WebSphere Application Server

Руководство по программированию для Edge Components

Версия 6.1

GC43-0477-00

WebSphere Application Server

Руководство по программированию для Edge Components

Версия 6.1

GC43-0477-00

Примечания

Перед началом работы с этой информацией и описанным в ней продуктом ознакомьтесь с общей информацией, приведенной в разделе “Примечания” на стр. 61.

Первое издание (май 2006 года)

Настоящее издание относится к:
WebSphere Application Server версии 6.1

а также ко всем последующим выпускам и модификациям, если в последующих изданиях не будет оговорено противное.

Для заказа публикаций обратитесь в местное представительство или филиал IBM.

© Copyright International Business Machines Corporation 2005. Все права защищены.

Содержание

© Copyright IBM Corp. 2005

Рисунки

© Copyright IBM Corp. 2005

О книге

В этом разделе описываются цели, структура и условные обозначения, используемые в документе *WebSphere Application Server Programming Guide for Edge Components*.

Для кого предназначена данная книга

В этой книге описываются прикладные программные интерфейсы (API), доступные для настройки продукта Edge components сервера WebSphere Application Server версии 6.1. Эти сведения предназначены для разработчиков программных модулей. Эта информация также может заинтересовать проектировщиков сетей и системных администраторов, которые могут найти здесь описание возможностей по настройке данного продукта.

Необходимые знания

Для понимания информации, приведенной в данной книге, необходимо иметь представление о приемах программирования на языках Java или C, в зависимости от используемого API. Методы и структуры для каждого из этих интерфейсов описаны в соответствующей документации, но вы должны уметь создавать собственные приложения, а также компилировать и тестировать их. Для некоторых интерфейсов приводятся примеры программ, однако их можно использовать только как руководство для создания собственных приложений.

Обозначения и терминология

В этой документации используются следующие обозначения.

Таблица 1. Обозначения в книге

Обозначение	Значение
Полужирное начертание	Относится к меню, элементам меню, меткам, кнопкам, значкам и папкам графического интерфейса пользователя (GUI). Также применяется для выделения названий команд в обычном тексте.
`Интервал`	Обозначает текст, который необходимо ввести в командной строке. Также применяется для обозначения текста на экране, примеров программ и частей файлов.
Курсив	Применяется для обозначения значений переменных (например, вместо переменной *ИмяФайла* необходимо указать имя файла). Также используется в заголовках книг, для выделения текста и пр.
Ctrl-*x*	Где *x* - это название клавиши или комбинация клавиш. Например, Ctrl-c означает нажать клавишу C, удерживая нажатой клавишу Ctrl.
Return	Применяется для обозначения клавиш Return, Enter или левой стрелки.
`%`	Обозначает командную строку в операционных системах Linux и UNIX, для использования которой полномочия пользователя root не требуются.
`#`	Обозначает командную строку в операционных системах Linux и UNIX, для использования которой требуются полномочия пользователя root.
`C:\`	Обозначает командную строку Windows.
Ввод команд	Если указано “ввести” или “выполнить” команду, введите команду и нажмите клавишу Return. Например, инструкция “Введите команду **ls**” означает, что в командной строке необходимо ввести **ls** и нажать клавишу Return.
[ ]	Обозначает необязательные параметры в синтаксисе команды.

© Copyright IBM Corp. 2005

Таблица 1. Обозначения в книге (продолжение)

Обозначение	Значение
{ }	Обозначает списки для выбора параметров команды.
|	Отделяет допустимые значения параметров, заключенных в { } (скобки) в синтаксисе команды.
...	Многоточие в синтаксисе команды означает, что предыдущий элемент можно ввести еще несколько раз. В примерах программ многоточие означает опущенный для краткости исходный код.

Специальные функции доступа

Специальные функции доступа позволяют работать с программой людям с ограниченными физическими возможностями, например, с ограниченной подвижностью или со сниженным зрением. В WebSphere Application Server версии 6.1 предусмотрены следующие специальные функции доступа:

- Возможность использования программ чтения с экрана и цифрового синтезатора речи для того, чтобы прослушать текст, отображенный на экране. Можно также применять программы распознавания речи, такие как IBM ViaVoice, чтобы вводить данные и осуществлять навигацию по пользовательскому интерфейсу.
- Возможность выполнения любых операций с помощью клавиатуры, без использования мыши.
- Возможность настройки и администрирования компонентов сервера приложений с помощью обычного текстового редактора или с помощью интерфейса командной строки. Дополнительные сведения о работе с отдельными компонентами можно найти в документации по этим компонентам.

Связанные документы и Web-сайты

- *Concepts, Planning, and Installation for Edge Components*, GC43-0478-00
- *Caching Proxy Administration Guide*, GC31-6920-00
- *Load Balancer Administration Guide*, GC31-6921-00
- Web-сайт корпорации IBM *www.ibm.com/*
- IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/*
- Web-сайт библиотеки IBM WebSphere Application Server*www.ibm.com/software/webservers/appserv/library.html*
- Web-сайт службы поддержки IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/support.html*
- IBM WebSphere Application Server Information Center *www.ibm.com/software/webservers/appserv/infocenter.html*
- IBM WebSphere Application Server Edge Components Information Center *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

Ждем ваших отзывов

Ваши отзывы помогают нам обеспечивать высокое качество и точность публикуемой информации. Если у вас есть комментарии по этой книге или другой документации для Edge components сервера WebSphere Application Server:

- Отправьте их по электронной почте: fsdoc@us.ibm.com. Не забудьте указать название книги, номер раздела, версию WebSphere Application Server и, по возможности, точное расположение текста (например, номер страницы или номер таблицы).

Глава 1. Общие сведения о настройке Edge components

В этой книге обсуждаются прикладные программные интерфейсы (API), предоставляемые для продукта Edge components сервера WebSphere Application Server. (Продукт Edge components для сервера WebSphere Application Server включает компоненты Caching Proxy и Load Balancer.) Для настройки взаимодействия между Edge components, а также с другими системами администраторы могут использовать различные интерфейсы.

ВАЖНЫЕ ИСКЛЮЧЕНИЯ:

- Компонент Caching Proxy недоступен при установке Edge Components на 64-разрядных процессорах Itanium 2 или AMD Opteron.
- Caching Proxy не устанавливается с экземплярами Edge component, работающими с Load Balancer для IPv4 и IPv6.

API, описанные в этом документе, относятся к нескольким категориям.

Настройка компонента Caching Proxy

Компонент Caching Proxy включает несколько интерфейсов, записанных в последовательность обработки, в которую можно добавлять пользовательские функции или использовать вместо стандартной обработки. Настройка включает изменение или расширение следующих задач:

- Идентификация клиента
- Идентификация запроса
- Преобразование URL в физические пути к файлам
- Обслуживание запросов
- Ведение протоколов
- Реагирование на условия возникновения ошибок

Пользовательские приложения (или модули Caching Proxy) вызываются в определенные моменты последовательности обработки сервера Proxy.

API компонента Caching Proxy применяется для реализации определенных функций системы. Например, поддержка LDAP реализована в виде модуля.

Глава 2, “API Caching Proxy”, на стр. 3 подробно описывает интерфейс и содержит инструкции по настройке сервера Proxy для работы с модулями.

Настройка компонента Load Balancer

Для настройки компонента Load Balancer используются пользовательские советники. Советники измеряются фактическое значение нагрузки на серверах. Пользовательские советники позволяют использовать собственные методы измерения нагрузки в конкретной системе. Эта функция особенно для полезна для измененных или специализированных Web-серверов.

Глава 3, “Пользовательские советники”, на стр. 41 содержит подробную информацию о разработке и использовании пользовательских советников. Также содержит примеры программ советников.

© Copyright IBM Corp. 2005

Поиск примера программы

Примеры программы для этих API можно найти на компакт-диске Edge Components в каталоге samples. Дополнительные примеры программы доступны на Web-сайте WebSphere Application Server, www.ibm.com/software/webservers/appserv/

Глава 2. API Caching Proxy

В этом разделе рассказано об интерфейсах прикладного программирования (API) Caching Proxy, т.е. о том, что это такое, зачем оно нужно и как работает.

ВАЖНАЯ ИНФОРМАЦИЯ: Caching Proxy может присутствовать во всех устанавливаемых конфигурациях Edge Components, за исключением следующих случаев:

- Caching Proxy не устанавливается с экземплярами Edge component, работающими на 64-разрядных процессорах AMD Opteron или Itanium 2.
- Caching Proxy не устанавливается с экземплярами Edge component, работающими с Load Balancer для IPv4 и IPv6.

Обзор API Caching Proxy

API - это интерфейс Caching Proxy, позволяющий вам расширять базовые возможности сервера Proxy. Вы можете создавать расширения или встраиваемые модули для реализации настраиваемых алгоритмов работы. Вот некоторые примеры применения API:

- Расширение базовой процедуры идентификации или замена ее на процедуру, общую для всего сайта.
- Добавление процедур обработки ошибок для отслеживания неполадок или предупреждения о потенциально опасных ситуациях.
- Выявление и отслеживание информации, поступающей от запрашивающих клиентов, включая, например, адреса ссылающихся серверов и коды браузеров.

API Caching Proxy предоставляет следующие преимущества:

- **Эффективность**
  - API разработаны специально для систем с многонитевой обработкой, используемых компонентом Caching Proxy.
- **Гибкость**
  - API поддерживают множество самых разных функций.
  - API не зависят от платформы и применяемого языка. Они работают на всех платформах, поддерживаемых Caching Proxy, а приложения встраиваемых модулей можно писать почти на любом языке программирования, поддерживаемом применяемой платформой.
- **Простота применения**
  - Простые типы данных передаются по ссылке, а не по значению (например, long *, char *).
  - Каждая функция имеет фиксированное число параметров.
  - Включены компоненты связывания языка C.
  - Встраиваемые модули не влияют на объем выделяемой памяти; они запрашивают и освобождают память независимо от других процессов Caching Proxy.

© Copyright IBM Corp. 2005

Процедура написания программ API

Перед написанием собственных программ модулей Caching Proxy необходимо ознакомиться с принципом работы сервера proxy. Работу сервера proxy при обработке запросов можно набить на несколько отдельных этапов. Для каждого из этих этапов вы можете указать с помощью API собственные функции. Например, хотите ли выполнить какие-либо действия после считывания клиентского запроса, но перед выполнением каких-либо других операций? Или, может быть, вам требуется выполнить какие-либо особые процедуры во время идентификации и повторить их после отправки запрошенного файла?

API предоставляют библиотеку заранее определенных функций. Ваши программы модулей могут вызывать библиотечные функции API для взаимодействия с процессами сервера proxy (например, для манипуляций с запросами, считывания и записи заголовков запросов, либо для занесения записей в протоколы сервера proxy). Эти библиотечные функции не следует путать с вашими собственными функциями модулей, вызываемыми сервером proxy. Заранее определенные функции описаны в разделе “Заранее определенные функции и макросы” на стр. 17.

Указать серверу proxy, какие из ваших функций модулей должны вызываться на каждом из этапов обработки, можно с помощью директив API Caching Proxy, заданных в файле конфигурации сервера. Эти директивы описаны в разделе “Директивы конфигурации Caching Proxy для этапов обработки API” на стр. 24.

В настоящий документ включены следующие сведения:

- Общая информация об этапах обработки Caching Proxy, допускающих настройку (см. раздел “Этапы обработки сервера”)
- Рекомендации по написанию модулей (см. раздел “Рекомендации” на стр. 8)
- Прототипы настраиваемых функций, которые вы можете написать для каждого выполняемого сервером этапа обработки, а также описание их кодов возврата (см. раздел “Прототипы функций модулей” на стр. 10)
- Определения заранее определенных функций и макросов, которые вы можете вызывать из модулей, а также описания их кодов возврата (см. раздел “Заранее определенные функции и макросы” на стр. 17)
- Директивы конфигурации API Caching Proxy (см. раздел “Директивы конфигурации Caching Proxy для этапов обработки API” на стр. 24)

С помощью этих компонентов и процедур вы сможете создавать собственные программы модулей Caching Proxy.

Этапы обработки сервера

Работу сервера proxy можно разделить на несколько этапов в соответствии с типом обработки, выполняемой на каждом из этих этапов. Каждый этап включает промежуточную точку, на которой может быть выполнена указанная часть вашей программы. Добавляя в файл конфигурации Caching Proxy (ibmproxy.conf) директивы API, вы указываете, какие функции модуля должны вызываться на каждом этапе. Указав для одного этапа несколько директив, вы можете вызывать для этого этапа несколько функций модуля.

Некоторые этапы входят в состав процесса обработки запроса сервером. Другими словами, сервер proxy выполняет эти этапы каждый раз, когда он обрабатывает полученный запрос. Другие этапы выполняются независимо от обработки запросов, т.е. сервер выполняет их независимо от того, выполняется какой-либо запрос или нет.

Ваши откомпилированные программы хранятся в общем объекте, например в файле DLL или .so, в зависимости от операционной системы. По мере выполнения этапов обработки запроса сервер вызывает связанные с каждым этапом функции модулей до тех пор, пока одна из функций не укажет, что она обработала запрос. Если вы указали для какого-либо этапа несколько функций модулей, то эти функции вызываются в том порядке, в котором соответствующие директивы указаны в файле конфигурации.

Если запрос не обрабатывается функцией модуля (из-за того, что вы не включили директиву API Caching Proxy для соответствующего этапа или функция модуля для этого этапа вернула значение HTTP_NOACTION), то сервер выполняет действие по умолчанию для этого шага.

Примечание: Это верно для всех этапов, за исключением этапа Service, для которого не предусмотрено действие по умолчанию.

рис. 1 на стр. 6 иллюстрирует этапы обработки сервера proxy и определяет последовательность выполнения этапов, связанных с обработкой запросов.

Рисунок 1. Диаграмма этапов сервера proxy

Четыре из показанных на диаграмме этапов выполняются независимо от обработки запросов клиентов. Эти этапы связаны с работой и обслуживанием сервера proxy. К их числу относятся:

- Инициализация сервера
- Полночь
- Советник GC
- Завершение работы сервера

В следующем списке разъясняется назначение каждого этапа, изображенного на рис. 1. Обратите внимание, что не все этапы обязательно выполняются при обработке каждого запроса.

ServerInit

Выполняет инициализацию при запуске сервера proxy и перед получением каких-либо запросов клиентов.

Midnight

Запускает модуль в полночь, вне контекста запроса. Этот этап показан на диаграмме отдельно, поскольку он не является частью обработки запроса; другими словами, его выполнение не связано с каким-либо конкретным запросом.

GCAdvisor

Влияет на решения о сборе мусора при обработке файлов, находящихся в кэше. Этот этап показан на диаграмме отдельно, поскольку он не является частью обработки запроса; другими словами, его выполнение не связано с каким-либо конкретным запросом. Сбор мусора выполняется при достижении кэшем максимального размера. (Информация о конфигурации сбора мусора в кэше приведена в руководстве *WebSphere Application Server Caching Proxy Administration Guide*.)

PreExit

Выполняется после чтения запроса, но перед любыми другими действиями.

Если по завершении обработки этот этап указывает, что запрос обработан (HTTP_OK), то сервер пропускает остальные этапы обработки запроса и выполняет только этапы Transmogrifier, Log и PostExit.

NameTrans

Преобразует виртуальный путь (из URL) в физический.

Проверка прав доступа

Использует сохраненные маркеры системы безопасности для проверки доступа к физическим каталогам, ACL, других средств управления доступом, а также для формирования заголовков WWW-Authenticate, необходимых для проведения базовой идентификации. Если вы создаете собственную функцию модуля, замещающую этот этап, то вы должны самостоятельно сформировать эти заголовки.

Дополнительная информация приведена в разделе “Идентификация и проверка прав доступа” на стр. 35.

Идентификация

Декодирует, проверяет и сохраняет маркеры системы безопасности.

Дополнительная информация приведена в разделе “Идентификация и проверка прав доступа” на стр. 35.

ObjectType

Находит объект файловой системы с заданным путем.

PostAuthorization

Выполняется после проверки прав доступа и обнаружения объекта, но перед собственно выполнением запроса.

Если по завершении обработки этот этап указывает, что запрос обработан (HTTP_OK), то сервер пропускает остальные этапы обработки запроса и выполняет только этапы Transmogrifier, Log и PostExit.

Service

Выполняет запрос (отправляя файл, запуская программу CGI и т.д.)

Proxy Advisor
Определяет принятие решений о кэшировании и работе proxy.

Transmogrifier
Предоставляет доступ на запись к элементу данных отправляемого клиенту ответа.

Log Обеспечивает ведение настраиваемых протоколов транзакций.

Error Обеспечивает настраиваемые процедуры обработки ошибок.

PostExit
Выполняет очистку ресурсов, выделенных для обработки запроса.

Server Termination
Выполняет очистку при обычном завершении работы сервера.

Рекомендации

- При написании собственных программ руководствуйтесь синтаксическими диаграммами и рекомендациями, приведенными для функций модуля сервера. Функциям модуля необходимо присваивать уникальные имена и при необходимости вызывать в них заранее определенные функции сервера.

  В системах AIX вам потребуется файл экспорта (например, libmyapp.exp), в котором перечислены функции модуля. Кроме того, необходимо будет выполнить связывание с файлом импорта API Caching Proxy libhttpdapi.exp.

  В системах Linux, HP-UX и Solaris необходимо выполнить связывание с библиотеками libhttpdapi и libc.

  В системах Windows вам потребуется файл определения модуля (.def), в котором перечислены функции модуля. Кроме того, необходимо будет выполнить связывание с файлом HTTPDAPI.LIB.

  Обязательно включите в определения функций файл HTAPI.h и используйте макрос HTTPD_LINKAGE. Этот макрос обеспечивает единый способ вызова всех функций.
- Сервер работает в многонитевой среде и ваши модули должны обеспечивать надежную работу в такой среде. Производительность реентерабельных приложений не снижается.
- Все выполняемые в модулях действия не должны влиять на возможность работы в многонитевой среде. Не выполняйте никаких действий, относящихся к области действия процесса, включая, например, выход, изменение ИД пользователя или регистрацию обработчика сигнала.
- Не используйте глобальные переменные, а если такая необходимость все-таки возникает, то защищайте их с помощью семафора взаимного исключения.
- При отправке данных клиенту с помощью функции HTTPD_write() обязательно задавайте заголовок Content-Type.
- Всегда проверяйте коды возврата и при необходимости обеспечивайте условную обработку.
- Выполняйте компиляцию и связывание программ с помощью документации по применяемому компилятору. В результате вы должны получить общий объект (например, DLL или файл .so) в соответствии с используемой операционной системой.

  В качестве руководства используйте следующие команды компиляции и связывания.

  – В системах **AIX**, с помощью IBM CSet++

    - **Компиляция:**

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

- **Связывание:**

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

  (Эта команда показана на двух строках лишь для более удобного чтения.)

– В системах **HP-UX** с помощью HP C/ANSI C Developer's Bundle и компилятора HP aC++

- **Компиляция:**

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

- **Связывание:**

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

– В системах **Linux** с помощью компилятора C Gnu (GCC) версии 3.2.X

- **Компиляция:**

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

- **Связывание:**

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

– В системах **Solaris** с помощью Sun Workshop

- **Компиляция:**

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

- **Связывание:**

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

– В системах **Windows** с помощью Microsoft Visual C++

- **Компиляция:**

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

- **Связывание:**

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

Для создания списка экспорта воспользуйтесь одним из следующих методов:

– Добавьте в исходный текст определения _declspec(dllexport)

– Укажите в командной строке LIB опцию /EXPORT:entryname

– Создайте файл определения модуля с оператором EXPORTS

- Добавьте в файл конфигурации директивы API Caching Proxy, связав функции модулей с соответствующими этапами обработки. Для каждого этапа обработки запроса сервером существует собственная директива. Для применения новых директив остановите и снова запустите сервер.

  Примечание: Caching Proxy не выгружает общие объекты (файлы DLL или .so) даже при перезапуске. Для освобождения общих объектов необходимо остановить сервер, а затем снова запустить его.

- Всесторонне тестируйте свои программы перед началом их использования в рабочей среде. Поскольку Caching Proxy - это сервер, работающий в многонитевой среде, то может потребоваться более строгое и многостороннее тестирование, чем при использовании обычного сервера с ветвлением. Ошибки в вашей программе могут привести к сбою сервера proxy, поскольку этот сервер вызывает вашу программу непосредственно и работает с ней в одном пространстве процессов.

Функции модулей

При написании собственных функций для вызова на различных этапах обработки запросов следуйте синтаксическим диаграммам, показанным в “Прототипы функций модулей” на стр. 10.

Каждая функция должна указывать в параметре кода возврата значение, указывающее на выполненное действие:

- Код HTTP_NOACTION (значение 0) указывает, что действия не были выполнены. При получении такого кода возврата сервер proxy выполняет действие по умолчанию для данного этапа.
- Допустимый код возврата HTTP указывает, что функция модуля выполнила обработку этапа. (Список допустимых кодов возврата приведен в разделе “Значения и коды возврата HTTP” на стр. 16.) Если передан допустимый код возврата HTTP, то другие функции модулей, указанные для данного этапа обработки запроса, не вызываются.

Прототипы функций модулей

В описании прототипов функций для каждого этапа обработки Caching Proxy указаны применяемые форматы и рассказано о выполняемых операциях. Обратите внимание, что имена функций не заданы заранее. Вы должны присвоить своим функциям уникальные имена, причем можете следовать собственному соглашению о присвоении имен. Для простоты в этом документе применяются имена, соответствующие этапам обработки.

В каждой из функций модуля допустимы определенные библиотечные функции API. Некоторые функции допустимы не на всех этапах. Следующие библиотечные функции API можно вызывать их всех функций модулей:

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- HTTPD_log*

Дополнительные допустимые и недопустимые функции API перечислены в описаниях прототипов функций.

Значение переданного вашей функции параметра *handle* можно передать в виде первого аргумента библиотечной функции. Библиотечные функции API описаны в разделе “Заранее определенные функции и макросы” на стр. 17.

ServerInit

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

Определенная для этого этапа функция вызывается один раз во время загрузки модуля при инициализации сервера. В этот момент вы можете выполнить собственные действия по инициализации перед началом обработки запросов.

Несмотря на то, что должны вызываться все функции инициализации сервера, возврат кода ошибки функцией на этом этапе приводит к тому, что все остальные функции, настроенные в том же модуле, что и функция, вернувшая код ошибки, игнорируются. (Это значит, что все функции, находящиеся в том же общем объекте, что и функция, вернувшая код ошибки, не вызываются.)

Параметр version содержит номер версии сервера proxy. Он передается сервером Caching Proxy.

PreExit

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

Определенная для этого этапа функция вызывается для каждого запроса после его считывания, но перед выполнением каких-либо других действий по обработке этого запроса. На этом этапе модуль может обращаться к клиентскому запросу до его обработки Caching Proxy.

Допустимые коды возврата для функции preExit:

- 0 (HTTP_NOACTION)
- 200 (HTTP_OK)
- Ошибки HTTP с кодами 4*xx* и 5*xx* (например, 404, HTTP_NOT_FOUND)

Другие коды возврата использовать нельзя.

Если функция возвращает результат HTTP_OK, то сервер proxy считает, что запрос обработан. Все последующие этапы обработки запроса в этом случае пропускаются и выполняются только этапы отправки ответа (Transmogrifier, Log и PostExit).

На этом этапе допустимы все библиотечные функции сервера.

Midnight

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

Определенная для этого этапа функция вызывается один раз в сутки, в полночь. Она не связана с контекстом запросов. Например, она может применяться для вызова дочернего процесса и анализа протоколов. (Обратите внимание, что высокая нагрузка на этом этапе может помешать нормальному ведению протоколов.)

Идентификация

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

Определенная для этого этапа функция вызывается для каждого запроса в соответствии со схемой идентификации запроса. Эта функция может применяться для настройки процедур проверки маркеров защиты, отправленных вместе с запросом.

NameTrans

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

Определенная для этого этапа функция вызывается для каждого запроса. Если вы хотите, чтобы функция модуля вызывалась только для запросов,

соответствующих определенному шаблону, то необходимо указать такой шаблон URL в директиве файла конфигурации. Этап преобразования имен NameTrans выполняется перед началом обработки запроса и обеспечивает механизм установления соответствия между URL и объектами файловой системы, например, именами файлов.

Проверка прав доступа

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

Определенная для этого этапа функция вызывается для каждого запроса. Если вы хотите, чтобы функция модуля вызывалась только для запросов, соответствующих определенному шаблону, то необходимо указать такой шаблон URL в директиве файла конфигурации. Этап проверки прав доступа (Authorization) выполняется перед началом обработки запроса и может применяться для проверки допустимости возврата найденного объекта клиенту. При использовании базовой идентификации необходимо сформировать обязательные заголовки WWW-Authenticate.

ObjectType

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

Определенная для этого этапа функция вызывается для каждого запроса. Если вы хотите, чтобы функция модуля вызывалась только для запросов, соответствующих определенному шаблону, то необходимо указать такой шаблон URL в директиве файла конфигурации. Этап определения типа объекта (ObjectType) выполняется перед началом обработки запроса. Он применяется для проверки наличия объекта и определения его типа.

PostAuthorization

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

Определенная для этого этапа функция вызывается после проверки прав доступа, но перед выполнением каких-либо других действий. Если функция возвращает результат HTTP_OK, то сервер proxy считает, что запрос обработан. Все последующие этапы обработки запроса в этом случае пропускаются и выполняются только этапы отправки ответа (Transmogrifier, Log и PostExit).

На этом этапе допустимы все библиотечные функции сервера.

Service

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

Определенная для этого этапа функция вызывается для каждого запроса. Если вы хотите, чтобы функция модуля вызывалась только для запросов, соответствующих определенному шаблону, то необходимо указать такой

шаблон URL в директиве файла конфигурации. Этап обслуживания (Service) выполняет запрос, если он не был выполнен на этапах PreExit или PostAuthorization.

На этом этапе допустимы все библиотечные функции сервера.

Информация о том, как настроить функцию для этапа Service на основе метода HTTP, а не на основе URL, приведена в описании директивы Enable в руководстве *WebSphere Application Server Caching Proxy Administration Guide*.

Transmogrifier

Функции, вызываемые на этом этапе обработки, могут применяться для фильтрации потока данных ответа в виде потока. На этом этапе последовательно вызываются четыре функции модуля, каждая из которых представляет собой элемент конвейера, по которому передаются данные. Для каждого этапа вызываются предоставленные вами функции *open*, *write*, *close* и *error* (именно в таком порядке). Каждая функция обрабатывает тот же поток данных, что и предыдущие функции.

Для этого этапа необходимо реализовать следующие четыре функции. (Имена ваших функций могут отличаться от приведенных.)

- **Open**

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

  Функция открытия (open) выполняет инициализацию (например, выделение буферов), необходимую для обработки данных из потока. Любой код возврата, отличный от HTTP_OK, приводит к аварийному прерыванию обработки (в этом случае функции write и close не вызываются). Ваша функция может вернуть пустой указатель, т.е. вы можете выделить пространство для структуры и снова получить указатель в параметре *correlator* последующих функций.

- **Write**

```
void  HTTPD_LINKAGE  writeFunction (
         unsigned char *handle,
         unsigned char *data,    /* данные ответа, переданные
                                    исходным сервером */
         unsigned long *length,  /* длина данных ответа */
         void *correlator,       /* указатель, возвращенный
                                    функцией 'open' */
         long *return_code
         )
```

  Функция записи (write) обрабатывает данные и может вызвать библиотечную функцию сервера HTTPD_write() с использованием новых или измененных данных. Модуль не должен пытаться освобождать полученный буфер и не должен ожидать, что сервер освободит полученный буфер.

  Даже если вы решили не изменять данные в области действия своей функции записи, то для передачи данных ответа клиенту вам все равно нужно будет вызвать функцию HTTPD_write() в области действия функции open, write или close. Аргумент *correlator* - это указатель на буфер данных, возвращенный в функции *open*.

- **Close**

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

Функция закрытия (close) выполняет все действия по очистке (такие как освобождение памяти и очистка буфера correlator), необходимые для завершения обработки данных в текущем потоке. Аргумент *correlator* - это указатель на буфер данных, возвращенный в функции *open*.

- **Error**

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

Функция обработки ошибок (error) позволяет выполнить необходимые действия по очистке или/или освобождению памяти перед отправкой страницы с сообщением об ошибке. На этом этапе вызываются ваши функции open, write и close, обрабатывающие страницу с сообщением об ошибке. Аргумент *correlator* - это указатель на буфер данных, возвращенный в функции *open*.

Примечания:

- При создании модуля для этапа Transmogrifier вы должны обязательно вызвать в области действия своих функций open, write и close функции HTTPD_open(), HTTPD_write() и HTTPD_close(). Функцию HTTPD_write() можно вызывать только после вызова HTTPD_open(). Назначение этих библиотечных функций заключается в передаче управления серверу, который должен будет вызвать следующую функцию в последовательности.
- Вызов функций HTTPD_* необходим для правильной работы этапа API Transmogrifier и для правильной работы сервера. Например, если не вызвать функции HTTPD_open() и HTTPD_close(), то клиенту не будут переданы заголовки.
- Помните, что при неправильной работе фильтрующих поток данных приложений возможно возникновение нежелательных эффектов. Например, могут перестать работать программы CGI, могут быть не показаны файлы GIF и могут неправильно работать другие двоичные потоки данных.
- Модулю необязательно буферизировать содержимое документов. Caching Proxy автоматически определяет длину содержимого.
- Если вы готовы передать серверу управление заголовками, то желательно вызвать функцию HTTPD_open(). Однако, если в дальнейшем вам нужно будет задать заголовок в программе API, то можно отложить вызов функции HTTPD_open() до выполнения функций write или close.

  Примечание: Перед вызовом функции HTTPD_open() необходимо задать заголовки с помощью функций HTTPD_set() и httpd_setvar().
- Поток данных не включает заголовки. Для работы с заголовками в модуле следует применять соответствующие функции set и extract. Функция *open* модуля не вызывается до тех пор, пока не будут прочитаны все заголовки.
- Вы можете использовать несколько модулей Transmogrifier, которые будут вызываться в том порядке, в котором они перечислены в файле конфигурации.

- Туннели SSL не передаются через модули Transmogrifier.

GCAdvisor

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

Определенная для этого этапа функция вызывается для каждого находящегося в кэше файла во время сбора мусора. Эта функция позволяет вам оказывать влияние на то, какие файлы должны сохраняться в кэше, а какие - удаляться. Дополнительная информация приведена в описании переменных GC_*.

Proxy Advisor

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

Определенная для этого этапа функция вызывается во время обслуживания каждого запроса proxy. Например, она может применяться для установки переменной USE_PROXY.

Log

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

Определенная для этого этапа функция вызывается для каждого запроса после его обработки и закрытия соединения с клиентом. Если вы хотите, чтобы функция модуля вызывалась только для запросов, соответствующих определенному шаблону, то необходимо указать такой шаблон URL в директиве файла конфигурации. Эта функция вызывается независимо от результата обработки запроса. Если вы не хотите, чтобы ваш модуль переопределял механизм ведения протоколов по умолчанию, то вместо кода возврата HTTP_OK передайте код HTTP_NOACTION.

Error

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

Определенная для этого этапа функция вызывается для всех запросов, при выполнении которых возникли ошибки. Если вы хотите, чтобы функция модуля вызывалась только для запросов с ошибками, соответствующих определенному шаблону, то необходимо указать такой шаблон URL в директиве файла конфигурации. Этот этап предоставляет возможность настройки алгоритма обработки ошибок.

PostExit

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

Определенная для этого этапа функция вызывается для каждого запроса, независимо от результата его обработки. Данный этап позволяет выполнить задачи очистки ресурсов, выделенных в модуле во время обработки запроса.

Server Termination

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

Определенная для этого этапа функция вызывается во время нормального завершения работы сервера. Она позволяет выполнить очистку ресурсов, выделенных на этапе ServerInit. На этом этапе не следует вызывать функции HTTP_* (результат может оказаться непредсказуемым). Если в файле конфигурации для этапа ServerTerm указано несколько директив с API Caching Proxy, то они все будут последовательно вызваны.

Примечание: Из-за существующего в настоящее время ограничения в коде Solaris модуль этапа ServerTerm не выполняется при завершении работы Caching Proxy на платформе Solaris с помощью команды **ibmproxy -stop**. Информация о запуске и завершении работы Caching Proxy приведена в руководстве *WebSphere Application Server Caching Proxy Administration Guide*.

Значения и коды возврата HTTP

Эти коды возврата соответствуют спецификации HTTP 1.1, RFC 2616, опубликованной консорциумом World Wide Web (www.w3.org/pub/WWW/Protocols/). Функции вашего модуля должны возвращать одно из этих значений.

Таблица 2. Коды возврата HTTP для функций API Caching Proxy

Значение	Код возврата
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

Таблица 2. Коды возврата HTTP для функций API Caching Proxy (продолжение)

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

Заранее определенные функции и макросы

Из своих собственных модулей вы можете вызывать библиотечные макросы и функции сервера. При этом необходимо использовать описанные ниже имена и форматы. В описаниях параметров буква `i` обозначает входной параметр, буква `o` - выходной, а буквы `i/o` указывают, что параметр используется как для ввода, так и для вывода информации.

Каждая из этих функций возвращает один их кодов возврата HTTPD, в зависимости от того, насколько успешно выполнен запрос. Эти коды описаны в разделе “Коды возврата заранее определенных функций и макросов” на стр. 23.

При вызове этих функций в качестве первого параметра необходимо указывать описатель, предоставленный вашему модулю. В противном случае функция вернет код ошибки HTTPD_PARAMETER_ERROR. Значение NULL недопустимо в качестве описателя.

HTTPD_authenticate()

Выполняет идентификацию ИД и/или пароля пользователя. Допускается только на этапах PreExit, Authentication, Authorization и PostAuthorization.

```
void  HTTPD_LINKAGE  HTTPD_authenticate (
    unsigned char *handle,             /* i; описатель */
         long *return_code         /* o; код возврата */
         )
```

HTTPD_cacheable_url()

Указывает, допустимо ли кэширование информации с указанным URL в соответствии со стандартами Caching Proxy.

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
    unsigned char *handle,             /* i; описатель */
        unsigned char *url,         /* i; проверяемый URL */
        unsigned char *req_method,  /* i; метод запроса для URL */
        long *retval                /* o; код возврата */
        )
```

Возвращенное значение HTTPD_SUCCESS указывает, что информация с данным URL может кэшироваться. Значение HTTPD_FAILURE указывает,что кэширование запрещено. Эта функция может также вернуть значение HTTPD_INTERNAL_ERROR.

HTTPD_close()

(Допускается только на этапе Transmogrifier.) Передает управление следующей процедуре *close* в стеке. Эту функцию следует вызывать на этапе Transmogrifier из функций open, write и close после выполнения всех требуемых операций. Данная функция уведомляет сервер Proxy, что ответ обработан и что этап Transmogrifier завершен.

```
void  HTTPD_LINKAGE  HTTPD_close (
    unsigned char *handle,             /* i; описатель */
         long *return_code         /* o; код возврата */
         )
```

HTTPD_exec()

Выполняет сценарий обработки запроса. Допускается на этапах PreExit, Service, PostAuthorization и Error.

```
void  HTTPD_LINKAGE  HTTPD_exec (
    unsigned char *handle,             /* i; описатель */
         unsigned char *name,           /* i; имя выполняемого сценария */
         unsigned long *name_length,   /* i; длина имени */
         long *return_code         /* o; код возврата */
         )
```

HTTPD_extract()

Извлекает значение переменной, связанной с запросом. Переменные, допустимые в параметре *name*, совпадают с используемыми в CGI. Дополнительная информация приведена в разделе “Переменные” на стр. 27. Обратите внимание, что данная функция допустима на всех этапах, однако некоторые переменные допустимы не на всех этапах.

```
void  HTTPD_LINKAGE  HTTPD_extract (
    unsigned char *handle,             /* i; описатель */
      unsigned char *name,         /* i; имя извлекаемой переменной */
         unsigned long *name_length,   /* i; длина имени */
      unsigned char *value,        /* o; буфер для размещения
                                       значения */
      unsigned long *value_length, /* i/o; размер буфера */
         long *return_code         /* o; код возврата */
      )
```

Если эта функция возвращает код HTTPD_BUFFER_TOO_SMALL, значит размер запрошенного буфера был недостаточен для извлеченного значения. В этом случае функция не использует буфер, но указывает в параметре value_length размер буфера, необходимый для успешного извлечения значения. Повторите операцию с буфером, размер которого равен или больше возвращенному значению value_length.

Примечание: Если переменная извлекается из заголовка HTTP, то функция HTTPD_extract() извлечет только первое совпадение, даже если запрос содержит несколько заголовков с одинаковыми именами. В этом случае вместо HTTPD_extract() можно использовать функцию httpd_getvar(), которая имеет также и другие преимущества. Дополнительная информация приведена в разделе 19.

HTTPD_file()

Отправляет файл в ответ на запрос. Допускается только на этапах PreExit, Service, Error, PostAuthorization и Transmogrifier.

```
void  HTTPD_LINKAGE  HTTPD_file (
    unsigned char *handle,             /* i; описатель */
         unsigned char *name,            /* i; имя отправляемого файла */
         unsigned long *name_length,   /* i; длина имени */
         long *return_code         /* o; код возврата */
         )
```

httpd_getvar()

Аналогична HTTPD_extract(), однако использовать эту функцию проще, поскольку она не требует указывать длину аргументов.

```
const  unsigned char *          /* o; значение переменной*/
HTTPD_LINKAGE
httpd_getvar(
    unsigned char *handle,             /* i; описатель */
       unsigned char *name,         /* i; имя переменной */
   unsigned long *n             /* i; индекс массива
                                       заголовка */
   )
```

Индекс массива начинается с 0. Для получения первого элемента массива укажите значение *n*, равное 0; для получения пятого элемента - значение *n*, равное 4.

Примечание: Не изменяйте и не уничтожайте возвращенное значение. Возвращенная строка заканчивается символом null.

HTTPD_log_access()

Записывает строку в протокол доступа сервера.

```
void  HTTPD_LINKAGE  HTTPD_log_access (
    unsigned char *handle,             /* i; описатель */
         unsigned char *value,           /* i; записываемые данные */
         unsigned long *value_length,    /* i; длина данных */
         long *return_code         /* o; код возврата */
         )
```

Обратите внимание, что для записи символа процента (%) в протокол доступа сервера escape-символы *не* нужны.

HTTPD_log_error()

Записывает строку в протокол ошибок сервера.

```
void  HTTPD_LINKAGE  HTTPD_log_error (
    unsigned char *handle,             /* i; описатель */
         unsigned char *value,           /* i; записываемые данные */
         unsigned long *value_length,    /* i; длина данных */
         long *return_code         /* o; код возврата */
         )
```

Обратите внимание, что для записи символа процента (%) в протокол ошибок сервера escape-символы *не* нужны.

HTTPD_log_event()

Записывает строку в протокол событий сервера.

```
void  HTTPD_LINKAGE  HTTPD_log_event (
    unsigned char *handle,             /* i; описатель */
         unsigned char *value,           /* i; записываемые данные */
         unsigned long *value_length,    /* i; длина данных */
         long *return_code         /* o; код возврата */
         )
```

Обратите внимание, что для записи символа процента (%) в протокол событий сервера escape-символы *не* нужны.

HTTPD_log_trace()

Записывает строку в протокол трассировки сервера.

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
    unsigned char *handle,             /* i; описатель */
         unsigned char *value,           /* i; записываемые данные */
         unsigned long *value_length,    /* i; длина данных */
         long *return_code         /* o; код возврата */
         )
```

Обратите внимание, что для записи символа процента (%) в протокол трассировки сервера escape-символы *не* нужны.

HTTPD_open()

(Допускается только на этапе Transmogrifier.) Передает управление следующей процедуре в стеке. Эту функцию следует вызывать на этапе Transmogrifier из функций open, write и close после установки всех необходимых заголовков и сообщения о готовности к записи.

```
void  HTTPD_LINKAGE  HTTPD_open (
    unsigned char *handle,             /* i; описатель */
         long *return_code         /* o; код возврата */
         )
```

HTTPD_proxy()

Создает запрос proxy. Допускается на этапах PreExit, Service и PostAuthorization.

Примечание: Это функция завершения. После ее вызова запрос считается выполненным.

```
void  HTTPD_LINKAGE  HTTPD_proxy (
    unsigned char *handle,             /* i; описатель */
         unsigned char *url_name,      /* i; URL для запроса
                                           proxy */
         unsigned long *name_length,   /* i; длина URL */
         void *request_body,           /* i; текст запроса */
         unsigned long *body_length,   /* i; длина текста */
         long *return_code         /* o; код возврата */
         )
```

HTTPD_read()

Считывает текст запроса клиента. Для заголовков применяется функция HTTPD_extract(). Допускается только на этапах PreExit, Authorization, PostAuthorization и Service и имеет смысл лишь в том случае, если выполнялся запрос PUT или POST. Вызывайте эту функцию в цикле до получения HTTPD_EOF. Если в запросе отсутствует текст, то функция возвращает ошибку.

```
void  HTTPD_LINKAGE  HTTPD_read (
    unsigned char *handle,             /* i; описатель */
         unsigned char *value,           /* i; буфер данных */
```

```
        unsigned long *value_length,    /* i/o; размер буфера
                                           (длина данных) */
        long *return_code         /* o; код возврата */
        )
```

HTTPD_restart()

Перезапускает сервер после обработки всех активных запросов. Допускается на всех этапах, кроме Server Initialization, Server Termination и Transmogrifier.

```
void  HTTPD_LINKAGE  HTTPD_restart (
        long *return_code         /* o; код возврата */
        )
```

HTTPD_set()

Устанавливает значение переменной, связанной с запросом. Переменные, допустимые в параметре *name*, совпадают с используемыми в CGI. Дополнительная информация приведена в разделе “Переменные” на стр. 27.

Обратите внимание, что с помощью этой функции можно также создавать переменные. Создаваемые переменные должны отвечать требованиям соглашений для префиксов HTTP_ и PROXY_, описанным в разделе “Переменные” на стр. 27. При создании переменной с префиксом HTTP_ эта переменная отправляется клиенту вместе с ответом без префикса HTTP_. Например, для задания заголовка Location можно воспользоваться функцией HTTPD_set() с переменной HTTP_LOCATION. Переменные, создаваемые с префиксом PROXY_, отправляются как заголовки запроса серверу информационного наполнения. Переменные, созданные с префиксом CGI_, передаются программам CGI.

Обратите внимание, что данная функция допустима на всех этапах, однако некоторые переменные допустимы не на всех этапах.

```
void  HTTPD_LINKAGE  HTTPD_set (
    unsigned char *handle,             /* i; описатель */
        unsigned char *name,           /* i; имя устанавливаемой
                                          переменной */
        unsigned long *name_length,    /* i; длина имени */
        unsigned char *value,          /* i; буфер со значением */
        unsigned long *value_length,   /* i; длина значения */
        long *return_code         /* o; код возврата */
        )
```

Примечание: С помощью функции httpd_setvar() можно задавать значения переменных, не указывая буфер и длину. Дополнительная информация приведена в разделе 21.

httpd_setvar()

Аналогична HTTPD_set(), однако использовать эту функцию проще, поскольку она не требует указывать длину аргументов.

```
long            /* o; код возврата */
HTTPD_LINKAGE   httpd_setvar (
    unsigned char *handle,             /* i; описатель */
       unsigned char *name,          /* i; имя переменной */
       unsigned char *value,         /* i; новое значение */
       unsigned long *addHdr         /* i; добавление или
                                       замена заголовка */
       )
```

параметр *addHdr* может иметь одно из следующих четырех значений:

- HTTPD_SETVAR_REPLACE — Заменить все вхождения переменной заголовка на новое значение.

- HTTPD_SETVAR_REPLACE_ADD — Если переменная заголовка существует, то заменить первое вхождение на новое значение. Если переменная не существует, то добавить новое значение в список заголовков.
- HTTPD_SETVAR_ADD — Добавить значение в список заголовков.
- HTTPD_SETVAR_REMOVE_ALL — Удалить все вхождения переменной заголовка.

Эти переменные определены в файле HTAPI.h.

httpd_variant_insert()

Вставляет вариант в кэш.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
    unsigned char *handle,            /* i; описатель */
         unsigned char *URI,       /* i; URI объекта */
    unsigned char *dimension,         /* i; размеренность варианта */
    unsigned char *variant,           /* i; значение варианта */
         unsigned char *filename,  /* i; файл объекта */
         long *return_code         /* o; код возврата */
         )
```

Примечания:

1. Аргумент dimension относится к заголовку, с помощью которого объект получается из URI. Например, в приведенном выше примере возможным значением dimension может быть User-Agent.
2. Аргумент variant относится к значению заголовка, заданного в аргументе dimension. Это значение берется из URI. Например, в приведенном примере возможно следующее значение аргумента variant:

   ```
   Mozilla 4.0 (compatible; BatBrowser 94.1.2; Bat OS)
   ```
3. Аргумент filename должен указывать на завершающуюся символом null строку с именем файла, в котором пользователь сохранил измененную информацию. Ответственность за удаление файла лежит на пользователе; это можно сделать после выхода из данной функции. Файл содержит только текст, без заголовков.
4. При кэшировании вариантов сервер обновляет заголовок content-length и добавляет заголовок с предупреждением 214. Теги strong entity удаляются.

httpd_variant_lookup()

Определяет, существуют ли в кэше заданный вариант.

```
void  HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,            /* i; описатель */
    unsigned char *URI,               /* URI объекта */
    unsigned char *dimension,         /* i; размеренность варианта */
    unsigned char *variant,           /* i; значение варианта */
             long *return_code);      /* o; код возврата */
```

HTTPD_write()

Записывает текст ответа. Допускается на этапах PreExit, Service, Error и Transmogrifier.

Если вы не задали тип информации перед первым вызовом этой функции, то сервер предполагает, что передается поток данных CGI.

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,            /* i; описатель */
    unsigned char *value,             /* i; отправляемые данные */
    unsigned char *value_length,      /* i; длина данных */
             long *return_code);      /* o; код возврата */
```

Примечание: Для настройки заголовков ответа применяется функция 21.

Примечание: После выхода из функции HTTPD_* вы можете освободить всю переданную ей память.

Коды возврата заранее определенных функций и макросов

В зависимости от того, насколько успешно выполнен запрос, сервер присвоит параметру кода возврата одно из следующих значений:

Таблица 3. Коды возврата

Значение	Код состояния	Описание
-1	HTTPD_UNSUPPORTED	Функция не поддерживается.
0	HTTPD_SUCCESS	Функция выполнена успешно, все поля вывода допустимы.
1	HTTPD_FAILURE	При выполнении функции возникла ошибка.
2	HTTPD_INTERNAL_ERROR	Произошла внутренняя ошибка; продолжать обработку запроса невозможно.
3	HTTPD_PARAMETER_ERROR	Функции передан один или несколько недопустимых параметров.
4	HTTPD_STATE_CHECK	Функция недопустима на данном этапе обработки.
5	HTTPD_READ_ONLY	(Возвращается только функциями HTTPD_set и httpd_setvar.) Переменная предназначена только для чтения и ее значение нельзя задавать из модуля.
6	HTTPD_BUFFER_TOO_SMALL	(Возвращается функциями HTTPD_set, httpd_setvar и HTTPD_read.) Размер предоставленного буфера слишком мал.
7	HTTPD_AUTHENTICATE_FAILED	(Возвращается только функцией HTTPD_authenticate.) Ошибка идентификации. Дополнительная информация приведена в переменных HTTP_RESPONSE и HTTP_REASON.
8	HTTPD_EOF	(Возвращается только функцией HTTPD_read.) Указывает на завершение текста запроса.
9	HTTPD_ABORT_REQUEST	Запрос отменен, поскольку клиент предоставил тег объекта, не соответствующий указанным в запросе условиям.
10	HTTPD_REQUEST_SERVICED	(Возвращается функцией HTTPD_proxy.) Вызванная функция завершила создание ответа на запрос.
11	HTTPD_RESPONSE_ALREADY_ COMPLETED	Функция не выполнена, поскольку ответ на этот запрос уже создан.
12	HTTPD_WRITE_ONLY	Переменная предназначена только для записи и ее значение нельзя считывать из модуля.

Директивы конфигурации Caching Proxy для этапов обработки API

Для каждого этапа обработки запроса существует директива конфигурации, с помощью которой вы можете указать, какая функция модуля должна вызываться и выполняться на этом этапе. Вы можете добавить эти директивы в файл конфигурации сервера (ibmproxy.conf) вручную или с помощью формы API обработки запросов в интерфейсе администрирования и настройки Caching Proxy.

Примечания по использованию API

- Все директивы API, за исключением директив Service и NameTrans, могут указываться в файле конфигурации в произвольном порядке. Однако следует помнить, что при наличии в файле нескольких записей для одной директивы API порядок следования записей имеет существенное значение (см. далее).
- Не обязательно включать запись для каждого этапа API. Если у вас нет модуля для какого-либо этапа, то просто не указывайте соответствующую директиву. В этом случае будет применяться стандартная процедура обработки.
- Директивы Service и NameTrans работают аналогично любым другим директивы преобразования имен (например, Pass) и результат их работы зависит от положения этих директив по отношению к другим директивам преобразования имен в файле конфигурации. Например, правило для шаблона /cgi-bin/foo.so должно располагаться перед правилом для шаблона /cgi-bin/*.

  Это значит, что сервер обрабатывает директивы Service, NameTrans, Exec, Fail, Map, Pass, Proxy, ProxyWAS и Redirect в том порядке, в котором они встречаются в файле конфигурации. Когда сервер успешно устанавливает соответствие между URL и файлом, он больше не считывает и не обрабатывает последующие директивы. (Исключением является директива Map. Подробная информация о правилах преобразования записей сервера proxy приведена в книге *WebSphere Application Server Caching Proxy Administration Guide*.)
- Для этапа обработки может быть определено несколько директив конфигурации. Например, вы можете указать в файле две директивы NameTrans, каждая из которых указывает на свою функцию модуля. При выполнении сервером этапа преобразования имен ваши функции преобразования будут вызываться в том порядке, в котором они заданы в файле конфигурации.

  Примечание: Если функция модуля, предоставленная Caching Proxy, использует ту же директиву API, что и созданный вами модуль, то поместите директиву своего модуля после директивы системного модуля.
- Некоторые функции модулей можно вызывать не для всех запросов:
  - В некоторые директивы включены маски URL. Указание маски URL позволяет вызывать приложение модуля только для тех запросов, URL которых отвечает заданному шаблону. Подробная информация о том, на каких этапах могут применяться маски URL, приведена в разделах “Синтаксис и директивы API” на стр. 25 и “Переменные директив API” на стр. 25.
  - Задав с помощью директивы Authentication тип идентификации, вы можете указать, что модуль должен вызываться только для определенных способов идентификации. В настоящее время протоколом HTTP поддерживается только базовая идентификация. Дополнительная информация приведена в разделе “Переменные директив API” на стр. 25.
- Если серверу не удается загрузить какую-либо функцию модуля или если директива ServerInit не вернула код возврата OK, то другие функции откомпилированного модуля Caching Proxy не вызываются. Результаты обработки

модуля, уже выполненные к этому моменту, игнорируются. На другие модули Caching Proxy, включенные в эти директивы, а также на их функции, данная ситуация не влияет.

Синтаксис и директивы API

Следующие директивы файла конфигурации должны быть указаны в файле ibmproxy.conf на одной строке без лишних пробелов. Должны присутствовать лишь те пробелы, которые явно указаны в данной таблице. Несмотря на то, что в некоторых примерах для упрощения чтения директивы перенесены на новую строку, на самом деле в соответствующих позициях директив не должно быть пробелов.

Таблица 4. Директивы API модуля Caching Proxy

ServerInit		*/путь/файл:функция init_string*
PreExit		*/путь/файл:функция*
Authentication	*type*	*/путь/файл:функция*
NameTrans	*/URL*	*/путь/файл:функция*
Authorization	*/URL*	*/путь/файл:функция*
ObjectType	*/URL*	*/путь/файл:функция*
PostAuth		*/путь/файл:функция*
Service	*/URL*	*/путь/файл:функция*
Полночь		*/путь/файл:функция*
Transmogrifier		*/путь/файл:функция_открытия: функция_записи: функция_закрытия:функция_ошибки*
Log	*/URL*	*/путь/файл:функция*
Error	*/URL*	*/путь/файл:функция*
PostExit		*/путь/файл:функция*
ServerTerm		*/путь/файл:функция*
ProxyAdvisor		*/путь/файл:функция*
GCAdvisor		*/путь/файл:функция*

Переменные директив API

Переменные в этих директивах имеют следующее значение:

тип Применяется только в директиве Authentication, чтобы указать, нужно или нет вызывать вашу функцию модуля. Допустимые значения:

- Basic — Функция модуля вызывается только для запросов с базовой идентификацией.
- * — Функция модуля вызывается для всех запросов. В настоящее время протоколом HTTP поддерживается только базовая идентификация. Для запросов с другими типами идентификации можно возвращать код ошибки, указывающий, что данный тип идентификации не поддерживается.

URL Указывает запросы, для которых вызывается функция модуля. Для запросов, URL которых соответствуют указанному шаблону, будет применяться данная функция. В директивах применяются виртуальные спецификации URL (без указания протокола), однако в них присутствует ведущий символ косой черты (/). Например, значение /www.ics.raleigh.ibm.com допустимо, а значение http://www.ics.raleigh.ibm.com - нет. Можно указывать как отдельные URL, так и шаблоны.

- конкретный URL — Функция модуля вызывается только для указанного URL.
- Шаблон URL — Функция модуля вызывается для всех URL, соответствующих шаблону. Шаблоны могут содержать символ подстановки * и могут указываться в формате /*URL**, /* или *

  Примечание: Для выполнения преобразования шаблон URL *обязательно* должен быть указан в директиве Service.

путь/файл
Полное имя файла откомпилированной программы.

функция
Имя, присвоенное функции модуля в программе.

Если необходимо обеспечить доступ к информации о пути, то в директиве Service после имени функции должна быть указана звездочка (*).

строка-инициализации
Это необязательная часть директивы ServerInit, которая может содержать любой текст, передаваемый функции модуля. Для получения текста из переменной INIT_STRING применяется функция httpd_getvar().

Дополнительная информация о директивах, включая описание синтаксиса, приведена в руководстве *WebSphere Application Server Caching Proxy Administration Guide*.

Совместимость с другими API

API Caching Proxy обеспечивают обратную совместимость с ICAPI и GWAPI вплоть до версии 4.6.1.

Портирование программ CGI

При портировании написанных на C приложений CGI в API Caching Proxy следует руководствоваться следующими правилами:

- Удалите точку входа main() или переименуйте ее, чтобы можно было создать DLL.
- Удалите глобальные переменные или защитите их семафором взаимного исключения.
- Измените в программе следующие вызовы:
  - Замените вызовы заголовков printf() на HTTPD_set() или httpd_setvar().
  - Замените вызовы данных printf() на HTTPD_write().
  - Замените вызовы getenv() на HTTPD_extract() или httpd_getvar(). Обратите внимание, что эти вызовы возвращают невыделенную память, поэтому вы должны освободить память результатов.
- Помните, что сервер работает в многонитевой среде и ваши функции модулей должны обеспечивать надежную работу в такой среде. Производительность реентерабельных функций не снижается.
- При отправке данных клиенту с помощью функции HTTPD_write() обязательно задавайте заголовок Content-Type.
- Внимательно проверьте отсутствие утечек памяти в коде.
- Обдумайте алгоритмы обработки ошибок. Если вы самостоятельно формируете сообщения об ошибках и возвращаете их в виде кода HTML, то при вызове функций необходимо возвращать код HTTPD_OK.

Справочная информация об API Caching Proxy

Переменные

При написании программ для API можно использовать переменные компонента Caching Proxy, предоставляющие информацию об удаленной системе клиента и сервера.

Примечания:

- В именах пользовательских переменных нельзя использовать приставку SERVER_. API компонента Caching Proxy разрешает использовать переменные, начинающиеся с приставки SERVER_, только для сервера, и поэтому эти переменные доступны только для чтения. Кроме того, приставки HTTP_ и PROXY_ используются в заголовках HTTP.
- Все заголовки запросов, отправляемые клиентом (например, Set-Cookie), имеют приставку HTTP_, а их значения можно извлечь. Для доступа к переменным заголовков запросов добавьте к их именам приставку HTTP_. Для создания новых переменных используется предопределенная функция httpd_setvar(). Дополнительная информация о заголовках приведена в разделе “Коды возврата заранее определенных функций и макросов” на стр. 23.
- Приставки переменных HTTP_ и PROXY_ применяются для обозначения переменных, связанных с заголовками запроса или ответа. Приставка HTTP_ используется для переменных, передаваемых между клиентом и компонентом Caching Proxy. Приставка PROXY_ используется для переменных, передаваемых между компонентом Caching Proxy и исходным сервером (или следующим сервером в цепочке proxy). Эти переменные действительны только во время обработки запросов.
  - Извлечение переменной HTTP_* позволяет получить значение заголовка в запросе клиента для сервера Proxy.
  - Переменная HTTP_* задает заголовок ответа, отправляемого сервером Proxy клиенту.
  - Извлечение переменной PROXY_* возвращает значение заголовка, который сервер содержимого отправил серверу Proxy.
  - Переменная PROXY_* задает значение заголовка запроса, который сервер Proxy отправляет серверу содержимого (или следующему серверу в цепочке серверов Proxy).

  рис. 2 описывает использование этих приставок при обработке запроса клиента компонентом Caching Proxy.

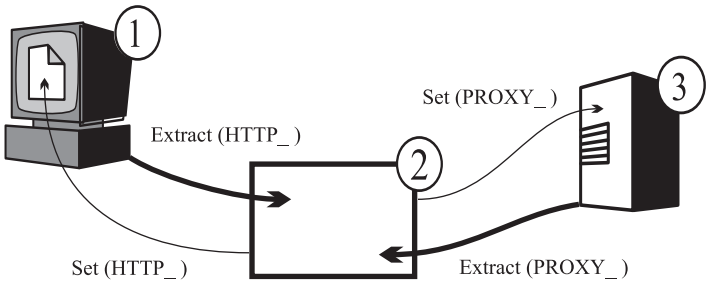

*Рисунок 2. Приставки переменных HTTP_ и PROXY_*. Условные обозначения:
1—Клиент 2—Caching Proxy 3—Исходный сервер

- Некоторые переменные доступны только для чтения. Переменные, доступные только для чтения, обозначают значения, которые можно извлекать из запроса или

ответа и использовать в предопределенной функции httpd_getvar(). При попытке изменить переменные, доступные только для чтения, при помощи функции httpd_setvar(), выдается код возврата HTTPD_READ_ONLY.

- Переменные, не обозначенные как доступные только для чтения, могут быть прочитаны и заданы в предопределенных функциях httpd_getvar() или httpd_setvar(). Эти переменные обозначают значения, которые можно извлекать из запроса или ответа; или значения, которые можно задавать или создавать при обработке запроса или ответа.

Определения переменных

Примечание: Переменные заголовков без приставок HTTP_ или PROXY_ являются неоднозначными. Во избежание неоднозначности рекомендуется всегда использовать приставку HTTP_ или PROXY_ в названиях переменных для заголовков.

ACCEPT_RANGES

Содержит значение заголовка ответа Accept-Ranges, которое указывает, может ли сервер содержимого отвечать на различные запросы. Для извлечения значения заголовка, который сервер содержимого отправляет серверу Proxy, используется переменная PROXY_ACCEPT_RANGES. Для установки значения заголовка, отправляемого сервером Proxy клиенту, используется переменная HTTP_ACCEPT_RANGES.

Примечание: Переменная ACCEPT_RANGES является неоднозначной. Во избежание неоднозначности следует использовать названия переменных HTTP_ACCEPT_RANGES и PROXY_ACCEPT_RANGES.

ALL_VARIABLES

Только для чтения. Содержит все переменные CGI. Например:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING

Только для чтения. Если сервер поддерживает идентификацию клиента, эта строка содержит расшифрованные идентификационные данные, применяемые для идентификации клиента.

AUTH_TYPE

Только для чтения. Если сервер поддерживает идентификацию клиента и сценарий является защищенным, то эта переменная содержит метод, применяемый для идентификации клиента. Например, `Basic`.

CACHE_HIT

Только для чтения. Определяет, найден ли в кэше запрос сервера Proxy. Возвращаемые значения:

- 0 - Запрос не найден в кэше.
- 1 - Запрос найден в кэше.

CACHE_MISS

Только для записи. Применяется для принудительной отправки промаха в кэш. Допустимые значения:

- 0 - Без принудительной отправки промаха в кэш.
- 1 - Принудительная отправка промаха в кэш.

CACHE_TASK
Только для чтения. Определяет, используется ли кэш. Возвращаемые значения:

- 0 - Запрос не обращался к кэшу.
- 1 - Запрос был выполнен из кэша.
- 2 - Запрашиваемый объект находился в кэше, но был недействителен.
- 3 - Запрашиваемый объект не был найден в кэше и, возможно, был добавлен в кэш.

Эта переменная может применяться на шагах PostAuthorization, PostExit, ProxyAdvisor или Log.

CACHE_UPDATE
Только для чтения. Определяет, обновил ли запрос сервера Proxy данные в кэше. Возвращаемые значения:

- 0 - Кэш не был обновлен.
- 1 - Кэш был обновлен.

CLIENT_ADDR или CLIENTADDR
То же, что и REMOTE_ADDR.

CLIENTMETHOD
То же, что и REQUEST_METHOD.

CLIENT_NAME или CLIENTNAME
То же, что и REMOTE_HOST.

CLIENT_PROTOCOL или CLIENTPROTOCOL
Содержит название и версию протокола, применяемого клиентом для отправки запроса. Например, `HTTP/1.1`.

CLIENT_RESPONSE_HEADERS
Только для чтения. Возвращает буфер, содержащий заголовки, которые сервер отправляет клиенту.

CONNECTIONS
Только для чтения. Содержит число установленных соединений или число активных запросов. Например, `15`.

CONTENT_CHARSET
Набор символов ответа для text/*, например, `US ASCII`. Извлечение этой переменной относится к заголовку content-charset из запроса клиента. Установка этой переменной влияет на значение заголовка content-charset в запросе для сервера содержимого.

CONTENT_ENCODING
Указывает кодировку документа, например, `x-gzip`. Извлечение этой переменной относится к заголовку content-encoding из запроса клиента. Установка этой переменной влияет на значение заголовка content-charset в запросе для сервера содержимого.

CONTENT_LENGTH
Извлечение этой переменной относится к заголовку из запроса клиента. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого.

Примечание: Переменная CONTENT_LENGTH является неоднозначной. Во избежание неоднозначности следует использовать названия переменных HTTP_CONTENT_LENGTH и PROXY_CONTENT_LENGTH.

CONTENT_TYPE

Извлечение этой переменной относится к заголовку из запроса клиента. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого.

Примечание: Переменная CONTENT_TYPE является неоднозначной. Во избежание неоднозначности следует использовать названия переменных HTTP_CONTENT_TYPE и PROXY_CONTENT_TYPE.

CONTENT_TYPE_PARAMETERS

Содержит остальные атрибуты MIME, за исключением набора символов. Извлечение этой переменной относится к заголовку из запроса клиента. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого.

DOCUMENT_URL

Содержит URL-адрес документа. Например:

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

DOCUMENT_URI

То же, что и DOCUMENT_URL.

DOCUMENT_ROOT

Только для чтения. Содержит путь к корневому каталогу документа в соответствии с правилами передачи.

ERRORINFO

Указывает код ошибки для определения страницы, содержащей ошибку. Например, `blocked`.

EXPIRES

Определяет срок действия документов, хранящихся в кэше сервера Proxy. Извлечение этой переменной относится к заголовку из запроса клиента. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого. Например:

```
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. Значения в диапазоне от 0,0 до 1,0 упорядочиваются по рангу, т.е. файл со значением GC_EVALUATION равным 0,1 имеет большую вероятность, что он будет удален, чем файл со значением GC_EVALUATION равным 0,9.

GC_EXPIRES

Только для чтения. Определяет, сколько секунд осталось до истечения срока действия файла в кэше. Для извлечения этой переменной используется модуль GC Advisor.

GC_FILENAME
Только для чтения. Обозначает файл, предназначенный для сбора мусора. Для извлечения этой переменной используется модуль GC Advisor.

GC_FILESIZE
Только для чтения. Обозначает размер файла, предназначенного для сбора мусора. Для извлечения этой переменной используется модуль GC Advisor.

GC_LAST_ACCESS
Только для чтения. Обозначает время последнего обращения к файлу. Для извлечения этой переменной используется модуль GC Advisor.

GC_LAST_CHECKED
Только для чтения. Обозначает время последней проверки файлов. Для извлечения этой переменной используется модуль GC Advisor.

GC_LOAD_DELAY
Только для чтения. Обозначает продолжительность процедуры извлечения файла. Для извлечения этой переменной используется модуль GC Advisor.

HTTP_COOKIE
Эта переменная содержит значение заголовка Set-Cookie, установленное клиентом. Также применяется для создания новой записи cookie в потоке ответа (между Proxy и клиентом). Установка этой переменной создает новый заголовок Set-Cookie в потоке запроса документа, независимо от наличия дубликата заголовка.

HTTP_HEADERS
Только для чтения. Применяется для извлечения всех заголовков запросов клиентов.

HTTP_REASON
Установка этой переменной влияет на значение строки reason в ответе HTTP. Установка этой переменной также влияет на значение строки reason в ответе, который сервер Proxy отправляет клиенту. Извлечение этой переменной возвращает строку reason в ответе, который сервер содержимого отправляет серверу Proxy.

HTTP_RESPONSE
Установка этой переменной влияет на значение кода в ответе HTTP. Установка этой переменной также влияет на код состояния в ответе, который сервер Proxy отправляет клиенту. Извлечение этой переменной возвращает код состояния в ответе, который сервер содержимого отправляет серверу Proxy.

HTTP_STATUS
Содержит код ответа HTTP и строку reason. Например, `200 OK`.

HTTP_USER_AGENT
Содержит значение заголовка запроса User-Agent, который представляет собой имя Web-обозревателя клиента, например, `Netscape Navigator / V2.02`. Установка этой переменной влияет на значение заголовка в ответе, который сервер Proxy отправляет клиенту. Извлечение этой переменной относится к заголовку из запроса клиента.

INIT_STRING
Только для чтения. Значение этой строки определяется директивой ServerInit. Эта переменная доступна для чтения только во время инициализации сервера.

LAST_MODIFIED
Извлечение этой переменной относится к заголовку из запроса клиента. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого. Например:

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

LOCAL_VARIABLES
Только для чтения. Все пользовательские переменные.

MAXACTIVETHREADS
Только для чтения. Максимальное число активных нитей.

NOTMODIFIED_TO_OK
Принудительная отправка полного ответа клиенту. Действительно на шагах PreExit и ProxyAdvisor.

ORIGINAL_HOST
Только для чтения. Возвращает имя хоста или IP-адрес, указанный в запросе.

ORIGINAL_URL
Только для чтения. Возвращает исходный URL, отправленный в запросе клиента.

OVERRIDE_HTTP_NOTRANSFORM
Разрешает изменение данных, если указан заголовок Cache-Control: no-transform. Установка этой переменной влияет на значение заголовка ответа клиенту.

OVERRIDE_PROXY_NOTRANSFORM
Разрешает изменение данных, если указан заголовок Cache-Control: no-transform. Установка этой переменной влияет на значение запроса для сервера содержимого.

PASSWORD
Для базовой идентификации содержит расшифрованный пароль. Например, `password`.

PATH Содержит полностью преобразованный путь.

PATH_INFO
Содержит дополнительную информацию о пути, отправленную Web-обозревателем. Например, `/foo`.

PATH_TRANSLATED
Содержит расшифрованную или преобразованную версию пути, который указан в переменной PATH_INFO. Например:

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

```
/wwwhome/foo
```

PPATH
Содержит частично преобразованный путь. Используется на этапе преобразования имен.

PROXIED_CONTENT_LENGTH
Только для чтения. Возвращает длину данных ответа, переданного через сервер Proxy.

PROXY_ACCESS
Определяет, является ли данный запрос proxy-запросом. Например, `NO`.

PROXY_CONTENT_TYPE
Содержит заголовок Content-Type запроса сервера Proxy, отправленного с помощью функции HTTPD_proxy(). Если информация отправляется с

использованием метода POST, эта переменная содержит тип данных. В файле конфигурации сервера Proxy можно создать собственный тип содержимого и отобразить его для программы просмотра. Извлечение этой переменной относится к значению заголовка из ответа сервера содержимого. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого. Например:

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

PROXY_CONTENT_LENGTH

Заголовок Content-Length запроса сервера Proxy, отправленного с помощью функции HTTPD_proxy(). Если информация отправляется с использованием метода POST, эта переменная содержит число символов данных. Обычно серверы не отправляют флаг конца файла при передаче информации, используя стандартный ввод. При необходимо значение CONTENT_LENGTH может применяться для определения конца входной строки. Извлечение этой переменной относится к значению заголовка из ответа сервера содержимого. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого. Например:

```
7034
```

PROXY_COOKIE

Эта переменная содержит значение заголовка Set-Cookie, установленное исходным сервером. Также применяется для создания новой записи cookie в потоке запроса. Установка этой переменной создает новый заголовок Set-Cookie в потоке запроса документа, независимо от наличия дубликата заголовка.

PROXY_HEADERS

Только для чтения. Применяется для извлечения заголовков Proxy.

PROXY_METHOD

Метод для запроса, отправленного при помощи функции HTTPD_proxy(). Извлечение этой переменной относится к значению заголовка из ответа сервера содержимого. Установка этой переменной влияет на значение заголовка в запросе для сервера содержимого.

QUERY_STRING

Если информация отправляется с использованием метода GET, эта переменная содержит информацию после вопросительного знака (?) в запросе. Эта информация должна быть декодирована программой CGI. Например:

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

RCA_OWNER

Только для чтения. Возвращает числовое значение узла, который владеет запрашиваемым объектом. Эта переменная может применяться на шагах PostExit, ProxyAdvisor или Log. Ее значение является значащим, только если сервер является частью кэш-массива, использующего метод удаленного доступа к кэшу (RCA).

RCA_TIMEOUTS

Только для чтения. Возвращает числовое значение, содержащее общее число тайм-аутов в запросах RCA. Эта переменная может применяться на любом шаге.

REDIRECT_*

Только для чтения. Содержит строку redirection для кода ошибки, соответствующего имени переменной (например, `REDIRECT_URL`). Список

допустимых переменных REDIRECT_ приведен в документации по веб-серверу Apache по адресу http://httpd.apache.org/docs-2.0/custom-error.html.

REFERRER_URL
Только для чтения. Содержит последний URL-адрес обозревателя. Позволяет клиенту указать URL-адрес ресурса, из которого было получено значение Request-URL. Например:

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

REMOTE_ADDR
Содержит IP-адрес Web-обозревателя. Например, `45.23.06.8`.

REMOTE_HOST
Содержит имя хоста Web-обозревателя. Например, `www.raleigh.ibm.com`.

REMOTE_USER
Если сервер поддерживает идентификацию клиента и сценарий является защищенным, то эта переменная содержит имя пользователя, указанное для идентификации. Например, `joeuser`.

REQHDR
Только для чтения. Содержит список заголовков, отправленных клиентом.

REQUEST_CONTENT_TYPE
Только для чтения. Возвращает тип содержимого тела запроса. Например:

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

REQUEST_CONTENT_LENGTH
Только для чтения. Если информация отправляется с использованием метода POST, эта переменная содержит число символов данных. Обычно серверы не отправляют флаг конца файла при передаче информации, используя стандартный ввод. При необходимо значение CONTENT_LENGTH может применяться для определения конца входной строки. Например, `7034`.

REQUEST_METHOD
Только для чтения. Содержит метод (соответствующий значению атрибута METHOD в формате HTML), применяемый для отправки запроса. Например, `GET или POST`.

REQUEST_PORT
Только для чтения. Возвращает номер порта, указанный в URL-адресе или порт, принятый по умолчанию для данного протокола.

RESPONSE_CONTENT_TYPE
Только для чтения. Если информация отправляется с использованием метода POST, эта переменная содержит тип данных. В файле конфигурации сервера Proxy можно создать собственный тип содержимого и отобразить его для программы просмотра. Например, `text/html`.

RESPONSE_CONTENT_LENGTH
Только для чтения. Если информация отправляется с использованием метода POST, эта переменная содержит число символов данных. Обычно серверы не отправляют флаг конца файла при передаче информации, используя стандартный ввод. При необходимо значение CONTENT_LENGTH может применяться для определения конца входной строки. Например, `7034`.

RULE_FILE_PATH
Только для чтения. Содержит полный путь и имя файла конфигурации.

SSL_SESSIONID
Только для чтения. Возвращает ИД сеанса SSL, если текущий запрос получен по SSL-соединению. В противном случае возвращает значение NULL.

SCRIPT_NAME
Содержит URL-адрес запроса.

SERVER_ADDR
Только для чтения. Содержит локальный IP-адрес сервера Proxy.

SERVER_NAME
Только для чтения. Содержит имя хоста сервера Proxy или IP-адрес сервера содержимого для данного запроса. Например, `www.ibm.com`.

SERVER_PORT
Только для чтения. Содержит номер порта сервера Proxy, на который отправлен запрос клиента. Например, `80`.

SERVER_PROTOCOL
Только для чтения. Содержит название и версию протокола, применяемого для отправки запроса. Например, `HTTP/1.1`.

SERVER_ROOT
Только для чтения. Содержит каталог, в котором установлен сервер Proxy.

SERVER_SOFTWARE
Только для чтения. Содержит имя и версию сервера Proxy.

STATUS
Содержит код ответа HTTP и строку reason. Например, 200 OK.

TRACE
Определяет объем информации трассировки. Допустимые значения:

- OFF - Без трассировки.
- V - Подробный режим.
- VV - Очень подробный режим.
- MTV - Слишком подробный режим.

URI Чтение/запись. То же, что и DOCUMENT_URL.

URI_PATH
Только для чтения. Возвращает часть пути только для URL-адреса.

URL Чтение/запись. То же, что и DOCUMENT_URL.

URL_MD4
Только для чтения. Возвращает имя возможного файла кэша для текущего запроса.

USE_PROXY
Обозначает имя сервера Proxy в цепочке для текущего запроса. Содержит URL-адрес. Например, `http://myproxy:8080`.

USERID
То же, что и REMOTE_USER.

USERNAME
То же, что и REMOTE_USER.

Идентификация и проверка прав доступа

Вначале определимся с применяемой терминологией:

Идентификация
Проверка связанных с запросом маркеров системы безопасности, позволяющая гарантировать правильность сведений о личности запрашивающего.

Проверка прав доступа
Процесс проверки наличия у запрашивающего прав доступа к ресурсу. В этом процессе применяются маркеры системы безопасности.

рис. 3 содержит описание процессов идентификации и проверки прав доступа сервером proxy.

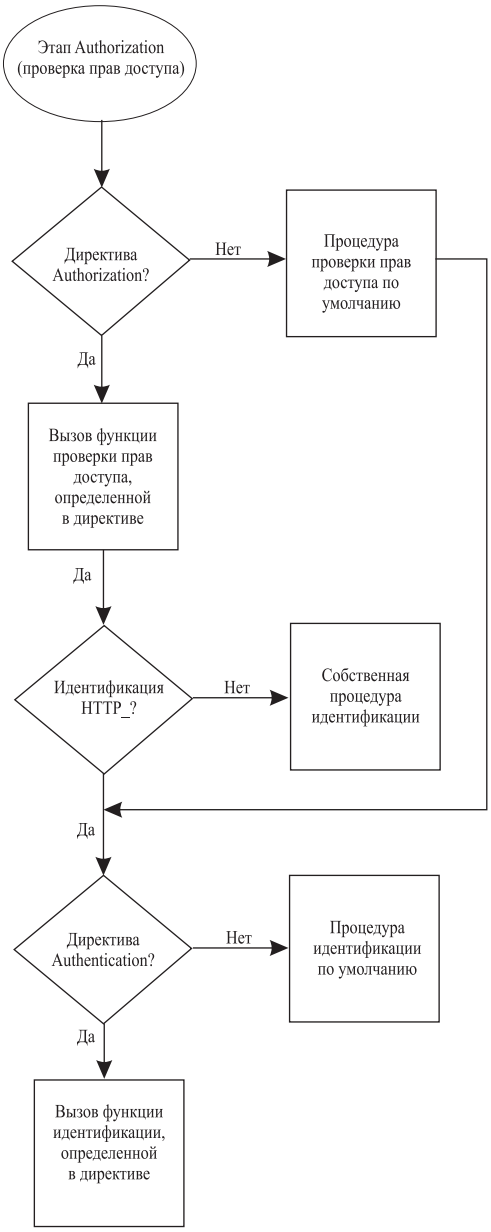

Рисунок 3. Процессы идентификации и проверки прав доступа сервером Proxy

Как показано на рис. 3, процессы идентификации и проверки прав доступа начинаются с инициализации процесса проверки прав доступа.

В Caching Proxy идентификация является частью процесса проверки прав доступа и выполняется только в том случае, когда необходимо проверить наличие прав доступа.

Процесс идентификации и проверки прав доступа

При обработке запроса, требующего проверки прав доступа, сервер proxy выполняет следующие действия.

1. Сначала сервер proxy проверяет наличие директив проверки прав доступа в своем файле конфигурации.
   - Если в файле конфигурации есть директива проверки прав доступа, то сервер вызывает определенную в этой директиве функцию и начинает идентификацию с шага 2.
   - Если директив проверки прав доступа нет, то сервер выполняет проверку прав доступа по умолчанию и переходит непосредственно к процедурам идентификации на этапе 3.
2. Сервер proxy начинает процесс идентификации путем проверки заголовка HTTP_authenticate в запросе клиента.
   - Если заголовок существует, то сервер продолжает процесс идентификации (см. этап 3).
   - Если заголовок отсутствует, то идентификация должна выполняться другим способом.
3. Сервер proxy проверяет наличие директивы идентификации в файле конфигурации proxy.
   - Если в файле конфигурации есть директива идентификации, то сервер вызывает определенную в ней функцию идентификации.
   - Если такой директивы нет, то сервер выполняет идентификацию по умолчанию.

Если в вашем модуле Caching Proxy предусмотрен собственный процесс проверки прав доступа, то он *переопределяет* применяемые сервером по умолчанию процессы идентификации и проверки прав доступа. Таким образом, если в файле конфигурации присутствуют директивы проверки прав доступа, то связанные с этими директивами функции модуля должны также обрабатывать все необходимые процедуры идентификации. Вы можете использовать заранее определенную функцию HTTPD_authenticate().

Существует три способа обеспечения идентификации в модуле проверки прав доступа:

- Написание собственных отдельных модулей идентификации и проверки прав доступа. Укажите эти функции в директивах Authorization и Authentication в файле конфигурации. Обязательно включите вызов функции HTTPD_authenticate() в функцию модуля проверки прав доступа.

  При выполнении этапа проверки прав доступа (Authorization) вызывается функция модули проверки прав доступа, которая в свою очередь вызывает функцию модуля идентификации.
- Написание собственной функции модуля проверки прав доступа, вызывающей функцию идентификации сервера по умолчанию. В файле конфигурации proxy укажите свою функцию в директиве Authorization. В этом случае директива Authentication не нужна. Обязательно включите вызов функции HTTPD_authenticate() в функцию модуля проверки прав доступа.

  При выполнении этапа проверки прав доступа (Authorization) вызывается функция модуля проверки прав доступа, которая в свою очередь вызывает функцию идентификации по умолчанию.
- Написание собственной функции модуля проверки прав доступа, в которую включены все необходимые действия по идентификации. В этом случае в модуль проверки прав доступа не нужно включать вызов функции HTTPD_authenticate().

Укажите свой модуль проверки прав доступа в директиве Authorization в файле конфигурации proxy. В этом случае директива Authentication не нужна.

При выполнении этапа проверки прав доступа вызывается функция модуля проверки прав доступа с включенными в ее состав средствами идентификации.

Если в модуле Caching Proxy не предусмотрен собственный процесс проверки прав доступа, то с помощью следующих методов вы все равно можете включить собственный процесс идентификации:

- Написание собственной функции модуля идентификации. Укажите эту функцию в директиве Authentication в файле конфигурации. В этом случае директива Authorization не нужна.

При выполнении этапа проверки прав доступа (Authorization) вызывается функция проверки прав доступа по умолчанию, которая в свою очередь вызывает вашу функцию модуля идентификации.

Обратите внимание на следующие особенности:

- Если в файле конфигурации нет директив Authorization или если указанные в них функции отказывают в обработке запроса путем возврата значения HTTP_NOACTION, то выполняется процедура проверки прав доступа сервера по умолчанию.
- Если в файле конфигурации есть директивы Authorization и их функции модуля включают вызов HTTPD_authenticate(), то сервер вызывает все функции идентификации, указанные в директивах Authentication. Если вы не определили директивы Authentication или указанные в них функции отказывают в обработке запроса путем возврата значения HTTP_NOACTION, то выполняется процедура идентификации сервера по умолчанию.
- Если в файле конфигурации сервера есть директивы Authorization, но их функции не включают вызов HTTPD_authenticate(), то сервер не вызывает функции идентификации. Вы должны реализовать собственные процедуры идентификации, применяемые в ходе проверки прав доступа, или вызвать другие модули идентификации.
- Если функция идентификации возвращает код `401` или `407`, то Caching Proxy автоматически запрашивает у браузера ИД пользователя и пароль. Однако для правильного выполнения этих действий вы все равно должны настроить защиту Caching Proxy.

Кэширование вариантов

Кэширование вариантов применяется для кэширования данных, представляющих собой видоизмененный вариант исходного документа (URI). Caching Proxy обрабатывает варианты, сформированные API. *Варианты* - это различные версии базового документа.

Обычно, когда исходные серверы отправляют варианты, они не указывают, что отправленные документы являются именно вариантами. Caching Proxy поддерживает только варианты, созданные модулями (например, при преобразовании кодовых страниц). Если модуль создает вариант документа на основе условия, не включенного в заголовок HTTP, то он должен с помощью функций этапа PreExit или PostAuthorization создать псевдозаголовок, который позволил бы Caching Proxy правильно идентифицировать существующие варианты.

Например, можно использовать программу API Transmogrifier для изменения запрошенных пользователями данных на основании отправленного браузером заголовка User-Agent. В функции *close* сохраните измененную информацию в файле

или укажите длину буфера и передайте этот буфер как аргумент данных. После этого можно воспользоваться функциями кэширования вариантов httpd_variant_insert() и httpd_variant_lookup() для размещения варианта в кэше.

Примеры API

Вам будет проще начать разработку собственных функций API Caching Proxy, если вы ознакомитесь с примерами программ, находящимися в каталоге samples установочного компакт-диска Edge Components. Дополнительная информация приведена на Web-сайте WebSphere Application Server www.ibm.com/software/webservers/appserv/.

Глава 3. Пользовательские советники

В этом разделе описывается создание пользовательских советников для компонента Load Balancer.

Советники предоставляют информацию о распределении нагрузки

Советники - это программные посредники компонента Load Balancer, предоставляющие информацию о нагрузке на выбранном сервере. Для каждого стандартного протокола (HTTP, SSL и т.д.) существуют специальные советники. Периодически основной код компонента Load Balancer выполняет цикл советника, в течение которого анализируется состояние каждого сервера конфигурации.

Написание пользовательских советников для Load Balancer позволяет настроить параметры определения нагрузки серверов.

В операционных системах семейства Windows: Если используется протокол IPv6, то при установке Load Balancer для IPv4 и IPv6 необходимо изменить файл **protocol**, расположенный в каталоге C:\windows\system32\drivers\etc\.

Для использования протокола IPv6 необходимо вставить в файл следующую строку:

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

Стандартный советник

Как правило, советники выполняют распределение нагрузки следующим образом.

1. Советник периодически устанавливает соединение с каждым сервером и отправляет ему сообщение-запрос. Содержимое сообщения зависит от протокола, запущенного на сервере; например, советник HTTP отправляет серверу запрос HEAD.
2. Советник ожидает ответ от сервера. После получения ответа советник вычисляет значение нагрузки для данного сервера и сообщает его пользователю. Для разных советников методы вычисления значения нагрузки могут быть различны; большинство советников измеряют время ожидания ответа от сервера (в миллисекундах).
3. Советник передает это значение диспетчеру Load Balancer. Кроме того, оно отображается в столбце Порт отчета диспетчера. На основе полученного значения, а также других показателей, установленных администратором, диспетчер принимает решение о распределении нагрузки между поступающими на сервер запросами.
4. Если сервер не отвечает на запрос, советник возвращает отрицательное значение (-1). На основе этой информации диспетчер принимает решение о приостановке службы для определенного сервера.

Стандартные советники, поставляемые вместе с компонентом Load Balancer, выполняют следующие функции. Подробная информация о советниках приведена в документе *WebSphere Application Server Load Balancer Administration Guide*

- Connect
- DB2
- DNS
- FTP

© Copyright IBM Corp. 2005

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

Для специальных протоколов, которые не поддерживаются стандартными советниками, необходимо создавать пользовательские советники.

Создание пользовательского советника

Пользовательский советник представляет собой небольшой код на языке Java, сохраненный в виде файле класса, который вызывается основным кодом компонента Load Balancer для определения загрузки сервера. Основной код предоставляет все необходимые административные службы, включая запуск и остановку экземпляра пользовательского советника, информацию о статусе советника, отчеты, ведение журнала событий, а также передачу результатов работы советника компоненту диспетчера.

Процедура вызова пользовательского советника включает следующие этапы.

1. Основной код Load Balancer устанавливает соединение с сервером.
2. Если сокет открыт, основной код вызывает функцию GetLoad соответствующего советника.
3. Функция GetLoad советника выполняет определенные пользователем действия для анализа состояния сервера, в том числе ожидает ответ от сервера. После получения ответа выполнение функции прекращается.
4. Основной код Load Balancer закрывает сокет и передает информацию о нагрузке диспетчеру. В зависимости от режима работы пользовательского советника (обычный режим или режим замены), основной код может выполнить некоторые дополнительные вычисления после завершения работы функции GetLoad.

Обычный режим и режим замены

Пользовательские советники могут взаимодействовать с Load Balancer либо в *обычном* режиме работы, либо в режиме *замены*.

Режим работы задается при помощи специального параметра метода конструктора в файле пользовательского советника. (Каждый советник может работать только в одном режиме, в зависимости от своего назначения).

В обычном режиме работы пользовательский советник обменивается данными с сервером, измеряет время обмена и вычисляет значение нагрузки. Затем основной код передает значение нагрузки диспетчеру. В случае успеха пользовательский советник возвращает нулевое значение, в противном случае - отрицательное значение.

Для выбора обычного режима работы присвойте флагу замены в конструкторе значение *false*.

В режиме замены основной код не выполняет каких-либо измерений времени. Вместо этого он выполняет указанные пользователем операции, основанные на специальных требованиях, и возвращает фактическое значение нагрузки. После получения этого значения основной код передает его в том же виде диспетчеру. Для получения оптимального результата рекомендуется настроить значение нагрузки в диапазоне от 10 до 1000, где 10 обозначает быстродействующий сервер, а 1000 - медленный.

Для выбора режима замены присвойте флагу замены в конструкторе значение *true*.

Соглашения об именах советников

Имя файла пользовательского советника должно быть указано в виде ADV_*имя*.java, где *имя* - это имя, выбранное для советника. Приставка ADV_ в начале полного имени файла должна содержать символы верхнего регистра, а все остальные символы должны быть указаны в нижнем регистре. Последнее требование позволяет вводить имя файла советника в команде запуска без учета регистра.

В соответствии с соглашениями об именах Java, имя класса, определенного в файле, должно совпадать с именем файла.

Компиляция

Пользовательские советники создаются на языке Java и компилируются при помощи компилятора Java, установленного в системе пользователя. В процедуре компиляции используются следующие файлы:

- Файл пользовательского советника
- Файл базовых классов ibmnd.jar, расположенный в каталоге *установочный_каталог*/servers/lib

Во время компиляции переменная среды classpath должна указывать и на файл пользовательского советника, и на файл базовых классов. Команда компилятора может быть указана в следующем формате:

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

В этом примере используется путь установки, принятый по умолчанию в операционных системах Linux и UNIX. Файлу советника присвоено имя ADV_*имя*.java, и он хранится в текущем каталоге.

В результате компиляции создается файл класса, ADV_*имя*.class. Перед запуском советника необходимо скопировать этот файл в каталог *установочный_каталог*/servers/lib/CustomAdvisors/.

Примечание: Пользовательские советники можно компилировать в одной операционной системе, а выполнять в другой. Например, вы можете скомпилировать советник в операционной системе Windows, скопировать файл класса (в двоичном формате) в операционную систему Linux и запустить в ней пользовательский советник.

Выполнение пользовательского советника

Для запуска пользовательского советника необходимо сначала скопировать файл класса советника в каталог lib/CustomAdvisors/ в системе, где установлен Load Balancer. Например, пользовательский советник с именем myping (ADV_myping.class) находится в каталоге *установочный_каталог*/servers/lib/CustomAdvisors/

Настройте Load Balancer, запустите диспетчер и выполните команду для запуска пользовательского советника. Для запуска пользовательского советника необходимо указать его имя без приставки ADV_ и расширения файла:

```
dscontrol advisor start myping номер_порта
```

Указанный в команде номер порта - это порт, к которому подключается советник при установлении соединения с целевым сервером.

Обязательные процедуры

Как и все стандартные советники, пользовательский советник расширяет функциональные возможности базового класса советника, ADV_Base. Базовый класс выполняет большинство функций советника, например, передает значения нагрузки диспетчеру для дальнейшей обработки. Кроме того, он создает сокеты и закрывает операции, а также содержит методы отправки и приема, необходимые для работы советника. Советник применяется исключительно для обмена данными с исследуемым сервером через указанный порт. Методы базового класса советника, основанные на протоколе TCP, используются для вычисления нагрузки. При необходимости для замены текущего значения нагрузки на новое значение, возвращаемое советником, можно использовать специальный флаг в конструкторе базового класса советника.

Примечание: Базовый класс советника отправляет значения нагрузки через указанные промежутки времени, в зависимости от значения, заданного в конструкторе. Если посредник не успел завершить обработку и не может вернуть текущее значение нагрузки, то базовый класс использует предыдущее значение.

Советники содержат следующие методы базового класса:

- Процедура конструктора. Конструктор вызывает конструктор базового класса.
- Метод ADV_AdvisorInitialize. Этот метод используется для выполнения дополнительных действий после завершения инициализации базового класса.
- Процедура getLoad. Используется для создания сокета; функция getLoad отправляет только запросы на передачу и прием.

Более подробную информацию об этих процедурах можно найти ниже.

Порядок поиска

Пользовательские советники вызываются, если поиск среди стандартных советников не дал результата. Если компоненту Load Balancer не удалось найти требуемый советник в списке стандартных советников, он просматривает список пользовательских советников. Дополнительная информация об использовании советников приведена в документе *WebSphere Application Server Load Balancer Administration Guide*.

Присваивание имен файлам и использование пути к файлу

При работе с пользовательскими советниками необходимо выполнять следующие требования к именам файлов и путям.

- Имя файла пользовательского советника должно содержать буквенные символы нижнего регистра, чтобы при вводе имен файлов в командной строке не учитывалось состояние регистра. Имя советника должно содержать приставку ADV_
- Файл класса пользовательского советника должен храниться в каталоге lib/CustomAdvisors. В операционных системах Linux and UNIX по умолчанию используется каталог /opt/ibm/edge/lb/servers/lib/CustomAdvisors, а в операционных системах Windows - каталог C:\Program Files\IBM\edge\lb\servers\lib\CustomAdvisors\.

Методы пользовательского советника и вызов функций

Конструктор (предоставляемый базовым классом советника)

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

sName
Имя пользовательского советника.

sVersion
Версия пользовательского советника.

iDefaultPort
Номер порта для подключения к серверу, если в вызове функции номер порта не указан.

iInterval
Интервал между запросами на сервер.

sDefaultLogFileName
Этот параметр является обязательным, однако он не используется. Он может принимать только одно значение: "" (нулевая строка)

replace
Определяет, работает ли советник в режиме *замены*. Допустимые значения:

- true – Заменять значение нагрузки, вычисленное основным кодом советника, на значение, полученное от пользовательского советника.
- false – Добавлять значение нагрузки, вычисленное пользовательским советником, к значению нагрузки, вычисленному основным кодом советника.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

Этот метод применяется для инициализации пользовательского советника. Этот метод вызывается после запуска базового модуля советника.

Обычно (в том числе и в стандартных советниках) этот метод не используется и содержит только оператор *return*. Этот метод может применяться для вызова метода suppressBaseOpeningSocket, который действителен только при вызове из метода ADV_AdvisorInitialize().

getLoad()

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

iConnectTime
Продолжительность времени (в мс), необходимая для установления соединения. Этот параметр измеряется основным кодом советника и передается коду пользовательского советника, который может принять или проигнорировать его при возвращении значения нагрузки. Если соединение не установлено, возвращается значение -1.

caller
Экземпляр базового класса советника, содержащего основные методы советника.

Вызов функций из пользовательских советников

В следующих разделах описываются методы (или функции), вызываемые из пользовательских советников. Эти методы поддерживаются основным кодом советника.

Некоторые из этих функций могут быть вызваны напрямую, например, *имя_функции()*, а для других необходимо указывать приставку `caller`. Приставка *Caller* обозначает экземпляр базового советника, который поддерживает запущенный пользовательский советник.

ADVLOG()

Функция ADVLOG применяется для записи текстового сообщения в файл журнала советника. Формат:

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

logLevel
Значение статуса сообщения, записываемого в файл журнала. Файл протокола советника состоит из нескольких частей; самые срочные сообщения имеют статус 0, менее срочные сообщения имеют более высокие значения статуса. Наиболее подробные сообщения имеют статус 5. Статусы применяются для управления типами сообщений, которые пользователь получает в режиме реального времени (Для установки уровня детальности сообщений используется команда **dscontrol**). Неисправимые ошибки всегда должны иметь уровень 0.

message
Сообщение, записываемое в файл журнала. Значение данного параметра представляет собой обычную Java-строку.

getAdvisorName()

Функция getAdvisorName возвращает Java-строку с суффиксом, который указывает имя пользовательского советника. Например, для советника ADV_cdload.java эта функция возвращает значение cdload.

У этой функции нет параметров.

Обратите внимание, что это значение не изменяется в процессе создания одного экземпляра советника.

getAdviseOnPort()

Функция getAdviseOnPort возвращает номер порта, к которому подключен пользовательский советник. Возвращаемое значение представляет собой целое число Java (int). У этой функции нет параметров.

Обратите внимание, что это значение не изменяется в процессе создания одного экземпляра советника.

caller.getCurrentServer()

Функция getCurrentServer возвращает IP-адрес текущего сервера. Возвращаемое значение представляет собой строку Java в формате IP-адреса, например, 128.0.72.139

Обычно этот адрес изменяется при каждом вызове пользовательского советника, поскольку основной код советника последовательно отпрашивает все серверы.

У этой функции нет параметров.

caller.getCurrentCluster()

Функция getCurrentCluster возвращает IP-адрес кластера сервера. Возвращаемое значение представляет собой строку Java в формате IP-адреса, например, 128.0.72.139

Обычно этот адрес изменяется при каждом вызове пользовательского советника, поскольку основной код советника последовательно опрашивает все кластеры сервера.

У этой функции нет параметров.

getInterval()

Функция getInterval возвращает интервал советника, т.е. число секунд между циклами советника. Это значение равно значению по умолчанию, указанному в конструкторе пользовательского советника, если оно не было изменено во время выполнения с помощью команды **dscontrol**.

Возвращаемое значение представляет собой целое число Java (int). У этой функции нет параметров.

caller.getLatestLoad()

Функция getLatestLoad позволяет получить последнее значение нагрузки для заданного сервера. Значения нагрузки сохраняются во внутренних таблицах основным кодом советника и демоном диспетчера.

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

Для определения одного объекта сервера необходимо три аргумента.

cluster_IP
: IP-адрес кластера объекта сервера, для которого необходимо получить текущее значение нагрузки. Этот аргумент должен представлять собой строку Java в формате IP-адреса, например, 245.145.62.81

port
: Номер порта объекта сервера, для которого необходимо получить текущее значение нагрузки.

server_IP
: IP-адрес объекта сервера, для которого необходимо получить текущее значение нагрузки. Этот аргумент должен представлять собой строку Java в формате IP-адреса, например, 192.255.201.3

Возвращаемое значение представляет собой целое число.

- Положительное значение обозначает фактическое значение нагрузки, присвоенное проверяемому объекту.
- Значение -1 обозначает, что запрашиваемый сервер не отвечает.

- Значение -2 обозначает, что статус запрашиваемого сервера неизвестен.

Эта функция применяется, когда необходимо, чтобы поведение одного протокола или порта зависело от поведения другого протокола или порта. Например, эта функция может применяться в пользовательском советнике для отключения определенного сервера приложений, если сервер Telnet на том же самом компьютере отключен.

caller.receive()

Функция receive принимает данные от сокета.

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

Параметр *response* представляет собой буфер, в который помещаются принятые данные. Функция возвращает одно из следующих значений:

- 0 обозначает, что данные отправлены успешно.
- Отрицательное число обозначает ошибку.

caller.send()

Функция send использует установленное соединение для отправки пакета данных на сервер через указанный порт.

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

Параметр *command* представляет собой строку, содержащую данные для передачи серверу. Функция возвращает одно из следующих значений:

- 0 обозначает, что данные отправлены успешно.
- Отрицательное число обозначает ошибку.

suppressBaseOpeningSocket()

Функция suppressBaseOpeningSocket позволяет основному коду советника установить TCP-соединение с сервером от имени пользовательского советника. Если советник не использует прямое соединение с сервером для определения его состояния, создавать данный сокет не требуется.

Данная функция вызывается из процедуры ADV_AdvisorInitialize, причем только один раз.

У этой функции нет параметров.

Примеры

Ниже приведены примеры использования пользовательских советников.

Стандартный советник

Следующий пример исходного кода похож на код для стандартного советника компонента Load Balancer на базе HTTP-протокола. Процедура выполнения:

1. Отправляется запрос на передачу, команда ″HEAD/HTTP″.
2. Принимается ответ. Информация не анализируется, но после получения ответа выполнение функции getLoad прекращается.
3. Функция getLoad возвращает 0 в случае успеха или -1 в случае ошибки.

Этот советник работает в обычном режиме, поэтому значение нагрузки вычисляется на основе времени (в мс), необходимого для создания сокета и выполнения операций отправки, приема и закрытия.

```
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";

//--------
// Конструктор

  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;                                    // обычно пустая процедура
  }

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

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

    iRc = caller.send(ADV_SEND_REQUEST);   // отправляет HTTP-запрос
                                             // серверу
    if (0 <= iRc) {    // если операция отправки выполнена успешно
      StringBuffer sbReceiveData = new StringBuffer(""); // выделение буфера
                                                          // для ответа
      iRc = caller.receive(sbReceiveData);   // получение результата

      // анализ полученного результата

      if (0 <= iRc) {       // если операция приема выполнена успешно
        iLoad = 0;          // возвращает 0 в случае успеха
      }                     // (значение нагрузки советника игнорируется
    }                       //  в обычном режиме)
    return iLoad;
  }
}
```

Советник с побочным потоком

Этот пример демонстрирует закрытие стандартного сокета, созданного основным кодом советника. Вместо него советник открывает сокет с побочным потоком Java для соединения с сервером. Данная процедура применяется в том случае, если сервер использует для приема запросов советника нестандартный порт.

В данном примере сервер использует порт 11999 и при получении запроса возвращает значение нагрузки с шестнадцатеричным целым числом ″4″. В этом случае советник работает в режиме замены, т.е. последнему параметру конструктора советника присвоено значение true, и основной код советника использует вместо прошедшего времени возвращаемое значение нагрузки.

Обратите внимание на функцию supressBaseOpeningSocket() в процедуре инициализации. Закрывать сокет, если данные не отправляются, не требуется. Например, создание сокета может потребоваться для проверки возможности установления соединения с сервером. Перед использованием данного примера определите назначение вашего приложения.

```
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;

  // создание массива байт с сообщением-запросом о нагрузке
  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);         // значение параметра режима замены равно true
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // сообщает основному коду о том, что не нужно
                                   // создавать стандартный сокет
    return;
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;   // порт для подключения к серверу

    string sServer = caller.getCurrentServer();   // адреса сервера
    try {
      socket soServer = new Socket(sServer, iControlPort);  // создать сокет
                                                            // для сервера
      DataInputStream disServer = new DataInputStream(
                                      soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                       soServer.getOutputStream());

      int iRecvTimeout = 10000;  // задает значение тайм-аута (в мс)
                                 // для приема данных
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // отправляет сообщение серверу
      dosServer.flush();

      iLoad = disServer.readByte();   // получает ответ от сервера
```

```
    } catch (exception e) {
      system.out.println("Caught exception " + e);
    }
    return iLoad;    // возвращает значение нагрузки для сервера
  }
}
```

Посредник с двумя портами

Следующий пример пользовательского советника демонстрирует возможность обнаружения сбоя одного порта сервера на основании его собственного состояния и состояния другого демона сервера, запущенного для другого порта того же сервера. Например, если демон HTTP перестает отвечать через порт 80, может потребоваться остановить перенаправление трафика демону SSL через порт 443.

Этот советник ведет себя агрессивнее, чем стандартные советники, поскольку любой сервер, который не отправляет ответ на его запрос, рассматривается ним как приостановленный и помечается как *отключенный*. Стандартные советники считают, что если сервер не отвечает, то он очень медленный. Этот же советник помечает сервер как отключенный как для порта HTTP, так и для порта SSL на основании отсутствия ответа от одного из них.

Для использования этого пользовательского советника администратор должен запустить два экземпляра советника: один для порта HTTP и второй для порта SSL. Посредник создает два экземпляра статических глобальных хэш-таблиц: один для порта HTTP и второй для порта SSL. В этих хэш-таблицах хранятся результаты попыток советника установить соединение с соответствующим демоном сервера. Значение, возвращаемое каждым советником в базовый класс советника, зависит от его способности установить соединение с собственным демоном сервера и способности другого советника установить соединение со своим демоном.

Используются следующие методы:

- ADV_nte(): простой объект-контейнер для хранения информации о сервере. Эти объекты хранятся в хэш-таблице. Каждый объект имеет свою временную метку, которая позволяет проверить его срок действия.
- putNte() и getNte(): синхронизированные методы, управляющие доступом двух экземпляров советника к хэш-таблице.
- getLoadHTTP: метод, отправляющий запросы на HTTP-сервер. Этот метод представляет собой низкоуровневую процедуру, он не собирает и не использует информацию о протоколе SSL.
- getLoadSSL(): метод, отправляющий запросы на SSL-сервер. Этот метод представляет собой низкоуровневую процедуру, он не собирает и не использует информацию о протоколе HTTP.
- getLoad(): процедура точки входа для данного пользовательского советника. Этот метод может обрабатывать данные обоих протоколов, а также сохранять и выбирать данные из хэш-таблицы. Данная процедура связывает два порта.

Обнаруживаются следующие условия возникновения ошибок:

- Сервер не отвечает — Базовые классы посредников периодически отправляют команду ping на сервер. Если адрес недоступен, сервер помечается как отключенный. Ни один из экземпляров пользовательского советника не вызывается, и оба сервера помечаются как отключенные.
- Один демон на сервере перестает отвечать, однако второй демон продолжает работу — Если базовый код при попытке установить соединение с сервером получает отказ, он помечает этот сервер как отключенный. Исходный код пользовательского советника для данного протокола не вызывается. Хотя

пользовательский советник для другого протокола продолжает обмениваться данными с этим сервером, он узнает из хэш-таблицы, что второй пользовательский советник не может установить соединение с демоном сервера. Таким образом, посредник для второго протокола также помечает этот сервер как отключенный.

- Один демон не отправляет ответ, а второй отправляет — Пользовательский советник для неответившего протокола обнаруживает сбой связи, помечает сервер как отключенный и сохраняет эти данные в хэш-таблице. Пользовательский советник для второго порта получает эти данные из хэш-таблицы и также помечает сервер как отключенный.

В следующем примере программы используется порт 80 для протокола HTTP и порт 443 для протокола SSL, однако вместо них можно указать любую другую комбинацию портов.

```
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;

//--------
// Определение элементов хэш-таблиц в пользовательском советнике

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

//--------
// конструктор

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

//--------
// проверка срока действия элемента
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();
         // присвоить параметру срока действия
         // значение, равное трем циклам советника
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

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

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

```
// получение значения

  public int getLoadValue() { return iLoad; }

//--------
// дубликат (позволяет избежать повреждения между нитями)

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

}

//--------
// определение пользовательского советника

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;

  //--------
  // определение таблиц для хранения информации о портах

  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";

  //--------
  // создание массива байт с приветственным сообщением SSL-клиента

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
    (byte)0x01,               // приветственное сообщение клиента
    (byte)0x03, (byte)0x00,   // версия протокола SSL
    (byte)0x00, (byte)0x03,   // длина ключа (в байтах)
    (byte)0x00, (byte)0x00,   // длина ИД сеанса (в байтах)
    (byte)0x00, (byte)0x10,   // длина данных вызова (в байтах)
    (byte)0x00, (byte)0x00, (byte)0x03,   // спецификация шифра
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)Ox20,  // данные вызова
    (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 };

  //--------
  // конструктор

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);    // false = load balancer измеряет время ответа
    setAdvisor ( this );
  }

  //--------
  // ADV_AdvisorInitialize
```

```
 public void ADV_AdvisorInitialize() {
   return;
 }

 //--------
 // синхронизированные процедуры доступа PUT и GET для хэш-таблиц

 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 - определение нагрузки HTTP на основе ответа сервера

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

   int iRc = caller.send(ADV_HTTP_REQUEST_STRING);  // отправляет сообщение-запрос
                                                    // для сервера
   if (0 <= iRc) {    // произошел сбой?
     StringBuffer sbReceiveData = new StringBuffer("")    // выделение буфера
                                                          // для ответа
     iRc = caller.receive(sbReceiveData);    // получение ответа от сервера

     if (0 <= iRc) {     // произошел сбой?
       if (0 < sbReceiveData.length()) {    // приняты ли данные?
         iLoad = SUCCESS;        // игнорирует принятые данные и
                                 // возвращает код SUCCESS
       }
     }
   }
   return iLoad;
 }

 //--------
 // getLoadSSL() - определение нагрузки SSL на основе ответа сервера

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

   int iSocket = caller.getAdvisorSocket();   // отправляет запрос серверу
   CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                    abClientHello);
   int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

   if (0 <= iRc) {    // произошел сбой?
     StringBuffer sbReceiveData = new StringBuffer("");  // выделение буфера
                                                         // для ответа
     iRc = caller.receive(sbReceiveData);     // получение ответа от
                                              // серверу
     if (0 <= iRc) {     // произошел сбой?
       if (0 < sbReceiveData.length()) {    // приняты ли данные?
         iLoad = SUCCESS;   // игнорирует принятые данные и возвращает код SUCCESS
       }
     }
   }
   return iLoad;
```

```
}

//--------
// getLoad - объединение результатов методов HTTP и SSL

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

  String sCluster = caller.getCurrentCluster();   // текущий адрес кластера
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;     // ключ хэш-таблицы

  if (ADV_TWOP_PORT_HTTP == iPort) {              // обработка HTTP-сервера
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);
                // получение значения нагрузки для HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);
               // сохранение значения нагрузки
                                                    // о протоколе SSL
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);
    // получение информации
    // для протокола HTTP
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) {
               // проверка временной метки
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {
        // работает ли SSL?
          iLoad = iLoadHTTP;
        } else {    // SSL не работает, поэтому HTTP-сервер
                    // помечается как отключенный
          iLoad= ADV_HOST_INACCESSIBLE;
        }
      } else {      // срок действия информации SSL истек,
                    // поэтому HTTP-сервер помечается как
                    // отключенный
        iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {        // значение нагрузки для SSL отсутствует,
                    // передача результатов выполнения метода
                    // getLoadHTTP()
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {
               // обработка SSL-сервера
    iLoadSSL = getLoadSSL(iConnectTime, caller);
               // получение значения нагрузки для SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL);
               // сохранение информации о нагрузке для SSL

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);
        // получение информации
       // для протокола HTTP
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) {
               // проверка временной метки
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {
               // работает ли SSL?
          iLoad = iLoadSSL;
        } else {   // HTTP-сервер не работает, поэтому он
```

```
                    // помечается как отключенный
            iLoad = ADV_HOST_INACCESSIBLE;
          }
        } else {     // срок действия информации HTTP истек,
                     // сервер помечается как отключенный
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {       // значение нагрузки для HTTP отсутствует,
                     // передача результатов выполнения метода
                     // getLoadSSL()
        iLoad = iLoadSSL;
      }
    }

  //--------
  // обработчик ошибок

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

Советник WebSphere Application Server

Пример пользовательского советника для сервера WebSphere Application Server расположен в каталоге *установочный_каталог*/servers/samples/CustomAdvisors/. В этом документе приводится только часть исходного кода.

- ADV_was.java: исходный файл советника, скомпилированный и запущенный в системе с установленным компонентом Load Balancer.
- LBAdvisor.java.servlet: исходный код сервлета, который необходимо переименовать на LBAdvisor.java, скомпилировать и запустить на сервере WebSphere Application Server.

Полный текст исходного кода советника намного сложнее приведенного здесь отрывка. В него добавлена специальная процедура анализа, более компактная по сравнению с процедурой StringTokenizer (см. пример выше).

Более сложная часть исходного кода содержится в файле сервлета Java. Кроме того, сервлет содержит два метода, необходимых в соответствии со спецификацией сервлета: init() и service(), и еще один метод: run(), необходимый для класса Java.lang.thread.

- метод init() вызывается один раз на этапе инициализации. Этот метод создает нить с именем _checker, которая выполняется независимо от функций, вызываемых из советника, и находится в режиме ожидания в течение некоторого времени до повторного вызова цикла обработки.
- метод service() выполняется при каждом вызове сервлета. В данном случае метод вызывается советником. Метод service() отправляет поток ASCII-символов в выходной поток.
- метод run() содержит основную часть исполняемого кода. Он вызывается методом start(), который вызывается из метода init().

Ниже приведены некоторые фрагменты кода сервлета.

```
...

  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;
  }

  ...
```

Использование данных, полученных от советников

Как при использовании стандартных функций, так и при добавлении новых функций к исходному коду пользовательского советника возникает необходимость проверить возвращаемые значения нагрузки и выполнить соответствующие действия. Для этой цели используются Java-класс StringTokenizer и связанные методы.

Пример стандартной команды HTTP: `GET /index.html HTTP/1.0`

Стандартный ответ на эту команду выглядит следующим образом:

```
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>
```

Наиболее важные элементы указаны в первой строке, в частности код возврата HTTP.

В соответствии со спецификацией протокола HTTP существуют следующие группы кодов возврата:

- Коды возврата 2xx: выполнено без ошибок
- Коды возврата 3xx: перенаправление
- Коды возврата 4xx: ошибки клиента
- Коды возврата 5xx: ошибки сервера

Если вам заранее известны возможные коды возврата сервера, такой уровень детализации данных, как в этом примере, вам может и не потребоваться. Следует отметить, что ограничение списка отслеживаемых кодов возврата может снизить гибкость вашей программы в будущем.

Ниже приведен пример автономной программы на языке Java, содержащей минимальный HTTP-клиент. Этот пример вызывает простой универсальный анализатор для проверки ответов 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());
    }
  }
}
```

Примечания

Первое издание (май 2006 года)

Настоящая документация была разработана для продуктов и услуг, предлагаемых на территории США.

IBM может не предоставлять продукты, услуги и функции, упомянутые в данном документе, в других странах. Информацию о продуктах и услугах, предлагаемых в вашей стране, вы можете получить в местном представительстве IBM. Ссылка на продукт, программу или услугу IBM не означает, что может применяться только этот продукт, программа или услуга IBM. Вместо них можно использовать любые другие функционально эквивалентные продукты, программы или услуги, не нарушающие прав IBM на интеллектуальную собственность. Ответственность за применение и проверку продуктов, программ и услуг, предоставляемых другими фирмами, лежит на пользователе.

IBM могут принадлежать патенты или заявки на патенты на технологии, обсуждаемые в этом документе. Предоставление вам настоящего документа не означает предоставления каких-либо лицензий на эти патенты. Запросы на лицензии следует отправлять в письменном виде по адресу:

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

Запросы на лицензии, связанные с информацией DBCS, следует направлять в отдел интеллектуальной собственности в местном представительстве IBM или в письменном виде по следующему адресу:

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

Следующий абзац не относится к Великобритании, а также к другим странам, в которых это заявление противоречит местному законодательству:

ФИРМА INTERNATIONAL BUSINESS MACHINES CORPORATION ПРЕДОСТАВЛЯЕТ НАСТОЯЩУЮ ПУБЛИКАЦИЮ НА УСЛОВИЯХ “КАК ЕСТЬ”, БЕЗ КАКИХ-ЛИБО ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ, ВКЛЮЧАЯ, НО НЕ ОГРАНИЧИВАЯСЬ ЭТИМ, НЕЯВНЫЕ ГАРАНТИИ СОБЛЮДЕНИЯ ПРАВ, КОММЕРЧЕСКОЙ ЦЕННОСТИ И ПРИГОДНОСТИ ДЛЯ КАКОЙ-ЛИБО ЦЕЛИ. В некоторых странах запрещается отказ от каких-либо явных и подразумеваемых гарантий при заключении определенных договоров, поэтому данное заявление может не действовать в вашем случае.

В данной публикации могут встретиться технические неточности и типографские опечатки. В содержание документа могут вноситься периодические изменения, которые будут отражены в последующих изданиях. IBM может вносить улучшения и изменения в продукты и программы, описанные в настоящей публикации, в любое время без уведомления.

© Copyright IBM Corp. 2005

Любые ссылки на Web-сайты других фирм приведены в данной публикации исключительно для удобства и не предназначены для поддержки или рекламы этих Web-сайтов. Материалы, размещенные на этих Web-сайтах, не являются частью информации по данному продукту IBM, и ответственность за применение этих материалов лежит на пользователе.

IBM может использовать и распространять любую предоставленную вами информацию на свое усмотрение без каких-либо обязательств перед вами.

Для получения информации об этой программе для обеспечения: (i) обмена информацией между независимо созданными программами и другими программами (включая данную) и (ii) взаимного использования информации, полученной в ходе обмена, пользователи данной программы могут обращаться по адресу:

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

В некоторых случаях такая информация может быть предоставлена на особых условиях, в том числе - за дополнительную плату.

Описанная в этой информации лицензионная программа и все связанные с ней лицензионные материалы предоставляются IBM в соответствии с условиями Международного соглашения о лицензии на программу IBM или любого другого эквивалентного соглашения.

Все приведенные показатели производительности были получены в управляемой среде. В связи с этим результаты, полученные в реальной среде, могут существенно отличаться от приведенных. Некоторые измерения могли быть выполнены в системах, находящихся на этапе разработки, поэтому результаты измерений, полученные в серийных системах, могут отличаться от приведенных. Кроме того, некоторые показатели были получены методом экстраполяции. Реальные результаты могут отличаться от указанных. Пользователи, работающие с этим документом, должны удостовериться, что используемые ими данные применимы в имеющейся среде.

Информация о продуктах других изготовителей получена от поставщиков этих продуктов, из их официальных сообщений и других общедоступных источников. IBM не выполняла тестирование этих продуктов других фирм и не может подтвердить точность заявленной информации об их производительности, совместимости и других свойствах. Запросы на получение дополнительной информации об этих продуктах должны направляться их поставщикам.

Все заявления, касающиеся намерений и планов IBM, могут изменяться и отзываться без предварительного уведомления, и отражают только текущие цели и задачи.

Эта информация содержит примеры данных и отчетов, применяемых в ежедневной работе. Для максимальной достоверности в них могут быть приведены имена отдельных лиц, названия компаний, товарных знаков и продуктов. Все эти имена и названия вымышлены, и любое их сходство с реальными именами, названиями и адресами носит совершенно случайный характер.

При просмотре этой информации в электронном виде фотографии и цветные иллюстрации могут быть не показаны.

Товарные знаки

Ниже перечислены товарные знаки корпорации IBM в США и/или других странах:

- AIX
- IBM
- ViaVoice
- WebSphere

Java и все товарные знаки на основе Java являются товарными знаками Sun Microsystems, Inc. в США и/или других странах.

Microsoft, Windows, Windows NT и логотип Windows являются товарными знаками корпорации Microsoft в США и/или других странах.

Intel, Intel Inside (логотипы), MMX и Pentium являются товарными знаками корпорации Intel в США и/или других странах.

UNIX является зарегистрированным товарным знаком The Open Group в США и/или других странах.

Linux - это товарный знак Линуса Торвальдса в США и других странах.

Названия других компаний, продуктов и услуг могут являться товарными или служебными знаками других компаний.

Индекс

© Copyright IBM Corp. 2005

Напечатано в Дании

GC43-0477-00

Spine information:

WebSphere Application Server Руководство по программированию для Edge Components *Версия 6.1* GC43-0477-00