WebSphere Application Server

Guida alla programmazione per Edge Components

Versione 6.1

GC13-3368-02

WebSphere Application Server

Guida alla programmazione per Edge Components

Versione 6.1

GC13-3368-02

Nota

Prima di usare questo prodotto e le relative informazioni, leggere le informazioni contenute nella sezione “Informazioni particolari” a pagina 61.

Prima edizione (Maggio 2006)

Questa edizione si applica a:
WebSphere Application Server, Versione 6.1

e a tutte le release e modifiche successive, finché non verrà diversamente indicato nelle nuove edizioni.

Ordinare le pubblicazioni mediante il rappresentante IBM o gli uffici IBM del proprio paese.

Indice

Figure

Informazioni su questa guida

In questa sezione vengono descritti lo scopo, l'organizzazione e le convenzioni di questo documento, la *Guida alla programmazione di WebSphere Application Server per Edge Components*.

A chi è rivolta questa guida

In questa guida vengono analizzate le API (application programming interface) disponibili per la personalizzazione di Edge Components per WebSphere Application Server, Versione 6.1. Le informazioni sono rivolte ai programmatori che devono scrivere applicazioni plugin e che devono eseguire altre operazioni di personalizzazione. Anche i progettisti di rete e gli amministratori di sistema possono consultare queste informazioni per conoscere i tipi di personalizzazione possibili.

Competenze prerequisite

L'uso delle informazioni contenute in questa guida richiede la conoscenza delle procedure di programmazione tramite l'uso dei linguaggi di programmazione Java o C, a seconda dell'API che si intende utilizzare. Sono documentati sia i metodi che le strutture disponibili in ciascuna interfaccia esposta, ma il programmatore deve sapere come costruire la propria applicazione, come compilarla per il proprio sistema e, infine, come sottoporla a test. Il codice di esempio viene fornito per alcune interfacce, ma gli esempi sono forniti solo come esempi di creazione della propria applicazione.

Convenzioni e terminologia utilizzati in questa guida

Questa documentazione utilizza le seguenti convenzioni tipografiche e di definizione dei tasti.

Tabella 1. Convenzioni utilizzate in questa guida

Convenzione	Significato
Grassetto	Quando si fa riferimento alle interfacce utente grafiche (GUI, Graphical User Interfaces), il grassetto evidenzia menu, voci di menu, etichette, pulsanti, icone e cartelle. Inoltre, può essere utilizzato per enfatizzare i nomi di comandi che, altrimenti, verrebbero confusi con il testo circostante.
`A spaziatura fissa`	Indica il testo da inserire davanti a un prompt di comandi. Inoltre, indica il testo su video, gli esempi di codice ed estratti di file.
Corsivo	Indica i valori delle variabili che l'utente deve inserire (ad esempio, il nome per sostituire *nomeFile* con il nome effettivo di un file). Il corsivo viene inoltre utilizzato per enfatizzare un concetto ed evidenziare i titoli di manuali.
Ctrl-*x*	Dove *x* è il nome di un tasto, indica una sequenza di caratteri di controllo. Ad esempio, Ctrl-c indica: tenere premuto il tasto Ctrl e contemporaneamente premere il tasto c.
Return	Indica il tasto etichettato con la parola Invio, Enter, Return o con una freccia verso sinistra.
%	Rappresenta il prompt della shell dei comandi in ambiente Linux e UNIX per un comando che non richiede privilegi root.

Tabella 1. Convenzioni utilizzate in questa guida (Continua)

Convenzione	Significato
#	Rappresenta il prompt della shell dei comandi in ambiente Linux e UNIX per un comando che richiede i privilegi root.
C:\	Rappresenta il prompt dei comandi in ambiente Windows.
Immissione di comandi	Quando si invita l'utente a "immettere" o "inserire" un comando, digitare il comando e premere Invio. Ad esempio, l'istruzione "Immettere il comando **ls**" indica: digitare **ls** al prompt dei comandi e premere Invio.
[ ]	Racchiude le voci facoltative nelle descrizioni della sintassi.
{ }	Racchiude gli elenchi da cui è necessario scegliere una voce nelle descrizioni della sintassi.
|	Separa le voci in un elenco di opzioni racchiuse tra parentesi { } nelle descrizioni della sintassi.
...	I puntini di sospensione nelle descrizioni della sintassi indicano che è possibile ripetere la voce precedente una o più volte. Negli esempi, indicano che le informazioni sono state omesse dall'esempio per motivi di brevità.

Accessibilità

Le funzioni di accessibilità consentono ad un utente con un svantaggi fisici, quali una mobilità o una vista limitata, di utilizzare agevolmente prodotti software. Queste sono le principali funzioni di accessibilità in WebSphere Application Server, Versione 6.1:

- È possibile utilizzare un software di lettura dello schermo e un sintetizzatore vocale digitale per ascoltare ciò che viene visualizzato sullo schermo. Inoltre, si può adoperare un software di riconoscimento vocale, quale IBM ViaVoice, per immettere dati e spostarsi all'interno dell'interfaccia utente.
- Le funzioni possono essere utilizzate tramite la tastiera invece che tramite il mouse.
- È possibile configurare e gestire le funzioni di Application Server utilizzando editor di testo standard e interfacce della riga comandi invece delle interfacce grafiche fornite. Per ulteriori informazioni sull'accessibilità di particolari funzioni, fare riferimento alla documentazione corrispondente.

Documenti correlati e siti Web

- *Informazioni di base, pianificazione e installazione per Edge Components*, GC13-3367-02
- *Guida alla gestione per Caching Proxy*, GC13-3366-02
- *Guida alla gestione*, GC13-3365-02
- Sito Web IBM *www.ibm.com/*
- IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/*
- Sito Web libreria IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/library.html*
- Sito Web supporto IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/support.html*
- Centro informazioni IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/infocenter.html*
- Centro informazioni IBM per Edge Components IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

Come inviare i propri commenti

I vostri commenti risultano di estrema importanza poiché consentono di fornire informazioni della massima accuratezza e qualità. In caso di commenti su questa guida o su altra documentazione relativa a Edge Components di WebSphere Application Server:

- Inviare i commenti per e-mail a fsdoc@us.ibm.com. Accertarsi di includere il nome della guida, il numero di sezione, la versione di WebSphere Application Server e, se il caso, il punto specifico del testo che si sta commentando, quale un numero di pagina o un numero di tabella.

Capitolo 1. Panoramica per la personalizzazione di Edge Components

In questa guida vengono analizzate le API (application programming interfaces) fornite per Edge Components di WebSphere Application Server. (Edge Components di WebSphere Application Server include Caching Proxy e Load Balancer.) Vengono fornite diverse interfacce che consentono agli amministratori di personalizzare le loro installazioni, di cambiare il modo in cui gli Edge Components interagiscono tra loro o di abilitare l'interazione con altri sistemi software.

IMPORTANTE: Caching Proxy è disponibile su tutte le installazioni di Edge Component, con le seguenti eccezioni:

- Caching Proxy non è disponibile per le installazioni di Edge Component su macchine con microprocessore Itanium 2 o AMD Opteron a 64-bit.
- Caching Proxy non è disponibile per installazioni di Edge Component di Load Balancer for IPv4 and IPv6.

Le API contenute in questo documento indicano diverse categorie.

Personalizzazione di Caching Proxy

Caching Proxy dispone di diverse interfacce scritte nella propria sequenza di elaborazione, dove l'elaborazione personalizzata può essere aggiunta o sostituita da una standard. Le personalizzazioni che possono essere eseguite includono l'alterazione o l'aumento di attività, quali:

- Autenticazione client
- Autorizzazione richieste
- Conversione di URL in percorsi di file fisici
- Supporto di richieste
- Registrazione
- Risposta alle condizioni di errore

I programmi applicativi personalizzati, noti anche come plugin di Caching Proxy, vengono chiamati in punti predeterminati nella sequenza di elaborazione del server proxy.

L'API Caching Proxy è stata utilizzata per implementare determinate funzioni di sistema. Ad esempio, il supporto LDAP del server proxy viene implementato come un plugin.

Capitolo 2, "API Caching Proxy", a pagina 3 descrive nei dettagli l'interfaccia e include le fasi di configurazione del server proxy per utilizzare i programmi del plugin.

Personalizzazione di Load Balancer

Load Balancer può essere personalizzato scrivendo i propri advisor. Gli advisor eseguono la misurazione del carico effettivo sui server. Con un advisor personalizzato, è possibile utilizzare un metodo fornito dall'utente ed è rilevante per la misurazione del carico da parte del sistema. Risulta particolarmente importante se si dispone di sistemi del server Web personalizzati o proprietari.

Capitolo 3, “Advisor personalizzati”, a pagina 41 fornisce informazioni dettagliate sulla scrittura e l’utilizzo degli advisor personalizzati. Include il codice di esempio dell’advisor.

Ubicazione del codice di esempio

Il codice di esempio per queste API è incluso nel CD-ROM di Edge Components nella directory esempi. Ulteriori esempi di codice sono disponibili sul sito web di WebSphere Application Server, www.ibm.com/software/webservers/appserv/

Capitolo 2. API Caching Proxy

In questa sezione si analizza l'API (application programming interface) Caching Proxy: di cosa si tratta, perché è utile e come funziona.

IMPORTANTE: Caching Proxy è disponibile su tutte le installazioni di Edge Component, con le seguenti eccezioni:

- Caching Proxy non è disponibile per le installazioni di Edge Component su macchine con microprocessore Itanium 2 o AMD Opteron a 64-bit.
- Caching Proxy non è disponibile per installazioni di Edge Component di Load Balancer for IPv4 and IPv6.

Panoramica sulle API Caching Proxy

L'API è un'interfaccia del Caching Proxy che consente di estendere le funzioni di base del server proxy. Per effettuare un'elaborazione personalizzata, è possibile scrivere estensioni o plugin, che includono i seguenti esempi:

- Potenziamento della routine di autenticazione di base o relativa sostituzione con un processo specifico del sito.
- Aggiunta della routine di gestione degli errori per tracciare i problemi o avvisare in caso di errori gravi.
- Informazioni sulla rilevazione e sulla traccia provenienti dal client richiedente, ad esempio riferimenti al server e codici agente utente.

L'API Caching Proxy fornisce i seguenti vantaggi:

- **Efficienza**
  - L'API è stata progettata specificatamente per il sistema di elaborazione basato su thread utilizzato dal Caching Proxy.
- **Flessibilità**
  - L'API contiene funzioni ricche e versatili.
  - L'API è indipendente dalla piattaforma ed è di tipo non linguistico. Infatti, viene eseguita su tutte le piattaforme del Caching Proxy e le applicazioni plugin possono essere scritte nella maggior parte dei linguaggi di programmazione supportati da tali piattaforme.
- **Facilità di utilizzo**
  - I tipi di dati semplici vengono inoltrati in base al riferimento piuttosto che in base al valore (ad esempio, long *, char *).
  - Ciascuna funzione ha un numero fisso di parametri.
  - Sono inclusi i collegamenti al linguaggio C.
  - I plugin non hanno alcun impatto sulla memoria assegnata; le applicazioni di plugin assegnano la memoria disponibile a prescindere dagli altri processi del Caching Proxy.

Procedura generale per la scrittura dei programmi API

Prima di scrivere i programmi plugin del Caching Proxy, è necessario essere a conoscenza delle modalità di funzionamento del server proxy. Tale funzionamento del server proxy può essere suddiviso in diverse fasi di elaborazione distinte. Per ciascuna di queste fasi è possibile fornire le proprie funzioni personalizzate

mediante l'utilizzo delle API. Ad esempio, si desidera effettuare un'operazione dopo aver letto una richiesta client e prima di eseguire un'altra elaborazione? Oppure si desidera eseguire routine speciali durante l'autenticazione e dopo l'invio del file richiesto?

Con l'API viene fornita una libreria di funzioni specifiche. I programmi di plugin possono chiamare le funzioni API predefinite affinché interagiscano con il processo del server proxy (ad esempio, per gestire le richieste, per leggere e scrivere le intestazioni della richiesta o per scrivere sui log del server proxy). Queste funzioni non devono essere confuse con quelle del plugin, scritte dall'utente, che vengono chiamate dal server proxy. Le funzioni predefinite vengono descritte in "Funzioni e macro predefinite" a pagina 17.

L'utente indica al server proxy di chiamare le funzioni di plugin nelle fasi appropriate utilizzando le direttive API Caching Proxy corrispondenti nel file di configurazione del server. Queste direttive vengono descritte in "Direttive di configurazione del Caching Proxy per le fasi dell'API" a pagina 23.

Questo documento include quanto segue:

- Una spiegazione basilare delle fasi del Caching Proxy che possono essere personalizzare (consultare "Fasi del processo server")
- Linee guida per la scrittura dei plugin (consultare "Linee guida" a pagina 8)
- Prototipi per le funzioni personalizzate che è possibile scrivere per ogni operazione eseguita dal server e i relativi codici di ritorno (consultare "Prototipi della funzione del plugin" a pagina 10)
- Definizioni delle macro e delle funzioni predefinite che è possibile richiamare dai plugin e i relativi codici di ritorno (consultare "Funzioni e macro predefinite" a pagina 17)
- Direttive di configurazione delle API di Caching Proxy (consultare "Direttive di configurazione del Caching Proxy per le fasi dell'API" a pagina 23)

Questi componenti e queste procedure possono essere utilizzati per scrivere i propri programmi di plugin del Caching Proxy.

Fasi del processo server

L'operazione di base del server proxy può essere suddivisa in fasi in base al tipo di elaborazione eseguita dal server durante quella fase specifica. Ogni fase include un punto di collegamento in cui è possibile eseguire una parte specifica del proprio programma. Aggiungendo le direttive API al file di configurazione del Caching Proxy (ibmproxy.conf), si indica la funzione di plugin che si desidera venga chiamata durante una particolare operazione. È possibile chiamare diverse funzioni di plugin durante una fase particolare del processo, includendo più di una direttiva per quella fase.

Alcune operazioni fanno parte del processo di richiesta del server. In altri termini, il server proxy esegue queste operazioni ogni volta che elabora una richiesta. Vengono poi eseguite altre operazioni indipendentemente dall'elaborazione della richiesta; vale a dire, il server esegue queste operazioni a prescindere dal fatto che ci sia o no una richiesta in fase di elaborazione.

Il programma compilato si trova in un oggetto condiviso, ad esempio, un file DLL o .so, a seconda del sistema operativo. Man mano che il server procede con le operazioni del processo di richiesta, chiama le funzioni plugin associate a ciascuna operazione fino a quando una delle funzioni non indica che la richiesta è stata

gestita. Se si specifica più di una funzione di plugin per una fase particolare, tali funzioni vengono definite nell'ordine in cui le relative direttive vengono visualizzate nel file di configurazione.

Se la richiesta non viene gestita da una funzione di plugin (o l'utente non ha incluso una direttiva API del Caching Proxy per quella fase o la funzione di plugin per quella fase ha restituito HTTP_NOACTION), il server esegue l'operazione predefinita.

Nota: ciò è valido per tutte le fasi, tranne per quella relativa al servizio; questa fase non prevede alcuna operazione predefinita.

Figura 1 a pagina 6 illustra le fasi del processo del server proxy e definisce l'ordine di elaborazione delle fasi correlate all'elaborazione della richiesta.

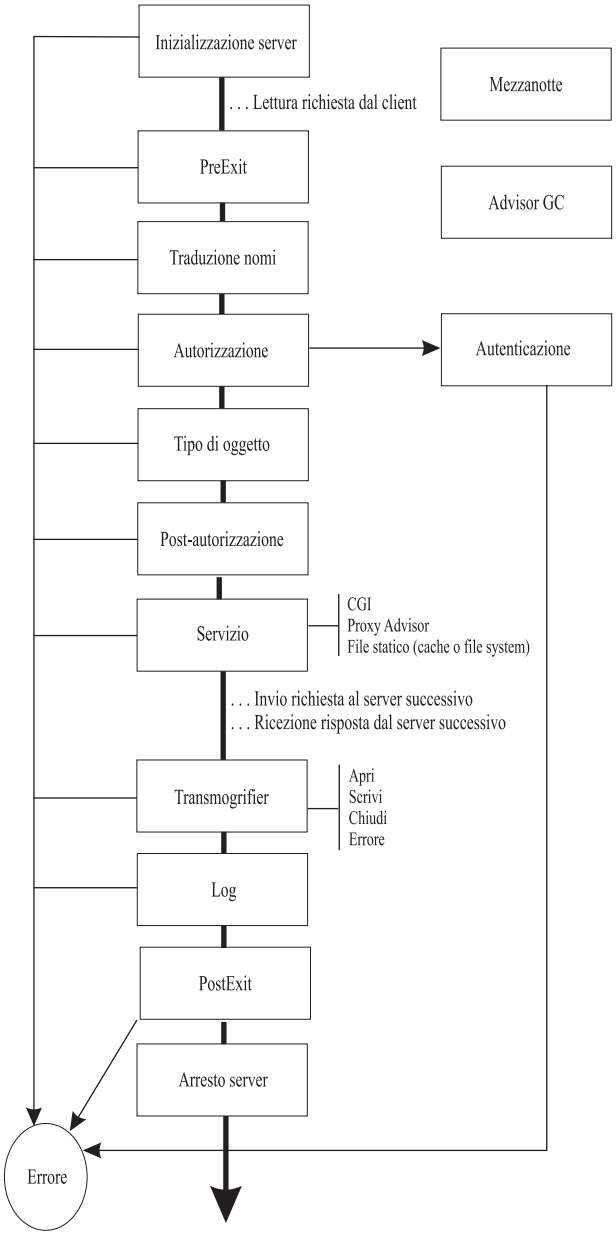

Figura 1. Grafico delle operazioni nel processo del server proxy

Quattro delle fasi presenti nel diagramma vengono eseguite indipendentemente dall'elaborazione di qualsiasi richiesta client. Tali fasi sono correlate all'esecuzione e alla gestione del server proxy. Esse includono quanto segue:

- Inizializzazione del server
- Mezzanotte
- Advisor GC
- Chiusura del server

L'elenco riportato di seguito spiega lo scopo di ciascuna fase illustrata nella Figura 1. Non tutte le fasi verranno, comunque, chiamate per una particolare richiesta.

Inizializzazione del server
Esegue l'inizializzazione all'avvio del server proxy e prima che qualsiasi richiesta client venga accettata.

Mezzanotte
Esegue un plugin a mezzanotte, senza nessun contesto di richiesta. Questa fase viene illustrata separatamente nel diagramma poiché non fa parte del processo di richiesta; in altri termini, la sua esecuzione è indipendente da qualsiasi richiesta.

Advisor GC
Influenza le decisioni relative alla raccolta di dati inutili per i file presenti nella cache. Questa fase viene illustrata separatamente nel diagramma poiché non fa parte del processo di richiesta; in altri termini, la sua esecuzione è indipendente da qualsiasi richiesta. La raccolta di dati inutili viene effettuata quando la dimensione della cache raggiunge il valore massimo. (Le informazioni relative alla configurazione della raccolta di dati inutili della cache sono incluse nella *Guida alla gestione per WebSphere Application Server Caching Proxy*.)

PreExit

Esegue l'elaborazione dopo la lettura di una richiesta ma prima di qualsiasi altra operazione.

Se si riceve un'indicazione che la richiesta è stata elaborata (HTTP_OK), il server ignora le altre fasi del processo di richiesta ed esegue solo le operazioni Transmogrifier, Log e PostExit.

Conversione nome
Converte il percorso virtuale (da una URL) in percorso fisico.

Autorizzazione

Utilizza i token di sicurezza memorizzati per controllare il percorso fisico delle protezioni, degli ACL e di altri controlli di accesso, nonché crea le intestazioni WWW-Authenticate necessarie per l'autenticazione di base. Se si scrive la propria funzione di plugin per sostituire questa fase, è necessario creare autonomamente queste intestazioni.

Per ulteriori informazioni, consultare "Autenticazione e autorizzazione" a pagina 36.

Autenticazione

Decodifica, verifica e memorizza i token di sicurezza.

Per ulteriori informazioni, consultare "Autenticazione e autorizzazione" a pagina 36.

Tipo di oggetto
Individua l'oggetto del file system indicato dal percorso.

Post autorizzazione

Esegue l'elaborazione dopo l'autorizzazione e l'ubicazione dell'oggetto, ma prima che la richiesta venga soddisfatta.

Se si riceve un'indicazione che la richiesta è stata elaborata (HTTP_OK), il server ignora le altre fasi del processo di richiesta ed esegue solo le operazioni Transmogrifier, Log e PostExit.

Servizio
Soddisfa la richiesta (inviando il file, eseguendo il CGI, ecc.)

Proxy Advisor
Influenza le decisioni relative al proxy e alla memorizzazione nella cache.

Transmogrifier
Fornisce l'accesso in scrittura alla parte di dati della risposta inviata al client.

Log Abilita la registrazione della transazione personalizzata.

Errore Abilita le risposte personalizzate alle condizioni di errore.

PostExit
Elimina le risorse assegnate per l'elaborazione della richiesta.

Chiusura del server
Esegue l'operazione di ripulitura quando si verifica una chiusura corretta.

Linee guida

- Scrivere il programma seguendo la sintassi e le linee guida fornite per le funzioni di plugin del server. Fornire a ciascuna funzione di plugin un nome funzione univoco e, se necessario, chiamare le funzioni predefinite del server.

  Sui sistemi AIX, è necessario un file di esportazione (ad esempio, libmyapp.exp) che elenchi le funzioni di plugin a cui è necessario collegare un file di importazione API del Caching Proxy, libhttpdapi.exp.

  Sui sistemi Linux, HP-UX e Solaris, è necessario effettuare un collegamento tra le librerie libhttpdapi e libc.

  Sui sistemi Windows, è necessario un file di definizione modulo (.def) che elenchi le funzioni di plugin a cui è necessario collegare HTTPDAPI.LIB.

  Accertarsi di includere HTAPI.h e di utilizzare la macro HTTPD_LINKAGE nelle proprie definizioni di funzione. Questa macro assicura che tutte le funzioni utilizzeranno le stesse convenzioni di chiamata.
- Il server è in esecuzione in un ambiente basato su più thread; pertanto, è necessario che i plugin facciano un uso limitato di thread. Se l'applicazione è rientrante, le prestazioni rimangono allo stesso livello.
- Effettuare le operazioni nei plugin in un ambito di thread. Non eseguire alcuna operazione nell'ambito di un processo, come ad esempio, chiusura, modifica dell'ID utente o registrazione di un handler del segnale.
- Non utilizzare le variabili globali oppure, nel caso fosse necessario utilizzarle, proteggerle con un semaforo ad esclusione reciproca.
- Ricordarsi di impostare l'intestazione Content-Type se si utilizza la funzione HTTPD_write() per inviare nuovamente i dati al client.
- Verificare sempre i codici di ritorno e, dove necessario, fornire l'elaborazione condizionale.
- Compilare e collegare il programma, facendo riferimento alla documentazione relativa al compilatore per creare un oggetto condiviso (ad esempio, un file DLL o .so), come richiesto per il sistema operativo.

  Utilizzare i seguenti comandi di compilazione e collegamento come linea guida.
  - **AIX**, utilizzando IBM CSet++
    - **Compilazione:**

      ```
      cc_r  -c -qdbxextra -qcpluscmt foo.c
      ```
    - **Collegamento:**

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

(Questo comando viene illustrato su due righe solo per facilitarne la lettura.)

– **HP-UX**, utilizzando HP C/ANSI C Developer's Bundle e HP aC++ Compiler
  - **Compilazione:**
    ```
    cc -Ae -c +Z +DAportable
    ```
  - **Collegamento:**
    ```
    aCC +Z -mt -c +DAportable
    ```
– **Linux**, utilizzando Gnu Compiler C (GCC) Version 3.2.X
  - **Compilazione:**
    ```
    gcc -c foo.c
    ```
  - **Collegamento:**
    ```
    ld -G -Bsymbolic -o libfoo.so foo.o -lhttpdapi -lc
    ```
– **Solaris**, utilizzando Sun Workshop
  - **Compilazione:**
    ```
    cc -mt -Bsymbolic -c foo.c
    ```
  - **Collegamento:**
    ```
    cc -mt -Bsymbolic -G -o libfoo.so foo.o -lhttpdapi -lc
    ```
– **Windows**, utilizzando Microsoft Visual C++
  - **Compilazione:**
    ```
    cl /c /MD /DWIN32 foo.c
    ```
  - **Collegamento:**
    ```
    link httpdapi.lib foo.obj /def:foo.def /out:foo.dll /dll
    ```

Per specificare le esportazioni, utilizzare uno dei seguenti metodi:

– Aggiungere le definizioni _declspec(dllexport) nell'origine.
– Specificare /EXPORT:entryname sulla riga comandi LIB.
– Creare un file di definizione modulo con un'istruzione EXPORTS.

- Aggiungere le direttive API del Caching Proxy al proprio file di configurazione per associare le funzioni di plugin del programma alle operazione appropriate. Nel processo di richiesta server ogni operazione dispone di una direttiva separata. Arrestare e riavviare il server affinché le nuove direttive siano convalidate.

  Nota: il Caching Proxy non scarica gli oggetti condivisi (i file DLL o .so), anche al riavvio. Per rilasciare gli oggetti condivisi, è necessario arrestare e riavviare il server.
- Verificare attentamente il programma prima di utilizzarlo in un ambiente di produzione. Poiché il Caching Proxy è un server basato su thread, è necessario che la verifica sia più rigorosa di quanto non lo sia per un server fork. Gli errori all'interno del programma possono causare l'interruzione del server proxy, in quanto quest'ultimo effettua una chiamata diretta al programma ed entrambi sono in esecuzione nello stesso spazio del processo.

Funzioni del plugin

Seguire la sintassi presentata in "Prototipi della funzione del plugin" a pagina 10 per scrivere le proprie funzioni del programma per le fasi di elaborazione della richiesta definite.

Ciascuna delle funzioni deve completare il parametro del codice di ritorno con un valore che indica l'azione eseguita:

- Il codice HTTP_NOACTION (valore 0) indica che non è stata eseguita nessuna operazione rilevante. Se viene restituito questo codice, il server proxy effettua l'azione predefinita per questa fase.
- Uno dei codici di ritorno HTTP validi indica che la funzione del plugin ha gestito l'operazione. (Consultare "Valori e codici di ritorno HTTP" a pagina 16 per un elenco di codici di ritorno validi.) Se viene fornito un codice di ritorno HTTP valido, non vengono chiamate altre funzioni del plugin per gestire quella fase della richiesta.

Prototipi della funzione del plugin

I prototipi della funzione per ogni fase del Caching Proxy mostrano il formato da utilizzare e illustrano il tipo di elaborazione da eseguire. Tenere presente che i nomi funzione non sono predefiniti. È necessario fornire nomi univoci alle funzioni utilizzando le proprie convenzioni di denominazione. Per rendere più semplice l'associazione, questo documento utilizza i nomi che si riferiscono alle fasi di elaborazione del server.

In ciascuna di queste funzioni del plugin, sono valide alcune funzioni API predefinite. Alcune delle funzioni predefinite non sono valide per tutte le fasi. Le funzioni API riportate di seguito sono valide se chiamate da tutte queste funzioni del plugin:

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- funzioni HTTPD_log*

Nelle descrizioni relative al prototipo della funzione, vi sono ulteriori funzioni API valide o non valide.

Il valore del parametro *handle* inviato alle funzioni può essere inoltrato come primo argomento per le funzioni predefinite. Le funzioni API predefinite sono descritte in "Funzioni e macro predefinite" a pagina 17.

Inizializzazione del server

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

A function defined for this step is called once when your module is loaded during server initialization. It is your opportunity to perform initialization before any requests have been accepted.

Although all server initialization functions are called, a error return code from a function in this step causes the server to ignore all other functions configured in the same module as the function that returned the error code. (That is, any other functions contained in the same shared object as the function that returned the error are not called.)

The version parameters contain the server proxy's version number; these are supplied by the Caching Proxy.

PreExit

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

A function defined for this step is called for each request after the request has been read but before any processing has occurred. A plug-in at this step can be used to access the client's request before it is processed by the Caching Proxy.

Valid return codes for the preExit function are the following:

- 0 (HTTP_NOACTION)
- 200 (HTTP_OK)
- HTTP errors in the 4*xx* or 5*xx* series (for example, 404, HTTP_NOT_FOUND)

Other return codes must not be used.

If this function returns HTTP_OK, the server proxy assumes that the request has been handled. All subsequent request processing steps are bypassed, and only the response steps (Transmogrifier, Log, and PostExit) are performed.

All predefined API functions are valid during this step.

Midnight

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

A function defined for this step runs daily at midnight and contains no request context. For example, it can be used to invoke a child process to analyze logs. (Note that extensive processing during this step can interfere with logging.)

Authentication

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

A function defined for this step is called for each request based on the request's authentication scheme. This function can be used to customize verification of the security tokens that are sent with a request.

Conversione nome

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

Per ogni richiesta viene chiamata una funzione definita per questa fase. Se si desidera che la funzione plugin venga chiamata solo per le richieste che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. La fase di Conversione nome si verifica prima che la richiesta venga elaborata e fornisce un meccanismo per la mappatura degli URL sugli oggetti, quali nomi di file.

Autorizzazione

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

Per ogni richiesta viene chiamata una funzione definita per questa fase. Se si desidera che la funzione plugin venga chiamata solo per le richieste che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. La fase di Autorizzazione si verifica prima che la richiesta venga elaborata e può essere utilizzata per verificare se l'oggetto identificato possa essere restituito al client. Se si effettua l'autenticazione di base, è necessario creare le intestazioni WWW-Authenticate.

Tipo di oggetto

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

Per ogni richiesta viene chiamata una funzione definita per questa fase. Se si desidera che la funzione plugin venga chiamata solo per le richieste che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. La fase Tipo di oggetto si verifica prima che la richiesta venga elaborata e può essere utilizzata per controllare l'esistenza dell'oggetto e per immettere l'oggetto.

PostAuthorization

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

Una funzione definita per questa fase viene richiamata dopo che la richiesta è stata autorizzata, ma prima di qualsiasi altra elaborazione. Se questa funzione restituisce HTTP_OK, il server proxy presume che la richiesta sia stata gestita. Tutte le successive fasi di richiesta vengono ignorate e verranno eseguite solo le fasi del processo di risposta (Transmogrifier, Log e PostExit).

In questa fase, sono valide tutte le funzioni predefinite del server.

Servizio

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

Per ciascuna richiesta viene chiamata una funzione definita per questa fase. Se si desidera che la funzione del plugin venga chiamata solo per le richieste che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. La fase relativa al Servizio soddisfa la richiesta, nel caso non fosse stata soddisfatta nelle fasi PreExit o PostAuthorization.

In questa fase, sono valide tutte le funzioni predefinite del server.

Fare riferimento alla direttiva Enable nella *Guida alla gestione per WebSphere Application Server Caching Proxy* per informazioni sulla configurazione della funzione Servizio da eseguire in base al metodo HTTP piuttosto che in base all'URL.

Transmogrifier

Le funzioni chiamate in questa fase del processo possono essere utilizzate per filtrare i dati di risposta come un flusso. Quattro delle funzioni del plugin per questa fase vengono chiamate in sequenza e ciascuna di esse agisce come un segmento del pipe tramite cui fluiscono i dati. Vale a dire che per ogni risposta vengono chiamate le funzioni *open*, *write*, *close* e *error* nello stesso ordine in cui sono state citate. Ciascuna di esse, a sua volta, elabora lo stesso flusso di dati.

In questa fase, è necessario implementare le seguenti quattro funzioni. (Non è necessario che i nomi delle funzioni corrispondano a questi nomi.)

- **Open**

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

  La funzione open esegue qualsiasi inizializzazione (ad esempio l'assegnazione del buffer) necessaria per elaborare i dati di questo flusso. Qualsiasi codice di ritorno diverso da HTTP_OK causa l'interruzione dell'operazione di filtraggio (le funzioni write e close non vengono chiamate). La funzione può restituire un puntatore vuoto, in questo modo l'utente può assegnare spazio a una struttura e fare in modo che il puntatore gli venga restituito nel parametro *correlator* delle funzioni successive.

- **Write**

```
void  HTTPD_LINKAGE  writeFunction (
         unsigned char *handle,
         unsigned char *data,    /* dati di risposta inviati dal
                                    server di origine */
         unsigned long *length,  /* lunghezza dei dati di risposta */
         void *correlator,       /* puntatore restituito dalla
                                    funzione 'open' */
         long *return_code
         )
```

  La funzione write elabora i dati e può richiamare la funzione HTTPD_write() predefinita del server con i dati nuovi o modificati. Il plugin non tenta di liberare il buffer inviato al server o presuppone che il server liberi il buffer ricevuto.

  Se, nell'ambito della funzione write, si decide di non modificare i dati, risulta ancora necessario chiamare la funzione HTTPD_write() nell'ambito della funzione open, write o close per inviare i dati della risposta al client. L'argomento *correlator* è il puntatore al buffer di dati restituito nella routine *open*.

- **Close**

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

  La funzione close effettua tutte le operazioni di ripulitura (ad esempio, svuotare e liberare il buffer del correlatore) necessarie per completare

l'elaborazione dei dati in questo flusso. L'argomento *correlator* è il puntatore al buffer di dati restituito nella routine *open*.

- **Error**

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

La funzione error abilita le prestazioni delle azioni di ripulitura, ad esempio lo svuotamento o la liberazione dei dati memorizzati nel buffer (o entrambe) prima che venga inviata una pagina di errore. A questo punto, vengono chiamate le funzioni open, write e close per elaborare la pagina di errore. L'argomento *correlator* è il puntatore al buffer di dati restituito nella routine *open*.

Note:

- durante la scrittura di un plugin per la fase Transmogrifier, è necessario chiamare HTTPD_open(), HTTPD_write() e HTTPD_close() nell'ambito delle funzioni open, write e close. HTTPD_write() può essere chiamato solo dopo la funzione HTTPD_open(). Lo scopo di queste funzioni predefinite è quello di fornire controllo al server in modo che sia possibile chiamare la funzione successiva nella sequenza.
- È necessario chiamare le funzioni HTTPD_* per la corretta esecuzione della fase dell'API Transmogrifier e del server. Ad esempio, se non vengono chiamati HTTPD_open() e HTTPD_close(), le intestazioni non vengono restituite al client.
- Tenere presente che possono verificarsi degli effetti indesiderati se le applicazioni di filtraggio dati non sono adeguatamente selettive durante il filtraggio dei flussi di dati. È possibile che le variabili CGI non funzionino se sono state filtrate in maniera scorretta, che i file GIF non vengano visualizzati e che gli altri flussi binari non funzionino come previsto.
- Non è necessario che il plugin memorizzi nel buffer il corpo del contenuto. Il Caching Proxy determina automaticamente la lunghezza del contenuto.
- Quando si è pronti a fornire il controllo delle intestazioni al server, si consiglia di chiamare HTTPD_open(). Tuttavia, se in seguito è necessario impostare un'intestazione nel programma API, attendere fino a quando la funzione write o close non chiami la funzione HTTPD_open().

  Nota: è necessario impostare tutte intestazioni mediante HTTPD_set() o httpd_setvar() prima di chiamare la funzione HTTPD_open().
- Il flusso di dati non include le intestazioni. I plugin devono utilizzare le funzioni set ed extract per gestire le intestazioni. La funzione *open* del plugin non viene chiamata fino a quando non sono state lette tutte le intestazioni.
- È possibile utilizzare più plugin transmogrifier, i quali vengono chiamati nell'ordine in cui appaiono nel file di configurazione.
- L'operazione di tunnel SSL non viene inoltrata mediante i plugin transmogrifier.

Advisor GC

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

Per ogni file presente nella cache durante la raccolta dei dati inutili, viene chiamata una funzione definita per questa fase. Tale funzione consente di influenzare i file che vengono conservati e quelli che vengono eliminati. Per ulteriori informazioni, consultare le variabili GC_*.

Proxy Advisor

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

Durante il servizio di ogni richiesta proxy, viene chiamata una funzione definita per questa fase. Tale funzione può essere utilizzata, ad esempio, per impostare la variabile USE_PROXY.

Log

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

Dopo che ogni richiesta è stata elaborata e che la comunicazione con il client è stata interrotta viene chiamata una funzione definita per questa fase. Se si desidera che la funzione plugin venga chiamata solo per le richieste che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. Questa funzione viene chiamata indipendentemente dal fatto che la richiesta sia stata elaborata correttamente. Se non si desidera che il plugin di log sostituisca il meccanismo di log, impostare il codice di ritorno su HTTP_NOACTION invece che su HTTP_OK.

Errore

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

Per ogni richiesta che ha esito negativo viene chiamata una funzione definita per questa fase. Se si desidera che la funzione del plugin venga chiamata solo per le richieste non riuscite che corrispondono alla maschera, è possibile specificare una maschera URL nella direttiva del file di configurazione. La fase Error consente di personalizzare la risposta all'errore.

PostExit

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

Per ogni richiesta, completata o non completata correttamente, viene chiamata una funzione definita per questa fase. Questa fase consente di deallocare tutte le risorse assegnate dal plugin per elaborare la richiesta.

Chiusura del server

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

Quando si verifica una chiusura corretta del server, viene chiamata una funzione definita per questa fase. Essa consente di deallocare le risorse assegnate durante la fase di Inizializzazione del server. Non chiamare le funzioni HTTP_* in questa fase (i risultati sono imprevedibili). Se esiste più di una direttiva API del Caching Proxy nel file di configurazione per la fase Chiusura del server, verranno chiamate tutte.

Nota: a causa di una limitazione nel codice Solaris, la fase del plugin Chiusura del server non viene eseguita se si utilizza il comando **ibmproxy -stop** per chiudere il Caching Proxy sulle piattaforme Solaris. Per informazioni sull'avvio e l'arresto del Caching Proxy, fare riferimento alla *Guida alla gestione per WebSphere Application Server Caching Proxy*.

Valori e codici di ritorno HTTP

Questi codici di ritorno seguono la specifica HTTP 1.1, RFC 2616, pubblicata dal World Wide Web Consortium (www.w3.org/pub/WWW/Protocols/). Le funzioni del plugin devono restituire uno di questi valori.

Tabella 2. Codici di ritorno HTTP per le funzioni API del Caching Proxy

Valore	Codice di ritorno
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
400	HTTP_BAD_REQUEST
401	HTTP_UNAUTHORIZED
403	HTTP_FORBIDDEN
404	HTTP_NOT_FOUND
405	HTTP_METHOD_NOT_ALLOWED

Tabella 2. Codici di ritorno HTTP per le funzioni API del Caching Proxy (Continua)

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

Funzioni e macro predefinite

È possibile richiamare le funzioni e le macro predefinite del server dalle funzioni del plugin. A tal fine, utilizzare i relativi nomi predefiniti e rispettare il formato descritto di seguito. Nelle descrizioni relative al parametro, la lettera `i` indica un parametro di input, la lettera `o` indica un parametro di output e le lettere `i/o` indicano che un parametro viene utilizzato sia per l'input sia per l'output.

Ciascuna di queste funzioni restituisce uno dei codici di ritorno HTTPD, a seconda della corretta esecuzione della richiesta. Questi codici vengono descritti in "Codici di ritorno delle funzioni e delle macro predefinite" a pagina 23.

Quando vengono chiamate queste funzioni, utilizzare come primo parametro l'handle fornito al plugin. Altrimenti, la funzione restituisce un codice di errore HTTPD_PARAMETER_ERROR. NULL non è accettato come handle valido.

HTTPD_authenticate()

Autentica un ID utente o una password, o entrambi. È valido unicamente nelle fasi PreExit, Autenticazione, Autorizzazione e Post autorizzazione.

```
void
HTTPD_LINKAGE  HTTPD_authenticate (
         unsigned char *handle,       /* i; handle */
         long *return_code         /* o; codice di ritorno */
         )
```

HTTPD_cacheable_url()

Viene restituito se il contenuto dell'URL specificato può essere memorizzato nella cache in base agli standard del Caching Proxy.

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
        unsigned char *handle,       /* i; handle */
        unsigned char *url,         /* i; URL da controllare */
        unsigned char *req_method,  /* i; metodo di richiesta per l'URL */
        long *retval                /* o; codice di ritorno */
        )
```

Il valore di ritorno HTTPD_SUCCESS indica che il contenuto dell'URL può essere memorizzato nella cache; mentre HTTPD_FAILURE indica il contrario. Anche HTTPD_INTERNAL_ERROR è un possibile codice di ritorno per questa funzione.

HTTPD_close()

(Valido solo nella fase Transmogrifier.) Trasferisce il controllo alla routine *close* successiva nello stack del flusso. Chiamare questa funzione dalle funzioni open, write o close di Transmogrifier dopo l'esecuzione di qualsiasi elaborazione desiderata. Questa funzione indica al server proxy che la risposta è stata elaborata e che la fase Transmogrifier è completa.

```
void  HTTPD_LINKAGE
HTTPD_close (
         unsigned char *handle,       /* i; handle */
         long *return_code         /* o; codice di ritorno */
         )
```

HTTPD_exec()

Esegue uno script per soddisfare questa richiesta. Valido nelle fasi PreExit, Servizio, Post autorizzazione e Errore.

```
void  HTTPD_LINKAGE  HTTPD_exec (
         unsigned char *handle,       /* i; handle */
         unsigned char *name,           /* i; nome dello script da eseguire */
         unsigned long *name_length,  /* i; lunghezza del nome */
         long *return_code         /* o; codice di ritorno */
         )
```

HTTPD_extract()

Estrae il valore di una variabile associata a questa richiesta. Le variabili valide per il parametro *name* sono uguali a quelle utilizzate nel CGI. Per ulteriori informazioni, consultare "Variabili" a pagina 27. Si noti che questa funzione è valida in tutte le fasi; tuttavia, non tutte le variabili sono valide in tutte le fasi.

```
void  HTTPD_LINKAGE  HTTPD_extract (
      unsigned char *handle,       /* i; handle */
      unsigned char *name,         /* i; nome della variabile da estrarre */
      unsigned long *name_length,  /* i; lunghezza del nome */
      unsigned char *value,        /* o; buffer in cui inserire
                                        il valore */
      unsigned long *value_length, /* i/o; dimensione del buffer */
      long *return_code         /* o; codice di ritorno */
      )
```

Se questa funzione restituisce il codice HTTPD_BUFFER_TOO_SMALL, significa che la dimensione del buffer richiesta non è sufficientemente grande per il valore estratto. In questo caso, la funzione non utilizza il buffer, ma aggiorna il parametro value_length con la dimensione del buffer necessaria per estrarre questo valore con esito positivo. Tentare nuovamente l'estrazione con un buffer che sia abbastanza grande da contenere il parametro value_length restituito.

Nota: se la variabile estratta è per un'intestazione HTTP, la funzione HTTPD_extract() estrarrà solo la prima occorrenza corrispondente, anche se la richiesta contiene più intestazioni aventi lo stesso nome.

La funzione httpd_getvar() può essere utilizzata al posto della funzione HTTPD_extract(), inoltre offre altri vantaggi. Fare riferimento a 19 per ulteriori informazioni.

HTTPD_file()

Invia un file per soddisfare questa richiesta. Valido solo nelle fasi PreExit, Servizio, Errore, Post autorizzazione e Transmogrifier.

```
void  HTTPD_LINKAGE
HTTPD_file (
         unsigned char *handle,       /* i; handle */
         unsigned char *name,         /* i; nome del file da inviare */
         unsigned long *name_length,  /* i; lunghezza del nome */
         long *return_code         /* o; codice di ritorno */
         )
```

httpd_getvar()

Ha la stessa funzione di HTTPD_extract(), fatta eccezione per il fatto che è più semplice da utilizzare in quanto l'utente non è tenuto a specificare le lunghezze degli argomenti.

```
const  unsigned char *          /* o; valore della variabile */
HTTPD_LINKAGE
httpd_getvar(
   unsigned char *handle,       /* i; handle */
   unsigned char *name,         /* i; nome variabile */
   unsigned long *n             /* i; numero indice per la matrice
                                      contenente l'intestazione */
   )
```

L'indice della matrice che contiene l'intestazione comincia con 0. Per ottenere la prima voce nella matrice, utilizzare il valore 0 al posto di *n*; per ottenere la quinta voce, utilizzare il valore 4 al posto di *n*.

Nota: non eliminare o modificare il contenuto del valore restituito. La stringa restituita termina con null.

HTTPD_log_access()

Scrive una stringa sul log di accesso del server.

```
void  HTTPD_LINKAGE  HTTPD_log_access (
         unsigned char *handle,       /* i; handle */
         unsigned char *value,            /* i; dati da scrivere */
         unsigned long *value_length,     /* i; lunghezza dei dati */
         long *return_code         /* o; codice di ritorno */
         )
```

Si noti che i simboli escape *non* sono richiesti durante la scrittura del simbolo di percentuale (%) nei log di accesso del server.

HTTPD_log_error()

Scrive una stringa sul log degli errori del server.

```
void  HTTPD_LINKAGE  HTTPD_log_error (
         unsigned char *handle,       /* i; handle */
         unsigned char *value,            /* i; dati da scrivere */
         unsigned long *value_length,     /* i; lunghezza dei dati */
         long *return_code         /* o; codice di ritorno */
         )
```

Si noti che i simboli escape *non* sono richiesti durante la scrittura del simbolo di percentuale (%) nei log degli errori del server.

HTTPD_log_event()

Scrive una stringa nel registro eventi del server.

```
void  HTTPD_LINKAGE
HTTPD_log_event (
         unsigned char *handle,       /* i; handle */
         unsigned char *value,            /* i; dati da scrivere */
         unsigned long *value_length,     /* i; lunghezza dei dati */
         long *return_code         /* o; codice di ritorno */
         )
```

Si noti che i simboli escape *non* sono richiesti durante la scrittura del simbolo di percentuale (%) nei registri eventi del server.

HTTPD_log_trace()

Scrive una stringa nel registro tracce del server.

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
         unsigned char *handle,       /* i; handle */
         unsigned char *value,            /* i; dati da scrivere */
         unsigned long *value_length,     /* i; lunghezza dei dati */
         long *return_code         /* o; codice di ritorno */
         )
```

Si noti che i simboli escape *non* sono richiesti durante la scrittura del simbolo di percentuale (%) nei registri di tracce del server.

HTTPD_open()

(Valido solo nella fase Transmogrifier.) Trasferisce il controllo alla successiva routine nello stack del flusso. Chiamare questa funzione dalle funzioni open, write o close di Transmogrifier dopo aver impostato tutte le intestazioni desiderate e se si è pronti ad avviare la routine di scrittura.

```
void  HTTPD_LINKAGE  HTTPD_open (
         unsigned char *handle,       /* i; handle */
         long *return_code         /* o; codice di ritorno */
         )
```

HTTPD_proxy()

Effettua una richiesta proxy. Valido nelle fasi PreExit, Servizio e Post autorizzazione.

Nota: questa è una funzione di completamento; infatti, dopo questa funzione, la richiesta è completa.

```
void  HTTPD_LINKAGE  HTTPD_proxy (
         unsigned char *handle,       /* i; handle */
         unsigned char *url_name,      /* i; URL per la
                                             richiesta proxy */
         unsigned long *name_length,   /* i; lunghezza dell'URL */
         void *request_body,           /* i; corpo della richiesta */
         unsigned long *body_length,   /* i; lunghezza del corpo */
         long *return_code         /* o; codice di ritorno */
         )
```

HTTPD_read()

Legge il corpo della richiesta del client. Utilizzare HTTPD_extract() per le intestazioni. Valido solo per le fasi PreExit, Autorizzazione, Post autorizzazione e Servizio ed è utile solo se è stata effettuata una richiesta PUT o POST. Chiamare questa funzione in un loop fino a quando non viene restituito HTTPD_EOF. Se non è presente nessun corpo per questa richiesta, la funzione ha esito negativo.

```
void  HTTPD_LINKAGE
HTTPD_read (
         unsigned char *handle,       /* i; handle */
         unsigned char *value,           /* i; buffer dei dati */
```

```
        unsigned long *value_length,    /* i/o; dimensione del buffer
                                                (lunghezza dati) */
        long *return_code         /* o; codice di ritorno */
        )
```

HTTPD_restart()

Riavvia il server dopo che tutte le richieste attive sono state elaborate. Valido in tutte le fasi, tranne per Inizializzazione del server, Chiusura del server e Transmogrifier.

```
void  HTTPD_LINKAGE  HTTPD_restart (
        long *return_code         /* o; codice di ritorno */
        )
```

HTTPD_set()

Imposta il valore di una variabile associata a questa richiesta. Le variabili valide per il parametro *name* sono uguali a quelle utilizzate nel CGI. Per ulteriori informazioni, consultare "Variabili" a pagina 27.

Si noti che con questa funzione è anche possibile creare delle variabili. Le variabili create sono soggette alle convenzioni dei prefissi HTTP_ e PROXY_, descritti in "Variabili" a pagina 27. Se si crea una variabile che comincia con HTTP_, essa viene inviata come intestazione nella risposta al client, senza il prefisso HTTP_. Ad esempio, per impostare un'intestazione di ubicazione, utilizzare HTTPD_set() con il nome variabile HTTP_LOCATION. Le variabili create con un prefisso PROXY_ vengono inviate come intestazioni nella richiesta al server dei contenuti. Le variabili create con un prefisso CGI_ vengono inviate ai programmi CGI.

Questa funzione è valida in tutte le fasi; tuttavia, non tutte le variabili sono valide in tutte le fasi.

```
void  HTTPD_LINKAGE  HTTPD_set (
        unsigned char *handle,       /* i; handle */
        unsigned char *name,          /* i; nome del valore da impostare */
        unsigned long *name_length,  /* i; lunghezza del nome */
        unsigned char *value,         /* i; buffer con valore */
        unsigned long *value_length,  /* i; lunghezza del valore */
        long *return_code         /* o; codice di ritorno */
        )
```

Nota: è possibile utilizzare la funzione httpd_setvar() per impostare un valore della variabile senza dover specificare un buffer e la lunghezza. Per informazioni, fare riferimento a 21.

httpd_setvar()

Ha la stessa funzione di HTTPD_set(), tranne per il fatto che è più semplice da utilizzare in quanto l'utente non deve specificare le lunghezze degli argomenti.

```
long             /* o; codice di ritorno */
HTTPD_LINKAGE   httpd_setvar (
       unsigned char *handle,       /* i; handle */
       unsigned char *name,         /* i; nome variabile */
       unsigned char *value,        /* i; nuovo valore */
       unsigned long *addHdr        /* i; aggiungere intestaz. o sostituirla */
       )
```

Il parametro *addHdr* ha quattro valori possibili:

- HTTPD_SETVAR_REPLACE — Indica di sostituire tutte le occorrenze della variabile intestazione con il nuovo valore.
- HTTPD_SETVAR_REPLACE_ADD — Se la variabile intestazione esiste già, sostituire la sua prima occorrenza con il nuovo valore; se la variabile non esiste, aggiungere il nuovo valore alle intestazioni.

- HTTPD_SETVAR_ADD — Indica di aggiungere questo valore alle intestazioni.
- HTTPD_SETVAR_REMOVE_ALL — Indica di cancellare tutte le occorrenze di questa variabile intestazione.

Questi valori vengono definiti in HTAPI.h.

httpd_variant_insert()

Inserisce una variante nella cache.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,       /* i; handle */
         unsigned char *URI,       /* i; URI di questo oggetto */
         unsigned char *dimension, /* i; dimensione della variante */
         unsigned char *variant,   /* i; valore della variante */
         unsigned char *filename,  /* i; file contenente l'oggetto */
         long *return_code         /* o; codice di ritorno */
         )
```

Note:

1. L'argomento dimensione si riferisce all'intestazione con cui questo oggetto si differenzia dall'URI. Ad esempio, nel precedente esempio, un valore dimensione possibile è User-Agent.
2. L'argomento variante si riferisce al valore dell'intestazione per l'intestazione fornita nell'argomento dimensione. Questo varia dall'URI. Ad esempio, nel precedente esempio, un valore possibile per l'argomento variante è il seguente:

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

3. L'argomento nomefile deve ountare a una copia del nome del file terminata da null in cui l'utente ha salvato il contenuto modificato. L'utente è responsabile della rimozione del file; questa operazione è sicura dopo essere stata restituita da questa funzione. Il file contiene solo il corpo senza intestazioni.
4. Durante la memorizzazione nella cache delle varianti, il server aggiorna l'intestazione content-lenght e aggiunge un'intestazione Warning: 214. Le tag di entità Strong vengono rimosse.

httpd_variant_lookup()

Determina se una variante specifica è già esistente nella cache.

```
void
HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,       /* i; handle */
    unsigned char *URI,             /* URI di questo oggetto */
    unsigned char *dimension, /* i; dimensione della variante */
    unsigned char *variant,   /* i; valore della variante */
             long *return_code);       /* o; codice di ritorno */
```

HTTPD_write()

Scrive il corpo della risposta. Valido nelle fasi PreExit, Servizio, Errore e Transmogrifier.

Se la prima volta che si chiama questa funzione non viene impostato il tipo di contenuto, il server presume che si sta inviando un flusso di dati CGI.

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,       /* i; handle */
    unsigned char *value,           /* i; dati da inviare */
    unsigned char *value_length,       /* i; lunghezza dei dati */
             long *return_code);       /* o; codice di ritorno */
```

Nota: per impostare le intestazioni di risposta, fare riferimento a 21.

Nota: Al termina di funzione HTTPD_*, si consiglia di liberare la memoria utilizzata.

Codici di ritorno delle funzioni e delle macro predefinite

Il server imposterà il parametro del codice di ritorno su uno di questi valori, a seconda della riuscita della richiesta:

Tabella 3. Codici di ritorno

Valore	Codice stato	Spiegazione
-1	HTTPD_UNSUPPORTED	La funzione non è supportata.
0	HTTPD_SUCCESS	La funzione ha avuto esito positivo e i campi di output sono validi.
1	HTTPD_FAILURE	La funzione ha avuto esito negativo.
2	HTTPD_INTERNAL_ERROR	È stato rilevato un errore interno, quindi non è possibile proseguire l'elaborazione di questa richiesta.
3	HTTPD_PARAMETER_ERROR	Sono stati inoltrati uno o più parametri non validi.
4	HTTPD_STATE_CHECK	La funzione non è valida in questa fase del processo.
5	HTTPD_READ_ONLY	(Restituito solo da HTTPD_set e httpd_setvar.) La variabile è in sola lettura e non può essere impostata dal plugin.
6	HTTPD_BUFFER_TOO_SMALL	(Restituito da HTTPD_set, httpd_setvar e HTTPD_read.) Il buffer fornito è troppo piccolo.
7	HTTPD_AUTHENTICATE_FAILED	(Restituito solo da HTTPD_authenticate.) L'autenticazione ha avuto esito negativo. Per ulteriori informazioni, esaminare le variabili HTTP_RESPONSE e HTTP_REASON.
8	HTTPD_EOF	(Restituito solo da HTTPD_read.) Indica la fine del corpo della richiesta.
9	HTTPD_ABORT_REQUEST	La richiesta è stata interrotta perché il client ha fornito una tag entità che non corrisponde alla condizione specificata dalla richiesta.
10	HTTPD_REQUEST_SERVICED	(Restituito da HTTPD_proxy.) La funzione chiamata ha completato la risposta per questa richiesta.
11	HTTPD_RESPONSE_ALREADY_ COMPLETED	La funzione ha avuto esito negativo perché la risposta per quella richiesta è già stata completata.
12	HTTPD_WRITE_ONLY	La variabile è in sola scrittura e non può essere letta dal plug-in.

Direttive di configurazione del Caching Proxy per le fasi dell'API

Ogni fase di questo processo di richiesta ha una direttiva di configurazione che viene utilizzata per indicare quale delle funzioni dei plugin si desidera chiamare ed eseguire durante quella fase specifica. Tali direttive possono essere aggiunte al

file di configurazione del server (ibmproxy.conf) modificandolo e aggiornandolo manualmente o utilizzando il modulo Elaborazione delle richieste API contenuto nei moduli Configurazione e gestione del Caching Proxy.

Note sull'utilizzo delle API

- Fatta eccezione per le direttive Servizio e Conversione nome, non è necessario che le direttive API di ogni fase vengano visualizzate in un ordine particolare nel file di configurazione. Si noti che l'ordine di più voci per una direttiva API è significativo, come descritto più avanti in questo elenco.
- Non è necessario includere una voce per ogni fase API. Se non si dispone di un plugin per una fase particolare, omettere la direttiva corrispondente e utilizzare l'elaborazione standard.
- Le direttive Servizio e Conversione nome funzionano allo stesso modo delle altre direttive di mappatura (ad esempio, la direttiva Pass) e dipendono dalla relativa occorrenza e posizione rispetto ad altre direttive di mappatura all'interno del file di configurazione. Ad esempio, una regola per /cgi-bin/foo.so deve essere visualizzata prima della regola per /cgi-bin/*.

  Ciò significa che il server elabora le direttive Service, NameTrans, Exec, Fail, Map, Pass, Proxy, ProxyWAS e Redirect nella relativa sequenza all'interno del file di configurazione. Se il server mappa con esito positivo un'URL su un file, esso non legge né elabora nessun altra direttiva. (La direttiva Map è un'eccezione. Per informazioni complete sulle regole di mappatura del server proxy, fare riferimento alla *Guida alla gestione per WebSphere Application Server Caching Proxy*.)
- È possibile che esista più di una direttiva di configurazione per una fase. Ad esempio, è possibile includere due direttive NameTrans, ciascuna indicante una funzione differente del plugin. Quando il server esegue la fase di conversione del nome, elabora le funzioni di conversione del nome nell'ordine in cui vengono visualizzate all'interno del file di configurazione.

  Nota: se una funzione di plugin fornita con il Caching Proxy utilizza la stessa direttiva API di un plugin scritto dall'utente, posizionare la direttiva del plugin dopo la direttiva del plugin di sistema.
- Non è necessario che per ogni richiesta vengano eseguite determinate funzioni di plugin:
  - Diverse direttive includono una maschera URL. Se si specifica una maschera URL con queste direttive, l'applicazione del plugin viene chiamata solo per le richieste i cui URL corrispondono al modello. Fare riferimento a "Sintassi e direttive API" a pagina 25 per informazioni sulle fasi che possono utilizzare le maschere URL e a "Variabili della direttiva API" a pagina 25 per informazioni su come utilizzare questa funzione.
  - Specificare uno schema di autenticazione con la direttiva di autenticazione per indicare che si desidera chiamare il plugin solo per determinati tipi di autenticazione. Attualmente, il protocollo HTTP supporta solo l'autenticazione di base. Consultare "Variabili della direttiva API" a pagina 25 per maggiori informazioni.
- Se il server non riesce a caricare una funzione specifica del plugin o se si dispone di una direttiva ServerInit che non restituisce un codice di ritorno OK, non viene chiamata nessun altra funzione per quel plugin del Caching Proxy compilato. Qualsiasi elaborazione specifica per quel plugin effettuata fino a quel momento, viene ignorata. Gli altri plugin del Caching Proxy inclusi in queste direttive e le relative funzioni non vengono influenzati.

Sintassi e direttive API

Queste direttive del file di configurazione devono essere visualizzate nel file ibmproxy.conf su un'unica riga, senza spazi se non quelli ivi specificati esplicitamente. Sebbene per agevolare la leggibilità, in alcuni esempi di sintassi potrebbero apparire delle interruzioni di riga, è necessario che in quei punti della direttiva reale non vi siano spazi.

Tabella 4. Direttive dell'API del plugin del Caching Proxy

ServerInit		*/path/file:function_name init_string*
PreExit		*/path/file:function_name*
Autenticazione	*tipo*	*/path/file:function_name*
NameTrans	*/URL*	*/path/file:function_name*
Autorizzazione	*/URL*	*/path/file:function_name*
Tipo di oggetto	*/URL*	*/path/file:function_name*
PostAuth		*/path/file:function_name*
Servizio	*/URL*	*/path/file:function_name*
Mezzanotte		*/path/file:function_name*
Transmogrifier		*/path/file:open_function_name: write_function_name: close_function_name:error_function*
Log	*/URL*	*/path/file:function_name*
Errore	*/URL*	*/path/file:function_name*
PostExit		*/path/file:function_name*
ServerTerm		*/path/file:function_name*
ProxyAdvisor		*/path/file:function_name*
GCAdvisor		*/path/file:function_name*

Variabili della direttiva API

Le variabili contenute in queste direttive hanno i seguenti significati:

tipo Utilizzata solo con la direttiva di autenticazione per specificare se la funzione del plugin deve essere chiamata o meno. I valori validi sono i seguenti:

- Basic — La funzione del plugin viene chiamata solo per le richieste di autenticazione di base.
- * — La funzione del plugin viene chiamata per tutte le richieste. Attualmente, il protocollo HTTP supporta solo l'autenticazione di base. Per le richieste di autorizzazione non di base, è possibile restituire un codice di errore non supportato da questo tipo di autenticazione.

URL Specifica le richieste per cui viene chiamata la funzione del plugin. Le richieste con gli URL che corrispondono a questa maschera richiederanno l'utilizzo della funzione del plugin. Le specifiche URL in queste direttive sono virtuali (non includono il protocollo) ma sono precedute da una barra (/). Ad esempio, /www.ics.raleigh.ibm.com è corretto, mentre http://www.ics.raleigh.ibm.com non lo è. È possibile specificare questo valore come un URL specifico o come una maschera.

- URL specifico — La funzione del plugin viene chiamata solo per quell'URL particolare.

- Maschera URL — La funzione del plugin viene chiamata per tutti gli URL che corrispondono alla maschera. Le maschere possono includere il carattere jolly * e possono essere specificate nelle forme */URL**, /* o *

  Nota: se si desidera che venga eseguita la conversione del percorso, è *richiesta* una maschera URL con la direttiva Servizio.

path/file
Il nome file completo del programma compilato.

function_name
Il nome assegnato alla funzione del plugin all'interno del programma.

Se si desidera accedere alle informazioni presenti sul percorso, la direttiva Servizio richiede un asterisco (*) dopo il nome della funzione.

init_string
Questa parte facoltativa della direttiva ServerInit può contenere qualsiasi testo che si desidera inviare alla funzione di plugin. Utilizzare httpd_getvar() per estrarre il testo dalla variabile INIT_STRING.

Per ulteriori informazioni su queste direttive, inclusa la sintassi, consultare la *Guida alla gestione per WebSphere Application Server Caching Proxy*.

Compatibilità con altre API

L'API del Caching Proxy è compatibile con le versioni precedenti di ICAPI e GWAPI, mediante la versione 4.6.1.

Spostamento dei programmi CGI

Utilizzare le seguenti linee guida per far sì che le applicazioni CGI scritte in C utilizzino l'API del Caching Proxy:

- Rimuovere il punto di ingresso main() o ridenominarlo in modo che si possa creare una DLL.
- Eliminare le variabili globali o proteggerle con un semaforo ad esclusione reciproca.
- Modificare le seguenti chiamate nei programmi:
  - Modificare le chiamate dell'intestazione printf() in HTTPD_set() o httpd_setvar().
  - Modificare le chiamate dati printf() in HTTPD_write().
  - Modificare le chiamate getenv() in HTTPD_extract() o httpd_getvar(). Si noti che in tal modo viene restituita memoria deallocata, quindi è necessario liberare il risultato.
- Il server è in esecuzione in un ambiente basato su più thread; pertanto, è necessario che i plugin facciano un uso limitato di thread. Se le funzioni sono rientranti, le prestazioni non diminuiscono.
- Ricordarsi di impostare l'intestazione Content-Type se si utilizza HTTPD_write() per inviare nuovamente i dati al client.
- Controllare con attenzione il codice per individuare eventuali errori di rilascio della memoria.
- Considerare i percorsi dell'errore. Se vengono generati messaggi di errore e inviati come HTML, è necessario che la funzione o le funzioni di servizio restituiscano HTTPD_OK.

Informazioni di riferimento sulle API del Caching Proxy

Variabili

Durante la scrittura dei programmi API, è possibile utilizzare le variabili del Caching Proxy che forniscono informazioni sul sistema server e client remoto.

Note:

- I nomi delle variabili definite dall'utente non possono avere il prefisso SERVER_. La funzione API del Caching Proxy riserva ogni variabile che comincia con SERVER_ per il server; tuttavia, queste variabili sono di sola lettura. Inoltre, anche i prefissi HTTP_ e PROXY_ sono riservati per le intestazioni HTTP.
- Tutte le intestazioni di richiesta inviate dal client (ad esempio Set-Cookie) sono precedute dal prefisso HTTP_, ed è possibile estrarre i relativi valori. Per accedere alle variabili che sono intestazioni di richiesta, anteporre al nome della variabile il prefisso HTTP_. È anche possibile creare nuove variabili utilizzando la funzione predefinita httpd_setvar(). Per i dettagli su queste intestazioni, consultare "Codici di ritorno delle funzioni e delle macro predefinite" a pagina 23.
- Vengono utilizzati due prefissi di variabili, HTTP_ e PROXY_, per indicare se una variabile si applica alle intestazioni di richiesta o di risposta. Il prefisso HTTP_ si riferisce alle variabili che fluiscono dal client al Caching Proxy. Il prefisso PROXY_ si riferisce alle variabili che fluiscono dal Caching Proxy al server di origine (o il server successivo in una catena proxy). Queste variabili sono valide solo durante le fasi di elaborazione delle richieste.
  - L'estrazione di una variabile HTTP_* fornisce il valore di un'intestazione presente nella richiesta del client sul server proxy.
  - L'impostazione di una variabile HTTP_* imposta l'intestazione di risposta inviata dal server proxy al client.
  - L'estrazione di una variabile PROXY_* fornisce il valore per un'intestazione restituita dal server dei contenuti al server proxy.
  - L'impostazione di una variabile PROXY_* imposta l'intestazione della richiesta inviata dal server proxy al server dei contenuti (o al successivo server in una catena di proxy).

  Figura 2 illustra l'utilizzo di questi prefissi quando Caching Proxy gestisce una richiesta client.

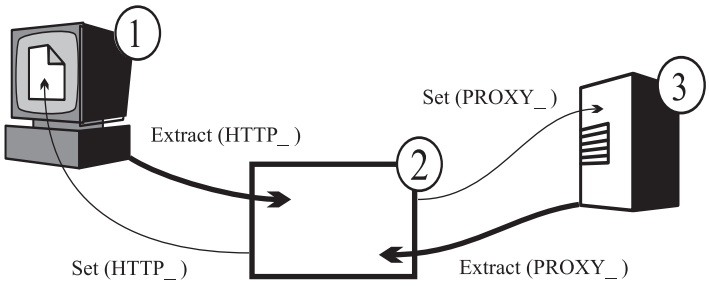

*Figura 2. Prefissi delle variabili HTTP_ e PROXY_*. Legenda: 1—Macchina client 2—Caching Proxy 3—Server di origine

- Alcune variabili sono di sola lettura. Le variabili di sola lettura rappresentano i valori estratti da una richiesta o da una risposta e vengono utilizzate nella

funzione predefinita httpd_getvar(). Se si tenta di modificare le variabili di sola lettura utilizzando la funzione httpd_setvar(), viene restituito il codice di ritorno HTTPD_READ_ONLY.

- Le variabili non identificate come di sola lettura possono essere lette e impostate nelle funzioni predefinite httpd_getvar() o httpd_setvar(). Queste variabili rappresentano i valori che si possono estrarre da una richiesta o da una risposta, o i valori che è possibile impostare o creare durante l'elaborazione di una richiesta o di una risposta.

Definizioni delle variabili

Nota: le variabili di intestazione che non cominciano con i prefissi HTTP_ o PROXY_ sono ambigue. Per evitare questa ambiguità, utilizzare sempre il prefisso HTTP_ o PROXY_ con i nomi delle variabili per le intestazioni.

ACCEPT_RANGES
Contiene il valore dell'intestazione di risposta Accept-Ranges, che specifica se il server dei contenuti può rispondere alle richieste di intervallo. Utilizzare PROXY_ACCEPT_RANGES per estrarre il valore dell'intestazione che viene inviata dal server dei contenuti al proxy. Utilizzare HTTP_ACCEPT_RANGES per impostare il valore dell'intestazione che viene inviata dal proxy al client.

Nota: ACCEPT_RANGES è ambiguo. Per eliminare l'ambiguità, utilizzare HTTP_ACCEPT_RANGES e PROXY_ACCEPT_RANGES.

ALL_VARIABLES
Sola lettura. Contiene tutte le variabili CGI. Ad esempio:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING
Sola lettura. Se il server supporta l'autenticazione del client, questa stringa contiene le credenziali non codificate da utilizzare per l'autenticazione del client.

AUTH_TYPE
Sola lettura. Se il server supporta l'autenticazione del client e lo script è protetto, questa variabile contiene il metodo utilizzato per autenticare il client. Ad esempio, `Basic`.

CACHE_HIT
Sola lettura. Identifica se la richiesta proxy è stata trovata o meno nella cache. I valori restituiti includono quanto segue:

- 0 - La richiesta non è stata trovata nella cache.
- 1 - La richiesta è stata trovata nella cache.

CACHE_MISS
Sola scrittura. Utilizzato per forzare un accesso alla cache non riuscito. I valori validi sono i seguenti:

- 0 - Non forzare un acesso alla cache non riuscito.
- 1 - Forzare un accesso alla cache non riuscito.

CACHE_TASK
Sola lettura. Identifica se è stata utilizzata la cache. I valori restituiti includono quanto segue:

- 0 - La richiesta non ha avuto accesso o non ha aggiornato la cache.
- 1 - La richiesta è stata soddisfatta dalla cache.

- 2 - L'oggetto richiesto si trovava nella cache, ma era necessario convalidarlo nuovamente.
- 3 - L'oggetto richiesto non si trovava nella cache e probabilmente è stato aggiunto.

Questa variabile può essere utilizzata nelle fasi PostAuthorization, PostExit, ProxyAdvisor o Log.

CACHE_UPDATE
Sola lettura. Identifica se la richiesta proxy ha aggiornato o meno la cache. I valori restituiti includono quanto segue:

- 0 - La cache non è stata aggiornata.
- 1 - La cache è stata aggiornata.

CLIENT_ADDR o CLIENTADDR
Uguale a REMOTE_ADDR.

CLIENTMETHOD
Uguale a REQUEST_METHOD.

CLIENT_NAME o CLIENTNAME
Uguale a REMOTE_HOST.

CLIENT_PROTOCOL o CLIENTPROTOCOL
Contiene il nome e la versione del protocollo che il client sta utilizzando per effettuare la richiesta. Ad esempio, `HTTP/1.1`.

CLIENT_RESPONSE_HEADERS
Sola lettura. Restituisce un buffer contenente le intestazioni che il server invia al client.

CONNECTIONS
Sola lettura. Contiene il numero di connessioni supportate o il numero di richieste attive. Ad esempio, `15`.

CONTENT_CHARSET
Serie di caratteri della risposta per text/*, ad esempio, `US ASCII` . L'estrazione di questa variabile è valida per l'intestazione content-charset del client. La relativa impostazione influisce sull'intestazione content-charset nella richiesta al server dei contenuti.

CONTENT_ENCODING
Specifica la codifica utilizzata nel documento, ad esempio, `x-gzip` . L'estrazione di questa variabile si applica all'intestazione content-encoding del client. La relativa impostazione influisce sull'intestazione content-charset nella richiesta al server dei contenuti.

CONTENT_LENGTH
L'estrazione di questa variabile si applica all'intestazione della richiesta del client. La relativa impostazione influisce sull'intestazione nella richiesta al server dei contenuti.

Nota: CONTENT_LENGTH è ambiguo. Per eliminare l'ambiguità, utilizzare HTTP_CONTENT_LENGTH e PROXY_CONTENT_LENGTH.

CONTENT_TYPE
L'estrazione di questa variabile si applica all'intestazione della richiesta del client. La relativa impostazione influisce sull'intestazione nella richiesta al server dei contenuti.

Nota: CONTENT_TYPE è ambiguo. Per eliminare l'ambiguità, utilizzare HTTP_CONTENT_TYPE e PROXY_CONTENT_TYPE.

CONTENT_TYPE_PARAMETERS
Contiene altri attributi MIME, ma non la serie di caratteri. L'estrazione di questa variabile si applica all'intestazione della richiesta del client. La relativa impostazione influisce sul valore dell'intestazione nella richiesta al server dei contenuti.

DOCUMENT_URL
Contiene l'URL (Uniform Request Locator). Ad esempio:

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

DOCUMENT_URI
Uguale a DOCUMENT_URL.

DOCUMENT_ROOT
Sola lettura. Contiene il percorso della root del documento, come definito dalle regole di inoltro.

ERRORINFO
Specifica il codice di errore per determinare la pagina errore. Ad esempio, `blocked`.

EXPIRES
Definisce la data di scadenza dei documenti memorizzati in una cache del proxy. L'estrazione di questa variabile si applica all'intestazione della richiesta del client. La relativa impostazione influisce sul valore dell'intestazione nella richiesta al server dei contenuti. Ad esempio:

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

GATEWAY_INTERFACE
Read-only. Contains the version of the API that the server is using. For example, `ICSAPI/2.0`.

GC_BIAS
Write-only. This floating-point value influences the garbage collection decision for the file being considered for garbage collection. The value entered is multiplied by the Caching Proxy's quality setting for that file type to determine ranking. Quality settings range from 0.0 to 0.1 and are defined by the AddType directives in the proxy configuration file (ibmproxy.conf).

GC_EVALUATION
Write-only. This floating-point value determines whether to remove (0.0) or keep (1.0) the file being considered for garbage collection. I valori compresi tra 0,0 e 1,0 vengono ordinati in base alla classificazione, vale a dire, è molto probabile che venga rimosso un file avente il valore GC_EVALUATION di 0,1 e non un file con il valore GC_EVALUATION di 0,9.

GC_EXPIRES
Sola lettura. Identifica quanti secondi devono trascorrere prima della scadenza del file in esame nella cache. Questa variabile può essere estratta solo da un plugin Advisor GC.

GC_FILENAME
Sola lettura. Identifica il file preso destinato alla raccolta di dati inutili. Questa variabile può essere estratta solo da un plugin Advisor GC.

GC_FILESIZE
Sola lettura. Identifica la dimensione del file destinato alla raccolta di dati inutili. Questa variabile può essere estratta solo da un plugin Advisor GC.

GC_LAST_ACCESS
Sola lettura. Identifica la data dell'ultimo accesso al file. Questa variabile può essere estratta solo da un plugin Advisor GC.

GC_LAST_CHECKED
Sola lettura. Identifica l'ultimo controllo dei file. Questa variabile può essere estratta solo da un plugin Advisor GC.

GC_LOAD_DELAY
Sola lettura. Identifica il tempo necessario per richiamare il file. Questa variabile può essere estratta solo da un plugin Advisor GC.

HTTP_COOKIE
Quando viene letta, questa variabile contiene il valore dell'intestazione Set-Cookie impostata dal client. Può essere utilizzata anche per impostare un nuovo cookie nel flusso di risposta (tra il proxy e il client). L'impostazione di questa variabile induce la creazione di una nuova intestazione Set-Cookie nel flusso di richieste documenti, a prescindere dalla presenza o meno di un'intestazione duplicata.

HTTP_HEADERS
Sola lettura. Utilizzato per estrarre tutte le intestazioni di richiesta client.

HTTP_REASON
L'impostazione di questa variabile influisce sulla stringa reason nella risposta HTTP. L'impostazione di questa variabile influisce sulla stringa reason nella risposta del proxy al client. L'estrazione di questa variabile restituisce la stringa reason nella risposta proveniente dal server dei contenuti al proxy.

HTTP_RESPONSE
L'impostazione di questa variabile influisce sul codice di risposta nella risposta HTTP. La sua impostazione influisce anche sul codice di stato nella risposta del proxy al client. L'estrazione di questa variabile restituisce il codice di stato nella risposta proveniente dal server dei contenuti al proxy.

HTTP_STATUS
Contiene il codice di risposta HTTP e la stringa reason. Ad esempio, `200 OK`.

HTTP_USER_AGENT
Contiene il valore dell'intestazione della richiesta User-Agent, vale a dire il nome del browser web del client, ad esempio, `Netscape Navigator / V2.02` . L'impostazione di questa variabile influisce sull'intestazione nella risposta del proxy al client. La relativa estrazione viene applicata all'intestazione della richiesta del client.

INIT_STRING
Sola lettura. La direttiva ServerInit definisce questa stringa. Questa variabile può essere letta solo durante l'inizializzazione del server.

LAST_MODIFIED
L'estrazione di questa variabile si applica all'intestazione della richiesta del client. La relativa impostazione influisce sull'intestazione nella richiesta al server dei contenuti. Ad esempio:

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

LOCAL_VARIABLES
Sola lettura. Tutte le variabili definite dall'utente.

MAXACTIVETHREADS
Sola lettura. Il numero massimo di thread attivi.

NOTMODIFIED_TO_OK
Forza una risposta completa al client. Valido nelle fasi PreExit e ProxyAdvisor.

ORIGINAL_HOST
Sola lettura. Restituisce il nome host o l'indirizzo IP di destinazione di una richiesta.

ORIGINAL_URL
Sola lettura. Restituisce l'URL originale inviato nella richiesta client.

OVERRIDE_HTTP_NOTRANSFORM
Abilita la modifica dei dati alla presenza di un'intestazione Cache-Control: no-transform. L'impostazione di questa variabile influisce sull'intestazione di risposta al client.

OVERRIDE_PROXY_NOTRANSFORM
Abilita la modifica dei dati alla presenza di un'intestazione Cache-Control: no-transform. L'impostazione di questa variabile influisce sulla richiesta al server dei contenuti.

PASSWORD
Per l'autenticazione di base, contiene la password decodificata. Ad esempio, `password`.

PATH Contiene il percorso completo convertito.

PATH_INFO
Contiene ulteriori informazioni sul percorso inviate dal browser Web. Ad esempio, `/foo`.

PATH_TRANSLATED
Contiene la versione decodificata o convertita delle informazioni sul precorso contenute in PATH_INFO. Ad esempio:

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

```
/wwwhome/foo
```

PPATH
Contiene il percorso parzialmente convertito. Utilizzarlo nella fase Conversione nome.

PROXIED_CONTENT_LENGTH
Sola lettura. Restituisce la lunghezza dei dati di risposta attualmente trasferiti tramite il server proxy.

PROXY_ACCESS
Definisce se la richiesta è una richiesta proxy. Ad esempio, `NO`.

PROXY_CONTENT_TYPE
Contiene l'intestazione Content-Type della richiesta proxy effettuata tramite HTTPD_proxy(). Se le informazioni vengono inviate con il metodo POST, questa variabile contiene il tipo di dati inclusi. È possibile creare il proprio tipo di contenuto nel file di configurazione del server proxy e mapparlo su un visualizzatore. L'estrazione di questa variabile si applica al valore dell'intestazione della risposta del server dei contenuti. La relativa impostazione influisce sull'intestazione per la richiesta al server dei contenuti. Ad esempio:

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

PROXY_CONTENT_LENGTH
L'intestazione Content-Length della richiesta proxy effettuata tramite HTTPD_proxy(). Se le informazioni vengono inviate con il metodo POST, questa variabile contiene il numero di caratteri dei dati. Generalmente, i server non inviano un'indicatore di fine file quando inoltrano le informazioni utilizzando un input standard. Se necessario, per stabilire la fine della stringa di input, è possibile utilizzare il valore CONTENT_LENGTH. L'estrazione di questa variabile si applica al valore dell'intestazione della risposta del server dei contenuti. La relativa impostazione influisce sull'intestazione per la richiesta al server dei contenuti. Ad esempio:

```
7034
```

PROXY_COOKIE
Quando viene letta, questa variabile contiene il valore dell'intestazione Set-Cookie impostata dal server di origine. Essa può essere utilizzata anche per impostare un nuovo cookie nel flusso di risposte. L'impostazione di questa variabile induce la creazione di una nuova intestazione Set-Cookie nel flusso di richieste documenti, a prescindere dalla presenza o meno di un'intestazione duplicata.

PROXY_HEADERS
Sola lettura. Utilizzato per estrarre le intestazioni Proxy.

PROXY_METHOD
Metodo per la richiesta effettuata tramite HTTPD_proxy(). L'estrazione di questa variabile si applica al valore dell'intestazione della risposta del server dei contenuti. La relativa impostazione influisce sull'intestazione per la richiesta al server dei contenuti.

QUERY_STRING
Se le informazioni vengono inviate utilizzando un metodo GET, questa variabile contiene le informazioni situate dopo un punto interrogativo (?) in una query. Queste informazioni devono essere decodificate dal programma CGI. Ad esempio:

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

RCA_OWNER
Sola lettura. Restituisce un valore numerico, che identifica il nodo dell'oggetto richiesto. Questa variabile può essere utilizzata nelle fasi PostExit, ProxyAdvisor o Log ed è significativa solo se il server fa parte di una matrice di cache che utilizza RCA (remote cache access).

RCA_TIMEOUTS
Sola lettura. Restituisce un valore numerico contenente il numero totale (aggregato) di timeout sulle richieste RCA per tutti i peer. È possibile utilizzare questa variabile in qualsiasi fase.

REDIRECT_*
Sola lettura. Contiene informazioni su una stringa di reindirizzamento per il codice di errore che corrisponde al nome della variabile (ad esempio, `REDIRECT_URL`). Nella documentazione online è possibile trovare un elenco delle variabili REDIRECT_ possibili per il server web Apache, all'indirizzo http://httpd.apache.org/docs-2.0/custom-error.html.

REFERRER_URL
Sola lettura. Contiene l'ubicazione più recente dell'URL del browser.

Consente al client di specificare, a vantaggio del server, l'indirizzo (URL) della risorsa da cui si ottiene Request-URL. Ad esempio:

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

REMOTE_ADDR
Contiene l'indirizzo IP del browser Web, se disponibile. Ad esempio, `45.23.06.8`.

REMOTE_HOST
Contiene il nome host del browser Web, se disponibile. Ad esempio, `www.raleigh.ibm.com`.

REMOTE_USER
Se il server supporta l'autenticazione del client e lo script è protetto, questa variabile contiene il nome utente inoltrato per l'autenticazione. Ad esempio, `joeuser`.

REQHDR
Sola lettura. Contiene un elenco delle intestazioni inviate dal client.

REQUEST_CONTENT_TYPE
Sola lettura. Restituisce il tipo di contenuto del corpo della richiesta. Ad esempio:

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

REQUEST_CONTENT_LENGTH
Sola lettura. Se le informazioni vengono inviate con il metodo POST, questa variabile contiene il numero di caratteri dei dati. Generalmente, i server non inviano un'indicatore di fine file quando inoltrano le informazioni utilizzando un input standard. Se necessario, per stabilire la fine della stringa di input, è possibile utilizzare il valore CONTENT_LENGTH. Ad esempio, `7034`.

REQUEST_METHOD
Sola lettura. Contiene il metodo (come specificato con l'attributo METHOD in un modulo HTML) utilizzato per inviare la richiesta. Ad esempio, `GET o POST`.

REQUEST_PORT
Sola lettura. Restituisce il numero di porta specificato nell'URL o una porta predefinita basata sul protocollo.

RESPONSE_CONTENT_TYPE
Sola lettura. Se le informazioni vengono inviate con il metodo POST, questa variabile contiene il tipo di dati inclusi. È possibile creare il proprio tipo di contenuto nel file di configurazione del server proxy e mapparlo su un visualizzatore. Ad esempio, `text/html`.

RESPONSE_CONTENT_LENGTH
Sola lettura. Se le informazioni vengono inviate con il metodo POST, questa variabile contiene il numero di caratteri dei dati. Generalmente, i server non inviano un'indicatore di fine file quando inoltrano le informazioni utilizzando un input standard. Se necessario, per stabilire la fine della stringa di input, è possibile utilizzare il valore CONTENT_LENGTH. Ad esempio, `7034`.

RULE_FILE_PATH
Sola lettura. Contiene il percorso completo del file system e il nome del file di configurazione.

SSL_SESSIONID
Sola lettura. Restituisce l'ID di sessione se la richiesta corrente viene

ricevuta su una connessione SSL. Restituisce NULL se la richiesta corrente non viene ricevuta su una connessione SSL.

SCRIPT_NAME
Contiene l'URL della richiesta.

SERVER_ADDR
Sola lettura. Contiene l'indirizzo IP locale del server proxy.

SERVER_NAME
Sola lettura. Contiene il nome host del server proxy o l'indirizzo IP del server dei contenuti per questa richiesta. Ad esempio, `www.ibm.com`.

SERVER_PORT
Sola lettura. Contiene il numero di porta del server proxy a cui è stata inviata la richiesta client. Ad esempio, `80`.

SERVER_PROTOCOL
Sola lettura. Contiene il nome e la versione del protocollo utilizzato per effettuare la richiesta. Ad esempio, `HTTP/1.1`.

SERVER_ROOT
Sola lettura. Contiene la directory in cui viene installato il programma del server proxy.

SERVER_SOFTWARE
Sola lettura. Contiene il nome e la versione del server proxy.

STATUS
Contiene il codice di risposta HTTP e la stringa reason. Ad esempio, 200 OK.

TRACE
Determina la quantità delle informazioni tracciate. I valori di ritorno includono:

- OFF - Nessuna traccia.
- V - Modalità Verbose.
- VV - Modalità Very Verbose.
- MTV - Modalità Much Too Verbose.

URI Lettura/scrittura. Uguale a DOCUMENT_URL.

URI_PATH
Sola lettura. Restituisce solo parte del percorso di un'URL.

URL Lettura/scrittura. Uguale a DOCUMENT_URL.

URL_MD4
Sola lettura. Restituisce il nome del file di cache potenziale per la richiesta corrente.

USE_PROXY
Identifica il proxy da concatenare per la richiesta corrente. Specificare l'URL. Ad esempio, `http://myproxy:8080`.

USERID
Uguale a REMOTE_USER.

USERNAME
Uguale a REMOTE_USER.

Autenticazione e autorizzazione

Breve riesame della terminologia:

Autenticazione
La verifica dei token di sicurezza associati a questa richiesta per accertare l'identità del richiedente.

Autorizzazione
Un processo che utilizza i token di sicurezza per determinare se il richiedente ha accesso alla risorsa.

Figura 3 illustra il processo di autorizzazione e autenticazione del server proxy.

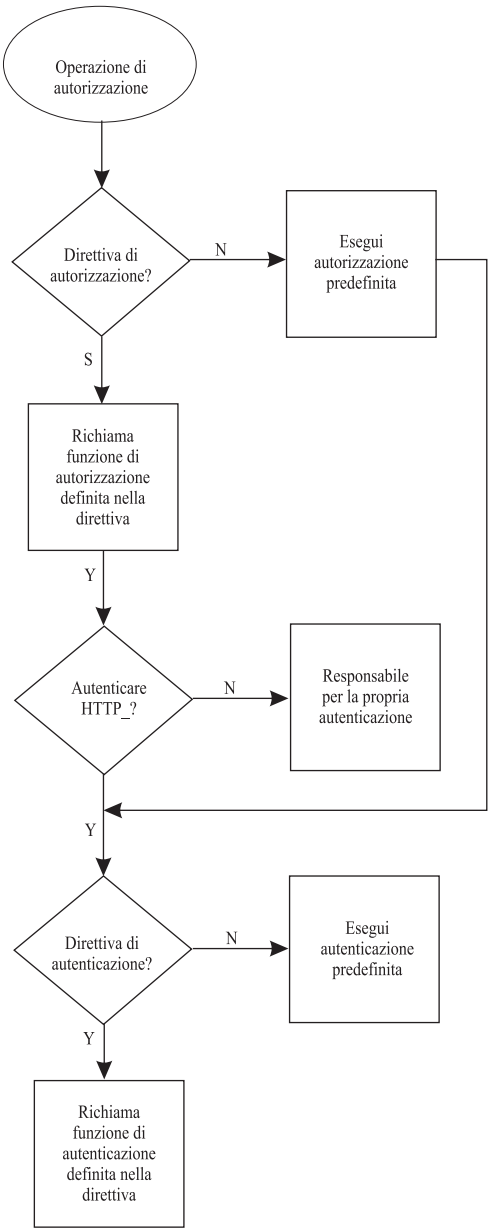

Figura 3. Processo di autorizzazione e autenticazione del server proxy

Come dimostrato in Figura 3 a pagina 36, l'avvio del processo di autorizzazione è il primo passo del processo di autorizzazione e autenticazione del server.

Nel Caching Proxy, l'autenticazione fa parte del processo di autorizzazione; si verifica solo quando viene richiesta l'autorizzazione.

Processo di autenticazione e autorizzazione

Il server proxy segue queste fasi durante l'elaborazione di una richiesta che richiede l'autorizzazione.

1. Innanzitutto, il server proxy esamina il file di configurazione per determinare se esiste o meno una direttiva di autorizzazione.
   - Se nel file di configurazione è presente una direttiva di autorizzazione, il server chiama la funzione di autorizzazione definita nella direttiva e avvia l'autenticazione con la fase 2.
   - Se non esiste alcuna direttiva di autorizzazione, il server esegue un'autorizzazione predefinita e procede direttamente con le procedure di autenticazione presenti nella fase 3.
2. Il server proxy avvia il processo di autenticazione verificando l'esistenza o meno dell'intestazione HTTP_authenticate nella richiesta client.
   - Se l'intestazione è presente, il server continua il processo di autenticazione (consultare il passo 3).
   - Se l'intestazione non è presente, l'autenticazione deve essere eseguita in base ad un altro metodo.
3. Il server proxy effettua il controllo per vedere se nel file di configurazione del proxy esiste una direttiva di autenticazione.
   - Se nel file di configurazione è presente una direttiva di autenticazione, il server chiama la funzione di autenticazione definita nella direttiva.
   - In caso contrario, il server effettua un'autenticazione predefinita.

Se il plugin del Caching Proxy fornisce il proprio processo di autorizzazione, *questo sostituisce* l'autorizzazione e l'autenticazione predefinita del server. Tuttavia, se nel file di configurazione sono presenti direttive di autorizzazione, è necessario che anche le funzioni del plugin ad esse associate gestiscano qualsiasi autorizzazione necessaria. A questo scopo viene fornita la funzione HTTPD_authenticate() predefinita.

Esistono tre metodi per fornire l'autenticazione nei plugin di autorizzazione:

- Scrivere autonomamente plugin di autorizzazione e autenticazione separati. Nel file di configurazione del proxy, per specificare queste funzioni utilizzare entrambe le direttive, quella di autorizzazione e quella di autenticazione. Accertarsi di aver incluso nella funzione di plugin di autorizzazione la chiamata funzione HTTPD_authenticate().

  Una volta effettuata l'operazione di autorizzazione, viene eseguita la funzione del plugin di autorizzazione, che a sua volta chiama la funzione del plugin di autenticazione.
- Scrivere la propria funzione del plugin di autorizzazione, ma fare in modo che tale funzione richiami l'autenticazione predefinita del server. Nel file di configurazione del proxy, utilizzare la direttiva di Autorizzazione per specificare la propria funzione. In tal caso, non è necessaria la direttiva di Autenticazione. Accertarsi di aver chiamato la funzione HTTPD_authenticate() nella funzione del plugin di autorizzazione.

Una volta effettuata l'operazione di autorizzazione, viene eseguita la funzione del plugin di autorizzazione che, a sua volta, chiama l'autenticazione predefinita del server.

- Scrivere la propria funzione del plugin di autorizzazione e includervi tutte le operazioni di autenticazione che devono essere elaborate. Non utilizzare la funzione HTTPD_authenticate() nel plugin di autorizzazione. Nel file di configurazione del proxy, utilizzare la direttiva di Autorizzazione per specificare il proprio plugin di autorizzazione. In tal caso, non è necessaria la direttiva di Autenticazione.

  Una volta eseguita l'operazione di autorizzazione, viene eseguita la funzione del plugin di autorizzazione, incluse eventuali autenticazioni.

Se il plugin del Caching Proxy non fornisce il processo di autorizzazione, è ancora possibile fornire un'autenticazione personalizzata utilizzando il seguente metodo:

- Scrivere la propria funzione del plugin di autenticazione. Nel file di configurazione del proxy, utilizzare le direttive di autenticazione per specificare la propria funzione. In questo caso, non è necessaria la direttiva di Autorizzazione.

Una volta effettuata l'operazione di Autorizzazione, viene eseguita l'autorizzazione predefinita del server che, a sua volta, chiama la funzione del plugin di autenticazione.

Tenere presenti i seguenti punti:

- Se nel proprio file di configurazione non si dispone di nessuna direttiva di Autorizzazione o se le funzioni del plugin specificate non gestiscono la richiesta restituendo HTTP_NOACTION, viene utilizzata l'autorizzazione predefinita del server.
- Se nel file di configurazione sono presenti le direttive di autorizzazione e le relative funzioni di plugin includono HTTPD_authenticate(), il server chiama tutte le funzioni di autenticazione specificate nelle direttive di autenticazione. Se, al contrario, non si dispone di nessuna direttiva di autenticazione definita o se le relative funzioni di plugin specificate non gestiscono la richiesta restituendo HTTP_NOACTION, viene utilizzata l'autenticazione predefinita del server.
- Se le direttive di autorizzazione sono presenti nel proprio file di configurazione ma le relative funzioni di plugin non includono HTTPD_authenticate(), il server non chiama nessuna funzione di autenticazione. È necessario scrivere il proprio processo di autenticazione come parte delle funzioni del plugin di autorizzazione o effettuare delle chiamate ad altri moduli di autenticazione.
- Se la funzione di autorizzazione restituisce i codici `401` o `407`, il Caching Proxy genera una richiesta in cui richiede al browser di restituire un ID utente e una password. Tuttavia, è ancora necessario configurare un'impostazione di protezione nel Caching Proxy affinché questa operazione possa essere eseguita correttamente.

Memorizzazione della variante nella cache

Utilizzare la funzione di memorizzazione nella cache della variante per memorizzare i dati, vale a dire un modulo modificato del documento originale (l'URI). Il Caching Proxy gestisce le varianti generate dall'API. Le *varianti* rappresentano versioni differenti di un documento di base.

In genere, quando i server di origine inviano delle varianti, la loro identificazione in quanto tali ha esito negativo. Il Caching Proxy supporta solo le varianti create dai plugin (ad esempio, la conversione della code page). Se un plugin crea una

variante in base a criteri non presenti nell'intestazione HTTP, è necessario includere un funzione della fase PreExit o PostAuthorization per creare una pseudo intestazione in modo che il Caching Proxy possa identificare correttamente la variante esistente.

Ad esempio, utilizzare un programma API Transmogrifier per modificare i dati richiesti dagli utenti in base al valore dell'intestazione User-Agent inviato dal browser. Nella funzione *close*, salvare il contenuto modificato in un file o specificare una lunghezza del buffer e inoltrare il buffer come argomento dati. Quindi, utilizzare le funzioni di memorizzazione nella cache della variante, httpd_variant_insert() e httpd_variant_lookup(), per inserire il contenuto nella cache.

Esempi di API

Per un'introduzione alle funzioni delle API del Caching Proxy, prendere in esame i programmi di esempio forniti nella directory esempi del CD-ROM di installazione di Edge Components. Altre informazioni sono disponibili sul sito web di WebSphere Application Server, www.ibm.com/software/webservers/appserv/.

Capitolo 3. Advisor personalizzati

Questa sezione analizza la scrittura degli advisor personalizzati per Load Balancer.

Gli advisor forniscono informazioni sul bilanciamento del carico

Gli advisor sono agenti software che operano all'interno di Load Balancer per fornire informazioni relative al carico su un server specifico. Esiste un advisor differente per ciascun protocollo standard (HTTP, SSL e altri). Periodicamente, il codice di base del Load Balancer esegue un ciclo di advisor, durante il quale valuta singolarmente lo stato di tutti i server della sua configurazione.

Scrivendo i propri advisor per il Load Balancer, è possibile personalizzare la modalità con cui viene ripartito il carico tra le macchine server.

Su sistemi Windows: Con le installazioni Load Balancer for IPv4 and IPv6, se si sta utilizzando il protocollo IPv6 e si desidera utilizzare gli advisor, è necessario modificare il file del **protocollo** che si trova nella directory C:\windows\system32\drivers\etc\.

Per IPv6, inserire la riga seguente nel file del protocollo:

```
ipv6-icmp
58   IPv6-ICMP      # IPv6 interface control message protocol
```

Funzione dell'advisor standard

In generale, gli advisor lavorano per consentire il bilanciamento del carico nel modo seguente.

1. Periodicamente, l'advisor avvia una connessione con ciascun server e invia un messaggio di richiesta. Il contenuto del messaggio è specifico del protocollo in esecuzione sul server; ad esempio, l'advisor HTTP invia una richiesta HEAD al server.
2. L'advisor resta in ascolto di una risposta da parte del server. Dopo aver ricevuto la risposta, l'advisor calcola e stabilisce il valore del carico per quel server. Advisor differenti calcolano il valore del carico in modi diversi, ma la maggior parte degli advisor standard misurano il tempo impiegato dal server per rispondere, quindi registrano quel valore in millisecondi.
3. L'advisor registra il carico sulla funzione di gestione del Load Balancer. Il carico viene visualizzato nella colonna Porta del report del gestore. Questi utilizza il carico registrato dall'advisor insieme ai pesi impostati dall'amministratore per stabilire come eseguire il bilanciamento del carico delle richieste in entrata sui server.
4. Se un server non risponde, l'advisor restituisce un valore negativo (-1) per il carico. Il gestore utilizza queste informazioni per stabilire quando sospendere il servizio di un particolare server.

Gli advisor standard forniti con Load Balancer includono advisor per le seguenti funzioni. Informazioni dettagliate su questi advisor sono disponibili nella *Guida alla gestione per WebSphere Application Server Load Balancer*

- Connect
- DB2

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

Per supportare i protocolli proprietari per cui non vengono forniti advisor standard è necessario scrivere advisor personalizzati.

Creazione di un advisor personalizzato

Un advisor personalizzato è una piccola parte del codice Java, fornito come file di classe, definito dal codice di base del Load Balancer per determinare il carico su un server. Il codice di base fornisce tutti i servizi di gestione necessari, inclusi l'avvio e l'arresto di un'istanza dell'advisor personalizzato, l'indicazione di stato e report, la registrazione di informazioni cronologiche in un file di log e la registrazione dei risultati dell'advisor sul componente gestore.

Quando il codice di base del Load Balancer chiama un advisor personalizzato, avviene quanto segue.

1. Il codice di base del Load Balancer avvia una connessione con la macchina server.
2. Se il socket viene aperto, il codice di base chiama la funzione GetLoad dell'advisor.
3. La funzione GetLoad dell'advisor esegue le operazioni definite dall'utente per valutare lo stato del server, che include anche l'attesa di una risposta da parte del server. Dopo aver ricevuto la risposta, la funzione termina l'esecuzione.
4. Il codice di base del Load Balancer chiude il socket con il server e invia le informazioni sul carico al gestore. A seconda del fatto se l'advisor personalizzato opera in modalità normale o in modalità di sostituzione, il codice di base, al termine della funzione GetLoad, effettua, a volte, ulteriori calcoli.

Modalità normale e modalità di sostituzione

Gli advisor personalizzati possono essere progettati per interagire con il Load Balancer in modalità *normale* o in modalità di *sostituzione*.

La scelta della modalità di funzionamento viene specificata nel file dell'advisor personalizzato come un parametro nel metodo del costruttore. (Ciascun advisor opera solo in una delle seguenti modalità, in base al proprio progetto.)

In modalità normale, l'advisor personalizzato scambia i dati con il server, il codice dell'advisor di base programma lo scambio e calcola il valore del carico. Il codice di base invia questo valore del carico al gestore. L'advisor personalizzato restituisce il valore zero per indicare la riuscita o un valore negativo per indicare un errore.

Per specificare la modalità normale, impostare l'indicatore di sostituzione nel costruttore su *false*.

In modalità di sostituzione, il codice di base non esegue nessuna misurazione temporizzata. Il codice dell'advisor personalizzato esegue qualsiasi operazione specificata, in base a requisiti univoci e restituisce un numero di carico effettivo. Il codice di base accetta il numero del carico e lo invia, inalterato, al gestore. Per ottenere risultati migliori, normalizzare i numeri del carico tra 10 e 1000; 10 indica un server veloce e 1000 indica un server lento.

Per specificare la modalità di sostituzione, impostare l'indicatore di sostituzione nel costruttore su *true*.

Convenzioni di denominazione dell'advisor

I nomi dei file dell'advisor personalizzato hanno il formato ADV_*nome*.java, dove *nome* è il nome scelto per il proprio advisor. Il nome completo deve cominciare con il prefisso ADV_ in lettere maiuscole, mentre tutti i caratteri successivi devono essere minuscoli. Le lettere minuscole assicurano che il comando di esecuzione dell'advisor non distingue tra maiuscole e al minuscole.

In base alle convenzioni Java, il nome classe definito all'interno del file deve corrispondere al nome del file.

Compilazione

Gli advisor personalizzati devono essere scritti in linguaggio Java e compilati con un compilatore Java installato sulla macchina di sviluppo. Durante la compilazione si fa riferimento ai seguenti file:

- File dell'advisor personalizzato
- File di classe di base, ibmnd.jar, che si trova nella directory *percorso_installazione*/servers/lib

Durante la compilazione, la variabile d'ambiente classpath deve indicare il file dell'advisor personalizzato e il file delle classi di base. Un comando di compilazione potrebbe avere il seguente formato:

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

Questo esempio utilizza il percorso di installazione Linux e UNIX predefinito. Il file dell'advisor viene definito ADV_*nome*.java e viene memorizzato nella directory corrente.

L'output della compilazione è un file di classe, ad esempio, ADV_*nome* .class. Prima di avviare l'advisor, copiare il file di classe nella directory *percorso_installazione*/servers/lib/CustomAdvisors/.

Nota: gli advisor personalizzati possono essere compilati su un sistema operativo ed eseguiti su un altro sistema. Ad esempio, è possibile compilare l'advisor su un sistema Windows, copiare il file di classe risultante (in formato binario) su una macchina Linux ed eseguire in Linux l'advisor personalizzato.

Esecuzione di un advisor personalizzato

Per eseguire l'advisor personalizzato, è necessario innanzitutto copiare il file di classe dell'advisor sulla directory secondaria lib/CustomAdvisors/ sulla macchina del Load Balancer. Ad esempio, per un advisor personalizzato definito myping, il percorso del file è *percorso_installazione*/servers/lib/CustomAdvisors/ADV_myping.class

Configurare il Load Balancer, avviare la funzione del gestore e immettere il comando per avviare l'advisor personalizzato. L'advisor personalizzato viene specificato tramite il nome, escludendo il prefisso ADV_ e l'estensione del file:

```
dscontrol advisor start
myping numero_porta
```

Il numero porta specificato nel comando è la porta su cui verrà avviata una connessione con il server di destinazione.

Routine richieste

Come tutti gli advisor, un advisor personalizzato estende la funzionalità della classe di base dell'advisor, definita ADV_Base. L'advisor di base esegue la maggior parte delle funzioni dell'advisor, come ad esempio l'invio dei carichi al gestore per essere utilizzati nell'algoritmo di valutazione del gestore. Inoltre, tale advisor effettua le operazioni di connessione e chiusura del socket e fornisce i metodi di invio e di ricezione per l'uso da parte dell'advisor. L'advisor viene utilizzato unicamente per l'invio e la ricezione dei dati sulla porta specifica del server esaminato. I metodi TCP forniti con l'advisor di base sono programmati per calcolare il carico. Se necessario, un'indicatore all'interno del costruttore dell'advisor di base sostituisce il carico esistente con il nuovo carico restituito dall'advisor.

Nota: in base al valore impostato nel costruttore, l'advisor di base fornisce il carico all'algoritmo di valutazione a intervalli specifici. Se l'advisor non ha completato l'elaborazione e non può restituire un carico valido, l'advisor di base utilizza il carico inviato precedentemente.

Gli advisor dispongono dei seguenti metodi classe di base:

- Una routine del costruttore. Il costruttore chiama il costruttore della classe di base.
- Un metodo ADV_AdvisorInitialize. Questo metodo consente di eseguire ulteriori operazioni al termine dell'inizializzazione della classe di base.
- Una routine getLoad. La classe dell'advisor di base esegue l'apertura del socket; per completare il ciclo dell'advisor, la funzione getLoad deve generare soltanto richieste di invio e di ricezione appropriate.

I dettagli relativi a queste routine necessarie sono riportati più avanti in questa stessa sezione.

Ordine di ricerca

Gli advisor personalizzati vengono chiamati dopo aver effettuato la ricerca di advisor nativi o standard. Se il Load Balancer non trova un advisor specificato nell'elenco di advisor standard, consulta l'elenco di advisor personalizzati. Nella *Guida alla gestione per WebSphere Application Server Load Balancer* sono disponibili ulteriori informazioni sull'utilizzo degli advisor.

Percorso del file e di denominazione

Tenere presenti i seguenti requisiti per i nomi e i percorsi dell'advisor personalizzato.

- L'advisor personalizzato deve essere denominato in caratteri alfabetici minuscoli per far sì che i comandi immessi da un operatore su una riga comandi non distinguano tra maiuscole e minuscole. Il nome dell'advisor deve essere preceduto dal prefisso ADV_
- La classe dell'advisor personalizzato deve trovarsi all'interno della directory secondaria lib/CustomAdvisors. L'ubicazione predefinita di questa directory è /opt/ibm/edge/lb/servers/lib/CustomAdvisors sui sistemi Linux e UNIX e C:\Programmi\IBM\edge\lb\servers\lib\CustomAdvisors\ sui sistemi Windows.

Chiamate di funzione e metodi dell'advisor personalizzato

Costruttore (fornito dall'advisor di base)

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

sName
Il nome dell'advisor personalizzato.

sVersion
La versione dell'advisor personalizzato.

iDefaultPort
Il numero della porta su cui contattare il server nel caso in cui non fosse stato specificato nessun numero di porta nella chiamata.

iInterval
L'intervallo in cui l'advisor effettuerà la query ai server.

sDefaultLogFileName
Questo parametro è necessario ma non viene utilizzato. L'unico valore accettabile è una stringa nulla, ""

sostituzione
Indica la possibilità che questo advisor funzioni o meno in modalità di *sostituzione*. I valori possibili sono i seguenti:

- true – Sostituire il carico calcolato dal codice di base dell'advisor con il valore registrato dall'advisor personalizzato.
- false – Aggiungere il valore del carico registrato dall'advisor personalizzato al valore del carico calcolato dal codice di base dell'advisor.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

Questo metodo viene fornito per effettuare qualsiasi inizializzazione che potrebbe essere necessaria per l'advisor personalizzato. Tale metodo viene chiamato dopo l'avvio del modulo di base dell'advisor.

In molti casi, inclusi gli advisor standard, questo metodo non viene utilizzato e il relativo codice è composto unicamente da un'istruzione *return*. Questo metodo può essere utilizzato per chiamare il metodo suppressBaseOpeningSocket, valido solo dall'interno di questo metodo.

getLoad()

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

iConnectTime
Il tempo, espresso in millisecondi, necessario per completare la connessione. Questa misurazione del carico viene effettuata dal codice di base dell'advisor e inoltrata al codice dell'advisor personalizzato, il quale, durante la restituzione del valore del carico, può utilizzare o ignorare la misurazione. Se la connessione non riesce, questo valore viene impostato su -1.

chiamante
L'istanza della classe di base dell'advisor dove vengono forniti i metodi di base dell'advisor.

Chiamate di funzione disponibili per gli advisor personalizzati

I metodi, o funzioni, descritti nelle seguenti sezioni possono essere chiamati dagli advisor personalizzati. Questi metodi sono supportati dal codice di base dell'advisor.

Alcune di queste chiamate di funzione possono essere effettuate direttamente, ad esempio *nome_funzione*(), ma altre necessitano del prefisso `chiamante`. *Chiamante* rappresenta l'istanza dell'advisor di base che supporta l'advisor personalizzato eseguito.

ADVLOG()

La funzione ADVLOG consente a un advisor personalizzato di scrivere un messaggio di testo sul file di log dell'advisor di base. Di seguito viene riportato il formato:

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

logLevel
Il livello di stato in cui viene scritto il messaggio sul file di log. Il file di log dell'advisor è organizzato in fasi; ai messaggi più urgenti viene attribuito il livello di stato 0, mentre quelli meno urgenti ricevono numeri più elevati. Al tipo di messaggio verbose viene attribuito un livello di stato pari a 5. Questi livelli vengono utilizzati per controllare in tempo reale i tipi di messaggi ricevuti dall'utente. (Il comando **dscontrol** viene utilizzato per impostare il livello di precisione delle informazioni dei messaggi). Gli errori gravi devono essere registrati sempre sul livello 0.

messaggio
Il messaggio da scrivere sul file di log. Il valore di questo parametro è una stringa Java standard.

getAdvisorName()
La funzione getAdvisorName restituisce una stringa Java con parte del suffisso del nome dell'advisor personalizzato. Ad esempio, per un advisor definito ADV_cdload.java, questa funzione restituisce il valore cdload.

Questa funzione non utilizza alcun parametro.

Si noti che non è possibile modificare questo valore durante la creazione di un'istanza di un advisor.

getAdviseOnPort()
La funzione getAdviseOnPort restituisce il numero porta su cui è in esecuzione l'advisor personalizzato chiamante. Il valore di ritorno è un numero intero (int)Java e la funzione non utilizza alcun parametro.

Si noti che non è possibile modificare questo valore durante la creazione di un'istanza di un advisor.

caller.getCurrentServer()
La funzione getCurrentServer restituisce l'indirizzo IP del server corrente. Il valore di ritorno è una stringa Java in formato indirizzo IP, ad esempio 128.0.72.139

Generalmente, questo indirizzo viene modificato ogni volta che viene chiamato l'advisor personalizzato, in quanto il codice di base dell'advisor richiede tutte le macchine server in serie.

Questa funzione non utilizza alcun parametro.

caller.getCurrentCluster()
La chiamata della funzione getCurrentCluster restituisce l'indirizzo IP del cluster di server corrente. Il valore di ritorno è una stringa Java in formato indirizzo IP, ad esempio 128.0.72.139

Generalmente, questo indirizzo viene modificato ogni volta che viene chiamato l'advisor personalizzato, in quanto il codice di base dell'advisor richiede tutti cluster di server in serie.

Questa funzione non utilizza alcun parametro.

getInterval()
La funzione getInterval restituisce l'intervallo dell'advisor, vale a dire, il numero di secondi trascorsi tra i cicli dell'advisor. Questo valore è uguale al valore predefinito impostato nel costruttore dell'advisor personalizzato, a meno che il valore non sia stato modificato durante il runtime utilizzando il comando **dscontrol**.

Il valore di ritorno è un numero intero (int) Java. La funzione non utilizza alcun parametro.

caller.getLatestLoad()
La funzione getLatestLoad consente a un advisor personalizzato di ottenere il valore del carico più recente per un determinato oggetto server. I valori del carico vengono gestiti in tabelle interne dal codice di base dell'advisor e dal daemon del gestore.

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

Tutti e tre gli argomenti definiscono un oggetto server.

IP_cluster
L'indirizzo IP del cluster dell'oggetto server per cui richiamare il valore corrente del carico. Questo argomento deve essere una stringa Java in formato indirizzo IP, ad esempio, 245.145.62.81

porta
Il numero della porta dell'oggetto server per cui richiedere il valore corrente del carico.

IP_server
L'indirizzo IP dell'oggetto server per cui richiedere il valore corrente del carico. Questo argomento deve essere una stringa Java in formato indirizzo IP, ad esempio, 192.255.201.3

Il valore di ritorno è un numero intero.

- Un valore di ritorno positivo indica il valore effettivo del carico assegnato per l'oggetto richiesto.
- Il valore -1 indica che il server richiesto non è attivo.
- Il valore -2 indica che lo stato del server richiesto è sconosciuto.

Questa chiamata di funzione è utile se si desidera che il funzionamento di un protocollo o di una porta dipenda dal funzionamento di un altro protocollo o porta. Ad esempio, si potrebbe utilizzare questa chiamata di funzione in un advisor personalizzato che ha disabilitato un particolare server delle applicazioni se su quella stessa macchina è stato disabilitato il server Telnet.

caller.receive()

La funzione di ricezione ottiene informazioni dalla connessione socket.

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

Il parametro *response* è un buffer di stringhe in cui vengono introdotti i dati richiamati. Inoltre, la funzione restituisce un valore intero avente il seguente significato:

- 0 indica che i dati sono stati inviati correttamente.
- Un numero negativo indica un errore.

caller.send()

La funzione di invio utilizza la connessione socket stabilita per inviare un pacchetto di dati al server, utilizzando la porta specificata.

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

Il parametro *command* è una stringa contenente i dati da inviare al server. La funzione restituisce un valore intero avente il seguente significato:

- 0 indica che i dati sono stati inviati correttamente.
- Un numero negativo indica un errore.

suppressBaseOpeningSocket()

La funzione suppressBaseOpeningSocket consente a un advisor personalizzato di specificare se il codice dell'advisor di base apre un socket TCP sul server per conto dell'advisor personalizzato. Se l'advisor non utilizza la comunicazione diretta con il server per stabilire il suo stato, potrebbe essere necessario aprire questo socket.

Questa chiamata di funzione può essere emessa solo una volta e deve essere emessa dalla routine ADV_AdvisorInitialize.

La funzione non utilizza alcun parametro.

Esempi

Gli esempi riportati di seguito mostrano come implementare gli advisor personalizzati.

Advisor standard

Questo codice di origine di esempio è simile all'advisor HTTP del Load Balancer standard. Funziona nel modo seguente:

1. Viene emessa una richiesta di invio, un comando "HEAD/HTTP".
2. Si riceve una risposta. Le informazioni non vengono analizzate, ma la risposta provoca la chiusura del metodo getLoad.
3. Il metodo getLoad restituisce 0 per indicare la riuscita o -1 per indicare un errore.

Questo advisor opera in modalità normale, quindi la misurazione del carico si basa sul tempo utilizzato, espresso in millisecondi, necessario per eseguire le operazioni di apertura, invio, ricezione e chiusura del 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";

//--------
// Costruttore

  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;                                  // generalmente una routine vuota
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;       // inizializzare sull'inaccessibile

    iRc = caller.send(ADV_SEND_REQUEST);     // inviare la richiesta HTTP al
                                             // server
    if (0 <= iRc) {                          // se l'invio riesce
      StringBuffer sbReceiveData = new StringBuffer("");   // assegnare un buffer
                                                           // per la risposta
      iRc = caller.receive(sbReceiveData);   // ricevere il risultato

      // se necessario, analizzare ora il risultato
```

```
      if (0 <= iRc) {          // se la ricezione è riuscita
        iLoad = 0;             // restituire 0 per indicare la riuscita
      }                        // (il valore del carico dell'advisor viene ignorato dalla
    }                          //  base in modalità normale)
    return iLoad;
  }
}
```

Advisor flusso laterale

Questo esempio illustra l'eliminazione del socket standard aperto dall'advisor di base. Al contrario, questo advisor apre un socket Java flusso laterale per interrogare un server. Questa procedura può essere utile per i server che utilizzano una porta differente dal quella utilizzata dal traffico client normale per ricevere una query dell'advisor.

In questo esempio, un server è in ascolto sulla porta 11999 e quando viene interrogato restituisce un valore di carico con un int esadecimale pari a ″4″. Questo esempio viene eseguito in modalità di sostituzione, vale a dire, l'ultimo parametro del costruttore advisor viene impostato su true e il codice di base dell'advisor utilizza il valore del carico restituito piuttosto che il tempo trascorso.

Si noti la chiamata a supressBaseOpeningSocket() nella routine di inizializzazione. Non è richiesta l'eliminazione del socket di base se non verranno inviati dati. Ad esempio, è possibile che si desideri aprire il socket per verificare che l'advisor possa contattare il server. Esaminare attentamente le esigenze della propria applicazione prima di optare per questa scelta.

```
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;

  // creare una matrice di byte con il messaggio di richiesta del carico
  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);         // il parametro della modalità di sostituzione è true
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // indicare al codice di base di non aprire
                                   // il socket standard
    return;
  }

//--------
```

```
// getLoad

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;   // la porta su cui comunicare con il server

    string sServer = caller.getCurrentServer();   // indirizzo del server da interrogare
    try {
      socket soServer = new Socket(sServer, iControlPort);  // aprire il socket sul
                                                             // server
      DataInputStream disServer = new DataInputStream(
                                     soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                      soServer.getOutputStream());

      int iRecvTimeout = 10000;  // impostare il timeout (in millisecondi)
                                 // per la ricezione dei dati
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // inviare un messaggio al server
      dosServer.flush();

      iLoad = disServer.readByte();   // ricevere la risposta dal server

    } catch (exception e) {
      system.out.println("Caught exception " + e);
    }
    return iLoad;    // restituire il carico registrato dal server
  }
}
```

Advisor a due porte

Questo esempio di advisor personalizzato mostra la capacità di rilevare l'errore di una porta di un server, sia in base al suo stato sia in base allo stato di un daemon server differente in esecuzione su un'altra porta sulla stessa macchina server. Ad esempio, se il daemon HTTP sulla porta 80 arresta la risposta, è possibile che si desideri arrestare il traffico di instradamento per il daemon SSL sulla porta 443.

Questo advisor è più aggressivo rispetto agli advisor standard, poiché considera non operativi tutti i server che non inviano una risposta e li contrassegna come *inattivi*. Gli advisor standard considerano i server inerti come molto lenti. Questo advisor contrassegna un server come inattivo sia per la porta HTTP sia per la porta SSL, in caso di mancata risposta da entrambe le porte.

Per utilizzare questo advisor personalizzato, l'amministratore avvia due istanze dell'advisor: una sulla porta HTTP e una sulla porta SSL. L'advisor crea le istanze di due tabelle hash globali statiche, una per HTTP e una per SSL. Ciascun advisor tenta di comunicare con il relativo daemon server e memorizza i risultati di questo evento nella tabella hash correlata. Il valore restituito da ciascun advisor alla classe dell'advisor di base dipende sia dalla capacità di comunicare con il server daemon sia dalla capacità dell'advisor partner di comunicare con il proprio daemon.

Vengono utilizzati i seguenti metodi personalizzati.

- ADV_nte() è un oggetto contenitore semplice per conservare le informazioni su un server. Questi oggetti vengono memorizzati nella tabella hash come elementi della tabella. Ciascun oggetto dispone di un indicatore di data e ora che viene utilizzato per stabilire se l'elemento è attuale o meno.
- putNte() e getNte() sono metodi sincronizzati che verificano che le due istanze dell'advisor accedano alla tabella hash in maniera controllata.

- getLoadHTTP è un metodo che verifica la sensibilità di un server HTTP. Si tratta di una routine di basso livello che non raccoglie né utilizza informazioni su SSL.
- getLoadSSL() è un metodo che verifica la sensibilità di un server SSL. Si tratta di una routine di basso livello che non raccoglie né utilizza informazioni su HTTP.
- getLoad() è la routine del punto d'ingresso per questo advisor personalizzato. È in grado di gestire entrambi i protocolli e di memorizzare e ricercare le informazioni dalla tabella hash. Questa è la routine che collega le due porte.

Vengono rilevate le seguenti condizioni di errore.

- Macchina del server che non risponde — Le classi dell'advisor di base inviano periodicamente un segnale di ping all'indirizzo del server. Se l'indirizzo non è raggiungibile, le classi dell'advisor di base contrassegnano il server come inattivo. Non viene chiamata nessuna delle due istanze dell'advisor personalizzato ed entrambi i server su quella macchina vengono contrassegnati come inattivi.
- Un daemon su una macchina server non risponde, ma l'altro è in funzione — Quando il codice di base tenta di aprire un socket con il server, la connessione viene rifiutata e l'advisor di base di questo protocollo contrassegna il server come inattivo. Il codice dell'advisor personalizzato per quel protocollo non viene chiamato. Nonostante l'advisor personalizzato per l'altro protocollo continui a comunicare con il relativo server, esso viene a conoscenza, tramite la tabella hash, che l'altro advisor personalizzato non può comunicare con il server daemon. Tuttavia, l'advisor del secondo protocollo contrassegna il relativo server come inattivo.
- Un daemon non invia una risposta, ma l'altro daemon sì — L'advisor personalizzato per il protocollo che non risponde rileva l'errore di comunicazione, contrassegna il server come inattivo e memorizza i dati nella tabella hash. L'advisor personalizzato per l'altra porta viene a conoscenza delle informazioni contenute nella tabella hash e contrassegna il server come inattivo.

Questo esempio viene scritto per collegare la porta 80 per HTTP e la porta 443 per SSL, ma può essere adattato per qualsiasi combinazione di porte.

```
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;

//--------
// Defin. l'elemento tab. per le tab. hash utilizzate in questo advisor personalizzato

class ADV_nte implements Cloneable {
  private string  sCluster;
  private int     iPort;
  private string  sServer;
  private int     iLoad;
  private Date    dTimestamp;

//--------
// costruttore

  public ADV_nte(string sClusterIn, int iPortIn, string sServerIn,
                 int iLoadIn) {
    sCluster = sClusterIn;
    iPort = iPortIn;
```

```
    sServer = sServerIn;
    iLoad = iLoadIn;
    dTimestamp = new Date();
  }

//--------
// verificare se questo elemento è corrente o scaduto
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();   // impostare la durata come
                                                        // 3 cicli di advisor
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

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

//--------
// accessori valore

  public int getLoadValue() { return iLoad; }

//--------
// clone (evita la corruzione tra i thread)

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

}

//--------
// definire l'advisor personalizzato

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;

  //--------
  // definire le tab. per conservare le info cronologiche specifiche della porta

  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";

  //--------
  // creare una matrice di byte con il messaggio hello del client SSL

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
```

```
    (byte)0x01,               // client hello
    (byte)0x03, (byte)0x00,   // versione SSL
    (byte)0x00, (byte)0x03,   // lunghezza specifica di codifica (byte)
    (byte)0x00, (byte)0x00,   // lunghezza ID di sessione (byte)
    (byte)0x00, (byte)0x10,   // lunghezza dati richiesta (byte)
    (byte)0x00, (byte)0x00, (byte)0x03,   // specifica di codifica
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)0x20,  // dati richiesta
    (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 };

  //--------
  // costruttore

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);    // false = load balancer programma la risposta
    setAdvisor ( this );
  }

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

  public void ADV_AdvisorInitialize() {
    return;
  }

  //--------
  // routine di accesso PUT e GET sincronizzate per le tabelle 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 - determinare il carico HTTP in base alla risposta del server

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

    int iRc = caller.send(ADV_HTTP_REQUEST_STRING);  // inviare il msg di richiesta
                                                     // al server
    if (0 <= iRc) {           // la richiesta ha restituito un errore?
      StringBuffer sbReceiveData = new StringBuffer("")    // assegnare un buffer
                                                           // per la risposta
      iRc = caller.receive(sbReceiveData);    // ricevere la risposta dal server

      if (0 <= iRc) {             // la ricezione ha restituito un errore?
        if (0 < sbReceiveData.length()) {      // vi sono dati?
          iLoad = SUCCESS;        // ignorare i dati richiamati e
                                  // restituire il codice di riuscita
        }
      }
    }
    return iLoad;
  }
```

```
//--------
// getLoadSSL() - determinare il carico SSL in base alla risposta del server

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

  int iSocket = caller.getAdvisorSocket();   // inviare una rich. esadec. al server
  CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                abClientHello);
  int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

  if (0 <= iRc) {           // la richiesta ha restituito un errore?
    StringBuffer sbReceiveData = new StringBuffer("");  // assegnare un buffer
                                                        // per la risposta
    iRc = caller.receive(sbReceiveData);     // ottenere una risposta dal
                                             // server
    if (0 <= iRc) {             // la ricezione ha restituito un errore?
      if (0 < sbReceiveData.length()) {      // vi sono dati?
        iLoad = SUCCESS;   // ignorare dati richiamati e restituire cod. riuscita
      }
    }
  }
  return iLoad;
}

//--------
// getLoad - associare i risultati dei metodi HTTP e SSL

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

  String sCluster = caller.getCurrentCluster();   // indirizzo corrente cluster
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;     // chiave tabella hash

  if (ADV_TWOP_PORT_HTTP == iPort) {              // gestire un server HTTP
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // richiamare carico per HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);  // salvare le info sul carico
                                                    // su SSL
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // richiamare le info
                                                          // su SSL
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) {         // controllare data e ora
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // l'SSL è
                                                                 // in funzione?
          iLoad = iLoadHTTP;
        } else {    // SSL non funziona, quindi contras. srvr HTTP come inattivo
          iLoad= ADV_HOST_INACCESSIBLE;
        }
      } else {      // le informazioni su SSL sono scadute, quindi contrassegnare
                    // il server HTTP come inattivo
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {        // nessuna informazione sul carico relativa all'SSL, registrare
                    // i risultati di getLoadHTTP()
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {           // gestire un server SSL
    iLoadSSL = getLoadSSL(iConnectTime, caller);   // ricevere il carico per SSL
```

```
      ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
      putNte(htTwopSSL, "SSL", sHashKey, nteSSL);   // salvare info sul carico SSL.

      ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);   // ottenere info su
                                                               // su SSL
      if (null != nteHTTP) {
        if (true == nteHTTP.isCurrent(this)) {       // controllare data e ora
          if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {  // HTTP è
                                                                  // in funzione?
            iLoad = iLoadSSL;
          } else {   // srvr HTTP non in funzione, quindi contras. SSL come inattivo
            iLoad = ADV_HOST_INACCESSIBLE;
          }
        } else {     // info da HTTP scadute, quindi contras. SSL come inattivo
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {       // nessuna informazione su HTTP, registrare
                     // i risultati di getLoadSSL()
        iLoad = iLoadSSL;
      }
    }

  //--------
  // handler dell'errore

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

Advisor di WebSphere Application Server

Un advisor personalizzato di esempio per WebSphere Application Server è incluso nella directory *percorso_installazione*/servers/samples/CustomAdvisors/. Il codice completo non viene duplicato in questo documento.

- ADV_was.java è il file del codice di origine dell'advisor compilato ed eseguito sulla macchina del Load Balancer.
- LBAdvisor.java.servlet è il codice di origine del servlet che deve essere ridenominato in LBAdvisor.java, compilato ed eseguito sulla macchina di WebSphere Application Server.

L'advisor completo è leggermente più complesso dell'esempio. Esso aggiunge una routine di analisi specializzata più compatta rispetto all'esempio StringTokenizer mostrato in precedenza.

La parte più complessa del codice di esempio è rappresentata dal servlet Java. Tra gli altri metodi, il servlet contiene due metodi richiesti dalla specifica del servlet: init() e service(), nonché un metodo, run(), richiesto dalla classe Java.lang.thread.

- init() viene chiamato una volta dal motore servlet al momento dell'inizializzazione. Questo metodo crea un thread definito _checker che viene eseguito indipendentemente dalle chiamate provenienti dall'advisor ed è inattivo per un determinato periodo di tempo prima di riprendere l'elaborazione del loop.
- service() viene chiamato dal motore servlet ogni volta che viene richiamato il servlet. In questo caso, il metodo viene chiamato dall'advisor. Il metodo service() invia un flusso di caratteri ASCII a un flusso di output.
- run() contiene la parte principale dell'esecuzione del codice. Viene chiamato dal metodo start() che, a sua volta, viene chiamato dall'interno del metodo init().

Di seguito vengono illustrati i frammenti rilevanti del codice servlet.

```
...

  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;
  }

  ...
```

Utilizzo dei dati restituiti dagli advisor

Nel caso in cui si utilizzi una chiamata standard a una parte esistente del server delle applicazioni o si aggiunga una nuova parte di codice alla controparte lato server del proprio advisor personalizzato, è possibile che si desideri esaminare i valori del carico restituiti e modificare il funzionamento del server. La classe Java StringTokenizer e i relativi metodi, rendono più semplice questo controllo.

Il contenuto di un comando HTTP tipico potrebbe essere `GET /index.html HTTP/1.0`

Una risposta tipica a questo comando potrebbe essere la seguente.

```
HTTP/1.1 200 OK
Date: Mon, 20 November 2000 14:09:57 GMT
Server: Apache/1.3.12 (Linux e 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>Questo server Web esegue Apache 1.3.12.
<P><HR>
<P><IMG SRC="apache_pb.gif" ALT="">
</BODY></HTML>
```

Gli elementi di interesse sono contenuti nella prima riga, in particolare il codice di ritorno HTTP.

La specifica HTTP classifica i codici di ritorno riepilogati nel modo seguente:

- i codici di ritorno 2xx indicano la riuscita
- i codici di ritorno 3xx indicano i reindirizzamenti
- i codici di ritorno 4xx indicano errori del client
- i codici di ritorno 5xx indicano errori del server

Se si conoscono perfettamente quali sono codici che il server può restituire, non è necessario che il codice sia così dettagliato come mostrato in questo esempio. Tuttavia, tenere presente che limitando i codici di ritorno che possono essere rilevati si potrebbe limitare la flessibilità futura del programma.

L'esempio riportato di seguito è un programma Java autonomo che contiene un client HTTP minimo. L'esempio richiama un programma di analisi semplice e generico per l'esame delle risposte 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());
    }
  }
}
```

Informazioni particolari

Prima edizione (Maggio 2006)

Queste informazioni sono state sviluppate per prodotti e servizi offerti negli Stati Uniti d'America.

IBM non può offrire in altri paesi i prodotti, i servizi o le funzioni descritti in questo documento. Per le informazioni sui prodotti ed i servizi disponibili al momento nella propria area, rivolgersi al rivenditore IBM locale. Qualunque riferimento relativo a prodotti, programmi o servizi IBM non implica che solo quei prodotti, programmi o servizi IBM possano essere utilizzati. In sostituzione a quelli forniti da IBM, possono essere usati prodotti, programmi o servizi funzionalmente equivalenti che non comportino violazione dei diritti di proprietà intellettuale o di altri diritti di IBM. È tuttavia responsabilità dell'utente valutare e verificare la funzionalità di tali prodotti, programmi e servizi non IBM.

IBM può avere brevetti o richieste di brevetti in corso relativi a quanto trattato nella presente pubblicazione. La fornitura di questa pubblicazione non implica la concessione di alcuna licenza su di essi. Chi desiderasse ricevere informazioni relative a licenze può rivolgersi per iscritto a:

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

Per domande sulle licenze relative a informazioni DBCS, contattare IBM Intellectual Property Department nel proprio paese oppure scrivere a:

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

Il seguente paragrafo non è valido per il Regno Unito o per tutti i paesi le cui leggi nazionali siano in contrasto con le disposizioni in esso contenute:

INTERNATIONAL BUSINESS MACHINES CORPORATION FORNISCE QUESTA PUBBLICAZIONE NELLO STATO IN CUI SI TROVA SENZA ALCUNA GARANZIA, ESPLICITA O IMPLICITA, IVI INCLUSE EVENTUALI GARANZIE DI COMMERCIABILITÀ' ED IDONEITÀ AD UNO SCOPO PARTICOLARE. Alcuni stati non consentono la rinuncia a garanzie esplicite o implicite in determinate transazioni, quindi, la presente dichiarazione potrebbe non essere a voi applicabile.

Questa pubblicazione potrebbe contenere imprecisioni tecniche o errori tipografici. Le informazioni incluse in questo documento non vengono modificate su base periodica; tali modifiche verranno incorporate nelle nuove edizioni della pubblicazione. IBM si riserva il diritto di apportare miglioramenti e/o modifiche ai prodotti e/o ai programmi descritti nella guida in qualsiasi momento e senza preavviso.

Tutti i riferimenti a siti Web non dell'IBM contenuti in questo documento sono forniti solo per consultazione. I materiali contenuti in questi siti Web non fanno parte di questo prodotto e il loro utilizzo è a discrezione dell'utente.

IBM può utilizzare o distribuire qualsiasi informazione fornita dall'utente nel modo più appropriato senza incorrere in alcuna obbligazione.

Coloro che detengono la licenza su questo programma e desiderano avere informazioni su di esso allo scopo di consentire: (i) uno scambio di informazioni tra programmi indipendenti ed altri (compreso questo) e (ii) l'uso reciproco di tali informazioni, dovrebbero rivolgersi a:

IBM Europe
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222-9183
Deutchland

Queste informazioni possono essere rese disponibili secondo condizioni contrattuali appropriate, compreso, in alcuni casi, l'addebito di un canone.

Il programma su licenza descritto in queste informazioni e tutto il materiale su licenza ad esso relativo sono forniti da IBM nel rispetto delle condizioni previste dall'accordo IBM International Program License Agreement o da accordi equivalenti.

Tutti i dati relativi alle prestazioni contenuti in questa pubblicazione sono stati determinati in ambiente controllato. Pertanto, i risultati ottenuti in ambienti operativi diversi possono variare in modo considerevole. Alcune misure potrebbero essere state fatte su sistemi di livelli di sviluppo per cui non si garantisce che queste saranno uguali su tutti i sistemi disponibili. Inoltre, alcune misure potrebbero essere state ricavate mediante estrapolazione. I risultati possono quindi variare. Gli utenti di questa pubblicazione devono verificare che i dati siano applicabili al loro specifico ambiente.

Le informazioni relative a prodotti non IBM sono state ottenute dai fornitori di tali prodotti. IBM non ha verificato tali prodotti e, pertanto, non può garantirne l'accuratezza delle prestazioni o la compatibilità. Eventuali commenti relativi alle prestazioni dei prodotti non IBM devono essere indirizzati ai fornitori di tali prodotti.

Tutte le dichiarazioni riguardanti la futura direzione o le intenzioni dell'IBM sono soggette a sostituzione o al ritiro senza preavviso e rappresentano unicamente scopi e obiettivi della IBM stessa.

Queste informazioni contengono esempi di dati e report utilizzati quotidianamente nelle operazioni aziendali. Per meglio illustrarli, tali esempi contengono nomi di persone, società, marchi e prodotti. Tutti i nomi contenuti nel manuale sono fittizi e ogni riferimento a nomi ed indirizzi reali è puramente casuale.

Se si stanno visualizzando queste informazioni in formato elettronico, le illustrazioni a colori e le foto potrebbero non essere visualizzate.

Marchi

I seguenti termini sono marchi di IBM Corporation, negli Stati Uniti, in altri paesi o in entrambi:

- AIX
- IBM
- ViaVoice
- Raccolta di produzione connessioni argomenti

Java e tutti i marchi basati su Java sono di Sun Microsystems, Inc. negli Stati Uniti, in altri paesi o in entrambi.

Microsoft, Windows, Windows NT e il logo Windows sono marchi di Microsoft Corporation negli Stati Uniti, in altri paesi o in entrambi.

Intel, Intel Inside (logos), MMX e Pentium sono marchi di Intel Corporation negli Stati Uniti, in altri paesi o in entrambi.

UNIX è un marchio registrato di The Open Group negli Stati Uniti e in altri paesi.

Linux è un marchio di Linus Torvalds, negli Stati Uniti, in altri paesi o in entrambi.

Altri nomi di società, prodotti o servizi possono essere marchi o marchi di servizi di altre società.

Indice analitico

L

M

O

P

R

S

T

W

Printed in Denmark by IBM Danmark A/S

GC13-3368-02

Spine information:

WebSphere Application Server

Guida alla programmazione per Edge Components

Versione 6.1

GC13-3368-02