JVM

Una máquina virtual Java (en inglés Java Virtual Machine, JVM) es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

El código binario de Java no es un lenguaje de alto nivel, sino un verdadero código máquina de bajo nivel, viable incluso como lenguaje de entrada para un microprocesador físico. Como todas las piezas del rompecabezas Java, fue desarrollado originalmente por Sun.

La JVM es una de las piezas fundamentales de la plataforma Java. Básicamente se sitúa en un nivel superior al hardware del sistema sobre el que se pretende ejecutar la aplicación, y este actúa como un puente que entiende tanto el bytecode como el sistema sobre el que se pretende ejecutar. Así, cuando se escribe una aplicación Java, se hace pensando que será ejecutada

en una máquina virtual Java en concreto, siendo esta la que en última instancia convierte de código bytecode a código nativo del dispositivo final.

La gran ventaja de la máquina virtual java es aportar portabilidad al lenguaje, de manera que desde Sun Microsystems se han creado diferentes máquinas virtuales java para diferentes arquitecturas, y, así, un programa .class escrito en Windows puede ser interpretado en un entorno Linux. Tan solo es necesario disponer de dicha máquina virtual para dichos entornos. De ahí el famoso axioma que sigue a Java: "escríbelo una vez, ejecútalo en cualquier parte", o "Write once, run anywhere".

Pero los intentos de la compañía propietaria de Java y productos derivados de construir microprocesadores que aceptaran el Java bytecode como su lenguaje de máquina fueron más bien infructuosos.

La máquina virtual de Java puede estar implementada en software, hardware, una herramienta de desarrollo o un navegador web; lee y ejecuta código precompilado bytecode que es independiente de la plataforma multiplataforma. La JVM provee definiciones para un conjunto de instrucciones, un conjunto de registros, un formato para archivos de clases, la pila, un heap con recolector de basura y un área de memoria. Cualquier implementación de la JVM que sea aprobada por SUN debe ser capaz de ejecutar cualquier clase que cumpla con la especificación.

Existen varias versiones, en orden cronológico, de la máquina virtual de Java. En general la definición del Java bytecode no cambia significativamente entre versiones, y si lo hace, los desarrolladores del lenguaje procuran que exista compatibilidad hacia atrás con los productos anteriores.

A partir de J2SE 5.0, los cambios en la especificación de la JVM han sido desarrollados bajo el auspicio de la Java Community Process (JCP) y especificada en la JSR 924.1 Desde el año 2006, cambios en la especificación para soportar las modificaciones del formato del fichero de clases (JSR 2022) se están llevando a cabo en una versión de mantenimiento en la JSR 924. Las especificaciones para la JVM están publicadas en lo que se conoce como "el libro azul".3 Así reza el prefacio: Esperamos que esta especificación documente suficientemente la Máquina Virtual de Java para hacer posibles implementaciones desde cero. Sun proporciona tests que verifican que las implementaciones de la Máquina Virtual de Java operen correctamente.

Kaffe es un ejemplo de una implementación de JVM desde cero. Sun es la propietaria de la marca registrada "Java", que usa para certificar aquellas implementaciones que se ajustan y son totalmente compatibles con sus especificaciones. Sun fue adquirida en el año 2010 por Oracle Corporation.

Entorno de ejecución

Para poder ejecutar una aplicación en una Máquina Virtual de Java, el programa código debe compilarse de acuerdo a un formato binario portable estandarizado, normalmente en forma de ficheros con extensión .class. Un programa puede componerse de múltiples clases, en cuyo caso cada clase tendrá asociado su propio archivo .class. Para facilitar la distribución de aplicaciones, los archivos de clase pueden empaquetarse juntos en un archivo con formato jar. Esta idea apareció en la época de los primeros applets de Java. Estas aplicaciones pueden descargar aquellos archivos de clase que necesitan en tiempo de ejecución, lo que suponía una sobrecarga considerable para la red en una época donde la velocidad suponía un problema. El empaquetado evita la sobrecarga por la continua apertura y cierre de conexiones para cada uno de los fragmentos necesarios.

El código resultante de la compilación es ejecutado por la JVM que lleva a cabo la emulación del conjunto de instrucciones, bien por un proceso de interpretación o más habitualmente mediante un compilador JIT (Just In Time), como el HotSpot de Sun. Esta última opción convierte el bytecode a código nativo de la plataforma destino, lo que permite una ejecución mucho más rápida. El inconveniente es el tiempo necesario al principio para la compilación.

En un sentido amplio, la Máquina Virtual de Java actúa como un puente entre el resultado de la compilación (el bytecode) y el sistema sobre el que se ejecuta la aplicación. Para cada dispositivo debe haber una JVM específica, ya sea un teléfono móvil, un PC con Windows XP o un microondas. En cualquier caso, cada máquina virtual conoce el conjunto de instrucciones de la plataforma destino, y traduce un código escrito en lenguaje Java (común para todas) al código nativo que es capaz de entender el hardware de la plataforma.

El verificador del bytecode

La JVM «verifica» en esto todo bytecode antes de ejecutarlo. Esto significa que solo una cantidad limitada de secuencias de bytecode forman programas válidos; por ejemplo, una instrucción JUMP (branch) puede apuntar solo a una instrucción dentro de la misma función. A causa de esto, el hecho de que JVM

es una arquitectura de pila no implica una carga en la velocidad para emulación sobre arquitecturas basadas en registros cuando usamos un compilador JIT: no hay diferencia para un compilador JIT si nombra registros con nombres imaginarios o posiciones de pila imaginarias que necesitan ser ubicadas a los registros de la arquitectura objetivo. De hecho, la verificación de código hace a la JVM diferente de una arquitectura clásica de pila cuya emulación eficiente con un compilador JIT es más complicada y típicamente realizado por un intérprete más lento.

La verificación de código también asegura que los patrones de bits arbitrarios no pueden usarse como direcciones. La protección de memoria se consigue sin necesidad de una unidad de Gestión de Memoria (MMU). Así, JVM es una forma eficiente de obtener protección de memoria en chips que no tienen MMU.

Bytecodes

La JVM tiene instrucciones para los siguientes grupos de tareas:

- Carga y almacenamiento
- Aritmética
- Conversión de tipos
- Creación y manipulación de objetos
- Gestión de pilas (push y pop)
- Transferencias de control (branching)
- Invocación y retorno a métodos
- Excepciones

La clave es la compatibilidad binaria. Cada sistema operativo de un host particular necesita su propia implementación de JVM y runtime. Estas JVM interpretan el bytecode semánticamente de la misma manera, pero la implementación actual puede variar. Más complicado que solo la emulación de bytecode es la implementación compatible y eficiente de las API Java, las cuales tienen que ser mapeadas para cada sistema operativo de host.