El lenguaje de programación Komodo

César Danilo Pedraza Montoya
cpedraza@unal.edu.co

Índice

1. Introducción

Komodo es un lenguaje de programación hecho para probar ideas rápidamente. Es ideal para problemas con estructuras discretas como números y palabras. Komodo intenta que operar con estas entidades sea tan fácil como sea posible mientras se minimiza la cantidad de código necesario para llegar a una implementación exitosa. La otra prioridad de Komodo es la sencillez: se busca que el lenguaje sea pequeño y con reglas simples, con el propósito de que pueda ser aprendido con facilidad.

Komodo está diseñado con la intención de convertirse en una herramienta útil para generar estructuras discretas que puedan depender de muchas restricciones, para así estudiarlas. Esta es una tarea común en el estudio de areas de las matemáticas como la combinatoria, la teoría de la computación, la teoría de grafos o la teoría de códigos. Usar el computador como una herramienta de exploración matemática es una práctica conocida como matemática experimental.

Este documento describe el lenguaje de programación Komodo. No es una guía de uso del lenguaje. También se exploran detalles del intérprete de Komodo creado por el autor. Sin embargo, no se describe todo el comportamiento esperado de una implementación del lenguaje, ni se proveen detalles del intérprete más allá de lo estructural.

2. Visión general

El propósito de Komodo tiene consecuencias en su estructura. Puesto que Komodo es un lenguaje para *scripting*, no es una prioridad que el lenguaje pueda procesarse a si mismo, o que la representación de los datos sea similar a la representación de los programas. Asimismo, el nivel de abstracción de Komodo y las limitaciones de recursos humanos hacen preferible implementar un intérprete en lugar de un compilador.

El estado actual de Komodo (la versión 0.3.0), no fue alcanzado a través de un diseño deliberado, sino con una construcción iterativa.

2.1. La estructura del intérprete

Como suele suceder con múltiples compiladores e intérpretes, el intérprete de Komodo funciona como una cadena de procesamiento. Se comienza procesando texto, y tras cada paso se obtiene una representación del programa más preparada para ser ejecutada. En el caso de Komodo, se tienen las siguientes etapas:

1. Analizador léxico
2. Analizador sintáctico
3. Post-analizador sintáctico o *weeder*
4. Evaluador
5. Entorno de tiempo de ejecución

3. Análisis léxico y sintáctico

1. **Analizador léxico**

   El analizador léxico convierte un programa en una sucesión de *tokens*. Estos *tokens* representan palabras y símbolos del lenguaje. Puesto que Komodo usa bloques de código basados en indentación, su analizador léxico usa información del contexto de un token para generarlo. En particular, se guarda información sobre el nivel de indentación actual para emitir los *tokens* correctos. Esto difiere de muchos lenguajes de programación que dependen de paréntesis para determinar los bloques de código, en cuyo caso el lenguaje de todos los tokens suele ser regular. Cabe aclarar que este no es el caso de los *tokens* de Komodo.

2. **Analizador sintáctico**

El analizador sintáctico convierte sucesiones de tokens en un árbol que describe la estructura sintáctica del programa, conocido como CST (del inglés *Concrete Syntax Tree*). Este árbol contiene todos los detalles del programa, y es generado casi en su totalidad de forma independiente del contexto. La mayoría de la estructura del programa se obtiene de este paso.

3. **Post-analizador sintáctico o *weeder***

   El *weeder* toma un CST y realiza dos tareas:

   - Eliminar detalles innecesarios para la evaluación del código,
   - Verificar condiciones del programa que serían más dificiles de verificar en etapas anteriores.

   El resultado es un árbol de sintaxis abstracto o AST (del inglés *Abstract Syntax Tree*), que no contiene detalles como la precedencia de operadores, espacios o indentación. También convierte ciertos operadores infijos en nodos más restringidos, para facilitar la evaluación y eliminar estados indeseables. El tipo de errores que el *weeder* captura son de naturaleza sintáctica y dependientes del contexto.

4. Evaluación de código

1. **Evaluador**

   El evaluador toma nodos del AST y los convierte en valores o acciones. Komodo es un lenguaje orientado a expresiones, por lo que las evaluaciones siempre retornan un valor. Las acciones se dan en un entorno que es modificado cuando los nodos del AST son evaluados.

2. **Entorno de tiempo de ejecución**

   El entorno de tiempo de ejecución es la representación de los tipos de Komodo y un modelo de memoria donde se almacenan estas representaciones. Conforme los nodos son evaluados, el entorno es modificado. El modelo de memoria es una pila de *scopes*, donde se añaden *scopes* nuevos si se entra a un bloque de código nuevo. Todas las llamadas a funciones tienen un *scope* propio. Todas los bloques indentados tienen un scope propio.

4.1. Tipos

4.1.1. Números

Komodo tiene tres representaciones para números: Enteros, decimales y fracciones. Todos tienen tamaño arbitrario, que crece bajo demanda.

4.1.1.1. Enteros

Los enteros tienen signo y tienen las operaciones de suma, multiplicación, división, residuo, exponenciación y corrimiento de bits, tanto a la izquierda como a la derecha. Los bits más significativos están a la izquierda. Son representados en tiempo de ejecución como arreglos dinámicos de enteros de longitud de la palabra de máquina.

4.1.1.2. Decimales

Los decimales tienen signo y tienen las operaciones de suma, multiplicación y división. Son representados como enteros de longitud arbitraria y un entero de longitud de máquina, que representa la ubicación del punto en el número entero.

4.1.1.3. Fracciones

Las fracciones tienen signo y tienen las operaciones de suma, multiplicación, división y exponenciación. Son representados como un par de enteros de longitud arbitraria.

4.1.2. Funciones

Las funciones de Komodo pueden escribirse de dos formas:

- De forma anónima, como una lista de parámetros y un bloque de código,
- con nombre, como una lista de patrones y una lista de bloques de código correspondiente.

Las funciones de Komodo pueden ser pasadas como argumentos de otras funciones.

Los *scopes* de Komodo son creados de forma léxica, lo que significa que los nombres referenciados en la función son los que se obtienen en el contexto de la definición de la función, y no en el contexto de sus ejecuciones.

4.1.3. Caracteres y cadenas

4.1.3.1. Caracteres

Los caracteres de Komodo son valores escalares de Unicode, por lo que pueden representar exactamente los valores descritos por el estándar UTF-8. Tienen una longitud fija 32 bits.

4.1.3.2. Cadenas

Las cadenas de Komodo están representadas como arreglos inmutables de bytes, que están codificados con UTF-8.

Se puede iterar de izquierda a derecha sobre las cadenas de Komodo de la misma forma que se hace con las listas.

4.1.4. Contenedores

Los contenedores almacenan otros valores, incluyendo los de su mismo tipo. Todos los contenedores de Komodo permiten almacenar valores de diferente tipo en el mismo contenedor simultáneamente.

4.1.4.1. Listas

Las listas de Komodo son de longitud arbitraria y puede accederse a sus elementos de dos formas:

- con índices enteros indexados desde cero,
- iterando sobre la lista de izquierda a derecha.

Están representadas como arreglos dinámicos.

4.1.4.2. Conjuntos

Los conjuntos de Komodo son de longitud arbitraria. Se puede iterar sobre ellos y verificar la pertenencia de elementos.

Están representados como árboles binarios de búsqueda.

4.1.4.3. Diccionarios - Objetos

Los diccionarios de Komodo son de longitud arbitraria. Son colecciones de parejas clave-valor, donde el tipo de ambos es arbitrario.

Se puede acceder a sus elementos de dos formas:

- Notación de objeto: `objeto.clave`, donde `objeto` es un diccionario y `clave` es interpretado como una cadena, que es buscada en el diccionario.
- Notación usual: `dic[clave]` donde `dic` es un diccionario y `clave` es un valor arbitrario.

Están representados como árboles binarios de búsqueda.

4.2. Patrones

El mecanismo principal para representar lógica en Komodo es la búsqueda de patrones o *pattern matching*. Esto consiste en contrastar valores con patrones. Si un valor es compatible con un patrón, se ejecuta el código asociado a ese patrón.

Toda expresión que represente un valor de Komodo también es un patrón. Toda expresión que contenga un comodín (representado en Komodo con la barra baja «_») es un patrón. Las listas y conjuntos son compatibles con un patrón especial que permite iterar sobre ellos, conocido popularmente como notación `cons`.

La verificación de patrones está implementada comparando el árbol de sintaxis del patrón en cuestión con el valor que se quiere comparar.

5. Ejecución de programas

5.1. El intérprete

El intérprete de Komodo es un binario compilado estáticamente. Además de la interfaz del sistema operativo, el intérprete no tiene dependencias en tiempo de ejecución.

El intérprete está escrito en el lenguaje de programación Rust. El ecosistema de Rust, de manera similar a lenguajes como OCaml, es favorable para construir herramientas para lenguajes de programación. El modelo de memoria de Rust no incluye manejo de memoria automático, sino un sistema que permite verificar reglas que garantizan seguridad de memoria en tiempo de compilación.

5.2. Módulos de código

Komodo permite importar otros archivos con código, ya sea de fuentes externas o de la librería estándar.

5.2.1. Librería estándar

Komodo tiene una pequeña librería estándar que se incluye con la instalación del intérprete.

5.2.2. Importación de código externo

Se pueden importar archivos arbitrarios que serán interpretados como programas de Komodo, y de los que se extraerán nombres. Se pueden usar rutas relativas del sistema operativo, y se usará la ubicación del archivo ejecutado actualmente como referencia.

5.3. Gestión de memoria

El intérprete gestiona la memoria automáticamente con un algoritmo de *mark-and-sweep* sin adiciones. El espacio de memoria crece a demanda en tiempo de ejecución.

5.4. Interacción con el sistema

Hasta ahora, Komodo sólo interactúa directamente con la entrada y salida estándar, e indirectamente con la importación de código.

6. Aspectos periféricos

Hay software adicional al intérprete que lo asiste o extiende su alcance.

6.1. Editor web

Una compilación del interprete a *Web Assembly* es usada para interactuar poder usar el intérprete en navegadores de Internet. Es una versión sin la librería estándar y con una interfaz simulada de la entrada y salida estándar.

6.2. Resaltado de sintaxis

Se escribió una gramática de *TextMate* para los *tokens* de Komodo, y así obtener resaltado de sintaxis en los editores de texto compatibles.

Referencias