(dir, options) {—dir— }`
- `pwd()`
- `mkdir(dir, options)`
- `mkdir(list, options)`
- `mkdir_p(dir, options)`
- `mkdir_p(list, options)`
- `rmdir(dir, options)`
- `rmdir(list, options)`
- `ln(old, new, options)`
- `ln(list, destdir, options)`
- `ln_s(old, new, options)`
- `ln_s(list, destdir, options)`
- `ln_sf(src, dest, options)`
- `cp(src, dest, options)`
- `cp(list, dir, options)`
- `cp_r(src, dest, options)`
- `cp_r(list, dir, options)`
- `mv(src, dest, options)`
- `mv(list, dir, options)`
- `rm(list, options)`
- `rm_r(list, options)`

- `rm_rf(list, options)`
- `install(src, dest, mode = :src'sj, options)`
- `chmod(mode, list, options)`
- `chmod_R(mode, list, options)`
- `chown(user, group, list, options)`
- `chown_R(user, group, list, options)`
- `touch(list, options)`

2.21. finalizer.rb

Objetivo: Define funciones especiales para el finalizer del *ObjectSpace*. Realmente, no sé para que sirve

2.22. find.rb

Objetivo: El módulo *Find* permite navegar dentro de un directorio recursivamente

Utilidad: Si necesitas buscar dentro de un conjunto de directorios, puede ser bastante útil

Uso:

```
require 'find'
Find.find("/home/usuario/") do |path|
  # Salto los directorio que se inicien con '.'
  if FileTest.directory?(path) and File.basename(path) =~ /^\.\/
    Find.prune
  else
    puts path
  end
end
```

2.23. forwardable.rb

Objetivo: El módulo *Forwardable* permite delegar métodos a un objeto, método a método. Se puede utilizar *Forwardable* para definir esta delegación al nivel de clase o *SingleForwardable* al nivel de objeto.

Es una alternativa a la delegación (ver 2.12) o a la herencia simple

Utilidad: Es un método interesante para definir clases u objetos que deben tener funcionalidades limitadas de una clase

Uso:

```
require 'forwardable'
# Definimos un ave genérica, que aletea y vuela
class Ave
  def aletea
    puts "El pajaro aletea!"
  end
  def vuela
    puts "El pajaro vuela!"
  end
end
# Tenemos un Kiwi, un ave especial que solo aletea, pero no vuela
class Kiwi
  extend Forwardable
  def initialize
    @q = Ave.new
  end
  def_delegator :@q, :aletea
end
k=Kiwi.new
  ==>#<Kiwi:0xb78c39dc @q=#<Ave:0xb78c39b4>>
# Nuestro kiwi claramente aletea
k.aletea
El pajaro aletea!
# Pero no vuela!
k.vuela
NoMethodError: undefined method 'vuela' for #<Kiwi:0xb78c39dc @q=#<Ave:0xb78c39b4>>
```

2.24. ftools.rb

Objetivo: Este archivo agrega varios métodos a la clase *File*. El autor recomienda utilizar Fileutils(Ver [2.20](#))

2.25. generator.rb

Objetivo: Transforma un iterador interno en uno externo. Esto es, lo transforma en un objeto independiente, de manera análoga a los iteradores en Java.

Utilidad: Si bien un iterador externo en ruby es casi un sacrilegio, da la interesante oportunidad de navegar por varias enumeraciones de forma simultánea de forma sincronizada

Uso:

```
require 'generator'
# Iterador solitario
g = Generator.new(['A', 'B', 'C', 'Z'])

while g.next?
  puts g.next
end

# Sincronización de enumeradores

s = SyncEnumerator.new([1,2,3], ['a', 'b', 'c'])
# Yields [1, 'a'], [2, 'b'], and [3, 'c']
s.each { |row| puts row.join(', ') }
```