Enlaces Relacionados

Códigos ya Escritos
Lecciones y Paso a Paso
Manuales
Curso de Java con tutor

MANUAL DE JAVA

CARACTERISTICAS DE JAVA

Las características principales que nos ofrece Java respecto a cualquier otro lenguaje de programación, son:

Es **SIMPLE** :

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje.

Java elimina muchas de las características de otros lenguajes como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el garbage collector (reciclador de memoria dinámica). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un thread de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de la memoria.

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

- aritmética de punteros
- no existen referencias
- registros (struct)
- definición de tipos (typedef)
- macros (#define)
- necesidad de liberar memoria (free)

Aunque, en realidad, lo que hace es eliminar las palabras reservadas (struct, typedef), ya que las clases son algo parecido.

Además, el intérprete completo de Java que hay en este momento es muy pequeño, solamente ocupa 215 Kb de RAM.

Es **ORIENTADO A OBJETOS** :

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y

Indice
Comenzando
Programas
Crear Web
Artículos
Más...

Foros

Foros

Acceso a tu cuenta

Regístrate
Accede a tu cuenta

polimorfismo. Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias . Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propietario del sistema operativo Next. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el runtime que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

Es **DISTRIBUIDO** :

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como http y ftp . Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Es **ROBUSTO** :

Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria. También implementa los arrays auténticos , en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobreescribir o corromper memoria resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.

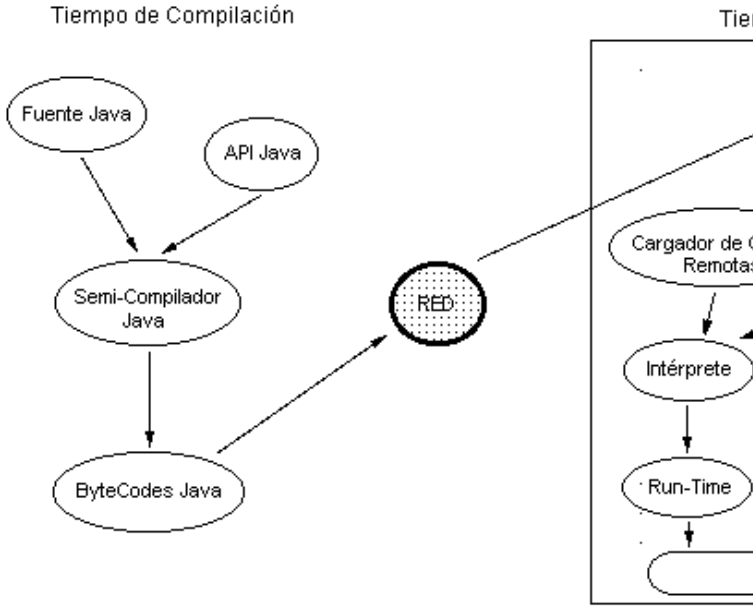
Además, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes , que son el resultado de la compilación de un programa Java. Es un código de máquina virtual que es interpretado por el intérprete Java. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes.

Java proporciona, pues:

- Comprobación de punteros
- Comprobación de límites de arrays
- Excepciones
- Verificación de byte-codes

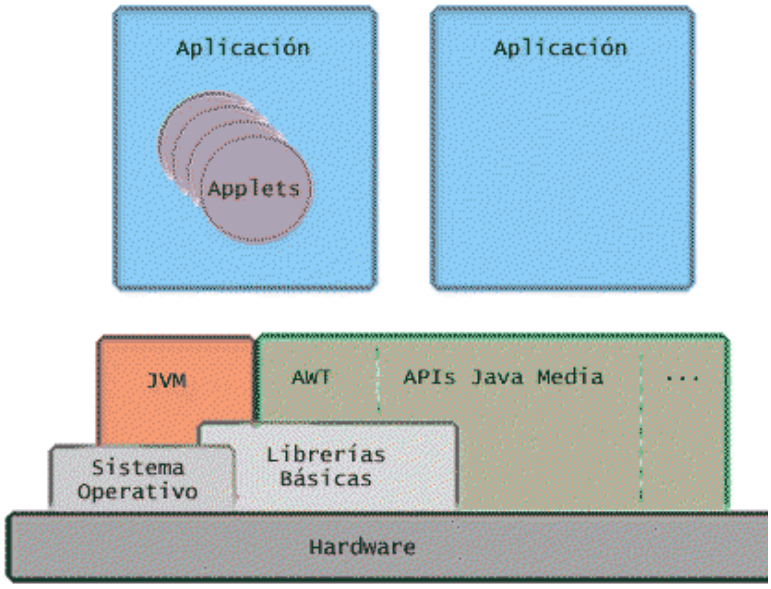
Es de **ARQUITECTURA NEUTRAL** :

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando en el porting a otras plataformas.



El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (byte-codes) está diseñado para ejecutarse en una máquina hipotética que es implementada por un sistema run-time, que sí es dependiente de la máquina.

En una representación en que tuviésemos que indicar todos los elementos que forman parte de la arquitectura de Java sobre una plataforma genérica, obtendríamos una figura como la siguiente:



En ella podemos ver que lo verdaderamente dependiente del sistema es la Máquina Virtual Java (JVM) y las librerías fundamentales, que también nos permitirían acceder directamente al hardware de la máquina. Además, habrá APIs de Java que también entren en contacto directo con el hardware y serán dependientes de la máquina, como ejemplo de este tipo de APIs podemos citar:

- Java 2D: gráficos 2D y manipulación de imágenes
- Java Media Framework : Elementos críticos en el tiempo: audio, video...
- Java Animation: Animación de objetos en 2D
- Java Telephony: Integración con telefonía
- Java Share: Interacción entre aplicaciones multiusuario
- Java 3D: Gráficos 3D y su manipulación

Es **SEGURO** :

La seguridad en Java tiene dos facetas. En el lenguaje, características como los punteros o el casting implícito que hacen los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria. Cuando se usa Java para crear un navegador, se combinan las características del lenguaje con protecciones de sentido común aplicadas al propio navegador.

El lenguaje C, por ejemplo, tiene lagunas de seguridad importantes, como son los errores de alineación . Los programadores de C utilizan punteros en conjunción con operaciones aritméticas. Esto le permite al programador que un puntero referencie a un lugar conocido de la memoria y pueda sumar (o restar) algún valor, para referirse

a otro lugar de la memoria. Si otros programadores conocen nuestras estructuras de datos pueden extraer información confidencial de nuestro sistema. Con un lenguaje como C, se pueden tomar números enteros aleatorios y convertirlos en punteros para luego acceder a la memoria:

```
printf( "Escribe un valor entero: " ); scanf( "%u",&puntero );
printf( "Cadena de memoria: %sn",puntero );
```

Otra laguna de seguridad u otro tipo de ataque, es el Caballo de Troya . Se presenta un programa como una utilidad, resultando tener una funcionalidad destructiva. Por ejemplo, en UNIX se visualiza el contenido de un directorio con el comando ls . Si un programador deja un comando destructivo bajo esta referencia, se puede correr el riesgo de ejecutar código malicioso, aunque el comando siga haciendo la funcionalidad que se le supone, después de lanzar su carga destructiva. Por ejemplo, después de que el caballo de Troya haya enviado por correo el /etc/shadow a su creador, ejecuta la funcionalidad de ls persentando el contenido del directorio. Se notará un retardo, pero nada inusual.

El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal -código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto-.

Si los byte-codes pasan la verificación sin generar ningún mensaje de error, entonces sabemos que:

- El código no produce desbordamiento de operandos en la pila
- El tipo de los parámetros de todos los códigos de operación son conocidos y correctos
- No ha ocurrido ninguna conversión ilegal de datos, tal como convertir enteros en punteros
- El acceso a los campos de un objeto se sabe que es legal: public, private, protected
- No hay ningún intento de violar las reglas de acceso y seguridad establecidas

El Cargador de Clases también ayuda a Java a mantener su seguridad, separando el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto limita cualquier aplicación del tipo Caballo de Troya , ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior.

Las clases importadas de la red se almacenan en un espacio de nombres privado, asociado con el origen. Cuando una clase del espacio de nombres privado accede a otra clase, primero se busca en las clases predefinidas (del sistema local) y luego en el espacio de nombres de la clase que hace la referencia. Esto imposibilita que una clase suplante a una predefinida.

En resumen, las aplicaciones de Java resultan extremadamente seguras, ya que no acceden a zonas delicadas de memoria o de sistema, con lo cual evitan la interacción de ciertos virus. Java no posee una semántica específica para modificar la pila de programa, la memoria libre o utilizar objetos y métodos de un programa sin los privilegios del kernel del sistema operativo. Además, para evitar modificaciones por parte de los crackers de la red, implementa un método ultraseguro de autentificación por clave pública. El Cargador de Clases puede verificar una firma digital antes de realizar una instancia de un objeto. Por tanto, ningún objeto se crea y almacena en memoria, sin que se validen los privilegios de acceso. Es decir, la seguridad se integra en el momento de compilación, con el nivel de detalle y de privilegio que sea necesario.

Dada, pues la concepción del lenguaje y si todos los elementos se mantienen dentro del estándar marcado por Sun, no hay peligro. Java imposibilita, también, abrir ningún fichero de la máquina local (siempre que se realizan operaciones con archivos, éstas trabajan sobre el disco duro de la máquina de donde partió el applet), no permite ejecutar ninguna aplicación nativa de una plataforma e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. Además, los intérpretes que incorporan los navegadores de la Web son aún más restrictivos. Bajo estas condiciones (y dentro de la filosofía de que el único ordenador seguro es el que está apagado, desenchufado, dentro de una cámara acorazada en un bunker y rodeado por mil soldados de los cuerpos especiales del ejército), se puede considerar que Java es un lenguaje seguro y que los applets están libres de virus.

Respecto a la seguridad del código fuente, no ya del lenguaje, JDK proporciona un desemsamblador de byte-code, que permite que cualquier programa pueda ser convertido a código fuente, lo que para el programador significa una vulnerabilidad total a su código. Utilizando javap no se obtiene el código fuente original, pero sí desmonta el programa mostrando el algoritmo que se utiliza, que es lo realmente interesante. La protección de los programadores ante esto es utilizar llamadas a programas nativos, externos (incluso en C o C++) de forma que no sea descompilable todo el código; aunque así se pierda portabilidad. Esta es otra de las cuestiones que Java tiene pendientes.

Es **PORTABLE** :

Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Además, Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

Es **INTERPRETADO** :

El intérprete Java (sistema run-time) puede ejecutar directamente el código objeto. Enlazar (linkar) un programa, normalmente, consume menos recursos que compilarlo, por lo que los desarrolladores con Java pasarán más tiempo desarrollando y menos esperando por el ordenador. No obstante, el compilador actual del JDK es bastante lento. Por ahora, que todavía no hay compiladores específicos de Java para las diversas plataformas, Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado y no ejecutado como sucede en cualquier programa tradicional.

Se dice que Java es de 10 a 30 veces más lento que C, y que tampoco existen en Java proyectos de gran envergadura como en otros lenguajes. La verdad es que ya hay comparaciones ventajosas entre Java y el resto de los lenguajes de programación, y una ingente cantidad de folletos electrónicos que supuran fanatismo en favor y en contra de los distintos lenguajes contendientes con Java. Lo que se suele dejar de lado en todo esto, es que primero habría que decidir hasta que punto Java, un lenguaje en pleno desarrollo y todavía sin definición definitiva, está maduro como lenguaje de programación para ser comparado con otros; como por ejemplo con Smalltalk, que lleva más de 20 años en cancha.

La verdad es que Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, es tanto interpretado como compilado. Y esto no es ningún contrasentido, me explico, el código fuente escrito con cualquier editor se compila generando el byte-code. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones máquina propias de cada plataforma y no tiene nada que ver con el p-code de Visual Basic. El byte-code corresponde al 80% de las instrucciones de la aplicación. Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Para ello hace falta el run-time, que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente el byte-code y añade el 20% de instrucciones que faltaban para su ejecución. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente al sistema operativo utilizado.

Es **MULTITHREADED** :

Al ser multithreaded (multihilvanado, en mala traducción), Java permite muchas actividades simultáneas en un programa. Los threads (a veces llamados, procesos ligeros), son básicamente pequeños procesos o piezas independientes de un gran proceso. Al estar los threads contruidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

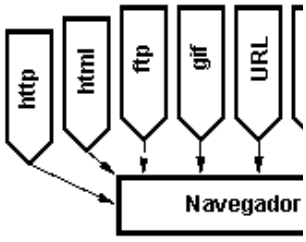
El beneficio de ser miltithreaded consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

Cualquiera que haya utilizado la tecnología de navegación concurrente, sabe lo frustrante que puede ser esperar por una gran imagen que se está trayendo. En Java, las imágenes se pueden ir trayendo en un thread independiente, permitiendo que el usuario pueda acceder a la información en la página sin tener que esperar por el navegador.

Es **DINAMICO** :

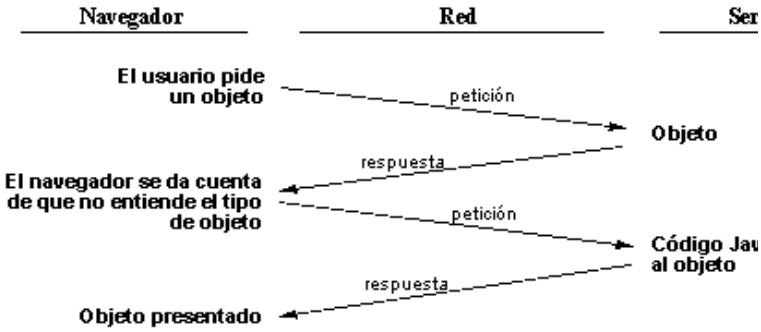
Java se beneficia todo lo posible de la tecnología orientada a objetos. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librería nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior).

Monolito: cada pieza de código se compacta dentro del código del navegador



Sistema Federado: e coordinador de pieza responsable de una f se pueden añadir d través de

Java también simplifica el uso de protocolos nuevos o actualizados. Si su sistema ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación que no sabe manejar, tal como se ha explicado en párrafos anteriores, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar.



Java, para evitar que los módulos de byte-codes o los objetos o nuevas clases, haya que estar trayéndolos de la red cada vez que se necesiten, implementa las opciones de persistencia, para que no se eliminen cuando de limpie la caché de la máquina.

Anterior Capítulo **Siguiente Capítulo**

Origen de Java **Instalación del JSDK**

- Versión imprimible de este documento

- Enviar por e-mail este documento

Página generada el 11-11-2016 a las 00:41:22