Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, creado por el programador japonés Yukihiro «Matz» Matsumoto, quien comenzó a trabajar en Ruby en 1993, y lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl con características de programación orientada a objetos similares a Smalltalk. Comparte también funcionalidad con otros lenguajes de programación como Lisp, Lua, Dylan y CLU. Ruby es un lenguaje de programación interpretado en una sola pasada y su implementación oficial es distribuida bajo una licencia de software libre.

El lenguaje fue creado por Yukihiro «Matz» Matsumoto, quien empezó a trabajar en Ruby el 24 de febrero de 1993, y lo presentó al público en el año 1995. En el círculo de amigos de Matsumoto se le puso el nombre de «Ruby» (en español *rubí*) como broma aludiendo al lenguaje de programación «Perl» (*perla*). La última versión estable de la rama 1.8 es la 1.8.7_p248, de la rama 1.9 es la 1.9.2_p180 . La versión en 1.9 que incorpora mejoras sustanciales en el rendimiento del lenguaje, que se espera queden reflejadas en la próxima versión estable de producción del lenguaje, Ruby 1.9.0.1 Diferencias en rendimiento entre la actual implementación de Ruby (1.8.6) y otros lenguajes de programación más arraigados han llevado al desarrollo de varias máquinas virtuales para Ruby. Entre ésas se encuentra JRuby, un intento de llevar Ruby a la plataforma Java, y Rubinius, un intérprete modelado basado en las máquinas virtuales de Smalltalk. Los principales desarrolladores han apoyado la máquina virtual proporcionada por el proyecto YARV, que se fusionó en el árbol de código fuente de Ruby el 31 de diciembre de 2006, y se dió a conocer como Ruby 1.9

El creador del lenguaje, Yukihiro «Matz» Matsumoto, ha dicho que Ruby está diseñado para la productividad y la diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. Sostiene que el diseño de sistemas necesita enfatizar las necesidades humanas más que las de la máquina:

> A menudo la gente, especialmente los ingenieros en computación, se centran en las máquinas. Ellos piensan, «Haciendo esto, la máquina funcionará más rápido. Haciendo esto, la máquina funcionará de manera más eficiente. Haciendo esto…»

> Están centrados en las máquinas, pero en realidad necesitamos centrarnos en las personas, en cómo hacen programas o cómo manejan las aplicaciones en los ordenadores. Nosotros somos los jefes. Ellos son los esclavos.

Ruby sigue el «principio de la menor sorpresa», lo que significa que el lenguaje debe comportarse de tal manera que minimice la confusión de los usuarios experimentados. Matz ha dicho que su principal objetivo era hacer un lenguaje que le divirtiera a él mismo, minimizando el trabajo de programación y la posible confusión.

Él ha dicho que no ha aplicado el principio de menor sorpresa al diseño de Ruby,pero sin embargo la frase se ha asociado al lenguaje de programación Ruby. La frase en sí misma ha sido fuente de controversia, ya que los no iniciados pueden tomarla como que las características de Ruby intentan ser similares a las características de otros lenguajes conocidos. En mayo de 2005 en una discusión en el grupo de noticias comp.lang.ruby, Matz trató de distanciar Ruby de la mencionada filosofía, explicando que cualquier elección de diseño será sorprendente para alguien, y que él usa un estándar personal de evaluación de la sorpresa. Si ese estándar personal se mantiene consistente habrá pocas sorpresas para aquellos familiarizados con el estándar.

Matz lo definió de esta manera en una entrevista :

> Todo el mundo tiene un pasado personal. Alguien puede venir de Python, otro de Perl, y pueden verse sorprendidos por distintos aspectos del lenguaje. Entonces podrían decir ‘Estoy sorprendido por esta característica del lenguaje, así que Ruby viola el principio de la menor sorpresa.’ Espera, espera. El principio de la menor sorpresa no es sólo para ti. El principio de la menor sorpresa significa el principio de ‘mi’ menor sorpresa. Y significa el principio de la menor sorpresa después de que aprendes bien Ruby. Por ejemplo, fui programador de C++ antes de empezar a diseñar Ruby. Programé exclusivamete en C++ durante dos o tres años. Y después de dos años de programar en C++, todavía me sorprendía.

Ruby es orientado a objetos: todos los tipos de datos son un objeto, incluidas las clases y tipos que otros lenguajes definen como primitivas, (como enteros, booleanos, y «nil»). Toda función es un método. Las variables siempre son referencias a objetos, no los objetos mismos. Ruby soporta herencia con enlace dinámico, mixins y métodos singleton (pertenecientes y definidos por un sola instancia más que definidos por la clase). A pesar de que Ruby no soporta herencia múltiple, la clases pueden importar módulos como mixins. La sintaxis procedural está soportada, pero todos los métodos definidos fuera del ámbito de un objeto son realmente métodos de la clase Object. Como esta clase es padre de todas las demás, los cambios son visibles para todas las clases y objetos.

Ruby ha sido descrito como un lenguaje de programación multiparadigma: permite programación procedural (definiendo funciones y variables fuera de las clases haciéndolas parte del objeto raíz Object), con orientación a objetos, (todo es un objeto) o funcionalmente (tiene funciones anónimas, clausuras o closures, y continuations; todas las sentencias tiene valores, y las funciones devuelven la última evaluación). Soporta introspección, reflexión y metaprogramación, además de soporte para hilos de ejecución gestionados por el intérprete. Ruby tiene tipado dinámico, y soporta polimorfismo de tipos (permite tratar a subclases utilizando la interfaz de la clase padre). Ruby no requiere de polimorfismo de funciones al no ser fuertemente tipado (los parámetros pasados a un método pueden ser de distinta clase en cada llamada a dicho método).

De acuerdo con las preguntas frecuentes de Ruby, «Si te gusta Perl, te gustará Ruby y su sintaxis. Si te gusta Smalltalk, te gustará Ruby y su semántica. Si te gusta Python, la enorme diferencia de diseño entre Python y Ruby/Perl puede que te convenza o puede que no.»

Características

- orientado a objetos

- cuatro niveles de ámbito de variable: global, clase, instancia y local.
- manejo de excepciones
- iteradores y clausuras o closures (pasando bloques de código)
- expresiones regulares nativas similares a las de Perl a nivel del lenguaje
- posibilidad de redefinir los operadores (sobrecarga de operadores)
- recolección de basura automática
- altamente portable
- Hilos de ejecución simultáneos en todas las plataformas usando *green threads*
- Carga dinámica de DLL/bibliotecas compartidas en la mayoría de las plataformas
- introspección, reflexión y metaprogramación
- amplia librería estándar
- soporta inyección de dependencias
- soporta alteración de objetos en tiempo de ejecución
- continuaciones y generadores

Ruby actualmente no tiene soporte completo de Unicode, a pesar de tener soporte parcial para UTF-8.

Interacción

La distribución oficial de Ruby incluye «irb»(Interactive Ruby Shell), un intérprete interactivo de línea de comandos que puede ser usado para probar código de manera rápida. El siguiente fragmento de código representa una muestra de una sesión usando irb:

```
$ irb
irb(main):001:0> puts "Hola mundo"
Hola mundo
=> nil
irb(main):002:0> 1+2
```

```
=> 3
```

La sintaxis de Ruby es similar a la de Perl o Python. La definición de clases y métodos está definida por palabras clave. Sin embargo, en Perl, las variables no llevan prefijos. Cuando se usa, un prefijo indica el ámbito de las variables. La mayor diferencia con C y Perl es que las palabras clave son usadas para definir bloques de código sin llaves. Los saltos de línea son significativos y son interpretados como el final de una sentencia; el punto y coma tiene el mismo uso. De forma diferente que Python, la indentación no es significativa.

Una de las diferencias entre Ruby y Python y Perl es que Ruby mantiene todas sus variables de instancia privadas dentro de las clases y solo la expone a través de métodos de acceso (attr_writer, attr_reader, etc). A diferencia de los métodos «getter» y «setter» de otros lenguajes como C++ o Java, los métodos de acceso en Ruby pueden ser escritos con una sola línea de código. Como la invocación de estos métodos no requiere el uso de paréntesis, es trivial cambiar una variable de instancia en una función sin tocar una sola línea de código o refactorizar dicho código.

Los descriptores de propiedades de Python son similares pero tienen una desventaja en el proceso de desarrollo. Si uno comienza en Python usando una instancia de variable expuesta públicamente y después cambia la implementación para usar una instancia de variable privada expuesta a través de un descriptor de propiedades, el código interno de la clase necesitará ser ajustado para usar la variable privada en vez de la propiedad pública.

Ruby elimina está decisión de diseño obligando a todas las variables de instancia a ser privadas, pero también proporciona una manera sencilla de declarar métodos set y get. Esto mantiene el principio de que en Ruby no se puede acceder a los miembros internos de una clase desde fuera de esta; en lugar de esto se pasa un mensaje (se invoca un método) a la clase y recibe una respuesta.

Licencia

El intérprete y las bibliotecas están licenciadas de forma dual (inseparable) bajo las licencias libres y de código abierto GPL y Licencia pública Ruby.

A partir de la versión 1.9.3 se opta por una licencia dual bajo las licencias BSD de dos cláusulas y Licencia pública Ruby.