WebSphere Application Server

Edge Components プログラミング・ガイド

バージョン 6.0.1

GC88-7045-01
(英文原典：GC31-6856-01)

WebSphere Application Server

Edge Components プログラミング・ガイド

バージョン 6.0.1

GC88-7045-01
(英文原典：GC31-6856-01)

ご注意

本書および本書で紹介する製品をご使用になる前に、 63 ページの『特記事項』に記載されている情報をお読みください。

この版は、以下のプログラムに適用されます。

WebSphere Application Server、バージョン 6.0.1

また、新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

http://www.ibm.com/jp/manuals/main/mail.html

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

http://www.ibm.com/jp/manuals/ の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原　典： GC31-6856-01
WebSphere Application Server
Programming Guide for Edge Components
Version 6.0.1

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.2

目次

図

本書について

このセクションでは、本書「*WebSphere® Application Server Edge Components プログラミング・ガイド*」の目的、編成、および規約について説明します。

本書の対象読者

本書では、WebSphere Application Server、バージョン 6.0.1 の Edge Components をカスタマイズするために使用できるアプリケーション・プログラミング・インターフェース (API) について説明します。この情報は、プラグイン・アプリケーションを作成したり、その他のカスタマイズを行うプログラマーを対象としています。ネットワーク設計者およびシステム管理者にとっても、この情報は可能なカスタマイズ・タイプの指示として役立つ場合があります。

前提知識

本書の情報を使用するには、 Java™ または C 言語 (使用予定の API に応じる) を使用したプログラミング手順を理解している必要があります。公開された各インターフェースで使用可能なメソッドおよび構造について文書化していますが、独自のアプリケーションの構成方法、ユーザー・システム用のコンパイル方法、および検査方法を知っている必要があります。一部のインターフェースにはサンプル・コードが提供されていますが、このサンプルは独自のアプリケーションを構成するための例としてのみ提供されています。

本書で使用されている規則と用語

本書では、以下のような書体およびキー操作の規則を使用しています。

表 1. 本書の規則

規則	意味
太字	グラフィカル・ユーザー・インターフェース (GUI) に関しては、太字は、メニュー、メニュー項目、ラベル、ボタン、アイコン、およびフォルダーを示します。また、太字にしないと周りのテキストと混同される恐れがあるコマンド名を強調するためにも使用されます。
モノスペース	コマンド・プロンプトから入力する必要のあるテキストを示します。また、モノスペースは、画面上のテキスト、コード例、およびファイルからの引用も示します。
イタリック	指定する必要のある可変値を示します (例: *fileName* にファイルの名前を指定します)。イタリックは、強調表示および書名の表示にも使用されます。
Ctrl-*x*	*x* がキーの名前である場合、制御文字のシーケンスを示します。例えば、 Ctrl-c は、Ctrl キーを押しながら c キーを押すという意味になります。
Return	Return、Enter または左矢印が表示されていたキーを指します。
%	Linux および UNIX® のコマンド・シェル・プロンプト (root 権限を必要としないコマンド用) を示します。
#	Linux および UNIX のコマンド・シェル・プロンプト (root 権限を必要とするコマンド用) を示します。

表 1. 本書の規則 (続き)

規則	意味
C:¥	Windows のコマンド・プロンプトを示します。
コマンド入力	コマンドを「入力」する「発行」するよう指示されたときは、コマンドを入力してから Return を押します。例えば、「**ls** コマンドを入力してください」という指示は、コマンド・プロンプトに **ls** と入力してから Return を押すという意味になります。
[ ]	構文記述内のオプション項目を囲みます。
{ }	構文記述内の、項目を選択する必要があるリストを囲みます。
|	構文記述の中で { } (中括弧) で囲まれた選択項目リストにある項目を区切るために使用されます。
...	構文記述内の省略記号は、前の項目を 1 回以上繰り返すことができることを示します。例にある省略記号は、簡潔にするために例から情報が省略されていることを示します。

アクセシビリティー

アクセシビリティー機能は、運動障害または視覚障害など身体に障害を持つユーザーがソフトウェア・プロダクトを快適に使用できるようにサポートします。

WebSphere Application Server、バージョン 6.0.1 の主なアクセシビリティー機能は次のとおりです。

- スクリーン・リーダー・ソフトウェアおよびディジタル・スピーチ・シンセサイザーを使用して、画面に表示された内容を聞くことができます。また、 IBM® ViaVoice™ などの音声認識ソフトウェアを使用して、データを入力し、ユーザー・インターフェースをナビゲートすることもできます。
- マウスの代わりにキーボードを使用して機能を操作できます。
- 提供されているグラフィカル・インターフェースの代わりに標準テキスト・エディターまたはコマンド行インターフェースを使用して、Application Server 機能を構成および管理できます。特定機能のアクセシビリティーに関する詳細については、これらの機能についての資料を参照してください。

関連資料および Web サイト

- *Edge Components 概念、計画とインストール*, GC88-7036-00
- *Caching Proxy 管理ガイド*, GC88-7050-00
- *Load Balancer 管理ガイド*, GC88-7053-00
- IBM home Web サイト *www.ibm.com/*
- IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/*
- IBM WebSphere Application Server ライブラリー Web サイト *www.ibm.com/software/webservers/appserv/library.html*
- IBM WebSphere Application Server サポート Web サイト *www.ibm.com/software/webservers/appserv/support.html*
- IBM WebSphere Application Server インフォメーション・センター *www.ibm.com/software/webservers/appserv/infocenter.html*

- IBM WebSphere Application Server Edge Components インフォメーション・センター *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

第 1 章 Edge Components カスタマイズの概説

本書では、WebSphere Application Server の Edge Components に提供されるアプリケーション・プログラム・インターフェース (API) について説明します。(WebSphere Application Server の Edge Components には、Caching Proxy と Load Balancer が含まれます。) 管理者は提供されるいくつかのインターフェースを使用してインストールをカスタマイズし、Edge Components 相互間の対話方法を変更するか、または他のソフトウェア・システムとの対話を可能にすることができます。

注: Caching Proxy は、Itanium 2 および AMD Opteron 64 ビット・プロセッサーで稼働するプラットフォームを除く、すべてのサポートされるプラットフォームで使用可能です。

本書の API はいくつかのカテゴリーを扱っています。

Caching Proxy のカスタマイズ

Caching Proxy にはその処理シーケンスに書き込まれるいくつかのインターフェースがあり、このシーケンスでカスタム処理を標準処理用に追加または置換できます。実行できるカスタマイズには、以下のようなタスクの変更および拡大が含まれます。

- クライアント認証
- 許可要求
- 物理ファイル・パスへの URL 変換
- サービス要求
- ロギング
- エラー条件への応答

カスタム・アプリケーション・プログラムは、Caching Proxy プラグインとしても知られていますが、プロキシー・サーバーの処理シーケンス中の、事前に定義されたポイントで呼び出されます。

Caching Proxy API は、システム機能をインプリメントするために使用されます。例えば、プロキシー・サーバーの LDAP サポートはプラグインとしてインプリメントされます。

インターフェースの詳細については、 3 ページの『第 2 章 Caching Proxy API』で説明します。これにはプラグイン・プログラムを使用する プロキシー・サーバーの構成ステップが含まれます。

Load Balancer のカスタマイズ

Load Balancer は、ユーザー独自のアドバイザーを作成することによってカスタマイズできます。アドバイザーはサーバー上で実際のロード測定を実行します。カスタム・アドバイザーがあれば、ロードを測定するためのシステムに関連する、自分で用意したメソッドを使用することができます。これは特に、カスタマイズ済みまたは所有 Web サーバー・システムがある場合に重要です。

カスタム・アドバイザーの作成および使用の詳細については、43 ページの『第 3 章 カスタム・アドバイザー』で説明します。これには、アドバイザーのサンプル・コードも含まれます。

サンプル・コードの検索

上記の API のサンプル・コードは、Edge Components CD-ROM の samples ディレクトリーに入っています。 WebSphere Application Server Web サイト www.ibm.com/software/webservers/appserv/ からこの他のコード・サンプルを入手できます。

第 2 章 Caching Proxy API

このセクションでは、Caching Proxy アプリケーション・プログラミング・インターフェース (API) の概念のほか、その利点や機能について説明します。

注: Caching Proxy は、Itanium 2 および AMD Opteron 64 ビット・プロセッサーで稼働するプラットフォームを除く、すべてのサポートされるプラットフォームで使用可能です。

Caching Proxy API の概説

この API は、Caching Proxy へのインターフェースであり、プロキシー・サーバーの基本機能を拡張することができます。拡張機能またはプラグインを作成し、次の例を含むカスタマイズされた処理を実行できます。

- 基本認証ルーチンを拡張するか、あるいはサイト特有のプロセスと置き換える。
- エラー処理ルーチンを追加して、問題を追跡したり、重大な状態に対して警報を出す。
- サーバー参照やユーザー・エージェント・コードなど、要求元クライアントから送られる情報を検出し追跡する。

Caching Proxy API には、次のような利点があります。

- **効率**
  - API は、Caching Proxy が使用するスレッド化処理システム用に特別に設計されています。
- **柔軟性**
  - API には豊富で用途の広い関数が入っています。
  - API はプラットフォームから独立し、言語的に中立です。これは、すべての Caching Proxy プラットフォームで稼働するため、それらのプラットフォームでサポートされている多くのプログラム言語でプラグイン・アプリケーションを作成できます。
- **使いやすさ**
  - 単純データ型が値ではなく、参照によって渡されます (例えば long * や char * など)。
  - 関数ごとにパラメーター数が固定されています。
  - C 言語バインディングが組み込まれています。
  - プラグインは、割り振り済みメモリーに影響を与えません。プラグイン・アプリケーションは、他の Caching Proxy プロセスとは無関係にメモリーの割り振りおよび解放を行います。

API プログラム作成の一般手順

Caching Proxy プラグイン・プログラムを作成するには、あらかじめプロキシー・サーバーの機能を理解しておく必要があります。プロキシー・サーバーの振る舞いは、いくつかの別個の処理ステップに分けることができます。これらのそれぞれのステップで、API を使用して独自にカスタマイズした機能を使用することが可能です。例えば、ある処理をクライアント要求の読み取り後、他の処理を行う前に行う、あるいは特殊ルーチンを認証時に実行してから、要求ファイルの送信後に再度行うなどを決めます。

事前定義機能のライブラリーには API が備わっています。ご使用のプラグイン・プログラムは、プロキシー・サーバー・プロセス (例えば、要求の操作、要求ヘッダーの読み取りまたは書き込み、あるいはプロキシー・サーバーのログへの書き込みなど) と対話するために事前定義 API 関数を呼び出すことができます。これらの関数を、自分で作成したプラグイン関数と混同しないでください。プラグイン関数は、プロキシー・サーバーによって呼び出されます。事前定義関数については、18 ページの『事前定義関数およびマクロ』で説明しています。

サーバー構成ファイル内の対応する Caching Proxy API ディレクティブを使用して、適切なステップでプラグイン関数を呼び出すように、プロキシー・サーバーに指示します。これらのディレクティブについては、24 ページの『API ステップの Caching Proxy 構成ディレクティブ』で説明しています。

本書には、次の内容が含まれています。

- カスタマイズ可能な Caching Proxy ステップの基本的な説明 (『サーバー・プロセス・ステップ』を参照)
- プラグイン作成のためのガイドライン (8 ページの『ガイドライン』を参照)
- サーバーが実行する各ステップに追加できるカスタマイズされた関数のプロトタイプ、およびその戻りコード (10 ページの『プラグイン関数プロトタイプ』を参照)
- プラグイン内から呼び出すことのできる事前定義関数とマクロの定義、およびその戻りコード (18 ページの『事前定義関数およびマクロ』を参照)
- Caching Proxy API 構成ディレクティブ (24 ページの『API ステップの Caching Proxy 構成ディレクティブ』を参照)

以下のコンポーネントとプロシージャーを使用して独自の Caching Proxy プラグイン・プログラムを作成することができます。

サーバー・プロセス・ステップ

プロキシー・サーバーの基本操作は、そのフェーズでサーバーが実行する処理のタイプに基づいていくつかのステップに分けることができます。各ステップには、プログラムの指定された部分を実行できる接続点があります。Caching Proxy 構成ファイル (ibmproxy.conf) に API ディレクティブを追加することによって、特定のステップで呼び出すプラグイン関数を指定します。そのステップに複数のディレクティブを組み込むことによって、特定のプロセス・ステップで複数のプラグイン関数を呼び出すことができます。

いくつかのステップはサーバー要求プロセスの一部です。言い換えると、プロキシー・サーバーは、要求を処理するたびにこれらのステップを実行します。その他のステップは要求処理とは無関係に実行されます。つまり、要求が処理されているかどうかにかかわらず、サーバーはこれらのステップを実行します。

コンパイルされたプログラムは、オペレーティング・システムに応じて、いずれかの共用オブジェクト (例えば、DLL または .so ファイル) に入っています。サーバーは、その要求プロセス・ステップを進めながら、いずれかの関数が要求の処理を終えたことを示すまでは、各ステップに関連付けられたプラグイン関数を呼び出します。特定のステップのプラグイン関数が複数ある場合は、これらの関数は構成ファイル内にあるディレクティブの順序で呼び出されます。

要求がプラグイン関数によって処理されない (そのステップに Caching Proxy API ディレクティブを組み込まなかったか、そのステップのプラグイン関数が HTTP_NOACTION を戻した) 場合には、サーバーはそのステップのデフォルト・アクションを実行します。

注 : これは、Service ステップを除くすべてのステップにあてはまります。Service ステップにはデフォルト・アクションはありません。

6 ページの図 1 は、プロキシー・サーバー・プロセスのステップを表し、要求処理に関連するステップの処理順序を定義しています。

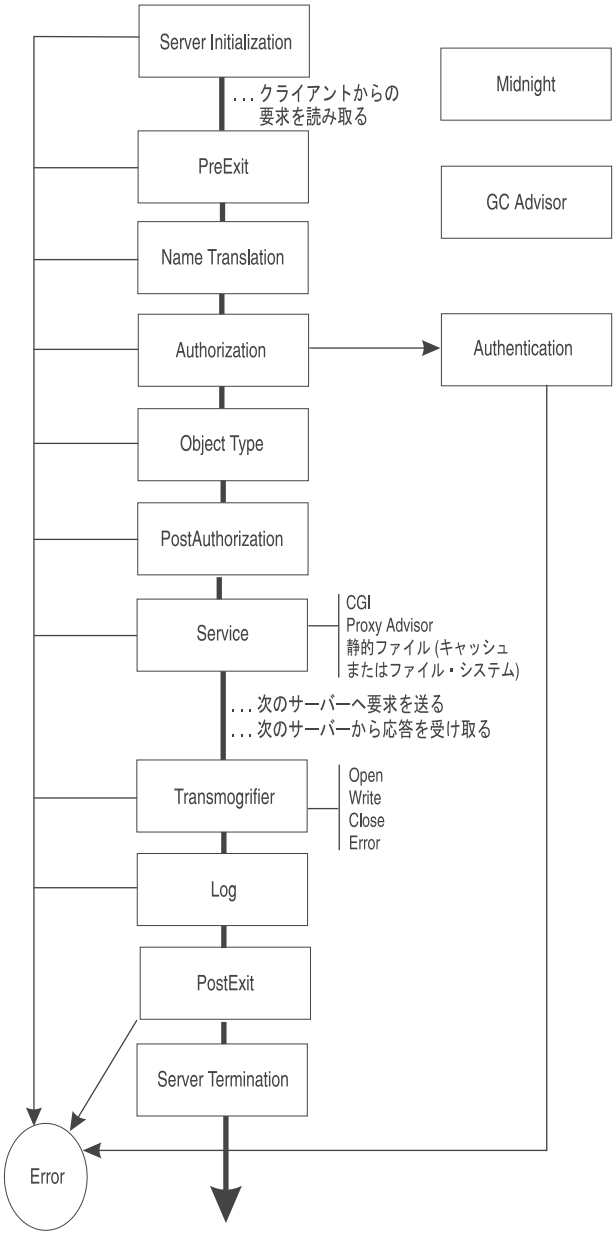

図 1. プロキシー・サーバー・プロセスの各ステップのフローチャート

この図の中の 4 つのステップは、クライアント要求の処理とは無関係に実行されます。これらのステップは、プロキシー・サーバーの実行および保守に関連します。これらのステップには以下が含まれます。

- Server Initialization
- Midnight
- GC Advisor
- Server Termination

以下のリストは、図 1 に示した各ステップの目的を示しています。すべてのステップが特定の要求に対して呼び出されるわけではないことに注意してください。

Server Initialization

プロキシー・サーバーが始動されたときに、クライアント要求が受け入れられる前に、初期化を実行します。

Midnight

夜間にプラグインを実行します。要求コンテキストはありません。このステップは要求プロセスの一部ではないため、図では別個に表示されています。つまり、このステップの実行はどの要求からも独立しています。

GC Advisor

キャッシュ内のファイルに関するガーベッジ・コレクションの決定に影響を与えます。このステップは要求プロセスの一部ではないため、図では別個に表示されています。つまり、このステップの実行はどの要求からも独立しています。ガーベッジ・コレクションは、キャッシュ・サイズが最大値に達したときに行われます。 (キャッシュ・ガーベッジ・コレクションの構成については、「*WebSphere Application Server Caching Proxy 管理ガイド*」に記載されています。)

PreExit

要求が読み込まれた後、まだ何も実行されないうちに処理を実行します。

このステップで要求が処理されたことを示す標識 (HTTP_OK) が戻された場合には、サーバーは要求プロセスの他のステップをバイパスし、Transmogrifier、Log、および PostExit ステップだけを実行します。

Name Translation

仮想パスを (URL から) 物理パスに変換します。

許可

保管されたセキュリティー・トークンを使用して保護、ACL、およびその他のアクセス制御用の物理パスを検査し、基本認証に必要な WWW 認証ヘッダーを生成します。独自のプラグイン関数を作成してこのステップを置き換える場合は、これらのヘッダーを自分で生成しなければなりません。

詳細については、 38 ページの『認証および許可』を参照してください。

Authentication

セキュリティー・トークンのデコード、検査、および保管を行います。

詳細については、 38 ページの『認証および許可』を参照してください。

Object Type

パスが示すファイル・システム・オブジェクトを探します。

Post Authorization

許可およびオブジェクト検索後に (ただし要求が満たされる前に) 処理を実行します。

このステップで要求が処理されたことを示す標識 (HTTP_OK) が戻された場合には、サーバーは要求プロセスの他のステップをバイパスし、Transmogrifier、Log、および PostExit ステップだけを実行します。

Service

ファイルの送信、CGI の実行などによって要求を満たします。

Proxy Advisor
プロキシーとキャッシングの決定に影響を与えます。

Transmogrifier
クライアントに送信される応答のデータ部分に書き込みアクセス権限を与えます。

Log カスタマイズされたトランザクション・ロギングを使用可能にします。

Error エラー条件に対して、カスタマイズされた応答を使用可能にします。

PostExit
要求処理に割り振られたリソースをクリーンアップします。

Server Termination
正常シャットダウンが行われたときにクリーンアップ処理を実行します。

ガイドライン

- サーバーのプラグイン関数用に用意された構文とガイドラインに従って、プログラムを作成してください。プラグイン関数ごとに固有の関数名を指定し、必要に応じてサーバーの事前定義関数を呼び出します。

  AIX® システムでは、プラグイン関数をリストするエクスポート・ファイル (例えば libmyapp.exp) が必要であり、 Caching Proxy API インポート・ファイル (libhttpdapi.exp) とリンクする必要があります。

  Linux、HP-UX、および Solaris システムでは、libhttpdapi および libc ライブラリーとリンクする必要があります。

  Windows® システムでは、プラグイン関数をリストするモジュール定義ファイル (.def) が必要であり、 HTTPDAPI.LIB とリンクする必要があります。

  関数定義に HTAPI.h を組み込み、HTTPD_LINKAGE マクロを使用します。このマクロを使用すると、すべての関数で必ず同じ呼び出し規則を使用できます。
- サーバーはマルチスレッド環境で稼働します。したがって、プラグインはスレッド・セーフでなければなりません。アプリケーションが再入可能であれば、パフォーマンスは低下しません。
- プラグインでは、スレッド・スコープのアクションに限定します。終了、ユーザー ID の変更、シグナル・ハンドラーの登録などの、プロセス・スコープのアクションは行わないでください。
- グローバル変数を使用しないでください。使用する必要がある場合は、相互排他セマフォーでグローバル変数を保護します。
- HTTPD_write() 関数を用いてデータをクライアントに送り戻す場合は、Content-Type ヘッダーを忘れずに設定してください。
- 必ず戻りコードを検査して、必要に応じて条件付き処理を行ってください。
- 使用するコンパイラーの資料を参照して、オペレーティング・システムの必要に応じてプログラムをコンパイルおよびリンクし、共用オブジェクト (例えば、DLL または .so ファイル) を構築してください。

  以下のコンパイル・コマンドおよびリンク・コマンドを、ガイドラインとして使用します。

– **AIX**、 IBM CSet++ を使用
  - **コンパイル :**

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

  - **リンク :**

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

  (このコマンドは、読みやすくするために 2 行にわたって示されています。)
– **HP-UX**、HP C/ANSI C Developer's Bundle および HP aC++ Compiler を使用。
  - **コンパイル :**

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

  - **リンク :**

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

– **Linux**、Gnu Compiler C (GCC) バージョン 3.2.X を使用
  - **コンパイル :**

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

  - **リンク :**

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

– **Solaris**、Sun Workshop を使用
  - **コンパイル :**

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

  - **リンク :**

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

– **Windows**、Microsoft® Visual C++ を使用
  - **コンパイル :**

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

  - **リンク :**

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

エクスポートを指定するには、以下のいずれかの方法を使用します。
– ソースに _declspec(dllexport) 定義を追加する。
– LIB コマンド行に /EXPORT:entryname を指定する。
– EXPORTS ステートメントを含むモジュール定義ファイルを作成する。

- Caching Proxy API ディレクティブを構成ファイルに追加して、プログラムのプラグイン関数を適切なステップに関連付けます。サーバー要求プロセスではステップごとに個別のディレクティブがあります。新規ディレクティブを有効にするには、サーバーを停止し再始動します。

  注: Caching Proxy は、再始動時でも共用オブジェクト (DLL または .so ファイル) をアンロードしません。共用オブジェクトを解放するには、サーバーを停止および始動する必要があります。
- プログラムは、実稼働環境で使用する前に厳密にテストしてください。Caching Proxy はスレッド化サーバーであるため、テストは fork 処理サーバーに必要なテ

ストより厳しくする必要があります。プロキシー・サーバーがプログラムを直接呼び出した後、サーバーとプログラムはともに同じプロセス・スペースで稼働するため、プログラム内で発生したエラーによりプロキシー・サーバーに障害が起きる場合があります。

プラグイン関数

定義済み要求処理ステップ用の独自のプログラム関数を作成する場合は、『プラグイン関数プロトタイプ』に示した構文に従ってください。

各ユーザー関数では、実行されたアクションを示す値が戻りコード・パラメーターに使用される必要があります。

- HTTP_NOACTION (値 0) というコードは、関係のあるアクションが行われなかったことを意味します。このコードが戻された場合、プロキシー・サーバーはこのステップのデフォルト・アクションを行います。
- 有効な HTTP 戻りコードのいずれかが、プラグイン関数がステップを処理したことを示します。 (有効な戻りコードのリストについては、 17 ページの『HTTP 戻りコードおよび値』を参照してください。) 有効な HTTP 戻りコードが指定された場合、他のプラグイン関数が呼び出されてこの要求の該当のステップが処理されるということはなくなります。

プラグイン関数プロトタイプ

Caching Proxy ステップごとの関数プロトタイプは、使用する形式を示し、それらが実行できる処理のタイプを示します。関数名は事前定義されないので、注意してください。関数には固有の名前を指定する必要がありますが、命名規則は自由に決めることができます。わかりやすくするため、本書では、サーバーの処理ステップに関連する名前を使用しています。

各プラグイン関数で、特定の事前定義 API 関数が有効です。すべてのステップで無効な事前定義関数もあります。以下の事前定義 API 関数は、これらのどのプラグイン関数からでも呼び出すことができます。

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- HTTPD_log* 関数

その他の有効または無効な API 関数については、関数のプロトタイプの説明に示されています。

関数に送られる *handle* パラメーターの値は、最初の引き数として事前定義関数に渡すことができます。事前定義 API 関数については、 18 ページの『事前定義関数およびマクロ』で説明しています。

Server Initialization

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

このステップに定義された関数は、サーバー初期化の間のモジュールのロード時に一度呼び出されます。初期化を行うことができるのはいずれかの要求が受け入れられる前です。

すべてのサーバー初期化関数が呼び出されますが、このステップの関数からエラー戻りコードが戻されると、サーバーはエラー・コードを戻した関数と同じモジュールで構成された他のすべての関数を無視します。 (つまり、エラーを戻した関数と同じ共用オブジェクトに入っている関数はどれも呼び出されません。)

バージョン・パラメーターにはプロキシー・サーバーのバージョン番号が含まれます。バージョン・パラメーターは Caching Proxy によって指定されます。

PreExit

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

このステップに定義された関数は、要求が読み取られた後、処理が行われる前にそれぞれの要求ごとに呼び出されます。このステップのプラグインを使用することにより、クライアントの要求が Caching Proxy によって処理される前に、この要求にアクセスできます。

preExit 関数の有効な戻りコードは次のとおりです。

- 0 (HTTP_NOACTION)
- 200 (HTTP_OK)
- 4*xx* または 5*xx* シリーズでの HTTP エラー (例えば 404 では HTTP_NOT_FOUND)

その他の戻りコードは使用しないでください。

この関数が HTTP_OK を戻した場合、プロキシー・サーバーは、要求が処理されたものと想定します。それ以降のすべての要求処理ステップはバイパスされ、応答ステップ (Transmogrifier、 Log、および PostExit) のみが実行されます。

このステップ中は、事前定義 API 関数はすべて有効です。

Midnight

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

このステップで定義された関数は毎日、真夜中に実行されます。要求コンテキストは含まれていません。例えば、この関数はログを分析する子プロセスを起動するために使用できます。 (このステップで処理が長引くと、ロギングの妨げになる可能性があることに注意してください。)

Authentication

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

このステップで定義された関数は、要求の認証方式に基づいて要求ごとに呼び出されます。この関数は、要求とともに送信されるセキュリティー・トークンの検査をカスタマイズするために使用できます。

Name Translation

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

このステップで定義された関数は、要求ごとに呼び出されます。テンプレートと一致する要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。 Name Translation ステップは要求が処理される前に実行され、URL をファイル名などのオブジェクトにマッピングするためのメカニズムを提供します。

許可

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

このステップで定義された関数は、要求ごとに呼び出されます。テンプレートと一致する要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。 Authorization ステップは要求が処理される前に実行され、識別されたオブジェクトをクライアントに戻すことができるかどうかを検査するために使用できます。基本認証を行う場合は、必要な WWW 認証ヘッダーを生成しなければなりません。

Object Type

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

このステップで定義された関数は、要求ごとに呼び出されます。テンプレートと一致する要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。 Object Type ステップは要求が処理される前に実行され、オブジェクトが存在するかどうかを調べてオブジェクトの型定義を行うために使用できます。

PostAuthorization

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

このステップで定義された関数は、要求を許可してから処理が起こるまえに呼び出されます。この関数が HTTP_OK を戻した場合、プロキシー・サーバーは、要求が処理されたものと想定します。それ以降のすべての要求ステップはバイパスされ、応答ステップ (Transmogrifier、 Log、および PostExit) のみが実行されます。

このステップ中は、サーバー事前定義関数はすべて有効です。

Service

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

このステップで定義された関数は、要求ごとに呼び出されます。テンプレートと一致する要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。 Service ステップは、要求が PreExit または PostAuthorization ステップで満たされなかった場合に、この要求を満たします。

このステップ中は、サーバー事前定義関数はすべて有効です。

URL ではなく HTTP メソッドに基づいて実行する Service 関数の構成については、*WebSphere Application Server Caching Proxy 管理ガイド* の Enable ディレクティブの項を参照してください。

Transmogrifier

このプロセス・ステップで呼び出される関数は、応答データをストリームとしてフィルターに掛けるために使用できます。このステップの 4 つのプラグイン関数は順番に呼び出され、それぞれがデータが流れるパイプのセグメントとして動作します。つまり、指定した *open*、*write*、*close*、および *error* 関数が応答ごとにこの順番で呼び出されます。それぞれの関数は、順番に同じデータ・ストリームを処理します。

このステップには、以下の 4 つの関数をインプリメントしなければなりません。 (使用する関数名は、以下で使用されているものと同じでなくてもかまいません。)

- **Open**

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

  open 関数は、このストリームのデータの処理に必要な初期化 (バッファー割り振りなど) を実行します。 HTTP_OK 以外の戻りコードが戻されると、このフィルターは打ち切りになります (write 関数も close 関数も呼び出されません)。関数は void ポインターを戻すことができるので、構造にスペースを割り振り、後続の関数の *correlator* パラメーターで戻されるポインターをもつことができます。

- **Write**

  ```
  void  HTTPD_LINKAGE  writeFunction (
           unsigned char *handle,
           unsigned char *data,    /* response data sent by the
  ```

```
                                origin server */
        unsigned long *length,  /* length of response data */
        void *correlator,       /* pointer returned by the
                                   'open' function */
        long *return_code
        )
```

write 関数はデータを処理し、新規または変更データを指定してサーバーの事前定義 HTTPD_write() 関数を呼び出すことができます。プラグイン側では、渡されたバッファーを解放しないでください。また、サーバーが受け取ったバッファーを解放することもありません。

write 関数の有効範囲でデータを変更しないようにする場合は、open、write、または close 関数の有効範囲で HTTPD_write() 関数を呼び出して、クライアントへの応答に対してデータを渡す必要があります。*correlator* 引き数はデータ・バッファーを指すポインターで、*open* ルーチンで戻されたものです。

- **Close**

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

close 関数は、このストリームのデータの処理を完了するために必要なクリーンアップ・アクション (correlator バッファーのフラッシュや解放など) を行います。 *correlator* 引き数はデータ・バッファーを指すポインターで、*open* ルーチンで戻されたものです。

- **Error**

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

error 関数により、エラー・ページが送信される前に、バッファー内にあるデータのフラッシュまたは解放 (あるいはその両方) などのクリーンアップ・アクションを行うことができるようになります。この時点で、エラー・ページを処理するために open、write、および close 関数が呼び出されます。*correlator* 引き数はデータ・バッファーを指すポインターで、*open* ルーチンで戻されたものです。

注:

- Transmogrifier ステップのプラグインを作成するときは、open、write、および close 関数の有効範囲内のいずれかの時点で HTTPD_open()、HTTPD_write()、および HTTPD_close() を呼び出す必要があります。HTTPD_write() は、HTTPD_open() 関数が呼び出された後にのみ呼び出すことができます。これらの事前定義関数の目的は、次の関数を呼び出せるように制御をサーバーに渡すことです。

- Transmogrifier API ステップとサーバーを正しく動作させるには、HTTPD_* 関数を呼び出さなければなりません。例えば、HTTPD_open() と HTTPD_close() を呼び出さなかった場合、クライアントにヘッダーが戻されません。
- データ・フィルター・アプリケーションがデータ・ストリームのフィルター操作で正しい選択を実行できないと、望ましくない結果になる場合があるので注意してください。CGI は正しくフィルター操作されないと機能しない場合があり、GIF ファイルは表示されず、他のバイナリー・ストリームは正常に機能しません。
- プラグインでは、コンテンツ本体をバッファーに入れる必要はありません。Caching Proxy はコンテンツの長さを自動的に判別します。
- HTTPD_open() は、ヘッダーの制御をサーバーに渡す準備が整ったときに呼び出すのが好ましい方法です。ただし、後で API プログラムでヘッダーを設定する必要がある場合は、 write または close 関数が HTTPD_open() 関数を呼び出すまで待つことができます。

  注: HTTPD_open() 関数を呼び出す前に、HTTPD_set() または httpd_setvar() を使用してヘッダーを設定しなければなりません。
- データ・ストリームにはヘッダーは組み込まれません。プラグインでは、set 関数と extract 関数を使用してヘッダーを操作する必要があります。プラグインの *open* 関数は、すべてのヘッダーが読み取られるまで呼び出されません。
- 複数の Transmogrifier プラグインを使用することができます。これらのプラグインは、構成ファイル内に示されている順序で呼び出されます。
- SSL トンネリングは、Transmogrifier プラグインを介しては渡されません。

GC Advisor

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

このステップで定義された関数は、ガーベッジ・コレクションの際、キャッシュ内のファイルごとに呼び出されます。この関数によって、保持するファイルと廃棄するファイルの決定に影響を与えることができます。詳細については、GC_* 変数を参照してください。

Proxy Advisor

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

このステップで定義された関数は、各プロキシー要求のサービス時に呼び出されます。例えば、USE_PROXY 変数を設定するために使用することができます。

Log

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

このステップで定義された関数は、要求が処理され、クライアントへの通信がクローズされた後で、それぞれの要求ごとに呼び出されます。テンプレートと一致する要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。この関数は、要求処理の成否に関係なく呼び出されます。ログ・プラグインにデフォルトのログ・メカニズムを変更させたくない場合は、戻りコードを HTTP_OK ではなく HTTP_NOACTION に設定します。

Error

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

このステップで定義された関数は、失敗した要求ごとに呼び出されます。テンプレートと一致する、失敗した要求のみについてプラグイン関数が呼び出されるようにしたい場合は、 URL テンプレートを構成ファイル・ディレクティブに指定します。 Error ステップを使用して、エラー応答をカスタマイズできます。

PostExit

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

このステップで定義された関数は、要求の成否に関係なく、要求ごとに呼び出されます。このステップを使用して、要求を処理するプラグインによって割り振られたリソースに対してクリーンアップ・タスクを行うことができます。

Server Termination

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

このステップで定義された関数は、サーバーが正常にシャットダウンしたときに呼び出されます。これにより、サーバー初期化ステップの際に割り振られたリソースをクリーンアップすることができます。このステップでは、いずれの HTTP_* 関数も呼び出さないでください (呼び出しの結果は保証できません)。構成ファイルに Server Termination 用の複数の Caching Proxy API ディレクティブがある場合は、それらがすべて呼び出されます。

注: Solaris コードの現行の制限のために、**ibmproxy -stop** コマンドを使用して Solaris プラットフォーム上の Caching Proxy をシャットダウンすると、Server Termination プラグイン・ステップは実行されません。Caching Proxy の開始および停止については、「*WebSphere Application Server Caching Proxy 管理ガイド*」を参照してください。

HTTP 戻りコードおよび値

これらの戻りコードは、World Wide Web Consortium によって公開されている HTTP 1.1 仕様書 (www.w3.org/pub/WWW/Protocols/) の RFC 2616 に準拠しています。プラグイン関数は、以下のいずれかの値を戻す必要があります。

表 2. Caching Proxy API 関数の HTTP 戻りコード

値	戻りコード
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
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

表 2. Caching Proxy API 関数の HTTP 戻りコード (続き)

501	HTTP_NOT_IMPLEMENTED
502	HTTP_BAD_GATEWAY
503	HTTP_SERVICE_UNAVAILABLE
504	HTTP_GATEWAY_TIMEOUT
505	HTTP_BAD_VERSION

事前定義関数およびマクロ

ユーザー独自のプラグイン関数から、サーバーの事前定義関数とマクロを呼び出すことができます。その事前定義された名前を使用し、以下の形式に従う必要があります。パラメーターの説明で、 `i` は入力パラメーターを示し、`o` は出力パラメーターを示し、 `i/o` はパラメーターが入力と出力の両方であることを示しています。

これらの関数は、要求の結果に応じて、いずれかの HTTPD 戻りコードを戻します。これらのコードについては、24 ページの『事前定義関数およびマクロからの戻りコード』で説明しています。

これらの関数を呼び出すときは、プラグインへ提供されたハンドルを最初のパラメーターとして使用します。最初のパラメーターとして使用しない場合、関数は HTTPD_PARAMETER_ERROR エラー・コードを戻します。 NULL は、有効なハンドルとしては受け入れられません。

HTTPD_authenticate()

ユーザー ID またはパスワード、あるいはその両方を認証します。 PreExit、Authentication、Authorization、および PostAuthorization ステップでのみ有効です。

```
void  HTTPD_LINKAGE  HTTPD_authenticate (
         unsigned char *handle,      /* i; handle */
         long *return_code           /* o; return code */
         )
```

HTTPD_cacheable_url()

Caching Proxy の標準に従って指定の URL コンテンツをキャッシュできるかどうかを戻します。

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
        unsigned char *handle,      /* i; handle */
        unsigned char *url,         /* i; URL to check */
        unsigned char *req_method,  /* i; request method for the URL */
        long *retval                /* o; return code */
        )
```

戻り値 HTTPD_SUCCESS は、URL コンテンツがキャッシュできることを示します。HTTPD_FAILURE はコンテンツがキャッシュできないことを示します。また、HTTPD_INTERNAL_ERROR もこの関数の可能な戻りコードです。

HTTPD_close()

(Transmogrifier ステップでのみ有効です。) ストリーム・スタック内の次の *close* ルーチンに制御権を移動します。この関数は、必要な処理が済んだ後に、Transmogrifier の open、write、close のいずれかの関数から呼び出して

ください。この関数は、応答が処理されて、Transmogrifier ステップが完了したことをプロキシー・サーバーに通知します。

```
void  HTTPD_LINKAGE  HTTPD_close (
        unsigned char *handle,       /* i; handle */
        long *return_code            /* o; return code */
        )
```

HTTPD_exec()

この要求を満たすために、スクリプトを実行します。PreExit、Service、PostAuthorization、および Error ステップで有効です。

```
void  HTTPD_LINKAGE  HTTPD_exec (
        unsigned char *handle,         /* i; handle */
        unsigned char *name,           /* i; name of script to run */
        unsigned long *name_length,    /* i; length of the name */
        long *return_code              /* o; return code */
        )
```

HTTPD_extract()

この要求に関連する変数の値を抽出します。*name* パラメーターで有効な変数は、CGI で使用されるものと同じです。詳細については、28 ページの『変数』を参照してください。この関数はすべてのステップで有効です。ただし、すべての変数がすべてのステップで有効であるとは限りません。

```
void  HTTPD_LINKAGE  HTTPD_extract (
      unsigned char *handle,       /* i; handle */
      unsigned char *name,         /* i; name of variable to extract */
      unsigned long *name_length,  /* i; length of the name */
      unsigned char *value,        /* o; buffer in which to put
                                       the value */
      unsigned long *value_length, /* i/o; buffer size */
      long *return_code            /* o; return code */
      )
```

この関数が HTTPD_BUFFER_TOO_SMALL コードを戻した場合、要求したバッファー・サイズは、抽出された値を入れるのに十分な大きさではありませんでした。この場合、この関数はバッファーを使用せず、この値を正常に抽出するために必要なバッファー・サイズで value_length パラメーターを更新します。少なくとも、戻された value_length と同じ大きさのバッファーを使用して抽出を再試行します。

注: 抽出される変数が HTTP ヘッダー用である場合、要求に同じ名前の複数のヘッダーが含まれていても、HTTPD_extract() 関数は最初に一致するオカレンスのみを抽出します。 httpd_getvar() 関数を HTTPD_extract() の代わりに使用することができます。 httpd_getvar() 関数を使用した場合、その他の利点もあります。詳細については、20 ページの httpd_getvar() 関数を参照してください。

HTTPD_file()

この要求を満たすために、ファイルを送信します。PreExit、Service、Error、PostAuthorization、および Transmogrifier ステップでのみ有効です。

```
void  HTTPD_LINKAGE  HTTPD_file (
        unsigned char *handle,           /* i; handle */
        unsigned char *name,             /* i; name of file to send */
        unsigned long *name_length,      /* i; length of the name */
        long *return_code                /* o; return code */
        )
```

httpd_getvar()

HTTPD_extract() と同じですが、引き数の長さを指定する必要がないので、簡単に使用できます。

```
const  unsigned char *          /* o; value of variable */
HTTPD_LINKAGE
httpd_getvar(
   unsigned char *handle,       /* i; handle */
   unsigned char *name,         /* i; variable name */
   unsigned long *n             /* i; index number for the array
                                      containing the header */
   )
```

ヘッダーが入っている配列の指標は 0 から始まります。配列で最初の項目を得るには、*n* に値 0 を使用します。5 番目の項目を得るには、*n* には値 4 を使用します。

注: 戻り値の内容を廃棄したり変更したりしないでください。戻されたストリングは、ヌル文字で終了しています。

HTTPD_log_access()

サーバーのアクセス・ログにストリングを書き込みます。

```
void  HTTPD_LINKAGE  HTTPD_log_access (
         unsigned char *handle,           /* i; handle */
         unsigned char *value,            /* i; data to write */
         unsigned long *value_length,     /* i; length of the data */
         long *return_code                /* o; return code */
         )
```

サーバー・アクセス・ログにパーセント記号 (%) を書き込むときに、エスケープ記号は必要がないことに注意してください。

HTTPD_log_error()

サーバーのエラー・ログにストリングを書き込みます。

```
void  HTTPD_LINKAGE  HTTPD_log_error (
         unsigned char *handle,          /* i; handle */
         unsigned char *value,           /* i; data to write */
         unsigned long *value_length,    /* i; length of the data */
         long *return_code               /* o; return code */
         )
```

サーバー・エラー・ログにパーセント記号 (%) を書き込むときに、エスケープ記号は必要がないことに注意してください。

HTTPD_log_event()

サーバーのイベント・ログにストリングを書き込みます。

```
void  HTTPD_LINKAGE  HTTPD_log_event (
         unsigned char *handle,          /* i; handle */
         unsigned char *value,           /* i; data to write */
         unsigned long *value_length,    /* i; length of the data */
         long *return_code               /* o; return code */
         )
```

サーバー・イベント・ログにパーセント記号 (%) を書き込むときに、エスケープ記号は必要がないことに注意してください。

HTTPD_log_trace()

サーバーのトレース・ログにストリングを書き込みます。

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
        unsigned char *handle,          /* i; handle */
        unsigned char *value,           /* i; data to write */
        unsigned long *value_length,    /* i; length of the data */
        long *return_code               /* o; return code */
        )
```

サーバー・トレース・ログにパーセント記号 (%) を書き込むときに、エスケープ記号は必要がないことに注意してください。

HTTPD_open()

(Transmogrifier ステップでのみ有効です。) ストリーム・スタック内の次のルーチンに制御権を移動します。必要なヘッダーを設定して、write ルーチンを開始する準備ができたら、 Transmogrifier の open、write、close のいずれかの関数からこれを呼び出してください。

```
void  HTTPD_LINKAGE  HTTPD_open (
        unsigned char *handle,      /* i; handle */
        long *return_code           /* o; return code */
        )
```

HTTPD_proxy()

プロキシー要求を行います。PreExit、Service、および PostAuthorization ステップでのみ有効です。

注: これは完了関数です。要求は、この関数の後に完了します。

```
void  HTTPD_LINKAGE  HTTPD_proxy (
        unsigned char *handle,        /* i; handle */
        unsigned char *url_name,      /* i; URL for the
                                             proxy request */
        unsigned long *name_length,   /* i; length of URL */
        void *request_body,           /* i; body of request */
        unsigned long *body_length,   /* i; length of body */
        long *return_code             /* o; return code */
        )
```

HTTPD_read()

クライアントの要求の本体を読み取ります。ヘッダーの読み取りには HTTPD_extract() を使用してください。PreExit、Authorization、PostAuthorization、および Service ステップでのみ有効であり、PUT または POST 要求が実行された場合にだけ使用できます。この関数は、HTTPD_EOF が戻されるまでループの中で呼び出してください。この要求に本体がない場合、この関数は失敗します。

```
void  HTTPD_LINKAGE  HTTPD_read (
        unsigned char *handle,          /* i; handle */
        unsigned char *value,           /* i; buffer for data */
        unsigned long *value_length,    /* i/o; buffer size
                                               (data length) */
        long *return_code               /* o; return code */
        )
```

HTTPD_restart()

すべてのアクティブ状態の要求を処理した後で、サーバーを再始動します。Server Initialization、 Server Termination、および Transmogrifier を除くすべてのステップで有効です。

```
void  HTTPD_LINKAGE  HTTPD_restart (
        long *return_code    /* o; return code */
        )
```

HTTPD_set()

この要求に関連する変数の値を設定します。*name* パラメーターで有効な変数は、CGI で使用されるものと同じです。詳細については、 28 ページの『変数』を参照してください。

この関数で変数を作成することもできますので注意してください。作成する変数は、 28 ページの『変数』で説明されている、接頭部 HTTP_ および PROXY_ の規則に準拠します。 HTTP_ で始まる変数を作成した場合、この変数は、接頭部 HTTP_ なしで、クライアントへの応答でヘッダーとして送信されます。例えば、Location ヘッダーを設定するには、変数名 HTTP_LOCATION で HTTPD_set() を使用します。接頭部 PROXY_ を付けて作成した変数は、コンテンツ・サーバーへの要求でヘッダーとして送信されます。接頭部 CGI_ を付けて作成した変数は CGI プログラムに渡されます。

この関数はすべてのステップで有効です。ただし、すべての変数がすべてのステップで有効であるとは限りません。

```
void  HTTPD_LINKAGE  HTTPD_set (
         unsigned char *handle,        /* i; handle */
         unsigned char *name,          /* i; name of value to set */
         unsigned long *name_length,   /* i; length of the name */
         unsigned char *value,         /* i; buffer with value */
         unsigned long *value_length,  /* i; length of value */
         long *return_code             /* o; return code */
         )
```

注: httpd_setvar() 関数を使用して、バッファーおよび長さを指定せずに変数値を設定することができます。詳細については、22 ページの httpd_setvar() 関数を参照してください。

httpd_setvar()

HTTPD_set() と同じですが、引き数の長さを指定する必要がないので、簡単に使用できます。

```
long             /* o; return code */
HTTPD_LINKAGE   httpd_setvar (
       unsigned char *handle,       /* i; handle */
       unsigned char *name,         /* i; variable name */
       unsigned char *value,        /* i; new value */
       unsigned long *addHdr        /* i; add header or replace it */
       )
```

addHdr パラメーターには、以下の 4 つの値を使用できます。

- HTTPD_SETVAR_REPLACE — ヘッダー変数のすべてのオカレンスを新しい値と置き換えます。
- HTTPD_SETVAR_REPLACE_ADD — ヘッダー変数が存在する場合は最初のオカレンスを新しい値に置き換え、変数が存在しない場合は、新しい値をヘッダーに追加します。
- HTTPD_SETVAR_ADD — この値をヘッダーに追加します。
- HTTPD_SETVAR_REMOVE_ALL — このヘッダー変数のすべてのオカレンスを削除します。

これらの値は HTAPI.h で定義されます。

httpd_variant_insert()

バリアントをキャッシュに挿入します。

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,    /* i; handle */
         unsigned char *URI,       /* i; URI of this object */
         unsigned char *dimension, /* i; dimension of variation */
         unsigned char *variant,   /* i; value of the variant */
         unsigned char *filename,  /* i; file containing the object */
         long *return_code         /* o; return code */
         )
```

注 :

1. dimension 引き数は、このオブジェクトが URI と異なる点となっているヘッダーを参照します。例えば、上記の例で、dimension 値を User-Agent にすることができます。
2. variant 引き数は、dimension 引き数で指定されたヘッダーの値を参照します。これは URI とは異なります。例えば、上記の例で、variant 引き数を次の値にすることができます。

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

3. filename 引き数は、ユーザーが変更済みコンテンツを保管したファイル名のヌル終了コピーを指す必要があります。ユーザーには、ファイルを除去する責任があります。このアクションは、この関数から戻った後で行うのが安全です。このファイルには、ヘッダーのない本体だけが入っています。
4. バリアントをキャッシュする場合、サーバーはコンテンツ長ヘッダーを更新して、Warning: 214 (警告 : 214) ヘッダーを追加します。ストロング・エンティティー・タグは除去されます。

httpd_variant_lookup()

キャッシュ内の指定のバリアントの有無を判別します。

```
void  HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,            /* i; handle */
    unsigned char *URI,               /* URI of this object */
    unsigned char *dimension,         /* i; dimension of variation */
    unsigned char *variant,           /* i; value of the variant */
             long *return_code);      /* o; return code */
```

HTTPD_write()

応答の本文を書き込みます。この関数は、PreExit、Service、Error、および Transmogrifier ステップで有効です。

初めてこの関数を呼び出す前にコンテンツ・タイプを設定しなかった場合、サーバーは、CGI データ・ストリームが送信されるものと想定します。

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,            /* i; handle */
    unsigned char *value,             /* i; data to send */
    unsigned char *value_length,      /* i; length of the data */
             long *return_code);      /* o; return code */
```

注: 応答ヘッダーを設定するには、22 ページの HTTPD_set() 関数を参照してください。

注: HTTPD_* 関数が戻った後で、それと一緒に渡したメモリーを解放しておくと安全です。

事前定義関数およびマクロからの戻りコード

サーバーは、要求の結果に応じて、以下のいずれかの値に戻りコード・パラメーターを設定します。

表 3. 戻りコード

値	状況コード	説明
-1	HTTPD_UNSUPPORTED	この関数はサポートされていません。
0	HTTPD_SUCCESS	この関数は正常に実行され、出力フィールドが有効です。
1	HTTPD_FAILURE	関数は失敗しました。
2	HTTPD_INTERNAL_ERROR	内部エラーを検出し、この要求の処理を続行できません。
3	HTTPD_PARAMETER_ERROR	1 つまたは複数の無効なパラメーターが渡されました。
4	HTTPD_STATE_CHECK	この関数はこのプロセス・ステップでは無効です。
5	HTTPD_READ_ONLY	(これを戻すのは HTTPD_set と httpd_setvar だけです。) 変数は読み取り専用であり、プラグインによって設定できません。
6	HTTPD_BUFFER_TOO_SMALL	(これを戻すのは HTTPD_set、httpd_setvar、および HTTPD_read だけです。)指定のバッファーが小さすぎます。
7	HTTPD_AUTHENTICATE_FAILED	(これを戻すのは HTTPD_authenticate だけです。) 認証は失敗しました。詳細については、HTTP_RESPONSE および HTTP_REASON 変数を調べてください。
8	HTTPD_EOF	(これを戻すのは HTTPD_read だけです。) 要求本文の終わりを示します。
9	HTTPD_ABORT_REQUEST	クライアントが要求が指定する条件と一致しないエンティティー・タグを指定したために、要求は打ち切られました。
10	HTTPD_REQUEST_SERVICED	(これを戻すのは HTTPD_proxy だけです。) 呼び出した関数はこの要求の応答を完了しました。
11	HTTPD_RESPONSE_ALREADY_COMPLETED	その要求の応答はすでに完了しているので、この関数は失敗しました。
12	HTTPD_WRITE_ONLY	変数は書き込み専用であり、プラグインによって読み取ることはできません。

API ステップの Caching Proxy 構成ディレクティブ

要求プロセスの各ステップには構成ディレクティブがあり、これを使用して、そのステップの間に呼び出して実行したいプラグイン関数を示すことができます。サーバーの構成ファイル (ibmproxy.conf) にそれらのディレクティブを追加するには、サーバーの構成ファイルを手操作で編集して更新するか、または Caching Proxy の「構成および管理」フォームの中の 「API 要求処理」フォームを使用します。

API 使用上の注意

- Service ディレクティブおよび NameTrans ディレクティブを除き、各ステップの API ディレクティブは構成ファイル内で特定の順序で表示する必要はありません。1 つの API ディレクティブにおける複数項目の順序が重要であることに注意してください。これについてはこのリストの後で説明します。
- すべての API ステップに項目を組み込む必要はありません。特定のステップにプラグインがない場合は、対応するディレクティブを省略するだけで、そのステップの標準処理が使用されます。
- Service および NameTrans ディレクティブは、他のマッピング・ディレクティブ (例えば、Pass ディレクティブ) と同じように機能し、構成ファイル内の他のマッピング・ディレクティブから見た相対的な出現位置と配置によって機能が異なります。例えば、/cgi-bin/foo.so についてのルールは、/cgi-bin/* のルールの前に記述しなければなりません。

  つまり、サーバーは、Service、NameTrans、Exec、Fail、Map、Pass、Proxy、ProxyWAS、および Redirect の各ディレクティブを構成ファイル内に並んでいる順序で処理します。サーバーが URL をファイルに正常にマップすると、サーバーはこれらのディレクティブ以外のディレクティブを読み取ったり、処理したりしません。 (Map ディレクティブは例外です。プロキシー・サーバーのマッピング・ルールについて詳しくは、「*WebSphere Application Server Caching Proxy 管理ガイド*」を参照してください。)
- 1 つのステップに対して、複数の構成ディレクティブを記述できます。例えば、NameTrans ディレクティブを 2 つ記述し、それぞれが別のプラグイン関数を指すようにすることができます。サーバーが Name Translation ステップを行うときには、構成ファイルに記述された順序で Name Translation (名前変換) 関数を処理します。

  注: Caching Proxy によって提供されるプラグイン関数が作成したプラグインと同じ API ディレクティブを使用する場合は、システム・プラグイン・ディレクティブの後にそのプラグインのディレクティブを入れてください。
- プラグイン関数の中には、すべての要求について実行する必要のないものがあります。
  - いくつかのディレクティブには URL マスクが含まれます。これらのディレクティブで URL マスクを指定すると、パターンに一致する URL を持つ要求についてのみ、プラグイン・アプリケーションが呼び出されます。 URL マスクを使用できるステップの詳細については 26 ページの『API ディレクティブおよび構文』を、この機能の使用方法については 26 ページの『API ディレクティブ変数』を参照してください。
  - あるタイプの認証についてのみプラグイン関数を呼び出したい場合には、Authentication ディレクティブを使用して認証方式を指定します。現時点では、基本認証だけが HTTP プロトコルによってサポートされています。追加情報については、 26 ページの『API ディレクティブ変数』を参照してください。
- サーバーが特定のプラグイン関数をロードできなかった場合、または OK 戻りコードを戻さない ServerInit ディレクティブが存在する場合は、そのコンパイル済み Caching Proxy プラグイン用の別のプラグインは呼び出されません。この時点

まで行われた、そのプラグインに固有の処理はすべて無視されます。このディレクティブに含まれるその他の Caching Proxy プラグイン、およびその関数には影響がありません。

API ディレクティブおよび構文

これらの構成ファイル・ディレクティブは、ここに明確に指定されたものを除いてスペースを入れずに、ibmproxy.conf ファイル中に 1 行で記述しなければなりません。構文例の一部では読みやすさのために改行していますが、実際のディレクティブではそこにスペースを入れないでください。

表 4. Caching Proxy プラグイン API ディレクティブ

ServerInit		*/path/file:function_name init_string*
PreExit		*/path/file:function_name*
Authentication	*type*	*/path/file:function_name*
NameTrans	*/URL*	*/path/file:function_name*
Authorization	*/URL*	*/path/file:function_name*
ObjectType	*/URL*	*/path/file:function_name*
PostAuth		*/path/file:function_name*
Service	*/URL*	*/path/file:function_name*
Midnight		*/path/file:function_name*
Transmogrifier		*/path/file:open_function_name: write_function_name: close_function_name:error_function*
Log	*/URL*	*/path/file:function_name*
Error	*/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*

API ディレクティブ変数

これらのディレクティブの変数には、以下の意味があります。

type プラグイン関数が呼び出されるかどうかを指定するために Authentication ディレクティブでのみ使用されます。有効な値は次のとおりです。

- Basic — プラグイン関数は、基本認証要求についてのみ呼び出されます。
- * — プラグイン関数は、すべての要求について呼び出されます。現時点では、基本認証だけが HTTP プロトコルによってサポートされています。基本認証以外の認証要求では、そのタイプの認証がサポートされていないことを示すエラー・コードを戻すことができます。

URL プラグイン関数が呼び出される要求を指定します。このテンプレートに一致する URL を持つ要求によって、プラグイン関数が使用されます。これらのディレクティブの URL 指定は、仮想 (プロトコルを組み込んでいない) ですが、前にスラッシュ (/) が付いています。例えば、

/www.ics.raleigh.ibm.com は正しい形ですが、http://www.ics.raleigh.ibm.com は正しくありません。この値は、特定の URL またはテンプレートとして指定することができます。

- 特定の URL — プラグイン関数は、完全に一致する URL についてのみ呼び出されます。
- URL テンプレート — プラグイン関数は、テンプレートに一致するすべての URL について呼び出されます。テンプレートにはワイルドカード文字 * を組み込むことがでます。テンプレートは */URL**、*/**、または * の形式で指定できます。

  注: パス変換を行いたい場合は、必ず、Service ディレクティブとともに URL テンプレートを指定しなければなりません。

path/file
コンパイル済みプログラムの完全修飾ファイル名。

function_name
プログラム内でプラグイン関数に指定した名前。

パス情報にアクセスする場合、Service ディレクティブには、関数名の後にアスタリスク (*) が必要です。

init_string
ServerInit ディレクティブのこのオプション部分には、プラグイン関数に渡したいテキストを入れることができます。 httpd_getvar() を使用して INIT_STRING 変数からテキストを抽出します。

これらのディレクティブの構文など、詳細については、「*WebSphere Application Server Caching Proxy 管理ガイド*」を参照してください。

他の API との互換性

Caching Proxy API は、バージョン 4.6.1 まで ICAPI および GWAPI と下位互換性があります。

CGI プログラムの移植

Caching Proxy API を使用するために C で作成された CGI アプリケーションを移植する場合、以下のガイドラインを使用します。

- main() エントリー・ポイントを除去するか名前を変更して、DLL を作成できるようにします。
- グローバル変数を除去するか、相互排他セマフォーで保護します。
- プログラム内の以下の呼び出しを変更します。
  - printf() ヘッダー呼び出しを HTTPD_set() または httpd_setvar() に変更します。
  - printf() データ呼び出しを HTTPD_write() に変更します。
  - getenv() 呼び出しを HTTPD_extract() または httpd_getvar() に変更します。これにより、未割り振りのメモリーが戻され、その結果を解放しなければならないので注意してください。

- サーバーはマルチスレッド環境で稼働し、プラグイン関数はスレッド・セーフでなければならないことに注意してください。関数が再入可能であれば、パフォーマンスは低下しません。
- HTTPD_write() を用いてデータをクライアントに送り戻す場合は、Content-Type ヘッダーを忘れずに設定してください。
- メモリー・リークがないか細部まで正確にコードを調べます。
- エラー・パスについて考慮します。エラー・メッセージを自身で生成し、それを HTML として送り戻す場合は、HTTPD_OK をサービス関数 (1 つまたは複数) から戻す必要があります。

Caching Proxy API 参照情報

変数

API プログラムの作成に際しては、リモート・クライアントおよびサーバー・システムに関する情報を提供する Caching Proxy 変数を使用することができます。

注 :

- ユーザー定義変数名に接頭部 SERVER_ を付けることはできません。Caching Proxy API 関数は SERVER_ で始まる変数をサーバー用に予約しており、したがって、それらの変数は読み取り専用です。また、接頭部 HTTP_ と PROXY_ も HTTP ヘッダー用に予約されています。
- クライアントが送信するすべての要求ヘッダー (Set-Cookie など) には HTTP_ という接頭部が付いており、それらの値を抽出することができます。要求ヘッダーである変数にアクセスするには、変数名に HTTP_ という接頭部を付けてください。 httpd_setvar() 事前定義関数を使用して新規変数を作成することもできます。これらのヘッダーの詳細については、24 ページの『事前定義関数およびマクロからの戻りコード』を参照してください。
- 2 つの変数接頭部 HTTP_ と PROXY_ は、変数が要求と応答のどちらのヘッダーに適用されるかを指示するために使用されます。 HTTP_ 接頭部はクライアントと Caching Proxy の間を流れる変数を参照します。 PROXY_ 接頭部は Caching Proxy と起点サーバー (またはプロキシー・チェーン内の次のサーバー) の間を流れる変数を参照します。これらの変数は、要求処理ステップでのみ有効になります。
  - HTTP_* 変数を抽出すると、プロキシー・サーバーに対するクライアントの要求で使用されていたヘッダーの値が得られます。
  - HTTP_* 変数を設定すると、プロキシー・サーバーからクライアントに送信される応答ヘッダーが設定されます。
  - PROXY_* 変数を抽出すると、コンテンツ・サーバーからプロキシー・サーバーに戻されたヘッダーの値が得られます。
  - PROXY_* 変数を設定すると、プロキシー・サーバーからコンテンツ・サーバー (またはプロキシー・チェーン内の次のサーバー) に送信される要求ヘッダーが設定されます。

  29 ページの図 2 は、Caching Proxy がクライアント要求を扱うときのこれらの接頭部の使用法を示します。

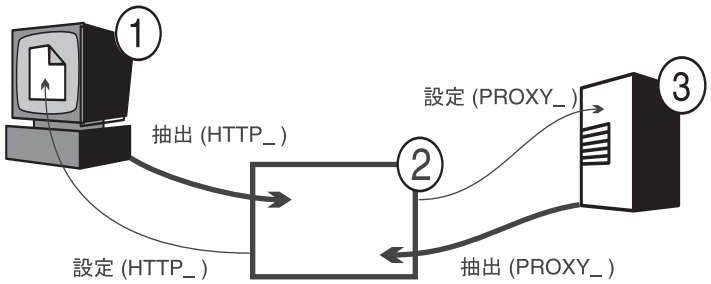

図 2. HTTP_ および PROXY_ 変数接頭部: (凡例 : 1 — クライアント・マシン 2 — Caching Proxy 3 — 起点サーバー)

- 読み取り専用の変数もあります。読み取り専用の変数は、要求または応答から抽出して事前定義関数 httpd_getvar() で使用できる値を表します。読み取り専用の変数を httpd_setvar() 関数を使用して変更しようとすると、戻りコード HTTPD_READ_ONLY が戻されます。
- 読み取り専用と識別されない変数は、事前定義関数 httpd_getvar() で読み取り、httpd_setvar() で設定することができます。これらの変数は、要求または応答から抽出できる値か、要求または応答を処理するときに設定または作成できる値を表します。

変数定義

注: HTTP_ または PROXY_ 接頭部が付いていないヘッダー変数はあいまいです。あいまいさを避けるために、ヘッダーの変数名とともに常に HTTP_ または PROXY_ 接頭部を使用してください。

ACCEPT_RANGES

Accept-Ranges 応答ヘッダーの値が入ります。これは、コンテンツ・サーバーが範囲要求に応答できるかどうかを指定します。

PROXY_ACCEPT_RANGES を使用して、コンテンツ・サーバーからプロキシーに送信されるヘッダー値を抽出してください。

HTTP_ACCEPT_RANGES を使用して、プロキシーからクライアントに送信されるヘッダー値を設定してください。

注: ACCEPT_RANGES はあいまいです。あいまいさを排除するには、代わりに HTTP_ACCEPT_RANGES および PROXY_ACCEPT_RANGES を使用してください。

ALL_VARIABLES

読み取り専用。すべての CGI 変数が入ります。以下に例を示します。

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING

読み取り専用。サーバーがクライアント認証をサポートする場合は、このストリングには、クライアントを認証するのに使用される、デコードされていないクリデンシャルが入ります。

AUTH_TYPE
読み取り専用。サーバーがクライアント認証をサポートし、スクリプトが保護されている場合は、この変数にはクライアントの認証に使用される方式が入ります。例えば `Basic` です。

CACHE_HIT
読み取り専用。キャッシュ内にプロキシー要求が見つかったかどうかを示します。以下の値が戻されます。

- 0 - 要求がキャッシュ内で見つかりませんでした。
- 1 - 要求がキャッシュ内で見つかりました。

CACHE_MISS
書き込み専用。キャッシュ・ミスを強制するかどうかを定義します。有効な値は次のとおりです。

- 0 - キャッシュ・ミスを強制しません。
- 1 - キャッシュ・ミスを強制します。

CACHE_TASK
読み取り専用。キャッシュが使用されたかどうかを識別します。以下の値が戻されます。

- 0 - 要求がキャッシュのアクセスまたは更新を行いませんでした。
- 1 - 要求がキャッシュから出されました。
- 2 - 要求されたオブジェクトがキャッシュに存在しましたが、再検証の必要がありました。
- 3 - 要求されたオブジェクトがキャッシュに存在せず、追加された可能性があります。

この変数は、PostAuthorization、PostExit、ProxyAdvisor、または Log ステップで使用できます。

CACHE_UPDATE
読み取り専用。プロキシー要求がキャッシュを更新したかどうかを示します。以下の値が戻されます。

- 0 - キャッシュは更新されませんでした。
- 1 - キャッシュは更新されました。

CLIENT_ADDR または CLIENTADDR
REMOTE_ADDR と同じ。

CLIENTMETHOD
REQUEST_METHOD と同じ。

CLIENT_NAME または CLIENTNAME
REMOTE_HOST と同じ。

CLIENT_PROTOCOL または CLIENTPROTOCOL
クライアントが要求を出すために使用するプロトコルの名前とバージョンが入ります。例えば `HTTP/1.1` です。

CLIENT_RESPONSE_HEADERS
読み取り専用。サーバーがクライアントに送信するヘッダーを含むバッファーを戻します。

CONNECTIONS
読み取り専用。使用されている接続の数またはアクティブな要求の数が入ります。例えば 15 です。

CONTENT_CHARSET
text/* 用の応答の文字セットです。例えば US ASCII です。この変数の抽出は、クライアントからのコンテンツ文字セットのヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるコンテンツ文字セットのヘッダーが影響を受けます。

CONTENT_ENCODING
文書で使用されるエンコードを指定します。例えば x-gzip です。この変数の抽出は、クライアントからのコンテンツ・エンコードのヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるコンテンツ文字セットのヘッダーが影響を受けます。

CONTENT_LENGTH
この変数の抽出は、クライアントの要求から得られるヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるヘッダーの値が影響を受けます。

注: CONTENT_LENGTH はあいまいです。あいまいさを排除する場合は、HTTP_CONTENT_LENGTH および PROXY_CONTENT_LENGTH を使用してください。

CONTENT_TYPE
この変数の抽出は、クライアントの要求から得られるヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるヘッダーの値が影響を受けます。

注: CONTENT_TYPE はあいまいです。あいまいさを排除する場合は、HTTP_CONTENT_TYPE および PROXY_CONTENT_TYPE を使用してください。

CONTENT_TYPE_PARAMETERS
その他の MIME 属性が入りますが、文字セットは入りません。この変数の抽出は、クライアント要求から得られるヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるヘッダーの値が影響を受けます。

DOCUMENT_URL
URL (Uniform Request Locator) が入ります。以下に例を示します。

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

DOCUMENT_URI
DOCUMENT_URL と同じ。

DOCUMENT_ROOT
読み取り専用。パス・ルールによる定義のとおりの文書ルート・パスが入ります。

ERRORINFO
エラー・ページを判別するエラー・コードを指定します。例えば blocked です。

EXPIRES
プロキシーのキャッシュに保管された文書の有効期限が切れる時期を定義します。この変数の抽出は、クライアント要求から得られるヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるヘッダーの値が影響を受けます。以下に例を示します。

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

GATEWAY_INTERFACE
読み取り専用。サーバーが使用している API のバージョンが入ります。例えば ICSAPI/2.0 です。

GC_BIAS
書き込み専用。この浮動小数点値は、ガーベッジ・コレクションを考慮されているファイルのガーベッジ・コレクションについての決定に影響します。入力される値には、ランキングを判別するために、そのファイル・タイプ用の Caching Proxy の品質設定値が掛けられます。品質設定値は 0.0 から 0.1 の範囲の値であり、プロキシー構成ファイル (ibmproxy.conf) 内の AddType ディレクティブによって定義されます。

GC_EVALUATION
書き込み専用。この浮動小数点値によって、ガーベッジ・コレクションを考慮されているファイルの除去 (0.0) と保持 (1.0) のどちらを行うかを判別します。 0.0 と 1.0 の間の値はランクによって順位を付けられます。つまり、GC_EVALUATION 値 0.1 を持つファイルは、GC_EVALUATION 値 0.9 を持つファイルよりも除去される可能性が高くなります。

GC_EXPIRES
読み取り専用。考慮中のファイルがキャッシュ内で有効期限切れとなるまでの残り秒数を示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

GC_FILENAME
読み取り専用。ガーベッジ・コレクション用に考慮されているファイルを示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

GC_FILESIZE
読み取り専用。ガーベッジ・コレクション用に考慮されているファイルのサイズを示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

GC_LAST_ACCESS
読み取り専用。ファイルが最後にアクセスされた時期を示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

GC_LAST_CHECKED
読み取り専用。ファイルが最後に検査された時期を示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

GC_LOAD_DELAY
読み取り専用。ファイルの検索に掛かる時間を示します。この変数を抽出できるのは、GC Advisor プラグインのみです。

HTTP_COOKIE
この変数には、読み取り時に、クライアントによって設定された Set-Cookie

ヘッダーの値が入ります。これは、応答ストリーム (プロキシーとクライアントの間) での新しい cookie の設定に使用することもできます。この変数を設定すると、重複するヘッダーの有無に関係なく、文書要求ストリームの中に新しい Set-Cookie ヘッダーが作成されます。

HTTP_HEADERS
読み取り専用。すべてのクライアント要求ヘッダーの抽出に使用されます。

HTTP_REASON
この変数を設定すると、HTTP 応答内の理由ストリングが影響を受けます。また、この設定により、クライアントに対するプロキシーの応答に含まれる理由ストリングも影響を受けます。この変数を抽出すると、コンテンツ・サーバーからプロキシーへの応答で理由ストリングが戻されます。

HTTP_RESPONSE
この変数を設定すると、HTTP 応答内の応答コードが影響を受けます。また、この設定により、クライアントに対するプロキシーの応答に含まれる状況コードも影響を受けます。この変数を抽出すると、コンテンツ・サーバーからプロキシーへの応答で状況コードが戻されます。

HTTP_STATUS
HTTP 応答コードおよび理由ストリングが入ります。例えば `200 OK` です。

HTTP_USER_AGENT
User-Agent 要求ヘッダーの値が入ります。これは、クライアント Web ブラウザーの名前 (例えば `Netscape Navigator / V2.02`) です。この変数を設定すると、クライアントに対するプロキシーの応答に含まれるヘッダーが影響を受けます。この変数の抽出は、クライアントの要求から得られるヘッダーに適用されます。

INIT_STRING
読み取り専用。このストリングは、ServerInit ディレクティブで定義されます。この変数は、Server Initialization ステップでのみ読み取ることができます。

LAST_MODIFIED
この変数の抽出は、クライアント要求から得られるヘッダーに適用されます。この変数を設定すると、コンテンツ・サーバーへの要求におけるヘッダーの値が影響を受けます。以下に例を示します。

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

LOCAL_VARIABLES
読み取り専用。全ユーザー定義変数。

MAXACTIVETHREADS
読み取り専用。アクティブ・スレッドの最大数。

NOTMODIFIED_TO_OK
クライアントへの完全応答を強制します。 PreExit および ProxyAdvisor ステップで有効です。

ORIGINAL_HOST
読み取り専用。要求のホスト名または宛先 IP アドレスを戻します。

ORIGINAL_URL
読み取り専用。クライアント要求で送信された元の URL を戻します。

OVERRIDE_HTTP_NOTRANSFORM
Cache-Control: no-transform ヘッダーがある場合にデータを変更できるようにします。この変数を設定すると、クライアントへの応答ヘッダーが影響を受けます。

OVERRIDE_PROXY_NOTRANSFORM
Cache-Control: no-transform ヘッダーがある場合にデータを変更できるようにします。この変数を設定すると、コンテンツ・サーバーへの要求が影響を受けます。

PASSWORD
基本認証の場合は、デコードされたパスワードが入ります。例えば `password` です。

PATH 完全変換パスが入ります。

PATH_INFO
Web ブラウザーが送信したものと同様の追加パス情報が入ります。例えば `/foo` です。

PATH_TRANSLATED
PATH_INFO に入っているパス情報のデコード・バージョンまたは変換バージョンが入ります。以下に例を示します。

```
d:¥wwwhome¥foo
```

```
/wwwhome/foo
```

PPATH
部分変換パスが入ります。これは、Name Translation ステップで使用してください。

PROXIED_CONTENT_LENGTH
読み取り専用。プロキシー・サーバーを介して実際に転送される応答データの長さを戻します。

PROXY_ACCESS
要求がプロキシー要求であるかどうかを定義します。例えば `NO` です。

PROXY_CONTENT_TYPE
HTTPD_proxy() を介して出されたプロキシー要求の Content-Type ヘッダーが入ります。情報が POST のメソッドで送信される場合、この変数には組み込まれたデータのタイプが入ります。プロキシー・サーバー構成ファイル内に独自のコンテンツ・タイプを作成し、このコンテンツ・タイプをビューアーにマップすることができます。この変数の抽出は、コンテンツ・サーバーの応答から得られるヘッダー値に適用されます。この変数を設定すると、コンテンツ・サーバーへの要求のヘッダーが影響を受けます。以下に例を示します。

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

PROXY_CONTENT_LENGTH
HTTPD_proxy() を介して出されたプロキシー要求の Content-Length ヘッダー。情報が POST のメソッドで送信される場合、この変数にはデータの文字数が入ります。一般に、サーバーは、標準入力を用いて情報を転送するときは、ファイルの終わりフラグを送信しません。必要な場合は、CONTENT_LENGTH 値を使用すると、入力ストリングの終わりを判別する

ことができます。この変数の抽出は、コンテンツ・サーバーの応答から得られるヘッダー値に適用されます。この変数を設定すると、コンテンツ・サーバーへの要求のヘッダーが影響を受けます。以下に例を示します。

```
7034
```

PROXY_COOKIE

この変数には、読み取り時に、起点サーバーによって設定された Set-Cookie ヘッダーの値が入ります。これは、要求ストリームでの新しい cookie の設定に使用することもできます。この変数を設定すると、重複するヘッダーの有無に関係なく、文書要求ストリームの中に新しい Set-Cookie ヘッダーが作成されます。

PROXY_HEADERS

読み取り専用。プロキシー・ヘッダーの抽出に使用されます。

PROXY_METHOD

HTTPD_proxy() を介して作成された要求のメソッドを示します。この変数の抽出は、コンテンツ・サーバーの応答から得られるヘッダー値に適用されます。この変数を設定すると、コンテンツ・サーバーへの要求のヘッダーが影響を受けます。

QUERY_STRING

情報が GET メソッドを使用して送信される場合、この変数には照会内の疑問符 (?) に続く情報が入ります。この情報は CGI プログラムによってデコードしなければなりません。以下に例を示します。

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

RCA_OWNER

読み取り専用。要求されたオブジェクトを所有していたノードを示す数値を戻します。この変数は、PostExit、ProxyAdvisor、または Log ステップで使用することが可能であり、サーバーがリモート・キャッシュ・アクセス (RCA) を使用するキャッシュ配列の一部であるときにのみ意味があります。

RCA_TIMEOUTS

読み取り専用。すべてのピアへの RCA 要求で発生したタイムアウトの合計回数を示す数値を戻します。この変数は、どのステップでも使用することができます。

REDIRECT_*

読み取り専用。変数名 (例えば REDIRECT_URL) に対応する、エラー・コードのリダイレクト・ストリングが入ります。考えられる REDIRECT_ 変数のリストは、Apache Web サーバーについてのオンライン文書に示されています (http://httpd.apache.org/docs-2.0/custom-error.html)。

REFERRER_URL

読み取り専用。ブラウザーの最後の URL ロケーションが入ります。この変数により、クライアントは、Request-URL の入手元であるリソースのアドレス (URL) を (サーバーに役立つように) 指定できます。以下に例を示します。

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

REMOTE_ADDR
Web ブラウザーの IP アドレスが入ります (使用可能な場合)。例えば 45.23.06.8 です。

REMOTE_HOST
Web ブラウザーのホスト名が入ります (使用可能な場合)。例えば www.raleigh.ibm.com です。

REMOTE_USER
サーバーがクライアント認証をサポートし、スクリプトが保護されている場合は、この変数は、認証のために渡されたユーザー名が入ります。例えば joeuser です。

REQHDR
読み取り専用。クライアントによって送信されたヘッダーのリストが入ります。

REQUEST_CONTENT_TYPE
読み取り専用。要求本文のコンテンツ・タイプを戻します。以下に例を示します。

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

REQUEST_CONTENT_LENGTH
読み取り専用。情報が POST のメソッドで送信される場合、この変数にはデータの文字数が入ります。一般に、サーバーは、標準入力を用いて情報を転送するときは、ファイル終わりフラグを送信しません。必要な場合は、CONTENT_LENGTH 値を使用すると、入力ストリングの終わりを判別することができます。例えば 7034 です。

REQUEST_METHOD
読み取り専用。要求の送信に使用されるメソッド (HTML フォームの METHOD 属性によって指定されたとおりの) が入ります。例えば GET または POST です。

REQUEST_PORT
読み取り専用。URL に指定されたポート番号またはプロトコルに基づいたデフォルト・ポートを戻します。

RESPONSE_CONTENT_TYPE
読み取り専用。情報が POST のメソッドで送信される場合、この変数には組み込まれたデータのタイプが入ります。プロキシー・サーバー構成ファイル内に独自のコンテンツ・タイプを作成し、このコンテンツ・タイプをビューアーにマップすることができます。例えば text/html です。

RESPONSE_CONTENT_LENGTH
読み取り専用。情報が POST のメソッドで送信される場合、この変数にはデータの文字数が入ります。一般に、サーバーは、標準入力を用いて情報を転送するときは、ファイル終わりフラグを送信しません。必要な場合は、CONTENT_LENGTH 値を使用すると、入力ストリングの終わりを判別することができます。例えば 7034 です。

RULE_FILE_PATH
読み取り専用。構成ファイルの完全修飾ファイル・システム・パスおよびファイル名が入ります。

SSL_SESSIONID
読み取り専用。現在の要求が SSL 接続上で受信されたものである場合は、SSL セッション ID を戻します。現在の要求が SSL 接続上で受信されたものでない場合は NULL を戻します。

SCRIPT_NAME
要求の URL が入ります。

SERVER_ADDR
読み取り専用。プロキシー・サーバーのローカル IP アドレスが入ります。

SERVER_NAME
読み取り専用。この要求に関するコンテンツ・サーバーのプロキシー・サーバー・ホスト名または IP アドレスが入ります。例えば `www.ibm.com` です。

SERVER_PORT
読み取り専用。クライアント要求が送信されたプロキシー・サーバーのポート番号が入ります。例えば `80` です。

SERVER_PROTOCOL
読み取り専用。要求を行うときに使用されるプロトコルの名前とバージョンが入ります。例えば `HTTP/1.1` です。

SERVER_ROOT
読み取り専用。プロキシー・サーバー・プログラムがインストールされているディレクトリーが入ります。

SERVER_SOFTWARE
読み取り専用。プロキシー・サーバーの名前とバージョンが入ります。

STATUS
HTTP 応答コードおよび理由ストリングが入ります。例えば 200 OK です。

TRACE
情報をトレースする程度を判別します。戻り値は、以下のとおりです。

- OFF - トレースなし。
- V - 詳細モード。
- VV - さらに詳細なモード。
- MTV - 最も詳細なモード。

URI 読み取り/書き込み。DOCUMENT_URL と同じ。

URI_PATH
読み取り専用。URL のパス部分だけが戻されます。

URL 読み取り/書き込み。DOCUMENT_URL と同じ。

URL_MD4
読み取り専用。現行要求の潜在的なキャッシュ・ファイルのファイル名を戻します。

USE_PROXY
現行要求について、チェーニング対象のプロキシーを識別します。 URL を指定します。例えば `http://myproxy:8080` です。

USERID
REMOTE_USER と同じ。

USERNAME
REMOTE_USER と同じ。

認証および許可

まず、用語を簡単に説明します。

Authentication
要求側の ID を確認するための、この要求に関連するセキュリティー・トークンの検証。

Authorization
セキュリティー・トークンを用いて、リソースに対する要求側のアクセス権の有無を判別するプロセス。

39 ページの図 3 は、プロキシー・サーバーによる認証および許可のプロセスを表しています。

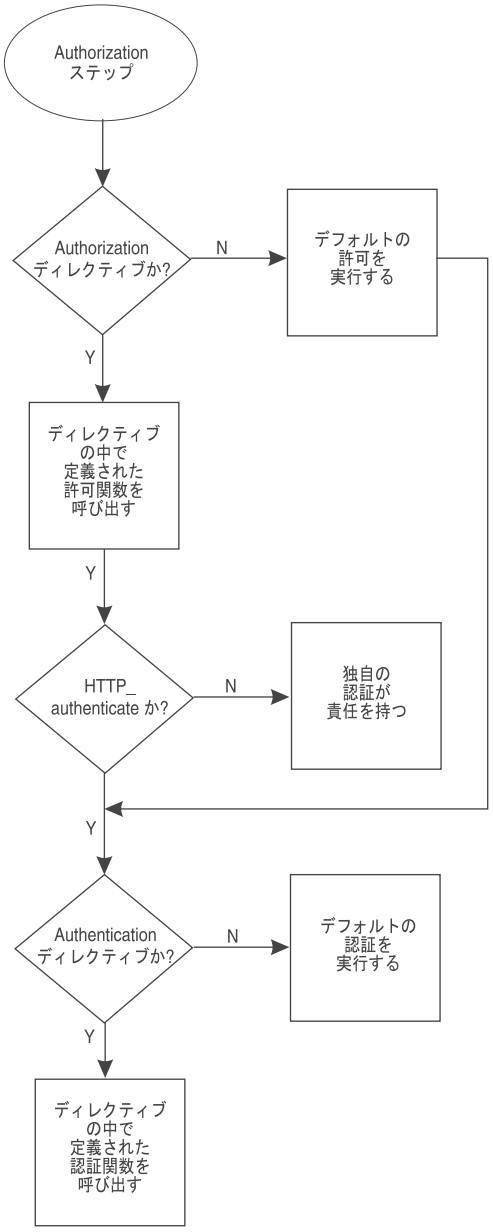

図 3. プロキシー・サーバーの認証および許可のプロセス

図 3 に示したように、サーバーによる許可と認証のプロセスの最初のステップは、許可プロセスの開始です。

Caching Proxy において、認証は許可プロセスの一部で、これが発生するのは、許可が必要な場合に限られます。

許可および認証プロセス

プロキシー・サーバーは、許可を必要とする要求を処理するとき以下のステップに従います。

1. 最初に、プロキシー・サーバーはその構成ファイルを検査し、許可ディレクティブがあるかどうかを判別します。

- 構成ファイルの中に許可ディレクティブが存在する場合、サーバーはそのディレクティブの中で定義されている許可関数を呼び出して、ステップ 2 で認証を開始します。
- 許可ディレクティブがない場合、サーバーはデフォルトの許可を実行し、その後、ステップ 3 の認証プロシージャーに直接進みます。

2. プロキシー・サーバーは、クライアント要求の中に HTTP_authenticate ヘッダーが存在するかどうかを検査することにより認証プロセスを開始します。
   - そのヘッダーが存在する場合、サーバーは認証プロセスを続行します (ステップ 3 を参照)。
   - そのヘッダーが存在しない場合は、別の方法で認証を行う必要があります。
3. プロキシー・サーバーは、プロキシー構成ファイルの中に認証ディレクティブが存在するかどうかを検査します。
   - 構成ファイルの中に認証ディレクティブが存在する場合、サーバーはそのディレクティブの中で定義されている認証関数を呼び出します。
   - ディレクティブがない場合、サーバーはデフォルトの認証を実行します。

Caching Proxy プラグインが独自の許可プロセスを備えている場合、そのプロセスがデフォルトのサーバーの許可および認証を*上書き* します。したがって、構成ファイルに許可ディレクティブがある場合には、それらに関連するプラグイン関数は、必要な認証を処理する必要もあります。これに使用するために、事前定義 HTTPD_authenticate() 関数が用意されています。

許可プラグインに認証を備えるには、次の 3 つの方法があります。

- 独自、かつ個別の許可プラグインおよび認証プラグインを作成します。プロキシー構成ファイルで、Authorization および Authentication の両ディレクティブを用いて、これらの関数を指定します。許可プラグイン関数には、HTTPD_authenticate() 関数呼び出しを組み込みます。

  Authorization ステップを実行すると、それが許可プラグイン関数を実行し、次にその許可が認証プラグイン関数を呼び出します。
- 独自の許可プラグイン関数を作成しますが、その関数がデフォルトのサーバー認証を呼び出せるようにします。プロキシー構成ファイルで、Authorization ディレクティブを用いて関数を指定します。この場合、Authentication ディレクティブは必要ありません。許可プラグイン関数で HTTPD_authenticate() 関数を呼び出してください。

  Authorization ステップを実行すると、それが許可プラグイン関数を実行し、次にその許可がデフォルト・サーバー認証を呼び出します。
- 独自の許可プラグイン関数を作成し、それにすべての必要な認証処理を組み込みます。許可プラグインに HTTPD_authenticate() 関数を使用しないでください。プロキシー構成ファイルで、Authorization ディレクティブを用いて許可プラグインを指定します。この場合、Authentication ディレクティブは必要ありません。

  Authorization ステップを実行すると、それが許可プラグイン関数と、その許可プラグイン関数に含まれる認証を実行します。

Caching Proxy プラグインが独自の許可プロセスを備えていない場合でも、次の方法を使用して、カスタマイズされた認証を提供することができます。

- 独自の認証プラグイン関数を作成します。プロキシー構成ファイルで、Authentication ディレクティブを用いて関数を指定します。この場合、Authorization ディレクティブは必要ありません。

Authorization ステップを実行すると、それがデフォルトのサーバー許可を実行し、次にその許可が認証プラグイン関数を呼び出します。

以下の点に注意してください。

- 構成ファイルに Authorization ディレクティブがない場合、または指定のプラグイン関数によって HTTP_NOACTION が戻されて要求の処理が拒否された場合は、サーバーのデフォルトの許可が行われます。
- 構成ファイルに Authorization ディレクティブが定義され、そのプラグイン関数に HTTPD_authenticate() が組み込まれている場合には、サーバーは Authentication ディレクティブで指定されたすべての認証関数を呼び出します。 Authentication ディレクティブを定義していない場合、またはそれらのディレクティブで指定したプラグイン関数によって HTTP_NOACTION が戻されて要求の処理が拒否された場合には、サーバーのデフォルトの認証が行われます。
- 構成ファイルに Authorization ディレクティブが存在していても、そのプラグイン関数に HTTPD_authenticate() が組み込まれていない場合、サーバーは認証関数を呼び出しません。許可プラグイン関数の一部として独自の認証処理を作成するか、他の認証モジュールに対して独自の呼び出しを行う必要があります。
- 許可の関数から `401` または `407` のコードが戻された場合、 Caching Proxy は自動的に身分証明要求を生成します (ユーザー ID とパスワードを戻すようにブラウザーに求める)。ただし、この場合でもこのアクションが正しく行われるように Caching Proxy で保護セットアップを構成する必要があります。

バリアント・キャッシュ

元の文書 (URI) が変更されたデータをキャッシュする場合は、バリアント・キャッシュを使用します。Caching Proxy は、API によって生成されたバリアントを処理します。バリアントとは、基本文書の別バージョンです。

一般に、起点サーバーはバリアントを送信するときに、それらがバリアントであることを識別できません。Caching Proxy は、プラグインによって作成されたバリアント (例えばコード・ページ変換) だけをサポートします。プラグインが HTTP ヘッダーに入っていない基準に基づいてバリアントを作成する場合は、プラグインに PreExit または PostAuthorization ステップ関数を組み込んで疑似ヘッダーを作成して、 Caching Proxy が既存のバリアントを正しく識別できるようにしなければなりません。

例えば、ブラウザーが送信する User-Agent ヘッダーの値に基づいてユーザーが要求したデータを変更するには、Transmogrifier API プログラムを使用します。 *close* 関数で、変更された内容をファイルに保管するか、またはバッファー長を指定してデータ引き数としてバッファーを渡します。その後で、バリアント・キャッシュ関数 httpd_variant_insert() および httpd_variant_lookup() を使用して、コンテンツをキャッシュに入れます。

API の例

Edge Components インストール CD-ROM の samples ディレクトリーに提供されているサンプル・プログラムを参照して、独自の Caching Proxy API 関数の作成に役立ててください。 WebSphere Application Server Web サイト www.ibm.com/software/webservers/appserv/ に追加情報があります。

第 3 章 カスタム・アドバイザー

このセクションでは、Load Balancer 用のカスタム・アドバイザーの作成について説明します。

アドバイザーによるロード・バランシング情報の提供

アドバイザーは、指定サーバー上のロードに関する情報を提供するために Load Balancer 内部で働くソフトウェア・エージェントです。各標準プロトコル (HTTP、SSL、その他) ごとに異なるアドバイザーが存在します。定期的に、 Load Balancer 基本コードがアドバイザー・サイクルを実行します。このサイクルで、構成内のすべてのサーバーの状況が個別に評価されます。

Load Balancer 用の独自のアドバイザーを作成することによって、サーバー・マシンのロードを判別する方法をカスタマイズできます。

標準アドバイザー機能

一般に、アドバイザーは、以下のようにロード・バランシングを可能にするために働きます。

1. 定期的に、アドバイザーは各サーバーとの接続をオープンし、サーバーに要求メッセージを送ります。メッセージの内容はサーバーで実行しているプロトコルに固有です。例えば、HTTP アドバイザーはサーバーに HEAD 要求を送ります。
2. アドバイザーはサーバーからの応答を listen します。応答を取得した後、アドバイザーはそのサーバーのロード値を計算して報告します。アドバイザーが異なればロード値を計算する方法も異なりますが、たいていの標準アドバイザーはサーバーが応答に要する時間を測定し、その値をロードとしてミリ秒単位で報告します。
3. アドバイザーは Load Balancer のマネージャー機能にロードを報告します。ロードはマネージャー報告書の「ポート」欄に表示されます。マネージャーはアドバイザーの報告したロードを管理者が設定した重みとともに使用して、サーバーへの着信要求のロード・バランシング方法を決定します。
4. サーバーが応答しない場合、アドバイザーはそのロードに対して負の値 (-1) を戻します。マネージャーはこの情報を使用して、特定のサーバーについてサービスを延期する時期を決定します。

Load Balancer に提供されている標準アドバイザーには、以下の機能のためのアドバイザーが含まれます。これらのアドバイザーに関する詳細情報は「*WebSphere Application Server Load Balancer 管理ガイド*」に記載されています。

- Connect
- DB2
- DNS
- FTP
- HTTP

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

標準アドバイザーが提供されていない所有プロトコルをサポートするには、カスタム・アドバイザーを作成しなければなりません。

カスタム・アドバイザーの作成

カスタム・アドバイザーはクラス・ファイルとして提供される小さい Java コードであり、これはサーバー上のロードを判別するために Load Balancer 基本コードによって呼び出されます。基本コードは、カスタム・アドバイザーのインスタンスの開始と停止、状況および報告書の提供、ヒストリー情報のログ・ファイルへの記録、さらにアドバイザー結果のマネージャー・コンポーネントへの報告など、必要なすべての管理サービスを提供します。

Load Balancer 基本コードがカスタム・アドバイザーを呼び出すと、以下のステップが実行されます。

1. Load Balancer 基本コードがサーバー・マシンとの接続をオープンします。
2. ソケットがオープンすると、基本コードは指定されたアドバイザーの GetLoad 関数を呼び出します。
3. アドバイザーの GetLoad 関数は、ユーザーがサーバーの状況を評価するために定義したステップ (サーバーからの応答を待つなど) を実行します。この関数は、応答が受け取られると実行を終了します。
4. Load Balancer 基本コードはサーバーとのソケットをクローズして、ロード情報をマネージャーに報告します。カスタム・アドバイザーが通常モードで作動するか置換モードで作動するかによって、基本コードは GetLoad 関数の終了後に追加の計算を行う場合があります。

通常モードと置換モード

カスタム・アドバイザーは、Load Balancer と*通常*モードまたは*置換*モードで対話するよう設計することができます。

操作モードの選択は、カスタム・アドバイザー・ファイルで、コンストラクター・メソッドのパラメーターとして指定されます。(各アドバイザーは設計に基づいて、上記のモードのいずれかでのみ稼動します。)

通常モードでは、カスタム・アドバイザーがサーバーとデータを交換し、基本アドバイザー・コードが交換の時間を測定して、ロード値を計算します。その後、基本コードがこのロード値をマネージャーに報告します。成功を示す場合は値ゼロが、またはエラーを示す場合は -1 がカスタム・アドバイザーから戻されます。

通常モードを指定するには、コンストラクター内の replace フラグを *false* に設定してください。

置換モードでは、基本コードはいかなる時間測定も行いません。カスタム・アドバイザーはその固有の要件に基づいて、指定された操作をすべて実行し、その後、実際のロード値を戻します。基本コードはそのロード値を受け入れ、それをそのままマネージャーに報告します。最良の結果を得るには、高速サーバーを表す 10 と低速サーバーを表す 1000 によって、ロード値を 10 から 1000 の間で正規化してください。

置換モードを指定するには、コンストラクター内の replace フラグを *true* に設定してください。

アドバイザー命名規則

カスタム・アドバイザー・ファイル名は ADV_*name*.java という形式に従わなければなりません。ここで *name* はアドバイザーのために選択する名前です。完全な名前は大文字の接頭部 ADV_ で始まり、後続の文字がすべての小文字でなければなりません。ここで小文字を使用することにより、アドバイザーを実行するためのコマンドでの大/小文字の区別が不要になります。

Java の規則に従って、ファイル内で定義されるクラスの名前はファイルの名前と一致しなければなりません。

コンパイル

カスタム・アドバイザーは Java 言語で作成し、そのコンパイルには、開発マシンにインストールされている Java コンパイラーを使用する必要があります。コンパイル時には、以下のファイルが参照されます。

- カスタム・アドバイザー・ファイル
- 基本クラス・ファイル ibmnd.jar (*install_path*/servers/lib ディレクトリーにあります)

コンパイル時には、クラスパス環境変数がカスタム・アドバイザー・ファイルと基本クラス・ファイルの両方を指していなければなりません。コンパイル・コマンドの形式は次のようになります。

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

この例では、デフォルトの Linux および UNIX インストール・パスが使用されています。アドバイザー・ファイルの名前が ADV_*name*.java で、アドバイザー・ファイルは現行ディレクトリーに保管されます。

コンパイルの出力は ADV_*name*.class などのクラス・ファイルです。アドバイザーを開始する前に、クラス・ファイルを *install_path*/servers/lib/CustomAdvisors/ ディレクトリーにコピーしてください。

注: カスタム・アドバイザーは、コンパイルした時のオペレーティング・システムとは別のオペレーティング・システム上で実行することができます。例えば、作成したアドバイザーを Windows システムでコンパイルし、結果のクラス・ファイル (バイナリー形式) を Linux マシンにコピーして、そこでカスタム・アドバイザーを実行することができます。

カスタム・アドバイザーの実行

カスタム・アドバイザーを実行するには、最初にアドバイザーのクラス・ファイルを Load Balancer マシン上の lib/CustomAdvisors サブディレクトリーにコピーする必要があります。例えば、myping というカスタム・アドバイザーの場合、ファイル・パスは、*install_path*/servers/lib/CustomAdvisors/ADV_myping.class になります。

Load Balancer を構成し、そのマネージャー機能を開始し、カスタム・アドバイザーを開始するコマンドを発行します。カスタム・アドバイザーは、次のように、その名前から ADV_ 接頭部とファイル拡張子を除いたものによって指定されます。

```
dscontrol advisor start myping port_number
```

コマンドに指定するポート番号は、アドバイザーがターゲット・サーバーとの接続をオープンするポートです。

必須なルーチン

すべてのアドバイザーと同様に、カスタム・アドバイザーは、ADV_Base と呼ばれるアドバイザー基本クラスの機能性を拡張します。アドバイザー・ベースは、マネージャーの重みアルゴリズムで使用するためにロードをマネージャーに報告するなど、アドバイザーの関数の大部分を実行します。アドバイザー・ベースはまた、ソケット接続およびクローズ操作を実行し、アドバイザーが使用する送信および受信メソッドを提供します。アドバイザーは、調査中のサーバー用に指定されたポートでデータを送信および受信するためだけに使用されます。アドバイザー・ベース内の TCP メソッドは、ロードを計算するために時間測定されます。アドバイザー・ベースのコンストラクター内のフラグが、既存のロードをアドバイザーから戻された新しいロードで上書きするかどうかを示します。

注: コンストラクターで設定された値に基づき、アドバイザー・ベースは指定された間隔でロードを重みアルゴリズムに提供します。アドバイザーが処理を完了せず、有効なロードを戻せない場合には、アドバイザー・ベースは前に報告されたロードを使用します。

アドバイザーには、以下の基本クラス・メソッドがあります。

- コンストラクター・ルーチン。このコンストラクターは基本クラス・コンストラクターを呼び出します。
- ADV_AdvisorInitialize メソッド。このメソッドは、基本クラスがその初期化を完了した後で追加のステップを実行する方法を提供します。

- getLoad ルーチン。基本アドバイザー・クラスがソケットのオープンを実行するので、getLoad 関数が適切な送信および受信要求を出すだけでアドバイザー・サイクルを完了することができます。

これらの必須ルーチンに関する詳細は、このセクションで後述します。

検索順序

カスタム・アドバイザーは、ネイティブ・アドバイザーまたは標準アドバイザーを検索した後に呼び出されます。 Load Balancer が指定のアドバイザーを標準アドバイザーのリスト中から見つけられない場合に、カスタム・アドバイザーのリストを調べます。アドバイザーの使用に関する追加情報は「*WebSphere Application Server Load Balancer 管理ガイド*」に記載されています。

命名およびファイル・パス

カスタム・アドバイザーの名前およびパスについては、以下の要件を忘れないでください。

- オペレーターがコマンド行からコマンドを入力するときに大/小文字を区別する必要がないようにするには、カスタム・アドバイザーの名前を英小文字にする必要があります。アドバイザー名には接頭部 ADV_ が必要です。
- カスタム・アドバイザー・クラスは、サブディレクトリー lib/CustomAdvisors 内に置かなければなりません。このディレクトリーのデフォルトのロケーションは、Linux および UNIX システムでは /opt/ibm/edge/lb/servers/lib/CustomAdvisors で、Windows システムでは C:¥Program Files¥IBM¥edge¥lb¥servers¥lib¥CustomAdvisors¥ になります。

カスタム・アドバイザー・メソッドおよび関数呼び出し

コンストラクター (advisor 基本によって提供される)

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

sName
カスタム・アドバイザーの名前。

sVersion
カスタム・アドバイザーのバージョン。

iDefaultPort
サーバーへの接続を行うポート番号 (ポート番号が呼び出しに指定されていない場合)。

iInterval
アドバイザーがサーバーを照会する間隔。

sDefaultLogFileName
このパラメーターは必須ですが、使用されません。許容値はヌル・ストリングである "" です。

replace
このアドバイザーが*置換* モードで機能するかどうかを示します。可能な値は、以下のとおりです。

- true - アドバイザー基本コードによって計算されるロードをカスタム・アドバイザーによって報告される値で置き換えます。
- false - カスタム・アドバイザーによって報告されるロード値をアドバイザー基本コードによって計算されるロード値に追加します。

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

このメソッドは、カスタム・アドバイザーに必要な初期化を実行するために提供されます。このメソッドが呼び出されるのはアドバイザー基本モジュールを開始した後です。

標準アドバイザーを含む多くの場合においてこのメソッドは使用されず、そのコードを構成するのは *return* ステートメントだけです。このメソッドは suppressBaseOpeningSocket メソッドを呼び出すために使用できます。suppressBaseOpeningSocket メソッドが有効であるのは上記のメソッド内から呼び出された場合のみです。

getLoad()

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

iConnectTime
接続の完了に要した時間 (ミリ秒)。このロード測定はアドバイザー基本コードによって実行されてカスタム・アドバイザー・コードに渡されます。ロード値を戻すときには、測定を使用することもまたは無視することもできます。接続が失敗すると、この値が -1 に設定されます。

caller
アドバイザー基本メソッドを提供するアドバイザー基本クラスのインスタンス。

カスタム・アドバイザーで使用可能な関数呼び出し

以下のセクションで説明するメソッドまたは関数は、カスタム・アドバイザーから呼び出せます。これらのメソッドはアドバイザー基本コードによってサポートされています。

一部の関数呼び出し (例えば *function_name* () など) は直接に作成できますが、その他は接頭部 caller を必要とします。*caller* は基本アドバイザー・インスタンスを示し、これは実行されるカスタム・アドバイザーをサポートします。

ADVLOG()

ADVLOG 関数により、カスタム・アドバイザーはテキスト・メッセージをアドバイザー基本ログ・ファイルに書き込むことができます。形式は次のとおりです。

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

logLevel
メッセージがログ・ファイルに書き込まれる状況レベル。アドバイザー・ログ・ファイルはステージで編成されます。最も緊急のメッセージには状況レベル 0 を指定し、緊急の度合いがそれより低いメッセージには大きい数値を指定します。メッセージの最も詳細なタイプには状況レベル 5 が指定されます。これらのレベルは、リアルタイムでユーザーが受け取るメッセージのタイプを制御するために使用されます (詳細の度合いを設定するには **dscontrol** コマンドが使用されます)。致命的なエラーは常にレベル 0 で記録する必要があります。

message
ログ・ファイルに書き込むメッセージ。このパラメーターの値は標準 Java ストリングです。

getAdvisorName()

getAdvisorName 関数は、カスタム・アドバイザー名の接尾部部分の Java ストリングを戻します。例えば、ADV_cdload.java という名前のアドバイザーでは、この関数は値 cdload を戻します。

この関数にはパラメーターを使用しません。

この値はアドバイザーのインスタンス化中に変更できないことに注意してください。

getAdviseOnPort()

getAdviseOnPort 関数は、呼び出しカスタム・アドバイザーを実行するポート番号を戻します。戻り値は、 Java 整数 (int) であり、この関数にはパラメーターを使用しません。

この値はアドバイザーのインスタンス化中に変更できないことに注意してください。

caller.getCurrentServer()

getCurrentServer 関数は、現行サーバーの IP アドレスを戻します。戻り値は標準小数点付き 10 進数形式の Java ストリングです (例えば 128.0.72.139 など)。

アドバイザー基本コードがすべてのサーバー・マシンを続けて照会するため、通常はカスタム・アドバイザーを呼び出すたびにこのアドレスが変更されます。

この関数にはパラメーターを使用しません。

caller.getCurrentCluster()

getCurrentCluster 関数呼び出しは、現行サーバー・クラスターの IP アドレスを戻します。戻り値は標準小数点付き 10 進数形式の Java ストリングです (例えば 128.0.72.139 など)。

アドバイザー基本コードがすべてのサーバー・クラスターを続けて照会するため、通常はカスタム・アドバイザーを呼び出すたびにこのアドレスが変更されます。

この関数にはパラメーターを使用しません。

getInterval()

getInterval 関数は、アドバイザー・サイクル間の秒数であるアドバイザー間隔を戻します。この値は、 **dscontrol** コマンドを使用して実行時に変更しない限り、カスタム・アドバイザーのコンストラクターに設定されたデフォルト値と同じです。

戻り値は Java 整数 (int) です。この関数にはパラメーターを使用しません。

caller.getLatestLoad()

getLatestLoad 関数により、カスタム・アドバイザーは指定サーバー・オブジェクトの最新ロード値を獲得できます。ロード値は、アドバイザー基本コードおよびマネージャー・デーモンによって内部テーブルで保守されています。

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

3 つの引き数は 1 つのサーバー・オブジェクトをともに定義します。

cluster_IP
現行ロード値を入手するサーバー・オブジェクトのクラスター IP アドレス。この引き数は、標準 IP アドレス表記の Java ストリングでなければなりません (例えば 245.145.62.81 など)。

port
現行ロード値を入手するサーバー・オブジェクトのポート番号。

server_IP
現行ロード値を入手するサーバー・オブジェクトの IP アドレス。この引き数は、標準 IP アドレス表記の Java ストリングでなければなりません (例えば 192.255.201.3 など)。

戻り値は整数です。

- 正の戻り値は、照会したオブジェクトに割り当てられた実際のロード値を表します。
- 値 -1 は、該当サーバーがダウンしていることを示します。
- 値 -2 は、該当サーバーの状況が不明であることを示します。

この関数呼び出しは、あるプロトコルまたはポートの動作を別のものの動作に依存させる場合に役立ちます。例えば、同一マシン上の Telnet サーバーが使用不可である場合に、特定アプリケーション・サーバーを使用不可にするカスタム・アドバイザーでこの関数呼び出しを使用する場合などです。

caller.receive()

receive 関数は、ソケット接続から情報を入手します。

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

パラメーター *response* は、検索されたデータが置かれるストリング・バッファーです。さらに、この関数は以下の重要度のある整数値を戻します。

- 0 はデータが正常に送信されたことを示します。
- 負数はエラーを示します。

caller.send()

send 関数は、指定ポートを使用してデータのパケットをサーバーに送信するために確立したソケット接続を使用します。

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

パラメーター *command* は、サーバーに送信するデータが入っているストリングです。この関数は、以下の重要度のある整数値を戻します。

- 0 はデータが正常に送信されたことを示します。
- 負数はエラーを示します。

suppressBaseOpeningSocket()

suppressBaseOpeningSocket 関数呼び出しにより、カスタム・アドバイザーの代わりに基本アドバイザー・コードがサーバーへの TCP ソケットをオープンするかどうかをカスタム・アドバイザーが指定できます。アドバイザーが状況を判別するためにサーバーとの直接通信を使用しない場合は、このソケットをオープンする必要はありません。

この関数呼び出しを出せるのは一度だけであり、ADV_AdvisorInitialize ルーチンから出さなければなりません。

この関数にはパラメーターを使用しません。

例

以下の例は、カスタム・アドバイザーをインプリメントできる方法を示します。

標準アドバイザー

このサンプル・ソース・コードは標準 Load Balancer HTTP アドバイザーに類似しています。以下のように機能します。

1. 送信要求 ″HEAD/HTTP″ コマンドが出されます。
2. 応答を受け取ります。情報は解析されませんが、応答により getLoad メソッドが終了します。
3. getLoad メソッドは成功を示す 0 または失敗を示す -1 を戻します。

このアドバイザーは通常モードで操作するので、ロード測定はソケット・オープン、送信、受信、およびクローズ操作を実行するために必要な経過時間 (ミリ秒) に基づきます。

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

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

  public ADV_sample() {
    super(ADV_NAME, "3.0.0.0-03.31.00",
```

```
          ADV_DEF_ADV_ON_PORT, ADV_DEF_INTERVAL, "",
          false);
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize() {
    return;                                  // usually an empty routine
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;       // initialize to inaccessible

    iRc = caller.send(ADV_SEND_REQUEST);     // send the HTTP request to
                                             // the server
    if (0 <= iRc) {                          // if the send is successful
      StringBuffer sbReceiveData = new StringBuffer("");   // allocate a buffer
                                                           // for the response
      iRc = caller.receive(sbReceiveData);   // receive the result

      // parse the result here if you need to

      if (0 <= iRc) {          // if the receive is successful
        iLoad = 0;             // return 0 for success
      }                        // (advisor's load value is ignored by
    }                          //  base in normal mode)
    return iLoad;
  }
}
```

サイド・ストリーム・アドバイザー

このサンプルでは、アドバイザー・ベースによる標準ソケットのオープンの抑制を例示しています。その代わりに、このアドバイザーはサイド・ストリーム Java ソケットをオープンしてサーバーを照会します。このプロシージャーは、通常のクライアント・トラフィックと異なるポートを使用してアドバイザー照会を listen するサーバーのために役立ちます。

この例では、サーバーはポート 11999 上で listen していて、照会されたときに 16 進 int ″4″ でロード値を戻します。このサンプルは置換モードで実行されます。つまり、アドバイザー・コンストラクターの最終パラメーターが true に設定されて、アドバイザー基本コードは経過時間ではなく戻されたロード値を使用します。

初期化ルーチンでの supressBaseOpeningSocket() に対する呼び出しに注意してください。データが送信されないときの基本ソケットの抑制は不要です。例えば、アドバイザーがサーバーに接続できることを確認するためにソケットをオープンする場合などです。この選択を行う前には、アプリケーションの必要性を注意深く調べてください。

```
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;

  // create an array of bytes with the load request message
  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);         // replace mode parameter is true
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // tell base code not to open the
                                   // standard socket
    return;
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;   // port on which to communicate with the server

    string sServer = caller.getCurrentServer();   // address of server to query
    try {
      socket soServer = new Socket(sServer, iControlPort);  // open socket to
                                                            // server
      DataInputStream disServer = new DataInputStream(
                                     soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                      soServer.getOutputStream());

      int iRecvTimeout = 10000;  // set timeout (in milliseconds)
                                 // for receiving data
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // send a message to the server
      dosServer.flush();

      iLoad = disServer.readByte();   // receive the response from the server

    } catch (exception e) {
      system.out.println("Caught exception " + e);
    }
    return iLoad;    // return the load reported from the server
  }
}
```

2 つのポート・アドバイザー

このカスタム・アドバイザー・サンプルは、サーバーの 1 つのポートに対する失敗を検出する機能を説明しています。これは、そのポートの状況と、同一サーバー・マシン上にある別のポート上で実行されている異なるサーバー・デーモンの状況の

両方に基づいています。例えば、ポート 80 の HTTP デーモンが応答を停止する場合には、ポート 443 の SSL デーモンへのルーティング・トラフィックも停止することができます。

このアドバイザーは、応答を送信しないサーバーは機能を停止したと見なして、ダウンのマークを付けるので、標準アドバイザーよりも積極的です。標準アドバイザーは応答のないサーバーを非常に低速であると見なします。このアドバイザーは HTTP ポートおよび SSL ポートのいずれかの応答がないと、両方のポートがダウンしたことを示すマークをサーバーに付けます。

このカスタム・アドバイザーを使用するには、アドバイザーの HTTP ポート上にあるインスタンスと SSL ポート上にあるインスタンスを管理者が開始します。アドバイザーは HTTP 用と SSL 用の 2 つの静的グローバル・ハッシュ・テーブルを検証します。各アドバイザーはそのサーバー・デーモンとの通信を試行し、そのハッシュ・テーブルにこのイベントの結果を保管します。各アドバイザーが基本アドバイザー・クラスに戻す値は、その固有のサーバー・デーモンと通信する能力およびそのデーモンと通信するパートナー・アドバイザーの能力によって異なります。

以下のカスタム・メソッドが使用されます。

- ADV_nte() はサーバーに関する情報を保持する単純な保管用オブジェクトです。これらのオブジェクトは、テーブル・エレメントとしてハッシュ・テーブルに保管されます。各オブジェクトには、エレメントが現行であるかどうかを判別するために使用するタイム・スタンプがあります。
- putNte() および getNte() は synchronized メソッドであり、2 つのアドバイザー・インスタンスが制御される方式でハッシュ・テーブルをアクセスすることを確実にします。
- getLoadHTTP は、HTTP サーバーの応答を照会するメソッドです。これは低レベルのルーチンであり、SSL に関する情報を収集または使用しません。
- getLoadSSL() は、SSL サーバーの応答を照会するメソッドです。これは低レベルのルーチンであり、HTTP に関する情報を収集または使用しません。
- getLoad() は、このカスタム・アドバイザーのエントリー・ポイント・ルーチンです。これは両方のプロトコルを処理でき、ハッシュ・テーブルからの情報を保管および取り出すことができます。これは 2 つのポートをリンクするルーチンです。

次のエラー条件が検出されます。

- 非応答サーバー・マシン — 基本アドバイザー・クラスは PING シグナルをサーバー・アドレスに定期的に送信します。アドレスが到達可能でない場合は、基本アドバイザー・クラスはサーバー・ダウンのマークを付けます。カスタム・アドバイザーの 2 つのインスタンスはどちらも呼び出されずに、そのマシン上の両サーバーはダウンのマークを付けられます。
- サーバー・マシン上の一方のデーモンが非応答になり、もう一方は作動します — 基本コードがサーバーとのソケットのオープンを試み、接続が拒否されると、このプロトコルの基本アドバイザーはサーバーにダウンのマークを付けます。そのプロトコルのカスタム・アドバイザー・コードは呼び出されません。もう一方のプロトコルのカスタム・アドバイザーがそのサーバーとの通信を継続しても、他方のカスタム・アドバイザーがサーバー・デーモンと通信できないことをハッシ

ュ・テーブルから通知されます。そのために 2 番目のプロトコルのアドバイザーもサーバーにダウンのマークを付けることになります。

- 一方のデーモンは応答を送信しませんが、もう一方のデーモンは送信します — 応答を送信しないプロトコルのカスタム・アドバイザーは通信の失敗を検出し、サーバーにダウンのマークを付け、さらにデータをハッシュ・テーブルに保管します。もう一方のポートのカスタム・アドバイザーは、ハッシュ・テーブルからの情報を得て、そのサーバーにダウンのマークを付けます。

このサンプルは HTTP 用のポート 80 および SSL 用の 443 をリンクするように書かれていますが、ポートの組み合わせは任意に調整できます。

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

//--------
// Define the table element for the hash tables used in this custom advisor

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

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

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

//--------
// check whether this element is current or expired
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();   // set lifetime as
                                                        // 3 advisor cycles
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

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

//--------
// value accessor(s)

  public int getLoadValue() { return iLoad; }
```

```
//--------
// clone (avoids corruption between threads)

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

}

//--------
// define the custom advisor

public class ADV_twop extends ADV_Base
   implements ADV_MethodInterface, ADV_AdvisorVersionInterface {

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

//--------
  // define tables to hold port-specific history information

  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";

  //--------
  // create byte array with SSL client hello message

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
    (byte)0x01,               // client hello
    (byte)0x03, (byte)0x00,   // SSL version
    (byte)0x00, (byte)0x03,   // cipher spec len (bytes)
    (byte)0x00, (byte)0x00,   // session ID len (bytes)
    (byte)0x00, (byte)0x10,   // challenge data len (bytes)
    (byte)0x00, (byte)0x00, (byte)0x03,   // cipher spec
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)Ox20,  // challenge data
    (byte)0xFD, (byte)0x3A, (byte)0x3C, (byte)0x18,
    (byte)0xAB, (byte)0x67, (byte)0xB0, (byte)0x52,
    (byte)0xB1, (byte)0x1D, (byte)0x55, (byte)0x44, (byte)0x0D, (byte)0x0A };

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

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);    // false = load balancer times the response
    setAdvisor ( this );
  }

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

  public void ADV_AdvisorInitialize() {
    return;
```

```
 }

 //--------
 // synchronized PUT and GET access routines for the hash tables

 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 - determine HTTP load based on server response

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

   int iRc = caller.send(ADV_HTTP_REQUEST_STRING);  // send request message
                                                    // to server
   if (0 <= iRc) {           // did the request return a failure?
     StringBuffer sbReceiveData = new StringBuffer("")    // allocate a buffer
                                                          // for the response
     iRc = caller.receive(sbReceiveData);    // get response from server

     if (0 <= iRc) {             // did the receive return a failure?
       if (0 < sbReceiveData.length()) {      // is data there?
         iLoad = SUCCESS;        // ignore retrieved data and
                                 // return success code
       }
     }
   }
   return iLoad;
 }

 //--------
 // getLoadSSL() - determine SSL load based on server response

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

   int iSocket = caller.getAdvisorSocket();   // send hex request to server
   CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                   abClientHello);
   int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

   if (0 <= iRc) {    // did the request return a failure?
     StringBuffer sbReceiveData = new StringBuffer("");  // allocate buffer
                                                         // for the response
     iRc = caller.receive(sbReceiveData);     // get a response from
                                              // the server
     if (0 <= iRc) {     // did the receive return a failure?
       if (0 < sbReceiveData.length()) {    // is data there?
         iLoad = SUCCESS;   // ignore retrieved data and return success code
       }
     }
   }
   return iLoad;
 }

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

```
// getLoad - merge results from the HTTP and SSL methods

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

  String sCluster = caller.getCurrentCluster();   // current cluster address
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;     // hash table key

  if (ADV_TWOP_PORT_HTTP == iPort) {              // handle an HTTP server
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // get the load for HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);  // save HTTP load
                                                    // information
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // get SSL
                                                          // information
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) {         // check the time stamp
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // is SSL
                                                                 // working?
          iLoad = iLoadHTTP;
        } else {    // SSL is not working, so mark the HTTP server down
          iLoad= ADV_HOST_INACCESSIBLE;
        }
      } else {      // SSL information is expired, so mark the
                    // HTTP server down
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {        // no load information about SSL, report
                    // getLoadHTTP() results
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {           // handle an SSL server
    iLoadSSL = getLoadSSL(iConnectTime, caller);   // get load for SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL);   // save SSL load info.

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);   // get HTTP
                                                             // information
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) {       // check the timestamp
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {  // is HTTP
                                                                // working?
          iLoad = iLoadSSL;
        } else {   // HTTP server is not working, so mark SSL down
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {     // expired information from HTTP, so mark SSL down
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {       // no load information about HTTP, report
                   // getLoadSSL() results
      iLoad = iLoadSSL;
    }
  }

//--------
// error handler

  else {
```

```
      iLoad = ADV_HOST_INACCESSIBLE;
    }
    return iLoad;
  }
}
```

WebSphere Application Server advisor

WebSphere Application Server のサンプル・カスタム・アドバイザーは、*install_path*/servers/samples/CustomAdvisors/ ディレクトリーに入っています。完全なコードはこの資料では掲載していません。

- ADV_was.java は、コンパイルされた、Load Balancer マシン上で実行されるアドバイザーのソース・コード・ファイルです。
- LBAdvisor.java.servlet は LBAdvisor.java への名前変更、コンパイル、および WebSphere Application Server マシン上での実行が必要であるサーブレット・ソース・コードです。

完全なアドバイザーはサンプルよりわずかに複雑です。上記の StringTokenizer の例よりコンパクトな特殊化された構文解析ルーチンを追加します。

サンプル・コードの複雑な部分は Java サーブレットにあります。多くのメソッドの中で、サーブレット仕様が必要とする init() および service() という 2 つのメソッドと、 Java.lang.thread クラスが必要とする run() という 1 つのメソッドが、サーブレットに入っています。

- init() は、初期化時にサーブレット・エンジンが一度呼び出します。このメソッドは、アドバイザーからの呼び出しを個別に実行して処理ループを再開するまでの時間はスリープする、_checker という名前のスレッドを作成します。
- service() は、サーブレットを起動するたびにサーブレット・エンジンが呼び出します。この場合には、メソッドはアドバイザーによって呼び出されます。service() メソッドは ASCII 文字のストリームを出力ストリームに送信します。
- run() にはコード実行のコアが入ります。これは、init() メソッド内部から呼び出される start() メソッドが呼び出します。

サーブレット・コードの関連フラグメントは、以下のようになります。

```
...

  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;
}

...
```

アドバイザーから戻されるデータの使用

アプリケーション・サーバーの既存パーツに対する標準呼び出しを使用したり、またはカスタム・アドバイザーのサーバー側で相対するコードの新規部分を追加して、戻されたロード値を調べてサーバー動作を変更することができます。 Java StringTokenizer クラスおよびその関連メソッドはこの調査の実行を簡単にします。

通常の HTTP コマンドのコンテンツは `GET /index.html HTTP/1.0` です。

このコマンドに対する通常の応答は、以下のようになります。

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

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

関心のある項目 (特に HTTP 戻りコード) は先頭行に入っています。

HTTP 仕様は戻りコードを分類し、以下のように要約できます。

- 2xx 戻りコードは正常に実行された
- 3xx 戻りコードは宛先変更された
- 4xx 戻りコードはクライアント・エラーである
- 5xx 戻りコードはサーバー・エラーである

サーバーが戻す可能性のあるコードが正確に分かる場合は、コードをこの例ほど詳細にする必要はありません。ただし、検出する戻りコードを制限すると、プログラムの将来の柔軟性を制限することになる場合があります。

以下の例は、最小限の HTTP クライアントが入っているスタンドアロン Java プログラムです。この例は HTTP 応答を調べるための単純な汎用パーサーを起動します。

```
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());
    }
  }
}
```

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものであり、本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
ATTN: Software Licensing
11 Stanwix Street
Pittsburgh, PA 15222-9183
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、 IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

商標

以下は、IBM Corporation の商標です。

- AIX
- IBM
- ViaVoice

- WebSphere

Java およびすべての Java 関連の商標およびロゴは、Sun Microsystems, Inc. の米国およびその他の国における商標または登録商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

Intel、Intel Inside (ロゴ)、MMX および Pentium は、Intel Corporation の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Linux は、Linus Torvalds の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ハ行]

[マ行]

[ラ行]

[数字]

A

C

E

G

H

I

L

M

N

O

P

R

S

T

W

Printed in Japan

GC88-7045-01

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12