Plataforma Java
Edición Empresarial

Versión 1.0

Dante Alighieri N 453 Villa Las Palmas Comuna de Maipú Santiago de Chile
Teléfono 09-397.26.01 E-Mail: **contactos@ucaisia.cl**

CONTROL DE VERSIONES

Versión	Fecha	Autor	Principales Cambios
1.0.	14/02/2008	Ulises Cáceres Rui-Perez	Versión original.

INDICE

INTRODUCCIÓN

Este documento ha sido creado con el objeto de entregar a la comunidad informática y a todos aquellos usuarios que les interese saber más sobre la plataforma Java edición empresarial. Para ello, se ha elaborado una ayuda o guía que permitirá entender, aprender, poner en práctica, la metodología utilizada bajo J2EE.

Los enlaces son directos de aquellos que en la red proveen de información y de aquellos que se dedican a realizar productos para la plataforma empresarial o simplemente desarrollar las técnicas y metodologías de la plataforma.

La información de éste documento es una recopilación de nuestra empresa en experiencias y casos prácticos desarrollados por nuestro personal, más el apoyo de documentación obtenida desde los diferentes sitios que comentan sobre el tema.

Java EE

Java Platform, Enterprise Edition o **Java EE** (anteriormente conocido como Java 2 Platform, Enterprise Edition o J2EE hasta la versión 1.4), es una plataforma de programación — parte de la Plataforma Java — para desarrollar y ejecutar software de aplicaciones en Lenguaje de programación Java con arquitectura de n niveles distribuida, basándose ampliamente en componentes de software modulares y ejecutándose sobre un servidor de aplicaciones. La plataforma Java EE está definida por una *especificación*. Similar a otras especificaciones del Java Community Process, Java EE es también considerada informalmente como un estándar debido a que los suministradores deben cumplir ciertos requisitos de conformidad para declarar que sus productos son *conformes a Java EE*; no obstante sin un estándar de ISO o ECMA.

Java EE incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. Java EE también configura algunas especificaciones únicas para Java EE para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets (siguiendo la especificación de Portlets Java), Java Server Pages y varias tecnologías de servicios Web. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez que es integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de en tareas de mantenimiento de bajo nivel.

Historia

La especificación original J2EE fue desarrollada por Sun Microsystems.

Comenzando con J2EE 1.3, la especificación fue desarrollada bajo el Java Community Process. JSR 58 especifica J2EE 1.3 y JSR 151 especifica J2EE 1.4.

El SDK de J2EE 1.3 fue liberado inicialmente como beta en Abril de 2001. La beta del SDK de J2EE 1.4 fue liberada por Sun en Diciembre de 2002.

La especificación Java EE 5 fue desarrollada bajo el JSR 244 y la liberación final fue hecha el 11 de Mayo de 2006.

API's Generales

Las API's de Java EE incluyen varias tecnologías que extienden la funcionalidad de las API's base de Java SE.

Javax.ejb

La API Enterprise JavaBeans define un conjunto de API's que un contenedor de objetos distribuidos soportará para suministrar persistencia, RPC's (usando RMI o RMI-IIOP), control de concurrencia, transacciones y control de acceso para objetos distribuidos.

Javax.naming

Los paquetes javax.naming, javax.naming.directory, javax.naming.event, javax.naming.ldap y javax.naming.spi definen la API de Java Naming and Directory Interface (JNDI).

Java.sql

Los paquetes java.sql y javax.sql definen la API de JDBC.

Java.transaction

Estos paquetes definen la API Java Transaction API (JTA).

Javax.xml

Estos paquetes definen la API JAXP.

Javax.jms

Estos paquetes definen la API JMS.

Desarrollo de bajo costo (Plataforma)

Uno de los beneficios de Java EE como plataforma es que es posible empezar con poco o ningún costo. La implementación Java EE de Sun Microsystems puede ser descargada gratuitamente, y hay muchas herramientas de códigos abiertos disponibles para extender la plataforma o para simplificar el desarrollo.

Ejemplos de herramientas de desarrollo Java de código abierto de terceras partes son:

- NetBeans IDE, un IDE basado en Java
- La plataforma Eclipse ,un IDE basado en Java
- Expand, un plugin de Eclipse, para desarrollo rápido.
- Jedit, de código abierto, un IDE basado en Java
- Apache Software Foundation Apache Ant, una herramienta de construcción automática
- Apache Software Foundation Apache Maven, una herramienta de construcción automática y gestión de dependencias
- JUnit, un Framework para Pruebas de unidad automatizadas
- Apache Software Foundation Apache Tomcat, un contenedor web de Servlet / JSP
- Jetty, un servidor Web y un contenedor Web Servlet / JSP
- Struts, un Framework para desarrollar aplicaciones Web EE conforme al modelo MVC
- OpenXava, un framework de código abierto para desarrollo fácil de aplicaciones de negocio J2EE
- JDeveloper, un IDE basado en Java y desarrollado por Oracle
- JBuilder, desarrollado por Borland

Servidores de aplicación Java EE 5 certificados

- GlassFish, un servidor de aplicaciones de código abierto de Sun
- JBoss, un servidor de aplicaciones de código abierto de JBoss Inc. (adquirida por Red Hat en Abril del 2006)
- Servidor de Aplicaciones SAP NetWeaver, Java EE 5 Edition de SAP
- JEUS 6, un Servidor de aplicaciones específico de Linux de TmaxSoft

Servidores de aplicaciones J2EE 1.4 certificados

- Geronimo, un servidor de aplicaciones de código abierto de Apache
- JOnAS, un servidor de aplicaciones de código abierto de ObjectWeb
- Servidor de Aplicaciones SAP NetWeaver, Java EE 5 Edition de SAP
- Sun Java System Web Server
- Sun Java System Application Server
- IBM WebSphere Application Server (WAS), un servidor de aplicaciones altamente escalable, completamente conforme a J2EE de IBM
- Servidor de Aplicaciones WebLogic de BEA Systems

Servidores de aplicaciones J2EE 1.3 certificados

- Servidor de Aplicaciones JRun de Macromedia

Publicaciones

- Perrone, Paul J., Chaganti, Krishna (2003). *J2EE Developer's Handbook*. Indianapolis, Indiana: Sam's Publishing. ISBN 0-672-32348-6.
- Bodoff, Stephanie (2004). *The J2EE Tutorial*. Boston: Addison-Wesley. ISBN 0-321-24575-X.

Véase También

- Contenedor Web
- Descriptor de Despliegue
- Java BluePrints

Enlaces Externos

- Sun's Official Java EE Tutorial
- Sun's Java EE Training
- Sun's Java EE page - official documentation
- Java EE 5 technologies and JSRs
- Sun's J2EE compatibility page - certified J2EE servers
- Javalobby.org - popular Java, JSP & J2EE developer forums
- TheServerSide.com - popular discussion site for J2EE developers
- J2EE Factory to Enterprise Middleware.
- Jim Farley 1 August 2000. (O'Reilly 2004.)
- JavaToolbox List of the available development tools and libraries for Java/J2EE
- Java BluePrints - Sun's best practices for Java 2, Enterprise Edition application development.
- JavaRSS.com - A Java portal of Java websites rich in Java & J2EE News, Articles, Blogs, Groups, Forums and Tags
- Enterprise Java Technologies Tech Tips
- Java EE Tips
- SimplerJ2EE.net - Simpler J2EE
- Tutorial de J2EE en el website Java en castellano

Plataforma Java

La **plataforma Java** es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a bytecode y un conjunto de herramientas de desarrollo. En este caso, la plataforma no es un hardware específico o un sistema operativo, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común.

La plataforma es así llamada la **Plataforma Java** (antes conocida como Plataforma Java 2), e incluye:

- Plataforma Java, Edición Estándar (Java Platform, Standard Edition), o Java SE (antes J2SE)
- Plataforma Java, Edición Empresa (Java Platform, Enterprise Edition), o Java EE (antes J2EE)
- Plataforma Java, Edición Micro (Java Platform, Micro Edition), o Java ME (antes J2ME)

Desde 2006, la versión actual de la Plataforma Java Standard Edition se le conoce como Java SE 6 como versión externa, y 1.6 como versión interna. Sin embargo, se prefiere el término versión 6. Una visión general de la multitud de tecnologías que componen la Plataforma Java puede encontrarse en la página de documentación del JDK.

Tecnologías Java

La Plataforma Java se compone de un amplio abanico de tecnologías, cada una de las cuales ofrece una parte del complejo de desarrollo o del entorno de ejecución en tiempo real. Por ejemplo, los usuarios finales suelen interactuar con la máquina virtual de Java y el conjunto estándar de bibliotecas. Además, las aplicaciones Java pueden usarse de forma variada, como por ejemplo ser incrustadas en una página Web. Para el desarrollo de aplicaciones, se utiliza un conjunto de herramientas conocidas como JDK (Java Development Kit, o herramientas de desarrollo para Java).

Java Runtime Environment

Un programa destinado a la Plataforma Java necesita dos componentes en el sistema donde se va a ejecutar: una máquina virtual de Java (JVM), y un conjunto de librerías para proporcionar los servicios que pueda necesitar la aplicación. La JVM que proporciona Sun Microsystems, junto con su implementación de las librerías estándar, se conocen como Java Runtime Environment (JRE) o Entorno en tiempo de ejecución para Java. El JRE es lo mínimo que debe contener un sistema para poder ejecutar una aplicación Java sobre el mismo.

Máquina Virtual Java

El corazón de la Plataforma Java es el concepto común de un procesador "virtual" que ejecuta programas escritos en el lenguaje de programación Java. En concreto, ejecuta el código resultante de la compilación del código fuente, conocido como bytecode. Este "procesador" es la máquina virtual de Java o JVM (Java Virtual Machine), que se encarga de traducir (interpretar o compilar al vuelo) el bytecode en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre que con una implementación adecuada de la JVM. Este hecho es lo que ha dado lugar a la famosa frase: "write once, run anywhere" (escribir una vez, ejecutar en cualquier parte). La condición es que no se utilicen llamadas nativas o funciones especificas de una plataforma y aún así no se asegura completamente que se cumpla una verdadera independencia de plataforma.

Desde la versión 1.2 de JRE, la implementación de la máquina virtual de Sun incluye un compilador JIT (Just In Time). De esta forma, en vez de la tradicional interpretación del código bytecode, que da lugar a una ejecución lenta de las aplicaciones, el JIT convierte el bytecode a código nativo de la plataforma destino. Esta segunda compilación del código penaliza en cuanto a tiempo, pero el código nativo resultante se ejecuta de forma más eficaz y rápida que si fuera interpretado. Otras técnicas de compilación dinámica del código durante el tiempo de ejecución permiten optimizar más aún el código, dejando atrás la losa que sobre Java caía en cuanto a su lentitud y en sus últimas versiones la JVM se ha optimizado a tal punto que ya no se considera una plataforma lenta en cuanto a ejecución de aplicaciones.

Sin embargo, no se puede decir que el resultado de la compilación de Java pueda compilar el código con un máximo de eficiencia, y aprovechar los beneficios en cuanto a velocidad de código máquina nativo. Aunque los compiladores cada vez son más avanzados, no todas las librerías de Java tienen asociado un código máquina equivalente que aprovechar. Por ejemplo, la librería "reflect", que permite a los programadores de Java explorar instrucciones que sólo está disponible en tiempo de ejecución, está pobremente representado por código máquina.

Java no fue la primera plataforma basada en el concepto de una máquina virtual, aunque es la que de más amplia difusión ha gozado. El empleo de máquinas virtuales se había centrado principalmente en el uso de emuladores para ayudar al desarrollo de hardware en construcción o sistemas operativos, pero la JVM fue diseñada para ser implementada completamente en software, y al mismo tiempo hacer que fuera portable a todo tipo de hardware.

Librerías de Java

En la mayoría de los sistemas operativos actuales, se ofrece una cantidad de código para simplificar la tarea de programación. Este código toma la forma, normalmente, de un conjunto de librerías dinámicas que las aplicaciones pueden llamar cuando lo necesiten. Pero la Plataforma Java está pensada para ser independiente del sistema operativo subyacente, por lo que las aplicaciones no pueden apoyarse en funciones dependientes de cada sistema en concreto. Lo que hace la Plataforma Java, es ofrecer un conjunto de liberarías estándar, que contiene mucha de las funciones reutilizables disponibles en los sistemas operativos actuales.

Las librerías de Java tienen tres propósitos dentro de la Plataforma Java. Al igual que otras librerías estándar, ofrecen al programador un conjunto bien definido de funciones para realizar tareas comunes, como manejar listas de elementos u operar de forma sofisticada sobre cadenas de caracteres. Además, las librerías proporcionan una interfaz abstracta para tareas que son altamente dependientes del hardware de la plataforma destino y de su sistema operativo. Tareas tales como manejo de las funciones de red o acceso a ficheros, suelen depender fuertemente de la funcionalidad nativa de la plataforma destino. En el caso concreto anterior, las librerías java.net y java.io implementan el código nativo internamente, y ofrecen una interfaz estándar para que aplicaciones Java puedan ejecutar tales funciones. Finalmente, no todas las plataformas soportan todas las funciones que una aplicación Java espera. En estos casos, las librerías bien pueden emular esas funciones usando lo que esté disponible, o bien ofrecer un mecanismo para comprobar si una funcionalidad concreta está presente.

Lenguajes

La palabra Java, por sí misma, se refiere habitualmente al lenguaje de programación Java, que fue diseñado para usar con la Plataforma Java. Los lenguajes de programación se encuentran fuera del ámbito de lo que es una "plataforma", aunque el lenguaje de programación Java es uno de los componentes fundamentales de la plataforma Java. El propio lenguaje y el entorno en tiempo de ejecución suelen considerarse una única entidad.
Sin embargo, se han desarrollado fuera del entorno de Sun, un gran número de compiladores para la máquina virtual de Java (JVM). Algunos de estos lenguajes son:

- AspectJ
- Groovy
- JRuby
- Jython
- NetREXX
- Nice
- Pizza
- Rhino
- Scala

Plataformas Similares

El éxito de Java y el concepto y eslogan "write once, run anywhere" (escribir una vez, ejecutar en cualquier parte), ha dado lugar a que se lleven a cabo iniciativas en el mismo sentido. El intento más claro es la plataforma .NET de Microsoft, que copia la mayoría de los conceptos e innovaciones de Java; de hecho, tiene una implementación de Java llamada Visual J# (antes conocida como J++).

Véase También

- Lenguaje de programación Java
- Java (Sun)

Enlaces Externos

- Plataforma JavaSE
- Plataforma JavaME
- Plataforma JavaEE
- Tecnología Java

Lenguaje de Programación Java

Java	
Paradigma:	Orientado a objetos
Apareció en:	1991
Diseñado por:	Sun Microsystems
Tipo de dato:	Fuerte, Estático
Implementaciones:	Numerosas
Influido por:	Objective-C, C++, Smalltalk, Eiffel
Ha influido:	C#, J#, JavaScript
Sistema operativo:	Multiplataforma
Licencia de software:	GPL / Java Community Process

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las librerías de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

Historia

La tecnología Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada *the Green Project* en Sun Microsystems en el año 1991. El equipo (*Green Team*), compuesto por trece personas y dirigido por James Gosling, trabajó durante 18 meses en Sand Hill Road en Menlo Park en su desarrollo.

El lenguaje se denominó inicialmente *Oak* (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse *Green* tras descubrir que *Oak* era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a *Java*.

El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: ***J**ames Gosling, **A**rthur **V**an Hoff, y **A**ndy Bechtolsheim*. Otros abogan por el siguiente acrónimo, ***J**ust **A**nother **V**ague **A**cronym* ("sólo otro acrónimo ambiguo más"). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el *número mágico*) de los archivos .class que genera el compilador, son en hexadecimal, 0xCAFEBABE. Otros simplemente dicen que el nombre fue sacado al parecer de una lista aleatoria de palabras.

Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++. Entre junio y julio de 1994, tras una sesión maratónica de tres días entre John Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador Web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de Sun World, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

La promesa inicial de Gosling era *Write Once, Run Anywhere* (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la JVM) ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores Web pronto incorporaron la posibilidad de ejecutar applets Java incrustadas en las páginas Web.

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la librería estándar.

Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa *Java Specification Requests* (JSR's) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la *Java Language Specification* (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

- **JDK 1.0** (23 de enero de 1996) — Primer lanzamiento. Comunicado de prensa
- **JDK 1.1** (19 de febrero de 1997) — Principales adiciones incluidas: comunicado de prensa
  - Una reestructuración intensiva del modelo de eventos AWT (Abstract Windowing Toolkit)
  - Clases internas (inner classes)
  - JavaBeans
  - JDBC (Java Database Connectivity), para la integración de bases de datos
  - RMI (Remote Method Invocation)

- **J2SE 1.2** (8 de diciembre de 1998) — Nombre clave *Playground*. Esta y las siguientes versiones fueron recogidas bajo la denominación **Java 2** y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Otras mejoras añadidas incluían: comunicado de prensa
  - La palabra reservada (keyword) strictfp
  - Reflexión en la programación
  - La API gráfica (Swing) fue integrada en las clases básicas
  - La máquina virtual (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez
  - Java Plug-in
  - Java IDL, una implementación de IDL (Interfaz para Descripción de Lenguaje) para la interoperabilidad con CORBA
  - Colecciones (Collections)
- **J2SE 1.3** (8 de mayo de 2000) — Nombre clave *Kestrel*.

Los cambios más notables fueron: comunicado de prensa lista completa de cambios
  - La inclusión de la máquina virtual de HotSpot JVM (la JVM de HotSpot fue lanzada inicialmente en abril de 1999, para la JVM de J2SE 1.2)
  - RMI fue cambiado para que se basara en CORBA
  - JavaSound
  - se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de librerías principales (anteriormente disponible como una extensión)
  - Java Platform Debugger Architecture (JPDA)
- **J2SE 1.4** (6 de febrero de 2002) — Nombre Clave *Merlín*. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Los cambios más notables fueron: comunicado de prensa lista completa de cambios
  - Palabra reservada assert (Especificado en JSR 41.)
  - Expresiones regulares modeladas al estilo de las expresiones regulares Perl
  - Encadenación de excepciones Permite a una excepción encapsular la excepción de bajo nivel original.
  - non-blocking NIO (New Input/Output) (Especificado en JSR 51.)
  - Logging API (Specified in JSR 47.)
  - API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG
  - Parser XML integrado y procesador XSLT (JAXP) (Especificado en JSR 5 y JSR 63.)
  - Seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS)
  - Java Web Start incluido (El primer lanzamiento ocurrió en Marzo de 2001 para J2SE 1.3) (Especificado en JSR 56.)
- **J2SE 5.0** (30 de septiembre de 2004) — Nombre clave: *Tiger*. (Originalmente numerado 1.5, esta notación aún es usada internamente.) Desarrollado bajo JSR 176, Tiger añadió un número significativo de nuevas características comunicado de prensa
  - Plantillas (genéricos) — provee convención de tipos (type safety) en tiempo de compilación para colecciones y elimina la necesidad de la mayoría de conversión de tipos (type casting). (Especificado por JSR 14.)
  - Metadatos — también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos. (Especificado por JSR 175.)
  - Autoboxing/unboxing — Conversiones automáticas entre tipos primitivos (Como los int) y clases de envoltura primitivas (Como Integer). (Especificado por JSR 201.)

- Enumeraciones — la palabra reservada enum crea una typesafe, lista ordenada de valores (como Día.LUNES, Dia.MARTES, etc.). Anteriormente, esto solo podía ser llevado a cabo por constantes enteras o clases construidas manualmente (enum pattern). (Especificado por JSR 201.)
- Varargs (número de argumentos variable) — El último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos (e.g. void drawtext(String... lines)). En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al método.
- Bucle for mejorado — La sintaxis para el bucle for se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Iterable, como la clase estándar Collection, de la siguiente forma:

```
void displayWidgets (Iterable<Widget> widgets) {
    for (Widget w : widgets) {
        w.display();
    }
}
```

Este ejemplo itera sobre el objeto Iterable widgets, asignando, en orden, cada uno de los elementos a la variable w, y llamando al método display() de cada uno de ellos. (Especificado por JSR 201.)

- **Java SE 6** (11 de diciembre de 2006) — Nombre clave *Mustang*. Estuvo en desarrollo bajo la JSR 270. En esta versión, Sun cambió el nombre "J2SE" por **Java SE** y eliminó el ".0" del número de versión. Está disponible en http://java.sun.com/javase/6/. Los cambios más importantes introducidos en esta versión son:
  - Incluye un nuevo marco de trabajo y API's que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript.
  - Incluye el motor Rhino, de Mozilla, una implementación de JavaScript en Java.
  - Incluye un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, como JAX-WS 2.0, JAXB 2.0, STAX y JAXP.
  - Mejoras en la interfaz gráfica y en el rendimiento.
- **Java SE 7** — Nombre clave *Dolphin*. En el año 2006 aún se encontraba en las primeras etapas de planificación. Se espera que su desarrollo dé comienzo en la primavera de 2006, y se estima su lanzamiento para 2008.
  - Soporte para XML dentro del propio lenguaje.
  - Un nuevo concepto de superpaquete.
  - Soporte para closures.
  - Introducción de anotaciones estándar para detectar fallos en el software.
- No oficiales:
  - NIO2
  - Java Module System.
  - Java Kernel.
  - Nueva API para el manejo de Días y Fechas, la cual reemplazara las antiguas clases Date y Calendar.
  - Posibilidad de operar con clases BigDecimal usando operandos.

Además de los cambios en el lenguaje, con el paso de los años se han efectuado muchos más cambios dramáticos en la librería de clases de Java (*Java class library*) que ha crecido de unos pocos cientos de clases en JDK 1.0 hasta más de tres mil en J2SE 5.0. API's completamente nuevas, como Swing y Java2D, han sido introducidas y muchos de los métodos y clases originales de JDK 1.0 están obsoletas.

En el 2005 se calcula en 4,5 millones el número de desarrolladores y 2.500 millones de dispositivos habilitados con tecnología Java.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre).

Filosofía

El lenguaje Java se creó con cinco objetivos principales:
1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA (Common Object Request Broker Architecture), Internet Communications Engine u OSGi respectivamente.

Orientado a Objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que use estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y librerías de objetos.

Independencia de la Plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Es lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, ‘’’write once, run everywhere’’’.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode) — instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (VM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran librerías adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean “compatibles”. Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas, aunque sigue siendo mucho más lento que otros lenguajes.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en cualquier parte” por “Escríbelo una vez, depúralo en todas partes”)

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

Recolector de Basura

Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria solicitada dinámicamente de forma manual:

En C++, el desarrollador puede asignar memoria en una zona conocida como *heap* (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada puede llevar a una *fuga de memoria*, ya que el sistema operativo seguirá pensando que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual *cuelgue*. No obstante, se debe señalar que C++ también permite crear objetos en la pila de llamadas de una función o bloque, de forma que se libere la memoria (y se ejecute el destructor del objeto) de forma automática al finalizar la ejecución de la función o bloque.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios — es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que sea más rápida que en C++.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

Sintaxis

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase (recordemos que una clase es un molde en donde pueden crearse varios objetos).

Hola Mundo

Aplicaciones Autónomas

```
// Hola.java
public class Hola
{
   public static void main(String[] args) {
      System.out.println("Hola, mundo!");
   }
}
```

Este ejemplo necesita una pequeña explicación.

- Todo en Java está dentro de una clase, incluyendo programas autónomos.
- El código fuente se guarda en archivos con el mismo nombre que la clase que contienen y con extensión ".java". Una clase (class) declarada pública (public) debe seguir este convenio. En el ejemplo anterior, la clase es Hola, por lo que el código fuente debe guardarse en el fichero "Hola.java"
- El compilador genera un archivo de clase (con extensión ".class") por cada una de las clases definidas en el archivo fuente. Una clase anónima se trata como si su nombre fuera la concatenación del nombre de la clase que la encierra, el símbolo "$", y un número entero.
- Los programas que se ejecutan de forma independiente y autónoma, deben contener el método "main()".
- La palabra reservada "void" indica que el método main no devuelve nada.
- El método main debe aceptar un array de objetos tipo String. Por acuerdo se referencia como "args", aunque puede emplearse cualquier otro identificador.
- La palabra reservada "static" indica que el método es un método de clase, asociado a la clase en vez de a instancias de la misma. El método main debe ser estático o ''de clase''.
- La palabra reservada **public** significa que un método puede ser llamado desde otras clases, o que la clase puede ser usada por clases fuera de la jerarquía de la propia clase. Otros tipos de acceso son "private" o "protected".
- La utilidad de impresión (en pantalla por ejemplo) forma parte de la librería estándar de Java: la clase '''System''' define un campo público estático llamado '''out'''. El objeto out es una instancia de '''PrintStream''', que ofrece el método '''println(String)''' para volcar datos en la pantalla (la salida estándar).
- Las aplicaciones autónomas se ejecutan dando al entorno de ejecución de Java el nombre de la clase cuyo método main debe invocarse. Por ejemplo, una línea de comando (en Unix o Windows) de la forma java –cp . Hola ejecutará el programa del ejemplo (previamente compilado y generado "Hola.class") . El nombre de la clase cuyo método main se llama puede especificarse también en el fichero "MANIFEST" del archivo de empaquetamiento de Java (.jar).

Applets

Las applets de Java son programas incrustados en otras aplicaciones, normalmente una página Web que se muestra en un navegador.

```
// Hola.java
import java.applet. Applet;
import java.awt. Graphics;

public class Hola extends Applet {
    public void paint(Graphics gc) {
        gc.drawString("Hola, mundo!", 65, 95);
    }
}
<!-- Hola.html -->
<html>
  <head>
    <title>Applet Hola Mundo</title>
  </head>
  <body>
    <applet code="Hola" width="200" height="200">
    </applet>
  </body>
</html>
```

La sentencia **import** indica al compilador de Java que incluya las clases **java.applet. Applet** y **java.awt. Graphics**, para poder referenciarlas por sus nombres, sin tener que anteponer la ruta completa cada vez que se quieran usar en el código fuente.

La clase **Hola** extiende (extends) a la clase Applet, es decir, es una subclase de ésta. La clase Applet permite a la aplicación mostrar y controlar el estado del applet. La clase Applet es un componente del AWT (Abstract Windowing Toolkit), que permite al applet mostrar una inteterfaz gráfica de usuario o GUI (Graphical User Interface), y responder a eventos generados por el usuario.

La clase Hola sobrecarga el método **paint(Graphics)** heredado de la superclase contenedora (Applet en este caso), para acceder al código encargado de dibujar. El método paint() recibe un objeto **Graphics** que contiene el contexto gráfico para dibujar el applet. El método **paint()** llama al método drawString(String, int, int) del objeto **Graphics** para mostrar la cadena de caracteres **Hola, mundo!** en la posición (65, 96) del espacio de dibujo asignado al applet.

La referencia al applet es colocada en un documento HTML usando la etiqueta **<applet>**. Esta etiqueta o tag tiene tres atributos: **code="Hola"** indica el nombre del applet, y **width="200" height="200"** establece la anchura y altura, respectivamente, del applet. Un applet también pueden alojarse dentro de un documento HTML usando los elementos object, o embed, aunque el soporte que ofrecen los navegadores Web no es uniforme.[applettag][mixedbrowser]

Servlets

Los servlets son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.

```
// Hola.java
import java.io.*;
import javax.servlet.*;

public class Hola extends GenericServlet
{
   public void service(ServletRequest request, ServletResponse response)
      throws ServletException, IOException
   {
      response.setContentType("text/html");
      PrintWriter pw = response.getWriter();
      pw.println("Hola, mundo!");
      pw.close();
   }
}
```

Las sentencias **import** indican al compilador de Java la inclusión de todas las clases públicas e interfaces de los paquetes **java.io** y **javax.servlet** en la compilación.

La clase **Hola** extiende (**extends**), es heredera de la clase **GenericServlet**. Esta clase proporciona la interfaz para que el servidor le pase las peticiones al servlet y el mecanismo para controlar el ciclo de vida del servlet.

La clase Hola sobrecarga el método **service(ServletRequest, ServletResponse)**, definido por la interfaz **servlet** para acceder al manejador de la petición de servicio. El método service() recibe un objeto de tipo **ServletRequest** que contiene la petición del cliente y un objeto de tipo **ServletResponse**, usado para generar la respuesta que se devuelve al cliente. El método service() puede *lanzar* (**throws**) excepciones de tipo ServletException e IOException si ocurre algún tipo de anomalía.

El método **setContentType(String)** en el objeto respuesta establece el tipo de contenido MIME a "text/html", para indicar al cliente que la respuesta a su petición es una página con formato HTML. El método **getWriter()** del objeto respuesta devuelve un objeto de tipo **PrintWriter**, usado como una *tubería* por la que viajarán los datos al cliente. El método **println(String)** escribe la cadena "Hola, mundo!" en la respuesta y finalmente se llama al método **close()** para cerrar la conexión, que hace que los datos escritos en la **tubería** o stream sean devueltos al cliente.

Aplicaciones con Ventanas

Swing es la librería para la interfaz gráfica de usuario avanzada de la plataforma Java SE.

```
// Hola.java
import javax.swing.*;

public class Hola extends JFrame {
    Hola() {
        setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        add(new JLabel("Hola, mundo!"));
        pack();
    }

    public static void main(String[] args) {
        new Hola().setVisible(true);
    }
}
```

Las sentencias **import** indican al compilador de Java que las clases e interfaces del paquete **javax.swing** se incluyan en la compilación.

La clase **Hola** extiende (**extends**) la clase **javax.swing.JFrame**, que implementa una ventana con una barra de título y un control para cerrarla.

El constructor **Hola()** inicializa el marco o frame llamando al método **setDefaultCloseOperation(int)** heredado de JFrame para establecer las operaciones por defecto cuando el control de cierre en la barra de título es seleccionado al valor WindowConstants.DISPOSE_ON_CLOSE. Esto hace que se liberen los recursos tomados por la ventana cuando es cerrada, y no simplemente ocultada, lo que permite a la máquina virtual y al programa acabar su ejecución. A continuación se crea un objeto de tipo JLabel con el texto "Hola, mundo!", y se añade al marco mediante el método **add(Component)**, heredado de la clase **Container**. El método **pack()**, heredado de la clase **Window**, es invocado para dimensionar la ventana y distribuir su contenido.

El método **main()** es llamado por la JVM al comienzo del programa. Crea una instancia de la clase **Hola** y hace la ventana sea mostrada invocando al método **setVisible(boolean)** de la superclase (clase de la que hereda) con el parámetro a true. Véase que, una vez el marco es dibujado, el programa no termina cuando se sale del método **main()**, ya que el código del que depende se encuentra en un hilo de ejecución independiente ya lanzado, y que permanecerá activo hasta que todas las ventanas hayan sido destruidas.

Entornos de Funcionamientos

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.

En Dispositivos Móviles y Sistemas empotrados

Desde la creación de la especificación J2ME (Java 2 Platform, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el mercado de dispositivos electrónicos de consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores específicamente diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos.

El modelo de desarrollo de estas aplicaciones es muy semejante a las *applets* de los navegadores salvo que en este caso se denominan *MIDlets*.
Véase Sun Mobile Device Tecnology

En la navegación Web

Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (*plug-in*) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

El éxito de este tipo de aplicaciones (la visión del equipo de Gosling) no fue realmente el esperado debido a diversos factores, siendo quizás el más importante la lentitud y el reducido ancho de banda de las comunicaciones en aquel entonces que limitaba el tamaño de las applets que se incrustaban en el navegador. La aparición posterior de otras alternativas (aplicaciones web dinámicas de servidor) dejó un reducido ámbito de uso para esta tecnología, quedando hoy relegada fundamentalmente a componentes específicos para la intermediación desde una aplicación web dinámica de servidor con dispositivos ubicados en la máquina cliente donde se ejecuta el navegador.

Las *applets* Java no son las únicas tecnologías (aunque sí las primeras) de componentes complejos incrustados en el navegador. Otras tecnologías similares pueden ser: ActiveX de Microsoft, Flash, Java Web Start, etc.

En Sistemas de Servidor

En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica).

Los servlets y las JSP's supusieron un importante avance ya que:
- el API de programación es muy sencilla, flexible y extensible.
- los servlets no son procesos independientes (como los CGI's) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.

- las JSP's son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.

La especificación de Servlets y JSP's define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.

A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con Framework (pe Struts, Webwork, Tapestry) que se sobreponen sobre los servlets y las JSP's para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la especialización de roles sea posible (desarrolladores, diseñadores gráficos, ...) y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSP's) son substancialmente las mismas.
Este modelo de trabajo se ha convertido en un estándar *de-facto* para el desarrollo de aplicaciones web dinámicas de servidor y otras tecnologías (pe. ASP) se han basado en él.

En Aplicaciones de Escritorio

Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC's de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que su ejecución en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las API's de desarrollo gráfico (AWT). Desde la aparición de la librería Swing la situación mejoró substancialmente y posteriormente con la aparición de librerías como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

Plataformas Soportadas

Una versión del entorno de ejecución Java JRE (Java Runtime Environment) está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

Industria Relacionada

Sun Microsystem, como creador del lenguaje de programación Java y de la plataforma JDK, mantiene fuertes políticas para mantener una especificación del lenguaje así como de la máquina virtual a través del JCP. Es debido a este esfuerzo que se mantiene un estándar de facto.

Son innumerables las compañías que desarrollan aplicaciones para Java y/o están volcadas con esta tecnología:

- La industria de la telefonía móvil está fuertemente influenciada por la tecnología Java.
- El entorno de desarrollo Eclipse ha tomado un lugar importante entre la comunidad de desarrolladores Java.
- La fundación Apache tiene también una presencia importante en el desarrollo de librerías y componentes de servidor basados en Java.
- IBM, BEA, IONA, Oracle,... son empresas con grandes intereses y productos creados en y para Java.

Críticas

Harold dijo en 1995 que Java fue creado para abrir una nueva vía en la gestión de software complejo, y es por regla general aceptado que se ha comportado bien en ese aspecto. Sin embargo no puede decirse que Java no tenga grietas, ni que se adapta completamente a todos los estilos de programación, todos los entornos, o todas las necesidades.

General

- Java no ha aportado capacidades estándares para aritmética en punto flotante. El estándar IEEE 754 para "Estándar para Aritmética Binaria en Punto Flotante" apareció en 1985, y desde entonces es el estándar para la industria. Y aunque la aritmética flotante de Java *(cosa que cambió desde el 13 de Noviembre de 2006, cuando se abrió el código fuente y se adoptó la licencia GPL, aparte de la ya existente)* se basa en gran medida en la norma del IEEE, no soporta aún algunas características. Más información al respecto puede encontrarse en la sección final de enlaces externos.

El Lenguaje

- En un sentido estricto, Java no es un lenguaje absolutamente orientado a objetos, a diferencia de, por ejemplo, Ruby o Smalltalk. Por motivos de eficiencia, Java ha relajado en cierta medida el paradigma de orientación a objetos, y así por ejemplo, no todos los valores son objetos.
- El código Java puede ser a veces redundante en comparación con otros lenguajes. Esto es en parte debido a las frecuentes declaraciones de tipos y conversiones de tipo manual (casting). También se debe a que no se dispone de operadores sobrecargados, y a una sintaxis relativamente simple. Sin embargo, J2SE 5.0 introduce elementos para tratar de reducir la redundancia, como una nueva construcción para los bucles '''foreach'''.
- A diferencia de C++, Java no dispone de operadores de sobrecarga definidos por el usuario. Sin embargo esta fue una decisión de diseño que puede verse como una ventaja, ya que esta característica puede hacer los programas difíciles de leer y mantener.

Apariencia

La apariencia externa (el '"look and feel"') de las aplicaciones GUI (Graphical User Interface) escritas en Java usando la plataforma Swing difiere a menudo de la que muestran aplicaciones nativas. Aunque el programador puede usar el juego de herramientas AWT (Abstract Windowing Toolkit) que genera objetos gráficos de la plataforma nativa, el AWT no es capaz de funciones gráficas avanzadas sin sacrificar la portabilidad entre plataformas; ya que cada una tiene un conjunto de APIs distinto, especialmente para objetos gráficos de alto nivel. Las herramientas de Swing, escritas completamente en Java, evitan este problema construyendo los objetos gráficos a partir de los mecanismos de dibujo básicos que deben estar disponibles en todas las plataformas. El inconveniente es el trabajo extra requerido para conseguir la misma apariencia de la plataforma destino. Aunque esto es posible (usando GTK+ y el Look-and-Feel de Windows), la mayoría de los usuarios no saben cómo cambiar la apariencia que se proporciona por defecto por aquella que se adapta a la de la plataforma. Mención merece la versión optimizada del lenguaje.

Rendimiento

El rendimiento de una aplicación está determinado por multitud de factores, por lo que no es fácil hacer una comparación que resulte totalmente objetiva. En tiempo de ejecución, el rendimiento de una aplicación Java depende más de la eficiencia del compilador, o la JVM, que de las propiedades intrínsecas del lenguaje. El bytecode de Java puede ser interpretado en tiempo de ejecución por la máquina virtual, o bien compilado al cargarse el programa, o durante la propia ejecución, para generar código nativo que se ejecuta directamente sobre el hardware. Si es interpretado, será más lento que usando el código máquina intrínseco de la plataforma destino. Si es compilado, durante la carga inicial o la ejecución, la penalización está en el tiempo necesario para llevar a cabo la compilación.

Algunas características del propio lenguaje conllevan una penalización en tiempo, aunque no son únicas de Java. Algunas de ellas son el chequeo de los límites de arrays, chequeo en tiempo de ejecución de tipos, y la indirección de funciones virtuales.

El uso de un recolector de basura para eliminar de forma automática aquellos objetos no requeridos, añade una sobrecarga que puede afectar al rendimiento, o ser apenas apreciable, dependiendo de la tecnología del recolector y de la aplicación en concreto. Las JVM modernas usan recolectores de basura que gracias a rápidos algoritmos de manejo de memoria, consiguen que algunas aplicaciones puedan ejecutarse más eficientemente.

El rendimiento entre un compilador JIT y los compiladores nativos puede ser parecido, aunque la distinción no está clara en este punto. La compilación mediante el JIT puede consumir un tiempo apreciable, un inconveniente principalmente para aplicaciones de corta duración o con gran cantidad de código. Sin embargo, una vez compilado, el rendimiento del programa puede ser comparable al que consiguen compiladores nativos de la plataforma destino, inclusive en tareas numéricas. Aunque Java no permite la expansión manual de llamadas a métodos, muchos compiladores JIT realizan esta optimización durante la carga de la aplicación y pueden aprovechar información del entorno en tiempo de ejecución para llevar a cabo transformaciones eficientes durante la propia ejecución de la aplicación. Esta recompilación dinámica, como la que proporciona la máquina virtual HotSpot de Sun, puede llegar a mejorar el resultado de compiladores estáticos tradicionales, gracias a los datos que sólo están disponibles durante el tiempo de ejecución.

Java fue diseñado para ofrecer seguridad y portabilidad, y no ofrece acceso directo al hardware de la arquitectura ni al espacio de direcciones. Java no soporta expansión de código ensamblador, aunque las aplicaciones pueden acceder a características de bajo nivel usando librerías nativas (JNI, Java Native Interfaces).

Recursos

JRE

El **JRE** (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

Componentes

- Bibliotecas de Java, que son el resultado de compilar el código fuente desarrollado por quien implementa la JRE, y que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas librerías son:
  - Las bibliotecas centrales, que incluyen:
    - Una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos.
    - Bibliotecas para análisis de XML.
    - Seguridad.
    - Bibliotecas de internacionalización y localización.
  - Bibliotecas de integración, que permiten la comunicación con sistemas externos. Estas librerías incluyen:
    - La API para acceso a bases de datos JDBC (Java DataBase Conectivity).
    - La interfaz JNDI (Java Naming and Directory Interface) para servicios de directorio.
    - RMI (Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas.
  - Bibliotecas para la interfaz de usuario, que incluyen:
    - El conjunto de herramientas nativas AWT (Abstract Windowing Toolkit), que ofrece componentes GUI (Graphical User Interface), mecanismos para usarlos y manejar sus eventos asociados.
    - Las Bibliotecas de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT.
    - APIs para la captura, procesamiento y reproducción de audio.
- Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java (JVM), que es la encargada de la ejecución del código de las librerías y las aplicaciones externas.
- Plugins o conectores que permiten ejecutar applets en los navegadores Web.
- Java Web Start, para la distribución de aplicaciones Java a través de Internet.
- Documentación y licencia.

API's

Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus API's (Application Program Interface) de forma que pertenezcan a cada una de las plataformas:

- Java ME (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
- Java SE (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
- Java EE (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados **paquetes**. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP (Java Community Process). Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

En 2004, IBM y BEA apoyaron públicamente la idea de crear una implementación de código abierto (open source) de Java, algo a lo que Sun, a fecha de 2006, se ha negado.

Extensiones y Arquitecturas relacionadas

Las extensiones de Java están en paquetes que cuelgan de la raíz javax: javax.*. No se incluyen en la JDK o el JRE. Algunas de las extensiones y arquitecturas ligadas estrechamente al lenguaje Java son:

- Java EE (Java Platform, Enterprise Edition; antes J2EE) —para aplicaciones distribuidas orientadas al entorno empresarial
- Java ME (Java Platform, Micro Edition; antes J2ME)—para dispositivos de recursos limitados como teléfonos móviles y PDAs
- JMF (Java Media Framework)
- JavaHelp
- JavaMail
- JNDI (Java Naming and Directory Interface)
- JSML (Java Speech API Markup Language)
- JDBC (Java Database Connectivity)
- JDO (Java Data Objects)
- JAI (Java Advanced Imaging)
- JAIN (Java API for Integrated Networks)
- JDMK (Java Dynamic Management Kit)
- Jini (una arquitectura de red para la construcción de sistemas distribuidos
- Jiro
- Java Card
- JavaSpaces
- JML (Java Modeling Language)
- JMI (Java Metadata Interface)
- JMX (Java Management Extensions)

- JSP (JavaServer Pages)
- JSF (JavaServer Faces)
- JNI (Java Native Interface)
- JXTA (Protocolos abiertos para redes virtuales Peer-to-Peer o P2P)
- Java 3D (Una API de alto nivel para programación gráfica en 3D)
- JOGL (Java OpenGL—Una API de bajo nivel para programación gráfica usando OpenGL)
- LWJGL (Light Weight Java Game Library—Una API de bajo nivel para acceso a OpenGL, OpenAL y varios dispositivos de entrada)
- MARF (Modular Audio Recognition Framework)
- OSGi (Dynamic Service Management and Remote Maintenance)

Java Código Abierto

Java se ha convertido en un lenguaje con una implantación masiva en todos los entornos (personales y empresariales). El control que mantiene Sun sobre éste genera reticencias en la comunidad de empresas con fuertes intereses en Java (pe IBM, Oracle) y obviamente en la comunidad de desarrolladores de software libre.
La evolución basada en un comité en el que participen todos los implicados no es suficiente y la comunidad demandaba desde hace tiempo la liberación de las API's y librerías básicas de la JDK.

¿Hasta dónde Java es Software Libre?

En diciembre de 2006, Sun está en pleno relanzamiento de su plataforma Java bajo la licencia GPL de GNU. Cuando este cambio de licencia haya terminado, esperamos que Java ya no sea una trampa (fuente, (Ver la nota que hay en el recuadro amarillo))

Compromiso de Sun Microsystems con el Código Abierto

La importancia del código abierto en relación con Java puede verse entre otras cosas, en que el presidente y CEO de Sun, Jonathan Schwartz, ha retado a la compañía a que ofrezca código abierto para todo el software que produce Sun (fuente), Sun ya hace mucho tiempo que empezó a apostar por el código abierto cuando liberó StarOffice (Llamado ahora OpenOffice).
Sun ha aportado más líneas de código abierto que cualquier otra organización (fuente) También Richard Stallman opina eso mismo (fuente)

- El éxito del código abierto - Artículo puesto en el sito oficial en Español de Sun Microsystems, en el que habla de la apuesta de Sun Microsystems por el código abierto.

Alternativas Libres

Existen alternativas suficientemente maduras para el entorno de ejecución y de desarrollo de Java con una gran cobertura de funcionalidades con respecto a las implementaciones comerciales de Sun, IBM, Bea, ...

- Blackdown Java para Linux, incluye un plugin para Mozilla
- GNU Classpath de GNU - actualmente está siendo fusionado con libgcj del Compilador para Java de GNU
- Apache Harmony de Apache

Críticas Referentes a Java y el Software Libre

- Free But Shackled — The Java Trap, de Richard Stallman, 12 de abril, 2004. (respuesta de James Gosling)
  - Traducción al Español de este artículo: Libre pero encadenado. La trampa del Java. (Nótese que hay una nota en un recuadro amarillo que habla de la situación actual con respecto a lo que se dice en ese artículo)

Notar que este artículo fue escrito antes de la liberación del código fuente de Java. En la actualidad la postura de la Free Software Foundation y de Richard Stallman han cambiado, mostrandose partidarios ambos por su uso en software libre.

Software Libre Basado en Java

- Azureus
- Limewire
- iRATE Radio
- Java Source dispone de una lista de software libre (licencias GPL, LGPL, Apache, BSD, ...) hecho en Java.

Véase También

- Java syntax
- Java keywords
- Java virtual machine
- Java platform
- Java applet
- Java Platform, Standard Edition (Java SE, J2SE)
- JavaOS
- Comparison of Java and C++
- Comparison of C# and Java
- Java User Group
- Java Community Process
- JavaOne
- Join Java programming language
- Javapedia
- Inferno operating system

Referencias

- Jon Byous, *Java technology: The early years*. Sun Developer Network, sin fecha[ca. 1998]. Recuperado 21 de abril de 2005.
- James Gosling, *A brief history of the Green project*. Java.net, sin fecha [ca. Q1/1998]. Recuperado 22 abril de 2005.
- James Gosling, Bill Joy, Guy Steele, y Gilad Bracha, *The Java language specification*, tercera edición. Addison-Wesley, 2005. ISBN 0-321-24678-0.
- Tim Lindholm y Frank Yellin. *The Java Virtual Machine specification*, segunda edición. Addison-Wesley, 1999. ISBN 0-201-43294-3.

Ejemplos de Programación Dinámica

- Ejecución de n tareas en tiempo mínimo en un sistema de dos procesadores A y B
- Programas en disco
- Problema de los sellos con programación dinámica
- Problema de la mochila con programación dinámica
- Problema del producto de una secuencia de matrices con programación dinámica
- Problema de las monedas con programación dinámica
- Camino de coste mínimo entre dos nodos de un grafo dirigido
- Problema de la división de peso
- Problema de las vacas con programación dinámica
- Problema del Cambio de Palabra Programación Dinámica en JAVA

Notas

1. Especificación del lenguaje Java
2. Especificación de la máquina virtual Java

Enlaces Externos

- Wikilibros alberga un libro o manual sobre **Programación en Java**.
- módulos Perl sobre Java en CPAN (en inglés)

Sun

- Sitio oficial de Sun Microsystems en español
- Sitio oficial de Java para desarrolladores, etc
- Sitio oficial de Java no técnico para usuarios no avanzados
- The Java Language Specification, Tercera edición Especificación oficial del lenguaje Java
- Tutorial de Sun sobre el Lenguaje de programación Java
- Libro blanco original de Java, 1996
- Pruebe su VM
- Curso Gratuito de Introducción a Java

Peticiones para la especificación de Java (Java Specification Requests)

Hay varias JSRs relacionadas al lenguaje Java y las APIs del núcleo.

- JSR 14 *Add Generic Types To The Java Programming Language* (J2SE 5.0)
- JSR 41 *A Simple Assertion Facility* (J2SE 1.4)
- JSR 47 *Logging API Specification* (J2SE 1.4)
- JSR 51 *New I/O APIs for the Java Platform* (J2SE 1.4)
- JSR 59 *J2SE Merlin Release Contents* (J2SE 1.4)
- JSR 121 *Application Isolation API* (aún no incluida)
- JSR 133 *Java Memory Model and Thread Specification Revision* (J2SE 5.0)
- JSR 166 *Concurrency Utilities* (J2SE 5.0)
- JSR 175 *A Metadata Facility for the Java Programming Language* (J2SE 5.0)
- JSR 176 *J2SE 5.0 (Tiger) Release Contents* (J2SE 5.0)

- JSR 201 *Extending the Java Programming Language with Enumerations, Autoboxing, Enhanced for loops and Static Import* (J2SE 5.0)
- JSR 203 *More New I/O APIs for the Java Platform ("NIO.2")* (Java SE 7)
- JSR 204 *Unicode Supplementary Character Support* (J2SE 5.0) – support for Unicode 3.1
- JSR 270 *Java SE 6 ("Mustang") Release Contents* (Java SE 6)
- JSR 901 *Java Language Specification* (J2SE 5.0)

Tutoriales

- The Java Tutorial de Sun Microsystems (online)
- David Flanagan, *Java in a Nutshell, Third Edition*. O'Reilly & Associates, 1999. ISBN 1-56592-487-8
- *Thinking in Java*, de Bruce Eckel (online)
- *Java Course*, de A.B. Downey.
- *Introduction to Programming Using Java* Texto online de David J. Eck
- How to Think Like a Computer Scientist versión de Java
- An introduction to Computer Science using Java By Bradley Kjell. This text focuses on the principles and fundamentals of programming languages and computers in general, and uses Java as it's language of instruction.
- A Java tutorial by Hamed Alhoori, instructor at the University Of Bahrain.
- Full Java Tutorial
- En Castellano:
  - Curso de Java practico
  - Applets Java
  - Tutorial de Java básico
  - Manual de Java
  - Apuntes de java
  - Introducción a java
  - Colección «Java a tope» de libros electrónicos (Universidad de Málaga. España)

Videos Tutoriales

- Videotutoriales básicos sobre java

Recursos

- Java (Sun)
- Java Mexico Comunidad de desarrolladores mexicanos.
- Desarrollo de Software en Java Comunidad virtual para el intercambio de información en castellano sobre desarrollo de software en Java para profesionales practicantes, docentes y estudiantes.
- Computer-Books.us Colección de libros sobre Java disponibles para descarga gratuita.
- Javapedia project
- La Wiki de Java.net
- Sun Certification Resource
- JavaRSS.com Portal de sitios Web sobre lenguaje Java.
- developerWorks Java Zone - Comunidad de recursos Java
- JavaWhat.com Directorio de recursos de Java
- Información, ejemplos de programas, mini tutoriales, fuentes de información
- Desarrollo en JavaDesarrollo en Java
- Ejemplos, programas, foros y manuales en java

IDEs para Java

- BEA Workshop – software comercial, desarrollado por BEA Systems, integrado con BEA WebLogic
- BlueJ – libre, desarrollado como un proyecto de investigación universiatario. BlueJ es también un entorno interactivo adecuado para aprender Java
- Eclipse – libre y de código abierto, Eclipse es desarrollado por la Fundación Eclipse
- IntelliJ IDEA – software comercial, IntelliJ IDEA es desarrollado por JetBrains
- JGrasp - Software gratuito desarrollado por el Department of Computer Science and Software Engineering en el the Samuel Ginn College of Engineering de la Universidad de Auburn. Es un ambiente de desarrollo muy ligero para Java, C, C++, Objective C, Ada y VHDL. Integra diagramas UML para Java y Diagramas de Estructuras de Control.
- JBuilder – software comercial (existe una versión gratuita). JBuilder es desarrollado por Borland
- JCreator – software comercial(existe una versión gratuita) desarrollado por Xinox
- JDeveloper – IDE gratuito desarrollado por Oracle Corporation e integrado con Oracle Application Server
- NetBeans – IDE y plataforma base para aplicaciones ricas de escritorio (Rich Apps) gratuito de código abierto desarrollado por NetBeans.org
- Sun Java Studio Enterprise – software comercial (gratis para los miembros de Sun Developer Network), desarrollado por Sun Microsystems
- Rational Application Developer for WebSphere Software – software comercial, desarrollado por IBM, integrado con WebSphere Application Server

Alternativas

Alternativas libres - Alternativas de Software libre.

Críticas

- Críticas referentes a Java y el código abierto
- Is Java the language you would have designed if you didn't have to be compatible with C?, de Bjarne Stroustrup
- Softpanorama Java Critique Page: Java vs Scripting Languages, de Nikolai Bezroukov
- How Java's Floating-Point Hurts Everyone Everywhere, de W. Kahan und Joseph D. Darcy en el *ACM 1998 Workshop on Java for High–Performance Network Computing*
- Java's Cover de Paul Graham

Servidor de Aplicación

En informática se denomina **servidor de aplicaciones** a un servidor en una red de computadores que ejecuta ciertas aplicaciones

Usualmente se trata de un dispositivo de software que proporciona servicios de aplicación a las computadoras cliente. Un servidor de aplicaciones generalmente gestiona la mayor parte (o la totalidad) de las funciones de lógica de negocio y de acceso a los datos de la aplicación. Los principales beneficios de la aplicación de la tecnología de servidores de aplicación son la centralización y la disminución de la complejidad en el desarrollo de aplicaciones. Si bien el término es aplicable a todas las plataformas de software, hoy en día el término *servidor de aplicaciones* se ha convertido en sinónimo de la plataforma J2EE de Sun Microsystems.

Servidores de Aplicación J2EE

Como consecuencia del éxito del lenguaje de programación Java, el término *servidor de aplicaciones* usualmente hace referencia a un servidor de aplicaciones J2EE. WebSphere (IBM), Oracle Application Server (Oracle Corporation) y WebLogic (BEA) están entre los servidores de aplicación J2EE privativos más conocidos. EAServer (Sybase Inc.) es también conocido por ofrecer soporte a otros lenguajes diferentes a Java, como PowerBuilder. El servidor de aplicaciones JOnAS, desarrollado por el consorcio ObjectWeb, fue el primer servidor de aplicaciones libre en lograr certificación oficial de compatibilidad con J2EE. JBoss es otro servidor de aplicaciones libre y muy popular en la actualidad. Mucha gente confunde a Tomcat (The Apache Software Foundation) con un servidor de aplicaciones, sin embargo es solamente un contenedor de servlets.

J2EE provee estándares que le permiten a un servidor de aplicaciones servir como "contenedor" de los componentes que conforman dichas aplicaciones. Estos componentes, escritos en lenguaje Java, usualmente se conocen como Servlets, Java Server Pages (JSP's) y Enterprise JavaBeans (EJB's) y permiten implementar diferentes capas de la aplicación, como la interfaz de usuario, la lógica de negocio, la gestión de sesiones de usuario o el acceso a bases de datos remotas.

La portabilidad de Java también ha permitido que los servidores de aplicación J2EE se encuentren disponibles sobre una gran variedad de plataformas, como Unix, Microsoft Windows y GNU/Linux.

Otros Servidores de Aplicación

El término *servidor de aplicaciones* también ha sido aplicado a otros productos no-J2EE. Por ejemplo, con el aumento de la popularidad de .NET, Microsoft califica a su producto Internet Information Server como un servidor de aplicaciones. Adicionalmente, se pueden encontrar servidores de aplicación de códigos abiertos y comerciales de otros proveedores; algunos ejemplos son Base4 Server y Zope.

Características Comunes

Los servidores de aplicación típicamente incluyen también *middleware* (o software de conectividad) que les permite intercomunicarse con variados servicios, para efectos de confiabilidad, seguridad, no-repudio, etc. Los servidores de aplicación también brindan a los desarrolladores una Interfaz para Programación de Aplicaciones (API), de tal manera que no tengan que preocuparse por el sistema operativo o por la gran cantidad de interfaces requeridas en una aplicación web moderna.

Los servidores de aplicación también brindan soporte a una gran variedad de estándares, tales como HTML, XML, IIOP, JDBC, SSL, etc., que les permiten su funcionamiento en ambientes web (como Internet) y la conexión a una gran variedad de fuentes de datos, sistemas y dispositivos.

Usos

Un ejemplo común del uso de servidores de aplicación (y de sus componentes) son los portales de Internet, que permiten a las empresas la gestión y divulgación de su información, y un punto único de entrada a los usuarios internos y externos. Teniendo como base un servidor de aplicación, dichos portales permiten tener acceso a información y servicios (como servicios Web) de manera segura y transparente, desde cualquier dispositivo.

Véase También

- Cliente/Servidor

Enlaces Externos

- Página oficial de J2EE

Java Community Process

El Proceso de la Comunidad Java, o **Java Community Process**, establecido en 1998, es un proceso formalizado el cual permite a las partes interesadas a involucrarse en la definición de futuras versiones y características de la plataforma Java.

El proceso JCP conlleva el uso de **Java Specification Request** (JSR), las cuales son documentos formales que describen las especificaciones y tecnologías propuestas para que sean añadidas a la plataforma Java. Las revisiones públicas formales de JSRs son controladas antes de que los JSR se conviertan en *final* y sean votados por el Comité Ejecutivo JCP. Un JSR final suministra una *implementación de referencia* la cual da una implementación libre de la tecnología en código fuente y un *Kit de Compatibilidad de Tecnología* para verificar la especificación de la API.

El JCP mismo está descrito por un JSR. Desde 2006, la versión actual de JCP en uso es 2.6 como se describe por JSR 215.
Hay sobre 300 JSRs. Algunas de las más importantes JSRs son:

JSR #	Especificación o Tecnología
3	Java Management Extensions (JMX) 1.0, 1.1, & 1.2
5	Java API for XML Processing (JAXP) 1.0
13	BigDecimal Mejorado(Plataforma Java, Standard Edition#java.math)
14	Añadir Tipos Genéricos al Lenguaje de Programación Java (para J2SE 5.0)
16	Java EE Connector Architecture (JCA) 1.0
19	Enterprise JavaBeans (EJB) 2.0
31	Arquitectura Java para Enlazado XML (JAXB) 1.0
37	Perfil de Dispositivo de Información Móvil (MIDP) 1.0 para Java ME
40	Interfaz de Metadatos Java (JMI) 1.0
47	Especificación de la API de Logging (para J2SE 1.4)
51	Nuevas APIs I/O para la Plataforma Java (NIO) (para J2SE 1.4)
52	Librería de Etiquetas Estándar de Páginas JavaServer (JSTL) 1.0 y 1.1
53	Especificaciones Java Servlet 2.3 y JavaServer Pages (JSP) 1.2
54	Conectividad de Base de Datos Java (JDBC) 3.0
58	Plataforma Java 2, Edición Empresas (J2EE) 1.3
59	Plataforma Java 2, Edición Estándar (J2SE) 1.4 (Merlin)
63	API Java para Procesamiento de XML (JAXP) 1.1 y 1.2
68	Plataforma Java, Edición Micro (Java ME) 1.0
73	Minería de Datos Java API (JDM) 1.0
94	API del Motor de Reglas Java
102	Modelo de Objetos de Documento Java (JDOM) 1.0
110	APIs Java para WSDL (WSDL4J) 1.0
118	Mobile Information Device Profile (MIDP) 2.0 para Java ME
133	Modelo de Memoria Java y Revisión de Especificación de Thread

151	Plataforma Java 2, Edición Empresa (J2EE) 1.4
152	JavaServer Pages (JSP) 2.0
153	Enterprise JavaBeans (EJB) 2.1
154	Especificaciones Java Servlet 2.4 y 2.5
160	API Remota de Java Management Extensions (JMX) 1.0
168	Especificación de Portlet Java 1.0
176	Plataforma Java 2, Edición Estándar (J2SE) 5.0 (Tiger)
181	Metadatos de Servicios Web para la Plataforma Java
198	Una API de Extensión Estándar para Entornos de Desarrollo Integrados(IDE)
199	API del compilador Java
203	Más APIs de Nueva I/O para la Plataforma Java (NIO2)
206	API Java para Procesamiento de XML (JAXP) 1.3
215	Proceso de Comunidad Java (JCP) 2.6
220	Enterprise JavaBeans (EJB) 3.0
221	Java Database Connectivity (JDBC) 4.0
244	Plataforma Java, Edición Empresas (Java EE) 5
245	JavaServer Pages (JSP) 2.1
247	API de Minería de Datos Java (JDM) 2.0
252	JavaServer Faces (JSF) 1.2
255	Java Management Extensions (JMX) 2.0
260	Actualización de la Tecnología de Etiquetas Javadoc
270	Plataforma Java, Edición Estándar (Java SE) 6 (Mustang)
308	Anotación Java en Tipos Java (Java SE 7)
900	Especificación del Lenguaje de programación Java, Tercera Edición (JLS) (para J2SE 5.0 incorpora cambios desde los JSRs 14, 41, 133, 175, 201, y 204)
907	Java Transaction API (JTA) 1.0 y 1.1
913	Proceso de Comunidad Java (JCP) 2.0, 2.1 & 2.5
914	Java Message Service (JMS) API 1.0 y 1.1
924	Especificación de la Maquina Virtual Java, Segunda Edición (JVM) (para J2SE 5.0)

Notas

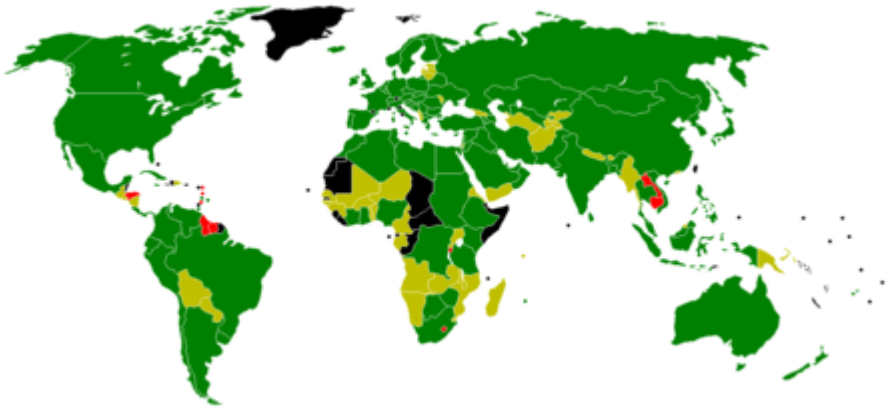
1. JSR 3 originalmente especificó el release JMX 1.0. Los dos releases "final" subsiguientes han dado JMX 1.1 y JMX 1.2. JMX 2.0 está especificado por JSR 255.
2. JSR 52 originalmente especificó el release JSTL 1.0. Un release subsiguiente de mantenimiento suministró JSTL 1.1.
3. JSR 63 originalmente especificó el release JAXP 1.1. Un release subsiguiente de mantenimiento de JSR 63 suministró la especificación JAXP 1.2. JAXP 1.3 se especifica por JSR 206.

4. JSR 154 especificó originalmente el release Java Servlet 2.4. Desde Febrero de 2006 un borrador de mantenimiento de la especificación servlet 2.5 está bajo revisión, planificada para terminar el 20 de Marzo de 2006.
5. JSR 913 originalmente especificó JCP 2.0. Fue modificado por varios cambios para votar las reglas para producir la versión 2.1 y después cambió las reglas de licencias, políticas y procesos para obtener la versión 2.5. JCP 2.6 está definido por el JSR 215.
6. JSR 924 originalmente especificó cambios a la JVM para soportar los cambios en J2SE 5.0.

Enlaces Externos

- Página Inicial de Java Community Process
- Lista de todos los JSRs considerados final
- Lista de todos los JSRs

Organización Internacional para la Estandarización



Mapa mundial de Estados con comités miembros de la ISO
Key:

- Miembros natos
- Miembros correspondientes
- Miembros suscritos
- Otros Estados clasificados ISO 3166-1, no miembros de la ISO

ISO Vocablo griego que significa igual

La **Organización Internacional para la Estandarización** o *International Organization for Standardization* , que nace después de la segunda guerra mundial (fue creada en 1946), es el organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica. Su función principal es la de buscar la estandarización de normas de productos y seguridad para las empresas u organizaciones a nivel internacional. La ISO es una red de los institutos de normas nacionales de 157 países, sobre la base de un miembro por el país, con una Secretaría Central en Ginebra, Suiza, que coordina el sistema. La Organización Internacional de Normalización (ISO), con base en Ginebra, Suiza, está compuesta por delegaciones gubernamentales y no gubernamentales subdivididos en una serie de subcomités encargados de desarrollar las guías que contribuirán al mejoramiento ambiental. Las normas desarrolladas por ISO son voluntarias, comprendiendo que ISO es un organismo no gubernamental y no depende de ningún otro organismo internacional, por lo tanto, no tiene autoridad para imponer sus normas a ningún país.

Es una organización internacional no gubernamental, compuesta por representantes de los organismos de normalización (ON's) nacionales, que produce normas internacionales industriales y comerciales. Dichas normas se conocen como **normas ISO** y su finalidad es la coordinación de las normas nacionales, en consonancia con el Acta Final de la Organización Mundial del Comercio, con el propósito de facilitar el comercio, facilitar el intercambio de información y contribuir con unos estándares comunes para el desarrollo y transferencia de tecnologías.

Estructura de la Organización

La Organización ISO está compuesta por tres tipos de miembros:

- **Miembros natos**, uno por país, recayendo la representación en el organismo nacional más representativo.
- **Miembros correspondientes**, de los organismos de países en vías de desarrollo y que todavía no poseen un comité nacional de normalización. No toman parte activa en el proceso de normalización pero están puntualmente informados acerca de los trabajos que les interesen.
- **Miembros suscritos**, países con reducidas economías a los que se les exige el pago de tasas menores que a los correspondientes.

ISO es un órgano consultivo de la Organización de las Naciones Unidas. Coopera estrechamente con la Comisión Electrotécnica Internacional (*International Electrotechnical Commission*, IEC) que es responsable de la estandarización de equipos eléctricos.

Nombre de la Organización

ISO no es un acrónimo; proviene del griego *iso*, que significa igual. Es un error común el pensar que ISO significa *International Standards Organization*, o algo similar; en inglés su nombre es *International Organization for Standardization*, mientras que en francés se denomina *Organisation Internationale de Normalisation*; el uso del acrónimo conduciría a nombres distintos: IOS en inglés y OIN en francés, por lo que los fundadores de la organización eligieron

ISO como la forma corta y universal de su nombre.

Principales Normas ISO

Algunos estándares son los siguientes:

- ISO 216 Medidas de papel: p.e. ISO A4
- ISO 639 Nombres de lenguas
- ISO 690:1987 regula las citas bibliográficas (corresponde a la norma UNE 50104:1994)
- ISO 690-2:1997 regula las citas bibliográficas de **documentos electrónicos**
- ISO 732 Formato de carrete de 120
- ISO 838 Estandar para perforadoras de papel
- ISO 1007 Formato de carrete de 135
- ISO/IEC 1539-1 Lenguaje de programación Fortran
- ISO 3029 Formato carrete de 126
- ISO 3166 códigos de países
- ISO 4217 códigos de divisas
- ISO 7811 Técnica de grabación en tarjetas de identificación
- ISO 8601 Representación del tiempo y la fecha. Adoptado en Internet mediante el *Date and Time Formats* de W3C que utiliza UTC.
- ISO 8859 codificaciones de caracteres que incluye ASCII como un subconjunto (Uno de ellos es el ISO 8859-1 que permite codificar las lenguas originales de Europa occidental, como el español)
- ISO/IEC 8652:1995 Lenguaje de programación Ada
- ISO 9000 Sistemas de Gestión de la Calidad - Fundamentos y vocabulario
- ISO 9001 Sistemas de Gestión de la Calidad - Requisitos
- ISO 9004 Sistemas de Gestión de la Calidad - Directrices para la mejora del desempeño
- ISO 9660 Sistema de archivos de CD-ROM

- ISO 9899 Lenguaje de programación C
- ISO 10279 Lenguaje de programación BASIC
- ISO 10646 Universal Character Set
- ISO/IEC 11172 MPEG-1
- ISO/IEC_12207 Tecnología de la información / Ciclo de vida del software
- ISO 13450 Formato de carrete de 110
- ISO/IEC 13818 MPEG-2
- ISO 14000 Estándares de Gestión Medioambiental en entornos de producción
- ISO/IEC 14496 MPEG-4
- ISO/IEC 15444 JPEG 2000
- ISO 15693 Estándar para "tarjetas de vecindad"
- ISO/IEC 17799 Seguridad de la información
- ISO 26300 OpenDocument
- ISO/IEC 17025 Requisitos generales relativos a la competencia de los laboratorios de ensayo y calibración
- ISO/IEC 26300: OpenDocument Format (.odf)
- ISO/IEC 27001 Sistema de Gestión de Seguridad de la Información
- ISO/IEC 20000 Tecnología de la información. Gestión del servicio.
- ISO 32000 Formato de Documento Portátil (.pdf)
- GARANTIA S-1: Garantía contra todo tipo de fallas en la grabación de discos compactos, excepto problemas de compatibilidad.

Véase También

- Normalización o estandarización
- Normas ISO 9000
- ISO 8601

Enlaces Externos

- Sitio Web Oficial de ISO
- Norma ISO 690-2 para citar documentos electrónicos

Application Programing Interface

Una **API** (del inglés **Application Programming Interface - Interfaz de Programación de Aplicaciones**) es el conjunto de **funciones** y **procedimientos** (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

Características

Una API representa un interfaz de comunicación entre componentes software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las APIs asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.
Por ejemplo, se puede ver la tarea de escribir "Hola Mundo" sobre la pantalla en diferentes niveles de abstracción:

1. Haciendo todo el trabajo desde el principio:
    1. Traza, sobre papel milimetrado, la forma de las letras (y espacio) "H, o, l, a, M, u, n, d, o".
    2. Crea una matriz de cuadrados negros y blancos que se asemeje a la sucesión de letras.
    3. Mediante instrucciones en ensamblador, escribe la información de la matriz en la memoria intermedia ("buffer") de pantalla.
    4. Mediante la instrucción adecuada, haz que la tarjeta gráfica realice el volcado de esa información sobre la pantalla.
2. Por medio de un sistema operativo para hacer parte del trabajo:
    1. Carga una fuente tipográfica proporcionada por el sistema operativo.
    2. Haz que el sistema operativo borre la pantalla.
    3. Haz que el sistema operativo dibuje el texto "Hola Mundo" usando la fuente cargada.
3. Usando una aplicación (que a su vez usa el sistema operativo) para realizar la mayor parte del trabajo:
    1. Escribe un documento HTML con las palabras "Hola Mundo" para que un navegador Web como Mozilla, Firefox, Opera o Internet Explorer pueda representarlo en el monitor.

Como se puede ver, la primera opción requiere más pasos, cada uno de los cuales es mucho más complicado que los pasos de las opciones siguientes. Además, no resulta nada práctico usar el primer planteamiento para representar una gran cantidad de información, como un artículo enciclopédico sobre la pantalla, mientras que el segundo enfoque simplifica la tarea eliminando un paso y haciendo el resto más sencillos y la tercera forma simplemente requiere escribir "Hola Mundo". Sin embargo, las APIs de alto nivel generalmente pierden flexibilidad; por ejemplo, resulta mucho más difícil en un navegador web hacer girar texto alrededor de un punto con un contorno parpadeante que programarlo a bajo nivel. Al elegir usar una API se debe llegar a un cierto equilibrio entre su potencia y simplicidad y su pérdida de flexibilidad.

Ejemplos de API

- Microsoft WMI
- Microsoft Win32 API
- Microsoft Framework .NET
- OpenGL
- SUN J2EE APIs
- API for SCSI device interfacing
- The Carbon APIs for the Macintosh OS
- Common Object Request Broker Architecture (CORBA)
- Javascript-C de Mozilla Spidermonkey
- Symfony para PHP

Enlaces Externos

- API de newsletter (en portugués)
- Google Web APIs (beta)
- Flickr API Documentation
- Tutorial: Llamar a las API de Windows

JDBC

JDBC es el acrónimo de *Java Database Connectivity*, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee en localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos a las que tenga permiso: consultas, actualizaciones, creado modificado y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc. a **Véase también:** ODBC

Enlaces Externos

- aulambra.com - JDBC: Java DataBase Connectivity, JDBC
- JDBC Technology (en inglés)
- JDBC Basics (en inglés)
- JDBC en español (en español)
- Consulta JDBC, ejemplo tipo (en español)

RMI

Este artículo necesita una revisión de gramática, ortografía y estilo.
Cuando el artículo esté corregido, borra esta plantilla, por favor.

RMI (***Java Remote Method Invocation***) es un mecanismo ofrecido en Java para invocar un método remotamente. Al ser RMI parte estándar del entorno de ejecución Java, usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java. Si se requiere comunicarse con otras tecnologías debe usarse CORBA o SOAP en lugar de RMI.

Al estar específicamente diseñado para Java, RMI puede darse el lujo de ser muy amigable para los programadores, proveyendo pasaje por referencia de objetos (cosa que no hace SOAP), "recolección de basura" distribuida y pasaje de tipos arbitrarios (funcionalidad no provista por CORBA).

Por medio de RMI, un programa Java puede exportar un objeto. A partir de esa operación este objeto está disponible en la red, esperando conexiones en un puerto TCP. Un cliente puede entonces conectarse e invocar métodos. La invocación consiste en el "marshalling" de los parámetros (utilizando la funcionalidad de "serialización" que provee Java), luego se sigue con la invocación del método (cosa que sucede en el servidor). Mientras esto sucede el llamador se queda esperando por una respuesta. Una vez que termina la ejecución el valor de retorno (si lo hay) es serializado y enviado al cliente. El código cliente recibe este valor como si la invocación hubiera sido local.

Contexto

Desde la versión 1.1 de JDK, Java tiene su propio ORB: RMI (Remote Method Invocation). A pesar de que RMI es un ORB en el sentido general, no es un modelo compatible con CORBA. RMI es nativo de Java, es decir, es una extensión al núcleo del lenguaje. RMI depende totalmente del núcleo de la Serialización de Objetos de Java, así como de la implementación tanto de la portabilidad como de los mecanismos de carga y descarga de objetos en otros sistemas, etc.

El uso de RMI resulta muy natural para todo aquel programador de Java ya que éste no tiene que aprender una nueva tecnología completamente distinta de aquella con la cual desarrollará. Sin embargo, RMI tiene algunas limitaciones debido a su estrecha integración con Java, la principal de ellas es que esta tecnología no permite la interacción con aplicaciones escritas en otro lenguaje.

RMI como extensión de Java, es una tecnología de programación, fue diseñada para resolver problemas escribiendo y organizando código ejecutable. Así RMI constituye un punto específico en el espacio de las tecnologías de programación junto con C, C++, Smalltalk, etc

Arquitectura

La arquitectura RMI puede verse como un modelo de cuatro capas:

Primera Capa

La primera capa es la de aplicación y se corresponde con la implementación real de las aplicaciones cliente y servidor. Aquí tienen lugar las llamadas a alto nivel para acceder y exportar objetos remotos. Cualquier aplicación que quiera que sus métodos estén disponibles para su acceso por clientes remotos debe declarar dichos métodos en una interfaz que extienda java.rmi.Remote. Dicha interfaz se usa básicamente para "marcar" un objeto como remotamente accesible. Una vez que los métodos han sido implementados, el objeto debe ser exportado. Esto puede hacerse de forma implícita si el objeto extiende la clase UnicastRemoteObject (paquete java.rmi.server), o puede hacerse de forma explícita con una llamada al método exportObject() del mismo paquete.

Segunda Capa

La capa 2 es la capa proxy, o capa stub-skeleton. Esta capa es la que interactúa directamente con la capa de aplicación. Todas las llamadas a objetos remotos y acciones junto con sus parámetros y retorno de objetos tienen lugar en esta capa.

Tercera Capa

La capa 3 es la de referencia remota, y es responsable del manejo de la parte semántica de las invocaciones remotas. También es responsable de la gestión de la replicación de objetos y realización de tareas específicas de la implementación con los objetos remotos, como el establecimiento de las persistencias semánticas y estrategias adecuadas para la recuperación de conexiones perdidas. En esta capa se espera una conexión de tipo stream (stream-oriented connection) desde la capa de transporte.

Cuarta Capa

La capa 4 es la de transporte. Es la responsable de realizar las conexiones necesarias y manejo del transporte de los datos de una máquina a otra. El protocolo de transporte subyacente para RMI es JRMP (Java Remote Method Protocol), que solamente es "comprendido" por programas Java.

Elementos

Toda aplicación RMI normalmente se descompone en 2 partes:

- Un servidor, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- Un cliente, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

Ejemplo

Un servidor RMI consiste en definir un objeto remoto que va a ser utilizado por los clientes. Para crear un objeto remoto, se define una interfaz, y el objeto remoto será una clase que implemente dicha interfaz. Veamos como crear un servidor de ejemplo mediante 3 pasos:

- Definir el interfaz remoto. Cuando se crea un interfaz remoto:
  - El interfaz debe ser público.
  - Debe extender (heredar de) el interfaz java.rmi.Remote, para indicar que puede llamarse desde cualquier máquina virtual Java.
  - Cada método remoto debe lanzar la excepción java.rmi.RemoteException en su claúsula throws, además de las excepciones que pueda manejar.
  -

```
public interface MiInterfazRemoto extends java.rmi.Remote
{
  public void miMetodo1() throws java.rmi.RemoteException;
  public int miMetodo2() throws java.rmi.RemoteException;
}
```

- Implementar el interfaz remoto

```
public class MiClaseRemota
 extends java.rmi.server.UnicastRemoteObject
 implements MiInterfazRemoto
{
 public MiClaseRemota() throws java.rmi.RemoteException
 {
   // Código del constructor
 }

 public void miMetodo1() throws java.rmi.RemoteException
 {
   // Aquí ponemos el código que queramos
   System.out.println("Estoy en miMetodo1()");
 }

 public int miMetodo2() throws java.rmi.RemoteException
 {
   return 5; // Aquí ponemos el código que queramos
 }

 public void otroMetodo()
 {
   // Si definimos otro método, éste no podría llamarse
   // remotamente al no ser del interfaz remoto
 }

 public static void main(String[] args)
 {
   try
   {
     MiInterfazRemoto mir = new MiClaseRemota();
     java.rmi.Naming.rebind("//" + java.net.InetAddress.getLocalHost().getHostAddress() +
                            ":" + args[0] + "/PruebaRMI", mir);
```

```
    }
    catch (Exception e)
    {
    }
  }
}
```

- Como se puede observar, la clase MiClaseRemota implementa el interfaz MiInterfazRemoto que hemos definido previamente. Además, hereda de UnicastRemoteObject, que es una clase de Java que podemos utilizar como superclase para implementar objetos remotos.
- Luego, dentro de la clase, definimos un constructor (que lanza la excepción RemoteException porque también la lanza la superclase UnicastRemoteObject), y los métodos de la/las interfaz/interfaces que implemente.
- Finalmente, en el método main, definimos el código para crear el objeto remoto que se quiere compartir y hacer el objeto remoto visible para los clientes, mediante la clase Naming y su método rebind(...).

Nota: Hemos puesto el método main() dentro de la misma clase por comodidad. Podría definirse otra clase aparte que fuera la encargada de registrar el objeto remoto.

- Compilar y ejecutar el servidor

Ya tenemos definido el servidor. Ahora tenemos que compilar sus clases mediante los siguientes pasos:

- Compilamos el interfaz remoto. Además lo agrupamos en un fichero JAR para tenerlo presente tanto en el cliente como en el servidor:

```
javac MiInterfazRemoto.java
jar cvf objRemotos.jar MiInterfazRemoto.class
```

- Luego, compilamos las clases que implementen los interfaces. Y para cada una de ellas generamos los ficheros Stub y Skeleton para mantener la referencia con el objeto remoto, mediante el comando rmic:

```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClaseRemota.java
rmic -d . MiClaseRemota
```

Observamos en nuestro directorio de trabajo que se han generado automáticamente dos ficheros .class (MiClaseRemota_Skel.class y MiClaseRemota_Stub.class) correspondientes a la capa stub-skeleton de la arquitectura RMI.

- Para ejecutar el servidor, seguimos los siguientes pasos:
  - Se arranca el registro de RMI para permitir registrar y buscar objetos remotos. El registro se encarga de gestionar un conjunto de objetos remotos a compartir, y buscarlos ante las peticiones de los clientes. Se ejecuta con la aplicación rmiregistry distribuida con Java, a la que podemos pasarle opcionalmente el puerto por el que conectar (por defecto, el 1099).

En el caso de Windows, se debe ejecutar:

```
start rmiregistry 1234
```

Y en el caso de Linux:

```
rmiregistry &
```

- Por último, se lanza el servidor:

```
java -Djava.rmi.server.hostname=127.0.0.1 MiClaseRemota 1234
```

- Crear un cliente RMI
  Vamos ahora a definir un cliente que accederá a el/los objeto/s remoto/s que creemos. Para ello seguimos los siguientes pasos:
-
  - Definir la clase para obtener los objetos remotos necesarios

La siguiente clase obtiene un objeto de tipo MiInterfazRemoto, implementado en nuestro servidor:

```
public class MiClienteRMI
{
 public static void main(String[] args)
 {
   try
   {
     MiInterfazRemoto mir = (MiInterfazRemoto)java.rmi.Naming.lookup("//" +
                         args[0] + ":" + args[1] + "/PruebaRMI");

     // Imprimimos miMetodo1() tantas veces como devuelva miMetodo2()
     for (int i=1;i<=mir.miMetodo2();i++) mir.miMetodo1();
     }
     catch (Exception e)
     {
       e.printStackTrace();
     }
   }
 }
}
```

Como se puede observar, simplemente consiste en buscar el objeto remoto en el registro RMI de la máquina remota. Para ello usamos la clase Naming y su método lookup(...).

-
  - Compilar y ejecutar el cliente

Una vez que ya tenemos definido el cliente, para compilarlo hacemos:

```
set CLASSPATH=%CLASSPATH%;.\objRemotos.jar;.
javac MiClienteRMI.java
```

Luego, para ejecutar el cliente hacemos:

```
java MiClienteRMI 127.0.0.1 1234
```

Se debe poder acceder al fichero Stub de la clase remota. Para ello, o bien lo copiamos al cliente y lo incluimos en su CLASSPATH, o lo eliminamos del CLASSPATH del servidor e incluimos su ruta en el java.rmi.codebase del servidor (si no se elimina del CLASSPATH del servidor, se ignorará la opción java.rmi.codebase, y el cliente no podrá acceder al Stub).. Si echamos un vistazo a la ventana donde está ejecutándose el servidor RMI, veremos como se ha encontrado el objeto remoto y ejecutado sus métodos:

Véase También

- CORBA
- DCOM
- SOAP
- Microsoft .NET

Enlaces Externos

- Ejemplo simple de RMI
- Java RMI tutorial Tutorial de Sun Microsystems. (en inglés)

Correo Electrónico

Correo electrónico, o en inglés **e-mail**, es un servicio de red que permite a los usuarios enviar y recibir mensajes rápidamente (también denominados **mensajes electrónicos** o **cartas electrónicas**) mediante sistemas de comunicación electrónicos. Principalmente se usa este nombre para denominar al sistema que provee este servicio en Internet, mediante el protocolo SMTP, aunque por extensión también puede verse aplicado a sistemas análogos que usen otras tecnologías. Por medio de mensajes de correo electrónico se puede enviar, no solamente texto, sino todo tipo de documentos. Su eficiencia, conveniencia y bajo costo están logrando que el correo electrónico desplace al correo ordinario para muchos usos habituales.

Origen

El correo electrónico antecede a la Internet, y de hecho, para que ésta pudiera ser creada, fue una herramienta crucial. En una demostración del MIT (*Massachusetts Institute of Technology*) de 1961, se exhibió un sistema que permitía a varios usuarios ingresar a una IBM 7094 desde terminales remotas, y así guardar archivos en el disco. Esto hizo posible nuevas formas de compartir información. El correo electrónico comenzó a utilizarse en 1965 en una supercomputadora de tiempo compartido y, para 1966, se había extendido rápidamente para utilizarse en las redes de computadoras.

En 1971 Ray Tomlinson incorporó el uso de la arroba (@). Eligió la arroba como divisor entre el usuario y la computadora en la que se aloja la casilla de correo porque en inglés @ se dice "at" (en). Así, fulano@maquina.com se lee *fulano en la máquina punto com*.

El nombre **correo electrónico** proviene de la analogía con el correo postal: ambos sirven para enviar y recibir mensajes, y se utilizan "buzones" intermedios (servidores), en donde los mensajes se guardan temporalmente antes de dirigirse a su destino, y antes de que el destinatario los revise.

Elementos

Para que una persona pueda enviar un correo a otra, ambas han de tener una **dirección de correo electrónico**. Esta dirección la tiene que dar un **proveedor de correo**, que son quienes ofrecen el servicio de envío y recepción. El procedimiento se puede hacer desde un **programa de correo** o desde un **correo web**.

Dirección de Correo

Una **dirección de correo electrónico** es un conjunto de palabras que identifican a una persona que puede enviar y recibir correo. Cada dirección es única y pertenece siempre a la misma persona.

Un ejemplo es **persona@servicio.com**, que se lee *persona arroba servicio punto com*. El signo @ (llamado arroba) siempre está en cada dirección de correo, y la divide en dos partes: el nombre de usuario (a la izquierda de la arroba; en este caso, **persona**), y el dominio en el que está (lo de la derecha de la arroba; en este caso, **servicio.com**). La arroba también se puede leer "*en*", ya que *persona@servicio.com* identifica al usuario *persona* que está **en** el servidor *servicio.com* (indica una relación de pertenencia).

Una dirección de correo se reconoce fácilmente porque siempre tiene la @, donde la @ significa "pertenece a..."; en cambio, una dirección de página web no. Por ejemplo, mientras que *http://www.servicio.com/* puede ser una página web en donde hay información (como en un libro), *persona@servicio.com* es la dirección de un correo: un buzón a donde se puede escribir.

Lo que hay a la derecha de la arroba es precisamente el nombre del *proveedor* que da el correo, y por tanto es algo que el usuario no puede cambiar. Por otro lado, lo que hay a la izquierda depende normalmente de la elección del usuario, y es un identificador cualquiera, que puede tener letras, números, y algunos signos.

Es aconsejable elegir en lo posible una dirección fácil de memorizar para así facilitar la transmisión correcta de ésta a quien desee escribir un correo al propietario, puesto que es necesario transmitirla de forma exacta, letra por letra. Un solo error hará que no lleguen los mensajes al destino.

Las letras que integran la dirección son indiferentes a que sean mayúscula o minúscula. Por ejemplo, *persona@servicio.com* es igual a *Persona@Servicio.Com*.

Proveedor de Correo

Para poder usar enviar y recibir correo electrónico, generalmente hay que estar registrado en alguna empresa que ofrezca este servicio (gratuita o de pago). El registro permite tener una *dirección de correo* personal única y duradera, a la que se puede acceder mediante un nombre de usuario y una Contraseña.

Hay varios tipos de proveedores de correo, que se diferencian sobre todo por la calidad del servicio que ofrecen. Básicamente, se pueden dividir en dos tipos: los correos gratuitos y los de pago.

Gratuitos

Los correos gratuitos son los más usados, aunque incluyen algo de publicidad: unos incrustada en cada mensaje, y otros en la interfaz que se usa para leer el correo.
Muchos sólo permiten ver el correo desde una página web propia del proveedor, para asegurarse de que los usuarios reciben la publicidad que se encuentra ahí. En cambio, otros permiten también usar un programa de correo configurado para que se descargue el correo de forma automática.

Una desventaja de estos correos es que en cada dirección, la parte que hay a la derecha de la @ muestra el nombre del proveedor; por ejemplo, el usuario *gapa* puede acabar teniendo *gapa@correo-gratuito.net*. Este tipo de direcciones desagradan a algunos (sobre todo, a empresas) y por eso es común comprar un dominio propio, para dar un aspecto más profesional.

De Pagos

Los correos de pago normalmente ofrecen todos los servicios disponibles. Es el tipo de correo que un proveedor de Internet da cuando se contrata la conexión. También es muy común que una empresa registradora de dominios venda, junto con el dominio, varias cuentas de correo para usar junto con ese dominio (normalmente, más de 1).

También se puede considerar *de pago* el método de comprar un nombre de dominio e instalar un ordenador servidor de correo con los programas apropiados (un MTA). No hay que pagar cuotas por el correo, pero sí por el dominio, y también los gastos que da mantener un ordenador encendido todo el día.

Correo Web

Casi todos los proveedores de correo dan el servicio de **correo web** (*webmail*): permiten enviar y recibir correos mediante una página web diseñada para ello, y por tanto usando sólo un programa navegador web. La alternativa es usar un *programa de correo* especializado.

El *correo web* es cómodo para mucha gente, porque permite ver y almacenar los mensajes desde cualquier sitio (en un servidor remoto, accesible por la página web) en vez de en un ordenador personal concreto.

Como desventaja, es difícil de ampliar con otras funcionalidades, porque la página ofrece unos servicios concretos y no podemos cambiarlos. Además, suele ser más lento que un *programa de correo*, ya que hay que estar continuamente conectado a páginas web y leer los correos de uno en uno.

Cliente Correo

También están los **clientes de correo electrónico**, que son programas para gestionar los mensajes recibidos y poder escribir nuevos.

Suelen incorporar muchas más funcionalidades que el *correo web*, ya que todo el control del correo pasa a estar en el ordenador del usuario. Por ejemplo, algunos incorporan potentes filtros anti-spam.

Por el contrario, necesitan que el proveedor de correo ofrezca este servicio, ya que no todos permiten usar un programa especializado (algunos sólo dan *correo web*). En caso de que sí lo permita, el proveedor tiene que explicar detalladamente cómo hay que configurar el programa de correo. Esta información siempre está en su página web, ya que es imprescindible para poder hacer funcionar el programa, y es distinta en cada proveedor. Entre los datos necesarios están: tipo de conexión (POP o IMAP), *dirección del servidor de correo*, *nombre de usuario* y *contraseña*. Con estos datos, el programa ya es capaz de obtener y descargar nuestro correo.

El funcionamiento de un *programa de correo* es muy diferente al de un *correo web*, ya que un programa de correo descarga de golpe *todos* los mensajes que tenemos disponibles, y luego pueden ser leídos sin estar conectados a Internet (además, se quedan grabados en el ordenador). En cambio, en una página web se leen de uno en uno, y hay que estar conectado a la red todo el tiempo.

Algunos ejemplos de programas de correo son Mozilla Thunderbird, Outlook Express y Eudora (ver lista completa).

Funcionamiento

Escritura del Mensaje

Se pueden mandar mensajes entre computadores personales o entre dos terminales de una computadora central. Los mensajes se archivan en un buzón (una manera rápida de mandar mensajes). Cuando una persona decide escribir un correo electrónico, su programa (o correo web) le pedirá como mínimo tres cosas:

- **Destinatario**: una o varias direcciones de correo a las que ha de llegar el mensaje
- **Asunto**: una descripción corta que verá la persona que lo reciba antes de abrir el correo
- El propio **mensaje**. Puede ser sólo texto, o incluir formato, y no hay límite de tamaño

Además, se suele dar la opción de incluir archivos *adjuntos* al mensaje. Esto permite traspasar datos informáticos de cualquier tipo mediante el correo electrónico.

Para especificar el **destinatario** del mensaje, se escribe su dirección de correo en el campo llamado **Para** dentro de la interfaz (ver imagen de arriba). Si el destino son varias personas, normalmente se puede usar una lista con todas las direcciones, separadas por comas o punto y coma.

Además del campo **Para** existen los campos *CC* y *CCO*, que son opcionales y sirven para hacer llegar copias del mensaje a otras personas:

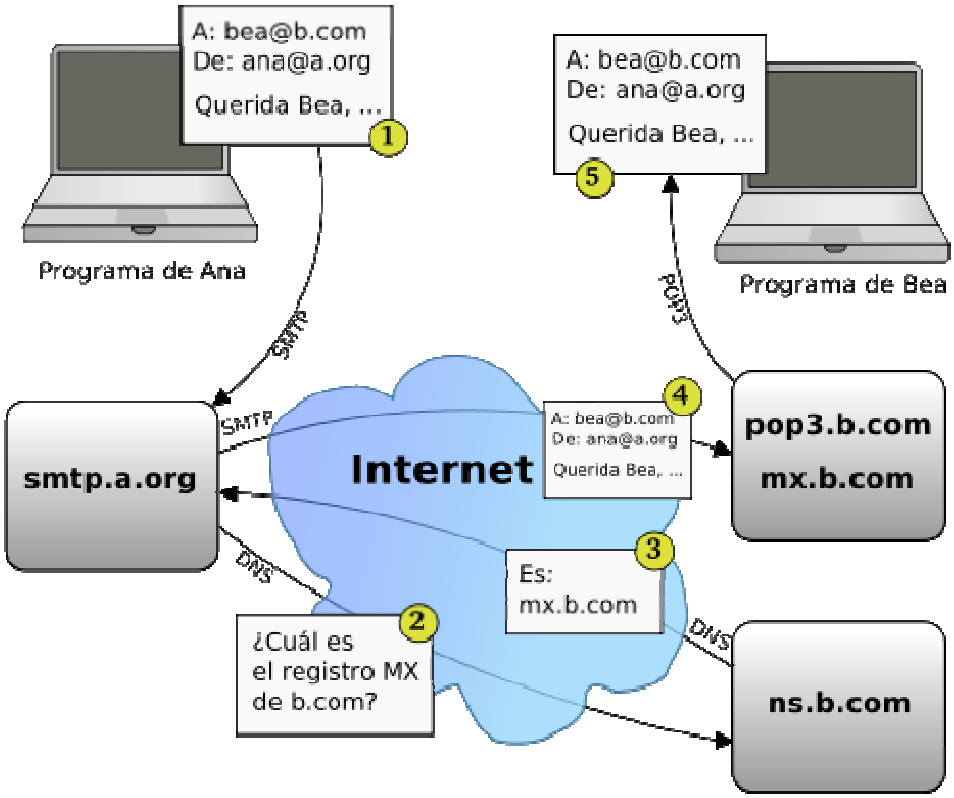
- Campo **CC** (*Copia de Carbón*): quienes estén en esta lista recibirán también el mensaje, pero verán que no va dirigido a ellos, sino a quien esté puesto en el campo **Para**. Como el campo **CC** lo ven todos los que reciben el mensaje, tanto el destinatario principal como los del campo **CC** pueden ver la lista completa.
- Campo **CCO** (*Copia de Carbón Oculta*): una variante del **CC**, que hace que los destinatarios reciban el mensaje sin aparecer en ninguna lista. Por tanto, el campo **CCO** nunca lo ve ningún destinatario.

Un ejemplo: *Ana* escribe un correo electrónico a *Beatriz* (su profesora), para enviarle un trabajo. Sus compañeros de grupo, *Carlos* y *David*, quieren recibir una copia del mensaje como comprobante de que se ha enviado correctamente, así que les incluye en el campo **CC**. Por último, sabe que a su hermano *Esteban* también le gustaría ver este trabajo aunque no forma parte del grupo, así que le incluye en el campo **CCO** para que reciba una copia sin que los demás se enteren. Entonces:

- *Beatriz* recibe el mensaje dirigido a ella (sale en el campo **Para**), y ve que *Carlos* y *David* también lo han recibido
- *Carlos* recibe un mensaje que no va dirigido a él, pero ve que aparece en el campo **CC**, y por eso lo recibe. En el campo **Para** sigue viendo a *Beatriz*
- *David*, igual que *Carlos*, ya que estaban en la misma lista (**CC**)
- *Esteban* recibe el correo de *Ana*, que está dirigido a *Beatriz*. Ve que *Carlos* y *David* también lo han recibido (ya que salen en el **CC**), pero no se puede ver a él mismo en ninguna lista, cosa que le extraña. Al final, supone que es que *Ana* le incluyó en el campo **CCO**.

Envío

El envío de un mensaje de correo es un proceso largo y complejo. Éste es un esquema de un caso típico:



En este ejemplo ficticio, *Ana* (**ana@a.org**) envía un correo a *Bea* (**bea@b.com**). Cada persona está en un servidor distinto (una en a.org, otra en b.com), pero éstos se pondrán en contacto para transferir el mensaje. Por pasos:

1. *Ana* escribe el correo en su programa cliente de correo electrónico. Al darle a *Enviar*, el programa contacta con el servidor de correo usado por *Ana* (en este caso, smtp.a.org). Se comunica usando un lenguaje conocido como protocolo SMTP. Le transfiere el correo, y le da la orden de enviarlo.
2. El servidor SMTP ve que ha de entregar un correo a alguien del dominio b.com, pero no sabe con qué ordenador tiene que contactar. Por eso consulta a su servidor DNS (usando el protocolo DNS), y le pregunta que quién es el encargado de gestionar el correo del dominio b.com. Técnicamente, le está preguntando el registro MX asociado a ese dominio.
3. Como respuesta a esta petición, el servidor DNS contesta con el nombre de dominio del servidor de correo de *Bea*. En este caso es mx.b.com; es un ordenador gestionado por el proveedor de Internet de *Bea*.
4. El servidor SMTP (smtp.a.org) ya puede contactar con mx.b.com y transferirle el mensaje, que quedará guardado en este ordenador. Se usa otra vez el protocolo SMTP.
5. Más adelante (quizás días después), *Bea* aprieta el botón "*Recibir nuevo correo*" en su programa cliente de correo. Esto empieza una conexión, mediante el protocolo POP3 o IMAP, al ordenador que está guardando los correos nuevos que le han llegado. Este ordenador (pop3.b.com) es el mismo que el del paso anterior (mx.b.com), ya que se encarga tanto de recibir correos del exterior como de entregárselos a sus usuarios. En el esquema, *Bea* recibe el mensaje de *Ana* mediante el protocolo POP3.

Ésta es la secuencia básica, pero pueden darse varios casos especiales:

- Si ambas personas están en la misma red (una Intranet de una empresa, por ejemplo), entonces no se pasa por Internet. También es posible que el servidor de correo de *Ana* y el de *Bea* sean el mismo ordenador.
- *Ana* podría tener instalado un servidor SMTP en su ordenador, de forma que el paso 1 se haría en su mismo ordenador. De la misma forma, *Bea* podría tener su servidor de correo en el propio ordenador.
- Una persona puede no usar un programa de correo electrónico, sino un webmail. El proceso es casi el mismo, pero se usan conexiones HTTP al webmail de cada usuario en vez de usar SMTP o IMAP/POP3.
- Normalmente existe más de un servidor de correo (MX) disponible, para que aunque uno falle, se siga pudiendo recibir correo.

Si el usuario quiere puede almacenar los mensajes que envía, bien de forma automática (con la opción correspondiente), bien sólo para los mensajes que así lo desee. Estos mensajes quedan guardados en la carpeta "Enviados".

Recepción

Cuando una persona recibe un mensaje de correo electrónico se puede ver en la bandeja de entrada un resumen de él:

- Remitente (o De o De: o From o From: -en inglés-): esta casilla indica quién envía el mensaje. Puede aparecer el nombre de la persona o entidad que nos lo envía (o su apodo o lo que desee el remitente). Si quien envía el mensaje no ha configurado su programa o correo web al respecto aparecerá su dirección de email
- Asunto: en este campo se ve el tema que trata el mensaje (o lo que el remitente de él desee). Si quien envía el mensaje ha dejado esta casilla en blanco se lee [ninguno] o [sin asunto]
  - Si el mensaje es una respuesta el asunto suele empezar por RE: o Re: (abreviatura de responder o *reply* -en inglés-, seguida de dos puntos). Aunque según de dónde proceda el mensaje pueden aparecer An: (del alemán *antwort*), Sv: (del sueco *svar*), etc.
  - Cuando el mensaje procede de un reenvío el asunto suele comenzar por RV: (abreviatura de reenviar) o Fwd: (del inglés *forward*), aunque a veces empieza por Rm: (abreviatura de remitir)
- Fecha: esta casilla indica cuándo fue enviado el mensaje o cuándo ha llegado a la bandeja de entrada del receptor. Puede haber dos casillas que sustituyan a este campo, una para indicar la fecha y hora de expedición del mensaje y otra para expresar el momento de su recepción

Además pueden aparecer otras casillas como:

- Tamaño: indica el espacio que ocupa el mensaje y, en su caso, fichero(s) adjunto(s)
- Destinatarios (o Para o Para: o To o To: -en inglés-): muestra a quiénes se envió el mensaje
- Datos adjuntos: si aparece una marca (habitualmente un clip) significa que el mensaje viene con uno o varios ficheros anexos
- Prioridad: expresa la importancia o urgencia del mensaje según el remitente (alta -se suele indicar con un signo de exclamación-, normal -no suele llevar marca alguna- o baja -suele indicarse con una flecha apuntando para abajo-)
- Marca (de seguimiento): si está activada (p.e. mostrando una bandera) indica que hay que tener en cuenta este mensaje (previamente lo ha marcado la persona que lo ha recibido)

- Inspeccionar u omitir: pinchando en esta casilla se puede marcar el mensaje para inspeccionarlo (suelen aparecer unas gafas en la casilla y ponerse de color llamativo -normalmente rojo- las letras de los demás campos). Pinchando otra vez se puede marcar para omitirlo (suele aparecer el símbolo de "prohibido el paso" en este campo y ponerse en un tono suave -normalmente gris- las letras de las demás casillas). Pinchando una vez más volvemos a dejar el mensaje sin ninguna de las dos marcas mencionadas
- Cuenta: Si utilizamos un cliente de correo electrónico configurado con varias cuentas de correo esta casilla indica a cuál de ellas ha llegado el mensaje en cuestión
- Primeras palabras del (cuerpo del) mensaje

Los mensajes recibidos pero sin haber sido leídos aún suelen mostrar su resumen en negrita. Después de su lectura figuran con letra normal. A veces si seleccionamos estos mensajes sin abrirlos podemos ver abajo una previsualización de su contenido.

Si el destinatario desea leer el mensaje tiene que abrirlo (normalmente haciendo (doble) clic sobre el contenido de su asunto con el puntero del ratón). Entonces el receptor puede ver un encabezado arriba seguido por el cuerpo del mensaje. En la cabecera del mensaje aparecen varias o todas las casillas arriba mencionadas (salvo las primeras palabras del cuerpo del mensaje). Los ficheros adjuntos, si existen, pueden aparecer en el encabezado o debajo del cuerpo del mensaje.

Una vez el destinatario ha recibido (y, normalmente, leído) el mensaje puede hacer varias cosas con él. Normalmente los sistemas de correo (tanto programas como webmail) ofrecen opciones como:

- **Responder**: escribir un mensaje a la persona que ha mandado el correo (que es sólo una). Existe la variante **Responder a todos**, que pone como destinatarios tanto al que lo envía como a quienes estaban en el campo *CC*
- **Reenviar** (o **remitir**): pasar este correo a una tercera persona, que verá quién era el origen y destinatario original, junto con el cuerpo del mensaje. Opcionalmente se le puede añadir más texto al mensaje o borrar los encabezados e incluso el cuerpo (o parte de él) de anteriores envíos del mensaje.
- **Marcar como spam**: separar el correo y esconderlo para que no moleste, de paso instruyendo al programa para que intente detectar mejor mensajes parecidos a éste. Se usa para evitar la publicidad no solicitada (spam)
- **Archivar**: guardar el mensaje en el ordenador, pero sin borrarlo, de forma que se pueda consultar más adelante. Esta opción no está en forma explícita, ya que estos programas guardan los mensajes automáticamente.
- **Borrar**: Se envía el mensaje a una carpeta *Elementos eliminados* que puede ser vaciada posteriormente.
- **Mover a carpeta** o **Añadir etiquetas**: algunos sistemas permiten catalogar los mensajes en distintos apartados según el tema del que traten. Otros permiten añadir marcas definidas por el usuario (ej: "trabajo", "casa", etc.).

Problemas

El principal problema actual es el **spam**, que se refiere a la recepción de correos no solicitados, normalmente de publicidad engañosa, y en grandes cantidades.
Además del *spam*, existen otros problemas que afectan a la seguridad y veracidad de este medio de comunicación:

- los **virus informáticos**, que se propagan mediante ficheros adjuntos infectando el ordenador de quien los abre
- el **phishing**, que son correos fraudulentos que intentan conseguir información bancaria
- los **engaños** (*hoax*), que difunden noticias falsas masivamente

- las **cadenas de correo electrónico**, que consisten en reenviar un mensaje a mucha gente; aunque parece inofensivo, la publicación de listas de direcciones de correo contribuye a la propagación a gran escala del *spam* y de mensajes con virus, *phishing* y *hoax*.

Precauciones Recomendables

Cuando recibamos un mensaje de correo electrónico que hable de algo que desconocemos (aunque nos lo haya mandado alguien que conocemos) conviene consultar su veracidad (por ejemplo a partir de buscadores de la web, tratando de consultar en el sitio web de la supuesta fuente de la información o en webs serias, fiables y especializadas en el tipo de información en cuestión). Sólo si estamos seguros de que lo que dice el mensaje es cierto e importante de ser conocido por nuestros contactos lo reenviaremos, teniendo cuidado de poner las direcciones de correo electrónico de los destinatarios en la casilla CCO (puede ser necesario poner sólo nuestra dirección de email en la casilla Para) y borrando del cuerpo del mensaje encabezados previos con direcciones de email (para facilitar la lectura es preferible copiar la parte del cuerpo del mensaje sin los encabezados previos y pegarla en un mensaje nuevo -o en el que aparece tras pinchar en reenviar tras borrar todo el texto, repetido a partir de previos envíos-). Así evitaremos la propagación del spam así como la de mensajes con virus (u otro tipo de malware), phishing o hoax. Conviene que hagamos saber esto a nuestros contactos en cuanto nos reenvían mensajes con contenido falso, sin utilizar la casilla CCO o sin borrar encabezados previos con direcciones de correo electrónico.

Cuando el mensaje recibido lleve uno o varios ficheros adjuntos tendremos cuidado, especialmente si el mensaje nos lo manda alguien que no conocemos. Hay peligro de que los archivos contengan virus (u otro tipo de malware). Sólo los abriremos si estamos seguros de su procedencia e inocuidad. Si, tras esto, comprobamos que los ficheros son inofensivos e interesantes para nuestros contactos podremos reenviarlo siguiendo las precauciones del párrafo anterior (en este caso, para que lleguen los ficheros adjuntos es más rápido pinchar en reenviar que crear un mensaje nuevo y volverlos a adjuntar -aunque tendremos cuidado de borrar todo el texto que repite previos reenvíos; quizá pegando después el cuerpo principal del mensaje recibido si tiene información de interés o relacionada con los archivos adjuntos-).

Cuando en un mensaje sospechoso se nos ofrezca darnos de baja de futura recepción de mensajes o de un boletín no haremos caso, es decir, no responderemos el mensaje, ni escribiremos a ninguna dirección supuestamente creada para tal fin (del tipo bajas@xxxxxxx.es o unsubscribe@xxxxxxx.com), ni pincharemos sobre un enlace para ello. Si hiciéramos algo de lo citado confirmaríamos a los spammers (remitentes de correo basura) que nuestra cuenta de correo electrónico existe y está activa y, en adelante, recibiríamos más spam. Si nuestro proveedor de correo lo ofrece podemos pinchar en "Es spam" o "Correo no deseado" o "Marcar como spam". Así ayudaremos a combatir el correo basura.

Servicios de correo electrónico

- Principales proveedores de servicios de correo electrónico gratuito:
  - Gmail: webmail, POP3 e IMAP
  - Hotmail: webmail
  - Yahoo!: webmail y POP3 con publicidad
  - Lycos: webmail

Los servicios de correo de pago los suelen dar las compañías de acceso a Internet o los registradores de dominios.
También hay servicios especiales, como Mailinator, que ofrece cuentas de correo temporales (caducan en poco tiempo) pero que no necesitan registro.

Programas para leer y organizar correos

- Principales programas
  - Evolution: Linux
  - Mail: MacOS X
  - Outlook Express: Windows
  - Thunderbird: Windows, Linux, MacOS X

Puede ver una lista más larga en el artículo **lista de clientes de correo electrónico**.

Programas servidores de correo

Éstos son usados en el ordenador servidor de correo para proporcionar el servicio a los clientes, que podrán usarlo mediante un *programa de correo*.

- Principales programas servidores
  - Microsoft Exchange Server: Windows
  - MailEnable: Windows
  - Exim: Unix
  - Sendmail: Unix
  - Qmail: Unix
  - Postfix: Unix
- Gestión de correo electrónico
  - Duroty: alternativa libre a Gmail

También existen otros programas para dar el servicio de *webmail*.

Véase También

- Lista de correo electrónico
- Tecnologías usadas en el correo electrónico:
  - MIME
  - SMTP
  - POP3
  - IMAP

Enlaces Externos

- Historia del correo electrónico:
  - 1794-2001
  - 2002
  - 2003
  - 2004

JMS

La API de **Servicios de Mensajería de Java** (también conocida por sus siglas **JMS**) es la solución creada por SUN para el uso de colas de mensajes. Este es un estándar de mensajería que permite a los componentes de aplicaciones basados en la plataforma de Java 2 crear, enviar, recibir y leer mensajes. También hace posible la comunicación confiable de manera síncrona y asíncrona.

El servicio de mensajería también es conocido como *Middleware Orientado a Mensajes* (**MOM** por sus siglas en inglés) y es una herramienta universalmente reconocida para la construcción de aplicaciones empresariales.

Dicha API es parte integral de la versión 2 de Java.

Existen dos modelos de la API JSM, los cuales son:

Modelo Punto a Punto (point to point): Este modelo cuenta con solo dos clientes, uno que envía el mensaje y otro que lo recibe. Este modelo asegura llegar su mensaje ya que si el receptor no esta disponible para aceptar el mensaje o atenderlo, de cualquier forma se le envía el mensaje y este se encola en una pila del tipo FIFO para luego ser recibido segun haya entrado.

Modelo Publicador/Suscriptor (Publish/Subscribe): Este modelo cuenta con varios clientes, unos que publican temas(tópicos) o eventos, y los que ven estos tópicos, a diferencia del modelo punto a punto este modelo tiende a tener más de un consumidor.
Ambos modelos pueden ser síncronos mediante el método receive y asíncronos por medio de un MessageListener.

Implementación

Para enviar o recibir mensajes los clientes tienen que conectarse por medio de los objetos administrados, este nombre lo reciben porque son creados por el administrador (j2eeadmin). Estos implementan las interfaces JMS y se sitúan en el espacio de nombres JNDI (Java Naming and Directory Interface) para que así los clientes puedan solicitarlos.

Hay dos tipos de objetos administrados en JMS:
- ConnectionFactory: Se usa para crear una conexión al proveedor del sistema de mensajes.
- Destination: Son los destinos de los mensajes que se envían y el recipiente de los mensajes que se reciben.

Mensajes:
Estos son la base de los JMS, se componen de tres elementos.
1.Header: Es la cabecera del mensaje, le sirve a los clientes y a los proveedores para poder identificarlos.
2.Properties: Son propiedades en general para personalizar y/o hacer más especifico un mensaje.
3.Body: Es el mensaje en sí mismo, hay varios tipos de “cuerpos” que puede llevar un mensaje:
- StreamMessage: Contiene un flujo (stream) de datos que se escriben y leen de manera secuencial.
- MapMessage: Contiene pares nombre-valor.
- TextMessage: Contiene un *String*.
- ObjectMessage: Contiene un objeto que implemente la interfaz *Serializable*.
- BytesMessage: Contiene un stream de bytes.

Clientes JMS:
Son clientes todos aquellos que envíen mensajes, como aquellos que los reciban, en algunos textos a los que envían mensajes son llamados proveedores y los que reciben consumidores o clientes. Para enviar o recibir mensajes tienen que tener una serie de pasos:
1.Conseguir un objeto *ConnectionFactory* a través de JNDI.
2.Conseguir un destino, mediante el objeto *Destination* a través de JNDI.
3.Usar *ConnectionFactory* para conseguir un objeto *Connection*.
4.Usar *Destination* para crear un objeto *Session*.

Ejemplos de Sistemas Comerciales de Cola de Mensajes

WebSphere MQ de IBM (antes MQ*Series)
Message Queue de Microsoft (MSMQ)
Java Message Service de Sun (JMS)
Data Distribution Service del OMG (DDS)

Enlaces Externos

- API y documentación
- Ejemplos de Uso de ambos modelos

Servicios Web

Un **servicio web** (en inglés *Web service*) es una colección de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son los comités responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web se ha creado el organismo WS-I, encargado de desarrollar diversos perfiles para definir de manera más exhaustiva estos estándares.

Estándares Empleados

- Web Services Protocol Stack: Así se denomina al conjunto de servicios y protocolos de los servicios Web.
- XML (Extensible Markup Language): Es el formato estándar para los datos que se vayan a intercambiar.
- SOAP (Simple Object Access Protocol) o XML-RPC (XML Remote Producer Call): Protocolos sobre los que se establece el intercambio.
- Otros protocolos: los datos en XML también pueden enviarse de una aplicación a otra mediante protocolos normales como HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol), o SMTP (Simple Mail Transfer Protocol).
- WSDL (Web Services Description Languages): Es el lenguaje de la interfaz pública para los servicios Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web.
- UDDI (Universal Description, Discovery and Integration): Protocolo para publicar la información de los servicios Web. Permite comprobar qué servicios web están disponibles.
- WS-Security (Web Service Security): Protocolo de seguridad aceptado como estándar por OASIS (Organization for the Advancement of Structured Information Standards). Garantiza la autenticación de los actores y la confidencialidad de los mensajes enviados...

Ventajas de los Servicios Web

- Aportan interoperabilidad entre aplicaciones de software independientemente de sus propiedades o de las plataformas sobre las que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad *firewall* sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y software de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.
- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar.

Inconvenientes de los Servicios Web

- Para realizar transacciones no pueden compararse en su grado de desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- Su rendimiento es bajo si se compara con otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA, o DCOM (Distributed Component Object Model). Es uno de los inconvenientes derivados de adoptar un formato basado en texto. Y es que entre los objetivos de XML no se encuentra la concisión ni la eficacia de procesamiento.
- Al apoyarse en HTTP, pueden esquivar medidas de seguridad basadas en *firewall* cuyas reglas tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.
- Https...

Razones para crear Servicios Web

La principal razón para usar servicios Web es que se basan en HTTP sobre TCP (Transmission Control Protocol) en el puerto 80. Dado que las organizaciones protegen sus redes mediante *firewalls* -que filtran y bloquean gran parte del tráfico de Internet-, cierran casi todos los puertos TCP salvo el 80, que es, precisamente, el que usan los navegadores. Los servicios Web se enrutan por este puerto, por la simple razón de que no resultan bloqueados.

Otra razón es que, antes de que existiera SOAP, no había buenas interfaces para acceder a las funcionalidades de otros ordenadores en red. Las que había eran *ad hoc* y poco conocidas, tales como EDI (Electronic Data Interchange), RPC, u otras Application Programming Interface APIs.

Una tercera razón por la que los servicios Web son muy prácticos es que pueden aportar gran independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, los cambios a lo largo del tiempo en uno no deben afectar al otro. Esta flexibilidad será cada vez más importante, dado que la tendencia a construir grandes aplicaciones a partir de componentes distribuidos más pequeños es cada día más acusada.

Se espera que para los próximos años mejoren la calidad y cantidad de servicios ofrecidos basados en los nuevos estándares.

Plataformas

Servidores de aplicaciones para servicios Web:

- IBM Lotus Domino a partir de la versión 7.0
- Axis y el servidor Jakarta Tomcat (de Apache)
- ColdFusion MX de Macromedia
- Java Web Services Development Pack (JWSDP) de Sun Microsystems (basado en Jakarta Tomcat)
- JOnAS (parte de *ObjectWeb* una iniciativa de código abierto)
- Microsoft .NET
- Novell exteNd (basado en la plataforma J2EE)
- WebLogic
- WebSphere

- Zope es un servidor de aplicaciones Web orientado a objetos desarrollado en el lenguaje de programación Python
- VERASTREAM de AttachmateWRQ para modernizar o integrar aplicaciones host IBM y VT
- Proyecto Mono, en inglés.

Temas Relacionados

- World Wide Web
- Web semántica

XML

Extensible Markup Language (XML)	
Extensión de archivo:	.xml
Tipo de MIME:	application/xml, text/xml
Desarrollado por:	World Wide Web Consortium
Tipo de formato:	Lenguaje de marcado
Extendido de:	SGML
Extendido a:	XHTML, RSS, Atom, ...
Estándar(es):	1 (Fourth Edition) 2 (Second Edition)

XML, sigla en inglés de *Extensible Markup Language* («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.
XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

Historia

XML proviene de un lenguaje inventado por IBM en los años setenta, llamado GML (*Generalized Markup Language*), que surgió por la necesidad que tenía la empresa de almacenar grandes cantidades de información. Este lenguaje gustó a la ISO, por lo que en 1986 trabajaron para normalizarlo, creando SGML (*Standard Generalized Markup Language*), capaz de adaptarse a un gran abanico de problemas. A partir de él se han creado otros sistemas para almacenar información.

En el año 1989 Tim Berners Lee creó la web, y junto con ella el lenguaje HTML. Este lenguaje se definió en el marco de SGML y fue de lejos la aplicación más conocida de este estándar. Los navegadores web sin embargo siempre han puesto pocas exigencias al código HTML que interpretan y así las páginas web son caóticas y no cumplen con la sintaxis. Estas páginas web dependen fuertemente de una forma específica de lidiar con los errores y las ambigüedades, lo que hace a las páginas más frágiles y a los navegadores más complejos.

Otra limitación de SGML es que cada documento pertenece a un vocabulario fijo, establecido por el DTD. No se pueden combinar elementos de diferentes vocabularios. Asimismo es imposible para un intérprete (por ejemplo un navegador) analizar el documento sin tener conocimiento de su gramática (del DTD). Por ejemplo, el navegador sabe que antes de una etiqueta ** ** debe haberse cerrado cualquier **<p>** previamente abierto. Los navegadores resolvieron esto incluyendo lógica ad hoc para el HTML, en vez de incluir un analizador genérico. Ambas opciones, de todos modos, son muy complejas para los navegadores. Se buscó entonces definir un subconjunto del SGML que permita:

- Mezclar elementos de diferentes lenguajes. Es decir que los lenguajes sean extensibles.
- La creación de analizadores simples, sin ninguna lógica especial para cada lenguaje.
- Empezar de cero y hacer hincapié en que no se acepte nunca un documento con errores de sintaxis.

Para hacer esto XML deja de lado muchas características de SGML que estaban pensadas para facilitar la escritura manual de documentos. XML en cambio está orientado a hacer las cosas más sencillas para los programas automáticos que necesiten interpretar el documento.

Ventajas del XML

- Es extensible, lo que quiere decir que una vez diseñado un lenguaje y puesto en producción, igual es posible extenderlo con la adición de nuevas etiquetas de manera de que los antiguos consumidores de la vieja versión todavía puedan entender el nuevo formato.
- El analizador es un componente estándar, no es necesario crear un analizador específico para cada lenguaje. Esto posibilita el empleo de uno de los tantos disponibles. De esta manera se evitan bugs y se acelera el desarrollo de la aplicación.
- Si un tercero decide usar un documento creado en XML, es sencillo entender su estructura y procesarlo. Mejora la compatibilidad entre aplicaciones.

Estructura de un Documento XML

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de pedazos de información. Ejemplos son un tema musical, que se compone de compases, que están formados a su vez con notas. Estas partes se llaman *elementos*, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma **<nombre>**, donde *nombre* es el nombre del elemento que se está señalando.
A continuación se muestra un ejemplo para entender la estructura de un documento XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE Edit_Mensaje SYSTEM "Lista_datos_mensaje.dtd"
                [<!ELEMENT Edit_Mensaje (Mensaje)*>]>

<Edit_Mensaje>

    <Mensaje>

        <Remitente>
            <Nombre>Nombre del remitente</Nombre>
            <Mail> Correo del remitente </Mail>
        </Remitente>

        <Destinatario>
            <Nombre>Nombre del destinatario</Nombre>
            <Mail>Correo del destinatario</Mail>
        </Destinatario>

        <Texto>
            <Parrafo>
                Este es mi documento con una estructura muy sencilla
                no contiene atributos ni entidades....
            </Parrafo>
        </Texto>
    </Mensaje>
</Edit_Mensaje>
```

Aquí está el ejemplo de código del DTD del documento "Edit_Mensaje":

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- Este es el DTD de Edit_Mensaje -->

<!ELEMENT Mensaje (Remitente, Destinatario, Asunto, Texto)*>
        <!ELEMENT Remitente (Nombre, Mail)>
                <!ELEMENT Nombre (#PCDATA)>
                <!ELEMENT Mail   (#PCDATA)>

        <!ELEMENT Destinatario (Nombre, Mail)>
                <!ELEMENT Nombre (#PCDATA)>
                <!ELEMENT Mail   (#PCDATA)>

        <!ELEMENT Asunto (#PCDATA)>

        <!ELEMENT Texto  (Parrafo)>
                <!ELEMENT Parrafo (#PCDATA)>
```

Documentos XML bien formados

Los documentos denominados como "bien formados" (del inglés *well formed*) son aquellos que cumplen con todas las definiciones básicas de formato y pueden, por lo tanto, ser analizados correctamente por cualquier analizador sintáctico (*parser*) que cumpla con la norma. Se separa esto del concepto de validez que se explica más adelante.

- Los **documentos han de seguir una estructura** estrictamente jerárquica con lo que respecta a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida en otra, es decir, las etiquetas deben estar correctamente anidadas. Los elementos con contenido deben estar correctamente cerrados.
- Los documentos XML sólo permiten un elemento raíz del que todos los demás sean parte, es decir, solo pueden tener un elemento inicial.
- Los valores atributos en XML siempre deben estar encerrados entre comillas simples o dobles.
- El XML es sensible a mayúsculas y minúsculas. Existe un conjunto de caracteres llamados espacios en blanco (espacios, tabuladores, retornos de carro, saltos de línea) que los procesadores XML tratan de forma diferente en el marcado XML.
- Es necesario asignar nombres a las estructuras, tipos de elementos, entidades, elementos particulares, etc. En XML los nombres tienen alguna característica en común.
- Las construcciones como etiquetas, referencias de entidad y declaraciones se denominan marcas; son partes del documento que el procesador XML espera entender. El resto del documento entre marcas son los datos "entendibles" por las personas.

Partes de un documento XML

Un documento XML está formado por el prólogo y por el cuerpo del documento.

Prólogo

Aunque no es obligatorio, los documentos XML pueden empezar con unas líneas que describen la versión XML, el tipo de documento y otras cosas.
El prólogo contiene:

- Una declaración XML. Es la sentencia que declara al documento como un documento XML.
- Una declaración de tipo de documento. Enlaza el documento con su DTD, o el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.
- Uno o más comentarios e instrucciones de procesamiento.

Cuerpo

A diferencia del prólogo, el cuerpo no es opcional en un documento XML, el cuerpo debe contener un y solo un elemento raíz, característica indispensable también para que el documento esté bien formado.

Elementos

Los elementos XML pueden tener contenido (más elementos, caracteres o ambos), o bien ser elementos vacíos.

Atributos

Los elementos pueden tener atributos, que son una manera de incorporar características o propiedades a los elementos de un documento. Deben de ir entre comillas.

Entidades Predefinidas

Entidades para representar caracteres especiales para que, de esta forma, no sean interpretados como marcado en el procesador XML.

Secciones CDATA

Es una construcción en XML para especificar datos utilizando cualquier carácter sin que se interprete como marcado XML. Solo se utiliza en los atributos. No confundir con 2(#PCDATA) que es para los elementos.

Comentarios

Comentarios a modo informativo para el programador que han de ser ignorados por el procesador.
Los comentarios en XML tienen el siguiente formato:

```
<!--- Esto es un comentario --->
  <!-- Otro comentario -->
```

Validez

Que un documento sea "bien formado" solamente habla de su estructura sintáctica básica, es decir que se componga de elementos, atributos y comentarios como XML manda que se escriban. Ahora bien, cada aplicación de XML, es decir cada lenguaje definido con esta tecnología, necesitará especificar cuál es exactamente la relación que debe verificarse entre los distintos elementos presentes en el documento.

Esta relación entre elementos se especifica en un documento externo o definición (expresada como DTD *(Document Type Definition = Definición de Tipo de Documento)* o como XSchema). Crear una definición equivale a crear un nuevo lenguaje de marcado, para una aplicación específica.

Document Type Definition (DTD)

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos. Los documentos XML que se ajustan a su DTD se denominan válidos.

Declaraciones Tipo Elemento

Los elementos deben ajustarse a un tipo de documento declarado en una DTD para que el documento sea considerado como válido.

Modelos de Contenidos

Un modelo de contenido es un patrón que establece los subelementos aceptados, y el orden en que se aceptan.

Declaraciones de Lista Atributos

Los atributos se usan para añadir información adicional a los elementos de un documento.

Tipos de Atributos

- Atributos CDATA y NMTOKEN
- Atributos enumerados y notaciones
- Atributos ID e IDREF

Declaraciones de Entidades

XML hace referencia a objetos que no deben ser analizados sintácticamente según las reglas XML, mediante el uso de entidades. Las entidades pueden ser:

- Internas o externas
- Analizadas o no analizadas
- Generales o parametrizadas

Espacios de Nombres

Los espacios de nombres XML permiten separar semánticamente los elementos que forman un documento XML.

XML Schemas (XSD)

Un Schema es algo similar a un DTD, define qué elementos puede contener un documento XML, cómo están organizados y qué atributos y de qué tipo pueden tener sus elementos.

Ventajas de los Schemas frente a los DTD's

- Usan sintaxis de XML, al contrario que los DTDs.
- Permiten especificar los tipos de datos.
- Son extensibles.

Herramientas para trabajar con documentos XML

De hecho cualquier procesador de texto, que sea capaz de producir archivos txt es capaz de generar XML, aunque en los entornos de desarrollo como Eclipse o Visual Studio, se facilita, ya que reconoce los formatos y ayuda a generar un XML bien formado.

Lista de Tecnologías XML

Extended Stylesheet Languaje (XSL)

EL Lenguaje de Hoja de Estilo Extensible (e***X***tensible ***S***tylesheet ***L***anguage, XSL) es una familia de lenguajes que permiten describir como los archivos codificados en xml serán formateados (para mostrarlos) o transformados. Hay tres lenguajes en esta familia: XSL Transformations (XSLT), XSL Formatting Objects (XSL-FO)y XML Path Language.

Lenguaje de Enlace XML (XLINK)

XLink es una aplicación XML que intenta superar las limitaciones que tienen los enlaces de hipertexto en HTML. Es una especificación que todavía está en desarrollo.

Otras Tecnologías

- Hojas de estilos
  - XSL-FO
  - XSLT
  - XLink
  - XPointer
  - XSL
  - hojas de estilo en cascada (CSS)
  - XLT (*X*ML representation of *L*exicons and *T*erminologies)
- Programación
  - JDOM
  - SAX
  - STAX
  - VTD-XML
- Consulta de datos
  - XQuery
  - Xpath
- Seguridad
  - Xades (XML Advanced Electronic Signatures)

Hay quien opina que XML es demasiado pesado para algunas aplicaciones y difícil de editar con un editor de texto simple. Por ello merece la pena mencionar algunas alternativas más ligeras y simples. Los lenguajes de marcas ligeros:

- Simple Outline XML: es un XML simplificado que se puede convertir sin problemas en XML *completo*.
- YAML y OGDL. Estos dos son ficheros de solo texto que no están emparentados con XML como el SOX, antes comentado. Más Información
- BBCode. Éste tiene un uso muy restringido para dar formato nada más.
- Slip

También hay por lo menos un lenguaje basado en XML en formato binario, llamado EBML.

Véase También

- XML-RPC Protocolo de llamada de procedimiento remoto (RPC) que utiliza XML para codificar los datos.
- SOAP Protocolo de llamadas a métodos remotos e intercambio de mensajes XML utilizando tecnología de objetos.

Enlaces Externos

- Introducción al XML Introducción conceptual al XML.
- programación.com Cursos, tutoriales, trucos, etc.
- El estándar XML (en inglés)
- The Extensible Stylesheet Language Family (XSL) (en inglés)
- XML.org (en inglés)
- XML Software Guide (en inglés)
- Página de XML de Microsoft (en inglés)
- Página oficial (en inglés)
- XML Document Authoring Tools (en inglés)
- XML módulos Perl en CPAN (en inglés)
- XeML.net
- Introducción básica al XML

Enterprise JavaBeans

Los **Enterprise JavaBeans** (también conocidos por sus siglas **EJB**) son una de las API que forman parte del estándar de construcción de aplicaciones empresariales J2EE de Sun Microsystems (ahora JEE 5.0). Su especificación detalla cómo los servidores de aplicaciones proveen objetos desde el lado del servidor que son, precisamente, los EJBs:

- comunicación remota utilizando CORBA
- transacciones
- control de la concurrencia
- eventos utilizando JMS (Java messaging service)
- servicios de nombres y de directorio
- seguridad
- ubicación de componentes en un servidor de aplicaciones.

La especificación de Enterprise Java Bean define los papeles jugados por el contenedor de EJB y los EJBs, además de disponer los EJBs en un contenedor.

Definición

Los EJBs proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los EJBs es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad, ...) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

No hay que confundir los Enterprise JavaBeans con los JavaBeans. Los JavaBeans también son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones, pero no pueden utilizarse en entornos de objetos distribuidos al no soportar nativamente la invocación remota (RMI).

Tipos de Enterprise JavaBeans

Existen tres tipos de EJBs:

- **EJBs de Entidad** (*Entity EJBs*): su objetivo es encapsular los objetos del lado del servidor que almacena los datos. Los EJBs de entidad presentan la característica fundamental de la persistencia:
  - **Persistencia gestionada por el contenedor** (CMP): el contenedor se encarga de almacenar y recuperar los datos del objeto de entidad mediante el mapeo de una tabla de la base de datos.
  - **Persistencia gestionada por el bean** (BMP): el propio objeto entidad se encarga, mediante una base de datos u otro mecanismo, de almacenar y recuperar los datos a los que se refiere, por lo cual, la responsabilidad de implementar los mecanismos de persistencia es del programador.
- **EJBs de Sesión** (*Session EJBs*): gestionan el flujo de la información en el servidor. Generalmente sirven a los clientes como una fachada de los servicios proporcionados por otros componentes disponibles en el servidor. Puede haber dos tipos:
  - **con estado** (*stateful*). Los beans de sesión con estado son objetos distribuidos que poseen un estado. El estado no es persistente, pero el acceso al bean se limita a un solo cliente.
  - **sin estado** (*stateless*). Los beans de sesión sin estado son objetos distribuidos que carecen de estado asociado permitiendo por tanto que se los acceda concurrentemente. No se garantiza que los contenidos de las variables de instancia se conserven entre llamadas al método.

- **EJBs dirigidos por mensajes** (*Message-driven EJBs*): son los únicos beans con funcionamiento asíncrono. Usando el *Java Messaging System* (JMS), se suscriben a un tema (*topic*) o a una cola (*queue*) y se activan al recibir un mensaje dirigido a dicho tema o cola. No requieren de su instanciación por parte del cliente.

Funcionamiento de un Enterprise JavaBean

Los EJBs se disponen en un contenedor EJB dentro del servidor de aplicaciones. La especificación describe cómo el EJB interactúa con su contenedor y cómo el código cliente interactúa con la combinación del EJB y el contenedor.

Cada EJB debe facilitar una clase de implementación Java y dos interfaces Java. El contenedor EJB creará instancias de la clase de implementación Java para facilitar la implementación EJB. Las interfaces Java son utilizadas por el código cliente del EJB. Las dos interfaces, conocidas como interfaz "home" e interfaz remoto, especifican las firmas de los métodos remotos del EJB. Los métodos remotos se dividen en dos grupos:

- métodos que no están ligados a una instancia específica, por ejemplo aquellos utilizados para crear una instancia EJB o para encontrar una entidad EJB existente. Estos métodos se declaran en la interfaz "home".
- métodos ligados a una instancia específica. Se ubican en la interfaz remota.

Dado que se trata simplemente de interfaces Java y no de clases concretas, el contenedor EJB genera clases para esas interfaces que actuarán como un proxy en el cliente. El cliente invoca un método en los proxies generados que a su vez sitúa los argumentos método en un mensaje y envía dicho mensaje al servidor EJB. Los proxies usan RMI-IIOP para comunicarse con el servidor EJB.

El servidor llamará a un método correspondiente a una instancia de la clase de implementación Java para manejar la llamada del método remoto.

Interfaz Home

Como indicamos anteriormente, la interfaz "home" permite al código cliente manipular ciertos métodos de clase del EJB, esto es, métodos que no están asociados a ninguna instancia particular.
La especificación EJB 1.1 establece el tipo de métodos de clase que se pueden definir como métodos que crean un EJB o para encontrar un EJB existente si es un "bean" de entidad.
La especificación EJB 2.0 permite a los desarrolladores de aplicaciones definir nuevos métodos de clase sin limitarse a su sola creación, borrado y búsqueda...

Interfaz Remota

La interfaz remota especifica los métodos de instancia públicos encargados de realizar las operaciones.
Una sesión bean puede implementar múltiples interfaces, con cada interfaz apuntada por un tipo de cliente diferente. Interfaz local es para aquellos clientes que corren en la misma maquina virtual que el contenedor EJB. Interfaz remote es para clientes remotos. Frente a una consulta del cliente, el contenedor retorna un stub serializado del objeto que implementa la interfaz remote. El stub conoce como pasar llamadas a procedimientos remotos (RPCs) al servidor. Este tipo de interfaz es también un POJI

Clase de Implementación EJB

Las clases de implementación EJB las suministran los desarrolladores de aplicaciones, que facilitan la lógica de negocio ("business logic") o mantienen los datos ("business data") de la interfaz de objeto, esto es, implementan todos los métodos especificados por la interfaz remota y, posiblemente, algunos de los especificados por la interfaz "home".

Correspondencia entre Métodos de Interfaz y Métodos de Implementación

Las llamadas al método en la interfaz "home" se remiten al método correspondiente de la clase de implementación del bean con el prefijo 'ejb' añadido y con la primera letra de la interfaz "home" convertida en mayúscula y manteniendo exactamente el mismo tipo de argumentos. Por ejemplo:
create ---> ejbCreate.
Las llamadas a métodos en la interfaz remota se remiten al método de implementación correspondiente del mismo nombre y argumentos

Historia de los Enterprise JavaBeans

La especificación EJB ha ido evolucionando a la par que lo hacía la propia especificación J2EE. Las diferentes versiones que han existido hasta la fecha son:

- EJB 1.0: la especificación original
- EJB 1.1: la primera incluida dentro de J2EE
- EJB 2.0: incluida en J2EE 1.3, añadía las interfaces Locales y los Message-Driven beans.
- EJB 2.1: incluida en la última revisión de J2EE, la 1.4.
- EJB 3.0: Ahora con Cluster y esta incluida en JEE 5.0

Existe una propuesta de una nueva revisión de la especificación de Enterprise JavaBeans, la EJB 3.0, cuyo objetivo es simplificar la implementación de beans sin por ello sacrificar su potencia y flexibilidad.

La nueva especificación de EJB 3.0 simplifica el proceso de creación de EJB y facilita la implementación de persistencia.
Esta especificación estará disponible en la nueva versión de J2EE nombrada JEE 5.0.

Bibliografía Recomendada

- *Enterprise JavaBeans, 4th edition*, (2004) de Richard Monson-Haefel [O'Reilly, ISBN 059600530X]
- *Mastering Enterprise JavaBeans, 2nd edition* (2001), de Ed Roman [Wiley&Sons, ISBN 0471417114]
- *Bitter EJB* (2003) de Bruce Tate [Manning, ISBN 1930110952]
- *Head First EJB* (2003), de Kathy Sierra y Bert Bates [O'Reilly, ISBN 0596005717]

Enlaces Externos

- Página de Sun sobre EJB
- Página de JBoss sobre EJB3
- Ejemplo sencillo de EJB

Jave Servlet

Los **servlets** son objetos que corren dentro del contexto de un contenedor de servlets (ej: Tomcat) y extienden su funcionalidad. También podrían correr dentro de un servidor de aplicaciones (ej: OC4J Oracle) que además de contenedor para servlet tendrá contenedor para objetos más avanzados como son los EJB (Tomcat sólo es un contenedor de servlets).

La palabra *servlet* deriva de otra anterior, *applet*, que se refería a pequeños programas escritos en Java que se ejecutan en el contexto de un navegador web. Por contraposición, un *servlet* es un programa que se ejecuta en un servidor.

El uso más común de los *servlets* es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

Aspectos Técnicos

Un servlet es un objeto que se ejecuta en un servidor o contenedor JEE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor web, generalmente es HTML. Otras opciones que permiten generar contenido dinámico son con los lenguajes ASP, PHP, JSP (un caso especial de servlet) y Python.

Los *servlets* forman parte de JEE (Java Enterprise Edition), que es una ampliación de JSE (Java Standard Edition).

Un *servlet* es un objeto Java que implementa la interfaz javax.servlet.Servlet o hereda alguna de las clases más convenientes para un protocolo específico (ej: javax.servlet.HttpServlet). Al implementar esta interfaz el servlet es capaz de interpretar los objetos de tipo HttpServletRequest y HttpServletResponse quienes contienen la información de la página que invocó al *servlet*.

Entre el servidor de aplicaciones (o web content) y el *servlet* existe un contrato que determina cómo han de interactuar. La especificación de éste se encuentra en los JSR (Java Specification Requests) del JCP (Java Community Process).

A fecha 2 de Julio de 2004, la especificación más reciente de los *servlets* se encuentra en el JSR 154, que define la versión 2.4 de los *servlets*.

Historia

La especificación original de Servlets fue creada por Sun Microsystems (la version 1.0 fue terminada en junio de 1997). Comenzando con la version 2.3, la especificación de Servlet fue desarrollada siguiendo el Proceso de la Comunidad Java (Java Community Process).

Portlet

Los ***portlets*** son componentes modulares de interfaz de usuario gestionadas y visualizadas en un portal web. Los *portlets* producen fragmentos de código de marcado que se agregan en una página de un portal. Típicamente, siguiendo la metáfora de escritorio, una página de un portal se visualiza como una colección de ventanas de *portlet* que no se solapan, donde cada una de estas muestra un *portlet*. Por lo tanto un *portlet* (o colección de *portlets*) se asemeja a una aplicación web que está hospedada en un portal. Como por ejemplo, un porlet de aplicación puede ser para el correo, el parte meteorológico, un foro, noticias, etc.

Se pretende que los estándares de los *portlets* permitan al desarrollador de software a que cree portlets que puedan ser utilizados en cualquier portal que soporte estos estándares.
Los portlets son similares a los servlets en que:

- Los portlets son manejados por un contenedor especializado
- Los portlets generan contenido dinámicamente
- El ciclo de vida de los portlets es controlado por el contenedor
- Los portlets interactúan con el cliente web mediante el uso del paradigma request/response

Los portlets son diferentes a los servlets en que:

- Los portlets son únicamente generados como fragmento de etiquetado y no como documentos completos.
- Los portlets son accesibles directamente a una URL
- Los portlets no pueden generar contenido arbitrario, ya que el contenido de los portlets va a estar incluido la página del portal. Si un servidor de un portal esta solicitando html/text, entonces todos los portlets deben ser generados en text/html. Por otro lado si el servidor del portal esta solicitando por WML, entonces cada portal deberá ser generado en contenido WML.

Funcionalidades Adicionales que Proporcionan los Portlets

Almacenamiento persistente para las preferencias: Los portlets proporcionan un objeto PortletPreferences para almacenar las preferencias de usuario. Estas preferencias son almacenadas en una base de datos persistente así estas se encontrarán disponibles así el servidor se reinicie. Como desarrollador no es necesario preocuparse por la implementación del almacenamiento.

Procesamiento de solicitudes: Los portlets disponen de una manipulación de peticiones más refinada. Un portlet puede obtener su solicitud cuando el usuario hace alguna acción sobre este. (Un estado llamado action phase (Fase de acción)), o porque el usuario adopto medidas sobre otro portlet y la pagina necesita se actualizada. Un portal dispone de diferentes métodos callback para el manejo de ambas situaciones.

Modos de los Portlets: Los portlets usan el concepto de mode para indicar que esta haciendo el usuario. Cuando usamos una aplicación de correo electrónico, esta puede ser usada para leer, escribir o revisar los mensajes del correo -- Estas se esperan que sean las funcionalidades que posee una aplicación de correo electrónico. Los portlets normalmente proporcionan esto en un modo Vista (VIEW). Pero hay otras actividades, como especificar el tiempo de actualización o la (re-)configuración de datos como el nombre de usuario y la contraseña. Estas actividades permiten al usuario configurar el comportamiento de la aplicación, por lo que se encuentran bajo el modo EDITAR (EDIT). La funcionalidad de ayuda de una aplicación de correo se enmarca sobre el modo de AYUDA (HELP). De esta manera para la lógica de negocio es necesario relacionar lógicamente un método doView() para el modo de vista, de igual manera doEdit() para la configuración de la aplicación y otro método doHelp() para lo relacionado con la ayuda. Esto hace sencillo para el administrador controlar el acceso al portlet, porque todo lo que se tiene que

hacer es cambiar los derechos de acceso del portlet y de esta manera establecer que cosas el usuario permite hacer.

Estado de la ventana: El estado de una ventana determina la cantidad de espacio podría asignársele al contenido generado por un portlet sobre el portal. Si se da clic en el botón maximizar el portlet utiliza todo el espacio disponible en la pantalla, de igual forma si este pasa a estado minimizado únicamente se mostrara la barra de titulo asociada al portlet.

Información de Usuario: Comúnmente, los portlets proporcionan contenido personalizado de acuerdo a los requerimientos del mismo. Para hacer esto efectivamente, es necesario contar con atributos como nombre, correo electrónico, teléfono, etc. El API de portlet dispone para esto el concepto de atributos de usuario (user attributes).

Estándares de Portlets

El propósito del protocolo WSRP (Web Services for Remote Portlets) es suministrar un estándar de servicios web que permita el "plug-and-play" de portlets en ejecución remotos desde fuentes dispares. Muchos sites permiten a los usuarios registrados personalizar su vista del sitio web activando o desactivando porciones de la página web, o añadiendo o eliminando características. Esto normalmente se realiza por parte de un conjunto de '***portlets'*** que juntos forman el portal.

La especificación Java Portlet (JSR168) permite la interoperabilidad de los portlets entre portales web diferentes. Esta especificación define un conjunto de API para interacción entre el contenedor portlet y el portlet que direcciona áreas de personalización, presentación y seguridad.

Apache Pluto es una implementación de referencia de JSR168. Otros vendedores suministran implementaciones comerciales del contenedor del portlet. Algunos de los vendedores líderes son IBM, Oracle y BEA Systems. Estos vendedores suministran implementaciones basadas en estándares así como también extensiones no aprobadas todavía por el cuerpo de estándares. Más aún, un gran número de soluciones de portales open-source soportan JSR168 tales como Apache Jetspeed-2 Enterprise Portal, JBoss Portal, Liferay Portal, y Stringbeans Portal.

Enlaces Externos

- JSR-168, la API Java Portlet
- Apache Pluto, la Implementación de Referencia JSR-168
- Portal JBoss
- Portlets Comunidad de Yahoogroups
- Comunidad de Portlets Open source en java.net
- Portlet FAQ
- ONJava.com
- Documento PDF JSR-168

Java Server Pages

JavaServer Pages (**JSP**) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo.

Esta tecnología es un desarrollo de la compañía Sun Microsystems. La Especificación JSP 1.2 fue la primera que se liberó y en la actualidad está disponible la Especificación JSP 2.1.

Las JSP's permiten la utilización de código Java mediante scripts. Además es posible utilizar algunas acciones JSP predefinidas mediante etiquetas. Estas etiquetas pueden ser enriquecidas mediante la utilización de Librerías de Etiquetas (TagLibs o Tag Libraries) externas e incluso personalizadas.

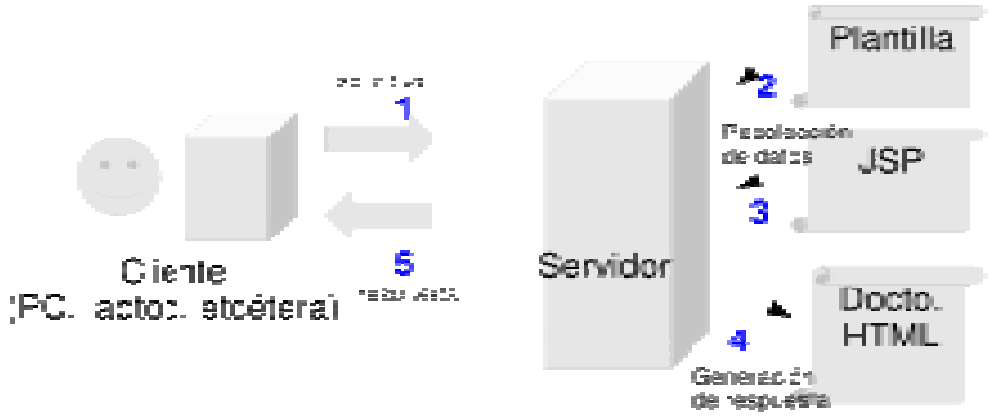
Microsoft, la más directa competencia de Sun, ha visto en esta estrategia de Sun una amenaza, lo que le ha llevado a que su plataforma .NET incluya su lenguaje de scripts ASP.NET que permite ser integrado con clases .NET (ya estén hechas en C++, VisualBasic o C#) del mismo modo que jsp se integra con clases Java.

Arquitectura

JSP puede considerarse como una manera alternativa, y simplificada, de construir servlets. Es por esto que una página puede hacer todo lo que un servlet puede hacer, y viceversa. Cada versión de la especificación de JSP está fuertemente vinculada a una versión en particular de la especificación de servlets.

El funcionamiento general de la tecnología JSP es que el Servidor de Aplicaciones interpreta el código contenido en la página JSP para construir el código Java del servlet a generar. Este servlet será el que genere el documento (típicamente HTML) que se presentará en la pantalla del Navegador del usuario.

JSP -> Servidor Aplicaciones (Servlets) -> Cliente (Navegador)



Es posible enriquecer el lenguaje de etiquetas utilizado por JSP. Para ello debemos extender la capa de alto nivel JSP mediante la implementación de Librerías de Etiquetas (Tags Libraries). Un ejemplo de estas librerías son las propocionadas por Sun bajo la denominación de JSTL o las distribuidas por Apache junto con el Framework de Struts.

TagLibs -> JSP -> Servidor Aplicaciones (Servlets) -> Cliente (Navegador)

El rendimiento de una página JSP es el mismo que tendría el servidor equivalente, ya que el código es compilado como cualquier otra clase Java. A su vez, la máquina virtual compilará dinámicamente a código de máquina las partes de la aplicación que lo requieran. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

La principal ventaja de **JSP** frente a otros lenguajes es que el lenguaje Java es un lenguaje de propósito general que excede el mundo web y que es apto para crear clases que manejen lógica de negocio y acceso a datos de una manera prolija. Esto permite separar en niveles las aplicaciones web, dejando la parte encargada de generar el documento HTML en el archivo JSP.

Otra ventaja es que JSP hereda la portabilidad de Java, y es posible ejecutar las aplicaciones en múltiples plataformas sin cambios. Es común incluso que los desarrolladores trabajen en una plataforma y que aplicación termine siendo ejecutada en otra.

Los servlets y Java Server Pages (JSPs) son dos métodos de creación de páginas web dinámicas en servidor usando el lenguaje Java. En ese sentido son similares a otros métodos o lenguajes tales como el PHP, los CGIs (common gateway interface), programas que generan páginas web en el servidor, o ASP (Active Server Pages), un método específico de Microsoft. Sin embargo, se diferencian de ellos en otras cosas.

Para empezar, los JSPs y servlets se ejecutan en una máquina virtual Java, lo cual permite que, en principio, se puedan usar en cualquier tipo de ordenador, siempre que exista una máquina virtual Java para él. Cada servlet (o JSP, a partir de ahora lo usaremos de forma indistinta) se ejecuta en su propia hebra, es decir, en su propio contexto; pero no se comienza a ejecutar cada vez que recibe una petición, sino que persiste de una petición a la siguiente, de forma que no se pierde tiempo en invocarlo (cargar programa + intérprete). Su persistencia le permite también hacer una serie de cosas de forma más eficiente: conexión a bases de datos y manejo de sesiones, por ejemplo.

Los JSPs son en realidad servlets: un JSP se compila a un programa en Java la primera vez que se invoca, y del programa en Java se crea una clase que se empieza a ejecutar en el servidor como un servlet. La principal diferencia entre los servlets y los JSPs es el enfoque de la programación: un JSP es una página Web con etiquetas especiales y código Java incrustado, mientras que un servlet es un programa que recibe peticiones y genera a partir de ellas una página web

Ejemplo de Documentos JSP

Ejemplo de código de una página JSP:

```
<%@ page errorPage="myerror.jsp" %>
<%@ page import="com.foo.bar" %>
<html>
<head>
<%! int serverInstanceVariable = 1;%>
...
<% int localStackBasedVariable = 1; %>
<table>
<tr><td></td></tr>
...
```

Ejemplo de una compilación o "Salida" JSP:

```
package jsp_servlet;
import java.util.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
import com.foo.bar; //importado como resultado de <%@ page import="com.foo.bar" %>
import ...
class _myservlet implements javax.servlet.Servlet, javax.servlet.jsp.HttpJspPage {
   //insertado como
   //resultado de <%! int serverInstanceVariable = 1;%>
   int serverInstanceVariable = 1;
   ...
   public void _jspService( javax.servlet.http.HttpServletRequest request,
                            javax.servlet.http.HttpServletResponse response )
     throws javax.servlet.ServletException,
            java.io.IOException
   {
      javax.servlet.ServletConfig config = ...;//obtener la configuración del servlet
      Object page = this;
      PageContext pageContext = ...;//obtener el contexto de la pagina para esta petición
      javax.servlet.jsp.JspWriter out = pageContext.getOut();
      HttpSession session = request.getSession( true );
      ...
```

Para ejecutar las páginas JSP, se necesita un servidor Web con un contenedor Web que cumpla con las especificaciones de JSP y de Servlet. Tomcat 5 es una completa implementación de referencia para las especificaciones Java Servlet 2.2 y JSP 1.1.

Sintaxis

Variables Implicitas

Las páginas JSP incluyen ciertas variables privilegiadas sin necesidad de declararlas ni configurarlas:

Variable	Clase
pageContext	javax.servlet.jsp.PageContext
request	javax.servlet.http.HttpServletRequest
response	javax.servlet.http.HttpServletResponse
session	javax.servlet.http.HttpSession
config	javax.servlet.ServletConfig
application	javax.servlet.ServletContext
out	javax.servlet.jsp.JspWriter
page	java.lang.Object
exception	java.lang.Exception

Directivas

Son instrucciones al JSP Runtime: no producen una salida visible al usuario sino que configura cómo se ejecutará la página JSP.
Su sintáxis sería:

```
<%@ directiva atributo="valor" %>
```

Las directivas disponibles son:

- **include**, incluye estáticamente el contenido de un fichero para generar la servlet mediante el atributo *file*.

```
<%@ include file="cabecera.html" %>
```

- **taglib**, importa librerías de etiquetas (Tag Libraries)

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>
```

- **page**, especifica parámetros relacionados con la página a procesar.

Atributo	Sintaxis	Utilización
import	<%@ page import="class; class" %>	Importa clases y paquetes Java para ser utilizadas dentro del fichero JSP.
session	<%@ page session="false" %>	Especifica si utiliza los datos contenidos en sesión; por defecto "true".
contentType	<%@ page contentType="class; class" %>	Especifica el tipo MIME del objeto "response"; por defecto "text/html; charset=ISO-8859-1".
buffer	<%@ page buffer="12KB" %>	Buffer utilizado por el objeto writer "out"; puede tomar el valor de "none"; por defecto "8KB".
errorPage	<%@ page errorPage="/path_to_error_page" %>	Especifíca la ruta de la página de error que será invocada en caso de producirse una excepción durante la ejecución de este fichero JSP.
isErrorPage	<%@ page isErrorPage="true" %>	Determina si este fichero JSP es una página que maneja excepciones. Únicamente a este tipo de páginas pueden acceder a la variable implícita "exception", que contiene la excepción que provocó la llamada a la página de error.

Scriptlets

Nos permite declarar variables, funciones y datos estáticos.

```
<%! int maxAlumnosClase = 30; %>
```

Las scriptles son partes de código Java incrustadas entre los elementos estáticos de la página.

```
<% ... código Java ... %>
```

Las expresiones se evalúan dentro de la servlet. No deben acabar en ";".

```
<%= maxAlumnosClase + 1%>
```

El siguiente ejemplo pondría como título de la página el atributo "titulo" contenido en el objeto request:

```
<%
  String titulo = "";
  if (request.getAttribute("titulo") != null) {
    titulo = (String) request.getAttribute ("titulo");
  }
%>
...
<title><%=titulo%></title>
....
```

Extiquetas

Etiquetas JSP para simplificar el código y dar mayor funcionalidad.
Desarrollar sitios web utilizando etiquetas presenta ciertas ventajas como:

- facilitar el aprendizaje.
- facilitar el mantenimiento.
- fomentar la modularidad y la reutilización.
- simplificar el código y reducir el número de lineas necesarias.

Su sintaxis sería:

```
<%@ taglib uri="/taglib/lycka" prefix="lycka" %>
...
<lycka:hola/>
...
```

A la hora de generar el código Java de la Servlet, esta etiqueta **hola** será interpretada por el Servidor de Aplicaciones como perteneciente a la librería de etiquetas (Tag Library) **lycka**. Esta librería estará identificada en el fichero descriptor de nuestra aplicación (web.xml) con el nombre de recurso (URI) **/taglib/lycka**.

```
<taglib-uri>/taglib/lycka</taglib-uri>
<taglib-location>/WEB-INF/tags/lycka.tld</taglib-location>
```

Una implementación de este fichero descriptor, /WEB-INF/tags/**lycka.tld** podría ser:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
  "http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">

<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>simp</shortname>
  <uri>http://www.hachisvertas.net/jcs/taglibs/lycka</uri>
  <info>A simple sample tag library</info>

  <tag>
    <name>hola</name>
    <tagclass>org.lcyka.taglibs.miEtiqueta</tagclass>
    <bodycontent>empty</bodycontent>
    <info>Alaba la belleza de mi gata.</info>
  </tag>
</taglib>
```

Y por fin, el servidor de aplicaciones sustituirá la etiqueta por su código Java asociado, **org.lcyka.taglibs.miEtiqueta**:

```
package org.lcyka.taglibs;
import ...;
public class miEtiqueta extends TagSupport {
  public int doStart {
    try {
      pageContext.getOut.print("Mi gata es preciosa");
    } catch (IOException ioe) {
    }
    return SKIP_BODY;
  }
```

Y finalmente el **navegador** mostraría:

```
Mi gata es preciosa
```

Etiquetas JSP

Son las etiquetas pertenecientes a la especificación JSP. Proporcionan una funcionalidad básica.

Un primer grupo de etiquetas proporciona funcionalidad a nivel de la página de una manera muy simple:

- **<jsp:forward>**, redirige la request a otra URL
- **<jsp:include>**, incluye el texto de un fichero dentro de la página
- **<jsp:plugin>**, descarga un plugin de Java (una applet o un Bean).

Un segundo grupo permite manipular componentes JavaBean sin conocimientos de Java.

- **<jsp:useBean>**, permite manipular un Bean (si no existe, se creará el Bean), especificando su ámbito (scope), la clase y el tipo.
- **<jsp:getProperty>**, obtiene la propiedad especificada de un bean previamente declarado y la escribe en el objeto response.
- **<jsp:setProperty>**, establece el valor de una propiedad de un bean previamente declarado.

Etiquetas JSTL

Son proporcinadas por Sun dentro de la distribución de JSTL.

- **core**, iteraciones, condicionales, manipulación de URL y otras funciones generales.
- **xml**, para la manipulación de XML y para XML-Transformation.
- **sql**, para gestionar conexiones a bases de datos.
- **i18n**, para la internacionalización y formateo de las cadenas de caracteres como cifras.

Etiquetas Struts TagLib

Distribuidas por Apache para funcionar junto con el Framework de Struts.

- **Bean**
- **HTML**
- **Logic**
- **Nested**

Etiquetas Personalizadas

Anteriormente hemos visto un ejemplo para crear una etiqueta personalizada almacenada en nuestra propia librería de etiquetas.
Para desarrollar etiquetas personalizadas, utilizaremos la API de las librerías de etiquetas (Tag Libraries).
La API de las Servlet de Java es:

```
javax.servlet.*
```

La API de JSP extiende de esta API,

```
javax.servlet.jsp.*
```

Finalmente, la API de las librerías de etiquetas (Tag Libraries) extiende de esta última,

```
javax.servlet.jsp.tagext.*
```

Lo más relevante de esta API son:

- Los interfaces
  - Tag, que todas las etiquetas deben implementar.
  - BodyTag, extiende a la anterior y define métodos adicionales para inspeccionar el cuerpo de una etiqueta.
- Las clases
  - BodyContent, un manejador (handler) para leer y escribir en el cuerpo de una etiqueta.
  - BodyTagSupport, que implementa el interfaz BodyTag.
  - TagAttributeInfo, para obtener la información de los atributos de la etiqueta declarados en el TLD.
  - TagData, que contiene los valores de los atributos.
  - TagExtraInfo, para especificar información extra de una etiqueta, como las variables que introduce en el código o los atributos que serán validados.
  - TagInfo, basado en la información de la TLD.
  - TagLibraryInfo, representa la información de una TLD.
  - TagSupport, implementa el interfaz Tag.
  - VariableInfo, contiene información como el tipo y ámbito de las variables creadas o modificadas por la etiqueta.

Podemos encontrar una descripción más detallada en http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/jsp/tagext/package-summary.html
Otro ejemplo de etiqueta podría ser el siguiente código Java:

```
package org.lycka.taglibs;
import ...;
public class LowerCaseTag extends BodyTagSupport {
  public int doAfterBody() throws JspException {
    try {
      BodyContent body = getBodyContent();
      JspWriter writer = body.getEnclosingWriter();
      String bodyString = body.getString();
      if ( bodyString != null ) {
        writer.print( bodyString.toLowerCase());
      }
    } catch(IOException ioe) {
      throw new JspException("Error: IOException while writing to the user");
    }
    return SKIP_BODY;
  }
}
```

Al encontrar el inicio de la etiqueta, el runtime primero se invocará el método doStart() una vez instanciada la clase. Puede devolver uno de los siguientes valores:

- SKIP_BODY, no procesa el contenido del cuerpo de la etiqueta.
- EVAL_BODY_INCLUDE , evalúa el cuerpo de la etiqueta.
- EVAL_BODY_TAG , evalúa el cuerpo de la etiqueta y lanza el resultado a otro stream almacenado en una propiedad de la etiqueta.

El método doAfterBody() después de procesar el cuerpo de la etiqueta.
Finalmente se invocará el método doEndTag(). Puede devolver:

- EVAL_PAGE, para seguir procesando la página JSP
- SKIP_PAGE, para dejar de procesar la página JSP, para por ejemplo redirigir la página

Declarado en el descriptor de la librería como

```
<tag>
  <name>lowercase</name>
  <tagclass>org.lycka.taglibs.LowerCaseTag</tagclass>
  <bodycontent>JSP</bodycontent>
  <info>Put body in lowercase.</info>
</tag>
```

Utilizado en la página JSP

```
<%@ taglib uri="/taglib/lycka" prefix="lycka" %>
...
<lycka:lowercase>Mi Gata es TAN Preciosa.</lycka:lowercase>
```

Y su salida sería

```
mi gata es tan preciosa.
```

Enlaces Externos

- Sun Microsystem
- Página oficial de JSP
- Apache
- Tomcat
- JSP en Lyckapedia