Ruby: La Joya de los lenguajes

Enrique Meza
Jarabe Software

emeza@jarabekm.com

Ruby creado por *Yukihiro Matsumoto* es otro lenguaje de programación orientado a objetos. Nació en 1993 cuando el autor no pudo encontrar un lenguaje de guiones (scripting) orientado a objetos y decidió escribir uno. El nombre de Ruby se eligió porque el autor buscaba otro nombre de joya que simbolizase la cercanía del lenguaje a Perl (" La perla de los lenguajes").

Tabla de contenidos

Introducción **3**
Todo es un Objeto **4**
Conclusiones y más documentación **7**

Introducción

¿ Que es Ruby ?

- Es un lenguaje orientado a objetos
- Es un lenguaje de guiones
- Es un lenguaje de ensamble
- Es un lenguaje de transparente.

Ruby es orientado a objetos desde su concepción :

Ruby toma la orientación a objetos pura de Smalltalk, pero quíta la desigual sintaxis y la confianza en un espacio de trabajo. Añade la conveniencia y el poder de Perl, pero sin todos esos casos especiales y mágicas conversiones. Envuélve en una sintaxis limpia basada en parte en Eiffel, y añade unos cuantos conceptos de Scheme, CLU, Sather, y Common Lisp. Ruby es un lenguaje que ya es más popular que Python en su Japón nativo.

Ruby es tan fuerte en las características de "scripting" como lo es Perl ya que tiene construidas las expresiones regulares compatibles a Perl 5. Tiene prácticamente todas las funciones de Perl con una sintaxis más simple.

Puede acceder a todas las llamadas "system calls" de Unix (vía las syscall al menos). Puede acceder a todas las "system calls" en Win32 (vía la Win32API)

Ruby es "transparente" como lenguaje ya que permite enfocarse directamente en el problema a resolver ya que permite iniciar la solución desde los níveles más altos de la abstracción. Ruby permite iniciar la codificación con las mínimas situaciones de error .

Ruby al menos es adecuado para las siguientes tareas:

- Procesamiento de textos
- Programación XML
- Aplicaciones GUI: Tk, GTK, Qt, etc.
- Para aplicaciones académicas por ejemplo matemáticas,física,astronomía,etc.
- Programación en General: Acceso a DBMS, PalmPilot,Web, etc.

Veamos ahora algunos ejemplos :

Primer Ejemplo

```
#! /usr/local/bin/ruby -w
a = %w( gato perro perico tamagochi )    # se crea un arreglo
a.each { |mascota| puts mascota }            # se iterate so-
bre el contenido
```

Segundo Ejemplo

```
#! /usr/local/bin/ruby -w
f = open("/etc/passwd")
while renglon = f.gets()
   print renglon if /emeza/ =~ renglon
end
```

Re-escribimos el Segundo Ejemplo

```
       #! /usr/local/bin/ruby -w
       File.foreach("/etc/passwd") do |renglon|
          print renglon if /emeza/ =~ renglon
end
```

Re-escribimos el Segundo Ejemplo

```
       #! /usr/local/bin/ruby -w
        puts File.open("/etc/passwd").grep(/emeza/)
```

Ejemplo de un Reporte

```
# data.txt
UK , 44 , Pound , 1.85 , 100
BELGIUM , 32 , Franc , 32.0 , 200
DENMARK , 45 , Krone , 6.0 , 2000
FINLAND , 358 , Markka , 4.69, 1000
FRAncE , 33 , Franc , 5.28, 50
ELSALVADOR , 503 , Colon , 8.74, 340
PHILIPPINES , 63 , Peso , 24.8, 1000
PAKISTAN , 92 , Rupee , 38.0, 1200
BAHRAIN , 973 , Dinar , 0.38 , 45
IRAQ , 964 , Dinar , 0.60, 10
JORDAN , 962 , Dinar , 0.70, 100
SAUDIARABIA , 966 , Riyal , 3.75, 1000

# Report.rb
#!/usr/local/bin/ruby -w

f = open("data.txt")
sum=0
par = "================================================\n"
aString = " Id Pais  Moneda   Taza    Cantidad Valor
=============================================================================\n"

print aString

while line = f.gets()
 pais, idi,moneda, taza, cantidad  = line.split(",")
 sum += cantidad.to_i/taza.to_f
 printf "%3i   %15s  %8s  %5.3f %4.2f %4.2f\n",idi,pais,moneda,taza,cantidad,cantidad.t
end
print par
print "Valor Total = $",sum,"\n"
print par
```

Todo es un Objeto

Ruby fué desarrollado teniendo en cuenta el principio de ortogonalidad :

- Cada dato es un objeto
- Cada Objeto es una instancia de una clase
- Cada clase es un objeto a su vez
- Cada clase tiene su metaclase.

Ejemplo 3

```
#! /usr/local/bin/ruby -w
# Definicion basica de una clase
class Vaca
    def muje   #  método que hace mujir a las vacas
        print "Muuuu :   Dos patas no!! \nCuatro patas si !!\n"
    end
end
VacaPinta = Vaca.new   # Creamos una vaca pinta
VacaPinta.muje                #Antes de cantar "Bestias de Inglaterra"
```

Ejemplo 4

```
#! /usr/local/bin/ruby -w
# Creamos un segundo metodo para simplificar
class Vaca  # 'mujido' es un metodo, no una variable
           def mujido
                    "moooo"
           end
          def muje
                  print "Esta vaca muje asi...  #{mujido}!\n"
           end
 end

VacaSuiza = Vaca.new
VacaSuiza.muje
```

Ejemplo 5

```
#! /usr/local/bin/ruby -w
# Herencia de métodos de una superclase
class Animal_de_Cuatro_patas
    def chilla
        print "Un #{self.class} chilla... #{chillido}!\n"
    end
 end
class Cerdo < Animal_de_Cuatro_patas
           def chillido
                    "Las Bestias de Inglaterra"
           end
end

Napoleon = Cerdo.new
Napoleon.chilla
SnowBall = Cerdo.new
SnowBall.chilla
```

Ejemplo 6

```
#!/usr/local/bin/ruby

class Animal_de_Cuatro_Patas
   def initialize(nombre, color)
     if nombre == ""
       @nombre = "un #{self.class} sin_nombre"
     else
       @nombre = nombre
     end
```

```
    if color == ""
      @color = default_color # Igual que self.default_color
    else
      @color = color
    end
  end

  def habla
    print @nombre, " dice :  ...", sonido, "\n"
  end

  def come(alimento)
    @alimento = alimento
    print @nombre, " come ", alimento, "\n"
  end

  def default_color; " azabache "; end

  def muestra_color
    print self.class, ": ",  @nombre, " tiene color #{@color}!\n"
  end
end

class Caballo < Animal_de_Cuatro_Patas
  def sonido; "Hoy Trabajaré más duro...."; end
end

class Oveja < Animal_de_Cuatro_Patas
  def sonido; "Dos patas Si, Cuatro Patas No"; end
  def default_color; "blanco"; end
end

unaOveja = Oveja.new( "Oveja_Rasurada", "negro")
unaOveja.muestra_color
unaOveja.habla

pequenaOveja = Oveja.new( "", "")
pequenaOveja.muestra_color
pequenaOveja.habla

otraOveja = Oveja.new( "Oveja_Sin_Lana", "") # Usa el Color de defecto de la Oveja
otraOveja.muestra_color
otraOveja.come("Pasto")

unCaballo = Caballo.new("","") # Usa el Color de Default del Animal_de_Cuatro_Patas
unCaballo.muestra_color
unCaballo.habla

otroCaballo = Caballo.new("Boxer","café") # Usa el Color de Default del  Animal_de_Cua
otroCaballo.muestra_color
otroCaballo.come("Alfalfa")
```

Clases y Métodos

Como muestra el ejemplo 1, las definiciones de clases en Ruby son notablemente simples: la palabra clave class seguida por el nombre de la clase, el cuerpo de la clase, y la palabra clave end para terminar todo. Ruby permite la herencia simple: cada clase tiene exactamente una superclase, que puede ser especificada como muestra el ejemplo 2. Una clase sin padre explícito se considera hija de la clase Object –la raíz de la jerarquía de clases, la cual es la única que no tiene superclases. Si te preocupa

que un modelo de herencia simple no sea suficiente, no temas. Hablaremos sobre las capacidades de mezclado ("mixin") en breve. Ejemplos[1]

Conclusiones y más documentación

Bloques e iteradores ¿Has querido alguna vez escribir tus propias estructuras de control, o empaquetar trozos de código dentro de un objeto? Los bloques de código de Ruby te permiten hacer justamente eso. Un bloque es simplemente un trozo de código entre llaves, o entre las palabras clave do y end. Cuando Ruby llega a un bloque, almacena el código del bloque para más tarde; el bloque no es ejecutado. De esta forma, un bloque es similar a un método anónimo. Los bloques sólo pueden aparecer en un fuente Ruby junto a llamadas a métodos. Un bloque asociado con una llamada a un método pueden ser invocado desde dentro de ese método. Esto puede parecer inocuo, pero esta sola facilidad te permite escribir callbacks y adaptadores, manejadores de transacciones, e implementar tus propios iteradores. Los bloques son además verdaderas clausuras, que recuerdan el contexto en el que fueron definidos, incluso si el contexto se ha salido del ámbito. Veamos los bloques simplemente como iteradores por ahora.

Módulos, Mezclas y Herencia Múltiple Los módulos son clases que no puedes instanciar: no puedes usar new para crear objetos desde ellos, y no pueden tener superclases. De entrada, parecen inútiles, pero en realidad los módulos tienen dos usos principales. Los módulos proporcionan un espacio de nombres. Las constantes y los métodos de clase pueden ser colocados en un módulo sin preocuparse de conflictos de nombre con constantes y métodos de otras clases en otros módulos. Esto es similar a la idea de utilizar métodos y variables estáticas en una clase de Java. En Java y Ruby puedes escribir Math.PI para acceder al valor de (aunque en Ruby, PI es una constante, en lugar de una variable final, y verás más comúnmente la notación Math::PI). Los módulos son además la base de las mezclas5, un mecanismo por el cual puedes añadir comportamiento "enlatado" a tus clases. Quizás la forma más fácil de pensar sobre las mezclas es imaginar que puedes escribir código en una interface de Java. Toda clase que implementa esa interface no sólo puede recibir un tipo de signatura; también puede recibir el código que implementa la signatura. Podemos investigar esto observando el módulo Enumerable, que añade métodos basados en colecciones a clases que implementan el método each.

Otro buen material Este artículo es demasiado corto para hacer justicia a todo lo que ofrece Ruby. No obstante, daremos un breve toque a algunas otras cuestiones a destacar. Las clases y los módulos nunca están cerrados. Puedes añadir y alterar cosas de todas las clases y módulos (incluso aquellos creados de fábrica en el mismo Ruby). Reflexiones. Además de soportar la reflexión en clases y objetos individuales, Ruby te permite recorrer la lista de objetos actualmente activos. Marshalling. Los objetos de Ruby pueden ser "serializados" y "des-serializados", permitiendo ser almacenados externamente y transmitidos a través de redes. Un completo sistema de objetos distribuidos, DRb, está escrito en alrededor de 200 líneas de código Ruby. Librerías. Ruby tiene una gran (y creciente) colección de librerías. Todos los protocolos principales de Internet están soportados, así como la mayoría de las principales bases de datos. Extender Ruby es simple comparado con (digamos) Perl. Hilos. Ruby tiene soporte interno para hilos, y no requiere del sistema operativo subyacente para soportar los hilos. Especialización de objetos. Puedes añadir métodos a objetos individuales, no sólo a clases. Esto puede ser útil cuando se definen comportamientos especializados para objetos (por ejemplo, determinando cómo deben responder a eventos del GUI). Excepciones. Ruby tiene un modelo de excepciones completamente orientado a objetos, y extensible. Recogida de basura. Los objetos Ruby son automáticamente eliminados de la memoria mediante un algoritmo de marcar-y-barrer. Además de simplificar la programación, la elección de marcar y barrer permite que la escritura

de extensiones sea más sencilla (no hay problemas de conteo de referencias). Comunidad activa de desarrolladores. La comunidad de desarrolladores de Ruby es aún un bazar, pequeño, íntimo y bullicioso. Los cambios son discutidos abiertamente y se realizan eficientemente. Puedes tener impacto sobre Ruby.

Conclusión Programar en Ruby es una experiencia inmensamente satisfactoria –el lenguaje parece ser capaz de representar conceptos de alto nivel de manera concisa, eficiente y legible. Es fácil de aprender, y al mismo tiempo es bastante profundo incluso para el más agotado recolector de lenguajes. Descarga una copia y pruébalo por tí mismo. Creemos que te gustará.

Ligas Importantes

Página Oficial de Ruby[2]

RubyCentral everything to do with the Ruby language.[3]

Bibliografía

Notas

1. http://ruby.jarabekm.com/taller.html
2. http://www.ruby-lang.org
3. http://www.rubycentral.com/

1. http://www.pragmaticprogrammer.com/ruby/downloads/index.html