Un tour por Java, Scala, Python, Ruby y Javascript

13 de agosto de 2010

Resumen

Introducción a elementos y técnicas de programación de 5 lenguajes, motivadas por un problema en que un cierto servicio remoto puede fallar en forma aleatoria. Las soluciones son comparadas por el nivel de expresividad que da cada lenguaje para que la implementación sea lo mas declarativa posible.

Palabras Claves: Código Abierto, Linux, Creative Commons, Lenguajes de programación, Java, Scala, Ruby, Python, Javascript.

Introducción

En los tiempos actuales proliferan los lenguajes de programación. El índice TIOBE[1] lista al menos 100 lenguajes usados por la industria. Es interesante notar como, para casi todos los lenguajes de la lista de los 20 mas populares, existen implementaciones open source. De hecho, en muchos casos, la evolución del lenguaje va de la mano de un proyecto open source, como es el caso de PHP, Python, Perl, Ruby, Clojure y Go.

Motivación

Cada lenguaje tiene sus fortalezas y debilidades. Existen lenguajes muy expresivos y productivos, pero que no tienen un rendimiento muy impresionante en tiempo de ejecución. Existen otros que balancean el contar con abstracciones modernas como POO sin sacrificar eficiencia. Y otros en que poco importan sus limitaciones si los efectos de red de contar con una comunidad enorme compensan cualquier problema de diseño o de rendimiento que pudiera existir.

En este escenario, un desarrollador debe saber escoger el lenguaje adecuado para cada problema. Este trabajo da una mirada a varios lenguajes, con un enfoque práctico dirigido por un problema concreto extraído de un proyecto real, y destacando construcciones básicas de cada lenguaje tanto en el paradigma de la programación orientada a objetos como de programación funcional.

Desarrollo del Tema

El problema escogido para dirigir el tour es uno muy sencillo: un cierto servicio falla en forma frecuente por lo que se debe de reintentar la llamada en repetidas ocasiones, esperando un cierto tiempo antes de dicho reintento. El problema está inspirado en el contexto real de servicios webs que limitan la cantidad de peticiones en un período de tiempo, como Google Maps.

El objetivo es que la solución sea reusable para reintentar cualquier llamada a un servicio o API y que el código sea lo mas claro posible.

Se desarrollaron soluciones en: Java, Scala, Python, Ruby y Javascript (todos lenguajes donde las implementaciones de referencia son open source). Dichas soluciones están disponibles en `http://tinyurl.com/2cr2vrl`. A continuación se detallan los puntos a destacar en cada implementación:

Java

- Se intentan soluciones en dos paradigmas: funcional y orientado a objetos. El funcional resulta más cercano al problema, pero el orientado a objetos es mas fácil de implementar en el lenguaje.
- Se muestra la técnica de encadenamiento de métodos para hacer el API mas legible.

Scala

- Tipado fuerte, pero con inferencia de tipos para que el desarrollador no deba especificarlos todo el tiempo.
- El lenguaje mezcla los paradigmas funcional y orientado a objetos. Se escogen las primitivas funcionales.
- La solución usa los "endulzantes sintácticos" de Scala para que la funcionalidad de reintentos se asemeje a una construcción incluida en el lenguaje mismo.

Python

- Sintaxis concisa. El problema se resuelve de manera más simple en un lenguaje dinámico.

Ruby

- Similar a Python, pero con distintas primitivas del lenguaje. Comparación de dichas primitivas.
- Al igual que en el caso de Scala, la solución da la impresión de que es parte del propio lenguaje.

Javascript

- ¡Se pueden escribir programas de propósito general en Javascript, usando un intérprete como Mozilla Spidermonkey o Google V8 (ambos open source)!
- Solución más similar a Python/Ruby que a Java. El nombre es engañoso.

Conclusión

Como conclusión se revisan las abstracciones que encajan con el problema y cuales son implementadas por qué lenguaje de los vistos:

- Funciones de primera clase: Scala, Python y Javascript.
- Funciones anónimas multi-línea: Scala y Javascript.
- Sintaxis especial para bloques: Ruby (y en cierta medida Scala, pero mediante "endulzante sintáctico").
- *Matching* de excepciones dinámico: Python y Ruby.
- Parámetros con nombre ("*keyword* arguments"): Scala y Python.
- Extensión de objetos predefinidos: Scala y Ruby.

En consecuencia, ni siquiera para este acotado problema hay un solo lenguaje ideal. Scala suma la mayor cantidad de abstracciones útiles, pero la implementación en Python es la mas concisa, seguida de cerca por la implementación en Ruby, que cuenta con la ventaja de funcionar mejor cuando el "servicio" consiste en mas de úna línea de código.

Y así es la vida. No hay *un* lenguaje ideal. Y es por eso que como desarrolladores debemos manejar una gama de lenguajes. Y en lugar de usar sólo un lenguaje como el único martillo que nos hace ver todos los problemas como clavos, saber escoger el lenguaje correcto para resolver los problemas que enfrentamos.

Referencias

[1] TIOBE, *Programming Community Index*, `http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html`

Área en la que se centra el trabajo: Desarrollo de software

Nivel: Básico

Enfoque de la presentación: La presentación sigue las soluciones al problema, intentando mostrar ciertas construcciones particulares de cada lenguaje pero estableciendo el modelo general de cada lenguaje:

- Java: Orientación a objetos popular.
- Scala: Fusión entre programación funcional y orientación a objetos.
- Python: Orientación a objetos basada en atributos
- Ruby: Orientación a objetos basada en mensajes
- Javascript: Basado en atributos, también usable como lenguaje funcional.