WebSphere Application Server

Podręcznik programowania dla produktu Edge Components

Wersja 6.1

GC85-0194-00

WebSphere Application Server

Podręcznik programowania dla produktu Edge Components

Wersja 6.1

GC85-0194-00

Uwaga

Przed rozpoczęciem korzystania z niniejszych informacji i opisywanego w nich produktu należy zapoznać się z ogólnymi informacjami znajdującymi się w sekcji “Uwagi” na stronie 57.

Pierwsza edycja (maj 2006)

To wydanie dotyczy wersji:
WebSphere Application Server Wersja 6.1

oraz wszystkich kolejnych wersji tego produktu, chyba że w nowym wydaniu zostanie stwierdzone inaczej.

Publikacje można zamówić za pośrednictwem przedstawiciela handlowego lub lokalnego oddziału firmy IBM.

Spis treści

Rysunki

Informacje na temat tego podręcznika

W tej sekcji opisano przeznaczenie, organizację i konwencje stosowane w niniejszej publikacji (*Podręcznik programowania dla produktu WebSphere Application Server Edge Components*).

Użytkownicy, dla których jest przeznaczony ten podręcznik

Ten podręcznik opisuje aplikacyjne interfejsy programistyczne (API) dostępne na potrzeby dostosowywania produktu Edge Components serwera WebSphere Application Server Wersja 6.1. Informacje te są przeznaczone dla programistów tworzących aplikacje wtyczek i wprowadzających inne modyfikacje. Korzystać z nich mogą również projektanci sieci i administratorzy systemów w celu uzyskania informacji o możliwych typach dostosowań.

Co czytelnik powinien już wiedzieć

Korzystanie z informacji zawartych w tym podręczniku wymaga zrozumienia procedur programistycznych w języku programowania Java lub C, w zależności od tego, jaki interfejs API będzie używany. Metody i struktury dostępne w każdym prezentowanym interfejsie są udokumentowane, ale czytelnik musi wiedzieć, w jaki sposób skonstruować własną aplikację, skompilować ją dla używanego systemu, a następnie przetestować. Dla niektórych interfejsów udostępniono kod przykładowy, ale stanowi on tylko przykład konstruowania własnej aplikacji.

Konwencje i terminologia używane w tym podręczniku

W tej dokumentacji użyto następujących konwencji drukarskich i konwencji tworzenia kluczy.

Tabela 1. Konwencje używane w tym podręczniku

Konwencja	Znaczenie
Pogrubienie	W przypadku odwołania do graficznego interfejsu użytkownika (GUI, graphical user interface) pogrubienie wskazuje menu, elementy menu, etykiety, przyciski, ikony i foldery. Może także zostać użyte do wyróżnienia nazw komend, które w przeciwnym razie mogłyby zostać pomylone z otaczającym tekstem.
Czcionka o stałej szerokości	Wskazuje tekst, który należy wprowadzić w wierszu komend. Za pomocą czcionki o stałej szerokości oznaczono również tekst ekranowy, przykłady kodu oraz cytaty z plików.
Kursywa	Wskazuje wartości zmiennych, które należy udostępnić (na przykład dla zmiennej *fileName* należy podać nazwę pliku). Kursywy użyto również jako wyróżnienia oraz oznaczono za jej pomocą tytuły podręczników.
Ctrl-*x*	Gdzie *x* to nazwa klawisza. Wskazuje kombinację klawiszy control-znak. Na przykład Ctrl-c oznacza konieczność przytrzymania wciśniętego klawisza Ctrl i naciśnięcia klawisza c.
Return	Odwołuje się do klawisza oznaczonego wyrazem Return, wyrazem Enter lub strzałką w lewo.
%	Reprezentuje zachętę powłoki komend powłoki systemów Linux i UNIX dla komendy, która nie wymaga uprawnień administratora.
#	Reprezentuje zachętę powłoki komend systemów Linux i UNIX dla komendy, która wymaga uprawnień administratora.
C:\	Reprezentuje zachętę wiersza komend systemu Windows.

Tabela 1. Konwencje używane w tym podręczniku (kontynuacja)

Konwencja	Znaczenie
Wprowadzanie komend	Jeśli w instrukcji napisano “wprowadź” lub “wydaj” komendę, należy wpisać komendę i nacisnąć klawisz Return. Na przykład instrukcja “Wprowadź komendę **ls**” oznacza, że należy wpisać **ls** w wierszu komend i nacisnąć klawisz Return.
[ ]	Oznaczają opcjonalne elementy w opisach składni.
{ }	Oznaczają listy, z których należy wybrać pozycję w opisach składni.
|	Oddziela pozycje na liście wyborów oznaczonych za pomocą znaków { } (nawias klamrowy) w opisach składni.
...	Wielokropki w opisach składni wskazują możliwość powtórzenia poprzedniego elementu jeden lub większą liczbę razy. Wielokropki w przykładach oznaczają, że pominięto informacje w celu zachowania zwięzłości.

Ułatwienia dostępu

Opcje ułatwień dostępu pomagają posługiwać się oprogramowaniem osobom niepełnosprawnym fizycznie, na przykład z ograniczeniami w zakresie ruchu lub z wadami wzroku. Poniżej przedstawiono najważniejsze opcje ułatwień dostępu produktu WebSphere Application Server Wersja 6.1:

- Korzystając z oprogramowania lektora ekranowego oraz cyfrowego syntezatora mowy, można usłyszeć tekst wyświetlany na ekranie. Możliwe jest także wprowadzanie danych oraz poruszanie się po interfejsie użytkownika za pomocą oprogramowania służącego do rozpoznawania mowy, takiego jak IBM ViaVoice.
- Operacje na opcjach można przeprowadzać, korzystając z klawiatury zamiast z myszy.
- Do konfigurowania opcji serwera Application Server i administrowania nimi można użyć standardowych edytorów tekstu lub interfejsów wiersza komend zamiast udostępnionych interfejsów graficznych. Więcej informacji o ułatwieniach dostępu do poszczególnych opcji można znaleźć w dokumentacji dotyczącej danej opcji.

Pokrewne dokumenty i serwisy WWW

- *Pojęcia, planowanie i instalowanie produktu Edge Components*, GC85-0198-00
- *Podręcznik administrowania komponentem Buforujący serwer proxy*, GC31-6920-00
- *System równoważenia obciążenia Podręcznik administrowania*, GC31-6921-00
- Główny serwis WWW firmy IBM: *www.ibm.com/*
- IBM WebSphere Application Server: *www.ibm.com/software/webservers/appserv/*
- Serwis WWW biblioteki produktu IBM WebSphere Application Server: *www.ibm.com/software/webservers/appserv/library.html*
- Serwis WWW wsparcia dla produktu IBM WebSphere Application Server: *www.ibm.com/software/webservers/appserv/support.html*
- Centrum informacyjne produktu IBM WebSphere Application Server: *www.ibm.com/software/webservers/appserv/infocenter.html*
- Centrum informacyjne produktu IBM WebSphere Application Server Edge Components: *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

Jak przesłać swoje komentarze

Twoja opinia jest ważna i pomaga dostarczać najbardziej dokładne informacje o najwyższej jakości. Jeśli masz komentarze na temat tego podręcznika lub wszelkiej innej dokumentacji produktu Edge Components serwera WebSphere Application Server:

- Wyślij swoje komentarze pocztą elektroniczną na adres fsdoc@us.ibm.com. Pamiętaj o podaniu nazwy podręcznika, odpowiadającego mu numeru części, wersji serwera WebSphere Application Server i, jeśli to konieczne, konkretnego miejsca w tekście, którego dotyczy komentarz (na przykład numeru strony lub numeru tabeli).

Rozdział 1. Przegląd możliwości dostosowywania produktu Edge Components

Ten podręcznik omawia aplikacyjne interfejsy programistyczne (API) udostępniane na potrzeby produktu Edge Components serwera WebSphere Application Server (produkt Edge Components serwera WebSphere Application Server zawiera komponent Buforujący serwer proxy i komponent System równoważenia obciążenia). Udostępniono kilka interfejsów, które umożliwiają administratorom dostosowywanie instalacji w celu modyfikowania sposobu współpracy komponentów Edge Components lub umożliwiania współpracy z innymi systemami oprogramowania.

Ważne: Komponent Buforujący serwer proxy jest dostępny we wszystkich instalacjach komponentów brzegowych, z następującymi wyjątkami:

- Buforujący serwer proxy nie jest dostępny w instalacjach komponentów brzegowych działających na 64-bitowych procesorach Itanium 2 lub AMD Opteron.
- Buforujący serwer proxy nie jest dostępny w instalacjach komponentów brzegowych produktu System równoważenia obciążenia dla protokołów IPv4 i IPv6.

Interfejsy API przedstawione w tym dokumencie należą do kilku kategorii.

Dostosowywanie komponentu Buforujący serwer proxy

Komponent Buforujący serwer proxy ma wbudowanych w sekwencję przetwarzania kilka interfejsów umożliwiających dodanie niestandardowego przetwarzania lub zastąpienie przetwarzania standardowego. Dostosowania, które mogą być wykonane, obejmują modyfikowanie lub rozszerzanie czynności takich jak następujące:

- Uwierzytelnianie klienta
- Autoryzacja żądania
- Konwersja adresów URL na ścieżki do plików fizycznych
- Obsługa żądań
- Rejestrowanie
- Odpowiadanie na warunki błędu

Niestandardowe aplikacje, nazywane też wtyczkami komponentu Buforujący serwer proxy, wywoływane są we wstępnie określonych punktach sekwencji przetwarzania serwera proxy.

Interfejs API komponentu Buforujący serwer proxy został wykorzystany do zaimplementowania pewnych funkcji systemu. Na przykład obsługa protokołu LDAP serwera proxy została zaimplementowana jako wtyczka.

Rozdział 2, “Interfejs API komponentu Buforujący serwer proxy”, na stronie 3 opisuje szczegółowo ten interfejs i zawiera kroki służące do konfiguracji serwera proxy do korzystania z programów wtyczek.

Dostosowywanie komponentu System równoważenia obciążenia

System równoważenia obciążenia można dostosowywać poprzez tworzenie własnych doradców. Doradcy wykonują pomiary bieżącego obciążenia na serwerach. W przypadku niestandardowego doradcy można użyć własnej metody pomiaru obciążenia, która będzie odpowiednia dla systemu użytkownika. Jest to szczególnie ważne w przypadku niestandardowych systemów serwerów WWW lub systemów innych firm.

Rozdział 3, “Niestandardowi doradcy”, na stronie 37 zawiera szczegółowe informacje na temat tworzenia i używania niestandardowych doradców. Zawiera też przykładowy kod doradcy.

Znajdowanie kodu przykładowego

Przykładowy kod dla tych interfejsów API jest zawarty na dysku CD-ROM o nazwie Edge Components, w katalogu samples. Dodatkowe przykłady kodu są dostępne w serwisie WWW serwera WebSphere Application Server: www.ibm.com/software/webservers/appserv/.

Rozdział 2. Interfejs API komponentu Buforujący serwer proxy

W tej sekcji omówiono interfejs API (application programming interface) komponentu Buforujący serwer proxy: co to jest, dlaczego jest użyteczny i w jaki sposób działa.

Ważne: Komponent Buforujący serwer proxy jest dostępny we wszystkich instalacjach komponentów brzegowych, z następującymi wyjątkami:

- Buforujący serwer proxy nie jest dostępny w instalacjach komponentów brzegowych działających na 64-bitowych procesorach Itanium 2 lub AMD Opteron.
- Buforujący serwer proxy nie jest dostępny w instalacjach komponentów brzegowych produktu System równoważenia obciążenia dla protokołów IPv4 i IPv6.

Przegląd funkcji API komponentu Buforujący serwer proxy

Ten API to interfejs komponentu Buforujący serwer proxy, który umożliwia rozszerzanie podstawowych funkcji serwera proxy. Możliwe jest pisanie rozszerzeń (wtyczek) służących do dostosowanego przetwarzania, w tym:

- Rozszerzenie podstawowej procedury uwierzytelniania lub zastąpienie jej procesem właściwym dla serwisu.
- Dodanie procedur obsługi błędów w celu śledzenia problemów lub ostrzegania przed poważnymi błędami.
- Wykrywanie i śledzenie informacji pochodzącej od żądającego klienta, takiej jak odwołania do serwera i kody agenta użytkownika.

Interfejs API komponentu Buforujący serwer proxy ma następujące zalety:

- **Efektywność**
  - Ten interfejs API został opracowany specjalnie dla systemu przetwarzania wątkowego używanego przez komponent Buforujący serwer proxy.
- **Elastyczność**
  - Interfejs API zawiera bogaty zestaw wszechstronnych funkcji.
  - Jego działanie nie jest zależne od platformy ani od języka. Można go uruchomić na wszystkich platformach komponentu Buforujący serwer proxy, a aplikacje wtyczek mogą być pisane w większości języków programowania obsługiwanych przez te platformy.
- **Łatwość użycia**
  - Proste typy danych są przekazywane przez referencję, a nie przez wartość (na przykład long *, char *).
  - Każda funkcja ma stałą liczbę parametrów.
  - Dołączone są również powiązania języka C.
  - Wtyczki nie mają wpływu na przydzielaną pamięć. Aplikacje wtyczek przydzielają i zwalniają pamięć niezależnie od innych procesów komponentu Buforujący serwer proxy.

Ogólna procedura pisania programów interfejsu API

Przed rozpoczęciem pisania programów wtyczek komponentu Buforujący serwer proxy należy zrozumieć zasadę działania serwera proxy. Zachowanie serwera proxy można podzielić na kilka osobnych kroków przetwarzania. Za pomocą tego interfejsu API można dostarczać własne, dostosowane funkcje dla każdego z tych kroków. Można na przykład

dodać funkcję wykonywaną po odczytaniu żądania klienta, ale przed wykonaniem dalszego przetwarzania. Inną możliwością jest przeprowadzenie specjalnych procedur podczas uwierzytelniania, a także po wysłaniu żądanego pliku.

Wraz z interfejsem API udostępniana jest biblioteka predefiniowanych funkcji. Programy wtyczek mogą wywoływać predefiniowane funkcje API w celu nawiązania interakcji z procesem serwera proxy (na przykład aby operować na żądaniach, odczytywać lub zapisywać nagłówki żądań lub zapisywać dane w dziennikach serwera proxy). Nie należy mylić tych funkcji z funkcjami pisanej wtyczki, które są wywoływane przez serwer proxy. Te predefiniowane funkcje opisano w sekcji “Predefiniowane funkcje i makra” na stronie 16.

Aby serwer proxy wywoływał funkcje wtyczki w odpowiednich krokach, należy zawrzeć odpowiednie dyrektywy interfejsu API komponentu Buforujący serwer proxy w pliku konfiguracyjnym serwera. Dyrektywy te opisano w sekcji “Dyrektywy konfiguracyjne komponentu Buforujący serwer proxy dla kroków interfejsu API” na stronie 22.

W tym dokumencie zawarto:

- Podstawowe wyjaśnienia dotyczące kroków związanych z komponentem Buforujący serwer proxy, które można dostosowywać (sekcja “Kroki przetwarzania na serwerze”)
- Wytyczne dotyczące pisania wtyczek (sekcja “Wytyczne” na stronie 7)
- Prototypy dostosowanych funkcji dla każdego kroku wykonywanego przez serwer oraz ich kody powrotu (sekcja “Prototypy funkcji wtyczki” na stronie 9)
- Definicje predefiniowanych funkcji i makr, które można wywołać z poziomu wtyczek, oraz ich kody powrotu (sekcja “Predefiniowane funkcje i makra” na stronie 16)
- Dyrektywy konfiguracyjne interfejsu API komponentu Buforujący serwer proxy (sekcja “Dyrektywy konfiguracyjne komponentu Buforujący serwer proxy dla kroków interfejsu API” na stronie 22)

Używając tych komponentów i procedur, można pisać własne programy wtyczek komponentu Buforujący serwer proxy.

Kroki przetwarzania na serwerze

Podstawowe działanie serwera proxy można podzielić na kroki na podstawie typu przetwarzania, które serwer wykonuje w czasie danej fazy. Każdy krok zawiera punkty, w których może być uruchomiona określona część programu. Dodając dyrektywy interfejsu API do pliku konfiguracyjnego komponentu Buforujący serwer proxy (ibmproxy.conf) można wskazać, która z funkcji wtyczki ma zostać wywołana podczas danego kroku. Można wywołać kilka funkcji wtyczki podczas danego kroku przetwarzania, dodając dla niego kilka dyrektyw.

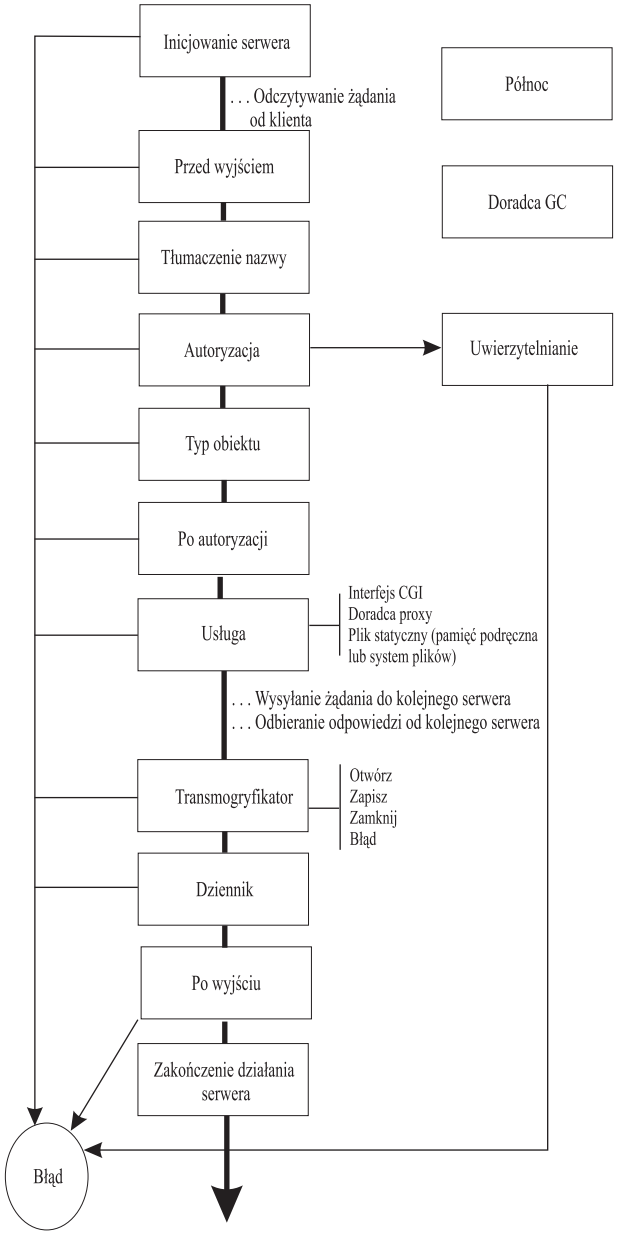
Niektóre kroki są częścią procesu przetwarzania żądań przez serwer. Innymi słowy, serwer proxy wykonuje te kroki za każdym razem, gdy przetwarza żądanie. Inne kroki są wykonywane niezależnie od przetwarzania żądań, to znaczy serwer wykonuje je niezależnie od tego, czy przetwarzane jest żądanie.

Skompilowany program rezyduje w obiekcie współużytkowanym, na przykład w pliku DLL lub .so, w zależności od systemu operacyjnego. Wykonując kolejne kroki przetwarzania żądania, serwer wywołuje funkcje wtyczki powiązane z poszczególnymi krokami do momentu, gdy jedna z funkcji wskaże, że żądanie zostało obsłużone. Jeśli dla danego kroku zostanie określona więcej niż jedna funkcja wtyczki, będą one wywoływane w takim porządku, w jakim ich dyrektywy występują w pliku konfiguracyjnym.

Jeśli żądanie nie zostanie obsłużone przez funkcję wtyczki (dla danego kroku nie została dodana dyrektywa interfejsu API komponentu Buforujący serwer proxy lub funkcja wtyczki zwróciła dla tego kroku wartość HTTP_NOACTION), serwer wykona akcję domyślną.

Uwaga: Dzieje się tak dla wszystkich kroków poza krokiem Usługa. Ten krok nie ma akcji domyślnej.

Rys. 1 ilustruje kroki wykonywane w procesie serwera proxy oraz porządek wykonywania kroków związanych z przetwarzaniem żądań.



Rysunek 1. Schemat blokowy kroków w procesie serwera proxy

Cztery z kroków przedstawionych na diagramie są wykonywane niezależnie od przetwarzania żądań klienta. Są one związane z działaniem i konserwacją serwera proxy. Te kroki to:

- Inicjowanie serwera
- Północ

- Doradca GC
- Zakończenie działania serwera

Na poniższej liście wyjaśniono cel każdego z kroków przedstawionych na Rys. 1 na stronie 5. Należy pamiętać, że nie wszystkie kroki muszą być wywoływane dla określonego żądania.

Inicjowanie serwera
Przeprowadza inicjowanie podczas uruchamiania serwera proxy, przed zaakceptowaniem jakichkolwiek żądań klientów.

Północ Uruchamia wtyczkę o północy (bez kontekstu żądania). Ten krok pokazano w diagramie osobno, ponieważ nie jest on częścią procesu przetwarzania żądań. Innymi słowy, jego wykonywanie jest niezależne od jakichkolwiek żądań.

Doradca GC
Wpływa na decyzje o czyszczeniu pamięci dotyczące plików w pamięci podręcznej. Ten krok pokazano w diagramie osobno, ponieważ nie jest on częścią procesu przetwarzania żądań. Innymi słowy, jego wykonywanie jest niezależne od jakichkolwiek żądań. Czyszczenie pamięci jest wykonywane, gdy wielkość pamięci podręcznej osiągnie wartość maksymalną. (Więcej informacji dotyczących czyszczenia pamięci podręcznej zawiera *Podręcznik administrowania komponentem Buforujący serwer proxy produktu WebSphere Application Server*).

Przed wyjściem

Przeprowadza przetwarzanie po odczytaniu żądania, lecz przed wykonaniem jakichkolwiek innych czynności.

Jeśli krok zwróci informacje o tym, że żądanie zostało przetworzone (HTTP_OK), serwer pominie inne kroki przetwarzania żądania i wykona tylko kroki Transmogryfikator, Dziennik oraz Po wyjściu.

Tłumaczenie nazwy
Tłumaczy ścieżkę wirtualną (z adresu URL) na ścieżkę fizyczną.

Autoryzacja

Korzysta z przechowywanych elementów zabezpieczeń, aby sprawdzić ścieżkę fizyczną pod kątem wszelkich zabezpieczeń, list ACL oraz innych elementów kontroli dostępu, a także generuje nagłówki WWW-Authenticate, które są wymagane do podstawowego uwierzytelniania. Jeśli krok ten ma zostać zastąpiony funkcją wtyczki utworzoną przez użytkownika, podczas jej pisania należy pamiętać o konieczności wygenerowania tych nagłówków.

Więcej informacji można znaleźć w sekcji “Uwierzytelnianie i autoryzacja” na stronie 33.

Uwierzytelnianie

Dekoduje, weryfikuje i zapisuje elementy zabezpieczeń.

Więcej informacji można znaleźć w sekcji “Uwierzytelnianie i autoryzacja” na stronie 33.

Typ obiektu
Znajduje wskazywany przez ścieżkę obiekt systemu plików.

Po autoryzacji

Przeprowadza przetwarzanie po autoryzacji i znalezieniu obiektu, lecz przed spełnieniem żądania.

Jeśli krok zwróci informacje o tym, że żądanie zostało przetworzone (HTTP_OK), serwer pominie inne kroki przetwarzania żądania i wykona tylko kroki Transmogryfikator, Dziennik oraz Po wyjściu.

Usługa Spełnia żądanie (wysyłając plik, uruchamiając interfejs CGI itp.)

Doradca proxy
Wpływa na decyzje dotyczące serwera proxy oraz buforowania.

Transmogryfikator
Przyznaje uprawnienia do zapisu dotyczące danych w odpowiedzi wysłanej do klienta.

Dziennik
Umożliwia dostosowane rejestrowanie transakcji.

Błąd Umożliwia utworzenie dostosowanych odpowiedzi na warunki błędów.

Po wyjściu
Czyści zasoby przydzielone na potrzeby przetwarzania żądań.

Zakończenie działania serwera
Wykonuje procedurę czyszczącą podczas poprawnego zamknięcia systemu.

Wytyczne

- Podczas pisania programu korzystaj ze składni oraz wytycznych udostępnionych dla funkcji wtyczki serwera. Każdej funkcji wtyczki nadaj unikalną nazwę. W razie potrzeby wywołaj predefiniowane funkcje serwera.

  W systemach AIX wymagany jest plik eksportu (na przykład libmyapp.exp), który zawiera wszystkie funkcje wtyczki. Podczas konsolidacji należy także uwzględnić plik importu interfejsu API komponentu Buforujący serwer proxy - libhttpdapi.exp.

  W systemach Linux, HP-UX oraz Solaris należy podczas konsolidacji uwzględnić biblioteki libhttpdapi oraz libc.

  W systemach Windows wymagany jest plik definicji modułów (.def), który zawiera informacje o wszystkich funkcjach wtyczki. Podczas konsolidacji należy uwzględnić plik HTTPDAPI.LIB.

  W definicjach funkcji należy uwzględnić plik nagłówkowy HTAPI.h oraz użyć makra HTTPD_LINKAGE. Makro to gwarantuje, że wszystkie funkcje będą używały tych samych konwencji wywoływań.
- Serwer jest uruchamiany w środowisku wielowątkowym, a zatem wtyczki muszą być wątkowo bezpieczne. Jeśli aplikacja jest wielobieżna, nie nastąpi spadek wydajności.
- Utrzymuj akcje wykonywane we wtyczkach w zasięgu wątku. Nie wykonuj żadnych akcji w zasięgu procesu (takich jak na przykład wychodzenie, zmiana ID użytkownika lub rejestrowanie procedury obsługi sygnału).
- Nie używaj zmiennych globalnych lub, jeśli ich użycie jest konieczne, chroń je za pomocą semafora wzajemnego wykluczania.
- Pamiętaj o ustawieniu nagłówka Content-Type, jeśli do wysłania danych z powrotem do klienta używana jest funkcja HTTPD_write().
- Zawsze sprawdzaj kody powrotu i używaj przetwarzania warunkowego tam, gdzie jest to konieczne.
- Skompiluj i skonsoliduj program, odwołując się do dokumentacji kompilatora w celu zbudowania obiektu współużytkowanego (na przykład pliku DLL lub .so) wymaganego przez system operacyjny.

  Poniższe komendy kompilowania i konsolidowania należy traktować jako wytyczne.
  - **System AIX**, kompilator IBM CSet++
    - **Kompilowanie:**

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

- **Konsolidowanie:**

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

(Tę komendę zaprezentowano w dwóch wierszach jedynie z uwagi na czytelność).

– **System HP-UX**, pakunek HP C/ANSI C Developer's Bundle i kompilator HP aC++

- **Kompilowanie:**

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

- **Konsolidowanie:**

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

– **System Linux**, kompilator Gnu Compiler C (GCC) wersja 3.2.X

- **Kompilowanie:**

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

- **Konsolidowanie:**

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

– **System Solaris**, kompilator Sun Workshop

- **Kompilowanie:**

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

- **Konsolidowanie:**

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

– **System Windows**, kompilator Microsoft Visual C++

- **Kompilowanie:**

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

- **Konsolidowanie:**

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

Aby określić elementy do wyeksportowania, użyj jednej z poniższych metod:

– Dodaj definicje _declspec(dllexport) w kodzie źródłowym.

– Określ zmienną /EXPORT:entryname w komendzie LIB.

– Utwórz plik definicji modułu z instrukcją EXPORTS.

- Dodaj dyrektywy interfejsu API komponentu Buforujący serwer proxy do pliku konfiguracyjnego, aby powiązać funkcje wtyczki z odpowiednimi krokami. Dla każdego kroku w procesie przetwarzania żądań serwera istnieje osobna dyrektywa. Zatrzymaj i zrestartuj serwer w celu uwzględnienia nowych dyrektyw.

  Uwaga: Komponent Buforujący serwer proxy nie usuwa z pamięci obiektów współużytkowanych (plików DLL lub .so) nawet podczas restartowania. Należy zatrzymać, a następnie uruchomić serwer, aby zwolnić obiekty współużytkowane.

- Dokładnie przetestuj program przed użyciem go w środowisku produkcyjnym. Ponieważ komponent Buforujący serwer proxy to serwer wątkowy, należy przeprowadzić dokładniejsze testy niż dla serwera rozwidlającego. Błędy w programie mogą spowodować awarię serwera proxy, ponieważ serwer proxy wywołuje ten program bezpośrednio, a zarówno serwer, jak i program są uruchamiane w tym samym obszarze procesu.

Funkcje wtyczki

Przy pisaniu funkcji programu dla określonych kroków przetwarzania żądań należy użyć składni zaprezentowanej w sekcji "Prototypy funkcji wtyczki" na stronie 9.

Każda z funkcji musi wypełnić parametr kodu powrotu wartością, która wskazuje jaka akcja została wykonana:

- Kod HTTP_NOACTION (wartość 0) oznacza, że nie została wykonana żadna istotna akcja. Jeśli zostanie zwrócony ten kod, serwer proxy wykonuje akcję domyślną dla tego kroku.
- Jeden z poprawnych kodów powrotu HTTP wskazuje, że funkcja wtyczki obsłużyła krok. (Listę poprawnych kodów powrotu można znaleźć w sekcji “Kody powrotu HTTP i ich wartości” na stronie 15). Po otrzymaniu poprawnego kodu powrotu HTTP nie są wywoływane żadne inne funkcje wtyczki służące do obsłużenia danego kroku żądania.

Prototypy funkcji wtyczki

W prototypach funkcji dla każdego kroku komponentu Buforujący serwer proxy pokazano format, którego należy użyć, oraz wyjaśniono typ przetwarzania, które można za ich pomocą wykonać. Należy pamiętać, że nazwy funkcji nie są predefiniowane. Funkcjom należy nadać unikalne nazwy. Można stosować własne konwencje nazewnictwa. W tym dokumencie dla ułatwienia użyto nazw odpowiadających krokom przetwarzania serwera.

W każdej funkcji wtyczki można używać określonych predefiniowanych funkcji API. Niektóre predefiniowane funkcje nie są poprawne dla wszystkich kroków. Poniższe predefiniowane funkcje API mogą być poprawnie wywołane z poziomu wszystkich funkcji wtyczki:

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- funkcje HTTPD_log*

Dodatkowe poprawne i niepoprawne funkcje API zostały podane w opisach prototypów funkcji.

Wartość parametru *handle* wysłanego do funkcji można przekazać do predefiniowanej funkcji jako pierwszy argument. Predefiniowane funkcje API opisano w sekcji “Predefiniowane funkcje i makra” na stronie 16.

Inicjowanie serwera

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

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

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

The version parameters contain the serwer proxy’s version number; these are supplied by the Buforujący serwer proxy.

PreExit

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

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

Valid return codes for the preExit function are the following:

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

Other return codes must not be used.

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

All predefined API functions are valid during this step.

Midnight

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

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

Authentication

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

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

Tłumaczenie nazwy

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań zgodnych z szablonem. Krok Tłumaczenie nazwy jest wykonywany przed przetworzeniem żądania i udostępnia mechanizm odwzorowania adresów URL na obiekty, takie jak nazwy plików.

Autoryzacja

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań zgodnych z szablonem. Krok Autoryzacja jest wykonywany przed przetworzeniem żądania i można go użyć w celu sprawdzenia, czy możliwe jest zwrócenie zidentyfikowanego obiektu do klienta. W przypadku przeprowadzania podstawowego uwierzytelniania należy wygenerować wymagane nagłówki WWW-Authenticate.

Typ obiektu

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań zgodnych z szablonem. Krok Typ obiektu jest wykonywany przed przetworzeniem żądania i można go użyć do sprawdzenia, czy obiekt istnieje, a także do określenia typu obiektu.

Po autoryzacji

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

Funkcja zdefiniowana dla tego kroku jest wywoływana po autoryzacji żądania, lecz przed rozpoczęciem przetwarzania. Jeśli ta funkcja zwróci kod HTTP_OK, serwer proxy przyjmuje, że żądanie zostało obsłużone. Wszystkie kolejne kroki zostaną pominięte z wyjątkiem kroków odpowiedzi (Transmogryfikator, Dziennik i Po wyjściu).

Wszystkie predefiniowane funkcje serwera są poprawne w tym kroku.

Usługa

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań zgodnych z szablonem. W kroku Usługa żądanie jest spełniane, jeśli nie zostało ono spełnione w krokach Przed wyjściem lub Po autoryzacji.

Wszystkie predefiniowane funkcje serwera są poprawne w tym kroku.

Informacje na temat konfigurowania funkcji Usługa w taki sposób, aby była wykonywana na podstawie metody HTTP, a nie na podstawie adresu URL, można znaleźć w opisie dyrektywy Enable w publikacji *Podręcznik administrowania komponentem Buforujący serwer proxy produktu WebSphere Application Server*.

Transmogryfikator

Funkcji wywoływanych w tym kroku procesu można używać do filtrowania danych odpowiedzi jako strumienia. W tym kroku kolejno wywoływane są cztery funkcje wtyczki. Każda z nich działa jako segment potoku, którym przepływają dane.

Oznacza to, że udostępnione przez użytkownika funkcje *otwórz*, *zapisz*, *zamknij* oraz *błąd* są wywoływane dla każdego żądania (w tym porządku). Każda funkcja kolejno przetwarza ten sam strumień danych.

Dla tego kroku należy zaimplementować następujące cztery funkcje. (Nazwy funkcji nie muszą być zgodne z podanymi).

- **Otwórz**

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

  Funkcja otwierania przeprowadza inicjowanie (na przykład przydzielanie buforu) wymagane do przetworzenia danych z danego strumienia. Każdy kod powrotu z wyjątkiem HTTP_OK powoduje przerwanie działania filtru (funkcje zapisu i zamykania nie są wywoływane). Funkcja może zwrócić pusty wskaźnik, umożliwiając przydzielenie obszaru dla struktury i przekazanie wskaźnika z powrotem w parametrze *correlator* kolejnych funkcji.

- **Zapisz**

```
void  HTTPD_LINKAGE  writeFunction (
         unsigned char *handle,
         unsigned char *data,    /* dane odpowiedzi wysłane przez
                                    serwer źródłowy */
         unsigned long *length,  /* długość danych odpowiedzi */
         void *correlator,       /* wskaźnik zwrócony przez
                                    funkcję otwierania */
         long *return_code
         )
```

  Funkcja zapisu przetwarza dane i może wywoływać predefiniowaną funkcję serwera HTTPD_write() z nowymi lub zmienionymi danymi. Wtyczka nie może próbować zwolnić buforu przekazanego do niej, ani oczekiwać, że serwer zwolni otrzymany bufor.

  Nawet jeśli dane nie zostaną zmienione podczas wykonywania funkcji zapisu, należy wywołać funkcję HTTPD_write() w ramach funkcji otwierania, zapisu lub zamykania w celu przekazania danych odpowiedzi do klienta. Argument *correlator* to wskaźnik buforu danych, który został zwrócony w procedurze *otwierania*.

- **Zamknij**

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

  Funkcja zamykania przeprowadza akcje procedury czyszczącej (na przykład opróżnianie lub zwalnianie buforu korelatora) wymagane do zakończenia przetwarzania danych strumienia. Argument *correlator* to wskaźnik buforu danych, który został zwrócony w procedurze *otwierania*.

- **Błąd**

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

  Funkcja błędu umożliwia przeprowadzenie akcji procedury czyszczącej, takich jak opróżnianie lub zwalnianie buforowanych danych (lub obu) przed wysłaniem strony błędu. W tym miejscu wywoływane są funkcje otwierania, zapisu i zamykania w celu przetworzenia strony błędu. Argument *correlator* to wskaźnik buforu danych, który został zwrócony w procedurze *otwierania*.

Uwagi:

- Pisząc wtyczkę dla kroku Transmogryfikator, należy wywołać funkcje HTTPD_open(), HTTPD_write() oraz HTTPD_close() w zasięgu funkcji otwierania, zapisu i zamykania. Funkcję HTTPD_write() można wywołać dopiero po wywołaniu funkcji HTTPD_open(). Zadaniem tych predefiniowanych funkcji jest przekazanie sterowania serwerowi, aby możliwe było wywołanie kolejnej funkcji.
- Wywołanie funkcji HTTPD_* jest konieczne do poprawnego działania kroku Transmogryfikator oraz serwera. Na przykład jeśli funkcje HTTPD_open() i HTTPD_close() nie zostaną wywołane, nagłówki nie zostaną zwrócone do klienta.
- Należy pamiętać, że mogą wystąpić niepożądane efekty, jeśli aplikacje filtrowania danych nie będą odpowiednio wybiórcze. Istnieje możliwość, że jeśli filtrowanie strumieni danych nie zostanie poprawnie skonfigurowane, interfejsy CGI nie będą działać, pliki GIF nie będą wyświetlane, a inne strumienie binarne nie będą zachowywać się w oczekiwany sposób.
- Wtyczka nie musi buforować treści. Komponent Buforujący serwer proxy automatycznie określa długość treści.
- Funkcję HTTPD_open() można wywołać, gdy sterowanie nagłówkami ma zostać przekazane serwerowi. Jednak jeśli konieczne będzie ustawienie nagłówka w dalszej części programu interfejsu API, funkcję HTTPD_open() można wywołać dopiero po wywołaniu funkcji zapisu lub zamykania.

  Uwaga: Przed wywołaniem funkcji HTTPD_open() należy ustawić nagłówki za pomocą funkcji HTTPD_set() lub httpd_setvar().
- Strumień danych nie zawiera nagłówków. Wtyczki muszą używać funkcji ustawiania i wyodrębniania, aby manipulować nagłówkami. Funkcja *otwierania* wtyczki nie jest wywoływana do momentu odczytania wszystkich nagłówków.
- Można używać wielu wtyczek transmogryfikatora. Są one wywoływane w porządku, w którym występują w pliku konfiguracyjnym.
- Tranzyt tunelowania SSL nie odbywa się przez wtyczki transmogryfikatora.

Doradca GC

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego pliku znajdującego się w pamięci podręcznej podczas czyszczenia pamięci. Za pomocą tej funkcji można wpływać na to, które pliki są zachowywane, a które są usuwane. Więcej informacji można znaleźć, przeglądając zmienne GC_*.

Doradca proxy

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

Funkcja zdefiniowana dla tego kroku jest wywoływana podczas obsługiwania każdego żądania proxy. Można jej użyć na przykład do ustawienia zmiennej USE_PROXY.

Dziennik

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania po jego przetworzeniu i zakończeniu komunikacji z klientem. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań zgodnych z szablonem. Ta funkcja jest wywoływana niezależnie od tego, czy przetwarzanie żądania zakończyło się powodzeniem. Jeśli wtyczka dziennika nie ma przesłaniać domyślnego mechanizmu dziennika, należy ustawić kod powrotu HTTP_NOACTION zamiast kodu HTTP_OK.

Błąd

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania, które się nie powiodło. Można określić szablon URL w dyrektywie pliku konfiguracyjnego, jeśli funkcja wtyczki ma być wywoływana tylko dla żądań, zgodnych z szablonem, które się nie powiodły. Krok Błąd umożliwia dostosowanie odpowiedzi dotyczącej błędu.

Po wyjściu

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

Funkcja zdefiniowana dla tego kroku jest wywoływana dla każdego żądania niezależnie od tego, czy zakończyło się ono powodzeniem. Ten krok umożliwia wykonanie czynności procedury czyszczącej dla zasobów przydzielonych przez wtyczkę na potrzeby przetworzenia żądania.

Zakończenie działania serwera

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

Funkcja zdefiniowana dla tego kroku jest wywoływana przy poprawnym zamknięciu systemu serwera. Umożliwia przeprowadzenie procedury czyszczącej dla zasobów przydzielonych podczas kroku Inicjowanie serwera. Nie należy wywoływać żadnych funkcji HTTP_* w tym kroku (rezultaty są nieprzewidywalne). Jeśli w pliku konfiguracyjnym dla kroku Zakończenie działania serwera znajduje się więcej niż jedna dyrektywa interfejsu API komponentu Buforujący serwer proxy, zostanie wywołana każda z nich.

Uwaga: Z powodu ograniczenia w kodzie systemu Solaris krok wtyczki Zakończenie działania serwera nie jest wykonywany, jeśli do zakończenia działania komponentu Buforujący serwer proxy na platformach Solaris użyta zostanie komenda **ibmproxy -stop**. Więcej informacji na temat uruchamiania i zatrzymywania komponentu Buforujący serwer proxy zawiera *Podręcznik administrowania komponentem Buforujący serwer proxy produktu WebSphere Application Server*.

Kody powrotu HTTP i ich wartości

Poniższe kody powrotu są zgodne ze specyfikacją HTTP 1.1, RFC 2616, opublikowaną przez organizację World Wide Web Consortium (www.w3.org/pub/WWW/Protocols/). Funkcje wtyczki muszą zwracać jedną z podanych wartości.

Tabela 2. Kody powrotu HTTP dla funkcji API komponentu Buforujący serwer proxy

Wartość	Kod powrotu
0	HTTP_NOACTION
100	HTTP_CONTINUE
101	HTTP_SWITCHING_PROTOCOLS
200	HTTP_OK
201	HTTP_CREATED
202	HTTP_ACCEPTED
203	HTTP_NON_AUTHORITATIVE
204	HTTP_NO_CONTENT
205	HTTP_RESET_CONTENT
206	HTTP_PARTIAL_CONTENT
300	HTTP_MULTIPLE_CHOICES
301	HTTP_MOVED_PERMANENTLY
302	HTTP_MOVED_TEMPORARILY
302	HTTP_FOUND
303	HTTP_SEE_OTHER
304	HTTP_NOT_MODIFIED
305	HTTP_USE_PROXY
307	HTTP_TEMPORARY_REDIRECT
400	HTTP_BAD_REQUEST
401	HTTP_UNAUTHORIZED
403	HTTP_FORBIDDEN
404	HTTP_NOT_FOUND
405	HTTP_METHOD_NOT_ALLOWED
406	HTTP_NOT_ACCEPTABLE
407	HTTP_PROXY_UNAUTHORIZED
408	HTTP_REQUEST_TIMEOUT
409	HTTP_CONFLICT
410	HTTP_GONE
411	HTTP_LENGTH_REQUIRED
412	HTTP_PRECONDITION_FAILED
413	HTTP_ENTITY_TOO_LARGE
414	HTTP_URI_TOO_LONG
415	HTTP_BAD_MEDIA_TYPE
416	HTTP_BAD_RANGE
417	HTTP_EXPECTATION_FAILED
500	HTTP_SERVER_ERROR

Tabela 2. Kody powrotu HTTP dla funkcji API komponentu Buforujący serwer proxy (kontynuacja)

501	HTTP_NOT_IMPLEMENTED
502	HTTP_BAD_GATEWAY
503	HTTP_SERVICE_UNAVAILABLE
504	HTTP_GATEWAY_TIMEOUT
505	HTTP_BAD_VERSION

Predefiniowane funkcje i makra

Predefiniowane funkcje i makra serwera można wywoływać z poziomu własnych funkcji wtyczki. Należy używać ich predefiniowanych nazw oraz formatu opisanego poniżej. W opisach parametrów litery we wskazują parametr wejścia, litery wy parametr wyjścia, natomiast litery we/wy wskazują, że parametr jest zarówno parametrem wejścia, jak i wyjścia.

Każda z tych funkcji zwraca jeden z kodów powrotu HTTPD w zależności od tego, czy żądanie zakończy się powodzeniem. Kody te opisano w sekcji “Kody powrotu z predefiniowanych funkcji i makr” na stronie 21.

Podczas wywoływania tych funkcji jako pierwszego parametru należy użyć uchwytu wtyczki. W innym przypadku funkcja zwróci kod błędu HTTPD_PARAMETER_ERROR. Wartość NULL nie jest akceptowana jako poprawny uchwyt.

HTTPD_authenticate()

Uwierzytelnia ID użytkownika lub hasło albo zarówno ID użytkownika, jak i hasło. Poprawna tylko dla kroków Przed wyjściem, Uwierzytelnianie, Autoryzacja i Po autoryzacji.

```
void  HTTPD_LINKAGE  HTTPD_authenticate (
         unsigned char *handle,      /* we; uchwyt */
         long *return_code           /* wy; kod powrotu */
         )
```

HTTPD_cacheable_url()

Sprawdza, czy można buforować określoną treść URL zgodnie ze standardami komponentu Buforujący serwer proxy.

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
        unsigned char *handle,      /* we; uchwyt */
        unsigned char *url,         /* we; adres URL do sprawdzenia */
        unsigned char *req_method,  /* we; metoda żądania dla adresu URL */
        long *retval                /* wy; kod powrotu */
        )
```

Zwrócona wartość HTTPD_SUCCESS wskazuje, że można buforować treść URL, natomiast wartość HTTPD_FAILURE wskazuje, że nie jest to możliwe. Dla tej funkcji możliwy jest także kod powrotu HTTPD_INTERNAL_ERROR.

HTTPD_close()

(Poprawna tylko w kroku Transmogryfikator). Przekazuje sterowanie kolejnej procedurze *zamykania* na stosie strumienia. Należy ją wywołać z funkcji otwierania, zapisu lub zamykania kroku Transmogryfikator po zakończeniu przetwarzania. Ta funkcja informuje serwer proxy, że odpowiedź została przetworzona oraz że krok Transmogryfikator został zakończony.

```
void  HTTPD_LINKAGE  HTTPD_close (
         unsigned char *handle,       /* we; uchwyt */
         long *return_code            /* wy; kod powrotu */
         )
```

HTTPD_exec()

Wywołuje skrypt w celu spełnienia tego żądania. Poprawna w krokach Przed wyjściem, Usługa, Po autoryzacji oraz Błąd.

```
void  HTTPD_LINKAGE  HTTPD_exec (
       unsigned char *handle,       /* we; uchwyt */
       unsigned char *name,         /* we; nazwa skryptu do uruchomienia */
       unsigned long *name_length,  /* we; długość nazwy */
       long *return_code            /* wy; kod powrotu */
         )
```

HTTPD_extract()

Wyodrębnia wartość zmiennej powiązanej z tym żądaniem. Poprawne zmienne dla parametru *name* są takie same, jak te używane w interfejsie CGI. Więcej informacji można znaleźć w sekcji “Zmienne” na stronie 25. Mimo że ta funkcja jest poprawna we wszystkich krokach, to niektóre zmienne mogą nie być poprawne w niektórych krokach.

```
void  HTTPD_LINKAGE  HTTPD_extract (
      unsigned char *handle,       /* we; uchwyt */
      unsigned char *name,         /* we; nazwa zmiennej do
                                        wyodrębnienia */
      unsigned long *name_length,  /* we; długość nazwy */
      unsigned char *value,        /* wy; bufor, w którym należy umieścić
                                        wartość */
      unsigned long *value_length, /* we/wy; wielkość buforu */
      long *return_code            /* wy; kod powrotu */
      )
```

Zwrócenie przez tę funkcję kodu HTTPD_BUFFER_TOO_SMALL oznacza, że żądana wielkość buforu nie była wystarczająca dla wyodrębnionej wartości. W takim przypadku funkcja nie użyje tego buforu, lecz nada parametrowi value_length wartość wielkości buforu, która jest konieczna do pomyślnego wyodrębnienia tej wartości. Należy ponowić ekstrakcję, używając buforu, który jest nie mniejszy niż zwrócona wartość value_length.

Uwaga: Jeśli wyodrębniana zmienna dotyczy nagłówka HTTP, funkcja HTTPD_extract() wyodrębni tylko pierwsze zgodne wystąpienie, nawet jeśli żądanie zawiera wiele nagłówków o tej samej nazwie. Zamiast funkcji HTTPD_extract() można użyć funkcji httpd_getvar(), która ma również inne zalety. Więcej informacji można znaleźć w sekcji 17.

HTTPD_file()

Wysyła plik w celu spełnienia żądania. Poprawna tylko w krokach Przed wyjściem, Usługa, Błąd, Po autoryzacji oraz Transmogryfikator.

```
void  HTTPD_LINKAGE  HTTPD_file (
         unsigned char *handle,          /* we; uchwyt */
         unsigned char *name,            /* we; nazwa pliku do wysłania */
         unsigned long *name_length,     /* we; długość nazwy */
         long *return_code               /* wy; kod powrotu */
         )
```

httpd_getvar()

Funkcja podobna do funkcji HTTPD_extract(), lecz łatwiejsza w użyciu, ponieważ użytkownik nie musi określać długości argumentów.

```
const  unsigned char *          /* wy; wartość zmiennej */
HTTPD_LINKAGE
httpd_getvar(
   unsigned char *handle,       /* we; uchwyt */
   unsigned char *name,         /* we; nazwa zmiennej */
   unsigned long *n             /* we; liczba indeksu tablicy
                                       zawierającej nagłówek */
   )
```

Indeks tablicy zawierającej nagłówek zaczyna się od cyfry 0. Aby pobrać pierwszy element tablicy, należy użyć w parametrze *n* wartości 0, natomiast aby uzyskać piąty element, należy użyć wartości 4.

Uwaga: Nie należy usuwać ani zmieniać treści zwróconej wartości. Zwrócony łańcuch jest zakończony znakiem o kodzie zero.

HTTPD_log_access()

Zapisuje łańcuch w dzienniku dostępu serwera.

```
void  HTTPD_LINKAGE  HTTPD_log_access (
         unsigned char *handle,           /* we; uchwyt */
         unsigned char *value,            /* we; dane do zapisania */
         unsigned long *value_length,     /* we; długość danych */
         long *return_code                /* wy; kod powrotu */
         )
```

Należy pamiętać, że symbole zmiany znaczenia *nie* są wymagane przy zapisywaniu symbolu procentów (%) w dziennikach dostępu serwera.

HTTPD_log_error()

Zapisuje łańcuch w dzienniku błędów serwera.

```
void  HTTPD_LINKAGE  HTTPD_log_error (
         unsigned char *handle,          /* we; uchwyt */
         unsigned char *value,           /* we; dane do zapisania */
         unsigned long *value_length,    /* we; długość danych */
         long *return_code               /* wy; kod powrotu */
         )
```

Należy pamiętać, że symbole zmiany znaczenia *nie* są wymagane przy zapisywaniu symbolu procentów (%) w dziennikach błędów serwera.

HTTPD_log_event()

Zapisuje łańcuch w dzienniku zdarzeń serwera.

```
void  HTTPD_LINKAGE  HTTPD_log_event (
         unsigned char *handle,          /* we; uchwyt */
         unsigned char *value,           /* we; dane do zapisania */
         unsigned long *value_length,    /* we; długość danych */
         long *return_code               /* wy; kod powrotu */
         )
```

Należy pamiętać, że symbole zmiany znaczenia *nie* są wymagane przy zapisywaniu symbolu procentów (%) w dziennikach zdarzeń serwera.

HTTPD_log_trace()

Zapisuje łańcuch w dzienniku śledzenia serwera.

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
         unsigned char *handle,          /* we; uchwyt */
         unsigned char *value,           /* we; dane do zapisania */
         unsigned long *value_length,    /* we; długość danych */
         long *return_code               /* wy; kod powrotu */
         )
```

Należy pamiętać, że symbole zmiany znaczenia *nie* są wymagane przy zapisywaniu symbolu procentów (%) w dziennikach śledzenia serwera.

HTTPD_open()

(Poprawna tylko w kroku Transmogryfikator). Przekazuje sterowanie kolejnej procedurze na stosie strumienia. Należy ją wywołać z poziomu funkcji otwierania, zapisu lub zamykania kroku Transmogryfikator po ustawieniu potrzebnych nagłówków, gdy można rozpocząć procedurę zapisu.

```
void  HTTPD_LINKAGE  HTTPD_open (
         unsigned char *handle,      /* we; uchwyt */
         long *return_code           /* wy; kod powrotu */
         )
```

HTTPD_proxy()

Tworzy żądanie proxy. Poprawna w krokach Przed wyjściem, Usługa oraz Po autoryzacji.

Uwaga: Jest to funkcja kończąca. Po jej wykonaniu żądanie jest zakończone.

```
void  HTTPD_LINKAGE  HTTPD_proxy (
     unsigned char *handle,        /* we; uchwyt */
     unsigned char *url_name,      /* we; adres URL dla
                                          żądania proxy */
     unsigned long *name_length,   /* we; długość adresu URL */
     void *request_body,           /* we; treść żądania */
     unsigned long *body_length,   /* we; długość treści */
     long *return_code             /* wy; kod powrotu */
     )
```

HTTPD_read()

Odczytuje treść żądania klienta. Dla nagłówków należy użyć funkcji HTTPD_extract(). Poprawna tylko w krokach Przed wyjściem, Autoryzacja, Po autoryzacji oraz Usługa. Użyteczna tylko w przypadku żądań PUT lub POST. Tę funkcję należy wywoływać w pętli do momentu zwrócenia kodu HTTPD_EOF. Jeśli dane żądanie nie zawiera treści, funkcja nie powiedzie się.

```
void  HTTPD_LINKAGE  HTTPD_read (
         unsigned char *handle,          /* we; uchwyt */
         unsigned char *value,           /* we; bufor danych */
         unsigned long *value_length,    /* we/wy; wielkość buforu
                                                (długość danych) */
         long *return_code               /* wy; kod powrotu */
         )
```

HTTPD_restart()

Restartuje serwer po przetworzeniu wszystkich aktywnych żądań. Poprawna we wszystkich krokach z wyjątkiem Inicjowanie serwera, Zakończenie działania serwera oraz Transmogryfikator.

```
void  HTTPD_LINKAGE  HTTPD_restart (
         long *return_code    /* wy; kod powrotu */
         )
```

HTTPD_set()

Ustawia wartość zmiennej powiązanej z tym żądaniem. Poprawne zmienne dla parametru *name* są takie same, jak te używane w interfejsie CGI. Więcej informacji można znaleźć w sekcji “Zmienne” na stronie 25.

Należy pamiętać, że za pomocą tej funkcji można także tworzyć zmienne. Zmienne te podlegają konwencjom tworzenia przedrostków HTTP_ oraz PROXY_, które zostały opisane w sekcji “Zmienne” na stronie 25. Jeśli zostanie utworzona zmienna, której nazwa rozpoczyna się od przedrostka HTTP_, będzie ona wysłana jako nagłówek w odpowiedzi do klienta bez przedrostka HTTP_. Na przykład aby ustawić nagłówek Location, należy użyć funkcji HTTPD_set() z nazwą zmiennej HTTP_LOCATION. Zmienne z przedrostkiem PROXY_ są wysyłane jako nagłówki żądania do serwera treści. Zmienne utworzone z przedrostkiem CGI_ są przekazywane do programów CGI.

Mimo że ta funkcja jest poprawna we wszystkich krokach, to niektóre zmienne mogą nie być poprawne w niektórych krokach.

```
void  HTTPD_LINKAGE  HTTPD_set (
       unsigned char *handle,        /* we; uchwyt */
       unsigned char *name,          /* we; nazwa wartości do ustawienia */
```

```
unsigned long *name_length,   /* we; długość nazwy */
unsigned char *value,         /* we; bufor z wartością */
unsigned long *value_length,  /* we; długość wartości */
long *return_code             /* wy; kod powrotu */
)
```

Uwaga: Aby ustawić wartość zmiennej bez konieczności określania buforu i długości, można użyć funkcji httpd_setvar(). Więcej informacji można znaleźć w sekcji 20.

httpd_setvar()

Funkcja podobna do funkcji HTTPD_set(), lecz łatwiejsza w użyciu, ponieważ użytkownik nie musi określać długości argumentów.

```
long             /* wy; kod powrotu */
HTTPD_LINKAGE   httpd_setvar (
       unsigned char *handle,       /* we; uchwyt */
       unsigned char *name,         /* we; nazwa zmiennej */
       unsigned char *value,        /* we; nowa wartość */
       unsigned long *addHdr        /* we; dodawanie lub
                                       zastępowanie nagłówka */
       )
```

Parametr *addHdr* może przyjąć cztery wartości:

- HTTPD_SETVAR_REPLACE — Zastępuje wszystkie wystąpienia zmiennej nagłówka nową wartością.
- HTTPD_SETVAR_REPLACE_ADD — Jeśli zmienna nagłówka istnieje, zastępuje jej pierwsze wystąpienie nową wartością. Jeśli zmienna nie istnieje, dodaje nową wartość do nagłówków.
- HTTPD_SETVAR_ADD — Dodaje tę wartość do nagłówków.
- HTTPD_SETVAR_REMOVE_ALL — Usuwa wszystkie wystąpienia tej zmiennej nagłówka.

Wartości te zostały zdefiniowane w pliku nagłówków HTAPI.h.

httpd_variant_insert()

Wprowadza wariant do pamięci podręcznej.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,    /* we; uchwyt */
         unsigned char *URI,       /* we; identyfikator URI obiektu */
         unsigned char *dimension, /* we; wymiar różnicy */
         unsigned char *variant,   /* we; wartość wariantu */
         unsigned char *filename,  /* we; plik zawierający obiekt */
         long *return_code         /* wy; kod powrotu */
         )
```

Uwagi:

1. Argument dimension odwołuje się do nagłówka, którym dany obiekt różni się od identyfikatora URI. W powyższym przykładzie możliwa wartość argumentu dimension to User-Agent.
2. Argument variant odwołuje się do wartości nagłówka podanego w argumencie dimension. Jest on różny od identyfikatora URI. W powyższym przykładzie możliwa jest następująca wartość argumentu variant:

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

3. Argument filename musi wskazywać kopię nazwy pliku (zakończoną znakiem o kodzie zero), w którym użytkownik zapisał zmienioną treść. Użytkownik jest odpowiedzialny za usunięcie pliku. Ta akcja jest bezpieczna po powrocie z tej funkcji. Plik zawiera tylko treść bez nagłówków.

4. Przy buforowaniu wariantów serwer aktualizuje nagłówek content-length oraz dodaje nagłówek Warning: 214. Znaczniki Strong są usuwane.

httpd_variant_lookup()

Określa, czy dany wariant istnieje w pamięci podręcznej.

```
void  HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,             /* we; uchwyt */
    unsigned char *URI,                /* identyfikator URI obiektu */
    unsigned char *dimension,          /* we; wymiar różnicy */
    unsigned char *variant,            /* we; wartość wariantu */
             long *return_code);       /* wy; kod powrotu */
```

HTTPD_write()

Zapisuje treść odpowiedzi. Poprawna w krokach Przed wyjściem, Usługa, Błąd oraz Transmogryfikator.

Jeśli typ treści nie zostanie ustawiony przed pierwszym wywołaniem tej funkcji, serwer zakłada, że wysyłany jest strumień danych CGI.

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,             /* we; uchwyt */
    unsigned char *value,              /* we; dane do wysłania */
    unsigned char *value_length,       /* we; długość danych */
             long *return_code);       /* wy; kod powrotu */
```

Uwaga: Informacje o nagłówkach odpowiedzi można znaleźć w sekcji 19.

Uwaga: Po powrocie z funkcji HTTPD_* można bezpiecznie zwolnić pamięć używaną przez tę funkcję.

Kody powrotu z predefiniowanych funkcji i makr

Serwer ustawi parametr kodu powrotu na jedną z poniższych wartości w zależności od tego, czy żądanie zakończyło się powodzeniem:

Tabela 3. Kody powrotu

Wartość	Kod statusu	Wyjaśnienie
-1	HTTPD_UNSUPPORTED	Funkcja nie jest obsługiwana.
0	HTTPD_SUCCESS	Funkcja została wykonana pomyślnie i pola wyjściowe są poprawne.
1	HTTPD_FAILURE	Funkcja nie powiodła się.
2	HTTPD_INTERNAL_ERROR	Napotkano błąd wewnętrzny i nie można kontynuować przetwarzania tego żądania.
3	HTTPD_PARAMETER_ERROR	Przekazano jeden lub większą liczbę niepoprawnych parametrów.
4	HTTPD_STATE_CHECK	Funkcja nie jest poprawna w tym kroku przetwarzania.
5	HTTPD_READ_ONLY	(Zwracany tylko przez funkcje HTTPD_set i httpd_setvar). Zmienna tylko do odczytu, nie może być ustawiana przez wtyczkę.
6	HTTPD_BUFFER_TOO_SMALL	(Zwracany przez funkcje HTTPD_set, httpd_setvar oraz HTTPD_read). Udostępniony bufor jest za mały.
7	HTTPD_AUTHENTICATE_FAILED	(Zwracany tylko przez funkcję HTTPD_authenticate). Uwierzytelnianie nie powiodło się. Aby uzyskać więcej informacji, należy sprawdzić zmienne HTTP_RESPONSE i HTTP_REASON.

Tabela 3. Kody powrotu (kontynuacja)

Wartość	Kod statusu	Wyjaśnienie
8	HTTPD_EOF	(Zwracany tylko przez funkcję HTTPD_read). Wskazuje koniec treści żądania.
9	HTTPD_ABORT_REQUEST	Żądanie zostało przerwane, ponieważ klient udostępnił znacznik obiektu, który nie był zgodny z warunkiem określonym przez żądanie.
10	HTTPD_REQUEST_SERVICED	(Zwracany przez funkcję HTTPD_proxy.) Wywołana funkcja zakończyła proces odpowiadania na żądanie.
11	HTTPD_RESPONSE_ALREADY_ COMPLETED	Funkcja nie powiodła się, ponieważ odpowiedź na to żądanie została już udzielona.
12	HTTPD_WRITE_ONLY	Zmienna tylko do zapisu, nie może być odczytywana przez wtyczkę.

Dyrektywy konfiguracyjne komponentu Buforujący serwer proxy dla kroków interfejsu API

Dla każdego kroku w procesie przetwarzania żądania istnieje dyrektywa konfiguracyjna służącą do określania funkcji wtyczki, które mają zostać wywołane i wykonane w danym kroku. Dyrektywy można dodawać do pliku konfiguracyjnego serwera (ibmproxy.conf) ręcznie edytując i aktualizując go, lub używając formularza Przetwarzanie żądań interfejsu API znajdującego się na stronie Konfigurowanie i administrowanie komponentu Buforujący serwer proxy.

Uwagi dotyczące używania interfejsu API

- Dyrektywy interfejsu API, z wyjątkiem dyrektyw Service oraz NameTrans, nie muszą występować w pliku konfiguracyjnym w określonym porządku. Należy pamiętać, że porządek wpisów dotyczących danej dyrektywy interfejsu API jest istotny, co opisano w dalszej części tej listy.
- Uwzględnienie wpisu dla każdego kroku interfejsu API nie jest konieczne. Jeśli dla danego kroku nie ma wtyczki, należy pominąć odpowiednią dyrektywę. Zostanie użyte standardowe przetwarzanie tego kroku.
- Dyrektywy Service i NameTrans działają tak, jak inne dyrektywy odwzorowania (na przykład dyrektywa Pass) i są zależne od swojego położenia względem innych dyrektyw odwzorowania w pliku konfiguracyjnym. Na przykład reguła dla pliku /cgi-bin/foo.so musi występować przed regułą dla ścieżki /cgi-bin/*.

  Oznacza to, że serwer przetwarza dyrektywy Service, NameTrans, Exec, Fail, Map, Pass, Proxy, ProxyWAS oraz Redirect w porządku, w którym występują w pliku konfiguracyjnym. Jeśli serwer pomyślnie odwzoruje adres URL na plik, nie czyta ani nie przetwarza kolejnych dyrektyw. (Dyrektywa Map jest wyjątkiem. *Podręcznik administrowania komponentem Buforujący serwer proxy produktu WebSphere Application Server* zawiera pełne informacje dotyczące reguł odwzorowywania serwera proxy).
- Dla jednego kroku może istnieć więcej niż jedna dyrektywa konfiguracyjna. Można na przykład uwzględnić dwie dyrektywy NameTrans, z których każda wskazuje inną funkcję wtyczki. Podczas przetwarzania kroku Tłumaczenie nazwy serwer przetwarza funkcje tłumaczenia nazw w porządku, w którym występują w pliku konfiguracyjnym.

  Uwaga: Jeśli funkcja wtyczki udostępniana z komponentem Buforujący serwer proxy używa tej samej dyrektywy interfejsu API co utworzona wtyczka, dyrektywę utworzonej wtyczki należy umieścić po systemowej dyrektywie wtyczki.
- Niektóre funkcje wtyczki nie muszą być wykonywane dla każdego żądania:

– Kilka dyrektyw zawiera maskę URL. Określenie maski URL wraz z tymi dyrektywami powoduje wywołanie aplikacji wtyczki tylko dla żądań, których adresy URL są zgodne ze wzorcem. W sekcji “Dyrektywy interfejsu API i ich składnia” można znaleźć informacje dotyczące kroków, w których można używać masek URL, natomiast w sekcji “Zmienne dyrektyw interfejsu API” znajdują się informacji na temat korzystania z tej opcji.
– Określenie schematu uwierzytelniania wraz z dyrektywą Authentication wskazuje, że wtyczka ma zostać wywołana tylko dla określonych typów uwierzytelniania. Protokół HTTP obsługuje obecnie tylko uwierzytelnianie podstawowe. Więcej informacji można znaleźć w sekcji “Zmienne dyrektyw interfejsu API”.

- Jeśli załadowanie określonej funkcji wtyczki przez serwer nie powiedzie się lub dyrektywa ServerInit nie zwróci kodu powrotu OK, nie zostaną wywołane żadne inne funkcje danej skompilowanej wtyczki komponentu Buforujący serwer proxy. Wykonane do tej pory etapy przetwarzania właściwe dla tej wtyczki zostaną zignorowane. Nie ma to wpływu na inne wtyczki komponentu Buforujący serwer proxy (oraz ich funkcje) uwzględnione w tych dyrektywach.

Dyrektywy interfejsu API i ich składnia

Dyrektywy pliku konfiguracyjnego muszą występować w pliku ibmproxy.conf w jednym wierszu, bez odstępów innych niż te jawnie określone poniżej. Chociaż w niektórych przykładach umieszczone zostały znaczniki łamania wiersza w celu zachowania czytelności, w rzeczywistych dyrektywach w tych miejscach nie mogą pojawić się odstępy.

Tabela 4. Dyrektywy interfejsu API wtyczek komponentu Buforujący serwer proxy

ServerInit		*/ścieżka/plik:nazwa_funkcji łańcuch_inicjowania*
PreExit		*/ścieżka/plik:nazwa_funkcji*
Authentication	*type*	*/ścieżka/plik:nazwa_funkcji*
NameTrans	*/URL*	*/ścieżka/plik:nazwa_funkcji*
Authorization	*/URL*	*/ścieżka/plik:nazwa_funkcji*
ObjectType	*/URL*	*/ścieżka/plik:nazwa_funkcji*
PostAuth		*/ścieżka/plik:nazwa_funkcji*
Service	*/URL*	*/ścieżka/plik:nazwa_funkcji*
Midnight		*/ścieżka/plik:nazwa_funkcji*
Transmogrifier		*/ścieżka/plik:nazwa_funkcji_otwierania: nazwa_funkcji_zapisu: nazwa_funkcji_zamykania:funkcja_błędu*
Log	*/URL*	*/ścieżka/plik:nazwa_funkcji*
Error	*/URL*	*/ścieżka/plik:nazwa_funkcji*
PostExit		*/ścieżka/plik:nazwa_funkcji*
ServerTerm		*/ścieżka/plik:nazwa_funkcji*
ProxyAdvisor		*/ścieżka/plik:nazwa_funkcji*
GCAdvisor		*/ścieżka/plik:nazwa_funkcji*

Zmienne dyrektyw interfejsu API

Zmienne w dyrektywach mają następujące znaczenie:

type Używana tylko z dyrektywą Authentication w celu określenia, czy funkcja wtyczki ma zostać wywołana. Poprawne wartości to:

- Basic — Funkcja wtyczki jest wywoływana tylko dla żądań podstawowego uwierzytelniania.

- * — Funkcja wtyczki jest wywoływana dla wszystkich żądań. Protokół HTTP obsługuje obecnie tylko podstawowe uwierzytelnianie. Dla żądań uwierzytelniania innego niż podstawowe można zwrócić kod błędu wskazujący, że ten typ uwierzytelniania nie jest obsługiwany.

URL Określa żądania, dla których wywoływana jest funkcja wtyczki. Żądania o adresach URL, które są zgodne z tym szablonem, spowodują użycie funkcji wtyczki. Specyfikacje adresu URL w tych dyrektywach są wirtualne (nie zawierają protokołu), ale są poprzedzane ukośnikiem (/). Na przykład adres /www.ics.raleigh.ibm.com jest poprawny, lecz http://www.ics.raleigh.ibm.com nie. Tę wartość można określić jako konkretny adres URL lub jako szablon.

- Konkretny adres URL — Funkcja wtyczki jest wywoływana tylko dla tego konkretnego adresu URL.
- Szablon URL — Funkcja wtyczki jest wywoływana dla wszystkich adresów URL zgodnych z szablonem. Szablony mogą zawierać znak zastępczy * i mogą być określane w formie /*URL** lub /* lub *.

Uwaga: Szablon URL jest *wymagany* dla dyrektywy Service, jeśli ma nastąpić tłumaczenie ścieżki.

ścieżka/plik
Pełna nazwa pliku skompilowanego programu.

nazwa_funkcji
Nazwa nadana funkcji wtyczki w programie.

Dyrektywa Service wymaga podania znaku gwiazdki (*) po nazwie funkcji, aby możliwy był dostęp do informacji o ścieżce.

łańcuch_inicjowania
Ta opcjonalna część dyrektywy ServerInit może zawierać dowolny tekst, który ma zostać przekazany do funkcji wtyczki. Aby wyodrębnić tekst ze zmiennej INIT_STRING, należy użyć funkcji httpd_getvar().

Więcej informacji dotyczących dyrektyw, w tym opisy składni, zawiera *Podręcznik administrowania komponentem Buforujący serwer proxy produktu WebSphere Application Server*.

Kompatybilność z innymi interfejsami API

Interfejs API komponentu Buforujący serwer proxy jest kompatybilny wstecz z interfejsami ICAPI i GWAPI do wersji 4.6.1.

Przenoszenie programów CGI

Aby przenieść aplikacje CGI napisane w języku C tak, aby używały interfejsu API komponentu Buforujący serwer proxy, należy skorzystać z poniższych wytycznych:

- Usuń punkt wejścia main() lub zmień jego nazwę tak, aby możliwe było zbudowanie biblioteki DLL.
- Usuń zmienne globalne lub ochroń je za pomocą semafora wzajemnego wykluczenia.
- Zmień następujące wywołania w programach:
  - Zmień wywołania funkcji printf() dotyczące nagłówków na HTTPD_set() lub httpd_setvar().
  - Zmień wywołania funkcji printf() dotyczące danych na HTTPD_write().
  - Zmień wywołania getenv() na HTTPD_extract() lub httpd_getvar(). Należy pamiętać, że ta funkcja zwraca nieprzydzieloną pamięć, należy więc zwolnić wynik.

- Pamiętaj, że serwer działa w środowisku wielowątkowym i funkcje wtyczki muszą być wątkowo bezpieczne. Jeśli funkcje są wielobieżne, nie nastąpi spadek wydajności.
- Pamiętaj o ustawieniu nagłówka Content-Type, jeśli używasz funkcji HTTPD_write() do wysłania danych z powrotem do klienta.
- Dokładnie sprawdź kod w celu znalezienia przecieków pamięci.
- Zastanów się nad ścieżkami błędów. W przypadku własnoręcznego generowania komunikatów o błędach i wysyłania ich z powrotem w formie kodu HTML należy zwrócić kod HTTPD_OK z funkcji usługi.

Informacje uzupełniające dotyczące interfejsu API komponentu Buforujący serwer proxy

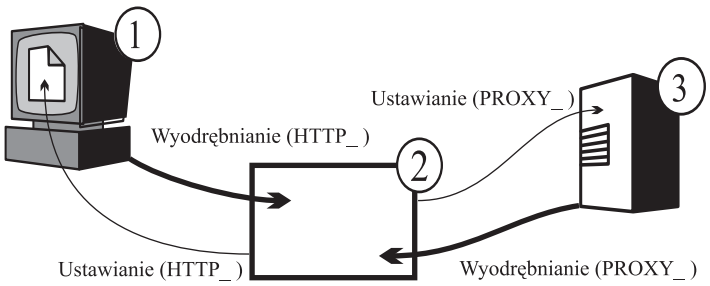
Zmienne

Pisząc programy z zastosowaniem interfejsu API, można używać zmiennych komponentu Buforujący serwer proxy, które udostępniają informacje o zdalnym kliencie i systemie serwera.

Uwagi:

- Nazwy zmiennych zdefiniowanych przez użytkownika nie mogą mieć przedrostka SERVER_. Funkcje API komponentu Buforujący serwer proxy rezerwują wszystkie zmienne z przedrostkiem SERVER_ dla serwera, dlatego zmienne te są przeznaczone tylko do odczytu. Dodatkowo dla nagłówków HTTP zarezerwowane są również przedrostki HTTP_ i PROXY_.
- Wszystkie nagłówki żądania wysyłane przez klienta (takie jak Set-Cookie) są poprzedzone przedrostkiem HTTP_, a ich wartości mogą być wyodrębniane. Aby uzyskać dostęp do zmiennych będących nagłówkami żądania, nazwę zmiennej należy poprzedzić przedrostkiem HTTP_. Nowe zmienne można tworzyć przy użyciu predefiniowanej funkcji httpd_setvar(). Aby uzyskać szczegółowe informacje na temat tych nagłówków, patrz “Kody powrotu z predefiniowanych funkcji i makr” na stronie 21.
- Dwa przedrostki zmiennych, HTTP_ oraz PROXY_, są używane do oznaczenia, czy zmienna dotyczy nagłówka żądania czy też odpowiedzi. Przedrostek HTTP_ odnosi się do zmiennych przekazywanych między klientem a komponentem Buforujący serwer proxy. Przedrostek PROXY_ odnosi się do zmiennych przekazywanych między komponentem Buforujący serwer proxy a serwerem źródłowym (lub następnym serwerem w łańcuchu proxy). Zmienne te są poprawne tylko podczas przetwarzania żądania.
  - Wyodrębnienie zmiennej HTTP_* pozwala uzyskać wartość nagłówka umieszczonego w żądaniu klienta do serwera proxy.
  - Ustawienie zmiennej HTTP_* powoduje ustawienie nagłówka odpowiedzi wysyłanego z serwera proxy do klienta.
  - Wyodrębnienie zmiennej PROXY_* pozwala uzyskać wartość nagłówka zwróconego z serwera treści do serwera proxy.
  - Ustawienie zmiennej PROXY_* powoduje ustawienie nagłówka żądania wysyłanego z serwera proxy do serwera treści (lub następnego serwera w łańcuchu proxy).

  Na Rys. 2 na stronie 26 przedstawiono użycie tych przedrostków w czasie obsługi żądania klienta przez komponent Buforujący serwer proxy.



Rysunek 2. Przedrostki zmiennych HTTP_ i PROXY_. Legenda: 1 — Klient 2 — Buforujący serwer proxy 3 — Serwer źródłowy

- Niektóre zmienne są przeznaczone tylko do odczytu. Zmienne tylko do odczytu reprezentują wartości, które można wyodrębnić z żądania lub odpowiedzi i użyć w predefiniowanej funkcji httpd_getvar(). W przypadku podjęcia próby zmodyfikowania zmiennej przeznaczonej tylko do odczytu przy użyciu funkcji httpd_setvar() zwracany jest kod powrotu HTTPD_READ_ONLY.
- Zmienne nieokreślone jako tylko do odczytu mogą być odczytywane i ustawiane przez predefiniowane funkcje httpd_getvar() i httpd_setvar(). Zmienne te reprezentują wartości, które można wyodrębnić z żądania lub odpowiedzi, lub wartości, które można ustawić lub utworzyć podczas przetwarzania żądania lub odpowiedzi.

Definicje zmiennych

Uwaga: Zmienne nagłówka nierozpoczynające się przedrostkami HTTP_ lub PROXY_ są wieloznaczne. Aby uniknąć niejednoznaczności, należy zawsze używać przedrostka HTTP_ lub PROXY_ przed nazwą zmiennej nagłówka.

ACCEPT_RANGES

Zawiera wartość nagłówka odpowiedzi Accept-Ranges, który określa, czy serwer treści może odpowiadać na żądania zakresu. Do wyodrębnienia wartości nagłówka wysłanego przez serwer treści do serwera proxy należy użyć zmiennej PROXY_ACCEPT_RANGES. Do ustawienia wartości nagłówka wysłanego z serwera proxy do klienta należy użyć zmiennej HTTP_ACCEPT_RANGES.

Uwaga: Zmienna ACCEPT_RANGES jest wieloznaczna. Aby wyeliminować niejednoznaczność, należy użyć zmiennej HTTP_ACCEPT_RANGES lub PROXY_ACCEPT_RANGES.

ALL_VARIABLES

Tylko do odczytu. Zawiera wszystkie zmienne CGI. Na przykład:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING

Tylko do odczytu. Jeśli serwer obsługuje uwierzytelnianie klienta, ten łańcuch zawiera zdekodowane referencje na potrzeby uwierzytelnienia klienta.

AUTH_TYPE

Tylko do odczytu. Jeśli serwer obsługuje uwierzytelnianie klienta i skrypt jest chroniony, ta zmienna zawiera metodę używaną do uwierzytelnienia klienta. Na przykład: Basic (Podstawowa).

CACHE_HIT

Tylko do odczytu. Określa, czy żądanie proxy zostało znalezione w pamięci podręcznej. Zwracane wartości:

- 0 - Żądanie nie zostało znalezione w pamięci podręcznej.

- 1 - Żądanie zostało znalezione w pamięci podręcznej.

CACHE_MISS

Tylko do zapisu. Używane do wymuszenia chybienia w pamięci podręcznej. Poprawne wartości to:

- 0 - Nie wymuszaj chybienia w pamięci podręcznej.
- 1 - Wymuszaj chybienie w pamięci podręcznej.

CACHE_TASK

Tylko do odczytu. Określa, czy została użyta pamięć podręczna. Zwracane wartości:

- 0 - Żądanie nie spowodowało uzyskania dostępu do pamięci podręcznej ani jej aktualizacji.
- 1 - Żądanie zostało obsłużone przy użyciu pamięci podręcznej.
- 2 - Żądany obiekt znajdował się w pamięci podręcznej, ale należało ponownie sprawdzić jego poprawność.
- 3 - Żądany obiekt nie znajdował się w pamięci podręcznej i prawdopodobnie został dodany.

Zmienna ta może być używana w krokach PostAuthorization, PostExit, ProxyAdvisor i Log.

CACHE_UPDATE

Tylko do odczytu. Określa, czy żądanie proxy spowodowało aktualizację pamięci podręcznej. Zwracane wartości:

- 0 - Pamięć podręczna nie była aktualizowana.
- 1 - Pamięć podręczna została zaktualizowana.

CLIENT_ADDR lub CLIENTADDR

Tak samo, jak dla zmiennej REMOTE_ADDR.

CLIENTMETHOD

Tak samo, jak dla zmiennej REQUEST_METHOD.

CLIENT_NAME lub CLIENTNAME

Tak samo, jak dla zmiennej REMOTE_HOST.

CLIENT_PROTOCOL lub CLIENTPROTOCOL

Zawiera nazwę i wersję protokołu, którego używa klient do zgłaszania żądań. Na przykład: HTTP/1.1.

CLIENT_RESPONSE_HEADERS

Tylko do odczytu. Zwraca bufor zawierający nagłówki, które serwer wysyła do klienta.

CONNECTIONS

Tylko do odczytu. Zawiera liczbę obsługiwanych połączeń lub liczbę aktywnych żądań. Na przykład: 15.

CONTENT_CHARSET

Zestaw znaków odpowiedzi dla tekstu, na przykład US ASCII. Zmienna ta wyodrębniana jest z nagłówka content-charset klienta. Ustawienie tej zmiennej wpływa na nagłówek content-charset żądania do serwera treści.

CONTENT_ENCODING

Określa kodowanie używane w dokumencie, na przykład x-gzip. Zmienna ta wyodrębniana jest z nagłówka content-encoding klienta. Ustawienie tej zmiennej wpływa na nagłówek content-charset żądania do serwera treści.

CONTENT_LENGTH
Ta zmienna wyodrębniana jest z nagłówka żądania klienta. Ustawienie tej zmiennej wpływa na wartość nagłówka żądania do serwera treści.

Uwaga: Zmienna CONTENT_LENGTH jest wieloznaczna. Aby wyeliminować niejednoznaczność, należy używać zmiennych HTTP_CONTENT_LENGTH i PROXY_CONTENT_LENGTH.

CONTENT_TYPE
Ta zmienna wyodrębniana jest z nagłówka żądania klienta. Ustawienie tej zmiennej wpływa na wartość nagłówka żądania do serwera treści.

Uwaga: Zmienna CONTENT_TYPE jest wieloznaczna. Aby wyeliminować niejednoznaczność, należy używać zmiennych HTTP_CONTENT_TYPE i PROXY_CONTENT_TYPE.

CONTENT_TYPE_PARAMETERS
Zawiera inne atrybuty MIME oprócz zestawu znaków. Zmienna ta wyodrębniana jest z nagłówka żądania klienta. Ustawienie tej zmiennej wpływa na wartość nagłówka żądania do serwera treści.

DOCUMENT_URL
Zawiera adres URL. Na przykład:

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

DOCUMENT_URI
Tak samo, jak dla zmiennej DOCUMENT_URL.

DOCUMENT_ROOT
Tylko do odczytu. Zawiera główną ścieżkę do dokumentu zdefiniowaną przez reguły przekazywania.

ERRORINFO
Zawiera kod błędu do określenia strony błędu. Na przykład: blocked.

EXPIRES
Określa, kiedy dokumenty przechowywane w pamięci podręcznej serwera proxy tracą ważność. Ta zmienna wyodrębniana jest z nagłówka żądania klienta. Ustawienie tej zmiennej wpływa na wartość nagłówka żądania do serwera treści. Na przykład:

```
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 Buforujący serwer 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. Wartości między 0.0 i 1.0 są porządkowane według oceny, to znaczy, że plik o wartości GC_EVALUATION równej 0.1 ma większe szanse na usunięcie, niż plik o wartości GC_EVALUATION równej 0.9.

GC_EXPIRES
Tylko do odczytu. Określa liczbę sekund, po upływie których rozważany do czyszczenia pamięci plik utraci ważność w pamięci podręcznej. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

GC_FILENAME
Tylko do odczytu. Określa plik, dla którego rozważane jest podjęcie czyszczenia pamięci. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

GC_FILESIZE
Tylko do odczytu. Określa wielkość pliku, dla którego rozważane jest podjęcie czyszczenia pamięci. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

GC_LAST_ACCESS
Tylko do odczytu. Określa, kiedy ostatnio uzyskiwany był dostęp do pliku. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

GC_LAST_CHECKED
Tylko do odczytu. Określa, kiedy ostatnio plik był sprawdzany. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

GC_LOAD_DELAY
Tylko do odczytu. Określa, ile czasu zajęło pobranie pliku. Zmienna ta może zostać wyodrębniona tylko przez wtyczkę GC Advisor.

HTTP_COOKIE
Podczas odczytu zmienna ta zawiera wartość nagłówka Set-Cookie ustawionego przez klienta. Może zostać także użyta do ustawienia nowej informacji cookie w strumieniu odpowiedzi (między serwerem proxy a klientem). Ustawienie tej zmiennej powoduje utworzenie nowego nagłówka Set-Cookie w strumieniu żądania dokumentu bez względu na to, czy istnieją zduplikowane nagłówki.

HTTP_HEADERS
Tylko do odczytu. Używana do wyodrębnienia wszystkich nagłówków żądania klienta.

HTTP_REASON
Ustawienie tej zmiennej wpływa na łańcuch przyczyny w odpowiedzi HTTP. Ustawienie jej wpływa także na łańcuch przyczyny w odpowiedzi serwera proxy do klienta. Wyodrębnienie tej zmiennej zwraca łańcuch przyczyny w odpowiedzi z serwera treści do serwera proxy.

HTTP_RESPONSE
Ustawienie tej zmiennej wpływa na kod odpowiedzi w odpowiedzi HTTP. Ustawienie jej wpływa także na kod statusu w odpowiedzi serwera proxy do klienta. Wyodrębnienie tej zmiennej zwraca kod statusu w odpowiedzi z serwera treści do serwera proxy.

HTTP_STATUS
Zawiera kod odpowiedzi HTTP oraz łańcuch przyczyny. Na przykład: 200 OK.

HTTP_USER_AGENT
Zawiera wartość nagłówka żądania User-Agent, która jest nazwą przeglądarki WWW klienta, na przykład Netscape Navigator / V2.02. Ustawienie tej zmiennej wpływa na nagłówek odpowiedzi serwera proxy do klienta. Ta zmienna wyodrębniana jest z nagłówka żądania klienta.

INIT_STRING
Tylko do odczytu. Ten łańcuch definiuje dyrektywa ServerInit. Zmienna ta może zostać odczytana tylko podczas kroku inicjowania serwera.

LAST_MODIFIED
Zmienna ta wyodrębniana jest z nagłówka żądania klienta. Ustawienie tej zmiennej wpływa na wartość nagłówka żądania do serwera treści. Na przykład:

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

LOCAL_VARIABLES
Tylko do odczytu. Wszystkie zmienne zdefiniowane przez użytkownika.

MAXACTIVETHREADS
Tylko do odczytu. Maksymalna liczba aktywnych wątków.

NOTMODIFIED_TO_OK
Wymusza pełną odpowiedź do klienta. Poprawna tylko w krokach PreExit i ProxyAdvisor.

ORIGINAL_HOST
Tylko do odczytu. Zwraca nazwę hosta lub docelowy adres IP żądania.

ORIGINAL_URL
Tylko do odczytu. Zwraca oryginalny adres URL wysłany w żądaniu klienta.

OVERRIDE_HTTP_NOTRANSFORM
Umożliwia modyfikację danych w obecności nagłówka Cache-Control: no-transform. Ustawienie tej zmiennej wpływa na nagłówek odpowiedzi do klienta.

OVERRIDE_PROXY_NOTRANSFORM
Umożliwia modyfikację danych w obecności nagłówka Cache-Control: no-transform. Ustawienie tej zmiennej wpływa na żądanie do serwera treści.

PASSWORD
Zawiera zdekodowane hasło dla uwierzytelniania podstawowego. Na przykład: password.

PATH Zawiera w pełni przetłumaczoną ścieżkę.

PATH_INFO
Zawiera dodatkowe informacje na temat ścieżki przesłane przez przeglądarkę WWW. Na przykład: /foo.

PATH_TRANSLATED
Zawiera zdekodowaną lub przetłumaczoną wersję informacji na temat ścieżki zawartych w zmiennej PATH_INFO. Na przykład:

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

```
/wwwhome/foo
```

PPATH
Zawiera częściowo przetłumaczoną ścieżkę. Należy jej użyć w kroku Name Translation.

PROXIED_CONTENT_LENGTH
Tylko do odczytu. Zwraca długość danych odpowiedzi przesłanych w rzeczywistości przez serwer proxy.

PROXY_ACCESS
Określa, czy żądanie jest żądaniem proxy. Na przykład: NO.

PROXY_CONTENT_TYPE
Zawiera nagłówek Content-Type żądania proxy zgłoszonego przy użyciu metody HTTPD_proxy(). Gdy informacje są przesyłane metodą POST, ta zmienna zawiera typ dołączonych danych. W pliku konfiguracyjnym serwera proxy można utworzyć własny typ treści i odwzorować go na przeglądarkę. Wartość tej zmiennej wyodrębniana jest z nagłówka odpowiedzi serwera treści. Ustawienie tej zmiennej wpływa na nagłówek żądania do serwera treści. Na przykład:

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

PROXY_CONTENT_LENGTH

Nagłówek Content-Length żądania proxy zgłoszonego przy użyciu metody HTTPD_proxy(). Gdy informacje są przesyłane metodą POST, ta zmienna zawiera liczbę znaków danych. Serwery nie wysyłają zwykle flagi końca pliku, kiedy przesyłają dalej informacje, korzystając ze standardowego wejścia. Jeśli jest taka potrzeba, można użyć zmiennej CONTENT_LENGTH do określenia końca łańcucha wejściowego. Wartość tej zmiennej wyodrębniana jest z nagłówka odpowiedzi serwera treści. Ustawienie tej zmiennej wpływa na nagłówek żądania do serwera treści. Na przykład:

```
7034
```

PROXY_COOKIE

Podczas odczytu zmienna ta zawiera wartość nagłówka Set-Cookie ustawionego przez serwer źródłowy. Może zostać również użyta do ustawienia nowej informacji cookie w strumieniu żądania. Ustawienie tej zmiennej powoduje utworzenie nowego nagłówka Set-Cookie w strumieniu żądania dokumentu bez względu na to, czy istnieją zduplikowane nagłówki.

PROXY_HEADERS

Tylko do odczytu. Używana do wyodrębnienia nagłówków Proxy.

PROXY_METHOD

Metoda dla żądania zgłoszonego przy użyciu metody HTTPD_proxy(). Wartość tej zmiennej wyodrębniana jest z nagłówka odpowiedzi serwera treści. Ustawienie tej zmiennej wpływa na nagłówek żądania do serwera treści.

QUERY_STRING

Gdy informacje są przesyłane przy użyciu metody GET, ta zmienna zawiera informacje następujące po znaku zapytania (?) w zapytaniu. Informacje te muszą zostać zdekodowane przez program CGI. Na przykład:

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

RCA_OWNER

Tylko do odczytu. Zwraca wartość liczbową określającą węzeł będący właścicielem żądanego obiektu. Zmienna ta może być używana w krokach PostExit, ProxyAdvisor lub Log i ma zastosowanie, tylko jeśli serwer jest częścią tablicy pamięci podręcznej korzystającej ze zdalnego dostępu do pamięci podręcznej (RCA).

RCA_TIMEOUTS

Tylko do odczytu. Zwraca wartość liczbową zawierającą łączną (zagregowaną) liczbę przekroczeń limitu czasu dla żądań RCA do wszystkich węzłów sieci. Można używać tej zmiennej w dowolnym kroku.

REDIRECT_*

Tylko do odczytu. Zawiera łańcuch przekierowania dla kodu błędu odpowiadającego nazwie zmiennej (na przykład REDIRECT_URL). Listę możliwych zmiennych REDIRECT_ można znaleźć w dokumentacji elektronicznej dla serwera WWW Apache pod adresem http://httpd.apache.org/docs-2.0/custom-error.html.

REFERRER_URL

Tylko do odczytu. Zawiera ostatni adres URL miejsca przeglądarki. Pozwala określić klientowi, z korzyścią dla serwera, adres URL zasobu, z którego uzyskano adres URL żądania. Na przykład:

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

REMOTE_ADDR

Zawiera adres IP przeglądarki WWW, jeśli jest dostępny. Na przykład: 45.23.06.8.

REMOTE_HOST
Zawiera nazwę hosta przeglądarki WWW, jeśli jest dostępna. Na przykład: www.raleigh.ibm.com.

REMOTE_USER
Jeśli serwer obsługuje uwierzytelnianie klienta i skrypt jest chroniony, ta zmienna zawiera nazwę użytkownika przekazaną do uwierzytelnienia. Na przykład: joeuser.

REQHDR
Tylko do odczytu. Zawiera listę nagłówków wysłanych przez klienta.

REQUEST_CONTENT_TYPE
Tylko do odczytu. Zwraca typ treści żądania. Na przykład:

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

REQUEST_CONTENT_LENGTH
Tylko do odczytu. Gdy informacje są przesyłane metodą POST, ta zmienna zawiera liczbę znaków danych. Serwery nie wysyłają zwykle flagi końca pliku, kiedy przesyłają dalej informacje, korzystając ze standardowego wejścia. Jeśli jest taka potrzeba, można użyć zmiennej CONTENT_LENGTH do określenia końca łańcucha wejściowego. Na przykład: 7034.

REQUEST_METHOD
Tylko do odczytu. Zawiera metodę (określoną w atrybucie METHOD formularza HTML) użytą do wysłania żądania. Na przykład: GET lub POST.

REQUEST_PORT
Tylko do odczytu. Zwraca numer portu określonego w adresie URL lub domyślny port dla danego protokołu.

RESPONSE_CONTENT_TYPE
Tylko do odczytu. Gdy informacje są przesyłane metodą POST, ta zmienna zawiera typ dołączonych danych. W pliku konfiguracyjnym serwera proxy można utworzyć własny typ treści i odwzorować go na przeglądarkę. Na przykład: text/html.

RESPONSE_CONTENT_LENGTH
Tylko do odczytu. Gdy informacje są przesyłane metodą POST, ta zmienna zawiera liczbę znaków danych. Serwery nie wysyłają zwykle flagi końca pliku, kiedy przesyłają dalej informacje, korzystając ze standardowego wejścia. Jeśli jest taka potrzeba, można użyć zmiennej CONTENT_LENGTH do określenia końca łańcucha wejściowego. Na przykład: 7034.

RULE_FILE_PATH
Tylko do odczytu. Zawiera pełną ścieżkę systemu plików i nazwę pliku konfiguracyjnego.

SSL_SESSIONID
Tylko do odczytu. Zwraca identyfikator sesji SSL, jeśli bieżące żądanie zostało odebrane w połączeniu SSL. Zwraca wartość NULL, jeśli bieżącego żądania nie odebrano w połączeniu SSL.

SCRIPT_NAME
Zawiera adres URL żądania.

SERVER_ADDR
Tylko do odczytu. Zawiera lokalny adres IP serwera proxy.

SERVER_NAME
Tylko do odczytu. Zawiera nazwę hosta serwera proxy lub adres IP serwera treści dla tego żądania. Na przykład: www.ibm.com.

SERVER_PORT
Tylko do odczytu. Zawiera numer portu serwera proxy, do którego zostało wysłane żądanie klienta. Na przykład: 80.

SERVER_PROTOCOL
Tylko do odczytu. Zawiera nazwę i wersję protokołu użytego do zgłoszenia żądania. Na przykład: HTTP/1.1.

SERVER_ROOT
Tylko do odczytu. Zawiera katalog, w którym zainstalowano program serwera proxy.

SERVER_SOFTWARE
Tylko do odczytu. Zawiera nazwę i wersję serwera proxy.

STATUS
Zawiera kod odpowiedzi HTTP oraz łańcuch przyczyny. Na przykład: 200 OK.

TRACE
Określa szczegółowość śledzenia. Zwracane wartości to:

- OFF - Brak śledzenia.
- V - Tryb szczegółowy (Verbose).
- VV - Tryb bardzo szczegółowy (Very Verbose).
- MTV - Tryb zbyt szczegółowy (Much Too Verbose).

URI Do odczytu/zapisu. Tak samo, jak dla zmiennej DOCUMENT_URL.

URI_PATH
Tylko do odczytu. Zwraca tylko część ścieżki z adresu URL.

URL Do odczytu/zapisu. Tak samo, jak dla zmiennej DOCUMENT_URL.

URL_MD4
Tylko do odczytu. Zwraca nazwę potencjalnego pliku pamięci podręcznej dla bieżącego żądania.

USE_PROXY
Określa serwer proxy do uwzględnienia w łańcuchu dla bieżącego żądania. Należy podać adres URL. Na przykład: http://myproxy:8080.

USERID
Tak samo, jak dla zmiennej REMOTE_USER.

USERNAME
Tak samo, jak dla zmiennej REMOTE_USER.

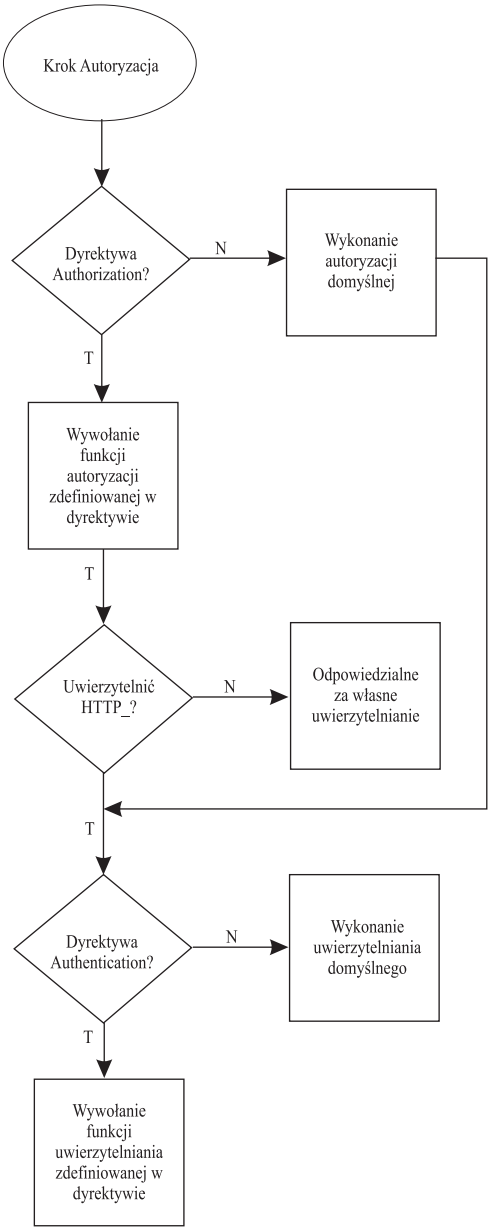
Uwierzytelnianie i autoryzacja

Najpierw krótki przegląd terminologii:

Uwierzytelnianie
Weryfikacja elementów zabezpieczeń powiązanych z żądaniem w celu określenia tożsamości podmiotu zgłaszającego żądanie.

Autoryzacja
Proces korzystający z elementów zabezpieczeń w celu określenia, czy podmiot zgłaszający żądanie ma dostęp do zasobu.

Rys. 3 na stronie 34 ilustruje proces uwierzytelniania i autoryzacji serwera proxy.



Rysunek 3. Proces uwierzytelniania i autoryzacji serwera proxy

Zgodnie z tym co zaprezentowano na diagramie Rys. 3 inicjowanie procesu autoryzacji jest pierwszym krokiem w procesie autoryzacji i uwierzytelniania serwera.

W komponencie Buforujący serwer proxy uwierzytelnianie jest częścią procesu autoryzacji. Jest wykonywane tylko wtedy, gdy wymagana jest autoryzacja.

Proces uwierzytelniania i autoryzacji

Podczas przetwarzania żądania, które wymaga autoryzacji, serwer proxy wykonuje następujące kroki.

1. Najpierw serwer proxy sprawdza plik konfiguracyjny, aby określić, czy istnieje dyrektywa autoryzacji.
   - Jeśli w pliku konfiguracyjnym znajduje się dyrektywa autoryzacji, serwer wywołuje funkcję autoryzacji zdefiniowaną w dyrektywie i rozpoczyna uwierzytelnianie od kroku 2 na stronie 35.

- W przypadku braku dyrektywy autoryzacji serwer przeprowadza autoryzację domyślną, a następnie przechodzi bezpośrednio do procedur uwierzytelniania w kroku 3.

2. Serwer proxy rozpoczyna proces uwierzytelniania od sprawdzenia, czy w żądaniu klienta istnieje nagłówek HTTP_authenticate.
   - Jeśli nagłówek istnieje, serwer przechodzi do procesu uwierzytelniania (patrz krok 3).
   - Jeśli nagłówek nie istnieje, uwierzytelnianie musi być wykonane przez inną metodę.
3. Serwer proxy sprawdza, czy w pliku konfiguracyjnym znajduje się dyrektywa uwierzytelniania.
   - Jeśli w pliku konfiguracyjnym znajduje się dyrektywa uwierzytelniania, serwer wywołuje funkcję uwierzytelniania zdefiniowaną w dyrektywie.
   - Jeśli nie ma dyrektywy, serwer przeprowadza uwierzytelnianie domyślne.

Jeśli wtyczka komponentu Buforujący serwer proxy udostępnia własny proces autoryzacji, *przesłania on* domyślną autoryzację i uwierzytelnianie serwera. Jeśli w pliku konfiguracyjnym znajdują się dyrektywy autoryzacji, funkcje wtyczki powiązane z nimi muszą także obsługiwać niezbędne uwierzytelnianie. Można użyć udostępnionej predefiniowanej funkcji HTTPD_authenticate().

Obsługę uwierzytelniania we wtyczkach autoryzacji można zapewnić na trzy sposoby:

- Napisać osobne wtyczki autoryzacji i uwierzytelniania. W pliku konfiguracyjnym serwera proxy należy użyć zarówno dyrektywy Authorization, jak i Authentication do określenia tych funkcji. Należy pamiętać o uwzględnieniu wywołania funkcji HTTPD_authenticate() w funkcji wtyczki autoryzacji.

  Podczas wykonywania kroku Autoryzacja wywoływana jest funkcja wtyczki autoryzacji, która z kolei wywołuje funkcję wtyczki uwierzytelniania.
- Napisać własną funkcję wtyczki autoryzacji, która będzie wywoływać domyślne uwierzytelnianie serwera. W pliku konfiguracyjnym serwera proxy należy użyć dyrektywy Authorization do określenia funkcji. W tym przypadku dyrektywa Authentication nie jest potrzebna. Należy pamiętać o uwzględnieniu wywołania funkcji HTTPD_authenticate() w funkcji wtyczki autoryzacji.

  Podczas wykonywania kroku Autoryzacja wywoływana jest funkcja wtyczki autoryzacji, która z kolei wywołuje domyślne uwierzytelnianie serwera.
- Napisać własną funkcję wtyczki autoryzacji, uwzględniając w niej całe wymagane przetwarzanie uwierzytelniania. Nie należy używać funkcji HTTPD_authenticate() we wtyczce autoryzacji. W pliku konfiguracyjnym serwera proxy należy użyć dyrektywy Authorization do określenia wtyczki autoryzacji. W tym przypadku dyrektywa Authentication nie jest potrzebna.

  Podczas wykonywania kroku Autoryzacja wywoływana jest funkcja wtyczki autoryzacji i wykonywane są zawarte w niej elementy uwierzytelniania.

Jeśli wtyczka komponentu Buforujący serwer proxy nie udostępnia własnego procesu autoryzacji, można udostępnić dostosowane uwierzytelnianie przy użyciu następującej metody:

- Napisz własną funkcję wtyczki uwierzytelniania. W pliku konfiguracyjnym serwera proxy użyj dyrektyw Authentication do określenia funkcji. W tym przypadku dyrektywa Authorization nie jest potrzebna.

Podczas wykonywania kroku Autoryzacja wywoływana jest domyślna autoryzacja serwera, która z kolei wywołuje funkcję wtyczki uwierzytelniania.

Należy pamiętać o następujących punktach:

- Jeśli w pliku konfiguracyjnym nie ma dyrektyw Authorization lub ich określone funkcje wtyczki odmówią obsługi żądania, zwracając kod powrotu HTTP_NOACTION, przeprowadzana jest domyślna autoryzacja serwera.
- Jeśli w pliku konfiguracyjnym znajdują się dyrektywy Authorization, a ich funkcje wtyczki uwzględniają funkcję HTTPD_authenticate(), serwer wywołuje funkcje uwierzytelniania określone w dyrektywach Authentication. Jeśli dyrektywy Authentication nie zostały zdefiniowane lub ich określone funkcje wtyczki odmówią obsługi żądania, zwracając kod powrotu HTTP_NOACTION, przeprowadzane jest domyślne uwierzytelnianie serwera.
- Jeśli w pliku konfiguracyjnym znajdują się dyrektywy Authorization, lecz ich funkcje wtyczki nie zawierają funkcji HTTPD_authenticate(), serwer nie wywołuje żadnych funkcji uwierzytelniania. Należy napisać własną procedurę przetwarzania uwierzytelniania jako część funkcji wtyczki autoryzacji lub wykonać własne wywołania do innych modułów uwierzytelniania.
- Komponent Buforujący serwer proxy automatycznie generuje wezwanie (zachętę do podania ID użytkownika i hasła w przeglądarce), jeśli funkcja autoryzacji zwróci kod 401 lub 407. Wciąż jednak należy skonfigurować ochronę w komponencie Buforujący serwer proxy, aby ta akcja została wykonana poprawnie.

Buforowanie wariantów

Funkcji buforowania wariantów należy używać do buforowania danych, które są zmodyfikowaną wersją dokumentu oryginalnego (identyfikatora URI). Komponent Buforujący serwer proxy obsługuje warianty wygenerowane przez ten interfejs API. *Warianty* to różne wersje dokumentu podstawowego.

Zazwyczaj jeśli serwery źródłowe wysyłają warianty, nie identyfikują ich jako takich. Komponent Buforujący serwer proxy obsługuje tylko warianty utworzone przez wtyczki (na przykład w wyniku konwersji strony kodowej). Jeśli wtyczka tworzy wariant na podstawie kryteriów, które nie zostały podane w nagłówku HTTP, musi zawierać w kroku Przed wyjściem lub Po autoryzacji funkcję, która utworzy pseudonagłówek. Dzięki temu komponent Buforujący serwer proxy będzie mógł poprawnie zidentyfikować istniejący wariant.

Na przykład można użyć programu interfejsu API transmogryfikatora do modyfikowania żądanych przez użytkowników danych na podstawie wartości nagłówka User-Agent wysyłanego przez przeglądarkę. W funkcji *zamykania* należy zapisać zmodyfikowaną treść do pliku lub określić długość buforu i przekazać bufor jako argument danych. Następnie należy użyć funkcji buforowania wariantów, httpd_variant_insert() i httpd_variant_lookup(), aby umieścić treść w pamięci podręcznej.

Przykłady interfejsu API

Przed rozpoczęciem pisania własnych funkcji API komponentu Buforujący serwer proxy należy zapoznać się z programami przykładowymi udostępnionymi w katalogu samples instalacyjnego dysku CD produktu Edge Components. Dodatkowe informacje są dostępne w serwisie WWW produktu WebSphere Application Server, www.ibm.com/software/webservers/appserv/.

Rozdział 3. Niestandardowi doradcy

Ta sekcja omawia tworzenie niestandardowych doradców dla komponentu System równoważenia obciążenia.

Dostarczanie przez doradców informacji na temat równoważenia obciążenia

Doradcy to programowe agenty, które działają w ramach komponentu System równoważenia obciążenia, udostępniając informacje o obciążeniu określonego serwera. Dla każdego standardowego protokołu (HTTP, SSL i innych) istnieje oddzielny doradca. Kod bazowy komponentu System równoważenia obciążenia wykonuje okresowo cykl doradcy, w czasie którego niezależnie sprawdzany jest status wszystkich serwerów znajdujących się w jego konfiguracji.

Możliwe jest tworzenie własnych doradców dla komponentu System równoważenia obciążenia w celu dostosowania sposobu określania obciążenia serwerów.

W systemach Windows: Jeśli używana jest instalacja komponentu System równoważenia obciążenia dla protokołów IPv4 i IPv6 oraz jeśli na danym komputerze używany jest protokół IPv6, w celu skorzystania z doradców należy zmodyfikować plik **protocol** znajdujący się w katalogu C:\windows\system32\drivers\etc\.

Należy wstawić następujący wiersz dla protokołu IPv6 w pliku protocol:

```
ipv6-icmp  58   IPv6-ICMP      # Protokół komunikatów sterujących interfejsu IPv6
```

Funkcja standardowego doradcy

Ogólnie doradcy umożliwiają równoważnie obciążenia w następujący sposób.

1. Doradca otwiera okresowo połączenie z każdym serwerem i wysyła do niego komunikat żądania. Treść komunikatu odnosi się do protokołu obsługiwanego przez serwer; na przykład doradca HTTP wysyła do serwera żądanie HEAD.
2. Doradca nasłuchuje odpowiedzi z serwera. Po uzyskaniu odpowiedzi oblicza i raportuje wartość obciążenia dla danego serwera. Różni doradcy obliczają wartość obciążania w odmienny sposób, ale większość standardowych doradców mierzy czas odpowiedzi serwera, a następnie zgłasza tę wartość w milisekundach jako obciążenie serwera.
3. Doradca raportuje obciążenie do funkcji menedżera komponentu System równoważenia obciążenia. Wartość ta jest przedstawiana w kolumnie Port raportu menedżera. Menedżer używa wartości obciążenia zgłoszonych przez doradcę wraz z wagami ustawionymi przez administratora do określenia sposobu równoważenia obciążenia żądań przychodzących do serwerów.
4. Jeśli serwer nie odpowiada, doradca zwraca ujemną wartość obciążenia (-1). Menedżer korzysta z tej informacji do określenia, kiedy należy zawiesić usługę danego serwera.

Wraz z komponentem System równoważenia obciążenia dostarczani są standardowi doradcy dla następujących funkcji. Aby uzyskać szczegółowe informacje na temat tych doradców, patrz *Podręcznik administrowania komponentem System równoważenia obciążenia produktu WebSphere Application Server*.

- Connect
- DB2
- DNS

- FTP
- HTTP
- HTTPS
- IMAP
- LDAP
- NNTP
- Ping
- POP3
- Reach
- Self
- SIP
- SMTP
- SSL
- Telnet
- WebSphere Application Server
- WebSphere Application Server Buforujący serwer proxy
- Menedżer obciążenia

Aby zapewnić obsługę protokołów, dla których nie istnieją standardowi doradcy, należy utworzyć doradców niestandardowych.

Tworzenie niestandardowego doradcy

Niestandardowy doradca to mały fragment kodu Java udostępniany w postaci pliku klasy, który jest wywoływany przez kod bazowy komponentu System równoważenia obciążenia w celu określenia obciążenia serwera. Kod bazowy udostępnia wszystkie niezbędne usługi administracyjne, włącznie z uruchamianiem i zatrzymywaniem instancji niestandardowego doradcy, udostępnianiem statusu i raportów, rejestrowaniem informacji o historii w pliku dziennika oraz raportowaniem wyników doradcy do komponentu menedżera.

Kiedy kod bazowy komponentu System równoważenia obciążenia wywołuje niestandardowego doradcę, wykonywane są następujące czynności.

1. Kod bazowy komponentu System równoważenia obciążenia otwiera połączenie z serwerem.
2. Jeśli gniazdo zostanie otwarte, kod bazowy wywołuje funkcję GetLoad określonego doradcy.
3. Funkcja GetLoad doradcy wykonuje czynności zdefiniowane przez użytkownika w celu oceny statusu serwera, włącznie z oczekiwaniem na odpowiedź serwera. Funkcja ta kończy działanie po odebraniu odpowiedzi serwera.
4. Kod bazowy komponentu System równoważenia obciążenia zamyka gniazdo z serwerem i raportuje informacje o obciążeniu do menedżera. W zależności od tego, czy niestandardowy doradca działa w trybie normalnym czy też trybie zastępowania, kod bazowy wykonuje czasem dodatkowe obliczenia po zakończeniu działania funkcji GetLoad.

Tryb normalny i tryb zastępowania

Niestandardowi doradcy mogą być zaprojektowani do interakcji z komponentem System równoważenia obciążenia w trybie *normalnym* lub w trybie *zastępowania*.

Wybór trybu działania jest określany w pliku niestandardowego doradcy jako parametr w metodzie konstruktora. Każdy doradca może działać tylko w jednym z tych trybów, co jest zależne od sposobu zaprojektowania tego doradcy.

W trybie normalnym niestandardowy doradca wymienia dane z serwerem, a kod bazowy doradcy mierzy czas wymiany i określa wartość obciążenia. Kod bazowy raportuje następnie tę wartość do menedżera. Niestandardowy doradca zwraca wartość 0, aby wskazać sukces, lub wartość -1, która oznacza błąd.

Aby określić tryb normalny, należy ustawić opcję zastępowania w konstruktorze na wartość *false*.

W trybie zastępowanie kod bazowy nie wykonuje żadnych pomiarów czasu. Kod niestandardowego doradcy wykonuje określone operacje na podstawie unikalnych wymagań, a następnie zwraca rzeczywistą wartość obciążenia. Kod bazowy akceptuje tę wartość i raportuje ją bez zmian do menedżera. Aby uzyskać najlepsze wyniki, należy znormalizować wartości obciążenia w zakresie od 10 do 1000, gdzie 10 oznacza szybki serwer, a 1000 - wolny serwer.

Aby określić tryb zastępowania, należy ustawić opcję zastępowania w konstruktorze na wartość *true*.

Konwencje nazewnictwa doradców

Nazwa pliku niestandardowego doradcy musi mieć formę ADV_*nazwa*.java, gdzie *nazwa* to nazwa wybrana dla doradcy. Pełna nazwa musi rozpoczynać się od przedrostka ADV_ z użyciem wielkich liter, a wszystkie pozostałe znaki muszą być wpisane z użyciem małych liter. Wymaganie użycia małych liter sprawia, że w komendzie służącej do uruchamiania doradcy nie jest rozróżniana wielkość liter.

Zgodnie z konwencjami języka Java, nazwa klasy zdefiniowanej w pliku musi być zgodna z nazwą tego pliku.

Kompilacja

Tworzenie niestandardowych doradców musi odbywać się w języku Java. Ich kompilację należy wykonywać za pomocą kompilatora języka Java zainstalowanego na komputerze używanym do programowania. W czasie kompilacji przywoływane są następujące pliki:

- Plik niestandardowego doradcy
- Plik klas bazowych ibmnd.jar, który znajduje się w katalogu *ścieżka_instalacji*/servers/lib.

W czasie kompilacji zmienna środowiskowa ścieżki klasy musi wskazywać zarówno plik niestandardowego doradcy, jak i plik klas bazowych. Komenda kompilacji może mieć następujący format:

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

W tym przykładzie używana jest domyślna ścieżka instalacji w systemach Linux i UNIX. Plik doradcy nosi nazwę ADV_*nazwa*.java i jest zapisany w bieżącym katalogu.

Wynikiem kompilacji jest plik klasy, na przykład ADV_*nazwa*.class. Przed uruchomieniem doradcy należy skopiować ten plik klasy do katalogu *ścieżka_instalacji*/servers/lib/CustomAdvisors/.

Uwaga: Niestandardowych doradców można kompilować w jednym systemie operacyjnym, a następnie uruchamiać w innym systemie. Doradcę można na przykład

skompilować w systemie Windows, skopiować wynikowy plik klasy (w formacie binarnym) na komputer z system Linux i uruchomić niestandardowego doradcę na tym komputerze.

Uruchamianie niestandardowego doradcy

Aby uruchomić niestandardowego doradcę, należy najpierw skopiować plik klasy doradcy do podkatalogu lib/CustomAdvisors/ na komputerze z komponentem System równoważenia obciążenia. Na przykład ścieżka do pliku niestandardowego doradcy o nazwie myping to *ścieżka_instalacji*/servers/lib/CustomAdvisors/ADV_myping.class.

Należy skonfigurować komponent System równoważenia obciążenia, uruchomić jego funkcję menedżera, a następnie wykonać komendę w celu uruchomienia niestandardowego doradcy. Niestandardowy doradca jest określany przez jego nazwę bez przedrostka ADV_ i rozszerzenia pliku:

```
dscontrol advisor start myping numer_portu
```

Numer portu określony w tej komendzie to port, który zostanie użyty przez doradcę do otwarcia połączenia z serwerem docelowym.

Wymagane procedury

Podobnie jak wszyscy inni doradcy, również niestandardowy doradca rozszerza funkcje klasy bazowej doradcy o nazwie ADV_Base. Klasa bazowa doradcy wykonuje większość funkcji doradcy, takich jak raportowanie wartości obciążenia do menedżera w celu ich użycia w algorytmie wagi menedżera. Klasa bazowa doradcy jest także odpowiedzialna za operacje połączenia i zamknięcia gniazda, a także udostępnia metody wysyłania i odbioru używane przez doradcę. Doradca jest używany tylko do wysyłania i odbierania poprzez określony port danych dotyczących badanego serwera. Czas wykonywania metod TCP udostępnianych w ramach klasy bazowej doradcy jest mierzony, co pozwala na obliczanie obciążenia. Opcja konstruktora klasy bazowej doradcy umożliwia w razie konieczności zastąpienie istniejącej wartości obciążenia nową wartością zwróconą z doradcy.

Uwaga: Na podstawie wartości ustawionej w konstruktorze klasa bazowa doradcy dostarcza wartość obciążenia do algorytmu wagi w określonych odstępach czasu. Jeśli doradca nie zakończył przetwarzania i nie może zwrócić poprawnej wartości obciążenia, klasa bazowa doradcy używa wcześniej zgłoszonej wartości obciążenia.

Doradcy zawierają następujące metody klasy bazowej:

- Procedura konstruktora. Konstruktor wywołuje konstruktor klasy bazowej.
- Metoda ADV_AdvisorInitialize. Ta metoda umożliwia wykonanie dodatkowych czynności po zakończeniu inicjowania klasy bazowej.
- Procedura getLoad. Klasa bazowa doradcy otwiera gniazdo. Funkcja getLoad musi jedynie wysłać odpowiednie żądania wysyłania i odbioru w celu zakończenia cyklu doradcy.

Szczegółowe informacje dotyczące wymaganych procedur są przedstawione w dalszej części tej sekcji.

Kolejność wyszukiwania

Niestandardowi doradcy są wywoływani po zakończeniu wyszukiwania standardowych (rodzimych) doradców. Jeśli komponent System równoważenia obciążenia nie znajdzie określonego doradcy na liście standardowych doradców, sprawdza listę doradców niestandardowych. Aby uzyskać dodatkowe informacje na temat używania doradców, patrz *Podręcznik administrowania komponentem System równoważenia obciążenia produktu WebSphere Application Server*.

Nazewnictwo i ścieżka do pliku

Poniżej przedstawiono wymagania wobec nazw i ścieżek doradców niestandardowych.

- Nazwa niestandardowego doradcy musi zawierać wyłącznie małe litery, aby wyeliminować kwestię rozróżniania wielkości liter, kiedy operator wpisuje komendy w wierszu komend. Nazwa doradcy musi zawierać przedrostek doradcy (ADV_).
- Klasa niestandardowego doradcy musi znajdować się w podkatalogu lib/CustomAdvisors. Domyślne położenie tego podkatalogu to /opt/ibm/edge/lb/servers/lib/CustomAdvisors w systemach Linux i UNIX oraz C:\Program Files\IBM\edge\lb\servers\lib\CustomAdvisors\ w systemach Windows.

Metody i wywołania funkcji niestandardowego doradcy

Konstruktor (udostępniany przez klasę bazową doradcy)

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

sName
Nazwa niestandardowego doradcy.

sVersion
Wersja niestandardowego doradcy.

iDefaultPort
Numer portu, poprzez który należy kontaktować się z serwerem, jeśli w wywołaniu nie określono żadnego numeru portu.

iInterval
Odstęp czasu między kolejnymi zapytaniami do serwerów wykonywanymi przez doradcę.

sDefaultLogFileName
Ten parametr jest wymagany, ale nie jest używany. Jedyną akceptowaną wartością jest łańcuch pusty "".

replace
Określa, czy doradca działa w trybie *zastępowania*. Możliwe wartości to:

- true – powoduje zastąpienie wartości obciążenia obliczonej przez kod bazowy doradcy wartością zwróconą przez niestandardowego doradcę.
- false – powoduje dodanie wartości obciążenia zgłoszonej przez niestandardowego doradcę do wartości obciążenia obliczonej przez kod bazowy doradcy.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

Metoda ta jest udostępniana w celu wykonywania inicjowania, które może być wymagane przez niestandardowego doradcę. Jest ona wywoływana po uruchomieniu podstawowego modułu doradcy.

W wielu sytuacjach, także w przypadku standardowych doradców, metoda ta nie jest używana, a jej kod zawiera tylko instrukcję *return*. Metody tej można używać do wywoływania metody suppressBaseOpeningSocket, której wywołanie jest poprawne tylko w tej metodzie.

getLoad()

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

iConnectTime
Czas (w milisekundach) potrzebny do zakończenia połączenia. Ta metoda pomiaru obciążenia jest wykonywana przez kod bazowy doradcy, a jej wynik jest przekazywany do kodu niestandardowego doradcy, który może go użyć lub zignorować przy zwracaniu wartości obciążenia. Jeśli nawiązanie połączenia nie powiedzie się, wartość ta jest ustawiana na -1.

caller
Instancja klasy bazowej doradcy, w której udostępnione są metody bazowe doradcy.

Wywołania funkcji dostępne dla niestandardowych doradców

Metody lub funkcje opisane w kolejnych sekcjach mogą być wywoływane przez niestandardowych doradców. Metody te są obsługiwane przez kod bazowy doradcy.

Niektóre z tych wywołań funkcji można wykonywać bezpośrednio, na przykład *nazwa_funkcji*(), ale inne wymagają przedrostka caller. Przedrostek *caller* oznacza instancję klasy bazowej doradcy obsługującą doradcę niestandardowego, który jest wykonywany.

ADVLOG()

Funkcja ADVLOG pozwala niestandardowemu doradcy na zapisywanie komunikatów tekstowych w pliku dziennika klasy bazowej doradcy. Stosowany jest następujący format:

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

logLevel
Poziom statusu, z którym komunikat jest zapisywany w pliku dziennika. Plik dziennika doradcy jest podzielony na sekcje. Najpilniejsze komunikaty otrzymują poziom statusu 0, natomiast mniej pilne komunikaty otrzymują wyższe wartości. Najbardziej szczegółowy typ komunikatu otrzymuje poziom statusu 5. Poziomy te służą do określania, które typy komunikatów będą odbierane przez użytkownika w czasie rzeczywistym. Do ustawiania szczegółowości służy komenda **dscontrol**. Bardzo poważne błędy powinny być zawsze rejestrowane na poziomie 0.

message
Komunikat, który zostanie zapisany w pliku dziennika. Wartość tego parametru jest standardowym łańcuchem języka Java.

getAdvisorName()

Funkcja getAdvisorName zwraca łańcuch języka Java zawierający przyrostek nazwy niestandardowego doradcy. Na przykład dla doradcy o nazwie ADV_cdload.java funkcja ta zwróci wartość cdload.

Funkcja ta nie ma żadnych parametrów.

Nie jest możliwa zmiana tej wartości w czasie jednego procesu tworzenia instancji doradcy.

getAdviseOnPort()

Funkcja getAdviseOnPort zwraca numer portu, na którym działa wywołujący ją doradca niestandardowy. Kod powrotu ma wartość będącą liczbą całkowitą (int) języka Java, a funkcja ta nie ma żadnych parametrów.

Nie jest możliwa zmiana tej wartości w czasie jednego procesu tworzenia instancji doradcy.

caller.getCurrentServer()

Funkcja getCurrentServer zwraca adres IP bieżącego serwera. Kod powrotu ma wartość będącą łańcuchem języka Java w formacie adresu IP, na przykład 128.0.72.139.

Zwykle adres ten zmienia się przy każdym wywołaniu niestandardowego doradcy, ponieważ kod bazowy doradcy odpytuje kolejno wszystkie serwery.

Funkcja ta nie ma żadnych parametrów.

caller.getCurrentCluster()

Wywołanie funkcji getCurrentCluster zwraca adres IP bieżącego klastra serwerów. Kod powrotu ma wartość będącą łańcuchem języka Java w formacie adresu IP, na przykład 128.0.72.139.

Zwykle adres ten zmienia się przy każdym wywołaniu niestandardowego doradcy, ponieważ kod bazowy doradcy odpytuje kolejno wszystkie klastry serwerów.

Funkcja ta nie ma żadnych parametrów.

getInterval()

Funkcja getInterval zwraca odstęp czasu dla doradcy, czyli liczbę sekund między kolejnymi cyklami doradcy. Wartość ta jest równa wartości domyślnej ustawionej w konstruktorze niestandardowego doradcy, chyba że wartość ta została zmodyfikowana w czasie wykonywania za pomocą komendy **dscontrol**.

Kod powrotu ma wartość będącą liczbą całkowitą (int) języka Java. Funkcja ta nie ma żadnych parametrów.

caller.getLatestLoad()

Funkcja getLatestLoad umożliwia niestandardowemu doradcy uzyskanie ostatniej wartości obciążenia dla danego obiektu serwera. Wartości obciążenia są przechowywane w wewnętrznych tabelach przez kod bazowy doradcy i demona menedżera.

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

Trzy argumenty definiują wspólnie jeden obiekt serwera.

cluster_IP
: Adres IP klastra obiektu serwera, dla którego zostanie uzyskana bieżąca wartość obciążenia. Argument ten musi być łańcuchem języka Java w formacie adresu IP, na przykład 245.145.62.81.

port
: Numer portu obiektu serwera, dla którego zostanie uzyskana bieżąca wartość obciążenia.

server_IP
: Adres IP obiektu serwera, dla którego zostanie uzyskana bieżąca wartość obciążenia. Argument ten musi być łańcuchem języka Java w formacie adresu IP, na przykład 192.255.201.3.

Kod powrotu ma wartość będącą liczbą całkowitą.

- Dodatnia wartość zwracana wskazuje rzeczywistą wartość obciążenia przypisaną do odpytywanego obiektu.
- Wartość -1 wskazuje, że wybrany serwer jest wyłączony.
- Wartość -2 wskazuje, że status wybranego serwera jest nieznany.

To wywołanie funkcji jest przydatne w sytuacji, gdy zachowanie określonego protokołu lub portu powinno być zależne od zachowania innego protokołu lub portu. Tego wywołania

funkcji można użyć na przykład w niestandardowym doradcy, który wyłącza określony serwer aplikacji, jeśli wyłączony jest serwer Telnet na tym samym komputerze.

caller.receive()

Funkcja receive pobiera informacje z połączenia gniazda.

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

Parametr *response* jest buforem łańcuchowym, w którym umieszczane są pobrane dane. Dodatkowo funkcja ta zwraca liczbę całkowitą, która ma następujące znaczenie:

- Wartość 0 wskazuje, że dane zostały pomyślnie wysłane.
- Ujemna wartość wskazuje błąd.

caller.send()

Funkcja send używa nawiązanego połączenia gniazda do wysłania pakietu danych do serwera poprzez określony port.

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

Parametr *command* to łańcuch zawierający dane do wysłania do serwera. Funkcja ta zwraca liczbę całkowitą, która ma następujące znaczenie:

- Wartość 0 wskazuje, że dane zostały pomyślnie wysłane.
- Ujemna wartość wskazuje błąd.

suppressBaseOpeningSocket()

Wywołanie funkcji suppressBaseOpeningSocket umożliwia niestandardowemu doradcy określenie, czy kod bazowy doradcy otwiera gniazdo TCP do serwera w imieniu niestandardowego doradcy. Jeśli doradca nie korzysta z bezpośredniej komunikacji z serwerem w celu określenia jego statusu, otwarcie tego gniazda może nie być konieczne.

To wywołanie funkcji może być wykonane tylko raz i to tylko w procedurze ADV_AdvisorInitialize.

Funkcja ta nie ma żadnych parametrów.

Przykłady

Poniższe przykłady przedstawiają sposoby implementacji niestandardowych doradców.

Standardowy doradca

Ten przykładowy kod źródłowy przypomina standardowego doradcę HTTP komponentu System równoważenia obciążenia. Poniżej przedstawiono sposób jego działania:

1. Wysyłana jest komenda ″HEAD/HTTP″, czyli żądanie wysłania.
2. Odbierana jest odpowiedź. Informacje nie są analizowane, ale odpowiedź powoduje zakończenie metody getLoad.
3. Metoda getLoad zwraca wartość 0, aby wskazać sukces, lub wartość -1, aby wskazać niepowodzenie.

Ten doradca działa w trybie normalnym, przez co pomiar obciążenia jest oparty na ilości czasu (w milisekundach) wymaganego do wykonania operacji otwarcia gniazda, wysłania, odbioru i zamknięcia gniazda.

```
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";

//--------
// Konstruktor

  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;                                  // Zwykle pusta procedura
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;       // Inicjowanie jako niedostępnego

    iRc = caller.send(ADV_SEND_REQUEST);     // Wysłanie żądania HTTP
                                               // z serwera
    if (0 <= iRc) {                          // Jeśli wysyłanie powiodło się,
      StringBuffer sbReceiveData = new StringBuffer("");   // przydzielenie buforu
                                                           // na odpowiedź
      iRc = caller.receive(sbReceiveData);   // Odebranie wyniku

      // Analiza wyniku, jeśli jest to konieczne

      if (0 <= iRc) {          // Jeśli odbieranie powiodło się,
        iLoad = 0;             // zwrócenie wartości 0 oznaczającej sukces
      }                        // (wartość obciążenia doradcy jest ignorowana
    }                          //  przez kod bazowy w trybie normalnym)
    return iLoad;
  }
}
```

Doradca strumienia bocznego

Ten przykład przedstawia sposób pominięcia standardowego gniazda otwartego przez klasę bazową doradcy. Doradca otwiera zamiast niego gniazdo strumienia bocznego Java w celu odpytania serwera. Ta procedura może być przydatna w przypadku serwerów używających do nasłuchu zapytań doradcy innego portu niż dla normalnego ruchu klientów.

W tym przykładzie serwer nasłuchuje na porcie 11999, a po zapytaniu zwraca wartość obciążenia w postaci szesnastkowej liczby całkowitej ″4″. Przykład ten działa w trybie zastępowania, co oznacza, że ostatni parametr konstruktora jest ustawiany na wartość ″true″, a kod bazowy doradcy używa zwróconej wartości obciążenia, a nie ilości czasu, jaki upłynął.

Należy zwrócić uwagę na wywołanie metody supressBaseOpeningSocket() w procedurze inicjowania. Pominięcie gniazda bazowego nie jest wymagane, jeśli nie będą wysyłane żadne dane. Gniazdo można otworzyć na przykład w celu upewnienia się, że doradca może skontaktować się z serwerem. Przed dokonaniem tego wyboru należy dokładnie zbadać wymagania aplikacji.

```
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;

  // Utworzenie tablicy bajtowej z komunikatem żądania ładowania
  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);         // Parametr trybu zastępowania ma wartość "true"
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // Informacja dla kodu bazowego, aby nie otwierał
                                   // gniazda standardowego
      return;
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;   // Port służący do komunikacji z serwerem

    string sServer = caller.getCurrentServer();   // Adres serwera do odpytania
    try {
      socket soServer = new Socket(sServer, iControlPort);  // Otwarcie gniazda
                                                            // do serwera
      DataInputStream disServer = new DataInputStream(
                                      soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                       soServer.getOutputStream());

      int iRecvTimeout = 10000;  // Ustawienie limitu czasu (w milisekundach)
                                 // dla odbioru danych
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // Wysłanie komunikatu do serwera
      dosServer.flush();

      iLoad = disServer.readByte();   // Odebranie odpowiedzi z serwera

    } catch (exception e) {
      system.out.println("Wychwycono wyjątek " + e);
    }
    return iLoad;    // Zwrócenie obciążenia zgłoszonego przez serwer
  }
}
```

Doradca dwuportowy

Ten przykład niestandardowego doradcy przedstawia możliwość wykrywania awarii portu serwera na podstawie jego statusu oraz statusu innego demona serwera, który działa na innym porcie na tym samym serwerze. Jeśli na przykład demon HTTP na porcie 80 przestaje odpowiadać, można również zatrzymać ruch związany z routingiem do demona SSL na porcie 443.

Ten doradca jest bardziej agresywny niż standardowi doradcy, ponieważ każdy serwer, który nie wysyła odpowiedzi, jest uznawany za niedziałający, a w konsekwencji oznaczany jako *wyłączony*. Standardowi doradcy przyjmują, że nie odpowiadające serwery są bardzo wolne. Ten doradca powoduje oznaczenie serwera jako wyłączonego zarówno dla portu HTTP, jak i dla portu SSL, na podstawie braku odpowiedzi z jednego z tych portów.

Aby użyć tego niestandardowego doradcy, administrator uruchamia jego dwie instancje - dla portu HTTP i dla portu SSL. Doradca tworzy instancje dwóch statycznych, globalnych tabel mieszających dla każdego z tych portów. Każdy doradca próbuje nawiązać komunikację z odpowiednim demonem serwera i umieszcza wyniki tego zdarzenia we własnej tabeli mieszającej. Wartość zwracana przez każdego doradcę do klasy bazowej doradcy jest zależna zarówno od możliwości komunikacji z własnym demonem serwera, jak i od możliwości komunikacji drugiego doradcy z jego demonem.

Użyte zostały następujące metody niestandardowe.

- ADV_nte() to prosty obiekt-kontener przechowujący informacje o serwerze. Obiekty te są przechowywane w tabeli mieszającej jako elementy tabeli. Każdy obiekt ma znacznik czasu służący do określenia, czy element jest aktualny.
- putNte() i getNte() to zsynchronizowane metody, które sprawiają, że dwie instancje doradcy uzyskują dostęp do tabeli mieszającej w sposób nadzorowany.
- getLoadHTTP() to metoda sprawdzająca możliwość reakcji serwera HTTP. Jest to procedura niskiego poziomu, która nie gromadzi ani nie używa informacji dotyczących protokołu SSL.
- getLoadSSL() to metoda sprawdzająca możliwość reakcji serwera SSL. Jest to procedura niskiego poziomu, która nie gromadzi ani nie używa informacji dotyczących protokołu HTTP.
- getLoad() to procedura punktu wejścia dla tego niestandardowego doradcy. Obsługuje ona obydwa protokoły i może zapisywać oraz pobierać informacje z tabeli mieszającej. Jest to procedura łącząca oba porty.

Wykrywane są następujące warunki błędu.

- Serwer nie odpowiada — klasy bazowe doradcy wysyłają okresowo sygnał ping na adres serwera. Jeśli ten adres jest nieosiągalny, klasy bazowe doradcy oznaczają serwer jako wyłączony. Nie jest wywoływana żadna z dwóch instancji niestandardowego doradcy, a obydwa serwery na tym komputerze są oznaczane jako wyłączone.
- Jeden z demonów na serwerze przestaje odpowiadać, ale drugi demon ciągle działa — kiedy kod bazowy próbuje otworzyć gniazdo z serwerem, połączenie zostaje odrzucone, a doradca bazowy dla tego protokołu oznacza serwer jako wyłączony. Kod niestandardowego doradcy dla tego protokołu nie jest wywoływany. Mimo iż niestandardowy doradca drugiego protokołu kontynuuje komunikację ze swoim serwerem, uzyskuje z tabeli mieszającej informację, że drugi doradca niestandardowy nie może komunikować się ze swoim demonem serwera. Z tej przyczyny również doradca drugiego protokołu oznacza swój serwer jako wyłączony.
- Jeden z demonów nie wysyła odpowiedzi, ale drugi demon odpowiada — niestandardowy doradca niedostępnego protokołu wykrywa brak możliwości komunikacji, oznacza serwer

jako wyłączony i zapisuje dane w tabeli mieszającej. Niestandardowy doradca drugiego portu uzyskuje tę informację z tabeli mieszającej i oznacza swój serwer jako wyłączony.

Przykład ten został utworzony w celu powiązania portu 80 dla protokołu HTTP i portu 443 dla protokołu SSL, ale można dostosować go do dowolnej kombinacji portów.

```
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;

//--------
// Zdefiniowanie elementu tabeli dla tabel mieszających
// używanych przez tego niestandardowego doradcę

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

//--------
  // Konstruktor

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

//--------
// Sprawdzenie, czy ten element jest aktualny czy też utracił ważność
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval(); // Ustawienie czasu życia
                                                      // na trzy cykle doradcy
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

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

//--------
// Akcesory wartości

  public int getLoadValue() { return iLoad; }

//--------
// Klonowanie (pozwala uniknąć uszkodzenia między wątkami)

  public synchronized Object Clone() {
    try {
      return super.clone();
```

```
    } catch (cloneNotSupportedException e) {
      return null;
    }
  }

}

//--------
// Zdefiniowanie niestandardowego doradcy

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;

  //--------
  // Zdefiniowanie tabel do przechowywania informacji o historii
  // dla poszczególnych portów

  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";

  //--------
  // Utworzenie tablicy bajtowej z komunikatem powitania klienta SSL

  public static final byte[] abClientHello = {
    (byte)0x80, (byte)0x1c,
    (byte)0x01,               // Powitanie klienta
    (byte)0x03, (byte)0x00,   // Wersja SSL
    (byte)0x00, (byte)0x03,   // Długość specyfikacji szyfru (w bajtach)
    (byte)0x00, (byte)0x00,   // Długość identyfikatora sesji (w bajtach)
    (byte)0x00, (byte)0x10,   // Długość danych wezwania (w bajtach)
    (byte)0x00, (byte)0x00, (byte)0x03,   // Specyfikacja szyfru
    (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)Ox20,  // Dane wezwania
    (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 };

  //--------
  // Konstruktor

  public ADV_twop() {
    super(ADV_TWOP_NAME, VERSION, ADV_TWOP_DEF_ADV_ON_PORT,
          ADV_TWOP_DEF_INTERVAL, "",
          false);    // false = komponent System równoważenia obciążenia
                     //         mierzy czas odpowiedzi
    setAdvisor ( this );
  }

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

  public void ADV_AdvisorInitialize() {
      return;
  }

  //--------
  // Zsynchronizowane procedury dostępu PUT i GET dla tabel mieszających
```

```
 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 - określenie obciążenia serwera HTTP
 //               na podstawie odpowiedzi serwera

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

   int iRc = caller.send(ADV_HTTP_REQUEST_STRING); // Wysłanie komunikatu
                                                   // żądania do serwera
   if (0 <= iRc) {    // Czy żądanie zwróciło niepowodzenie?
     StringBuffer sbReceiveData = new StringBuffer("") // Przydzielenie buforu
                                                       // na odpowiedź
     iRc = caller.receive(sbReceiveData); // Uzyskanie odpowiedzi z serwera

     if (0 <= iRc) {     // Czy metoda receive zwróciła niepowodzenie?
       if (0 < sbReceiveData.length()) {    // Czy istnieją dane?
         iLoad = SUCCESS;        // Zignorowanie pobranych danych
                                 // i zwrócenie kodu sukcesu
       }
     }
   }
   return iLoad;
 }

 //--------
 // getLoadSSL() - określenie obciążenia serwera SSL
 //                na podstawie odpowiedzi serwera

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

   int iSocket = caller.getAdvisorSocket();   // Wysłanie żądania szesnastkowego
                                              // do serwera
   CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                                 abClientHello);
   int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

   if (0 <= iRc) {    // Czy żądanie zwróciło niepowodzenie?
     StringBuffer sbReceiveData = new StringBuffer(""); // Przydzielenie buforu
                                                        // na odpowiedź
     iRc = caller.receive(sbReceiveData);     // Uzyskanie odpowiedzi
                                              // z serwera
     if (0 <= iRc) {     // Czy metoda receive zwróciła niepowodzenie?
       if (0 < sbReceiveData.length()) {    // Czy istnieją dane?
         iLoad = SUCCESS;   // Zignorowanie pobranych danych
                            // i zwrócenie kodu sukcesu
       }
     }
   }
   return iLoad;
 }

 //--------
 // getLoad - scalenie wyników metod HTTP i SSL
```

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

  String sCluster = caller.getCurrentCluster(); // Adres bieżącego klastra
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;  // Klucz tabeli mieszającej

  if (ADV_TWOP_PORT_HTTP == iPort) {              // Obsługa serwera HTTP
    iLoadHTTP = getLoadHTTP(iConnectTime, caller); // Uzyskanie obciążenia
                                                   // serwera HTTP

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);  // Zapisanie informacji
                                                    // o obciążeniu
                                                    //serwera SSL
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey); // Uzyskanie
                                                         // informacji
                                                         // dotyczących
                                                         // serwera SSL
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) { // Sprawdzenie znacznika czasu
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) { // Czy serwer
                                                              // SSL działa?
      iLoad = iLoadHTTP;
        } else {    // Serwer SSL nie działa, należy więc oznaczyć
                    // serwer HTTP jako wyłączony
          iLoad= ADV_HOST_INACCESSIBLE;
        }
      } else {      // Informacje o serwerze SSL utraciły ważność,
                    // należy więc oznaczyć serwer HTTP jako wyłączony
    iLoad = ADV_HOST_INACCESSIBLE;
      }
    } else {        // Brak informacji o obciążeniu serwera SSL,
                    // zgłoszenie wyników metody getLoadHTTP()
      iLoad = iLoadHTTP;
    }
  }
  else if (ADV_TWOP_PORT_SSL == iPort) {         // Obsługa serwera SSL
    iLoadSSL = getLoadSSL(iConnectTime, caller); // Uzyskanie obciążenia
                                                 // serwera SSL

    ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
    putNte(htTwopSSL, "SSL", sHashKey, nteSSL); // Zapisanie informacji
                                                // o obciążeniu
                                                // serwera SSL

    ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey); // Uzyskanie
                                                           // informacji
                                                           // dotyczących
                                                           // serwera SSL
    if (null != nteHTTP) {
      if (true == nteHTTP.isCurrent(this)) { // Sprawdzenie znacznika czasu
        if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) { // Czy serwer
                                                               // HTTP
                                                               // działa?
      iLoad = iLoadSSL;
        } else {   // Serwer HTTP nie działa, należy więc oznaczyć
                   // serwer SSL jako wyłączony
    iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {     // Informacje z serwera HTTP utraciły ważność,
                   // należy więc oznaczyć serwer SSL jako wyłączony
```

```
      iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {       // Brak informacji o obciążeniu serwera HTTP,
                     // zgłoszenie wyników metody getLoadSSL()
        iLoad = iLoadSSL;
      }
    }

  //--------
  // procedura obsługi błędów

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

Doradca serwera WebSphere Application Server

Przykładowy doradca niestandardowy dla serwera WebSphere Application Server znajduje się w katalogu *ścieżka_instalacji*/servers/samples/CustomAdvisors/. Pełny kod tego doradcy nie zostanie powtórzony w niniejszym dokumencie.

- ADV_was.java to plik kodu źródłowego doradcy, który jest kompilowany i uruchamiany na komputerze komponentu System równoważenia obciążenia.
- LBAdvisor.java.servlet to plik zawierający kod źródłowy serwletu, którego nazwę należy zmienić na LBAdvisor.java, a następnie skompilować i uruchomić na komputerze serwera WebSphere Application Server.

Pełny doradca jest trochę bardziej złożony niż ten przykład. Doradca dodaje specjalną procedurę analizowania, która jest bardziej zwarta niż przedstawiony powyżej przykład StringTokenizer.

Bardziej skomplikowana część kodu przykładowego znajduje się w serwlecie Java. Oprócz innych metod, serwlet zawiera dwie metody wymagane przez specyfikację serwletu: init() i service(), a także metodę run() wymaganą przez klasę Java.lang.thread.

- Metoda init() jest wywoływana jeden raz przez mechanizm serwletu w czasie inicjowania. Metoda ta tworzy wątek o nazwie _checker, który działa niezależnie od wywołań doradcy i usypia na pewien czas przed wznowieniem pętli przetwarzania.
- Metoda service() jest wywoływana przez mechanizm serwletu przy każdym wywołaniu serwletu. W tym przypadku metoda ta jest wywoływana przez doradcę. Metoda service() wysyła strumień znaków ASCII do strumienia wyjściowego.
- Metoda run() zawiera najważniejszą część wykonywanego kodu. Jest ona wywoływana przez metodę start(), która z kolei jest wywoływana w metodzie init().

Odpowiednie fragmenty kodu serwletu zostały przedstawione poniżej.

```
...

  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;
}

...
```

Korzystanie z danych zwróconych przez doradców

Niezależnie od tego, czy używane jest standardowe wywołanie istniejącej części serwera aplikacji, czy też dodawany jest fragment kodu, który będzie odpowiednikiem niestandardowego doradcy po stronie serwera, prawdopodobnie konieczne będzie zbadanie zwróconych wartości obciążenia i zmodyfikowanie zachowania serwera. Klasa Java StringTokenizer i jej powiązane metody ułatwiają proces analizowania danych.

Typowa komenda HTTP może mieć postać GET /index.html HTTP/1.0

Typowa odpowiedź na tę komendę może mieć następującą postać.

```
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: pl

<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 3.2 Final//EN">
<HTML><HEAD><TITLE>Strona testowa</TITLE></HEAD>
<BODY><H1>Serwer Apache</H1>
<HR>
<P><P>Na tym serwerze WWW działa program Apache 1.3.12.
<P><HR>
<P><IMG SRC="apache_pb.gif" ALT="">
</BODY></HTML>
```

Najbardziej interesujące informacje, a w szczególności kod powrotu HTTP, są zawarte w pierwszym wierszu.

Specyfikacja protokołu HTTP klasyfikuje kody powrotu w następujący sposób:

- Kody powrotu 2xx wskazują sukces.
- Kody powrotu 3xx wskazują przekierowania.
- Kody powrotu 4xx wskazują błędy klienta.
- Kody powrotu 5xx wskazują błędy serwera.

Jeśli dokładnie wiadomo, jakie kody mogą zostać zwrócone przez serwer, zastosowany kod nie musi być aż tak szczegółowy jak w tym przykładzie. Należy jednak pamiętać, że ograniczenie wykrywanych kodów powrotu może wpłynąć negatywnie na elastyczność programu w przyszłości.

Poniższy przykład przedstawia autonomiczny program Java zawierający podstawowego klienta HTTP. Przykład wywołuje prosty analizator składni ogólnego przeznaczenia służący do badania odpowiedzi 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("Wersja HTTP: " + sHTTPVersion);
      } else {
        System.out.println("Niepoprawna wersja HTTP: " + sHTTPVersion);
      }
      } else {
      System.out.println("Nic nie zwrócono");
      return;
    }

    if (true == st.hasMoreTokens()) {
      sHTTPReturnCode = st.nextToken();
```

```
    try {
        iRc = Integer.parseInt(sHTTPReturnCode);
      } catch (NumberFormatException ne) {;}

      switch (iRc) {
      case(200):
        System.out.println("Kod odpowiedzi HTTP: OK, " + iRc);
        break;
      case(400): case(401): case(402): case(403): case(404):
        System.out.println("Kod odpowiedzi HTTP: Błąd klienta, " + iRc);
        break;
      case(500): case(501): case(502): case(503):
        System.out.println("Kod odpowiedzi HTTP: Błąd serwera, " + iRc);
        break;
      default:
        System.out.println("Kod odpowiedzi HTTP: Nieznany, " + iRc);
        break;
      }
    }

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

Uwagi

Pierwsza edycja (maj 2006)

Niniejsza publikacja została przygotowana z myślą o produktach i usługach oferowanych w Stanach Zjednoczonych.

IBM może nie oferować w innych krajach produktów, usług lub opcji, omawianych w tej publikacji. Informacje o produktach i usługach dostępnych w danym kraju można uzyskać od lokalnego przedstawiciela IBM. Odwołanie do produktu, programu lub usługi IBM nie oznacza, że można użyć wyłącznie tego produktu, programu lub usługi. Zamiast nich można zastosować ich odpowiednik funkcjonalny pod warunkiem, że nie narusza to praw własności intelektualnej IBM. Jednakże cała odpowiedzialność za ocenę przydatności i sprawdzenie działania produktu, programu lub usługi pochodzących od producenta innego niż IBM spoczywa na użytkowniku.

IBM może posiadać patenty lub złożone wnioski patentowe na towary i usługi, o których mowa w niniejszej publikacji. Przedstawienie niniejszej publikacji nie daje żadnych uprawnień licencyjnych do tychże patentów. Pisemne zapytania w sprawie licencji można przesyłać na adres:

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

Zapytania w sprawie licencji na informacje dotyczące zestawów znaków dwubajtowych (DBCS) należy kierować do lokalnych działów własności intelektualnej IBM (IBM Intellectual Property Department) lub zgłaszać na piśmie pod adresem:

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

Poniższy akapit nie obowiązuje w Wielkiej Brytanii, a także w innych krajach, w których jego treść pozostaje w sprzeczności z przepisami prawa miejscowego:

INTERNATIONAL BUSINESS MACHINES CORPORATION DOSTARCZA NINIEJSZY DOKUMENT W STANIE, W JAKIM SIĘ ZNAJDUJE (“AS IS”), BEZ UDZIELANIA JAKICHKOLWIEK GWARANCJI (W TYM TAKŻE RĘKOJMI), WYRAŹNYCH LUB DOMNIEMANYCH, A W SZCZEGÓLNOŚCI DOMNIEMANYCH GWARANCJI PRZYDATNOŚCI HANDLOWEJ LUB PRZYDATNOŚCI DO OKREŚLONEGO CELU. Ustawodawstwa niektórych krajów nie dopuszczają zastrzeżeń dotyczących gwarancji wyraźnych lub domniemanych w odniesieniu do pewnych transakcji; w takiej sytuacji powyższe zdanie nie ma zastosowania.

Informacje zawarte w niniejszej publikacji mogą zawierać nieścisłości techniczne lub błędy drukarskie. Informacje te są okresowo aktualizowane, a zmiany te zostaną uwzględnione w kolejnych wydaniach tego dokumentu. IBM zastrzega sobie prawo do wprowadzania ulepszeń i/lub zmian w produktach i/lub programach opisanych w tej publikacji w dowolnym czasie, bez wcześniejszego powiadomienia.

Wszelkie wzmianki w tej publikacji na temat stron internetowych innych podmiotów zostały wprowadzone wyłącznie dla wygody użytkowników i w żadnym wypadku nie stanowią zachęty do ich odwiedzania. Materiały dostępne na tych stronach nie są częścią materiałów opracowanych dla tego produktu IBM, a użytkownik korzysta z nich na własną odpowiedzialność.

IBM ma prawo do korzystania i rozpowszechniania informacji przysłanych przez użytkownika w dowolny sposób, jaki uzna za właściwy, bez żadnych zobowiązań wobec ich autora.

Licencjobiorcy tego programu, którzy chcieliby uzyskać informacje na temat programu w celu: (i) wdrożenia wymiany informacji między niezależnie utworzonymi programami i innymi programami (łącznie z tym opisywanym) oraz (ii) wspólnego wykorzystywania wymienianych informacji, powinni skontaktować się z:

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

Informacje takie mogą być udostępnione, o ile spełnione zostaną odpowiednie warunki, w tym, w niektórych przypadkach, uiszczenie odpowiedniej opłaty.

Licencjonowany program opisany w niniejszej publikacji oraz wszystkie inne licencjonowane materiały dostępne dla tego programu są dostarczane przez IBM na warunkach określonych w Międzynarodowej Umowie Licencyjnej IBM na Program lub w innych podobnych umowach zawartych między IBM i użytkownikami.

Wszelkie dane dotyczące wydajności zostały zebrane w kontrolowanym środowisku. W związku z tym rezultaty uzyskane w innych środowiskach operacyjnych mogą się znacząco różnić. Niektóre pomiary mogły być dokonywane na systemach będących w fazie rozwoju i nie ma gwarancji, że pomiary te wykonane na ogólnie dostępnych systemach dadzą takie same wyniki. Niektóre z pomiarów mogły być estymowane przez ekstrapolację. Rzeczywiste wyniki mogą być inne. Użytkownicy powinni we własnym zakresie sprawdzić odpowiednie dane dla ich środowiska.

Informacje dotyczące produktów innych podmiotów uzyskano od dostawców tych produktów, z opublikowanych zapowiedzi lub innych powszechnie dostępnych źródeł. IBM nie testował tych produktów i nie może potwierdzić dokładności pomiarów wydajności, kompatybilności ani żadnych innych danych związanych z tymi produktami. Pytania dotyczące możliwości produktów innych podmiotów należy kierować do dostawców tych produktów.

Wszelkie stwierdzenia dotyczące przyszłych kierunków rozwoju i zamierzeń IBM mogą zostać zmienione lub wycofane bez powiadomienia.

Publikacja ta zawiera przykładowe dane i raporty używane w codziennych operacjach działalności gospodarczej. W celu kompleksowego ich zilustrowania, podane przykłady mogą zawierać nazwiska osób prywatnych, nazwy przedsiębiorstw oraz nazwy produktów. Wszystkie te nazwy/nazwiska są fikcyjne i jakiekolwiek podobieństwo do istniejących nazw/nazwisk i adresów jest całkowicie przypadkowe.

W przypadku przeglądania niniejszych informacji w formie elektronicznej, zdjęcia i kolorowe ilustracje mogą nie być wyświetlane.

Znaki towarowe

Poniżej zostały wymienione znaki towarowe IBM Corporation w Stanach Zjednoczonych i/lub w innych krajach:

- AIX
- IBM
- ViaVoice
- WebSphere

Java i wszystkie znaki towarowe związane z Java są znakami towarowymi firmy Sun Microsystems, Inc. w Stanach Zjednoczonych i/lub w innych krajach.

Microsoft, Windows, Windows NT i logo Windows są znakami towarowymi firmy Microsoft Corporation w Stanach Zjednoczonych i/lub w innych krajach.

Intel, logo Intel Inside, MMX i Pentium są znakami towarowymi firmy Intel Corporation w Stanach Zjednoczonych i/lub w innych krajach.

UNIX jest zastrzeżonym znakiem towarowym grupy The Open Group w Stanach Zjednoczonych i w innych krajach.

Linux jest znakiem towarowym Linusa Torvaldsa w Stanach Zjednoczonych i/lub w innych krajach.

Pozostałe nazwy firm, produktów i usług mogą być znakami towarowymi lub znakami usług innych podmiotów.

Indeks

N

P

R

S

T

U

W

Z

Drukowane w USA

GC85-0194-00

Spine information:

WebSphere Application Server

Podręcznik programowania dla produktu Edge Components

Wersja 6.1

GC85-0194-00