WebSphere Application Server

Guía de programación de Edge Components

Versión 6.0.2

GC11-3032-02

WebSphere Application Server

Guía de programación de Edge Components

Versión 6.0.2

GC11-3032-02

Nota

Antes de utilizar esta información y el producto al que da soporte, asegúrese de leer la información general del apartado "Avisos" en la página 61.

Tercera edición (junio de 2005)

Esta publicación es la traducción del original inglés *WebSphere Application Server: Programming Guide for Edge Components* (GC31-6856-02).

Esta edición se aplica a:
WebSphere Application Server, Versión 6.0.2

y a todos los releases y modificaciones posteriores hasta que se indique lo contrario en nuevas ediciones.

Realice el pedido de las publicaciones a través del representante de IBM o de la sucursal de IBM que presta servicio en su localidad.

Contenido

Figuras

Acerca de este manual

En este apartado se describe la finalidad, la organización y los convenios de este documento, *WebSphere Application Server Guía de programación de Edge Components*.

A quién va dirigido este manual

Este manual describe las interfaces de programación de aplicaciones (API) que están disponibles para personalizar la familia Edge Components de WebSphere Application Server, Versión 6.0.2. Esta información está pensada para programadores que escriben aplicaciones de plug-ins y realizan otras personalizaciones. Puede que diseñadores de redes y administradores de sistemas también estén interesados en esta información como indicación de los tipos de personalización que son posibles.

¿Qué debe conocer ya?

La utilización de la información de este manual exige la comprensión de los procedimientos de programación con los lenguajes de programación Java o C, en función de la API que prevea utilizar. Los métodos y estructuras disponibles en cada interfaz expuesta se documentan, pero debe saber cómo construir una aplicación propia, compilarla para el sistema y probarla. Se proporciona código de ejemplo para algunas interfaces, pero sólo se proporcionan ejemplos para construir una aplicación propia.

Convenios y terminología que se utilizan en este manual

Esta documentación utiliza los siguientes convenios tipográficos y de teclas.

Tabla 1. Convenios que se utilizan en este manual

Convenio	Significado
Negrita	Cuando se hace referencia a interfaces gráficas de usuario (las GUI), se indican en negrita menús, elementos de los menús, etiquetas, botones, iconos y carpetas. También puede utilizarse para enfatizar nombres de mandatos que, de lo contrario, podrían confundirse con el texto de alrededor.
`Monoespaciado`	Indica texto que es necesario entrar en un indicador de mandatos. Además, el monoespaciado indica texto que aparece en la pantalla, ejemplos de código y extractos de archivos.
Cursiva	Indica valores de variable que debe proporcionar el usuario (por ejemplo, el usuario facilitará el nombre de un archivo para *NombreArchivo*). La cursiva también indica énfasis y los títulos de manuales.
Control-*x*	Donde *x* es el nombre de una tecla, indica una secuencia de Control-carácter. Por ejemplo, Control-c significa pulsar y mantener pulsada la tecla Control mientras se pulsa la tecla c.
Intro	Se refiere a la tecla etiquetada con la palabra Intro o con la flecha hacia la izquierda.
`%`	Representa el indicador de shell de mandatos de Linux y UNIX para un mandato que no requiere privilegios de root.
`#`	Representa el indicador de shell de mandatos de Linux y UNIX de un mandato que requiere privilegios de root.
`C:\`	Representa el indicador de mandatos de Windows.

Tabla 1. Convenios que se utilizan en este manual (continuación)

Convenio	Significado
Entrada de mandatos	Cuando se le indique que "entre" o "emita" un mandato, escriba el mandato y luego pulse Intro. Por ejemplo, la instrucción "Entre el mandato **ls**" significa que debe escribir **ls** en un indicador de mandatos y, después, pulsar Intro.
[ ]	Encierran elementos opcionales en las descripciones de sintaxis.
{ }	Encierran listas de las que debe elegirse un elemento en las descripciones de sintaxis.
|	Separa elementos en una lista de opciones encerradas entre los signos { } (llaves) en las descripciones de sintaxis.
...	Los puntos suspensivos que aparecen en las descripciones de sintaxis indican que es posible repetir el elemento anterior una o más veces. Los puntos suspensivos que aparecen en los ejemplos indican que se ha omitido información en el ejemplo para una mayor brevedad.

Accesibilidad

Las características de accesibilidad ayudan al usuario que tiene discapacidades físicas, como por ejemplo una movilidad restringida o una visión limitada, a utilizar satisfactoriamente los productos de software. Éstas son las principales características de accesibilidad en WebSphere Application Server, Versión 6.0.2:

- Puede utilizar el software lector de pantalla y un sintetizador de discurso digital para oír lo que se visualiza en la pantalla. También puede utilizar el software de reconocimiento de voz, como por ejemplo IBM ViaVoice, para entrar datos y para navegar por la interfaz de usuario.
- Puede utilizar las características utilizando el teclado en lugar del ratón.
- Puede configurar y administrar las características de Application Server utilizando editores de texto estándares o interfaces de línea de mandatos en lugar de las interfaces gráficas proporcionadas. Para obtener más información sobre la accesibilidad de características específicas, consulte la documentación sobre dichas características.

Documentos relacionados y sitios Web

- *Conceptos, planificación e instalación de Edge Components*, GC10-9973-02
- *Caching Proxy Administration Guide*, GC11-3031-02
- *Load Balancer Administration Guide*, GC11-3030-02
- Sitio Web de inicio de IBM *www.ibm.com/*
- IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/*
- Sitio Web de bibliotecas de IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/library.html*
- Sitio Web de soporte de IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/support.html*
- Centro de información de IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/infocenter.html*
- Centro de información de IBM WebSphere Application Server Edge Components *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

Cómo enviar comentarios

Sus comentarios son importantes para ayudarnos a proporcionar información de la más alta precisión y calidad. Si tiene algún comentario sobre este manual u otra documentación de la familia Edge Components de WebSphere Application Server,

- Envíe sus comentarios por correo electrónico a hojacom@es.ibm.com. Asegúrese de incluir el nombre del manual, el número de pieza del manual, la versión de WebSphere Application Server y, si procede, la ubicación específica del texto sobre el que realiza el comentario (por ejemplo, un número de página o un número de tabla).

Capítulo 1. Visión general de la personalización de Edge Components

Este manual trata las interfaces de programación de aplicaciones (API) que se proporcionan para la familia Edge Components de WebSphere Application Server. La familia Edge Components de WebSphere Application Server incluye Caching Proxy y Load Balancer. Se proporcionan varias interfaces que permiten a los administradores personalizar sus instalaciones, cambiar el modo en que Edge Components interactúan entre sí o habilitar la interacción con otros sistemas de software.

IMPORTANTE: Caching Proxy está disponible en todas las instalaciones de Edge Components, con las excepciones siguientes:

- Caching Proxy no está disponible para instalaciones de Edge Components que se ejecuten en procesadores Itanium 2 o AMD Opteron de 64 bits.
- Caching Proxy no está disponible para instalaciones de Edge Components de Load Balancer para IPv6.

Las API de este documento abarcan varias categorías.

Personalización de Caching Proxy

Caching Proxy tiene varias interfaces grabadas en su secuencia de proceso donde se pueden añadir o sustituir procesos personalizados para procesos estándar. Las personalizaciones que se pueden ejecutar son la modificación o el aumento de tareas como las siguientes:

- autenticación de clientes,
- autorización de peticiones,
- conversión de URL en vías de acceso a archivos físicos,
- servicio de peticiones,
- registro cronológico y
- respuesta a condiciones de error.

Se llama a los programas de aplicación personalizados, que también se denominan plug-ins de Caching Proxy, en momentos predeterminados de la secuencia de proceso del servidor proxy.

La API de Caching Proxy se ha utilizado para implementar ciertas funciones del sistema. Por ejemplo el soporte de LDAP del servidor proxy se implementa como plug-in.

En el Capítulo 2, “API de Caching Proxy”, en la página 3 se describe la interfaz detalladamente y se detallan los pasos para configurar el servidor proxy para utilizar programas de plug-ins.

Personalización de Load Balancer

Load Balancer se puede personalizar escribiendo sus propios asesores. Los asesores realizan la medición de carga real en los servidores. Con un asesor personalizado, puede utilizar un método propio y que sea relevante para el sistema a fin de medir la carga. Esto es especialmente importante si dispone de sistemas de servidor Web personalizados o de propiedad.

En el Capítulo 3, “Asesores personalizados”, en la página 41 se proporciona información detallada sobre cómo escribir y utilizar asesores personalizados. Incluye código de asesores de ejemplo.

Localización del código de ejemplo

El código de ejemplo de estas API se incluye en el CD-ROM de Edge Components en el directorio samples. Hay disponibles ejemplos de código adicionales en el sitio Web de WebSphere Application Server, www.ibm.com/software/webservers/appserv/.

Capítulo 2. API de Caching Proxy

En este apartado se describe la interfaz de programación de aplicaciones (API) de Caching Proxy: qué es, por qué es útil y cómo funciona.

IMPORTANTE: Caching Proxy está disponible en todas las instalaciones de Edge Components, con las excepciones siguientes:

- Caching Proxy no está disponible para instalaciones de Edge Components que se ejecuten en procesadores Itanium 2 o AMD Opteron de 64 bits.
- Caching Proxy no está disponible para instalaciones de Edge Components de Load Balancer para IPv6.

Visión general de la API de Caching Proxy

La API es una interfaz con Caching Proxy que permite ampliar las funciones básicas del servidor proxy. Puede escribir extensiones, o plug-ins, para realizar procesos personalizados, incluidos los ejemplos siguientes:

- Mejorar la rutina de autenticación básica o sustituirla por un proceso específico del sitio.
- Añadir rutinas de manejo de errores para realizar el seguimiento de problemas o alertas sobre condiciones graves.
- Detectar y realizar un seguimiento de la información procedente del cliente que realiza la petición, como referencias de servidor y códigos de agente de usuario.

La API de Caching Proxy ofrece las ventajas siguientes:

- **Eficacia**
  - La API está diseñada de forma específica para el sistema de proceso con hebras que utiliza Caching Proxy.
- **Flexibilidad**
  - La API contiene funciones enriquecidas y versátiles.
  - La API es independiente de la plataforma y compatible con distintos lenguajes. Se ejecuta en todas las plataformas de Caching Proxy y se pueden escribir aplicaciones de plug-ins en la mayor parte de los lenguajes de programación soportados por estas plataformas.
- **Facilidad de uso**
  - Los tipos de datos simples se pasan por referencia y no por valor (por ejemplo, long *, char *).
  - Cada función tiene un número fijo de parámetros.
  - Se incluyen enlaces en lenguaje C.
  - Los plug-ins no tienen ningún impacto en la memoria asignada; las aplicaciones de plug-ins asignan y liberan memoria independientemente de otros procesos de Caching Proxy.

Procedimiento general para escribir programas de la API

Antes de escribir sus programas de plug-ins de Caching Proxy, tiene que comprender cómo funciona el servidor proxy. El comportamiento del servidor proxy se puede dividir en varios pasos de proceso distintos. Para cada uno de estos pasos, puede proporcionar funciones personalizadas propias mediante la API.

Por ejemplo, si desea realizar alguna acción después de que se lea una petición de cliente, pero antes de llevar a cabo otro proceso. O quizá desea llevar a cabo rutinas especiales durante la autenticación y de nuevo después de que se envíe el archivo solicitado.

Con la API se proporciona una biblioteca de funciones predefinidas. Los programas de plug-ins pueden llamar a funciones predefinidas de la API para que interactúen con el proceso del servidor proxy (por ejemplo, para manipular peticiones, leer o grabar cabeceras de peticiones o grabar en las anotaciones cronológicas del servidor proxy). Estas funciones no deben confundirse con las funciones de plug-in que escriba, a las que llama el servidor proxy. Las funciones predefinidas se describen en el apartado "Funciones y macros predefinidas" en la página 17.

Dará instrucciones al servidor proxy para que llame a las funciones de plug-in en los pasos adecuados mediante las directivas de la API de Caching Proxy correspondientes en el archivo de configuración del servidor. Estas directivas se describen en el apartado "Directivas de configuración de Caching Proxy para pasos de la API" en la página 24.

Este documento incluye lo siguiente:

- Una explicación básica de los pasos de Caching Proxy que se pueden personalizar (consulte el apartado "Pasos de proceso del servidor").
- Directrices para escribir plug-ins (consulte el apartado "Directrices" en la página 8).
- Prototipos de las funciones personalizadas que se pueden escribir para cada paso que realiza el servidor y sus códigos de retorno (consulte el apartado "Prototipos de función de plug-in" en la página 10).
- Definiciones de las funciones y las macros predefinidas a las que se puede llamar desde los plug-ins y sus códigos de retorno (consulte el apartado "Funciones y macros predefinidas" en la página 17).
- Directivas de configuración de la API de Caching Proxy (consulte el apartado "Directivas de configuración de Caching Proxy para pasos de la API" en la página 24).

Puede utilizar estos componentes y procedimientos para escribir sus propios programas de plug-ins de Caching Proxy.

Pasos de proceso del servidor

El funcionamiento básico del servidor proxy se puede dividir en pasos según el tipo de proceso que el servidor lleva a cabo durante cada fase. Cada paso incluye una coyuntura en la que una parte específica del programa se puede ejecutar. Al añadir directivas de la API al archivo de configuración de Caching Proxy (ibmproxy.conf), se indican las funciones de plug-in a las que se desea llamar durante un paso concreto. Puede llamar a varias funciones de plug-in durante un paso de proceso concreto si incluye más de una directiva para ese paso.

Algunos pasos forman parte del proceso de peticiones del servidor. En otras palabras, el servidor proxy ejecuta estos pasos cada vez que procesa una petición. Otros pasos se realizan de forma independiente del proceso de peticiones, es decir, el servidor ejecuta estos pasos independientemente de si se ha procesado o no una petición.

El programa compilado reside en un objeto compartido, por ejemplo un archivo DLL o .so, en función del sistema operativo. A medida que el servidor avanza por los pasos de proceso de peticiones, llama a las funciones de plug-in asociadas con cada paso hasta que una de las funciones indica que ha gestionado la petición. Si especifica más de una función de plug-in para un paso concreto, las funciones se llaman en el orden en el que aparecen sus directivas en el archivo de configuración.

Si una función de plug-in no gestiona la petición (no ha incluido una directiva de la API de Caching Proxy para ese paso o la función de plug-in de ese paso ha devuelto HTTP_NOACTION), el servidor lleva a cabo la acción por omisión para ese paso.

Nota: Esto es así para todos los pasos excepto el paso Service; el paso Service no tiene una acción por omisión.

En la Figura 1 en la página 6 se detallan los pasos del proceso del servidor proxy y se define el orden de proceso de los pasos relacionados con el proceso de peticiones.

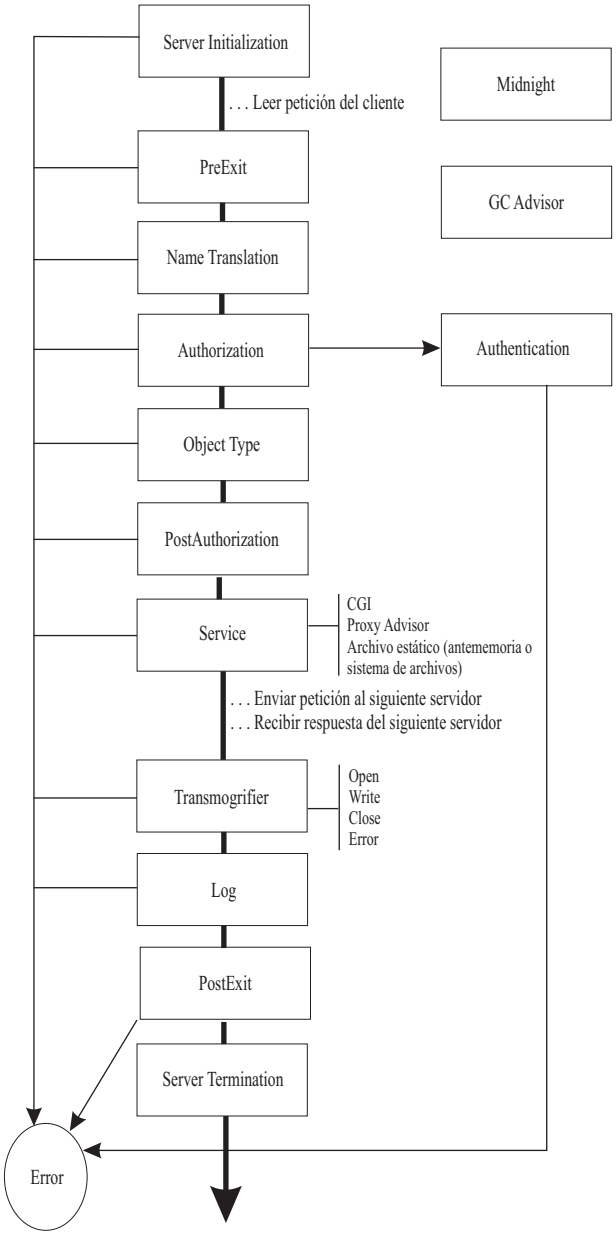

Figura 1. Diagrama de flujo de los pasos del proceso del servidor proxy

Cuatro de los pasos del diagrama se ejecutan independientemente del proceso de las peticiones del cliente. Estos pasos están relacionados con la ejecución y el mantenimiento del servidor proxy. Son los siguientes:

- Server Initialization
- Midnight
- GC Advisor
- Server Termination

En la lista siguiente se explica la finalidad de cada uno de los pasos detallados en la Figura 1. Tenga en cuenta que no se garantiza que se llame a todos los pasos para una petición concreta.

Server Initialization
Realiza una inicialización cuando se inicia el servidor proxy y antes de que se acepten las peticiones del cliente.

Midnight
Ejecuta un plug-in a medianoche, sin contexto de petición. Este paso se muestra de forma independiente en el diagrama porque no forma parte del proceso de petición; en otras palabras, su ejecución es independiente de cualquier petición.

GC Advisor
Influye en las decisiones de recogida de basura para los archivos de la antememoria. Este paso se muestra de forma independiente en el diagrama porque no forma parte del proceso de petición; en otras palabras, su ejecución es independiente de cualquier petición. La recogida de basura se realiza cuando el tamaño de antememoria alcanza el valor máximo. En la publicación *WebSphere Application Server Caching Proxy Administration Guide* encontrará información acerca de la configuración de la recogida de basura de la antememoria.

PreExit

Lleva a cabo un proceso después de leer una petición pero antes de realizar cualquier otra acción.

Si este paso devuelve una indicación de que la petición se ha procesado (HTTP_OK), el servidor omite el resto de pasos del proceso de peticiones y realiza sólo los pasos Transmogrifier, Log y PostExit.

Name Translation
Convierte la vía de acceso virtual (de un URL) en la vía de acceso física.

Authorization

Utiliza los símbolos de seguridad almacenados para comprobar la vía de acceso física para protecciones, ACL y otros controles de acceso y genera las cabeceras de autenticación WWW obligatorias para la autenticación básica. Si escribe una función de plug-in propia para sustituir este paso, debe generar estas cabeceras personalmente.

Consulte el apartado “Autenticación y autorización” en la página 36 para obtener más información.

Authentication

Decodifica, verifica y almacena símbolos de seguridad.

Consulte el apartado “Autenticación y autorización” en la página 36 para obtener más información.

Object Type
Localiza el objeto del sistema de archivos que indica la vía de acceso.

Post Authorization

Lleva a cabo el proceso después de la autorización y la localización del objeto, pero antes de que se dé respuesta a la petición.

Si este paso devuelve una indicación de que la petición se ha procesado (HTTP_OK), el servidor omite el resto de pasos del proceso de peticiones y realiza sólo los pasos Transmogrifier, Log y PostExit.

Service
Da respuesta a la petición (enviando el archivo, ejecutando el CGI, etc.).

Proxy Advisor
Influye en las decisiones de proxy y antememoria.

Transmogrifier
Proporciona acceso de escritura a la parte de datos de la respuesta enviada al cliente.

Log Habilita el registro cronológico de transacciones personalizado.

Error Permite respuestas personalizas a condiciones de error.

PostExit
Borra los recursos asignados para el proceso de peticiones.

Server Termination
Realiza un proceso de limpieza cuando tiene lugar una conclusión normal.

Directrices

- Escriba el programa, siguiendo la sintaxis y las directrices que se proporcionan para las funciones de plug-in del servidor. Asigne a cada una de las funciones de plug-in un nombre de función exclusivo y llame las funciones predefinidas del servidor, según sea necesario.

  En sistemas AIX, es necesario un archivo de exportación (por ejemplo, libmyapp.exp) que enumere las funciones de plug-in y debe establecer un enlace con el archivo de importación de la API de Caching Proxy, libhttpdapi.exp.

  En sistemas Linux, HP-UX y Solaris, debe establecer un enlace con las bibliotecas libhttpdapi y libc.

  En sistemas Windows, necesita un archivo de definición de módulo (.def) que enumere las funciones de plug-in y debe establecer un enlace con HTTPDAPI.LIB.

  Asegúrese de incluir HTAPI.h y de utilizar la macro HTTPD_LINKAGE en las definiciones de funciones. Esta macro garantiza que todas las funciones utilicen los mismos convenios de llamada.
- El servidor se ejecuta en un entorno de varias hebras y, por lo tanto, los plug-ins deben ser seguros para hebras. Si la aplicación es reentrante, el rendimiento no disminuye.
- Mantenga las acciones de los plug-in en un ámbito de hebra. No realice ninguna acción en un ámbito de proceso, por ejemplo, saliendo, cambiando el ID de usuario o registrando un manejador de señales.
- No utilice variables globales o, si debe utilizarlas, proteja las variables globales con un semáforo de exclusión mutua.
- Recuerde que debe establecer la cabecera Content-Type si va a utilizar la función HTTPD_write() para enviar datos de vuelta al cliente.
- Compruebe siempre los códigos de retorno y proporcione un proceso condicional donde sea necesario.
- Compile y enlace el programa, consultando la documentación del compilador para crear un objeto compartido (por ejemplo, un archivo DLL o .so) según sea necesario para el sistema operativo.

  Utilice los mandatos de compilación y enlace como directriz.

  – **AIX**, con IBM CSet++

    - **Compilar:**

      ```
      cc_r  -c -qdbxextra -qcpluscmt foo.c
      ```

- **Enlazar:**

```
cc_r -bM:SRE -bnoentry -o libfoo.so foo.o -bI:libhttpdapi.exp
     -bE:foo.exp
```

Este mandato se muestra en dos líneas para poderlo leer.

– **HP-UX**, con el paquete del desarrollador HP C/ANSI C y HP aC++ Compiler
  - **Compilar:**

```
cc -Ae -c +Z +DAportable
```

  - **Enlazar:**

```
aCC +Z -mt -c +DAportable
```

– **Linux**, con Gnu Compiler C (GCC) Versión 3.2.X
  - **Compilar:**

```
gcc -c foo.c
```

  - **Enlazar:**

```
ld -G -Bsymbolic -o libfoo.so foo.o -lhttpdapi -lc
```

– **Solaris**, con Sun Workshop
  - **Compilar:**

```
cc -mt -Bsymbolic -c foo.c
```

  - **Enlazar:**

```
cc -mt -Bsymbolic -G -o libfoo.so foo.o -lhttpdapi -lc
```

– **Windows**, con Microsoft Visual C++
  - **Compilar:**

```
cl /c /MD /DWIN32 foo.c
```

  - **Enlazar:**

```
link httpdapi.lib foo.obj /def:foo.def /out:foo.dll /dll
```

Para especificar exportaciones, utilice uno de estos métodos:

– Añada definiciones _declspec(dllexport) en el origen.
– Especifique /EXPORT:nombre_entrada en la línea de mandatos LIB.
– Cree un archivo de definición de módulo con una sentencia EXPORTS.

- Añada directivas de la API de Caching Proxy al archivo de configuración para asociar las funciones de plug-in del programa con los pasos adecuados. Existe una directiva independiente para cada paso en el proceso de peticiones del servidor. Detenga y reinicie el servidor para que las nuevas directivas entren en vigor.

  Nota: Caching Proxy no descarga los objetos compartidos (archivos DLL o .so) ni en el reinicio. Debe detener y, a continuación, iniciar el servidor para liberar los objetos compartidos.

- Pruebe el programa de forma rigurosa antes de utilizarlo en un entorno de producción. Puesto que Caching Proxy es un servidor con hebras, debe aplicar unas pruebas más rigurosas que las necesarias para un servidor de bifurcación. Los errores del programa pueden hacer que el servidor proxy falle, porque el servidor proxy llama al programa directamente y los dos se ejecutan en el mismo espacio de proceso.

Funciones de plug-in

Siga la sintaxis que se describe en el apartado “Prototipos de función de plug-in” en la página 10 para escribir sus propias funciones de programa para los pasos de proceso de peticiones definidos.

Cada una de las funciones debe rellenar el parámetro de código de retorno con un valor que indique la acción que se ha llevado a cabo:

- El código HTTP_NOACTION (valor 0) significa que no se ha llevado a cabo ninguna acción relevante. Si se devuelve este código, el servidor proxy adopta la acción por omisión para este paso.
- Uno de los códigos de retorno HTTP válidos indica que la función de plug-in ha gestionado el paso. Consulte el apartado "Códigos y valores de retorno de HTTP" en la página 16 para obtener una lista de códigos de retorno válidos. Si se ofrece un código de retorno HTTP, no se llama a ninguna otra función de plug-in para manejar ese paso de la petición.

Prototipos de función de plug-in

Los prototipos de función de cada paso de Caching Proxy muestran el formato que se debe utilizar y explica el tipo de proceso que pueden realizar. Tenga en cuenta que los nombres de función no están predefinidos. Debe asignar a las funciones nombres exclusivos y puede elegir convenios de nombres propios. Para que la asociación resulte más fácil, en este documento se utilizan nombres relacionados con los pasos de proceso del servidor.

En cada una de estas funciones de plug-in, son válidas algunas funciones predefinidas de la API. Otras funciones predefinidas no son válidas para todos los pasos. Las siguientes funciones predefinidas de la API son válidas cuando se llaman desde todas estas funciones de plug-in:

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- funciones HTTPD_log*

En las descripciones de prototipos de función se indican las funciones válidas o no válidas adicionales de la API.

El valor del parámetro *handle* enviado a las funciones se puede pasar como primer argumento a las funciones predefinidas. Las funciones predefinidas de la API se describen en el apartado "Funciones y macros predefinidas" en la página 17.

Server Initialization

```
void HTTPD_LINKAGE ServerInitFunction (
     unsigned char *handle,
     unsigned long *major_version,
     unsigned long *minor_version,
     long *return_code
     )
```

La función definida para este paso se llama una sola vez cuando se carga el modelo durante la inicialización del servidor. Es la oportunidad de realizar una inicialización antes de que se acepten peticiones.

Aunque se llame a todas las funciones de inicialización del servidor, un código de retorno de error de una función en este paso hace que el servidor ignore el resto de funciones configuradas en el mismo módulo como la función que ha devuelto el código de error, es decir, no se llama al resto de funciones contenidas en el mismo objeto compartido que la función que ha devuelto el error.

Los parámetros de versión contienen el número de versión del servidor proxy; Caching Proxy los proporciona.

PreExit

```
void  HTTPD_LINKAGE  PreExitFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición después de que se haya leído la petición pero antes de que tenga lugar un proceso. Se puede utilizar un plug-in en este paso para acceder a la petición del cliente antes de que Caching Proxy lo procese.

Los códigos de retorno válidos para la función preExit son los siguientes:

- 0 (HTTP_NOACTION)
- 200 (HTTP_OK)
- Errores HTTP en las series 4*xx* o 5*xx* (por ejemplo, 404, HTTP_NOT_FOUND)

No se deben utilizar otros códigos de retorno.

Si esta función devuelve HTTP_OK, el servidor proxy presupone que la petición se ha gestionado. Los pasos posteriores de proceso de peticiones se pasan por alto y sólo se llevan a cabo los pasos de respuesta (Transmogrifier, Log y PostExit).

Todas las funcionas predefinidas de la API son válidas durante este paso.

Midnight

```
void  HTTPD_LINKAGE  MidnightFunction (
         unsigned char *handle,
         long *return_code
         )
```

Una función definida para este paso se ejecuta diariamente a medianoche y no contiene ningún contexto de petición. Por ejemplo, se puede utilizar para invocar un proceso hijo para analizar anotaciones cronológicas. Tenga en cuenta que un proceso exhaustivo durante este paso puede interferir con el registro cronológico.

Authentication

```
void  HTTPD_LINKAGE  AuthenticationFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición según el esquema de autenticación de la petición. Esta función se puede utilizar para personalizar la verificación de los símbolos de seguridad que se envían con una petición.

Name Translation

```
void  HTTPD_LINKAGE  NameTransFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las peticiones que coincidan con la plantilla. El paso de conversión de nombres tiene lugar antes de que se procese la petición y proporciona un mecanismo para correlacionar URL con objetos, como nombres de archivo.

Authorization

```
void  HTTPD_LINKAGE  AuthorizationFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las peticiones que coincidan con la plantilla. El paso de autorización tiene lugar antes de que se procese la petición y se puede utilizar para verificar que el objeto identificado se pueda devolver al cliente. Si va a realizar una autenticación básica, debe generar las cabeceras WWW-Authenticate obligatorias.

Object Type

```
void  HTTPD_LINKAGE  ObjTypeFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las peticiones que coincidan con la plantilla. El paso de tipo de objeto tiene lugar antes de que se procese la petición y se puede utilizar para comprobar si el objeto existe y para realizar una configuración de tipo de objeto.

PostAuthorization

```
void  HTTPD_LINKAGE  PostAuthFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama después de que se haya autorizado la petición pero antes de que tenga lugar un proceso. Si esta función devuelve HTTP_OK, el servidor proxy presupone que la petición se ha gestionado. Los pasos posteriores de petición se pasan por alto y sólo se llevan a cabo los pasos de respuesta (Transmogrifier, Log y PostExit).

Todas las funciones predefinidas de servidor son válidas durante este paso.

Service

```
void  HTTPD_LINKAGE  ServiceFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama para cada petición. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las

peticiones que coincidan con la plantilla. El paso Service da respuesta a la petición, si no se ha dado respuesta en los pasos PreExit o PostAuthorization.

Todas las funciones predefinidas de servidor son válidas durante este paso.

Consulte la directiva Enable en la publicación *WebSphere Application Server Caching Proxy Administration Guide* para obtener información sobre la configuración de la función Service que se debe ejecutar según el método HTTP y no según el URL.

Transmogrifier

Las funciones a las que se llama en este paso de proceso se pueden utilizar para filtrar datos de respuesta como secuencia de datos. Se llama a cuatro funciones de plug-in para este paso de forma secuencial y cada una actúa como segmento de conducto a través del que fluyen los datos. Es decir, se llama a las funciones *open*, *write*, *close* y *error* que proporcione, en ese orden, para cada respuesta. Cada función procesa la misma secuencia de datos por su parte.

Para este paso, debe implementar las cuatro funciones siguientes. Los nombres de función no tienen que coincidir con estos nombres.

- **Open**

```
void *  HTTPD_LINKAGE  openFunction (
         unsigned char *handle,
         long *return_code
         )
```

  La función open realiza cualquier inicialización (como una asignación de almacenamiento intermedio) obligatoria para procesar los datos de esta secuencia de datos. Cualquier código de retorno que no sea HTTP_OK hace que este filtro termine anormalmente (no se llama a las funciones write y close). La función puede devolver un puntero void para que pueda asignar espacio para una estructura y que se le devuelva el puntero en el parámetro *correlator* de las funciones posteriores.

- **Write**

```
void  HTTPD_LINKAGE  writeFunction (
         unsigned char *handle,
         unsigned char *data,    /* datos de respuesta enviados por el
                                    servidor de origen */
         unsigned long *length,  /* longitud de los datos de respuesta */
         void *correlator,       /* puntero devuelto por la
                                    función 'open' */
         long *return_code
         )
```

  La función write procesa los datos y puede llamar a la función HTTPD_write() predefinida del servidor con los datos nuevos o cambiados. El plug-in no debe intentar liberar el almacenamiento intermedio que se le ha pasado ni esperar que el servidor libere el almacenamiento intermedio que reciba.

  Si decide no cambiar los datos en el ámbito de la función write, aún puede llamar a la función HTTPD_write() en el ámbito de la función open, write o close a fin de pasar los datos de la respuesta al cliente. El argumento *correlator* es el puntero que apunta al almacenamiento intermedio de datos que se ha devuelto en la rutina *open*.

- **Close**

  ```
  void  HTTPD_LINKAGE  closeFunction (
           unsigned char *handle,
           void *correlator,
           long *return_code
           )
  ```

  La función close realiza las acciones de limpieza (como desechar y liberar el almacenamiento intermedio de correlator) obligatorias para realizar el proceso de los datos para esta secuencia. El argumento *correlator* es el puntero que apunta al almacenamiento intermedio de datos que se ha devuelto en la rutina *open*.

- **Error**

  ```
  void  HTTPD_LINKAGE  errorFunction (
           unsigned char *handle,
           void *correlator,
           long *return_code
           )
  ```

  La función error habilita la realización de acciones de limpieza, como desechar o liberar los datos en almacenamiento intermedio (o ambas acciones) antes de que se envíe una página de error. En este punto, se llama a las funciones open, write y close para procesar la página de error. El argumento *correlator* es el puntero que apunta al almacenamiento intermedio de datos que se ha devuelto en la rutina *open*.

Notas:

- Cuando escriba un plug-in para el paso Transmogrifier, debe llamar a HTTPD_open(), HTTPD_write() y HTTPD_close() en algún momento en el ámbito de las funciones open, write y close. Sólo se puede llamar a HTTPD_write() después de que se haya llamado a la función HTTPD_open(). La finalidad de estas funciones predefinidas es dar el control al servidor de manera que se pueda invocar la siguiente función de la secuencia.
- Es necesario llamar a las funciones HTTPD_* para el paso de la API Transmogrifier y para que el servidor funcione correctamente. Por ejemplo, si no se llama a HTTPD_open() y a HTTPD_close(), no se devuelven cabeceras al cliente.
- Tenga en cuenta que se pueden producir efectos no deseados si las aplicaciones de filtrado de datos no son lo suficientemente selectivas en su filtrado de secuencias de datos. Es posible que las CGI no funcionen si se filtran de forma incorrecta, que los archivos GIF no se muestren y que otras secuencias binarias no funcionen como se esperaba.
- No es necesario que el plug-in guarde en almacenamiento intermedio el cuerpo del contenido. Caching Proxy determina de forma automática la longitud del contenido.
- Se recomienda llamar a HTTPD_open() cuando se está preparado para dar el control de las cabeceras al servidor. No obstante, si tiene que establecer una cabecera más adelante en el programa de la API, puede esperar a que la función write o close llame a la función HTTPD_open().

  Nota: Debe establecer las cabeceras mediante HTTPD_set() o httpd_setvar() antes de llamar a la función HTTPD_open().
- La secuencia de datos no incluye cabeceras. Los plug-in deben utilizar las funciones set y extract para manipular cabeceras. La función *open* del plug-in no se invoca hasta que se han leído todas las cabeceras.

- Puede utilizar varios plug-ins de Transmogrifier, que se invocan en el orden en el que aparecen en el archivo de configuración.
- La gestión de túneles SSL no se pasa a través de los plug-ins de Transmogrifier.

GC Advisor

```
void  HTTPD_LINKAGE  GCAdvisorFunction (
        unsigned char *handle,
        long *return_code
        )
```

La función definida para este paso se llama para cada archivo de la antememoria durante la recogida de basura. Esta función permite asesorar sobre los archivos que se conservan y los que se descartan. Para obtener más información, consulte las variables GC_*.

Proxy Advisor

```
void  HTTPD_LINKAGE  ProxyAdvisorFunction (
        unsigned char *handle,
        long *return_code
        )
```

Se invoca una función definida durante el servicio de cada petición de proxy. Por ejemplo, se puede utilizar para establecer la variable USE_PROXY.

Log

```
void  HTTPD_LINKAGE  LogFunction (
        unsigned char *handle,
        long *return_code
        )
```

La función definida para este paso se llama para cada petición después de que se haya procesado la petición y se haya cerrado la comunicación con el cliente. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las peticiones que coincidan con la plantilla. Se llama a esta función independientemente de la ejecución satisfactoria o no del proceso de peticiones. Si no desea que el plug-in de anotaciones cronológicas altere temporalmente el mecanismo de anotaciones cronológicas por omisión, establezca el código de retorno en HTTP_NOACTION, en lugar de HTTP_OK.

Error

```
void  HTTPD_LINKAGE  ErrorFunction (
        unsigned char *handle,
        long *return_code
        )
```

La función definida para este paso se llama para cada petición que falla. Se puede especificar una plantilla de URL en la directiva del archivo de configuración si desea que se llame a la función de plug-in sólo para las peticiones que hayan fallado y que coincidan con la plantilla. El paso Error le ofrece la oportunidad de personalizar la respuesta al error.

PostExit

```
void  HTTPD_LINKAGE  PostExitFunction (
        unsigned char *handle,
        long *return_code
        )
```

La función definida para este paso se llama para cada petición, independientemente de la ejecución satisfactoria o no de la petición. Este paso permite realizar tareas de limpieza para los recursos asignados por el plug-in a fin de procesar la petición.

Server Termination

```
void  HTTPD_LINKAGE  ServerTermFunction (
         unsigned char *handle,
         long *return_code
         )
```

La función definida para este paso se llama cuando tiene lugar una conclusión normal del servidor. Permite limpiar los recursos asignados durante el paso Server Initialization. No llame a funciones HTTP_* en este paso (los resultados son imprevisibles). Si tiene más de una directiva de la API de Caching Proxy en el archivo de configuración para Server Termination, se les realizará una llamada.

Nota: Debido a una limitación actual en el código de Solaris, el paso del plug-in Server Termination no se ejecuta cuando se utiliza el mandato **ibmproxy -stop** para concluir Caching Proxy en plataformas Solaris. Consulte la publicación *WebSphere Application Server Caching Proxy Administration Guide* para obtener información acerca del inicio y la detención de Caching Proxy.

Códigos y valores de retorno de HTTP

Estos códigos de retorno siguen la especificación HTTP 1.1, RFC 2616, publicada por World Wide Web Consortium (www.w3.org/pub/WWW/Protocols/). Las funciones de plug-in deben devolver uno de estos valores.

Tabla 2. Códigos de retorno HTTP para funciones de la API de Caching Proxy

Valor	Código de retorno
0	HTTP_NOACTION
100	HTTP_CONTINUE
101	HTTP_SWITCHING_PROTOCOLS
200	HTTP_OK
201	HTTP_CREATED
202	HTTP_ACCEPTED
203	HTTP_NON_AUTHORITATIVE
204	HTTP_NO_CONTENT
205	HTTP_RESET_CONTENT
206	HTTP_PARTIAL_CONTENT
300	HTTP_MULTIPLE_CHOICES
301	HTTP_MOVED_PERMANENTLY
302	HTTP_MOVED_TEMPORARILY
302	HTTP_FOUND
303	HTTP_SEE_OTHER
304	HTTP_NOT_MODIFIED
305	HTTP_USE_PROXY
307	HTTP_TEMPORARY_REDIRECT

Tabla 2. Códigos de retorno HTTP para funciones de la API de Caching Proxy (continuación)

400	HTTP_BAD_REQUEST
401	HTTP_UNAUTHORIZED
403	HTTP_FORBIDDEN
404	HTTP_NOT_FOUND
405	HTTP_METHOD_NOT_ALLOWED
406	HTTP_NOT_ACCEPTABLE
407	HTTP_PROXY_UNAUTHORIZED
408	HTTP_REQUEST_TIMEOUT
409	HTTP_CONFLICT
410	HTTP_GONE
411	HTTP_LENGTH_REQUIRED
412	HTTP_PRECONDITION_FAILED
413	HTTP_ENTITY_TOO_LARGE
414	HTTP_URI_TOO_LONG
415	HTTP_BAD_MEDIA_TYPE
416	HTTP_BAD_RANGE
417	HTTP_EXPECTATION_FAILED
500	HTTP_SERVER_ERROR
501	HTTP_NOT_IMPLEMENTED
502	HTTP_BAD_GATEWAY
503	HTTP_SERVICE_UNAVAILABLE
504	HTTP_GATEWAY_TIMEOUT
505	HTTP_BAD_VERSION

Funciones y macros predefinidas

Puede llamar a las funciones y las macros predefinidas del servidor desde funciones de plug-in propias. Debe utilizar sus nombres predefinidos y respetar el formato que se describe a continuación. En las descripciones de parámetros, la letra `e` indica un parámetro de entrada, la letra `s` indica un parámetro de salida y `e/s` indica que se utiliza un parámetro tanto para la entrada como para la salida.

Cada una de estas funciones devuelve uno de los códigos de retorno HTTPD, en función de si la petición se ha llevado a cabo de forma satisfactoria. Estos códigos se describen en el apartado "Códigos de retorno de funciones y macros predefinidas" en la página 23.

Utilice el descriptor de contexto que se proporciona para el plug-in como primer parámetro cuando llame a estas funciones. De lo contrario, la función devuelve un código de error HTTPD_PARAMETER_ERROR. NULL no se acepta como descriptor de contexto válido.

HTTPD_authenticate()

Autentica un ID de usuario o una contraseña, o ambos. Sólo es válida en los pasos PreExit, Authentication, Authorization y PostAuthorization.

```
void  HTTPD_LINKAGE  HTTPD_authenticate (
         unsigned char *handle,      /* e; descriptor de contexto */
         long *return_code           /* s; código de retorno */
         )
```

HTTPD_cacheable_url()

Indica si el contenido del URL especificado se puede almacenar en antememoria de acuerdo con los estándares de Caching Proxy.

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
        unsigned char *handle,      /* e; descriptor de contexto */
        unsigned char *url,         /* e; URL que se debe comprobar */
        unsigned char *req_method,  /* e; método de petición del URL */
        long *retval                /* s; código de retorno */
        )
```

El valor de retorno HTTPD_SUCCESS indica que el contenido del URL se puede almacenar en antememoria; HTTPD_FAILURE indica que el contenido no se puede almacenar en antememoria. HTTPD_INTERNAL_ERROR también es un código de retorno posible para esta función.

HTTPD_close()

Sólo es válida para el paso Transmogrifier. Transfiere el control a la siguiente rutina *close* de la pila de secuencias. Llame a esta función desde las funciones open, write o close de Transmogrifier después de que se realicen los procesos que desee. Esta función notifica al servidor proxy que la respuesta se ha procesado y que el paso Transmogrifier se ha realizado.

```
void  HTTPD_LINKAGE  HTTPD_close (
         unsigned char *handle,       /* e; descriptor de contexto */
         long *return_code            /* s; código de retorno */
         )
```

HTTPD_exec()

Ejecuta un script para dar respuesta a esta petición. Es válida en los pasos PreExit, Service, PostAuthorization y Error.

```
void  HTTPD_LINKAGE  HTTPD_exec (
         unsigned char *handle,         /* e; descriptor de contexto */
         unsigned char *name,           /* e; nombre del script que debe
                                           ejecutarse */
         unsigned long *name_length,    /* e; longitud del nombre */
         long *return_code              /* s; código de retorno */
         )
```

HTTPD_extract()

Extrae el valor de una variable asociada con esta petición. Las variables válidas del parámetro *name* son las mismas que las que se utilizan en la CGI. Consulte el apartado “Variables” en la página 27 para obtener más información. Tenga en cuenta que esta función es válida en todos los pasos; no obstante, no todas las variables son válidas en todos los pasos.

```
void  HTTPD_LINKAGE  HTTPD_extract (
      unsigned char *handle,       /* e; descriptor de contexto */
      unsigned char *name,         /* e; nombre de la variable que debe
                                      extraerse */
      unsigned long *name_length,  /* e; longitud del nombre */
      unsigned char *value,        /* s; almacenamiento intermedio en el
                                      que se debe colocar el valor */
      unsigned long *value_length, /* e/s; tamaño de almacenamiento
                                      intermedio */
      long *return_code            /* s; código de retorno */
      )
```

Si esta función devuelve el código HTTPD_BUFFER_TOO_SMALL, el tamaño del almacenamiento intermedio que ha solicitado no era lo suficientemente grande para el valor extraído. En tal caso, la función no utiliza el almacenamiento intermedio, pero actualiza el parámetro value_length con el tamaño del almacenamiento intermedio que necesite a fin de extraer satisfactoriamente este valor. Vuelva a intentar la extracción con un almacenamiento intermedio que tenga, como mínimo, el mismo tamaño que el valor de longitud_valor devuelto.

Nota: Si la variable que se va a extraer es para una cabecera HTTP, la función HTTPD_extract() extraerá sólo la primera aparición coincidente, aunque la petición contenga varias cabeceras con el mismo nombre. La función httpd_getvar() se puede utilizar en lugar de HTTPD_extract() y también tiene otras ventajas. Consulte la página 19 para obtener más información.

HTTPD_file()

Envía un archivo para dar respuesta a esta petición. Sólo es válida en los pasos PreExit, Service, Error, PostAuthorization y Transmogrifier.

```
void  HTTPD_LINKAGE  HTTPD_file (
         unsigned char *handle,          /* e; descriptor de contexto */
         unsigned char *name,            /* e; nombre del archivo que
                                            debe enviarse */
         unsigned long *name_length,     /* e; longitud del nombre */
         long *return_code               /* s; código de retorno */
         )
```

httpd_getvar()

Es la misma que HTTPD_extract(), excepto que es más fácil de utilizar porque el usuario no tiene que especificar longitudes para los argumentos.

```
const  unsigned char *          /* s; valor de la variable */
HTTPD_LINKAGE
httpd_getvar(
   unsigned char *handle,       /* e; descriptor de contexto */
   unsigned char *name,         /* e; nombre de la variable */
   unsigned long *n             /* e; número de índice de la matriz
                                   que contiene la cabecera */
   )
```

El índice de la matriz que contiene la cabecera empieza por 0. Para obtener el primer elemento de la matriz, utilice el valor 0 para *n*; para obtener el quinto elemento, utilice el valor 4 para *n*.

Nota: No descarte ni cambie el contenido del valor devuelto. La serie devuelta tiene una terminación con un valor null.

HTTPD_log_access()

Graba una serie en las anotaciones cronológicas de acceso del servidor.

```
void  HTTPD_LINKAGE  HTTPD_log_access (
         unsigned char *handle,        /* e; descriptor de contexto */
         unsigned char *value,         /* e; datos que se deben grabar */
         unsigned long *value_length,  /* e; longitud de los datos */
         long *return_code             /* s; código de retorno */
         )
```

Tenga en cuenta que los símbolos de escape *no* son obligatorios cuando se graba el símbolo de porcentaje (%) en las anotaciones cronológicas de acceso del servidor.

HTTPD_log_error()

Graba una serie en las anotaciones cronológicas de errores del servidor.

```
void  HTTPD_LINKAGE  HTTPD_log_error (
         unsigned char *handle,       /* e; descriptor de contexto */
         unsigned char *value,        /* e; datos que se deben grabar */
         unsigned long *value_length, /* e; longitud de los datos */
         long *return_code            /* s; código de retorno */
         )
```

Tenga en cuenta que los símbolos de escape *no* son obligatorios cuando se graba el símbolo de porcentaje (%) en las anotaciones cronológicas de errores del servidor.

HTTPD_log_event()

Graba una serie en las anotaciones cronológicas de sucesos del servidor.

```
void  HTTPD_LINKAGE  HTTPD_log_event (
         unsigned char *handle,       /* e; descriptor de contexto */
         unsigned char *value,        /* e; datos que se deben grabar */
         unsigned long *value_length, /* e; longitud de los datos */
         long *return_code            /* s; código de retorno */
         )
```

Tenga en cuenta que los símbolos de escape *no* son obligatorios cuando se graba el símbolo de porcentaje (%) en las anotaciones cronológicas de sucesos del servidor.

HTTPD_log_trace()

Graba una serie en las anotaciones cronológicas de rastreo del servidor.

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
         unsigned char *handle,       /* e; descriptor de contexto */
         unsigned char *value,        /* e; datos que se deben grabar */
         unsigned long *value_length, /* e; longitud de los datos */
         long *return_code            /* s; código de retorno */
         )
```

Tenga en cuenta que los símbolos de escape *no* son obligatorios cuando se graba el símbolo de porcentaje (%) en las anotaciones cronológicas de rastreo del servidor.

HTTPD_open()

Sólo es válida para el paso Transmogrifier. Transfiere el control a la siguiente rutina de la pila de secuencias. Llame a esta función desde las funciones open, write o close de Transmogrifier después de establecer las cabeceras que desee y si está preparado para iniciar la rutina write.

```
void  HTTPD_LINKAGE  HTTPD_open (
         unsigned char *handle,      /* e; descriptor de contexto */
         long *return_code           /* s; código de retorno */
         )
```

HTTPD_proxy()

Realiza una petición de proxy. Es válida en los pasos PreExit, Service y PostAuthorization.

Nota: Es una función de finalización; la petición finaliza después de esta función.

```
void  HTTPD_LINKAGE  HTTPD_proxy (
         unsigned char *handle,      /* e; descriptor de contexto */
         unsigned char *url_name,    /* e; URL de la
                                        petición de proxy */
         unsigned long *name_length, /* e; longitud del URL */
```

```
        void *request_body,         /* e; cuerpo de la petición */
        unsigned long *body_length, /* e; longitud del cuerpo */
        long *return_code           /* s; código de retorno */
        )
```

HTTPD_read()

Lee el cuerpo de la petición del cliente. Utilice HTTPD_extract() para las cabeceras. Sólo es válida en los pasos PreExit, Authorization, PostAuthorization y Service y sólo es útil si se ha realizado una petición PUT o POST. Llame a esta función en un bucle hasta que se devuelva HTTPD_EOF. Si no hay ningún cuerpo para esta petición, esta función falla.

```
void  HTTPD_LINKAGE  HTTPD_read (
        unsigned char *handle,       /* e; descriptor de contexto */
        unsigned char *value,        /* e; almacenamiento intermedio
                                        para datos */
        unsigned long *value_length, /* e/s; tamaño de almacenamiento
                                        intermedio (longitud de datos) */
        long *return_code            /* s; código de retorno */
        )
```

HTTPD_restart()

Reinicia el servidor después de que se hayan procesado todas las peticiones activas. Es válida en todos los pasos excepto Server Initialization, Server Termination y Transmogrifier.

```
void  HTTPD_LINKAGE  HTTPD_restart (
        long *return_code    /* s; código de retorno */
        )
```

HTTPD_set()

Establece el valor de una variable asociada con esta petición. Las variables válidas del parámetro *name* son las mismas que las que se utilizan en la CGI. Consulte el apartado “Variables” en la página 27 para obtener más información.

Tenga en cuenta que también puede crear variables con esta función. Las variables que cree están sujetas a los convenios de los prefijos HTTP_ y PROXY_, que se describen en el apartado “Variables” en la página 27. Si crea una variable que empieza por HTTP_, se envía como cabecera en la respuesta al cliente, sin el prefijo HTTP_. Por ejemplo, para establecer una cabecera Location, utilice HTTPD_set() con el nombre de variable HTTP_LOCATION. Las variables creadas con un prefijo PROXY_ se envían como cabeceras en la petición al servidor de contenido. Las variables creadas con un prefijo CGI_ se pasan a los programas de CGI.

Esta función es válida en todos los pasos; no obstante, no todas las variables son válidas en todos los pasos.

```
void  HTTPD_LINKAGE  HTTPD_set (
        unsigned char *handle,       /* e; descriptor de contexto */
        unsigned char *name,         /* e; nombre de la variable que se
                                        debe establecer */
        unsigned long *name_length,  /* e; longitud del nombre */
        unsigned char *value,        /* e; almacenamiento intermedio con
                                        valor */
        unsigned long *value_length, /* e; longitud del valor */
        long *return_code            /* s; código de retorno */
        )
```

Nota: Puede utilizar la función httpd_setvar() para establecer un valor de variable sin tener que especificar un almacenamiento intermedio y una longitud. Consulte la página 22 para obtener información.

httpd_setvar()

Es la misma que HTTPD_set(), excepto que es más fácil de utilizar porque el usuario no tiene que especificar longitudes para los argumentos.

```
long            /* s; código de retorno */
HTTPD_LINKAGE   httpd_setvar (
       unsigned char *handle,       /* e; descriptor de contexto */
       unsigned char *name,         /* e; nombre de la variable */
       unsigned char *value,        /* e; nuevo valor */
       unsigned long *addHdr        /* e; añadir cabecera o sustituirla */
       )
```

El parámetro *addHdr* tiene cuatro valores posibles:

- HTTPD_SETVAR_REPLACE: sustituye todas las apariciones de la variable de cabecera por el valor nuevo.
- HTTPD_SETVAR_REPLACE_ADD: si existe una variable de cabecera, sustituye la primera aparición por el nuevo valor; si la variable no existe, añade el nuevo valor a las cabeceras.
- HTTPD_SETVAR_ADD: añade este valor a las cabeceras.
- HTTPD_SETVAR_REMOVE_ALL: suprime todas las apariciones de esta variable de cabecera.

Estos valores se definen en HTAPI.h.

httpd_variant_insert()

Inserta un variant en la antememoria.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,    /* e; descriptor de contexto */
         unsigned char *URI,       /* e; URI de este objeto */
         unsigned char *dimension, /* e; dimensión de variant */
         unsigned char *variant,   /* e; valor de variant */
         unsigned char *filename,  /* e; archivo que contiene el objeto */
         long *return_code         /* s; código de retorno */
         )
```

Notas:

1. El argumento dimensión hace referencia a la cabecera que establece la variación de este objeto a partir del URI. Por ejemplo, en el ejemplo anterior, un valor de dimensión posible es User-Agent.
2. El argumento variant hace referencia al valor de la cabecera que se indica en el argumento dimension. Varía a partir del URI. Por ejemplo, en el ejemplo anterior, un valor posible para el argumento variant es el siguiente:

   ```
   Mozilla 4.0 (compatible; BatBrowser 94.1.2; Bat OS)
   ```

3. El argumento filename debe apuntar a una copia con terminación en valor nulo del nombre de archivo en el que el usuario ha guardado el contenido modificado. El usuario es responsable de eliminar el archivo; esta acción es segura después de volver de esta función. El archivo sólo contiene el cuerpo sin cabeceras.
4. Cuando se almacenan variants en antememoria, el servidor actualiza la cabecera content-length y añade un aviso: cabecera 214. Los códigos de entidad Strong se eliminan.

httpd_variant_lookup()

Determina si existe un variant determinado en la antememoria.

```
void  HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,            /* e; descriptor de contexto */
    unsigned char *URI,               /* URI de este objeto */
```

```
    unsigned char *dimension,          /* e; dimensión de variant */
    unsigned char *variant,            /* e; valor de variant */
             long *return_code);       /* s; código de retorno */
```

HTTPD_write()

Graba el cuerpo de la respuesta. Es válida en los pasos PreExit, Service, Error y Transmogrifier.

Si no establece el tipo de contenido antes de llamar a esta función por primera vez, el servidor presupone que va a enviar una secuencia de datos de CGI.

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,             /* e; descriptor de contexto */
    unsigned char *value,              /* e; datos que se deben enviar */
    unsigned char *value_length,       /* e; longitud de los datos */
             long *return_code);       /* s; código de retorno */
```

Nota: Para establecer cabeceras de respuesta, consulte la página 21.

Nota: Después de que se devuelva una función HTTPD_*, lo seguro es liberar la memoria que se ha pasado con dicha función.

Códigos de retorno de funciones y macros predefinidas

El servidor establecerá el parámetro de código de retorno en uno de estos valores, en función de si la ejecución de la petición es satisfactoria:

Tabla 3. Códigos de retorno

Valor	Código de estado	Explicación
-1	HTTPD_UNSUPPORTED	La función no recibe soporte.
0	HTTPD_SUCCESS	La función se ha realizado satisfactoriamente y los campos de salida son válidos.
1	HTTPD_FAILURE	La función ha fallado.
2	HTTPD_INTERNAL_ERROR	Se ha encontrado un error interno y el proceso de esta petición no puede continuar.
3	HTTPD_PARAMETER_ERROR	Se han pasado uno o más parámetros no válidos.
4	HTTPD_STATE_CHECK	La función no es válida en este paso de proceso.
5	HTTPD_READ_ONLY	Sólo lo devuelve HTTPD_set y httpd_setvar. La variable es de sólo lectura y no la puede establecer el plug-in.
6	HTTPD_BUFFER_TOO_SMALL	Sólo lo devuelve HTTPD_set, httpd_setvar y HTTPD_read. El almacenamiento intermedio era demasiado pequeño.
7	HTTPD_AUTHENTICATE_FAILED	Sólo lo devuelve HTTPD_authenticate. La autenticación ha fallado. Examine las variables HTTP_RESPONSE y HTTP_REASON para obtener más información.
8	HTTPD_EOF	Sólo lo devuelve HTTPD_read. Indica el final del cuerpo de la petición.

Tabla 3. Códigos de retorno (continuación)

Valor	Código de estado	Explicación
9	HTTPD_ABORT_REQUEST	La petición ha terminado anormalmente porque el cliente ha proporcionado un código de entidad que no coincidía con la condición especificada por la petición.
10	HTTPD_REQUEST_SERVICED	Lo devuelve HTTPD_proxy. La función a la que se ha llamado ha dado respuesta a esta petición.
11	HTTPD_RESPONSE_ALREADY_ COMPLETED	La función ha fallado porque ya se ha dado respuesta a esa petición.
12	HTTPD_WRITE_ONLY	La variable es de sólo escritura y no la puede leer el plug-in.

Directivas de configuración de Caching Proxy para pasos de la API

Cada paso de este proceso de petición dispone de una directiva de configuración que se utiliza para indicar las funciones de plug-in que desee llamar y ejecutar durante ese paso. Puede añadir estas directivas al archivo de configuración del servidor (ibmproxy.conf) editándolo y actualizándolo manualmente o utilizando el formulario de proceso de petición de API de los formularios de configuración y administración de Caching Proxy.

Notas sobre el uso de la API

- Salvo las directivas Service y NameTrans, las directivas API de cada paso no tienen que aparecer en ningún orden concreto en el archivo de configuración. Tenga en cuenta que el orden de varias entradas de una directiva de la API es importante, como se describe más adelante en esta lista.
- No es necesario incluir una entrada para cada paso de la API. Si no tiene un plug-in para un paso concreto, omita la directiva correspondiente y se utilizará el proceso estándar para ese paso.
- Las directiva Service y NameTrans funcionan como el resto de directivas de correlación (por ejemplo, la directiva Pass) y dependen de su aparición y ubicación con respecto a otras directivas de correlación del archivo de configuración. Por ejemplo, una regla para /cgi-bin/foo.so debe aparecer antes de la regla para /cgi-bin/*.

  Esto significa que el servidor procesa las directivas Service, NameTrans, Exec, Fail, Map, Pass, Proxy, ProxyWAS y Redirect en su secuencia dentro del archivo de configuración. Cuando el servidor correlaciona de forma satisfactoria un URL con un archivo, no lee ni procesa ninguna de estas directivas. (La directiva Map es una excepción. Consulte la publicación *WebSphere Application Server Caching Proxy Administration Guide* para obtener información exhaustiva acerca de las reglas de correlación del servidor proxy).
- Puede tener más de una directiva de configuración para un paso. Por ejemplo, puede incluir dos directivas NameTrans, cada una apuntando a una función de plug-in distinta. Cuando el servidor lleva a cabo el paso de conversión de nombres, procesa las funciones de conversión de nombres en el orden en el que aparecen en el archivo de configuración.

  Nota: Si una función de plug-in proporcionada con Caching Proxy utiliza la misma directiva de la API que un plug-in que haya escrito, coloque la directiva del plug-in después de la directiva del plug-in del sistema.

- Algunas funciones de plug-in no se tienen que ejecutar para cada petición:
  - Varias directivas incluyen una máscara de URL. La especificación de una máscara de URL con estas directivas hace que se llame a la aplicación de plug-in sólo para las peticiones cuyos URL coincidan con ese patrón. Consulte el apartado "Directivas y sintaxis de la API" para obtener información acerca de los pasos que pueden utilizar máscaras de URL y el apartado "Variables de directivas de la API" en la página 26 para obtener información acerca de cómo utilizar esta función.
  - Especifique un esquema de autenticación con la directiva Authentication para indicar que desea que se llame al plug-in sólo para ciertos tipos de autenticación. Actualmente, el protocolo HTTP sólo da soporte a la autenticación básica. Consulte el apartado "Variables de directivas de la API" en la página 26 para obtener más información.
- Si el servidor no puede cargar una función de plug-in específica o si tiene una directiva ServerInit que no devuelve un código de retorno OK, no se llama a ninguna otra función de ese plug-in de Caching Proxy compilado. El proceso específico de ese plug-in que se haya realizado hasta ese momento se ignorará. Los demás plug-ins de Caching Proxy que incluya en estas directivas, y sus funciones, no se ven afectados por ello.

Directivas y sintaxis de la API

Estas directivas de archivo de configuración deben aparecer en el archivo ibmproxy.conf en una línea, sin espacios que no sean los especificados explícitamente aquí. Aunque aparecen saltos de línea por cuestiones de legibilidad en algunos de los ejemplos de sintaxis, no debe haber espacios en esos puntos en la directiva real.

Tabla 4. Directivas de la API del plug-in de Caching Proxy

ServerInit		*/vía_acceso/archivo:nombre_función serie_inicialización*
PreExit		*/vía_acceso/archivo:nombre_función*
Authentication	*tipo*	*/vía_acceso/archivo:nombre_función*
NameTrans	*/URL*	*/vía_acceso/archivo:nombre_función*
Authorization	*/URL*	*/vía_acceso/archivo:nombre_función*
ObjectType	*/URL*	*/vía_acceso/archivo:nombre_función*
PostAuth		*/vía_acceso/archivo:nombre_función*
Service	*/URL*	*/vía_acceso/archivo:nombre_función*
Midnight		*/vía_acceso/archivo:nombre_función*
Transmogrifier		*/vía_acceso/archivo:nombre_función_open: nombre_función_write: nombre_función_close:función_error*
Log	*/URL*	*/vía_acceso/archivo:nombre_función*
Error	*/URL*	*/vía_acceso/archivo:nombre_función*
PostExit		*/vía_acceso/archivo:nombre_función*
ServerTerm		*/vía_acceso/archivo:nombre_función*
ProxyAdvisor		*/vía_acceso/archivo:nombre_función*
GCAdvisor		*/vía_acceso/archivo:nombre_función*

Variables de directivas de la API

Las variables de estas directivas tienen los siguientes significados:

tipo Se utiliza sólo con la directiva Authentication para especificar si se va a llamar o no a la función de plug-in. Los valores válidos son los siguientes:

- Basic: se llama a la función de plug-in sólo para peticiones de autenticación básicas.
- *: se llama a la función de plug-in para todas las peticiones. Actualmente, el protocolo HTTP sólo da soporte a la autenticación básica. Para peticiones de autenticación no básica, puede devolver un código de retorno que indique que este tipo de autenticación no recibe soporte.

URL Especifica las peticiones para las que se llama a la función de plug-in. Las peticiones con URL que coincidan con esta plantilla harán que se utilice la función de plug-in. Las especificaciones de URL de estas directivas son virtuales (no incluyen el protocolo), pero van precedidas por una barra inclinada (/). Por ejemplo, /www.ics.raleigh.ibm.com es correcto, pero http://www.ics.raleigh.ibm.com no lo es. Puede especificar este valor como URL específico o como plantilla.

- URL específico: se llama a la función de plug-in sólo para ese URL exacto.
- Plantilla de URL: se llama a la función de plug-in para todos los URL que coinciden con la plantilla. Las plantillas pueden incluir el carácter comodín * y se pueden especificar con los formatos */URL** o bien /* o bien *.

Nota: Se *exige* una plantilla de URL con la directiva Service si desea que tenga lugar la conversión de vías de acceso.

vía de acceso/archivo
Nombre de archivo plenamente cualificado del programa compilado.

nombre_función
Nombre asignado a la función de plug-in dentro del programa.

La directiva Service exige un asterisco (*) después del nombre de función si desea tener acceso a la información de vía de acceso.

serie_inicialización
Esta parte opcional de la directiva ServerInit puede contener cualquier texto que desee pasar a la función de plug-in. Utilice httpd_getvar() para extraer el texto de la variable INIT_STRING.

Para obtener más información, incluida la sintaxis, para estas directivas, consulte la publicación *WebSphere Application Server Caching Proxy Administration Guide*.

Compatibilidad con otras API

La API de Caching Proxy es compatible con versiones anteriores de ICAPI y GWAPI, desde la versión 4.6.1.

Establecimiento de puertos de programas de CGI

Siga estas directrices para establecer puertos para aplicaciones CGI escritas en C a fin de utilizar la API de Caching Proxy:

- Elimine el punto de entrada main() o renómbrelo para poder crear una DLL.
- Elimine las variables globales o protéjalas con un semáforo de exclusión mutua.

- Cambie las llamadas siguientes en los programas:
  - Cambie las llamadas a la cabecera printf() por HTTPD_set() o httpd_setvar().
  - Cambie las llamadas a datos printf() por HTTPD_write().
  - Cambie las llamadas a getenv() por HTTPD_extract() o httpd_getvar(). Tenga en cuenta que esto devuelve memoria no asignada, por lo que debe liberar el resultado.
- Recuerde que el servidor se ejecuta en un entorno de varias hebras y que las funciones de plug-in deben ser seguras para hebras. Si las funciones son reentrantes, el rendimiento no disminuye.
- Recuerde que debe establecer la cabecera Content-Type si va a utilizar HTTPD_write() para enviar datos de vuelta al cliente.
- Compruebe con detenimiento si el código tiene fugas de memoria.
- Tenga en cuenta las vías de acceso de error que tenga establecidas. Si genera mensajes de error personalmente y los vuelve a enviar como HTML, debe devolver HTTPD_OK desde la función o las funciones de servicio.

Información de referencia sobre la API de Caching Proxy

Variables

Al escribir programas de la API, puede utilizar las variables de Caching Proxy que proporcionan información acerca del cliente remoto y el sistema servidor.

Notas:

- Los nombres de variables definidas por el usuario no pueden tener un prefijo SERVER_. La función de la API de Caching Proxy reserva las variables que empiezan por SERVER_ para el servidor y, por lo tanto, estas variables son sólo de lectura. Además, los prefijos HTTP_ y PROXY_ también se reservan para cabeceras HTTP.
- Todos las cabeceras de petición enviadas por el cliente (como Set-Cookie) tienen el prefijo HTTP_ y sus valores se pueden extraer. Para acceder a las variables que son cabeceras de petición, asigne el prefijo HTTP_ al nombre de la variable. También puede crear variables nuevas con la función predefinida httpd_setvar(). Para obtener información detallada sobre estas cabeceras, consulte el apartado “Códigos de retorno de funciones y macros predefinidas” en la página 23.
- Se utilizan dos prefijos de variables, HTTP_ y PROXY_, para indicar si una variable es aplicable a cabeceras para la petición o para la respuesta. El prefijo HTTP_ hace referencia a variables que fluyen entre el cliente y Caching Proxy. El prefijo PROXY_ hace referencia a variables que fluyen entre Caching Proxy y el servidor de origen (o el servidor siguiente de una cadena de proxy). Estas variables sólo son válidas durante los pasos de proceso de peticiones.
  - La extracción de una variable HTTP_* genera el valor de una cabecera que estaba en la petición del cliente al servidor proxy.
  - Al establecer una variable HTTP_* se establece la cabecera de respuesta que se envía del servidor proxy al cliente.
  - La extracción de una variable PROXY_* genera el valor de una cabecera que ha devuelto el servidor de contenido al servidor proxy.
  - Al establecer una variable PROXY_* se establece la cabecera de petición que se envía del servidor proxy al servidor de contenido (o al servidor siguiente de una cadena de proxy).

  En la Figura 2 en la página 28 se ofrece una demostración del uso de estos prefijos mientras Caching Proxy gestiona una petición de cliente.

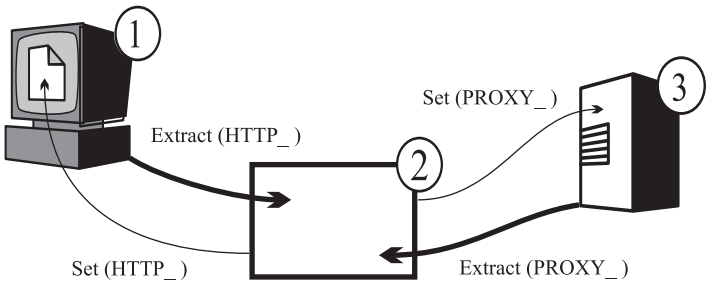

*Figura 2. Prefijos de variable HTTP_ y PROXY_*. Leyenda: 1—Máquina cliente 2—Caching Proxy 3—Servidor de origen

- Algunas variables son de sólo lectura. Las variables de sólo lectura representan los valores que se pueden extraer de una petición o una respuesta y utilizar en la función httpd_getvar() predefinida. Se genera un código de retorno HTTPD_READ_ONLY si intenta cambiar variables de sólo lectura utilizando la función httpd_setvar().
- Las variables no definidas como de sólo lectura se pueden leer y establecer en las funciones predefinidas httpd_getvar() o httpd_setvar(). Estas variables representan valores que se pueden extraer de una petición o una respuesta o valores que se pueden establecer al procesar una petición o una respuesta.

Definiciones de variables

Nota: Las variables de cabecera que no empiezan por los prefijos HTTP_ o PROXY_ son ambiguas. Para evitar la ambigüedad, utilice siempre el prefijo HTTP_ o PROXY_ con nombres de variable para cabeceras.

ACCEPT_RANGES
Contiene el valor de la cabecera de respuesta Accept-Ranges, que especifica si el servidor de contenido puede responder a peticiones de rangos. Utilice PROXY_ACCEPT_RANGES para extraer el valor de cabecera que envía el servidor de contenido al proxy. Utilice HTTP_ACCEPT_RANGES para establecer el valor de cabecera que se envía del proxy al cliente.

Nota: ACCEPT_RANGES es ambigua. Para eliminar la ambigüedad, utilice, en su lugar, HTTP_ACCEPT_RANGES y PROXY_ACCEPT_RANGES.

ALL_VARIABLES
Es de sólo lectura. Contiene todas las variables de CGI. Por ejemplo:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING
Es de sólo lectura. Si el servidor da soporte a la autenticación de clientes, esta serie contiene las credenciales no decodificadas que se deben utilizar para autenticar el cliente.

AUTH_TYPE
Es de sólo lectura. Si el servidor da soporte a la autenticación de clientes y el script está protegido, esta serie contiene el método utilizado para autenticar el cliente. Por ejemplo, `Basic`.

CACHE_HIT
Es de sólo lectura. Identifica si la petición del proxy se ha encontrado o no en la antememoria. Los valores que se devuelven son los siguientes:

- 0: la petición no se ha encontrado en la antememoria.
- 1: la petición no se ha encontrado en la antememoria.

CACHE_MISS
Es de sólo escritura. Se utiliza para forzar un fallo de antememoria. Los valores válidos son los siguientes:

- 0: no fuerza un fallo de antememoria.
- 1: fuerza un fallo de antememoria.

CACHE_TASK
Es de sólo lectura. Identifica si se ha utilizado la antememoria. Los valores que se devuelven son los siguientes:

- 0: la petición no ha accedido a la antememoria ni la ha actualizado.
- 1: la petición se ha servido desde la antememoria.
- 2: el objeto solicitado estaba en la antememoria pero se tenía que volver a validar.
- 3: el objeto solicitado no estaba en la antememoria y probablemente se ha añadido.

Esta variable se puede utilizar en los pasos PostAuthorization, PostExit, ProxyAdvisor o Log.

CACHE_UPDATE
Es de sólo lectura. Identifica si la petición del proxy ha actualizado la antememoria. Los valores que se devuelven son los siguientes:

- 0: la antememoria no se ha actualizado.
- 1: la antememoria se ha actualizado.

CLIENT_ADDR o CLIENTADDR
Es igual que REMOTE_ADDR.

CLIENTMETHOD
Es igual que REQUEST_METHOD.

CLIENT_NAME o CLIENTNAME
Es igual que REMOTE_HOST.

CLIENT_PROTOCOL o CLIENTPROTOCOL
Contiene el nombre y la versión del protocolo que el cliente va a utilizar para realizar la petición. Por ejemplo, `HTTP/1.1`.

CLIENT_RESPONSE_HEADERS
Es de sólo lectura. Devuelve un almacenamiento intermedio que contiene las cabeceras que el servidor envía al cliente.

CONNECTIONS
Es de sólo lectura. Contiene el número de conexiones que se van a servir o el número de peticiones activas. Por ejemplo, `15`.

CONTENT_CHARSET
El juego de caracteres de la respuesta de text/*, por ejemplo, `US ASCII`. La extracción de esta variable es aplicable a la cabecera content-charset desde el cliente. Su establecimiento afecta a la cabecera content-charset en la petición al servidor de contenido.

CONTENT_ENCODING
Especifica la codificación que se utiliza en el documento, por ejemplo, `x-gzip`. La extracción de esta variable es aplicable a la cabecera content-encoding desde el cliente. Su establecimiento afecta a la cabecera content-charset en la petición al servidor de contenido.

CONTENT_LENGTH
La extracción de esta variable es aplicable a la cabecera content-charset desde la petición del cliente. Su establecimiento afecta al valor de la cabecera en la petición al servidor de contenido.

Nota: CONTENT_LENGTH es ambigua. Para eliminar la ambigüedad, utilice, en su lugar, HTTP_CONTENT_LENGTH y PROXY_CONTENT_LENGTH.

CONTENT_TYPE
La extracción de esta variable es aplicable a la cabecera content-charset desde la petición del cliente. Su establecimiento afecta al valor de la cabecera en la petición al servidor de contenido.

Nota: CONTENT_TYPE es ambigua. Para eliminar la ambigüedad, utilice, en su lugar, HTTP_CONTENT_TYPE y PROXY_CONTENT_TYPE.

CONTENT_TYPE_PARAMETERS
Contiene otros atributos MIME, pero no el juego de caracteres. La extracción de esta variable es aplicable a la cabecera content-charset desde la petición del cliente. Su establecimiento afecta al valor de la cabecera en la petición al servidor de contenido.

DOCUMENT_URL
Contiene el URL (Uniform Request Locator). Por ejemplo:

```
http://www.anynet.com/~userk/main.htm
```

DOCUMENT_URI
Es igual que DOCUMENT_URL.

DOCUMENT_ROOT
Es de sólo lectura. Contiene la vía de acceso raíz al documento, como definen las reglas de paso.

ERRORINFO
Especifica el código de error para determinar la página de error. Por ejemplo, `blocked`.

EXPIRES
Define cuándo caducan los documentos almacenados en la antememoria de un proxy. La extracción de esta variable es aplicable a la cabecera content-charset desde la petición del cliente. Su establecimiento afecta al valor de la cabecera en la petición al servidor de contenido. Por ejemplo:

```
Mon, 01 Mar 2002 19:41:17 GMT
```

GATEWAY_INTERFACE
Es de sólo lectura. Contiene la versión de la API que el servidor va a utilizar. Por ejemplo, `ICSAPI/2.0`.

GC_BIAS
Es de sólo grabación. Este valor de coma flotante influye en la decisión de recogida de basura sobre el archivo que se va a considerar para la recogida de basura. El valor entrado se multiplica por el valor de calidad del Caching Proxy para que ese tipo de archivo determine la clasificación. Los

valores de calidad van de 0.0 a 0.1 y los definen las directivas AddType en el archivo de configuración de proxy (ibmproxy.conf).

GC_EVALUATION
Es de sólo grabación. Este valor de coma flotante determina si se debe eliminar (0.0) o conservar (1.0) el archivo que se va a considerar para la recogida de basura. Los valores entre 0.0 y 1.0 se ordenan por rangos, es decir, un archivo con el valor de GC_EVALUATION 0.1 es más probable que se elimine que un archivo con el valor de GC_EVALUATION 0.9.

GC_EXPIRES
Es de sólo lectura. Identifica el número de segundos que quedan hasta que el archivo que se está considerando caduque en la antememoria. Sólo puede extraer esta variable un plug-in de asesor de GC.

GC_FILENAME
Es de sólo lectura. Identifica el archivo que se va a considerar para la recogida de basura. Sólo puede extraer esta variable un plug-in de asesor de GC.

GC_FILESIZE
Es de sólo lectura. Identifica el tamaño del archivo que se va a considerar para la recogida de basura. Sólo puede extraer esta variable un plug-in de asesor de GC.

GC_LAST_ACCESS
Es de sólo lectura. Identifica cuando se ha accedido al archivo por última vez. Sólo puede extraer esta variable un plug-in de asesor de GC.

GC_LAST_CHECKED
Es de sólo lectura. Identifica cuándo se han comprobado los archivos por última vez. Sólo puede extraer esta variable un plug-in de asesor de GC.

GC_LOAD_DELAY
Es de sólo lectura. Identifica cuánto se ha tardado en recuperar el archivo. Sólo puede extraer esta variable un plug-in de asesor de GC.

HTTP_COOKIE
Cuando se lee, esta variable contiene el valor de la cabecera Set-Cookie establecida por el cliente. También se puede utilizar para establecer una nueva cookie en la secuencia de respuesta (entre el proxy y el cliente). Al establecer esta variable, se crea una nueva cabecera Set-Cookie en la secuencia de peticiones del documento, independientemente de si existe una cabecera duplicada.

HTTP_HEADERS
Es de sólo lectura. Se utiliza para extraer todas las cabeceras de petición del cliente.

HTTP_REASON
El establecimiento de esta variable afecta a la serie de motivo de la respuesta HTTP. Su establecimiento también afecta a la serie de motivo de la respuesta del proxy al cliente. La extracción de esta variable devuelve la serie de motivo en la respuesta del servidor de contenido al proxy.

HTTP_RESPONSE
El establecimiento de esta variable afecta al código de respuesta en la respuesta HTTP. Su establecimiento también afecta al código de estado de la respuesta del proxy al cliente. La extracción de esta variable devuelve el código de estado en la respuesta del servidor de contenido al proxy.

HTTP_STATUS
Contiene el código de respuesta HTTP y la serie de respuesta. Por ejemplo, `200 OK`.

HTTP_USER_AGENT
Contiene el valor de la cabecera de petición User-Agent, que es el nombre del navegador Web del cliente, por ejemplo, `Netscape Navigator / V2.02`. El establecimiento de esta variable afecta a la cabecera en la respuesta del proxy al cliente. Su extracción se aplica a la cabecera desde la petición del cliente.

INIT_STRING
Es de sólo lectura. La directiva ServerInit define esta serie. Esta variable sólo puede ser de lectura durante el paso Server Initialization.

LAST_MODIFIED
La extracción de esta variable es aplicable a la cabecera content-charset desde la petición del cliente. Su establecimiento afecta al valor de la cabecera en la petición al servidor de contenido. Por ejemplo:

```
Mon, 01 Mar 1998 19:41:17 GMT
```

LOCAL_VARIABLES
Es de sólo lectura. Todas las variables definidas por el usuario.

MAXACTIVETHREADS
Es de sólo lectura. Número máximo de hebras activas.

NOTMODIFIED_TO_OK
Fuerza una respuesta completa al cliente. Es válida en los pasos PreExit y ProxyAdvisor.

ORIGINAL_HOST
Es de sólo lectura. Devuelve el nombre de sistema principal o la dirección IP de destino de una petición.

ORIGINAL_URL
Es de sólo lectura. Devuelve el URL original enviado en la petición del cliente.

OVERRIDE_HTTP_NOTRANSFORM
Habilita la modificación de los datos en presencia de una cabecera Cache-Control: no-transform. El establecimiento de esta variable afecta a la cabecera de respuesta al cliente.

OVERRIDE_PROXY_NOTRANSFORM
Habilita la modificación de los datos en presencia de una cabecera Cache-Control: no-transform. El establecimiento de esta variable afecta a la respuesta al servidor de contenido.

PASSWORD
Para autenticación básica, contiene la contraseña decodificada. Por ejemplo, `password`.

PATH Contiene la vía de acceso totalmente convertida.

PATH_INFO
Contiene información de vía de acceso adicional tal como la envía el navegador Web. Por ejemplo, `/foo`.

PATH_TRANSLATED
Contiene la versión decodificada o convertida de la información de vía de acceso contenida en PATH_INFO. Por ejemplo:

```
d:\wwwhome\foo
```

```
/wwwhome/foo
```

PPATH
Contiene la vía de acceso parcialmente convertida. Utilícela en el paso Name Translation.

PROXIED_CONTENT_LENGTH
Es de sólo lectura. Devuelve la longitud de los datos de respuesta que se han transferido realmente a través del servidor proxy.

PROXY_ACCESS
Define si la petición es una petición de proxy. Por ejemplo, `NO`.

PROXY_CONTENT_TYPE
Contiene la cabecera Content-Type de la petición de proxy realizada a través de HTTPD_proxy(). Cuando se envía información con el método de POST, esta variable contiene el tipo de datos incluidos. Puede crear un tipo de contenido propio en el archivo de configuración del servidor proxy y correlacionarlo con un visor. La extracción de esta variable se aplica al valor de cabecera desde la respuesta del servidor de contenido. Su establecimiento afecta a la cabecera de la petición al servidor de contenido. Por ejemplo:

```
application/x-www-form-urlencoded
```

PROXY_CONTENT_LENGTH
Cabecera Content-Length de la petición de proxy realizada a través de HTTPD_proxy(). Cuando se envía esta información con el método de POST, esta variable contiene el número de caracteres de datos. Los servidores no suelen enviar un distintivo de final de archivo cuando reenvían la información mediante la entrada estándar. Si es necesario, puede utilizar el valor de CONTENT_LENGTH para determinar el final de la serie de entrada. La extracción de esta variable se aplica al valor de cabecera desde la respuesta del servidor de contenido. Su establecimiento afecta a la cabecera de la petición al servidor de contenido. Por ejemplo:

```
7034
```

PROXY_COOKIE
Cuando se lee, esta variable contiene el valor de la cabecera Set-Cookie establecida por el servidor de origen. También se puede utilizar para establecer una nueva cookie en la secuencia de peticiones. Al establecer esta variable, se crea una nueva cabecera Set-Cookie en la secuencia de peticiones del documento, independientemente de si existe una cabecera duplicada.

PROXY_HEADERS
Es de sólo lectura. Se utiliza para extraer las cabeceras de proxy.

PROXY_METHOD
Método de la petición realizada a través de HTTPD_proxy(). La extracción de esta variable se aplica al valor de cabecera desde la respuesta del servidor de contenido. Su establecimiento afecta a la cabecera de la petición al servidor de contenido.

QUERY_STRING
Cuando se envía información utilizando un método de GET, esta variable contiene la información que sigue a un interrogante (?) en una consulta. El programa de CGI debe decodificar esta información. Por ejemplo:

```
NAME=Eugene+T%2E+Fox&ADDR=etfox%7Cibm.net&INTEREST=xyz
```

RCA_OWNER
Es de sólo lectura. Devuelve un valor numérico que indica el nodo

propietario del objeto solicitado. Esta variable se puede utilizar en los pasos PostExit, ProxyAdvisor o Log y sólo tiene sentido cuando el servidor forma parte de una matriz de antememoria que utiliza el acceso a antememoria remota (RCA).

RCA_TIMEOUTS
Es de sólo lectura. Devuelve un valor numérico, que contiene el número total (agregado) de tiempos de espera excedidos en las peticiones de RCA a todos los iguales. Puede utilizar esta variable en cualquier paso.

REDIRECT_*
Es de sólo lectura. Contiene una serie de redirección para el código de error que corresponde al nombre de variable (por ejemplo, `REDIRECT_URL`). Puede encontrar una lista de variables REDIRECT_ posibles en la documentación en línea de Apache Web Server en http://httpd.apache.org/docs-2.0/custom-error.html.

REFERRER_URL
Es de sólo lectura. Contiene la última ubicación de URL del navegador. Permite al cliente especificar, a beneficio del servidor, la dirección (URL) del recurso del que se ha obtenido Request-URL. Por ejemplo:

```
http://www.company.com/homepage
```

REMOTE_ADDR
Contiene la dirección IP del navegador Web, si está disponible. Por ejemplo, `45.23.06.8`.

REMOTE_HOST
Contiene el nombre de sistema principal del navegador Web, si está disponible. Por ejemplo, `www.raleigh.ibm.com`.

REMOTE_USER
Si el servidor da soporte a la autenticación de clientes y el script está protegido, esta variable contiene el nombre de usuario que se ha pasado para autenticación. Por ejemplo, `joeuser`.

REQHDR
Es de sólo lectura. Contiene una lista de las cabeceras enviadas por el cliente.

REQUEST_CONTENT_TYPE
Es de sólo lectura. Devuelve el tipo de contenido del cuerpo de la petición. Por ejemplo:

```
application/x-www-form-urlencoded
```

REQUEST_CONTENT_LENGTH
Es de sólo lectura. Cuando se envía esta información con el método de POST, esta variable contiene el número de caracteres de datos. Los servidores no suelen enviar un distintivo de final de archivo cuando reenvían la información mediante la entrada estándar. Si es necesario, puede utilizar el valor de CONTENT_LENGTH para determinar el final de la serie de entrada. Por ejemplo, `7034`.

REQUEST_METHOD
Es de sólo lectura. Contiene el método (tal como se especifica con el atributo METHOD en un formulario HTML) que se utiliza para enviar la petición. Por ejemplo, `GET o POST`.

REQUEST_PORT
Es de sólo lectura. Devuelve el número de puerto especificado en el URL o un puerto por omisión basado en el protocolo.

RESPONSE_CONTENT_TYPE
Es de sólo lectura. Cuando se envía información con el método de POST, esta variable contiene el tipo de datos incluidos. Puede crear un tipo de contenido propio en el archivo de configuración del servidor proxy y correlacionarlo con un visor. Por ejemplo, `text/html`.

RESPONSE_CONTENT_LENGTH
Es de sólo lectura. Cuando se envía esta información con el método de POST, esta variable contiene el número de caracteres de datos. Los servidores no suelen enviar un distintivo de final de archivo cuando reenvían la información mediante la entrada estándar. Si es necesario, puede utilizar el valor de CONTENT_LENGTH para determinar el final de la serie de entrada. Por ejemplo, `7034`.

RULE_FILE_PATH
Es de sólo lectura. Contiene la vía de acceso al sistema de archivos plenamente cualificada y el nombre del archivo de configuración.

SSL_SESSIONID
Es de sólo lectura. Devuelve el ID de sesión de SSL si la petición actual se ha recibido en una conexión SSL. Devuelve NULL si la petición actual no se ha recibido en una conexión SSL.

SCRIPT_NAME
Contiene el URL de la petición.

SERVER_ADDR
Es de sólo lectura. Contiene la dirección IP local del servidor proxy.

SERVER_NAME
Es de sólo lectura. Contiene el nombre de sistema principal del servidor proxy o dirección IP del servidor de contenido para esta petición. Por ejemplo, `www.ibm.com`.

SERVER_PORT
Es de sólo lectura. Contiene el número de puerto del servidor proxy al que se ha enviado la petición de cliente. Por ejemplo, `80`.

SERVER_PROTOCOL
Es de sólo lectura. Contiene el nombre y la versión del protocolo que se ha utilizado para realizar la petición. Por ejemplo, `HTTP/1.1`.

SERVER_ROOT
Es de sólo lectura. Contiene el directorio donde está instalado el programa del servidor proxy.

SERVER_SOFTWARE
Es de sólo lectura. Contiene el nombre y la versión del servidor proxy.

STATUS
Contiene el código de respuesta HTTP y la serie de respuesta. Por ejemplo, 200 OK.

TRACE
Determina cuánta información se rastreará. Los valores que se devuelven son:

- OFF: sin rastreo.
- V: modalidad de salida detallada.
- VV: modalidad de salida muy detallada.
- MTV: modalidad de salida demasiado detallada.

URI Lectura/grabación. Es igual que DOCUMENT_URL.

URI_PATH
Es de sólo lectura. Devuelve sólo la parte de vía de acceso de un URL.

URL Lectura/escritura. Es igual que DOCUMENT_URL.

URL_MD4
Es de sólo lectura. Devuelve el nombre del archivo posible de antememoria para la petición actual.

USE_PROXY
Identifica el proxy con el que se debe encadenar para la petición actual. Especifique el URL. Por ejemplo, `http://myproxy:8080`.

USERID
Es igual que REMOTE_USER.

USERNAME
Es igual que REMOTE_USER.

Autenticación y autorización

En primer lugar, revisemos brevemente la terminología:

Autenticación
Verificación de los símbolos de seguridad asociados con esta petición a fin de determinar la identidad del solicitante.

Autorización
Proceso que utiliza los símbolos de seguridad para determinar si el solicitante tiene acceso al recurso.

En la Figura 3 en la página 37 se detalla el proceso de autenticación y autorización del servidor proxy.

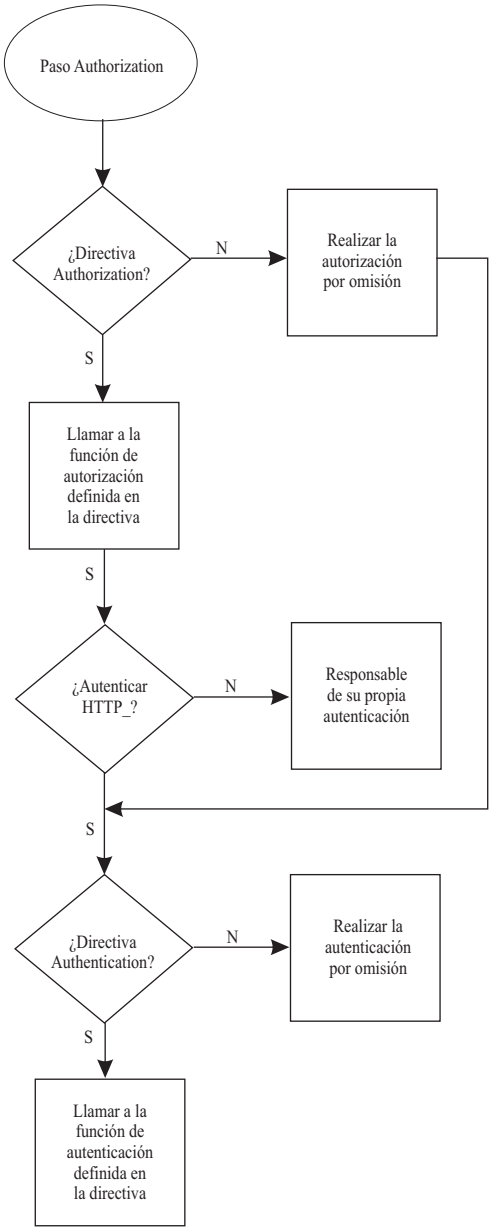

Figura 3. Proceso de autenticación y autorización del servidor proxy

Como se muestra en la Figura 3, el inicio del proceso de autorización es el primer paso del proceso de autorización y autenticación del servidor.

En Caching Proxy, la autenticación es parte del proceso de autorización; sólo tiene lugar cuando se exige una autorización.

Proceso de autenticación y autorización

El servidor proxy sigue estos pasos al procesar una petición que exige autorización.

1. En primer lugar, el servidor proxy examina su archivo de configuración para determinar si hay o no una directiva de autorización.
   - Si hay una directiva de autorización en el archivo de configuración, el servidor llama a la función de autorización definida en la directiva e inicia la autenticación con el paso 2 en la página 38.

- Si no hay ninguna directiva de autorización, el servidor lleva a cabo una autorización por omisión y, a continuación, continúa directamente con los procedimientos de autenticación del paso 3.

2. El servidor proxy inicia el proceso de autenticación comprobando si la cabecera HTTP_authenticate está presente en la petición del cliente.
   - Si la cabecera está presente, el servidor continúa el proceso de autenticación (consulte el paso 3).
   - Si la cabecera no está presente, otro método debe realizar la autenticación.
3. El servidor proxy comprueba si hay una directiva de autenticación presente en el archivo de configuración.
   - Si hay presente una directiva de autenticación en el archivo de configuración, el servidor llama a la función de autenticación definida en la directiva.
   - Si no hay ninguna directiva, el servidor realiza una autenticación por omisión.

Si el plug-in de Caching Proxy proporciona su propio proceso de autorización, *altera temporalmente* la autorización y la autenticación de servidor por omisión. Por lo tanto, si tiene directivas de autorización en el archivo de configuración, las funciones de plug-in asociadas con ellas deben gestionar también las autenticaciones necesarias. Se proporciona la función HTTPD_authenticate() predefinida para que la utilice.

Estos son los tres modos de proporcionar autenticación en los plug-ins de autorización:

- Escriba sus propios plug-ins de autorización y autenticación independientes. En el archivo de configuración de proxy, utilice las directivas Authorization y Authentication para especificar estas funciones. Asegúrese de incluir la llamada a la función HTTPD_authenticate() en la función de plug-in de autorización.

  Cuando se ejecuta el paso Authorization, realiza la función de plug-in de autorización, que, por su parte, llama a la función de plug-in de autenticación.
- Escriba su propia función de plug-in de autorización, pero haga que llame a la autenticación de servidor por omisión. En el archivo de configuración de proxy, utilice la directiva Authorization para especificar la función. En tal caso, no necesita la directiva Authentication. Asegúrese de llamar a la función HTTPD_authenticate() en la función de plug-in de autorización.

  Cuando se ejecuta el paso Authorization, lleva a cabo la función de plug-in de autorización, que, por su parte, llama a la autenticación de servidor por omisión.
- Escriba su propia función de plug-in de autorización e incluya en ella todo el proceso de autenticación necesario. No utilice la función HTTPD_authenticate() en el plug-in de autorización. En el archivo de configuración de proxy, utilice la directiva Authorization para especificar el plug-in de autorización. En tal caso, no necesita la directiva Authentication.

  Cuando se ejecuta el paso Authorization, realiza la función de plug-in de autorización y las autenticaciones que incluya.

Si el plug-in de Caching Proxy no proporciona un proceso de autorización propia, aún puede proporcionar una autenticación personalizada utilizando el método siguiente:

- Escriba su propia función de plug-in de autenticación. En el archivo de configuración de proxy, utilice las directivas Authentication para especificar la función. En tal caso, no necesita la directiva Authorization.

Cuando se ejecuta el paso Authorization, realiza la autorización de servidor por omisión, que, por su parte, llama a la función de plug-in de autenticación.

Recuerde los puntos siguientes:

- Si no tiene directivas Authorization en el archivo de configuración o si sus funciones de plug-in especificadas rechazan la gestión de la petición devolviendo HTTP_NOACTION, tiene lugar la autorización por omisión del servidor.
- Si tiene directivas Authorization en el archivo de configuración y sus funciones de plug-in incluyen HTTPD_authenticate(), el servidor llama a las funciones de autenticación especificadas en las directivas Authentication. Si no ha definido ninguna directiva Authentication o sus funciones de plug-in especificadas rechazan gestionar la petición devolviendo HTTP_NOACTION, tiene lugar la autenticación por omisión del servidor.
- Si tiene directivas Authorization en el archivo de configuración pero sus funciones de plug-in no incluyen HTTPD_authenticate(), el servidor no llama a ninguna función de autenticación. Debe escribir su propio proceso de autenticación como parte de las funciones de plug-in de autorización o realizar sus propias llamadas a otros módulos de autenticación.
- Caching Proxy genera de forma automática una petición (pide al navegador que devuelva un ID de usuario y una contraseña) si la función de autorización devuelve los códigos `401` o `407`. No obstante, aún debe crear una configuración de protección en Caching Proxy para que esta acción se realice correctamente.

Almacenamiento de variants en antememoria

Utilice el almacenamiento de variants en antememoria para guardar en antememoria los datos que se hayan modificado del documento original (el URI). Caching Proxy gestiona los variants generados por la API. Los *variants* son distintas versiones de un documento base.

Por lo general, cuando los servidores de origen envían variants, no pueden identificarse como tales. Caching Proxy sólo da soporte a variants creados por plug-ins (por ejemplo, conversión de páginas de código). Si un plug-in crea un variant según criterios que no están en la cabecera HTTP, debe incluir una función de paso PreExit o PostAuthorization para crear una seudocabecera a fin de que Caching Proxy puede identificar correctamente el variant existente.

Por ejemplo, utilice el programa de la API de Transmogrifier para modificar los datos que los usuarios soliciten según el valor de la cabecera User-Agent que el navegador envíe. En la función *close*, guarde el contenido modificado en un archivo o especifique un longitud de almacenamiento intermedio y pase el almacenamiento intermedio como argumento de datos. A continuación, utilice las funciones de almacenamiento de variants en antememoria httpd_variant_insert() y httpd_variant_lookup() para colocar el contenido en la antememoria.

Ejemplos de API

Para ayudarle a empezar a utilizar funciones de la API de Caching Proxy propias, observe los programas de ejemplo que se proporcionan en el directorio samples del CD-ROM de instalación de Edge Components. . Hay información adicional disponible en el sitio Web de WebSphere Application Server, www.ibm.com/software/webservers/appserv/.

Capítulo 3. Asesores personalizados

En este apartado se describe cómo escribir asesores personalizados para Load Balancer.

Los asesores proporcionan información de equilibrio de carga

Los asesores son agentes de software que funcionan dentro de Load Balancer para proporcionar información acerca de la carga en un servidor determinado. Existe un asesor distinto para cada protocolo estándar (HTTP, SSL y otros). Periódicamente, el código base de Load Balancer realiza un ciclo de asesor durante el que evalúa de forma individual el estado de todos los servidores de su configuración.

Al escribir asesores propios para Load Balancer, puede personalizar cómo se determina la carga de las máquinas de servidor.

En sistemas Windows: Cuando utilice una instalación Load Balancer para IPv6, si utiliza el protocolo IPv6 en la máquina y desea utilizar asesores deberá modificar el archivo **protocol** que se encuentra en el directorio C:\windows\system32\drivers\etc\.

Para IPv6, inserte la siguiente línea en el archivo protocol:

```
ipv6-icmp  58   IPv6-ICMP      # Protocolo de mensajes de control de interfaz IPv6
```

Función de asesor estándar

Por lo general, los asesores sirven para habilitar el equilibrio de carga de la manera siguiente.

1. El asesor abre periódicamente una conexión con cada servidor y le envía un mensaje de petición. El contenido del mensaje es específico del protocolo que se está ejecutando en el servidor; por ejemplo, el asesor HTTP envía una petición HEAD al servidor.
2. El asesor escucha una respuesta del servidor. Después de obtener la respuesta, el asesor calcula y notifica el valor de carga para ese servidor. Los distintos asesores calculan el valor de carga de modos distintos, pero la mayor parte de los asesores estándar miden el tiempo que el servidor tarda en responder y, a continuación, notifican ese valor en milisegundos como carga.
3. El asesor notifica la carga a la función del gestor de Load Balancer. La carga aparece en la columna de puertos del informe del gestor. El gestor utiliza la carga que ha notificado el asesor junto son los pesos establecidos por el administrador para determinar cómo equilibrar la carga de las peticiones entrantes para los servidores.
4. Si un servidor no responde, el asesor devuelve un valor negativo (-1) para la carga. El gestor utiliza esta información para determinar cuándo se debe suspender el servicio para un servidor concreto.

Los asesores estándar que se proporcionan con Load Balancer incluyen asesores para las funciones siguientes. Hay información detallada disponible sobre estos asesores en la publicación *WebSphere Application Server Load Balancer Administration Guide*.

- Connect
- DB2

- DNS
- FTP
- HTTP
- HTTPS
- IMAP
- LDAP
- NNTP
- Ping
- POP3
- Reach
- Self
- SMTP
- SSL
- Telnet
- WebSphere Application Server
- WebSphere Application Server Caching Proxy
- Workload Manager

Para dar soporte a los protocolos de propiedad para los que no se proporcionan asesores estándar, debe escribir asesores personalizados.

Creación de un asesor personalizado

Un asesor personalizado es un pequeña porción de código Java, proporcionada como archivo de clases, a la que llama el código base de Load Balancer para determinar la carga en un servidor. El código base proporciona todos los servicios administrativos necesarios, incluidos el inicio y la detención de una instancia del asesor personalizado, con estados e informes, registro de la información de historial en un archivo de anotaciones cronológicas y notificación de los resultados del asesor al componente de gestor.

Cuando el código base de Load Balancer llama a un asesor personalizado, tienen lugar los pasos siguientes:

1. El código base de Load Balancer abre una conexión con la máquina del servidor.
2. Si el socket se abre, el código base llama a la función GetLoad del asesor especificado.
3. La función GetLoad del asesor lleva a cabo los pasos que el usuario ha definido para evaluar el estado del servidor, incluida la espera de una respuesta del servidor. La función finaliza la ejecución cuando se recibe la respuesta.
4. El código base de Load Balancer cierra el socket con el servidor y notifica la información de carga al gestor. En función de si el asesor personalizado funciona en modalidad normal o en modalidad de sustitución, el código base a veces realiza cálculos adicionales después de que finalice la función GetLoad.

Modalidad normal y modalidad de sustitución

Se pueden diseñar asesores personalizados para interactuar con Load Balancer en modalidad *normal* o modalidad de *sustitución*.

La elección de la modalidad de funcionamiento se especifica en el archivo del asesor personalizado como parámetro en el método constructor. Cada asesor sólo funcionan en una de estas modalidades, en función de su diseño.

En modalidad normal, el asesor personalizado intercambia datos con el servidor y el código del asesor base controla el tiempo del intercambio y calcula el valor de carga. El código base notifica, a continuación, este valor de carga al gestor. El asesor personalizado devuelve el valor cero para indicar que la operación ha sido satisfactoria o -1 para indicar que se ha producido un error.

Para especificar la modalidad normal, establezca el distintivo de sustitución del constructor en *false*.

En modalidad de sustitución, el código base no realiza ninguna medición de tiempo. El código del asesor personalizado realiza las operaciones que se especifiquen, según sus requisitos exclusivos y, a continuación, devuelve un número de carga real. El código base acepta el número de carga y lo notifica, sin alterar, al gestor. Para obtener resultados óptimos, normalice los números de carga entre 10 y 1000, donde 10 representa un servidor rápido y 1000 representa un servidor lento.

Para especificar una modalidad de sustitución, establezca el distintivo replace del constructor en *true*.

Convenios de denominación de asesores

Los nombres de archivo de asesor personalizado deben tener el formato ADV_*nombre*.java, donde *nombre* es el nombre que elija para el asesor. El nombre completo debe empezar por el prefijo ADV_ en mayúsculas y los caracteres posteriores deben ser en minúscula. El requisito de minúsculas garantiza que el mandato para ejecutar el asesor no sea sensible a mayúsculas y minúsculas.

Según las convenciones Java, el nombre de la clase definida en el archivo debe coincidir con el nombre del archivo.

Compilación

Los asesores personalizados se deben escribir en lenguaje Java y se deben compilar con un compilador Java que esté instalado en la máquina de desarrollo. Durante la compilación, se hace referencia a los archivos siguientes:

- Archivo del asesor personalizado
- Archivo de clases base, ibmnd.jar, que se encuentra en el directorio *vía_acceso_instalación*/servers/lib.

La variable de entorno classpath debe apuntar tanto al archivo de asesor personalizado como al archivo de clases base durante la compilación. Un mandato de compilación puede tener el formato siguiente:

```
javac -classpath /opt/ibm/edge/lb/servers/lib/ibmnd.jar ADV_nombre.java
```

En este ejemplo se utiliza la vía de acceso de instalación de Linux y UNIX por omisión. El archivo de asesor se denomina ADV_*nombre*.java y el archivo de asesor se almacena en el directorio actual.

La salida de la compilación es un archivo de clases, por ejemplo ADV_*nombre*.class. Antes de iniciar el asesor, copie el archivo de clases al directorio *vía_acceso_instalación*/servers/lib/CustomAdvisors/.

Nota: Los asesores personalizados se pueden compilar en un sistema operativo y ejecutarse en otro. Por ejemplo, puede compilar el asesor en un sistema Windows, copiar el archivo de clases resultante (en formato binario) en una máquina Linux y ejecutar allí el asesor personalizado.

Ejecución de un asesor personalizado

Para ejecutar el asesor personalizado, debe copiar en primer lugar el archivo de clases del asesor en el subdirectorio lib/CustomAdvisors/ de la máquina de Load Balancer. Por ejemplo, para un asesor personalizado denominado miping, la vía de acceso del archivo es
vía_acceso_instalación/servers/lib/CustomAdvisors/ADV_miping.class.

Configure Load Balancer, inicie su función de gestor y ejecute el mandato para iniciar el asesor personalizado. El asesor personalizado se especifica por su nombre, excluyendo el prefijo ADV_ y la extensión de archivo:

```
dscontrol advisor start miping número_puerto
```

El número de puerto especificado en el mandato es el puerto en el que el asesor abrirá una conexión con el servidor de destino.

Rutinas obligatorias

Como todos los asesores, un asesor personalizado amplía las funciones de la clase base del asesor, que se denomina ADV_Base. La base del asesor realiza la mayor parte de las funciones del asesor, como volver a notificar las cargas al gestor para que las utilice en el algoritmo de peso del gestor. La base del asesor también realiza operaciones de conexión y cierre de socket y proporciona métodos de envío y recepción para que el asesor las utilice. El asesor sólo se utiliza para enviar y recibir datos en el puerto especificado del servidor que se está investigando. Los métodos TCP que se proporcionan en la base del asesor se temporizan para calcular la carga. Un distintivo dentro del constructor de la base del asesor sobrescribe la carga existente con la nueva carga que devuelve el asesor, si se desea.

Nota: Según un valor establecido en el constructor, la base del asesor proporciona la carga al algoritmo de peso a intervalos específicos. Si el asesor no ha realizado el proceso y no puede devolver una carga válida, la base del asesor utiliza la carga notificada anteriormente.

Los asesores tienen los siguientes métodos de clase base:

- Rutina de constructor. El constructor llama al constructor de clase base.
- Método ADV_AdvisorInitialize. Este método proporciona una manera de realizar pasos adicionales después de que la clase base finalice su inicialización.
- Rutina getLoad. La clase de asesor base abre el socket; la función getLoad sólo tiene que ejecutar las peticiones de envío y recepción adecuadas para finalizar el ciclo de asesoramiento.

Los detalles sobre estas rutinas obligatorias aparecen más adelante en este apartado.

Orden de búsqueda

Se llama a asesores personalizados después de buscar asesores nativos o estándar. Si Load Balancer no encuentra un asesor especificado en la lista de asesores

estándar, consulta la lista de asesores personalizados. Hay información adicional disponible sobre la utilización de asesores en la publicación *WebSphere Application Server Load Balancer Administration Guide*.

Denominación y vía de acceso a archivos

Recuerde los requisitos siguientes para los nombres y las vías de acceso de los asesores personalizados.

- El asesor personalizado debe tener un nombre en caracteres alfabéticos en minúscula a fin de eliminar la necesidad de entrar caracteres sensibles a mayúsculas y minúsculas cuando un operador escribe mandatos en una línea de mandatos. El nombre del asesor debe tener el prefijo ADV_.
- La clase de asesor personalizado debe estar en el subdirectorio lib/CustomAdvisors. La ubicación por omisión de este directorio es /opt/ibm/edge/lb/servers/lib/CustomAdvisors en sistemas Linux y UNIX y C:\Archivos de programa\IBM\edge\lb\servers\lib\CustomAdvisors\ en sistemas Windows.

Métodos y llamadas a funciones de asesor personalizado

Constructor (lo proporciona la base del asesor)

```
void ADV_Base Constructor (
        string sName;
        string sVersion;
        int iDefaultPort;
        int iInterval;
        string sDefaultLogFileName;
        boolean replace
)
```

sName
Nombre del asesor personalizado.

sVersion
Versión del asesor personalizado.

iDefaultPort
Número de puerto en el que se debe contactar con el servidor si no se especifica ningún número de puerto en la llamada.

iInterval
Intervalo transcurrido el cual el asesor consultará a los servidores.

sDefaultLogFileName
Este parámetro es obligatorio pero no se utiliza. El único valor aceptable es una serie nula, "".

replace
Si este asesor funciona o no en modalidad de *sustitución*. Los valores posibles son los siguientes:

- true: sustituye la carga que calcula el código base del asesor por el valor que ha notificado el asesor personalizado.
- false: añade el valor de carga que ha notificado el asesor personalizado al valor de carga calculado por el código base del asesor.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

Este método se proporciona para realizar cualquier inicialización que pueda ser obligatoria para el asesor personalizado. Se llama a este método después de que se inicie el módulo base del asesor.

En muchos casos, incluidos los asesores estándar, este método no se utiliza y su código consta sólo de una sentencia *return*. Este método se puede utilizar para llamar al método suppressBaseOpeningSocket, que sólo es válido desde este método.

getLoad()

```
int getLoad(
        int iConnectTime;
        ADV_Thread *caller
)
```

iConnectTime
Tiempo, en milisegundos, que la conexión ha tardado en realizarse. Esta medición de carga la realiza el código base del asesor y se pasa al código del asesor personalizado, que puede utilizar o ignorar la medición al devolver el valor de carga. Si la conexión falla, este valor se establece en -1.

caller
Instancia de la clase base del asesor donde se proporcionan métodos base de asesor.

Llamadas a funciones disponibles para los asesores personalizados

Los métodos, o funciones, que se describen en los apartados siguientes, se pueden llamar desde asesores personalizados. El código base del asesor da soporte a estos métodos.

Algunas de estas llamadas a funciones se pueden realizar directamente, por ejemplo, *nombre_función*(), pero otras exigen el prefijo `caller`. *Caller* representa la instancia de asesor base que da soporte al asesor base que se va a ejecutar.

ADVLOG()

La función ADVLOG permite que un asesor personalizado grabe un mensaje de texto en el archivo de anotaciones cronológicas base del asesor. El formato es:

```
void  ADVLOG  (int logLevel, string message)
```

logLevel
Nivel de estado en el que el mensaje se graba en el archivo de anotaciones cronológicas. El archivo de anotaciones cronológicas del asesor se organiza en etapas; a los mensajes más urgentes se les asigna el nivel de estado 0 y los mensajes menos urgentes reciben los números más altos. Al tipo de mensaje más detallado se le asigna el nivel de estado 5. Estos niveles se utilizan para controlar los tipos de mensajes que el usuario recibe en tiempo real (el mandato **dscontrol** se utiliza para establecer el grado de detalle). Los errores muy graves se deben registrar en el nivel 0.

message
Mensaje que se debe grabar en el archivo de anotaciones cronológicas. El valor de este parámetro es una serie Java estándar.

getAdvisorName()

La función getAdvisorName devuelve una serie Java con la parte de sufijo del nombre del asesor personalizado. Por ejemplo, para un asesor denominado ADV_cdload.java, esta función devuelve el valor cdload.

Esta función no adopta ningún parámetro.

Tenga en cuenta que no es posible que este valor cambie durante la creación de una instancia de un asesor.

getAdviseOnPort()

La función getAdviseOnPort devuelve el número de puerto en el que se está ejecutando el asesor personalizado que realiza la llamada. El valor de retorno es un entero Java (int) y la función no adopta ningún parámetro.

Tenga en cuenta que no es posible que este valor cambie durante la creación de una instancia de un asesor.

caller.getCurrentServer()

La función getCurrentServer devuelve la dirección IP del servidor actual. El valor de retorno es una serie Java en formato de dirección IP, por ejemplo, 128.0.72.139.

Normalmente, esta dirección cambia cada vez que se llama al asesor personalizado porque el código base del asesor consulta todas las máquinas de servidor en serie.

Esta función no adopta ningún parámetro.

caller.getCurrentCluster()

La función getCurrentCluster devuelve la dirección IP del clúster de servidores actual. El valor de retorno es una serie Java en formato de dirección IP, por ejemplo, 128.0.72.139.

Normalmente, esta dirección cambia cada vez que se llama al asesor personalizado porque el código base del asesor consulta todas los clústeres de servidores en serie.

Esta función no adopta ningún parámetro.

getInterval()

La función getInterval devuelve un intervalo de asesor, es decir, el número de segundos entre ciclos de asesor. Este valor es igual al valor por omisión establecido en el constructor del asesor personalizado, a menos que se haya modificado el valor durante la ejecución mediante el mandato **dscontrol**.

El valor de retorno es un entero Java (int). La función no adopta ningún parámetro.

caller.getLatestLoad()

La función getLatestLoad permite que un asesor personalizado obtenga el valor de carga más reciente para un objeto de servidor determinado. El código base del asesor y el daemon de gestor mantienen los valores de carga en tablas internas.

```
int caller.getLatestLoad (string cluster_IP, int port, string server_IP)
```

Los tres argumentos juntos definen un objeto de servidor.

cluster_IP
Dirección IP del clúster del objeto de servidor para el que se debe obtener el valor de carga actual. Este argumento debe ser una serie Java en formato de dirección IP, por ejemplo 245.145.62.81.

port
Número de puerto del objeto de servidor para el que se debe obtener el valor de carga actual.

server_IP
Dirección IP del objeto de servidor para el que se debe obtener el valor de carga actual. Este argumento debe ser una serie Java en formato de dirección IP, por ejemplo 192.255.201.3.

El valor de retorno es un entero.

- Un valor de retorno positivo representa el valor de carga real asignado para el objeto que se ha consultado.
- El valor -1 indica que el servidor al que se le ha realizado la petición no está en funcionamiento.
- El valor -2 indica que el estado del servidor al que se ha realizado la petición es desconocido.

Esta llamada a función es útil si desea que el comportamiento de un protocolo o un puerto dependa del comportamiento de otro. Por ejemplo, puede utilizar esta llamada a función en un asesor personalizado que haya inhabilitado un servidor de aplicaciones concreto si el servidor Telnet de esa misma máquina se había inhabilitado.

caller.receive()

La función receive obtiene información de la conexión de socket.

```
caller.receive(stringbuffer *response)
```

El parámetro *response* es un almacenamiento intermedio de series en el que se colocan los datos recuperados. Además, la función devuelve un valor entero con el significado siguiente:

- 0 indica que los datos se han enviado satisfactoriamente.
- Un número negativo indica un error.

caller.send()

La función send utiliza la conexión de socket establecida para enviar un paquete de datos al servidor, con el puerto especificado.

```
caller.send(string command)
```

El parámetro *command* es una serie que contiene los datos que se deben enviar al servidor. La función devuelve un valor entero con el significado siguiente:

- 0 indica que los datos se han enviado satisfactoriamente.
- Un número negativo indica un error.

suppressBaseOpeningSocket()

La llamada a la función suppressBaseOpeningSocket permite que un asesor personalizado especifique si el código de asesor base abre un socket TCP para el servidor en nombre del asesor personalizado. Si el asesor no utiliza una comunicación directa con el servidor para determinar su estado, puede que no sea necesario abrir el socket.

La llamada a función sólo se puede emitir una vez y debe emitirse desde la rutina ADV_AdvisorInitialize.

La función no adopta ningún parámetro.

Ejemplos

En los ejemplos siguientes se muestra cómo se pueden implementar asesores personalizados.

Asesor estándar

Este código fuente de ejemplo es similar al asesor HTTP de Load Balancer. Funciona de la manera siguiente:

1. Se emite una petición de envío, mandato "HEAD/HTTP".
2. Se recibe una respuesta. La información no se analiza, pero la respuesta hace que el método getLoad finalice.
3. El método getLoad devuelve 0 para indicar que la ejecución ha sido satisfactoria o -1 para indicar que la ejecución no lo ha sido.

El asesor funciona en modalidad normal, por lo que la medición de la carga se basa en el tiempo transcurrido en milisegundos para realizar las operaciones de apertura, envío, recepción y cierre de socket.

```
package CustomAdvisors;
import com.ibm.internet.lb.advisors.*;
public class ADV_sample extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME ="Sample";
  static final int ADV_DEF_ADV_ON_PORT = 80;
  static final int ADV_DEF_INTERVAL = 7;
  static final string ADV_SEND_REQUEST =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_Load_Balancer_HTTP_Advisor\r\n\r\n";

//--------
// Constructor

  public ADV_sample() {
    super(ADV_NAME, "3.0.0.0-03.31.00",
          ADV_DEF_ADV_ON_PORT, ADV_DEF_INTERVAL, "",
          false);
    super.setAdvisor( this );
  }

//--------
// ADV_AdvisorInitialize

  public void ADV_AdvisorInitialize() {
    return;                                  // normalmente una rutina vacía
  }

//--------
// getLoad

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;       // inicializa para inaccesible

    iRc = caller.send(ADV_SEND_REQUEST);     // envía la petición HTTP al
                                             // servidor
    if (0 <= iRc) {                          // si el envío es satisfactorio
      StringBuffer sbReceiveData = new StringBuffer(""); // asigna un almacenamiento
                                                         // intermedio para la
                                                         // respuesta
      iRc = caller.receive(sbReceiveData);   // recibe el resultado

      // analiza el resultado aquí si es necesario

      if (0 <= iRc) {          // si la recepción es satisfactoria
```

```
        iLoad = 0;             // devuelve 0 si es satisfactoria
      }                        // (la base ignora el valor de carga
    }                          //  del asesor en modalidad normal)
    return iLoad;
  }
}
```

Asesor de secuencia lateral

Este ejemplo ilustra la supresión del socket estándar que ha abierto la base del asesor. En lugar de esto, este asesor abre un socket Java de secuencia lateral para consultar a un servidor. Este procedimiento puede ser útil para los servidores que utilicen un puerto distinto al del tráfico de cliente normal para escuchar una consulta de asesor.

En este ejemplo, un servidor escucha en el puerto 11999 y, cuando se le solicita, devuelve un valor de carga con un entero hexadecimal ″4″. Este ejemplo se ejecuta en modalidad de sustitución, es decir, el último parámetro del constructor del asesor se establece en true y el código base del asesor utiliza el valor de carga devuelto en lugar del tiempo transcurrido.

Observe la llamada a supressBaseOpeningSocket() en la rutina de inicialización. La supresión del socket base no es obligatoria cuando no se envíen datos. Por ejemplo, puede que desee abrir el socket para asegurarse de que el asesor puede ponerse en contacto con el servidor. Examine las necesidades de la aplicación con detenimiento antes de realizar la elección.

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

public class ADV_sidea extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME = "sidea";
  static final int ADV_DEF_ADV_ON_PORT = 12345;
  static final int ADV_DEF_INTERVAL = 7;

  // crea una matriz de bytes con el mensaje de petición de carga
  static final byte[] abHealth = {(byte)0x00, (byte)0x00, (byte)0x00,
                                  (byte)0x04};

  public ADV_sidea() {
    super(ADV_NAME, "3.0.0.0-03.31.00", ADV_DEF_ADV_ON_PORT,
          ADV_DEF_INTERVAL, "",
          true);         // el parámetro de modalidad de sustitución es true
    super.setAdvisor( this );
  }

//--------
// ADV_AdvisorInitialize

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // indica al código base que no abra el
                                   // socket estándar
    return;
  }

//--------
// getLoad
```

```
  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999; // puerto en el que se debe comunicar con el servidor

    string sServer = caller.getCurrentServer();   // dirección del servidor que se
                                                  // debe consultar
    try {
      socket soServer = new Socket(sServer, iControlPort);  // abre el socket para
                                                            // el servidor
      DataInputStream disServer = new DataInputStream(
                                    soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                     soServer.getOutputStream());

      int iRecvTimeout = 10000;  // tiempo de espera establecido (en milisegundos)
                                 // para recibir datos
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // envía un mensaje al servidor
      dosServer.flush();

      iLoad = disServer.readByte();   // recibe la respuesta del servidor

    } catch (exception e) {
      system.out.println("Caught exception " + e);
    }
    return iLoad;    // devuelve la carga notificada desde el servidor
  }
}
```

Asesor de dos puertos

Este ejemplo de asesor personalizado es una demostración de la posibilidad de detectar una anomalía en un puerto de un servidor según su propio estado y el estado de un daemon de servidor distinto que se ejecute en otro puerto de la misma máquina de servidor. Por ejemplo, si el daemon HTTP en el puerto 80 deja de responder, puede que también desee detener el direccionamiento de tráfico al daemon SSL en el puerto 443.

Este asesor es más agresivo que los asesores estándar, porque considera que cualquier servidor que no envía una respuesta ha dejado de funcionar y lo marca como *sin funcionamiento*. Los asesores estándar consideran que los servidores que no responden son muy lentos. Este asesor marca un servidor como que no está en funcionamiento tanto para el puerto HTTP como para el puerto SSL si uno de los dos puertos no responde.

Para utilizar este asesor personalizado, el administrador inicia dos instancias del asesor: una en el puerto HTTP y otra en el puerto SSL. El asesor crea instancias de dos tablas hash globales estáticas, una para HTTP y una para SSL. Cada asesor intenta comunicar con su daemon de servidor y almacena el resultado de este suceso en su tabla hash. El valor que cada asesor devuelve a la clase de asesor base depende tanto de la capacidad de comunicar con su propio daemon de servidor como de la capacidad del asesor asociado para comunicarse con su daemon.

Se utilizan los siguientes métodos personalizados.

- ADV_nte() es un objeto de contenedor simple para mantener información acerca de un servidor. Estos objetos se almacenan en la tabla hash como elementos de tabla. Cada objeto tiene una indicación de la hora que se utiliza para determinar si el elemento es actual.
- putNte() y getNte() son métodos sincronizados que garantizan que las dos instancias de asesor accedan a la tabla hash de forma controlada.
- getLoadHTTP es un método que consulta la capacidad de respuesta de un servidor HTTP. Es una rutina de bajo nivel y no recopila ni utiliza información acerca de SSL.
- getLoadSSL() es un método que consulta la capacidad de respuesta de un servidor SSL. Es una rutina de bajo nivel y no recopila ni utiliza información acerca de HTTP.
- getLoad() es la rutina de punto de entrada para este asesor personalizado. Puede gestionar ambos protocolos y puede almacenar y recuperar información de la tabla hash. Esta es la rutina que enlaza los dos puertos.

Se detectan las siguientes condiciones de error.

- Máquina de servidor que no responde: las clases de asesor base envían una señal ping periódicamente a la dirección del servidor. Si no se puede acceder a la dirección, las clases de asesor base marcan el servidor como sin funcionamiento. No se llama a ninguna de las dos instancias del asesor personalizado y los dos servidores de esa máquina se marcan como sin funcionamiento.
- Un daemon de una máquina de servidor no responde, pero el otro funciona: cuando el código base intenta abrir un socket con el servidor, la conexión se rechaza y el asesor base de este protocolo marca el servidor como sin funcionamiento. No se llama al código de asesor personalizado de ese protocolo. Aunque el asesor personalizado del otro protocolo continúa comunicando con su servidor, la tabla hash le indica que el otro asesor personalizado no puede comunicar con su daemon de servidor. Por lo tanto, el asesor del segundo protocolo marca su servidor como sin funcionamiento.
- Un daemon no envía una respuesta, pero el otro daemon sí: el asesor personalizado del protocolo que no responde detecta la anomalía de comunicación, marca el servidor como sin funcionamiento y almacena los datos en la tabla hash. La tabla hash indica esta información al asesor personalizado del otro puerto y éste marca su servidor como sin funcionamiento.

Este ejemplo se ha escrito para enlazar los puertos 80 para HTTP y 443 para SSL, pero se puede adaptar a cualquier combinación de puertos.

```
package CustomAdvisors;
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Date;
import com.ibm.internet.lb.advisors.*;
import com.ibm.internet.lb.common.*;
import com.ibm.internet.lb.manager.*;
import com.ibm.internet.lb.server.SRV_ConfigServer;

//--------
// Define el elemento de tabla de las tablas hash utilizadas en este
// asesor personalizado

class ADV_nte implements Cloneable {
  private string  sCluster;
  private int     iPort;
  private string  sServer;
```

```
  private int     iLoad;
  private Date    dTimestamp;

//--------
// constructor

  public ADV_nte(string sClusterIn, int iPortIn, string sServerIn,
                 int iLoadIn) {
    sCluster = sClusterIn;
    iPort = iPortIn;
    sServer = sServerIn;
    iLoad = iLoadIn;
    dTimestamp = new Date();
  }

//--------
// comprueba si este elemento es actual o ha caducado
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval(); // establece tiempo de vida
                                                      // como 3 ciclos de asesor
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

    if (dNow.after(dExpires)) {
      bCurrent = false;
    } else {
      bCurrent = true;
    }
    return bCurrent;
  }

//--------
// objeto(s) usuario de valor

  public int getLoadValue() { return iLoad; }

//--------
// clona (evita la corrupción entre hebras)

  public synchronized Object Clone() {
    try {
      return super.clone();
    } catch (cloneNotSupportedException e) {
      return null;
    }
  }

}

//--------
// define el asesor personalizado

public class ADV_twop extends ADV_Base
   implements ADV_MethodInterface, ADV_AdvisorVersionInterface {

  static final int ADV_TWOP_PORT_HTTP = 80;
  static final int ADV_TWOP_PORT_SSL = 443;

  //--------
  // define las tablas para mantener información de historial específica de puertos

  static HashTable htTwopHTTP = new Hashtable();
  static HashTable htTwopSSL = new Hashtable();

  static final String ADV_TWOP_NAME = "twop";
  static final int ADV_TWOP_DEF_ADV_ON_PORT = 80;
```

```
  static final int ADV_TWOP_DEF_INTERVAL = 7;
  static final string ADV_HTTP_REQUEST_STRING =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_LB_Custom_Advisor\r\n\r\n";

  //--------
  // crea una matriz de bytes con el mensaje de saludo del cliente SSL

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
    (byte)0x01,               // saludo del cliente
    (byte)0x03, (byte)0x00,   // versión SSL
    (byte)0x00, (byte)0x03,   // longitud de especificación de cifrado (bytes)
    (byte)0x00, (byte)0x00,   // longitud de ID de sesión (bytes)
    (byte)0x00, (byte)0x10,   // longitud de datos de desafío (bytes)
    (byte)0x00, (byte)0x00, (byte)0x03,   // especificación de cifrado
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)0x20,  // datos de desafío
    (byte)0xFD, (byte)0x3A, (byte)0x3C, (byte)0x18,
    (byte)0xAB, (byte)0x67, (byte)0xB0, (byte)0x52,
    (byte)0xB1, (byte)0x1D, (byte)0x55, (byte)0x44, (byte)0x0D, (byte)0x0A };

  //--------
// constructor

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);    // false = Load Balancer temporiza la respuesta
    setAdvisor ( this );
  }

  //--------
// ADV_AdvisorInitialize

  public void ADV_AdvisorInitialize() {
    return;
  }

  //--------
  // rutinas de acceso PUT y GET sincronizadas para las tablas hash

  synchronized ADV_nte getNte(Hashtable ht, String sName, String sHashKey) {
    ADV_nte nte = (ADV_nte)(ht.get(sHashKey));
    if (null != nte) {
      nte = (ADV_nte)nte.clone();
    }
    return nte;
  }
 synchronized void putNte(Hashtable ht, String sName, String sHashKey,
                          ADV_nte nte) {
   ht.put(sHashKey,nte);
    return;
 }

  //--------
  // getLoadHTTP - determina la carga HTTP según la respuesta del servidor

  int getLoadHTTP(int iConnectTime, ADV_Thread caller) {
    int iLoad = ADV_HOST_INACCESSIBLE;

    int iRc = caller.send(ADV_HTTP_REQUEST_STRING);  // envía un mensaje de petición
                                                     // al servidor
    if (0 <= iRc) {       // ¿ha devuelto una anomalía la petición?
      StringBuffer sbReceiveData = new StringBuffer("")  // asigna un almacenamiento
                                                         // intermedio para
                                                         // la respuesta
      iRc = caller.receive(sbReceiveData);    // obtener respuesta del servidor
```

```
    if (0 <= iRc) {      // ¿ha devuelto una anomalía la recepción?
      if (0 < sbReceiveData.length()) {   // ¿hay datos?
        iLoad = SUCCESS;        // ignora los datos recuperados y
                                // devuelve un código de ejecución satisfactoria
      }
    }
  }
  return iLoad;
}

//--------
// getLoadSSL() - determina la carga de SSL según la respuesta del servidor

int getLoadSSL(int iConnectTime, ASV_Thread caller) {
  int iLoad = ADV_HOST_INACCESSIBLE;

  int iSocket = caller.getAdvisorSocket();   // envía una petición hexadecimal
                                             // al servidor
  CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                               abClientHello);
  int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

  if (0 <= iRc) {       // ¿ha devuelto una anomalía la petición?
    StringBuffer sbReceiveData = new StringBuffer(""); // asigna un almacenamiento
                                                       // intermedio para
                                                       // la respuesta
    iRc = caller.receive(sbReceiveData);     // obtiene una respuesta del
                                            // servidor
    if (0 <= iRc) {      // ¿ha devuelto una anomalía la recepción?
      if (0 < sbReceiveData.length()) {   // ¿hay datos?
        iLoad = SUCCESS;   // ignora los datos recuperados y devuelve un código
                           // de ejecución satisfactoria
      }
    }
  }
  return iLoad;
}

//--------
// getLoad - fusiona los resultados de los métodos HTTP y SSL

public int getLoad(int iConnectTime, ADV_Thread caller) {
  int iLoadHTTP;
  int iLoadSSL;
  int iLoad;
  int iRc;

  String sCluster = caller.getCurrentCluster();   // dirección de clúster actual
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;     // clave de tabla hash

  if (ADV_TWOP_PORT_HTTP == iPort) {              // gestiona un servidor HTTP
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // obtiene la carga de HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);  // guarda la información
                                                    // de carga HTTP
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // obtiene información
                                                          // de SSL
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) {         // comprueba la indicación
                                                    // de la hora
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // ¿funciona
                                                                 // SSL?
      iLoad = iLoadHTTP;
```

```
        } else {    // SSL no funciona, por lo que marca el servidor HTTP como
                    // sin funcionamiento
          iLoad= ADV_HOST_INACCESSIBLE;
        }
      } else {      // la información de SSL ha caducado, por lo que marca el
                    // servidor HTTP como sin funcionamiento
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {        // no hay información de carga acerca de SSL, se informa de
                    // los resultados de getLoadHTTP()
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {           // gestiona un servidor SSL
    iLoadSSL = getLoadSSL(iConnectTime, caller);   // obtiene la carga de SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL);   // guarda la información de
                                                  // carga de SSL.

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey); // obtiene información
                                                           // de HTTP
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) {       // comprueba la indicación
                                                   // de la hora
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {  // ¿funciona
                                                                // HTTP?
      iLoad = iLoadSSL;
        } else {   // el servidor HTTP no funciona, por lo que marca SSL como
                   // sin funcionamiento
      iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {     // información caducada de HTTP, por lo que marca SSL como
                   // sin funcionamiento
      iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {       // no hay información de carga acerca de HTTP, se informa de
                   // los resultados de getLoadSSL()
      iLoad = iLoadSSL;
    }
  }

  //--------
  // manejador de errores

    else {
      iLoad = ADV_HOST_INACCESSIBLE;
    }
    return iLoad;
  }
}
```

Asesor de WebSphere Application Server

Se incluye un asesor personalizado de ejemplo para WebSphere Application Server en el directorio *vía_acceso_instalación*/servers/samples/CustomAdvisors/. En este documento no se duplica el código completo.

- ADV_was.java es el archivo de código fuente del asesor que se compila y ejecuta en la máquina de Load Balancer.
- LBAdvisor.java.servlet es el código fuente de servlet cuyo nombre debe cambiarse por LBAdvisor.java, compilado, y ejecutarse en la máquina de WebSphere Application Server.

El asesor completo sólo es un poco más complejo que el del ejemplo. Añade una rutina de análisis especializada que es más compacta que la del ejemplo de StringTokenizer mostrado anteriormente.

La parte más compleja del código de ejemplo está en el servlet Java. Entre otros métodos, el servlet contiene dos métodos que exige la especificación de servlet, init() y service(), y un método run(), que exige la clase Java.lang.thread.

- El motor de servlets llama a init() una vez en el momento de la inicialización. Este método crea una hebra denominada _checker que se ejecuta de forma independiente de las llamadas del asesor y que queda inactiva durante un período de tiempo antes de reanudar el bucle de proceso.
- El motor de servlets llama a service() cada vez que se invoca el servlet. En tal caso, el asesor llama al método. El método service() envía una secuencia de caracteres ASCII a una secuencia de salida.
- run() contiene el núcleo de la ejecución de código. Se llama desde el método start() que, a su vez, se llama desde el método init().

Los fragmentos relevantes del código del servlet aparecen a continuación.

```
...

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
    ...
    _checker = new Thread(this);
    _checker.start();
  }

  public void run() {
    setStatus(GOOD);

    while (true) {
      if (!getKeepRunning())
    return;
      setStatus(figureLoad());
      setLastUpdate(new java.util.Date());

    try {
        _checker.sleep(_interval * 1000);
      } catch (Exception ignore) { ; }
    }
  }

  public void service(HttpServletRequest req, HttpServletResponse res)
                      throws ServletException, IOException {

    ServletOutputStream out = null;
    try {
      out = res.getOutputStream();
    } catch (Exception e) { ... }
    ...
    res.setContentType("text/x-application-LBAdvisor");
    out.println(getStatusString());
    out.println(getLastUpdate().toString());
    out.flush();
    return;
  }

  ...
```

Utilización de datos devueltos por asesores

Tanto si utiliza una llamada estándar a una parte existente del servidor de aplicaciones como si añade una nueva pieza de código al equivalente del asesor personalizado del lado del servidor, es posible que desee examinar los valores de carga devueltos y cambiar el funcionamiento del servidor. La clase Java StringTokenizer y sus métodos asociados hacen que esta investigación sea sencilla.

El contenido de un mandato HTTP típico puede ser `GET /index.html HTTP/1.0`.

Una respuesta típica a este mandato puede ser la siguiente:

```
HTTP/1.1 200 OK
Date: Mon, 20 November 2000 14:09:57 GMT
Server: Apache/1.3.12 (Linux and UNIX)
Content-Location: index.html.en
Vary: negotiate
TCN: choice
Last-Modified: Fri, 20 Oct 2000 15:58:35 GMT
ETag: "14f3e5-1a8-39f06bab;39f06a02"
Accept-Ranges: bytes
Content-Length: 424
Connection: close
Content-Type: text/html
Content-Language: en

<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 3.2 Final//EN">
<HTML><HEAD><TITLE>Test Page</TITLE></HEAD>
<BODY><H1>Apache server</H1>
<HR>
<P><P>This Web server is running Apache 1.3.12.
<P><HR>
<P><IMG SRC="apache_pb.gif" ALT="">
</BODY></HTML>
```

Los elementos de interés están contenidos en la primera línea, específicamente el código de retorno HTTP.

La especificación HTTP clasifica los códigos de retornos que se pueden resumir de la manera siguiente:

- Los códigos de retorno 2xx son ejecuciones satisfactorias.
- Los códigos de retorno 3xx son redirecciones.
- Los códigos de retorno 4xx son errores de cliente.
- Los códigos de retorno 5xx son errores de servidor.

Si debe saber exactamente los códigos que el servidor puede devolver, puede que el código no tenga que ser tan detallado como en este ejemplo. No obstante, recuerde que al limitar los códigos de retorno que detecte también puede limitar la futura flexibilidad del programa.

El ejemplo siguiente es un programa Java autónomo que contiene un cliente HTTP mínimo. El ejemplo invoca un analizador simple general para examinar respuestas HTTP.

```
import java.io.*;
import java.util.*;
import java.net.*;

public class ParseTest {
  static final int iPort = 80;
  static final String sServer = "www.ibm.com";
  static final String sQuery = "GET /index.html HTTP/1.0\r\n\r\n";
  static final String sHTTP10 = "HTTP/1.0";
  static final String sHTTP11 = "HTTP/1.1";

  public static void main(String[] Arg) {
    String sHTTPVersion = null;
    String sHTTPReturnCode = null;
    String sResponse = null;
    int iRc = 0;
    BufferedReader brIn = null;
    PrintWriter psOut = null;
    Socket soServer= null;
    StringBuffer sbText = new StringBuffer(40);

    try {
      soServer = new Socket(sServer, iPort);
      brIn = new BufferedReader(new InputStreamReader(
                                  soServer.getInputStream()));
      psOut = new PrintWriter(soServer.getOutputStream());
      psOut.println(sQuery);
      psOut.flush();
      sResponse = brIn.readLine();
    try {
        soServer.close();
      } catch (Exception sc) {;}
    }  catch (Exception swr) {;}

    StringTokenizer st = new StringTokenizer(sResponse, " ");
    if (true == st.hasMoreTokens()) {
      sHTTPVersion = st.nextToken();
      if (sHTTPVersion.equals(sHTTP110) || sHTTPVersion.equals(sHTTP11)) {
        System.out.println("HTTP Version: " + sHTTPVersion);
    } else {
        System.out.println("Invalid HTTP Version: " + sHTTPVersion);
      }
    } else {
      System.out.println("Nothing was returned");
    return;
    }

    if (true == st.hasMoreTokens()) {
      sHTTPReturnCode = st.nextToken();
    try {
        iRc = Integer.parseInt(sHTTPReturnCode);
      } catch (NumberFormatException ne) {;}

      switch (iRc) {
      case(200):
        System.out.println("HTTP Response code: OK, " + iRc);
        break;
      case(400): case(401): case(402): case(403): case(404):
        System.out.println("HTTP Response code: Client Error, " + iRc);
        break;
      case(500): case(501): case(502): case(503):
        System.out.println("HTTP Response code: Server Error, " + iRc);
        break;
```

```
      default:
        System.out.println("HTTP Response code: Unknown, " + iRc);
        break;
      }
    }

    if (true == st.hasMoreTokens()) {
      while (true == st.hasMoreTokens()) {
        sbText.append(st.nextToken());
        sbText.append("  ");
        }
      System.out.println("HTTP Response phrase: " + sbText.toString());
    }
  }
}
```

Avisos

Tercera edición (junio de 2005)

Esta información se ha desarrollado para productos y servicios proporcionados en los Estados Unidos.

Puede que IBM no proporcione los productos, servicios o funciones tratados en este manual en otros países. Consulte al representante de IBM de su localidad para obtener información acerca de los productos y servicios que están actualmente disponibles en su localidad. Cualquier referencia a un producto, programa o servicio de IBM no pretende indicar ni implica que sólo se pueda utilizar este producto, programa o servicio de IBM. En su lugar, se puede utilizar cualquier producto, programa o servicio funcionalmente equivalente que no vulnere ningún derecho de propiedad intelectual de IBM. Sin embargo, es responsabilidad del usuario evaluar y verificar el funcionamiento de cualquier producto, programa o servicio que no sea de IBM.

IBM puede tener patentes o aplicaciones pendientes de patente que conciernan al tema descrito en este documento. El suministro de este documento no le da ninguna licencia sobre estas patentes. Puede enviar preguntas acerca de licencias por escrito a:

IBM Corporation
Attn.: G71A/503
P.O. box 12195
3039 Cornwallis Rd.
Research Triangle Park, N.C. 27709-2195
Estados Unidos

Para preguntas acerca de licencias referentes a información de doble byte (DBCS), póngase en contacto con el Departamento de propiedad intelectual de IBM de su país o envíe sus preguntas por escrito a:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japón

El siguiente párrafo no se aplica al Reino Unido ni a ningún otro país donde estas disposiciones no coincidan con la legislación local:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROPORCIONA ESTE DOCUMENTO “TAL CUAL” SIN GARANTÍA DE NINGÚN TIPO,YA SEA EXPLÍCITA O IMPLÍCITA, INCLUYENDO, PERO SIN LIMITARSE A, LAS GARANTÍAS O CONDICIONES IMPLÍCITAS DE NO VULNERACIÓN, COMERCIALIZACIÓN O IDONEIDAD PARA UNA FINALIDAD DETERMINADA. Algunas legislaciones no contemplan la exclusión de garantías, explícitas o implícitas en algunas transacciones, por lo que puede haber usuarios a los que no les afecte dicha regla.

Esta publicación puede contener imprecisiones técnicas o errores tipográficos. La información que ofrece está sometida a modificaciones periódicas, las cuales se van

incorporando en ediciones posteriores. IBM se reserva el derecho de realizar mejoras y/o cambios en los productos o programas descritos en esta publicación en cualquier momento sin previo aviso.

Cualquier referencia en esta información a sitios Web que no son de IBM se proporciona solamente para su comodidad y no equivale de ninguna manera a una aprobación de esos sitios Web. Los materiales de esos sitios Web no forman parte de los materiales de este producto de IBM y la utilización de esos sitios Web se realiza bajo el propio riesgo del usuario.

IBM puede utilizar o distribuir cualquier información que el usuario le proporcione de la manera que considere adecuada sin incurrir en ninguna obligación con el usuario.

Los usuarios autorizados de este programa que deseen tener información sobre el mismo con el propósito de posibilitar: (i) el intercambio de información entre programas creados independientemente y otros programas (incluyendo éste) y (ii) la utilización mutua de la información que se ha intercambiado, deben ponerse en contacto con:

IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222-9183
Estados Unidos

Esta información puede estar disponible, bajo las condiciones y los términos adecuados, incluyendo en algunos casos, el pago de una cuota.

El programa con licencia descrito en este documento y todos los materiales con licencia disponibles para el mismo son proporcionados por IBM bajo los términos del IBM International Program License Agreement o cualquier acuerdo equivalente entre nosotros.

Cualquier información de rendimiento contenida aquí fue determinada en un entorno controlado. Por tanto, los resultados obtenidos en otros entornos operativos pueden variar de forma significativa. Pueden haberse realizado algunas mediciones en sistemas en nivel de desarrollo y no existen garantías de que estas mediciones sean las mismas en sistemas disponibles para todos los usuarios. Además, algunas mediciones pueden haberse calculado mediante extrapolaciones. Los resultados reales pueden variar. Los usuarios de este documento deben verificar los datos aplicables para su entorno específico.

La información referente a productos que no son de IBM se ha obtenido de los suministradores de estos productos, sus anuncios publicados u otras fuentes disponibles para el público. IBM no ha probado estos productos y no puede confirmar la precisión del rendimiento, compatibilidad y otras afirmaciones relacionadas con productos que no son de IBM. Las preguntas acerca de las posibilidades de productos que no son de IBM deben dirigirse a los suministradores de estos productos.

Todas las declaraciones referentes a acciones e intenciones futuras de IBM pueden cambiar o ser retiradas sin aviso previo y solamente representan objetivos.

Esta información contiene ejemplos de datos e informes utilizados en operaciones diarias de negocios. Para ilustrarlos de la manera más completa posible, los ejemplos pueden incluir nombres de personas, compañías, marcas y productos.

Todos estos nombres son ficticios y cualquier parecido con nombres y direcciones utilizadas por una empresa de negocios real es mera coincidencia.

Si accede a esta información mediante una presentación visual, las fotografías e ilustraciones en color no aparecerán.

Marcas registradas

Los siguientes términos son marcas registradas IBM Corporation en Estados Unidos, otros países o en ambos:

- AIX
- IBM
- ViaVoice
- WebSphere

Java y todas las marcas registradas y logotipos basados en Java son marcas registradas de Sun Microsystems, Inc. en Estados Unidos, en otros países o en ambos.

Microsoft, Windows, Windows NT y el logotipo de Windows son marcas registradas de Microsoft Corporation en Estados Unidos, en otros países o en ambos.

Intel, Intel Inside (logos), MMX y Pentium son marcas registradas de Intel Corporation en Estados Unidos, en otros países o en ambos.

UNIX es una marca registrada de The Open Group en Estados Unidos y en otros países.

Linux es una marca registrada de Linus Torvalds en Estados Unidos, en otros países o en ambos.

Otros nombres de empresas, productos o servicios pueden ser marcas registradas o marcas de servicio de terceros.

Índice

L

M

N

O

P

R

S

T

V

W

Printed in Denmark by IBM Danmark A/S

GC11-3032-02

Spine information:

WebSphere Application Server Guía de programación de Edge Components Versión 6.0.2 GC11-3032-02