WebSphere Application Server

Programozási útmutató az Edge összetevőkhöz

6.1 Változat

GC22-0411-00

WebSphere Application Server

Programozási útmutató az Edge összetevőkhöz

6.1 Változat

GC22-0411-00

Megjegyzés

A jelen információk és a megfelelő termék használata előtt tanulmányozza át a következő részben található általános információkat: “Megjegyzések” oldalszám: 59.

Első kiadás (2006. Május)

Ez a kiadás a következő termékhez készült:
WebSphere Application Server, 6.1 Változat

továbbá minden későbbi kiadáshoz és módosításhoz, amennyiben az újabb kiadások ettől eltérően nem jelzik.

A kiadványokat az IBM képviselőjétől vagy az IBM helyi irodájától rendelheti meg.

Tartalom

Ábrák

Néhány szó a könyvről

A dokumentumnak ebben a részében a *WebSphere Application Server programozási útmutató az Edge összetevőkhöz* célját, szervezését és jelöléseit ismertetjük.

Kinek ajánljuk a könyvet?

A könyv a WebSphere Application Server, 6.1 Változat Edge összetevők személyre szabására használható alkalmazás programozási felületeket ismerteti. Az információk célközönségét a bedolgozó alkalmazásokat és egyéb egyedi összetevőket készítő programozók adják. A hálózattervezők és a rendszergazdák érdeklődésére szintén számot tarthat a dokumentum, mint a személyre szabási lehetőségek típusainak összefoglalása.

A szükséges előismeretek

A könyv információinak használatához - az alkalmazni kíván alkalmazás programozási felülettől függően - szükség van a Java vagy a C programozási nyelvben alkalmazott programozási eljárások ismeretére. Az elérhetővé tett felületekben található adatszerkezetek és metódusok dokumentálva vannak, de tisztában kell lennie a saját alkalmazásának elkészítéséhez, saját rendszerén való lefordításához és teszteléséhez szükséges eljárásokkal. Bizonyos felületekhez mintakódokat is biztosítunk, ám a minták kizárólag példaként szolgálnak az Ön saját alkalmazásának összeállításához.

A könyvben alkalmazott jelölések és szakkifejezések

A jelen dokumentáció az alábbi szedési és jelölési konvenciókat alkalmazza.

1. táblázat: A könyvben alkalmazott jelölések

Jelölés	Jelentés
Félkövér	A grafikus felhasználói felületek elemeire való hivatkozáskor a félkövér szöveg menüt, menüelemet, címkét, gombot, ikont vagy mappát jelöl. Egyes esetekben olyan parancsnév kiemelésére szolgál, amelyet egyébként nehéz volna megkülönböztetni a körülötte található szövegtől.
Monospace betűtípus	A parancssorba beírandó szöveget ad meg. A képernyőn megjelenő szövegek, a kódminták és a fájlkivonatok szintén Monospace szedésűek.
Dőlt	Változóértéket jelöl, amelyet Önnek kell megadnia (például a *fájlnév* értékben meg kell adnia egy fájl nevét). A dőlt betűket kiemelésre és könyvek címének megadására is alkalmazzuk.
Ctrl-*x*	Ahol az *x* egy billentyű neve, a kombináció pedig egy Ctrl+karakter billentyűkombinációt jelöl. A Ctrl-c például azt jelenti, hogy lenyomva kell tartania a Ctrl gombot, miközben a c gombot is lenyomja.
Return	A Return vagy Enter szóval vagy balra mutató nyíllal jelölt billentyűre hivatkozik.
%	A Linux és UNIX parancssori héjat jelöli olyan parancsnál, amelynek futtatásához nincs szükség root jogokra.
#	A Linux és a UNIX parancssori héjat jelöli olyan parancsnál, amelynek futtatásához root jogosultságokra van szükség.
C:\	A Windows parancssorát jelöli.
Parancsok beírása	Ha parancs “beírására” vagy “kiadására” kap utasítást, akkor gépelje be a parancsot, majd nyomja meg a Return gombot. Az “Adja ki az **ls** parancsot” például azt jelenti, hogy gépelje be az **ls** karaktersorozatot a parancssorba, majd nyomja meg a Return gombot.
[ ]	Elhagyható elemeket határol a szintaxisok leírásában.

1. táblázat: A könyvben alkalmazott jelölések (Folytatás)

Jelölés	Jelentés
{ }	A szintaxisok leírásában listákat határol, amelyek elemei közül ki kell választania egyet.
|	A szintaxisok leírásában a lehetséges választások { } (kapcsos zárójelek) közé foglalt listájának elemeit választja el egymástól.
...	A szintaxisok leírásában a három pont azt jelzi, hogy a megelőző elemet egy vagy több alkalommal meg lehet ismételni. A példákban a három pont arra utal, hogy a tömörség megőrzése miatt a példából bizonyos információk kimaradtak.

Kisegítő lehetőségek

A kisegítő lehetőségek a fizikai fogyatékkal élőket, például a mozgásukban vagy látásukban korlátozottakat segítik a szoftvertermékek eredményes használatában. A WebSphere Application Server, 6.1 Változat fontosabb kisegítő lehetőségei a következők:

- Képernyőolvasó szoftver és digitális beszédszintetizátor, amely képes szóban átadni a képernyő tartalmát. Szükség esetén hangfelismerő szoftver is rendelkezésére áll, ilyen például az IBM ViaVoice, amelyet adatbevitelre és a felhasználói felületen való navigálásra használhat.
- Az egér helyett a billentyűzettel is működtetheti a szolgáltatásokat.
- Az Application Server szolgáltatásait a grafikus felület helyett általános szövegszerkesztővel vagy parancssori felületen keresztül is konfigurálhatja és adminisztrálhatja. Az egyes szolgáltatások kisegítő lehetőségeiről a szolgáltatások dokumentációjában talál részletesebb információkat.

Kapcsolódó dokumentumok és webhelyek

- *Fogalmak, tervezés és az Edge összetevők telepítése*, GC31-6918-00
- *Caching Proxy adminisztrációs útmutató*, GC31-6920-00
- *Load Balancer adminisztrációs útmutató*, GC31-6921-00
- Az IBM webhelyének kezdőlapja *www.ibm.com/*
- IBM WebSphere Application Server *www.ibm.com/software/webservers/appserv/*
- IBM WebSphere Application Server könyvtár webhely *www.ibm.com/software/webservers/appserv/library.html*
- IBM WebSphere Application Server támogatási webhely *www.ibm.com/software/webservers/appserv/support.html*
- IBM WebSphere Application Server információs központ *www.ibm.com/software/webservers/appserv/infocenter.html*
- IBM WebSphere Application Server Edge összetevők, információs központ *www.ibm.com/software/webservers/appserv/ecinfocenter.html*

Hogyan küldheti el megjegyzéseit?

Szeretnénk a legpontosabb és legjobb minőségű információkkal ellátni Önt, és ehhez fontosak számunkra az Ön visszajelzései. Ha bármilyen megjegyzése van a könyvvel vagy a WebSphere Application Server Edge összetevőkhöz készült dokumentációval kapcsolatban:

- Küldje el megjegyzéseit e-mailben a következő címre: fsdoc@us.ibm.com. Kérjük, adja meg a könyv címét és cikkszámát, a WebSphere Application Server változatát, valamint - ha lehetséges - annak a szövegrésznek a helyét (például oldal- vagy táblázatszámot), amelyre a megjegyzése vonatkozik.

1. fejezet Az Edge összetevők személyre szabásának áttekintése

Ez a könyv a WebSphere Application Server Edge összetevők személyre szabására használható alkalmazás programozási felületeket tárgyalja. (A WebSphere Application Server Edge összetevők elemei: Caching Proxy és Load Balancer.) A szoftver számos felületet magába foglal, ezekkel a rendszergazdák egyénre szabhatják a telepítéseket, módosíthatják az Edge összetevők közötti párbeszédek módját, illetve lehetővé tehetik az egyéb szoftverrendszerekkel folytatott párbeszédet.

FONTOS: A Caching Proxy az Edge összetevők minden telepítésekor elérhető, kivéve a következőket:

- A Caching Proxy nem érhető el az Edge összetevők Itanium 2 vagy 64 bites AMD Opteron processzorokon futó telepítéseiben.
- A Caching Proxy nem érhető el az Edge összetevők Load Balancer for IPv4 and IPv6 telepítéseiben.

A dokumentumban tárgyalt alkalmazás programozási felületek több kategóriába tartoznak.

Caching Proxy személyre szabása

A Caching Proxy számos a saját feldolgozási sorozatába illesztett felülettel rendelkezik, amelyek alkalmasak a szabványos feldolgozási műveletek bővítésére vagy egyedi eljárásokra való lecserélésére. A személyre szabás egyaránt jelentheti bizonyos feladatok módosítását vagy hozzáadását, mint például:

- Ügyfelek hitelesítése
- Kérések hitelesítése
- URL címek lefordítása fizikai fájlelérési útvonalakra
- Kérések kiszolgálása
- Naplózás
- Válaszadás hibahelyzetekre

Az egyedi alkalmazások, amelyeket Caching Proxy bedolgozóknak is nevezünk, a proxyszerver feldolgozási sorozatának meghatározott pontjain hívódnak meg.

A Caching Proxy alkalmazás programozási felülete bizonyos rendszerszolgáltatások megvalósításában is szerepet kapott. A proxyszerver LDAP-támogatása például beépülővel valósul meg.

2. fejezet, “Caching Proxy API”, oldalszám: 3 - A felület és a proxyszerver bedolgozó programok használatára való konfigurálásának részletes ismertetése.

Load Balancer személyre szabása

A Load Balancer saját tanácsadók készítésével szabható egyénre. A tanácsadók feladata a tényleges terhelések mérése a szervereken. Egyedi tanácsadóval saját, a rendszer jellemzőinek leginkább megfelelő módszereket valósíthat meg. Ez különösen akkor lehet fontos, ha egyénre szabott vagy saját fejlesztésű webkiszolgáló rendszerrel rendelkezik.

3. fejezet, “Egyéni tanácsadók”, oldalszám: 39- Részletes információk az egyéni tanácsadók készítéséről és használatáról. A szakasz minta-tanácsadókódot is tartalmaz.

Mintakódok

Az alkalmazás programozási felületekhez mintakódokat az Edge összetevők CD-ROM-lemezén talál, a samples könyvtárban. A WebSphere Application Server webhelyén további kódmintákat is találhat: www.ibm.com/software/webservers/appserv/

2. fejezet Caching Proxy API

Ebben a részben a Caching Proxy alkalmazás programozási felületét tárgyaljuk: pontosan mit is jelent ez, mire használható és hogyan működik.

FONTOS: A Caching Proxy az Edge összetevők minden telepítésekor elérhető, kivéve a következőket:

- A Caching Proxy nem érhető el az Edge összetevők Itanium 2 vagy 64 bites AMD Opteron processzorokon futó telepítéseiben.
- A Caching Proxy nem érhető el az Edge összetevők Load Balancer for IPv4 and IPv6 telepítéseiben.

Caching Proxy API - áttekintés

Az alkalmazás programozási felület a Caching Proxy interfésze, amely lehetővé teszi a proxyszerver alapszolgáltatásainak kibővítését. Lehetőséget ad bővítmények és bedolgozók írására, valamint egyénre szabott feldolgozás megvalósítására, mint például:

- Az alapszintű hitelesítő rutin továbbfejlesztése vagy helyspecifikus eljárásra való cseréje.
- Hibakezelő rutinok hozzáadása, amelyekkel követhetők a problémák, illetve vészhelyzet esetén riasztások adhatók ki.
- A kéréseket intéző ügyfelektől beérkező információk felismerése és nyomkövetése; ilyenek például a szerverhivatkozások és a felhasználói ügynökök kódjai.

A Caching Proxy alkalmazás programozási felületének használata a következő előnyökkel jár:

- **Hatékonyság**
  - Az alkalmazás programozási felületet kifejezetten a Caching Proxy által alkalmazott többszálú feldolgozó rendszerhez tervezték.
- **Rugalmasság**
  - Az API funkciógazdag és sokoldalú függvényeket tartalmaz.
  - Az API platform- és nyelvfüggetlen. Minden Caching Proxy platformon futtatható, a bedolgozó alkalmazások szinte bármelyik a megfelelő platformok által támogatott programozási nyelven elkészíthetők.
- **Könnyű használat**
  - Az egyszerű adattípusok átadása referencia, és nem érték szerint történik (például: long *, char *).
  - Minden függvénynek rögzített számú paramétere van.
  - A C nyelvi kötések mellékelve vannak.
  - A bedolgozók működése nincs kihatással a lefoglalt memóriára, a bedolgozó alkalmazások a Caching Proxy egyéb folyamataitól függetlenül végzik a memória lefoglalását és felszabadítását.

Az API programok írásának általános eljárása

Mielőtt hozzáfogna saját Caching Proxy bedolgozó programjainak megírásához, meg kell ismernie a proxyszerver működését. A proxyszerver működése több, egymástól határozottan elkülöníthető feldolgozási lépésre osztható fel. Az API segítségével a lépések mindegyikénél lehetőség van saját, egyéni funkciók megvalósítására. Lehetséges például, hogy el szeretne

végezni valamilyen műveletet az ügyfél kérésének beolvasása után és az egyéb feldolgozások elvégzése előtt. Esetleg különleges rutinok futtatására lehet szüksége a hitelesítés során és a kért fájl elküldése után.

Az alkalmazás programozási felülethez előre megadott függvények függvénytára tartozik. A bedolgozó programok az API előre megadott függvényeit meghívva párbeszédet folytathatnak a proxyszerver folyamatával (például módosíthatják a kéréseket, olvashatják és írhatják a fejléceket, illetve írhatnak a proxyszerver naplójába). Ezeket a függvényeket nem szabad összekeverni az Ön által írt bedolgozó függvényekkel, amelyek meghívását a proxyszerver végzi. Az előre meghatározott függvények leírását a következő részben találja meg: “Előre meghatározott függvények és makrók” oldalszám: 16.

A proxyszervert a megfelelő lépésnél a szükséges bedolgozó függvények meghívására a szerver konfigurációs fájljában megadott Caching Proxy API utasításokkal veheti rá. Ezeknek az utasításoknak a leírását a következő részben találja: “Caching Proxy konfigurációs utasítások az API lépésekhez” oldalszám: 22.

A jelen dokumentum a következő témákat tárgyalja:

- A Caching Proxy egyénre szabható lépéseinek alapszintű ismertetése (lásd: “A szerveroldali feldolgozás lépései”)
- Irányelvek bedolgozók írásához (lásd: “Irányelvek” oldalszám: 7)
- A szerver által végrehajtott egyes lépésekhez készített egyéni függvények prototípusai, valamint ezek visszatérési kódjai (lásd: “A bedolgozó függvények prototípusai” oldalszám: 9)
- A bedolgozókból meghívható, előre meghatározott függvények és makrók definíciója és visszatérési kódjai (lásd: “Előre meghatározott függvények és makrók” oldalszám: 16)
- Caching Proxy API konfigurálási utasítások (lásd: “Caching Proxy konfigurációs utasítások az API lépésekhez” oldalszám: 22)

Saját Caching Proxy bedolgozó programjainak megírásához ezek az összetevők és eljárások állnak rendelkezésére.

A szerveroldali feldolgozás lépései

A proxyszerver alapműködése annak alapján bontható lépésekre, hogy az adott fázisban a szerver milyen típusú feldolgozást végez. Minden lépéshez tartozik egy csatlakozási pont, amelyen lehetőség van a saját program futtatására. A Caching Proxy konfigurációs fájljához (ibmproxy.conf) a megfelelő utasításokat hozzáadva megadható, hogy az egyes lépések során mely bedolgozó függvényeket kell meghívni. Egy-egy lépéshez több utasítás is megadható, így a folyamatok egyes lépéseihez több bedolgozó függvény is meghívható.

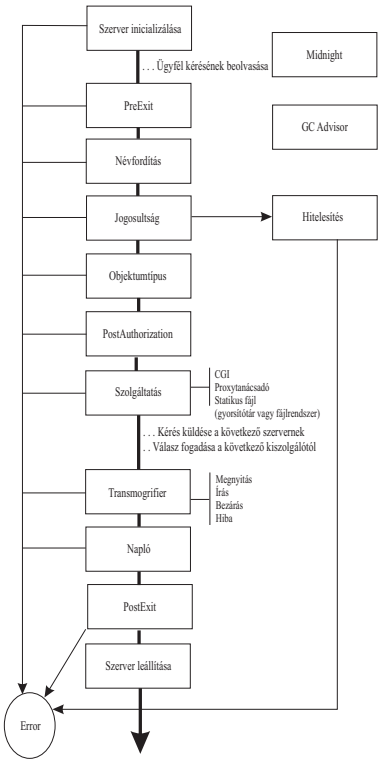
Egyes lépések a kiszolgálókérési folyamat részét képezik. Más szóval, a proxyszerver ezeket a lépéseket minden olyan alkalommal elvégzi, amikor feldolgoz valamilyen kérést. Más lépések elvégzése független a kérésfeldolgozástól, vagyis a szerver ezeket attól függetlenül hajtja végre, hogy éppen ki kell-e elégítenie valamilyen kérést.

A lefordított saját program egy megosztott objektumba kerül, például DLL vagy .so fájlba, az operációs rendszertől függően. Miközben a szerver végighalad a kérés feldolgozásának lépésein, meghívja az egyes lépésekhez rendelt bedolgozó függvényeket, egészen addig, amíg az egyik függvény azt nem jelzi, hogy lekezelte a kérést. Ha egy lépéshez egynél több bedolgozó függvényt ad meg, a függvények meghívása a hozzájuk tartozó utasítások a konfigurációs fájlban való megadásának sorrendjében történik.

Ha a kérést egyik bedolgozó függvény sem kezelte le (akár mert az adott lépéshez nem adott meg Caching Proxy API utasítást, akár mert a lépéshez tartozó bedolgozó függvény HTTP_NOACTION értéket adott vissza), a szerver végrehajtja a lépéshez tartozó saját alapértelmezett műveletét.

Megjegyzés: Ez minden lépésre igaz, kivéve a Service lépést, amelynek nincs alapértelmezett művelete.

1. ábra: - A proxyszerver által végrehajtott lépések, illetve a kérések feldolgozásával kapcsolatos lépések végrehajtási sorrendje.



1. ábra: A proxyszerver folyamatot alkotó lépések folyamatábrája

Az ábrán látható lépések közül négynek a végrehajtása az ügyfelek kéréseinek feldolgozásától függetlenül történik. Ezek a lépések a proxyszerver futtatásával és karbantartásával kapcsolatosak. A következők tartoznak ide:

- Server Initialization
- Midnight
- GC Advisor
- Server Termination

Az alábbiakban az 1. ábra: oldalszám: 5 által megadott lépések szerepét ismertetjük. Megjegyezzük, hogy egy-egy kéréshez nem feltétlenül kerül meghívásra az összes lépés.

Server Initialization, szerver inicializálása
A proxyszerver indításakor, a felhasználói kérések fogadása előtt végez inicializálást.

Midnight, éjfél
Éjfélkor lefuttat egy bedolgozót, kéréskörnyezet nélkül. Ez a lépés külön szerepel az ábrán, mert a kérésfeldolgozási folyamatnak nem része, vagyis végrehajtása a kérésektől független.

GC Advisor, GC tanácsadó
A gyorsítótárban lévő fájlokra vonatkozó szemétgyűjtési döntéseket befolyásolja. Ez a lépés külön szerepel az ábrán, mert a kérésfeldolgozási folyamatnak nem része, vagyis végrehajtása a kérésektől független. A szemétgyűjtésre akkor kerül sor, ha a gyorsítótár mérete eléri a maximális értéket. (A gyorsítótár szemétgyűjtéséről részletesebb információkat a *WebSphere Application Server Caching Proxy adminisztrációs útmutató* tartalmaz.)

PreExit

A kérés beolvasása után és az egyéb műveletek elvégzése előtt végez feldolgozást.

Ha ez a lépés azt jelzi, hogy a kérés feldolgozása megtörtént (HTTP_OK), akkor a szerver kihagyja a kérésfeldolgozási folyamat további lépéseit, és csak a Transmogrifier, a Log és a PostExit lépéseket hajtja végre.

Name Translation, névfordítás
A virtuális elérési utat (az URL címből) a fizikai útvonalra fordítja le.

Authorization, jogosultságkezelés

Elmentett biztonsági tokenek segítségével ellenőrzi a fizikai elérési út védelmét, a hozzáférés-vezérlési listákat és az egyéb hozzáférés-szabályozókat, valamint előállítja az alapszintű hitelesítéshez szükséges WWW-Authenticate fejléceket. Ha a lépés helyettesítésére saját bedolgozó függvényt ír, akkor a fejlécek előállításáról is gondoskodnia kell.

További információk: “Hitelesítés és jogosultságkezelés” oldalszám: 34

Authentication, hitelesítés

Visszafejti, ellenőrzi és tárolja a biztonsági tokeneket.

További információk: “Hitelesítés és jogosultságkezelés” oldalszám: 34

Object Type, objektumtípus
Az elérési útban megadott rendszerobjektumot keresi meg.

Post Authorization, utó-jogosultságkezelés

A feldolgozást a hitelesítés és az objektum megkeresése után, de még a kérés kielégítése előtt végzi.

Ha ez a lépés azt jelzi, hogy a kérés feldolgozása megtörtént (HTTP_OK), akkor a szerver kihagyja a kérésfeldolgozási folyamat további lépéseit, és csak a Transmogrifier, a Log és a PostExit lépéseket hajtja végre.

Service, szolgáltatás
Kielégíti a kérést (a fájl elküldésével, a CGI futtatásával stb.)

Proxy Advisor, proxytanácsadó
A proxyzási és a gyorsítótárási döntéseket befolyásolja.

Transmogrifier, átalakító
Írási hozzáférést ad az ügyfélnek küldött válasz adatrészéhez.

Log, napló
Egyéni tranzakciónaplózást tesz lehetővé.

Error, hiba
Egyéni válaszadást tesz lehetővé a hibahelyzetekre.

PostExit
Felszabadítja a kérésfeldolgozáshoz lefoglalt erőforrásokat.

Server Termination, szerver futásának lezárása
A normál leállítások során takarítási jellegű műveleteket végez el.

Irányelvek

- Saját programjait a szerver bedolgozó függvényeihez megadott szintaxist és irányelveket követve írja meg. Minden bedolgozó függvénynek egyedi nevet adjon, a szerver előre meghatározott függvényeit pedig szükség szerint hívja meg.

  Az AIX rendszereken szükség van egy export fájlra (például: libsajatprogram.exp), amely tartalmazza a bedolgozó függvények listáját, illetve a programot össze kell szerkeszteni a Caching Proxy API import fájljával, a libhttpdapi.exp fájllal.

  Linux, HP-UX és Solaris alatt a libhttpdapi és a libc függvénytárakkal kell összeszerkeszteni.

  A Windows alapú rendszereken egy moduldefiníciós fájlra (.def) van szükség, amely felsorolja a bedolgozó függvényeket, illetve a HTTPDAPI.LIB fájllal kell összeszerkeszteni a programot.

  Ügyeljen arra, hogy a HTAPI.h fájlt be kell illesztenie, a függvények meghatározásakor pedig a HTTPD_LINKAGE makrót kell használnia. A makró biztosítja, hogy az összes függvény ugyanazokat a hívási jelöléseket használja.
- A szerver többszálú környezetben fut, ezért a bedolgozóknak szálbiztosnak kell lenniük. Az alkalmazás újra hívhatósága nem okozza a teljesítmény csökkenését.
- A bedolgozók műveleteit tartsa a szálak hatókörén belül. A folyamatok hatókörében végezzen műveleteket, mint például kilépés, a felhasználói azonosító megváltoztatása vagy jelzéskezelő végrehajtása.
- Ne használjon globális változókat, illetve, ha mégis használnia kell ilyeneket, kölcsönös kizárási szemaforokkal védje őket.
- Ha a HTTPD_write() függvénnyel adatokat küld az ügyfélnek, akkor ne felejtse el megadni a Content-Type fejlécet.
- Mindig adjon visszatérési kódot, és ahol szükséges, ott feltételes feldolgozást végezzen.
- A program fordítását és összeállítását a fordítóprogram leírásának megfelelően végezve, az adott operációs rendszer igényei szerint állítson elő egy megosztott objektumot (például DLL vagy .so fájlt).

  Útmutatásként alkalmazza az alábbi fordítási és összeállítási parancsokat.

  – **AIX**, IBM CSet++ használatával

    - **Fordítás:**

```
cc_r  -c -qdbxextra -qcpluscmt program.c
```

- **Összeszerkesztés:**

```
cc_r -bM:SRE -bnoentry -o libprogram.so program.o -bI:libhttpdapi.exp
     -bE:program.exp
```

(A parancsot csak az olvashatóság kedvéért osztottuk fel két sorra.)

– **HP-UX**, HP C/ANSI C Developer's Bundle és HP aC++ Compiler használatával

- **Fordítás:**

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

- **Összeszerkesztés:**

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

– **Linux**, a Gnu Compiler C (GCC) 3.2.X változatának használatával

- **Fordítás:**

```
gcc -c program.c
```

- **Összeszerkesztés:**

```
ld -G -Bsymbolic -o libprogram.so program.o -lhttpdapi -lc
```

– **Solaris**, a Sun Workshop használatával

- **Fordítás:**

```
cc -mt -Bsymbolic -c program.c
```

- **Összeszerkesztés:**

```
cc -mt -Bsymbolic -G -o libprogram.so program.o -lhttpdapi -lc
```

– **Windows**, Microsoft Visual C++ használatával

- **Fordítás:**

```
cl /c /MD /DWIN32 program.c
```

- **Összeszerkesztés:**

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

Az exportok megadásához az alábbi módszerek közül választhat:

– Adja hozzá a _declspec(dllexport) meghatározásokat a forráshoz.

– Alkalmazza a /EXPORT:bejegyzésnév parancsot a LIB parancssorban.

– Hozzon létre egy EXPORTS utasítást tartalmazó modulmeghatározó fájlt.

- Adja hozzá a megfelelő Caching Proxy API utasításokat a konfigurációs fájlhoz, amivel társíthatja a program bedolgozó függvényeit a megfelelő lépésekkel. A szerver kérésfeldolgozási folyamatának minden lépéséhez külön utasítás tartozik. Az új utasításokat a szerver leállításával és újraindításával léptetheti érvénybe.

  Megjegyzés: A Caching Proxy még újraindításkor sem távolítja el a megosztott objektumokat (DLL vagy .so fájlokat). A megosztott objektumok felszabadításához le kell állítania és újra el kell indítania a szervert.

- Az éles környezetben való használat előtt gondosan tesztelje a programot. Mivel a Caching Proxy több szálra osztott szerver, gondosabb tesztelést kell végeznie, mintha elágazásokat alkalmazó szerverhez fejlesztene. A saját program hibái a teljes proxyszerver leállását is okozhatják, ugyanis a proxyszerver közvetlenül hívja meg a programot, és mindkettő ugyanazon a folyamatterületen fut.

Bedolgozó függvények

Saját, a kérésfeldolgozási lépésekhez tartozó programfüggvényeinek megírásakor kövesse a "A bedolgozó függvények prototípusai" oldalszám: 9 című részben megadott szintaxist.

A függvények mindegyikének értéket kell adnia a visszatérési kód paraméternek, jelezve, hogy milyen művelet elvégzésére került sor:

- A HTTP_NOACTION kód (0-s érték) azt jelenti, hogy nem történt a tárgyra vonatkozó művelet. Ha ezt a kódot kapja, akkor a proxyszerver a lépéshez tartozó alapértelmezett műveletet hajtja végre.
- Az érvényes HTTP visszatérési kódok azt jelzik, hogy a bedolgozó függvény lekezelte a lépést. (Az érvényes visszatérési kódok listáját a “HTTP visszatérési kódok és értékek” oldalszám: 15 című rész tartalmazza.) Érvényes HTTP visszatérési kód átadásakor az adott lépésben a kérés kezeléséhez további függvényeket a rendszer nem hív meg.

A bedolgozó függvények prototípusai

A Caching Proxy egyes lépéseihez megadott függvényprototípusok bemutatják a megfelelő formátumot, valamint kapcsolódik hozzájuk az általuk végrehajtható feldolgozás típusának ismertetése. Megjegyezzük, hogy a függvénynevek nincsenek előre meghatározva. A saját függvényeknek egyedi nevet kell adni, az elnevezési megállapodást mindenki maga határozhatja meg. A megértés könnyítése érdekében a jelen dokumentumban a szerver feldolgozási lépéseire utaló neveket használtunk.

Minden bedolgozó függvény esetében bizonyos előre meghatározott API függvények érvényesek. Az előre meghatározott függvények egy része egyes lépéseknél nem érvényes. Az alábbi előre meghatározott API függvényeket az összes bedolgozó függvényből meg lehet hívni:

- HTTPD_set
- HTTPD_extract
- httpd_setvar
- httpd_getvar
- HTTPD_log* függvények

A további API függvények érvényességét vagy érvénytelenségét az egyes függvények prototípusának leírásánál tüntettük fel.

A függvényeknek átadott *handle* paraméter értéke az előre meghatározott függvények első argumentumaként adható át. Az előre meghatározott API függvények leírását a következő rész tartalmazza: “Előre meghatározott függvények és makrók” oldalszám: 16

Server Initialization, szerver inicializálása

```
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 proxyszerver’s version number; these are supplied by the Caching Proxy.

PreExit

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

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

Valid return codes for the preExit function are the following:

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

Other return codes must not be used.

If this function returns HTTP_OK, the proxyszerver 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.

Name Translation, névfordítás

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

Az ehhez a lépéshez megadott függvény minden kéréshez meghívódik. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató kérések esetében hívódjon meg. A névfordítási lépés a kérés feldolgozása előtt játszódik le, és módot ad az URL címek objektumokhoz, például fájlnevekhez való hozzárendelésére.

Authorization, jogosultságkezelés

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

Az ehhez a lépéshez megadott függvény minden kéréshez meghívódik. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató kérések esetében hívódjon meg. A jogosultságkezelési lépés a kérés feldolgozása előtt játszódik le, és lehetőséget ad annak ellenőrzésére, hogy a kiválasztott objektum átadható-e az ügyfélnek. Ha alapszintű hitelesítést végez, akkor létre kell hoznia a szükséges WWW-Authenticate fejléceket.

Object Type, objektumtípus

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

Az ehhez a lépéshez megadott függvény minden kéréshez meghívódik. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató kérések esetében hívódjon meg. Az objektumtípus lépés a kérés feldolgozása előtt játszódik le, segítségével ellenőrizhető, hogy az objektum létezik-e, illetve elvégezhető az objektum típusának meghatározása.

PostAuthorization, utó-jogosultságkezelés

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

Az ehhez a lépéshez megadott függvény meghívására a kérés jogosultságkezelése után és a feldolgozás megkezdése előtt kerül sor. Ha a függvény visszatérési értéke HTTP_OK, akkor a proxyszerver úgy veszi, hogy a kérés lekezelése megtörtént. A kérésfeldolgozás további részei kimaradnak, és a rendszer csak a válaszadási lépéseket (Transmogrifier, Log és PostExit) hajtja végre.

Ebben a lépésben a szerver összes előre meghatározott függvénye használható.

Service, szolgáltatás

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

Az ehhez a lépéshez megadott függvény minden kéréshez meghívódik. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató kérések esetében hívódjon meg. A szolgáltatás lépés feladata a kérés kielégítése, amennyiben ez a PreExit vagy a PostAuthorization lépés során még nem történt meg.

Ebben a lépésben a szerver összes előre meghatározott függvénye használható.

Ha a szolgáltatás függvényt az URL helyett a HTTP metódus alapján szeretné futtatni, akkor tanulmányozza a *WebSphere Application Server Caching Proxy adminisztrációs útmutató* az Enable utasításról szóló részét.

Átalakító

Az ebben a feldolgozási lépésben meghívott függvényekkel a válaszadatokat adatfolyamként lehet szűrni. Ebben a lépésben sorban négy bedolgozó függvény hívódik meg, mindegyik egy az adatokat továbbadó csővezeték egy-egy

szakaszaként viselkedik. Minden válaszhoz tehát a megadott *megnyitás*, *írás*, *lezárás* és *hiba* függvény kerül meghívásra, ebben a sorrendben. Mindegyik függvény ugyanazzal az adatfolyammal dolgozik.

Ehhez a lépéshez az alábbi négy függvényt kell megvalósítani. (A függvényneveknek nem kell egyezniük az itt szereplőkkel.)

- **Megnyitás**

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

  A megnyitás függvény végzi az adatfolyam adatainak feldolgozásához szükséges inicializálási műveleteket (például a pufferfoglalást). Minden a HTTP_OK kódtól eltérő visszatérési kód a szűrő futásának megszakítását okozza (az írás és a bezárás függvény meghívása elmarad). A függvények részéről lehetőség van kitöltetlen mutató visszaadására, így mód nyílik arra, hogy helyet foglaljon egy adatszerkezet számára, majd a további függvények *correlator* paraméterében visszakapja a mutatót.

- **Írás**

```
void  HTTPD_LINKAGE  writeFunction (
         unsigned char *handle,
         unsigned char *data,    /* a származási szerver által
                                    küldött adatok */
         unsigned long *length,  /* a válaszadatok hossza */
         void *correlator,       /* a megnyitás függvény által
                                    visszaadott mutató */
         long *return_code
         )
```

  Az írás függvény feldolgozza az adatokat, majd képes a szerver előre meghatározott HTTPD_write() függvényének az új vagy megváltozott adatokkal való meghívására. A bedolgozónak nem szabad felszabadítania a számára átadott puffert, illetve ezt a szervertől sem várhatja.

  Ha saját írás függvényével nem kívánja módosítani az adatokat, HTTPD_write() függvényt továbbra is meg kell hívnia, akár saját megnyitás, írás vagy bezárás függvényéből, ugyanis ekkora kerül sor az ügyfélnek szánt válasz adatainak átadására. A *correlator* argumentum az a mutató, amely a saját *megnyitás* rutinban visszaadott adatpufferre vezet.

- **Lezárás**

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

  A lezárás függvény az adatfolyam feldolgozásának befejezéséhez szükséges takarítási műveleteket végzi el (például a correlator puffer kiürítését és felszabadítását). A *correlator* argumentum az a mutató, amely a saját *megnyitás* rutinban visszaadott adatpufferre vezet.

- **Hiba**

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

  A hiba függvény segítségével hibaoldal küldése előtt takarítás jellegű műveleteket lehet végrehajtani, mint például a pufferelt adatok kiírása és/vagy felszabadítása (illetve mindkettő). Ezen a ponton a rendszer a megnyitás, az írás, és a lezárás

függvényt egyaránt meghívja a hibaoldal feldolgozására. A *correlator* argumentum az a mutató, amely a saját *megnyitás* rutinban visszaadott adatpufferre vezet.

Megjegyzések:

- Amikor az átalakító lépéshez ír bedolgozót, a megnyitás, az írás és a lezárás függvény futása során valamikor meg kell hívnia a HTTPD_open(), a HTTPD_write() és a HTTPD_close() függvényt. A HTTPD_write() csak a HTTPD_open() függvény meghívása után hívható. Ezeknek az előre meghatározott függvényeknek az a céljuk, hogy biztosítsák az ellenőrzést a szerver számára, lehetővé téve a soron következő függvény meghívását.
- A HTTPD_* függvények meghívása az átalakító API lépés és a szerver megfelelő működéséhez szükséges. Ha például a HTTPD_open() és a HTTPD_close() meghívása elmarad, az ügyfél nem kapja meg a fejléceket.
- Ügyeljen arra, hogy ha az adatszűrő alkalmazások hibásan választják ki a szűrendő adatokat, az nem kívánt hatásokkal járhat. Előfordulhat, hogy a CGI programok hibás szűrés mellett nem működnek megfelelően, a GIF fájlok nem jelennek meg, illetve az egyéb bináris adatfolyamok viselkedése a várttól eltérően alakul.
- A bedolgozónak nem muszáj pufferelnie a tartalom törzsét. A Caching Proxy automatikusan meghatározza a tartalom hosszát.
- A HTTPD_open() függvényt akkor érdemes meghívnia, ha készen áll a fejlécek feletti ellenőrzés a szervernek való átadására. Ugyanakkor, ha egy fejlécet később kell megadnia az API programban, akkor a HTTPD_open() függvény meghívásával akár az írás vagy a lezárás függvényig is várhat.

  Megjegyzés: A HTTPD_open() függvény meghívása előtt a HTTPD_set() vagy a httpd_setvar() függvénnyel minden fejlécet be kell állítania.
- Az adatfolyam nem tartalmazza a fejléceket. A bedolgozók a beállító és a kibontó függvényekkel tudják kezelni a fejléceket. A bedolgozó *megnyitás* függvénye csak az összes fejléc beolvasása után hívódik meg.
- Több átalakító bedolgozó használatára is van lehetőség, ezek meghívása a konfigurációs fájlban való szereplésük sorrendjében történik.
- Az SSL bújtatás az átalakító bedolgozókon nem halad keresztül.

GC Advisor, GC tanácsadó

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

Az ehhez a lépéshez megadott függvény a szemétgyűjtés során a gyorsítótár minden fájljához meghívódik. A függvény révén befolyásolni tudja, hogy mely fájlok maradjanak meg, és melyeket törölje a rendszer. További információkat a GC_* változók kapcsán kaphat.

Proxy Advisor, proxytanácsadó

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

Az ehhez a lépéshez megadott függvény meghívására minden proxykérés kiszolgálásakor sor kerül. Segítségével például beállítható a USE_PROXY változó.

Log, napló

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

Az ehhez a lépéshez megadott függvény meghívására minden kérésnél sor kerül, miután a kérés kielégítése megtörtént, illetve az ügyféllel folytatott kommunikáció befejeződött. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató kérések esetében hívódjon meg. A függvény meghívása független a kérés feldolgozásának sikerességétől vagy sikertelenségétől. Ha nem szeretné, hogy a napló bedolgozó felülbírálja az alapértelmezett naplózási mechanizmust, akkor a visszatérési kódot HTTP_OK helyett HTTP_NOACTION értékre állítsa.

Error, hiba

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

Az ehhez a lépéshez megadott függvény minden sikertelen kéréshez meghívásra kerül. A konfigurációs fájl utasításában lehetőség van egy URL-sablon megadására, amivel elérhető, hogy a bedolgozó függvény csak a sablonnal egyezést mutató sikertelen kérések esetében hívódjon meg. A hiba lépés lehetőséget ad a hibaválasz egyénre szabására.

PostExit

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

Az ehhez a lépéshez megadott függvény minden kéréshez meghívódik, tekintet nélkül annak sikerességére vagy sikertelenségére. A lépés lehetőséget biztosít takarítási jellegű feladatok a bedolgozó által a kérés feldolgozásához lefoglalt erőforrásokra vonatkozó elvégzésére.

Server Termination, szerver futásának lezárása

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

Az ehhez a lépéshez megadott függvény meghívására a szerver normál leállításakor kerül sor. Segítségével lehetőség nyílik a a szerver inicializálásának lépése során lefoglalt erőforrások felszabadítására. Ebben a lépésben nem szabad meghívni egyik HTTP_* függvényt sem (az eredményt ellenkező esetben nem lehet megjósolni). Ha a konfigurációs fájlban egynél több Caching Proxy API utasítást ad meg a szerver futásának lezárásához, akkor a rendszer mindegyiket meghívja.

Megjegyzés: A Solaris kód jelenlegi korlátai miatt a szerver futásának lezárása bedolgozó lépés végrehajtására nem kerül sor, ha Solaris platformon az **ibmproxy -stop** paranccsal történik a Caching Proxy leállítása. A Caching Proxy leállításával és elindításával kapcsolatban a *WebSphere Application Server Caching Proxy adminisztrációs útmutató* tartalmaz részletes információkat.

HTTP visszatérési kódok és értékek

Az alábbi visszatérési kódok megfelelnek a HTTP 1.1-es specifikációknak, amelyeket a World Wide Web Consortium (www.w3.org/pub/WWW/Protocols/) az RFC 2616 dokumentumban tett elérhetővé. A bedolgozó függvényeknek az alábbi értékek egyikével kell visszatérniük.

2. táblázat: HTTP visszatérési kódok a Caching Proxy API függvényekhez

Érték	**Visszatérési kód**
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

2. táblázat: HTTP visszatérési kódok a Caching Proxy API függvényekhez (Folytatás)

500	HTTP_SERVER_ERROR
501	HTTP_NOT_IMPLEMENTED
502	HTTP_BAD_GATEWAY
503	HTTP_SERVICE_UNAVAILABLE
504	HTTP_GATEWAY_TIMEOUT
505	HTTP_BAD_VERSION

Előre meghatározott függvények és makrók

Saját bedolgozó függvényeiből szükség esetén meghívhatja a szerver előre meghatározott függvényeit és makróit. Ekkor ezek előre meghatározott nevét és az alább ismertetett formátumot kell alkalmaznia. A paraméterek leírásában az i betű a bemenő paramétert jelzi, az o a kimenő paramétert, az i/o jelzés pedig arra utal, hogy az adott paraméter be- és kivitelre egyaránt használható.

A függvények mindegyike a HTTPD visszatérési kódok valamelyikét adja vissza, függően a kérés sikerességétől. A kódok leírását a következő rész tartalmazza: “Az előre meghatározott függvények és makrók visszatérési kódjai” oldalszám: 21.

A függvények meghívásakor első paraméterként a bedolgozónak átadott fájlleírót alkalmazza. Ellenkező esetben a függvény HTTPD_PARAMETER_ERROR hibakódot fog visszaadni. A NULL nem számít érvényes fájlleírónak.

HTTPD_authenticate()

Egy felhasználó azonosítóját és/vagy jelszavát hitelesíti. Csak a PreExit, az Authentication, az Authorization és a PostAuthorization lépés során érvényes.

```
void  HTTPD_LINKAGE  HTTPD_authenticate (
         unsigned char *handle,      /* i; fájlleíró */
         long *return_code           /* o; visszatérési kód */
         )
```

HTTPD_cacheable_url()

Megadja, hogy a Caching Proxy szabványai szerint az adott URL gyorsítótárazható-e.

```
void HTTPD_LINKAGE  HTTPD_cacheable_url (
        unsigned char *handle,      /* i; fájlleíró */
        unsigned char *url,         /* i; ellenőrizni kívánt URL */
        unsigned char *req_method,  /* i; kérési módszer az URL címhez */
        long *retval                /* o; visszatérési kód */
        )
```

A HTTPD_SUCCESS visszatérési érték azt jelzi, hogy az URL tartalma gyorsítótárazható, a HTTPD_FAILURE érték pedig arra utal, hogy a tartalom nem gyorsítótárazható. A függvény visszatérési kódja HTTPD_INTERNAL_ERROR is lehet.

HTTPD_close()

(Csak az átalakító lépés során használható.) Átadja a vezérlést az adatfolyamkészlet *lezárás* rutinjának. A függvényt az átalakító lépés megnyitás, írás vagy lezárás függvényéből hívhatja meg, miután a kívánt feldolgozási műveleteket elvégezte. A függvény értesíti a proxyszervert, hogy a válasz feldolgozása megtörtént, valamint az átalakító lépés befejeződött.

```
void  HTTPD_LINKAGE  HTTPD_close (
        unsigned char *handle,      /* i; fájlleíró */
        long *return_code           /* o; visszatérési kód */
        )
```

HTTPD_exec()

A kérés kielégítéséhez lefuttat egy parancsfájlt. A PreExit, a Service, a PostAuthorization és az Error lépésben érvényes.

```
void  HTTPD_LINKAGE  HTTPD_exec (
        unsigned char *handle,         /* i; fájlleíró */
        unsigned char *name,           /* i; a futtatandó parancsfájl neve */
        unsigned long *name_length,    /* i; a név hossza */
        long *return_code              /* o; visszatérési kód */
        )
```

HTTPD_extract()

Egy a kéréshez tartozó változó értékét bontja ki. A *name* paraméter érvényes változói azonosak a CGI-ben használtakkal. További információk: “Változók” oldalszám: 25 Megjegyezzük, hogy bár ez a függvény minden lépésben érvényes, változókra ez már nem igaz.

```
void  HTTPD_LINKAGE  HTTPD_extract (
      unsigned char *handle,       /* i; fájlleíró */
      unsigned char *name,         /* i; a kibontani kívánt változó neve */
      unsigned long *name_length,  /* i; a név hossza */
      unsigned char *value,        /* o; az a puffer, amelybe el kell helyezni
                                        az értéket */
      unsigned long *value_length, /* i/o; a puffer mérete */
      long *return_code            /* o; visszatérési kód */
      )
```

Ha a függvény a HTTPD_BUFFER_TOO_SMALL kódot adja vissza, akkor a kért puffer mérete túl kicsi volt a kibontott érték tárolásához. Ebben az esetben a függvény nem veszi igénybe puffert, de a value_length paramétert frissíti az érték sikeres kibontásához szükséges pufferméretttel. Ekkor a kibontást egy legalább a value_length értékben kapott méretű pufferrel kell megismételni.

Megjegyzés: Ha a kibontandó változó egy HTTP-fejléchez tartozik, akkor a HTTPD_extract() függvény csak az első egyező példányt bontja ki, még akkor is, ha a kérés több azonos nevű fejlécet is tartalmaz. A HTTPD_extract() helyett a httpd_getvar() függvény is használható, illetve utóbbinak egyéb előnyei is vannak. További információk: 17

HTTPD_file()

A kérés kielégítéséhez elküld egy fájlt. Csak a PreExit, a Service, az Error, a PostAuthorization és a Transmogrifier lépésben érvényes.

```
void  HTTPD_LINKAGE  HTTPD_file (
        unsigned char *handle,          /* i; fájlleíró */
        unsigned char *name,            /* i; az elküldeni kívánt fájl neve */
        unsigned long *name_length,     /* i; a név hossza */
        long *return_code               /* o; visszatérési kód */
        )
```

httpd_getvar()

Azonos a HTTPD_extract() függvénnyel, azzal a különbséggel, hogy könnyebb használni, ugyanis a felhasználónak nem kell megadnia az argumentumok hosszát.

```
const  unsigned char *          /* o; a változó értéke */
HTTPD_LINKAGE
httpd_getvar(
   unsigned char *handle,       /* i; fájlleíró */
```

```
unsigned char *name,         /* i; változó neve */
unsigned long *n             /* i; a fejlécet tartalmazó tömb
                                indexszáma */
)
```

A fejlécet tartalmazó tömb indexelése nullával kezdődik. Ha a tömb első elemét szeretné megkapni, az *n* változónak nullás értéket adjon, míg például a negyedik elemet az *n* változónak négyes értéket adva kaphatja meg.

Megjegyzés: A kapott érték tartalmát ne törölje és ne változtassa meg. A kapott karaktersorozat null értékkel végződik.

HTTPD_log_access()

Egy karaktersorozatot ír a szerver hozzáférési naplójába.

```
void  HTTPD_LINKAGE  HTTPD_log_access (
        unsigned char *handle,          /* i; fájlleíró */
        unsigned char *value,           /* i; a kiírni kívánt adatok */
        unsigned long *value_length,    /* i; az adatok hossza */
        long *return_code               /* o; visszatérési kód */
        )
```

Megjegyezzük, hogy százalék (%) szimbólum a szerver hozzáférési naplójába való kiírásakor *nincs* szükség kilépési (escape) szimbólum használatára.

HTTPD_log_error()

Egy karaktersorozatot ír a szerver hibanaplójába.

```
void  HTTPD_LINKAGE  HTTPD_log_error (
        unsigned char *handle,          /* i; fájlleíró */
        unsigned char *value,           /* i; a kiírni kívánt adatok */
        unsigned long *value_length,    /* i; az adatok hossza */
        long *return_code               /* o; visszatérési kód */
        )
```

Megjegyezzük, hogy százalék (%) szimbólum a szerver hibanaplójába való kiírásakor *nincs* szükség kilépési (escape) szimbólum használatára.

HTTPD_log_event()

Egy karaktersorozatot ír a szerver eseménynaplójába.

```
void  HTTPD_LINKAGE  HTTPD_log_event (
        unsigned char *handle,          /* i; fájlleíró */
        unsigned char *value,           /* i; a kiírni kívánt adatok */
        unsigned long *value_length,    /* i; az adatok hossza */
        long *return_code               /* o; visszatérési kód */
        )
```

Megjegyezzük, hogy százalék (%) szimbólum a szerver eseménynaplójába való kiírásakor *nincs* szükség kilépési (escape) szimbólum használatára.

HTTPD_log_trace()

Egy karaktersorozatot ír a szerver nyomkövetési naplójába.

```
void  HTTPD_LINKAGE  HTTPD_log_trace (
        unsigned char *handle,          /* i; fájlleíró */
        unsigned char *value,           /* i; a kiírni kívánt adatok */
        unsigned long *value_length,    /* i; az adatok hossza */
        long *return_code               /* o; visszatérési kód */
        )
```

Megjegyezzük, hogy százalék (%) szimbólum a szerver nyomkövetési naplójába való kiírásakor *nincs* szükség kilépési (escape) szimbólum használatára.

HTTPD_open()

(Csak az átalakító lépés során használható.) Átadja a vezérlést az adatfolyamkészlet

következő rutinjának. A Transmogrifier lépést megnyitás, írás vagy lezárás függvényéből hívhatja meg, miután minden kívánt fejlécet beállított, és készen áll az írási rutin elindítására.

```
void  HTTPD_LINKAGE  HTTPD_open (
        unsigned char *handle,      /* i; fájlleíró */
        long *return_code           /* o; visszatérési kód */
        )
```

HTTPD_proxy()

Egy proxykérést indít el. A PreExit, a Service és a PostAuthorization lépésben használható.

Megjegyzés: Befejező jellegű függvény, lefutása után a kérés kiszolgálása befejezettnek tekintendő.

```
void  HTTPD_LINKAGE  HTTPD_proxy (
        unsigned char *handle,      /* i; fájlleíró */
        unsigned char *url_name,      /* i; a proxykérés
                                           URL címe */
        unsigned long *name_length,   /* i; az URL hossza */
        void *request_body,           /* i; a kérés törzse */
        unsigned long *body_length,   /* i; a törzs hossza */
        long *return_code             /* o; visszatérési kód */
        )
```

HTTPD_read()

Az ügyfél kérésének törzsét olvassa be. A fejlécekhez használja a HTTPD_extract() függvényt. Csak a PreExit, az Authorization, a PostAuthorization és a Service lépésben használható, és csak PUT vagy POST kérés esetén van értelme alkalmazni. A függvényt hurokban kell meghívni, amíg a HTTPD_EOF visszatérési értéket nem adja. Ha a kérésnek nincs törzse, a függvény futása meghiúsul.

```
void  HTTPD_LINKAGE  HTTPD_read (
        unsigned char *handle,      /* i; fájlleíró */
        unsigned char *value,           /* i; puffer az adatok számára */
        unsigned long *value_length,    /* i/o; pufferméret
                                               (adathossz) */
        long *return_code               /* o; visszatérési kód */
        )
```

HTTPD_restart()

Az összes aktív kérés feldolgozása után újraindítja a szervert. Az összes lépésben érvényes, kivéve a Server Initialization, a Server Termination és a Transmogrifier lépést.

```
void  HTTPD_LINKAGE  HTTPD_restart (
        long *return_code             /* o; visszatérési kód */
        )
```

HTTPD_set()

Beállítja a kéréshez rendelt változó értékét. A *name* paraméter esetében érvényes változók azonosak a CGI-ben használtakkal. További információk: “Változók” oldalszám: 25

Megjegyezzük, hogy a függvénnyel létrehozni is lehet változókat. A létrehozott változókra a HTTP_ és PROXY_ előtagokkal kapcsolatos megállapodások vonatkoznak, amelyek ismertetését a “Változók” oldalszám: 25 rész tartalmazza. Ha HTTP_ kezdetű változót hoz létre, azt az ügyfél is megkapja a válasz egyik fejléceként, de a HTTP_ előtag nélkül. Ha például Location fejlécet szeretne beállítani, a HTTPD_set() függvényt a HTTP_LOCATION változónévvel kell alkalmaznia. A PROXY_ előtaggal létrehozott változók a tartalomszerverhez intézett kérésekbe kerülnek fejlécként. A CGI_ előtagot kapó változókat a rendszer a CGI programoknak adja át.

A függvény minden lépésben érvényes, ugyanakkor a változókra ez nem igaz.

```
void  HTTPD_LINKAGE  HTTPD_set (
         unsigned char *handle,      /* i; fájlleíró */
         unsigned char *name,          /* i; a beállítani kívánt érték neve */
         unsigned long *name_length,   /* i; a név hossza */
         unsigned char *value,         /* i; az értéket tartalmazó puffer */
         unsigned long *value_length,  /* i; az érték hossza */
         long *return_code             /* o; visszatérési kód */
         )
```

Megjegyzés: A httpd_setvar() függvénnyel úgy is beállíthatja változó értékét, hogy nem ad meg puffert és hosszt. További információk: 20

httpd_setvar()

Azonos a HTTPD_set() függvénnyel, azzal a különbséggel, hogy könnyebb használni, ugyanis esetében a felhasználónak nem kell megadnia az argumentumok hosszát.

```
long             /* o; visszatérési kód */
HTTPD_LINKAGE   httpd_setvar (
       unsigned char *handle,       /* i; fájlleíró */
       unsigned char *name,         /* i; változó neve */
       unsigned char *value,        /* i; új érték */
       unsigned long *addHdr        /* i; fejléc hozzáadása vagy cseréje */
       )
```

Az *addHdr* paraméter négyféle értéket vehet fel:

- HTTPD_SETVAR_REPLACE — A fejlécváltozó összes példányának cseréje az új értékre.
- HTTPD_SETVAR_REPLACE_ADD — Ha a fejlécváltozó létezik, az első előfordulás lecserélése az új értékre; ha a változó nem létezik, akkor az új érték hozzáfűzése a fejlécekhez.
- HTTPD_SETVAR_ADD — Az érték hozzáfűzése a fejlécekhez.
- HTTPD_SETVAR_REMOVE_ALL — A fejlécváltozó összes előfordulásának eltávolítása.

Az értékek meghatározása a HTAPI.h fájlban található.

httpd_variant_insert()

Variáns beillesztése a gyorsítótárba.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,         /* i; fájlleíró */
         unsigned char *URI,       /* i; az objektum URI azonosítója */
         unsigned char *dimension, /* i; a variáns mérete */
         unsigned char *variant,   /* i; a variáns értéke */
         unsigned char *filename,  /* i; az objektumot tartalmazó fájl */
         long *return_code         /* o; visszatérési kód */
         )
```

Megjegyzések:

1. A dimension argumentum arra a fejlécre hivatkozik, amelyben az objektum eltér az URI azonosítótól. A fenti esetben a dimension értéke például User-Agent lehetne.
2. A variant argumentum a dimension argumentumban megadott fejléc fejlécértékére vonatkozik. Ez az URI azonosítótól függ. A fenti esetben a variant argumentum értéke például a következő lehetne:

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

3. A filename argumentumnak azon fájl nevének a null végződésű példányára kell mutatnia, amelybe a felhasználó a módosított tartalmat mentette. A fájl

eltávolítása a felhasználó feladata, a műveletet a függvényből való visszatérés után lehet biztonságosan elvégezni. A fájl csak a törzset tartalmazza, fejlécek nélkül.

4. A variánsok gyorsítótárazásakor a szerver frissíti a content-length fejlécet, és hozzáadja a Warning: 214 fejlécet. A strong egyedcímkéket eltávolítja.

httpd_variant_lookup()

Meghatározza, hogy egy adott variáns megtalálható-e a gyorsítótárban.

```
void  HTTPD_LINKAGE  httpd_variant_lookup (
    unsigned char *handle,            /* i; fájlleíró */
    unsigned char *URI,               /* az objektum URI azonosítója */
    unsigned char *dimension,         /* i; a variáns mérete */
    unsigned char *variant,           /* i; a variáns értéke */
             long *return_code);      /* o; visszatérési kód */
```

HTTPD_write()

A válasz törzsét írja ki. A PreExit, a Service, az Error és a Transmogrifier lépésben érvényes.

Ha a függvény első meghívása előtt nem állítja be a tartalom típusát, a szerver feltételezi, hogy CGI adatfolyamot küld.

```
void  HTTPD_LINKAGE  HTTPD_write (
    unsigned char *handle,            /* i; fájlleíró */
    unsigned char *value,             /* i; az elküldeni kívánt adatok */
    unsigned char *value_length,      /* i; az adatok hossza */
             long *return_code);      /* o; visszatérési kód */
```

Megjegyzés: A válaszfejlécek beállításával kapcsolatban lásd: 19.

Megjegyzés: A HTTPD_* függvények visszatérése után a nekik átadott memória felszabadítása biztonságosan elvégezhető.

Az előre meghatározott függvények és makrók visszatérési kódjai

A szerver a kérés kiszolgálásának sikerességétől függően az alábbi értékek egyikére állítja be a visszatérési kód paramétert:

3. táblázat: Visszatérési kódok

Érték	Állapotkód	Magyarázat
-1	HTTPD_UNSUPPORTED	A függvény nem támogatott.
0	HTTPD_SUCCESS	A függvény futása sikeres volt, és a kimeneti mezők érvényesek.
1	HTTPD_FAILURE	A függvény futása sikertelen volt.
2	HTTPD_INTERNAL_ERROR	Belső hiba történt, a kérés feldolgozása nem folytatható.
3	HTTPD_PARAMETER_ERROR	Legalább egy átadott paraméter érvénytelen volt.
4	HTTPD_STATE_CHECK	Az adott függvény az adott feldolgozási lépésben nem használható.
5	HTTPD_READ_ONLY	(Csak a HTTPD_set és a httpd_setvar visszatérési értéke lehet.) A változó írásvédett, a bedolgozó nem állíthatja be az értékét.
6	HTTPD_BUFFER_TOO_SMALL	(A HTTPD_set, a httpd_setvar és a HTTPD_read visszatérési értéke lehet.) A rendelkezésre bocsátott puffer túl kicsit volt.

3. táblázat: Visszatérési kódok (Folytatás)

Érték	Állapotkód	Magyarázat
7	HTTPD_AUTHENTICATE_FAILED	(Csak a HTTPD_authenticate visszatérési értéke lehet.) A hitelesítés meghiúsult. További információkat a HTTP_RESPONSE és a HTTP_REASON változó vizsgálatával kaphat.
8	HTTPD_EOF	(Csak a HTTPD_read visszatérési értéke lehet.) A kérés törzsének végét jelzi.
9	HTTPD_ABORT_REQUEST	A kérés teljesítése megszakadt, mert az ügyfél olyan egyedcímkét adott át, amely nem felelt meg a kérés által meghatározott feltételeknek.
10	HTTPD_REQUEST_SERVICED	(A HTTPD_proxy visszatérési értéke lehet.) A meghívott függvény befejezte a kérésre történő válaszadást.
11	HTTPD_RESPONSE_ALREADY_ COMPLETED	A függvény futása sikertelen volt, mert a kérésre történő válaszadás már befejeződött.
12	HTTPD_WRITE_ONLY	A változó csak írható, a bedolgozó nem tudja olvasni.

Caching Proxy konfigurációs utasítások az API lépésekhez

A kérésfeldolgozás minden lépéséhez tartozik egy konfigurációs utasítás, amellyel meg lehet adni, hogy a bedolgozó függvények melyikét kell meghívni és lefuttatni az adott lépés során. Az utasításokat a szerver konfigurációs fájljához (ibmproxy.conf) annak kézi szerkesztésével és frissítésével is hozzá lehet adni, de erre a célra az API kérésfeldolgozási űrlap is használható a Caching Proxy konfigurációs és adminisztrációs űrlapjai közül.

Megjegyzések az API használatához

- A Service és a NameTrans utasítás kivételével az egyes lépésekhez tartozó API utasítások konfigurációs fájlbeli megadási sorrendje nincs megkötve. Megjegyezzük, hogy ha az egyes API utasításokhoz több bejegyzés is tartozik, akkor ezek sorrendje fontos; erről a későbbiek során még lesz szó.
- Nem muszáj az összes API lépéshez bejegyzést megadni. Ha egy adott lépéshez nem rendelkezik bedolgozóval, akkor hagyja el a megfelelő utasítást, a rendszer ilyenkor a normál feldolgozási műveleteket végzi el.
- A Service és a NameTrans utasítás a többi leképező utasításhoz hasonlóan működik (mint például a Pass utasítás), valamint függenek előfordulásuktól és a többi leképezési utasítás a konfigurációs fájlon belüli elhelyezkedésétől. A /cgi-bin/pelda.so fájlra vonatkozó szabálynak például a /cgi-bin/* szabály előtt kell szerepelnie.

  Ez azt jelenti, hogy a szerver a Service, a NameTrans, az Exec, a Fail, a Map, a Pass, a Proxy, a ProxyWAS és a Redirect utasításokat a konfigurációs fájlon belüli előfordulásuk sorrendjében dolgozza fel. Ha a szerver sikeresen megfeleltetett egy URL címet egy fájlnak, akkor az említett utasítások közül a fennmaradókat már nem olvassa be és dolgozza fel. (A Map utasítás kivételt képet. A proxyszerverek leképezési szabályaikról részletesebb információkat a *WebSphere Application Server Caching Proxy adminisztrációs útmutató* tartalmaz.)
- Adott lépéshez több konfigurációs utasítás is tartozhat. Lehetőség van például két eltérő bedolgozó függvényre hivatkozó NameTrans utasítás megadására. Amikor a szerver elvégzi a névfordítási lépést, a névfordítási függvényeket a konfigurációs fájlban való megadásuk sorrendjében futtatja le.

Megjegyzés: Ha egy a Caching Proxyval megadott bedolgozó függvény ugyanazt az API utasítást alkalmazza, mint egy saját bedolgozó, akkor a bedolgozó utasítását a rendszer bedolgozó utasítása után kell elhelyezni.

- Bizonyos bedolgozó függvényeket nem kell minden kéréshez lefuttatni:
  - Számos utasítás URL-maszkot is tartalmaz. Ha ezekhez az utasításokhoz URL-maszk is tartozik, akkor a bedolgozó alkalmazás csak azokhoz a kérésekhez hívódik meg, amelyek URL címe egyezést mutat a mintával. További információk arról, hogy mely lépések alkalmasak URL-maszkok használatára: “API utasítások és szintaxis”; valamint információk a szolgáltatás használatáról: “API utasítások változói”
  - Ha egy bedolgozót csak bizonyos típusú hitelesítéseknél szeretne meghívni, akkor az Authentication utasítással adhatja meg, hogy melyik bedolgozó meghívására van szükség. A HTTP protokoll jelenleg csak az alapszintű hitelesítést támogatja. További információk: “API utasítások változói”
- Ha a szerver nem tudja betölteni valamelyik bedolgozó függvényt, illetve olyan ServerInit utasítással rendelkezik, amely nem ad OK visszatérési kódot, akkor az adott Caching Proxy bedolgozóhoz nem kerül sor további bedolgozó függvények meghívására. A rendszer minden az adott pontig végrehajtott, az adott bedolgozóra jellemző feldolgozást figyelmen kívül hagy. Az ezekben az utasításokban megadott egyéb Caching Proxy bedolgozókat és függvényeiket mindez nem érinti.

API utasítások és szintaxis

A konfigurációs fájlbeli utasításoknak az ibmproxy.conf fájlban egyetlen sorban kell szerepelniük, az itt kifejezetten megadottak kivételével szóközök nélkül. Bár a megadott szintaxispéldákban az olvashatóság kedvéért szerepel néhány sortörés, az utasítások tényleges megadásakor ezeken a pontokon nem lehet szóköz.

4. táblázat: Caching Proxy bedolgozó API utasítások

ServerInit		*/elérési_út/fájl:függvény_neve inicializálási_karaktersorozat*
PreExit		*/elérési_út/fájl:függvény_neve*
Authentication	*típus*	*/elérési_út/fájl:függvény_neve*
NameTrans	*/URL*	*/elérési_út/fájl:függvény_neve*
Authorization	*/URL*	*/elérési_út/fájl:függvény_neve*
ObjectType	*/URL*	*/elérési_út/fájl:függvény_neve*
PostAuth		*/elérési_út/fájl:függvény_neve*
Service	*/URL*	*/elérési_út/fájl:függvény_neve*
Midnight		*/elérési_út/fájl:függvény_neve*
Transmogrifier		*/elérési_út/fájl:megnyitás_függvény_neve: írás_függvény_neve: lezárás_függvény_neve:hiba_függvény*
Log	*/URL*	*/elérési_út/fájl:függvény_neve*
Error	*/URL*	*/elérési_út/fájl:függvény_neve*
PostExit		*/elérési_út/fájl:függvény_neve*
ServerTerm		*/elérési_út/fájl:függvény_neve*
ProxyAdvisor		*/elérési_út/fájl:függvény_neve*
GCAdvisor		*/elérési_út/fájl:függvény_neve*

API utasítások változói

Az utasításokban használt változók a következő jelentésekkel bírnak:

típus Csak az Authentication utasításnál szerepel, meghatározza, hogy a bedolgozó függvény meghívásra kerül-e vagy sem. Az érvényes értékek a következők:

- Basic — A bedolgozó függvény meghívására csak az alapszintű hitelesítési kéréseknél kerül sor.
- * — A rendszer minden kérésnél meghívja a bedolgozó függvényt. A HTTP protokoll jelenleg csak az alapszintű hitelesítést támogatja. A nem alapszintű hitelesítési kérések esetében lehetőség van hibakód visszaadására, jelezve, hogy az adott típusú hitelesítés nem támogatott.

URL Azokat a kéréseket határozza meg, amelyek esetében meg kell hívni a bedolgozó függvényt. A sablonnal egyezést mutató URL című kérések hatására a rendszer igénybe veszi a bedolgozó függvényt. Az URL címeket ezekben az utasításokban virtuálisan kell megadni (a protokoll megadása nélkül), a bevezető perjelet (/) viszont nem szabad elhagyni. Például a /www.ics.raleigh.ibm.com helyes, a http://www.ics.raleigh.ibm.com ellenben nem. Az értéket meghatározott URL címként vagy sablonként lehet megadni.

- meghatározott URL — A bedolgozó függvény meghívására csak a pontos URL esetében kerül sor.
- URL-sablon — A bedolgozó függvény meghívására minden a sablonnal egyezést mutató URL esetében sor kerül. A sablonokban szerepelhet a * helyettesítő karakter, és /*URL** vagy /* vagy * formában adhatók meg.

  Megjegyzés: Ha útvonalfordításra van szükség, akkor a Service utasításnál *kötelező* URL-sablont megadni.

elérési út/fájl
A lefordított program teljes képzésű fájlneve.

függvény_neve
A bedolgozó függvénynek a programon belül adott név.

A Service utasítás esetében egy csillagot (*) kell illesztenie a függvény neve után, amennyiben az elérési út információkhoz is hozzá szeretne férni.

inicializálási_karaktersorozat
A ServerInit utasítás elhagyható része, tetszőleges a bedolgozó függvénynek átadni kívánt szöveget tartalmazhat. Az INIT_STRING változóból a httpd_getvar() függvénnyel lehet kibontani a szöveget.

További információkat, ideértve az utasítások szintaxisát is, a *WebSphere Application Server Caching Proxy adminisztrációs útmutató* tartalmaz.

Kompatibilitás más alkalmazás programozási felületekkel

A Caching Proxy API a 4.6.1-es változatig visszafelé kompatibilis az ICAPI és a GWAPI felülettel.

CGI programok portolása

A C nyelven írt CGI alkalmazások a Caching Proxy API használatára végzett portolásakor kövesse az alábbi irányelveket:

- Távolítsa el vagy nevezze át a main() belépési pontot, így lehetősége nyílik DLL összeállítására.
- Távolítsa el a globális változókat, vagy lássa el őket kölcsönös kizárást biztosító szemaforral.
- A programokban a következő hívásokat kell módosítani:
  - A printf() fejléchívásokat módosítsa HTTPD_set() vagy httpd_setvar() hívásokra.

  - A printf() adathívásokat változtassa HTTPD_write() hívásokra.
  - A getenv() hívásokat változtassa HTTPD_extract() vagy httpd_getvar() hívásokra. Megjegyezzük, hogy ilyenkor le nem foglalt memóriát kap vissza, ezért az eredményt fel kell szabadítania.
- Tartsa szem előtt, hogy a szerver többszálú környezetben fut, és a bedolgozó függvényeknek szálbiztosaknak kell lenniük. A függvények újra hívhatósága nem okozza a teljesítmény romlását.
- Ha a HTTPD_write() hívással adatokat ad át az ügyfélnek, akkor ne felejtse el megadni a Content-Type fejlécet.
- Gondosan ellenőrizze, hogy a kód nem okoz-e memóriaszivárgást.
- Gondolja át a hibajelzések útvonalát. Ha maga állítja elő a hibaüzeneteket, és azokat HTML formátumban adja át, akkor saját szolgáltatási függvényéből vagy függvényeiből HTTPD_OK visszatérési kódot kell adnia.

Caching Proxy API - referencia

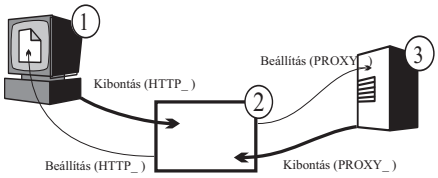
Változók

API programok írásakor a Caching Proxy változói révén információkat lehet szerezni a távoli ügyfélről és a szerverrendszerről.

Megjegyzések:

- A felhasználó által megadott változók neve nem kaphat SERVER_ előtagot. A Caching Proxy API függvénye minden SERVER_ kezdetű változót a szerver számára tart fenn, ezért ezek a változók csak olvashatók. A HTTP_ és a PROXY_ előtagok - a HTTP-fejlécek számára - szintén fenntartottak.
- Az ügyfél által küldött kérésfejlécek (mint például Set-Cookie) mindegyikének előtagja HTTP_, és lehetőség van értékük kinyerésére. A kérésfejléc-változókat a változók nevét HTTP_ előtaggal ellátva lehet elérni. Az előre meghatározott httpd_setvar() függvény segítségével új változókat is létre lehet hozni. A fejlécekkel kapcsolatban a következő részben talál részletesebb információkat: “Az előre meghatározott függvények és makrók visszatérési kódjai” oldalszám: 21.
- Azt, hogy az egyes változók a kérés vagy a válasz fejlécéhez tartoznak, két változóelőtag, a HTTP_ és a PROXY_ segítségével lehet eldönteni. A HTTP_ előtag az ügyfél és a Caching Proxy között továbbított változókat jelöli. A PROXY_ előtag a Caching Proxy és a származási szerver (illetve a proxyláncolat következő szervere) között továbbított változókat jelöli. A változók csak a kérésfeldolgozási lépések során érvényesek.
  - Egy HTTP_* változó kibontásával megkapható az ügyfél a proxyszervernek elküldött kérésében szereplő fejléc értéke.
  - Egy HTTP_* változó beállításával megadható a proxyszerver által az ügyfélnek küldött válaszfejléc.
  - Egy PROXY_* változó kibontásával megkapható egy a tartalomszerverről a proxyszervernek továbbított fejléc értéke.
  - Egy PROXY_* változó beállításával megadható a proxyszerver által a tartalomszervernek (illetve a proxyláncolat következő szerverének) küldött kérésfejléc.

  2. ábra: oldalszám: 26 - Az előtagok az ügyfélkérések a Caching Proxy általi kezelésekor való használatának szemléltetései.



2. ábra: HTTP_ és PROXY_ változóelőtagok. Jelmagyarázat: 1—Ügyfélgép 2—Caching Proxy 3—Származási szerver

- A változók egy része csak olvasható. A csak olvasható változók olyan értékeket hordoznak, amelyeket kérésekből vagy válaszokból lehet kinyerni, és az előre meghatározott httpd_getvar() függvényben lehet használni. Ha a httpd_setvar() függvénnyel megpróbál megváltoztatni egy csak olvasható változót, akkor visszatérési értékként a HTTPD_READ_ONLY értéket kapja.
- A nem írásvédett változókat a httpd_getvar() és a httpd_setvar() előre meghatározott függvénnyel lehet olvasni és írni. Ezek a változók kérésekből vagy válaszokból kinyerhető értékeket tartalmaznak; illetve olyan értékeket, amelyeket a kérés vagy a válasz feldolgozásakor lehet beállítani vagy létrehozni.

Változódefiníciók

Megjegyzés: A HTTP_ vagy PROXY_ előtag nélküli fejlécváltozók kétértelműek. A kétértelműség elkerülése érdekében a fejlécváltozók nevének megadásakor mindig használja a HTTP_ vagy a PROXY_ előtagot.

ACCEPT_RANGES

Az Accept-Ranges válaszfejléc értékét tartalmazza, amely azt adja meg, hogy a tartalomszerver tud-e tartománykérésekre válaszolni. A tartalomszerver által a proxynak küldött fejlécértéket a PROXY_ACCEPT_RANGES segítségével lehet kinyerni. A proxy által az ügyfélnek küldött fejléc értékét a HTTP_ACCEPT_RANGES segítségével lehet beállítani.

Megjegyzés: Az ACCEPT_RANGES kétértelmű. A kétértelműség elkerülése érdekében alkalmazza helyette a HTTP_ACCEPT_RANGES és a PROXY_ACCEPT_RANGES megnevezést.

ALL_VARIABLES

Csak olvasható. Az összes CGI változót tartalmazza. Például:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING

Csak olvasható. Ha a szerver támogatja az ügyfelek hitelesítését, akkor ez a karaktersorozat tartalmazza a visszafejtett, az ügyfél hitelesítésére használható hitelesítési adatokat.

AUTH_TYPE

Csak olvasható. Ha a szerver támogatja az ügyfelek hitelesítését, és a parancsfájl védett, akkor ez a változó adja meg az ügyfél hitelesítésére alkalmazott módszert. Például: Basic.

CACHE_HIT

Csak olvasható. Azt adja meg, hogy a proxykérést sikerült-e megtalálni a gyorsítótárban. A visszaadott értékek a következők lehetnek:

- 0 - A kérést nem sikerült megtalálni a gyorsítótárban.
- 1 - A kérést sikerült megtalálni a gyorsítótárban.

CACHE_MISS

Csak írható. Gyorsítótár-tévesztés kényszerítésére használható. Az érvényes értékek a következők:

- 0 - Nincs gyorsítótár-tévesztés kényszerítve.
- 1 - Gyorsítótár-tévesztés kényszerítése.

CACHE_TASK

Csak olvasható. Megadja, hogy sor került-e a gyorsítótár használatára. A visszaadott értékek a következők lehetnek:

- 0 - A kérés nem érte el, illetve nem frissítette a gyorsítótárat.
- 1 - A kérés kiszolgálása a gyorsítótárból történt.
- 2 - A kért objektum megtalálható volt a gyorsítótárban, de újra kellett ellenőrizni.
- 3 - A kért objektum nem volt megtalálható a gyorsítótárban, és valószínűleg hozzá lett adva.

A változó a PostAuthorization, a PostExit, a ProxyAdvisor és a Log lépés során használható.

CACHE_UPDATE
Csak olvasható. Azt adja meg, hogy a proxykérés frissítette-e a gyorsítótárat. A visszaadott értékek a következők lehetnek:

- 0 - A gyorsítótár nem frissült.
- 1 - A gyorsítótár frissült.

CLIENT_ADDR or CLIENTADDR
Ugyanaz, mint a REMOTE_ADDR.

CLIENTMETHOD
Ugyanaz, mint a REQUEST_METHOD.

CLIENT_NAME or CLIENTNAME
Ugyanaz, mint a REMOTE_HOST.

CLIENT_PROTOCOL or CLIENTPROTOCOL
Az ügyfél által a kérés intézésére használt protokoll nevét és változatát tartalmazza. Például: HTTP/1.1.

CLIENT_RESPONSE_HEADERS
Csak olvasható. A szerver által az ügyfélnek elküldött fejléceket tartalmazó puffert adja vissza.

CONNECTIONS
Csak olvasható. A kiszolgált kapcsolatok vagy az aktív kérések számát tartalmazza. Például: 15.

CONTENT_CHARSET
A text/* válasz karakterkészlete, például US ASCII. A változó kibontása az ügyfél content-charset fejlécére vonatkozik. Beállítása a tartalomszerverhez intézett content-charset fejlécet érinti.

CONTENT_ENCODING
A dokumentumban alkalmazott kódolást adja meg, mint például x-gzip. A változó kibontása az ügyféltől kapott content-encoding fejlécre vonatkozik. Beállítása a tartalomszerverhez intézett content-charset fejlécet érinti.

CONTENT_LENGTH
A változó kibontása az ügyfél kérésének fejlécére vonatkozik. Beállítása a tartalomszerverhez intézett kérés fejlécének értékét határozza meg.

Megjegyzés: A CONTENT_LENGTH kétértelmű. A kétértelműség elkerülése érdekében alkalmazza a HTTP_CONTENT_LENGTH és a PROXY_CONTENT_LENGTH megnevezést.

CONTENT_TYPE
A változó kibontása az ügyfél kérésének fejlécére vonatkozik. Beállítása a tartalomszerverhez intézett kérés fejlécének értékét határozza meg.

Megjegyzés: A CONTENT_TYPE kétértelmű. A kétértelműség megszűntetése érdekében alkalmazza a HTTP_CONTENT_TYPE és a PROXY_CONTENT_TYPE megnevezést.

CONTENT_TYPE_PARAMETERS
Egyéb, a karakterkészlettől eltérő MIME attribútumokat tartalmaz. A változó kibontása az ügyfél kérésének fejlécére vonatkozik. Beállítása a tartalomszervernek küldött kérés fejlécének értékét érinti.

DOCUMENT_URL
Az egységes erőforrás-mutatót (URL) tartalmazza. Például:

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

DOCUMENT_URI
Ugyanaz, mint a DOCUMENT_URL.

DOCUMENT_ROOT
Csak olvasható. A dokumentum gyökér elérési útját tartalmazza, az átadási szabályok szerint.

ERRORINFO
A hibalap meghatározásához használható hibakódot tartalmazza. Például: blocked.

EXPIRES
Azt adja meg, hogy a proxy gyorsítótárában tárolt dokumentum mikor évül el. A változó kibontása az ügyfél kérésében szereplő fejlécre vonatkozik. Beállítása a tartalomszerverhez intézett kérés fejlécének értékét érinti. Például:

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

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

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

GC_EVALUATION
Write-only. This floating-point value determines whether to remove (0.0) or keep (1.0) the file being considered for garbage collection. A 0,0 és az 1,0 közötti értékek sorrendezettek, vagyis a 0,1-es GC_EVALUATION értékű fájlt a rendszer nagyobb valószínűséggel távolítja el, mint a 0,9-es GC_EVALUATION értékkel rendelkezőt.

GC_EXPIRES
Csak olvasható. Azt adja meg, hogy az adott fájl hány másodperc múlva évül el a gyorsítótárban. A változót csak GC Advisor bedolgozó tudja kibontani.

GC_FILENAME
Csak olvasható. Megadja, hogy a fájlt érinti-e a szemétgyűjtés. A változót csak GC Advisor bedolgozó tudja kibontani.

GC_FILESIZE
Csak olvasható. A fájl a szemétgyűjtés során figyelembe vett méretét adja meg. A változót csak GC Advisor bedolgozó tudja kibontani.

GC_LAST_ACCESS
Csak olvasható. A fájl utolsó elérésének idejét adja meg. A változót csak GC Advisor bedolgozó tudja kibontani.

GC_LAST_CHECKED
Csak olvasható. A fájlok utolsó ellenőrzésének időpontját adja meg. A változót csak GC Advisor bedolgozó tudja kibontani.

GC_LOAD_DELAY
Csak olvasható. Megadja, hogy mennyi ideig tartott a fájl lekérése. A változót csak GC Advisor bedolgozó tudja kibontani.

HTTP_COOKIE
Beolvasáskor a változó a Set-Cookie fejléc az ügyfél által beállított értékét tartalmazza. Új cookie a válaszadatfolyamban (a proxy és az ügyfél közötti

adatfolyam) való beállítására is alkalmas. A változó beállításának hatására létrejön egy új Set-Cookie fejléc a dokumentumkérési adatfolyamban, függetlenül attól, hogy létezik-e már ilyen fejléc vagy sem.

HTTP_HEADERS
Csak olvasható. Az ügyfélkérés összes fejlécének kibontására használható.

HTTP_REASON
A változó beállítása a HTTP-válaszban található ok-karaktersorozatot érinti. Beállításával a proxy az ügyfélnek adott válaszában szereplő ok-karaktersorozat is megváltozik. A változó kibontásával megkapható a tartalomszerver által a proxynak adott válaszban szereplő ok-karaktersorozat.

HTTP_RESPONSE
A változó beállítása a HTTP-válaszban szereplő válaszkódot módosítja. Beállításával a proxy az ügyfélnek adott válaszában szereplő állapotkód is megváltozik. A változó kibontásával megkapható a tartalomszerver által a proxynak adott válaszban szereplő állapotkód.

HTTP_STATUS
A HTTP-válaszkódot és az ok-karaktersorozatot tartalmazza. Például: 200 OK.

HTTP_USER_AGENT
A User-Agent kérésfejléc értékét tartalmazza, ami megegyezik az ügyfél böngészőjének nevével, mint például: Netscape Navigator / V2.02. A változó beállításával megváltozik a proxy az ügyfélnek adott válaszában szereplő fejléc. A változó kibontása az ügyfél kérésében szereplő fejlécre vonatkozik.

INIT_STRING
Csak olvasható. Ezt a karaktersorozatot a ServerInit utasítás határozza meg. A változó kiolvasására csak a szerverinicializálási lépés során van lehetőség.

LAST_MODIFIED
A változó kibontása az ügyfél kérésének fejlécére vonatkozik. Beállítása a tartalomszerverhez intézett kérés fejlécének értékét határozza meg. Például:

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

LOCAL_VARIABLES
Csak olvasható. Az összes felhasználó által megadott változó.

MAXACTIVETHREADS
Csak olvasható. Az aktív szálak maximális száma.

NOTMODIFIED_TO_OK
Teljes választ kényszerít az ügyfél felé. Csak a PreExit és a ProxyAdvisor lépés során érvényes.

ORIGINAL_HOST
Csak olvasható. Egy kérés céljának IP címét vagy hosztnevét adja vissza.

ORIGINAL_URL
Csak olvasható. Az ügyfélkérésben elküldött eredeti URL címet adja vissza.

OVERRIDE_HTTP_NOTRANSFORM
Cache-Control: no-transform fejléc jelenlétekor lehetővé teszi adatok módosítását. A változó beállítása az ügyfélnek adott válaszfejlécet érinti.

OVERRIDE_PROXY_NOTRANSFORM
Cache-Control: no-transform fejléc jelenlétekor lehetővé teszi adatok módosítását. A változó beállítása a tartalomszerverhez intézett kérést módosítja.

PASSWORD
Alapszintű hitelesítésnél a visszafejtett jelszót tartalmazza. Például: jelszó.

PATH A teljesen lefordított elérési utat tartalmazza.

PATH_INFO
Az elérési úttal kapcsolatos kiegészítő információkat tartalmazza, a webböngésző által elküldött formában. Például: /adatok.

PATH_TRANSLATED
A PATH_INFO által tartalmazott, az elérési úttal kapcsolatos információk visszafejtett vagy lefordított változatát tartalmazza. Például:

```
d:\www\adatok
```

```
/www/adatok
```

PPATH
A részlegesen lefordított elérési utat tartalmazza. A Name Translation lépésben használható.

PROXIED_CONTENT_LENGTH
Csak olvasható. A proxyszerveren keresztül ténylegesen továbbított válaszadatok hosszát adja vissza.

PROXY_ACCESS
Megadja, hogy a kérés proxykérés-e. Például: NO.

PROXY_CONTENT_TYPE
A HTTPD_proxy() híváson keresztül végrehajtott proxykérés Content-Type fejlécét tartalmazza. Ha az információk küldése POST módszerrel történik, a változó a megadott adatok típusát tartalmazza. A proxyszerver konfigurációs fájljában saját tartalomtípust is létrehozhat, és azt hozzárendelheti a megfelelő megjelenítőhöz. A változó kibontása a tartalomszerver válaszában szereplő fejlécértékre vonatkozik. Beállítása a tartalomszervernek küldött kérés fejlécét érinti. Például:

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

PROXY_CONTENT_LENGTH
A HTTPD_proxy() híváson keresztül végrehajtott proxykérés Content-Length fejlécét tartalmazza. Ha az információk küldése POST módszerrel történik, ez a változó adja meg az adatkarakterek számát. A szerverek általában nem küldenek fájl vége jelzőt, ha a szabványos bemenet alkalmazásával továbbítanak információkat. Szükség esetén a CONTENT_LENGTH értékkel határozhatja meg a bemeneti karaktersorozat végét. A változó kibontása a tartalomszerver válaszában kapott fejlécértékre vonatkozik. Beállítása a tartalomszervernek küldött kérés fejlécét érinti. Például:

```
7034
```

PROXY_COOKIE
Beolvasáskor a változó a Set-Cookie fejléc a származási szerver által beállított értékét tartalmazza. Új cookie a kérésfolyamban való beállítására is alkalmas. A változó beállításának hatására létrejön egy új Set-Cookie fejléc a dokumentumkérési adatfolyamban, függetlenül attól, hogy létezik-e már ilyen fejléc vagy sem.

PROXY_HEADERS
Csak olvasható. A Proxy fejlécek kibontására használható.

PROXY_METHOD
A HTTPD_proxy() híváson keresztül intézett kérés módszere. A változó kibontása a tartalomszerver válaszában kapott fejlécértékre vonatkozik. Beállítása a tartalomszervernek küldött kérés fejlécét érinti.

QUERY_STRING
Ha az információk küldése a GET metódussal történik, akkor ez a változó tartalmazza a lekérdezésnek a kérdőjelet (?) követő részét. Az információk visszafejtését a CGI programnak kell elvégeznie. Például:

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

RCA_OWNER
Csak olvasható. Számértékkel tér vissza, megadva azt a csomópontot, amely a kért objektumot birtokolta. A változó a PostExit, a ProxyAdvisor és a Log lépés során használható, és csak akkor van érdemi szerepe, ha a szerver távoli gyorsítótár-elérést (RCA) alkalmazó gyorsítótártömb része.

RCA_TIMEOUTS
Csak olvasható. Számértéket ad vissza, amely az összes partnerhez intézett minden RCA kérésen fellépett időtúllépések összesített számát adja meg. A változó bármelyik lépésben használható.

REDIRECT_*
Csak olvasható. Egy átirányítási karaktersorozatot tartalmaz a változónévhez tartozó hibakódhoz (például: ÁTIRÁNYÍTÁSI_URL). A REDIRECT_ változók lehetséges értékei az Apache webszerver online dokumentációjában található meg, amely a http://httpd.apache.org/docs-2.0/custom-error.html címen érhető el.

REFERRER_URL
Csak olvasható. A böngésző utolsó URL címét tartalmazza. Lehetővé teszi, hogy az ügyfél - a szerver üzemeltetőjét segítve - megadja annak az erőforrásnak a címét (URL címét), amelytől a Request-URL-t kapta. Például:

```
http://www.vallalat.com/honlap
```

REMOTE_ADDR
A webböngésző IP címét tartalmazza, amennyiben elérhető. Például: 45.23.06.8.

REMOTE_HOST
A webböngésző hosztnevét tartalmazza, amennyiben elérhető. Például: www.raleigh.ibm.com.

REMOTE_USER
Ha a szerver támogatja az ügyfelek hitelesítését, és a parancsfájl védett, akkor ez a változó a hitelesítés céljából átadott felhasználónevet tartalmazza. Például: janos_felhasznalo.

REQHDR
Csak olvasható. Az ügyfél által elküldött fejlécek listáját tartalmazza.

REQUEST_CONTENT_TYPE
Csak olvasható. A kérés törzsének tartalomtípusát adja vissza. Például:

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

REQUEST_CONTENT_LENGTH
Csak olvasható. Ha az információk küldése POST módszerrel történik, ez a változó adja meg az adatkarakterek számát. A szerverek általában nem küldenek fájl vége jelzőt, ha a szabványos bemenet alkalmazásával továbbítanak információkat. Szükség esetén a CONTENT_LENGTH értékkel határozhatja meg a bemeneti karaktersorozat végét. Például: 7034.

REQUEST_METHOD
Csak olvasható. A kérés elküldésére használt módszert (a HTML űrlap METHOD attribútumában van meghatározva) adja meg. Például: GET vagy POST.

REQUEST_PORT
Csak olvasható. Az URL címben megadott portszámot, illetve a protokoll alapján egy alapértelmezett portszámot ad vissza.

RESPONSE_CONTENT_TYPE
Csak olvasható. Ha az információk küldése POST módszerrel történik, a változó a

megadott adatok típusát tartalmazza. A proxyszerver konfigurációs fájljában saját tartalomtípust is létrehozhat, és azt hozzárendelheti a megfelelő megjelenítőhöz. Például: text/html.

RESPONSE_CONTENT_LENGTH
Csak olvasható. Ha az információk küldése POST módszerrel történik, ez a változó adja meg az adatkarakterek számát. A szerverek általában nem küldenek fájl vége jelzőt, ha a szabványos bemenet alkalmazásával továbbítanak információkat. Szükség esetén a CONTENT_LENGTH értékkel határozhatja meg a bemeneti karaktersorozat végét. Például: 7034.

RULE_FILE_PATH
Csak olvasható. A konfigurációs fájl teljes képzésű fájlrendszerbeli elérési útját és fájlnevét tartalmazza.

SSL_SESSIONID
Csak olvasható. Az SSL munkamenet azonosítóját adja vissza, ha az aktuális kérés SSL-kapcsolat felett érkezett. Ha az aktuális kérés nem SSL-kapcsolat felett érkezett, akkor NULL értéket ad.

SCRIPT_NAME
A kérés URL címét tartalmazza.

SERVER_ADDR
Csak olvasható. A proxyszerver helyi IP címét tartalmazza.

SERVER_NAME
Csak olvasható. A proxyszerver hosztnevét vagy a tartalomszerver IP címét tartalmazza a kéréshez. Például: www.ibm.com.

SERVER_PORT
Csak olvasható. A proxyszerver azon portjának számát tartalmazza, amelyre az ügyfél kérése beérkezett. Például: 80.

SERVER_PROTOCOL
Csak olvasható. A kérés elküldésére alkalmazott protokoll nevét és változatát tartalmazza. Például: HTTP/1.1.

SERVER_ROOT
Csak olvasható. A proxyszerverprogram telepítési könyvtárát tartalmazza.

SERVER_SOFTWARE
Csak olvasható. A proxyszerver nevét és változatát tartalmazza.

STATUS
A HTTP-válaszkódot és az ok-karaktersorozatot tartalmazza. Például: 200 OK.

TRACE
Meghatározza, hogy a rendszer mennyi információt fog nyomkövetni. A lehetséges visszatérési értékek:

- OFF - Nincs nyomkövetés.
- V - Részletes mód.
- VV - Nagyon részletes mód.
- MTV - Rendkívül részletes mód.

URI Írható/olvasható. Ugyanaz, mint a DOCUMENT_URL.

URI_PATH
Csak olvasható. Egy URL elérési út részét adja vissza.

URL Írható/olvasható. Ugyanaz, mint a DOCUMENT_URL.

URL_MD4
Csak olvasható. A jelenlegi kérés lehetséges gyorsítótárbeli fájljának nevét adja vissza.

USE_PROXY
Azt a proxyt adja meg, amelyhez az aktuális kérés kapcsán láncolni kell. Adja meg az URL címet. Például: http://proxy:8080.

USERID
Ugyanaz, mint a REMOTE_USER.

USERNAME
Ugyanaz, mint a REMOTE_USER.

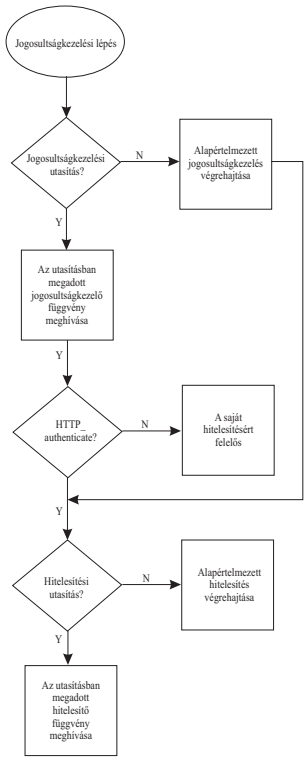
Hitelesítés és jogosultságkezelés

A szakkifejezések rövid áttekintése:

Hitelesítés
A kéréshez tartozó biztonsági tokenek ellenőrzése, amelynek során a rendszer megbizonyosodik a kérő személyazonosságáról.

Jogosultságkezelés
Az a folyamat, amely biztonsági tokenek alkalmazásával meghatározza, hogy a kérőnek joga van-e az erőforrás elérésére.

3. ábra: oldalszám: 35 - A proxyszerver hitelesítési és jogosultságkezelési folyamata.



3. ábra: A proxyszerver hitelesítési és jogosultságkezelési folyamata

Amint a 3. ábra: látható, a szerver hitelesítési és jogosultságkezelési folyamatának első lépése a hitelesítés kezdeményezése.

A Caching Proxy esetében a hitelesítés a jogosultságkezelési folyamat része, csak akkor kerül rá sor, amikor jogosultságkezelésre van szükség.

Hitelesítési és jogosultság-kezelési folyamat

A proxyszerver a jogosultság-kezelést igénylő kérések feldolgozásakor a következő lépéseket követi.

1. Először a proxyszerver megvizsgálja saját konfigurációs fájlját, és meghatározza, hogy szerepel-e benne authorization utasítás.

- Ha a konfigurációs fájlban szerepel authorization utasítás, akkor a szerver meghívja az utasításban megadott jogosultság-kezelő függvényt, és a 2. lépéssel megkezdi a hitelesítést.
- Ha nem talál authorization utasítást, a szerver elvégzi az alapértelmezett jogosultság-kezelést, majd közvetlenül a 3. lépés hitelesítési eljárásaira lép.

2. A proxyszerver azzal kezdi a hitelesítési folyamatot, hogy megvizsgálja, az ügyfél kérésében található-e HTTP_authenticate fejléc.
   - Ha talál ilyen fejlécet, a szerver folytatja a hitelesítési folyamatot (lásd: 3. lépés).
   - Ha nincs ilyen fejléc, akkor a hitelesítést másik módszerrel kell elvégezni.
3. A proxyszerver ellenőrzi, hogy van-e authentication utasítás a proxykonfigurációs fájlban.
   - Ha a konfigurációs fájlban van authentication utasítást, akkor a szerver meghívja az utasításban megadott hitelesítési függvényt.
   - Ha nincs ilyen utasítás, akkor a szerver az alapértelmezett hitelesítést végzi el.

Ha a saját készítésű Caching Proxy bedolgozó saját jogosultságkezelési folyamattal rendelkezik, akkor az *felülbírálja* a szerver alapértelmezett jogosultságkezelési és hitelesítési eljárását. Ha tehát a konfigurációs fájlban van authorization utasítás, akkor a hozzájuk társított bedolgozó függvényeknek szintén kezelniük kell a szükséges hitelesítési eljárásokat. Az előre meghatározott HTTPD_authenticate() függvény a rendelkezésére áll.

A jogosultságkezelő bedolgozókban háromféle módon lehet hitelesítést végezni:

- Írjon saját, külön jogosultságkezelő és hitelesítő bedolgozót. A proxykonfigurációs fájlban az Authorization és az Authentication utasítások segítségével adja meg ezeket a függvényeket. Ügyeljen arra, hogy saját jogosultságkezelő bedolgozójából meghívja a HTTPD_authenticate() függvényt.

  Az Authorization lépés végrehajtásakor lefut a saját készítésű jogosultságkezelő bedolgozó függvény, amely viszont meghívja a hitelesítő bedolgozó függvényt.
- Írjon saját jogosultságkezelő bedolgozó függvényt, de a szerver alapértelmezett hitelesítési eljárását hívja meg vele. A proxykonfigurációs fájlban az Authorization utasítással adja meg a saját függvényt. Ebben az esetben az Authentication utasításra nincs szükség. Ne feledkezzen meg arról, hogy saját jogosultságkezelő függvényéből meg kell hívnia a HTTPD_authenticate() függvényt.

  Az Authorization lépés végrehajtásakor lefut a saját készítésű bedolgozó függvény, amely viszont meghívja a szerver alapértelmezett hitelesítési eljárását.
- Írjon saját jogosultságkezelő bedolgozó függvényt, és ebben valósítson meg minden szükséges hitelesítési műveletet. Saját jogosultságkezelő bedolgozójában ne használja a HTTPD_authenticate() függvényt. A proxykonfigurációs fájlban az Authorization utasítással adja meg a saját készítésű jogosultságkezelési bedolgozót. Ebben az esetben az Authentication utasításra nincs szükség.

  Az Authorization lépés elvégzésekor lefut a jogosultságkezelő bedolgozó függvény is, illetve minden benne szereplő hitelesítési eljárás.

Testreszabott hitelesítésre az alábbi megoldással akkor is van lehetőség, ha a saját Caching Proxy bedolgozó nem biztosít külön jogosultságkezelési eljárást:

- Készítsen saját hitelesítő bedolgozó függvényt. A proxykonfigurációs fájlban az Authentication utasítással adja meg a saját függvényét. Ebben az esetben az Authorization utasítás használatára nincs szükség.

Az Authorization lépés végrehajtásakor a szerver alapértelmezett jogosultságkezelése is lefut, amely viszont meghívja a saját hitelesítő bedolgozó függvényt.

Vegye figyelembe a következőket:

- Ha a konfigurációs fájlban nincs Authorization utasítás, vagy ha az utasításokban megadott bedolgozó függvények a HTTP_NOACTION választ adva elutasítják a kérés kezelését, akkor a szerver az alapértelmezett jogosultságkezelést végzi el.
- Ha a konfigurációs fájlban található Authorization utasítás, és az általa megadott bedolgozó függvény magába foglalja a HTTPD_authenticate() hívást, akkor a szerver minden Authentication utasítással megadott hitelesítő függvényt meghív. Ha nem adott meg Authentication utasítást, vagy ha az utasításokban megadott bedolgozó függvények a HTTP_NOACTION válassza elutasítják a kérés kezelését, akkor a szerver az alapértelmezett hitelesítést végzi el.
- Ha a konfigurációs fájlban található Authorization utasítás, de a hivatkozott bedolgozó függvények nem foglalják magukba a HTTPD_authenticate() hívást, akkor a szerver nem hív meg hitelesítő függvényt. Ekkor a hitelesítést a saját jogosultságkezelő bedolgozó részeként kell megvalósítani, illetve saját hívást kell indítani más hitelesítő modulok felé.
- Ha a saját jogosultságkezelő függvény 401-es vagy 407-es kóddal tér vissza, akkor a Caching Proxy automatikusan előállít egy kihívást (felszólítva a böngészőt egy felhasználói azonosító és jelszó visszaadására). A Caching Proxy védelmi beállításait ekkor is meg kell adni, ellenkező esetben előfordulhat, hogy a művelet hibásan zajlik le.

Variáns gyorsítótárazása

A variánsok gyorsítótárazásával az eredeti dokumentum (URI) módosított példányát lehet gyorsítótárazni. A Caching Proxy az API által előállított variánsok mindegyikét kezeli. A *variánsok* egy alapdokumentum különféle változatai.

Általában jellemző, hogy amikor a származási szerverek elküldik a variánsokat, akkor nem jelzi, hogy valójában variánsokról van szó. A Caching Proxy csak a bedolgozók által (például kódlap-átalakítás miatt) előállított variánsokat támogatja. Ha egy bedolgozó a HTTP-fejlécben nem szereplő feltétel alapján hoz létre variánst, akkor egy PreExit vagy PostAuthorization lépésbeli függvénnyel létre kell hoznia egy álfejlécet, amely alapján a Caching Proxy megfelelően azonosítani tudja a meglévő variánst.

Például egy Transmogrifier API programmal a böngésző által küldött User-Agent fejléc értéke alapján módosítani lehet a felhasználók által kért adatokat. A *lezárás* függvényben mentse el a módosított tartalmat egy fájlba, vagy adjon meg egy pufferhosszt, majd adja át a puffert mint adatargumentumot. Ez után a httpd_variant_insert() és a httpd_variant_lookup() variáns-gyorsítótárazó függvények segítségével helyezze el a tartalmat a gyorsítótárba.

API példák

Ha további segítségre van szüksége saját Caching Proxy API függvényeinek elkészítéséhez, akkor tanulmányozza az Edge összetevők telepítő CD-ROM-lemezének samples könyvtárában található mintaprogramokat. További információkat a WebSphere Application Server webhelyen talál: www.ibm.com/software/webservers/appserv/.

3. fejezet Egyéni tanácsadók

Ebben a részben a Load Balancer egyéni tanácsadóit tárgyaljuk.

Terheléskiegyenlítési információkat biztosító tanácsadók

A tanácsadók szoftveres ügynökök, amelyek a Load Balanceren belül futva információkat nyújtanak egy adott szerver terheléséről. Minden szabványos protokollhoz (HTTP, SSL stb.) külön tanácsadó létezik. A Load Balancer alapkódja rendszeres időközönként végrehajt egy tanácsadói ciklust, amelynek során egyedileg kiértékeli minden a konfigurációjában szereplő szerver állapotát.

Saját Load Balancer tanácsadók készítésével személyre szabhatja a szervergépek terhelésének meghatározására alkalmazott eljárást.

Windows rendszereken: Load Balancer for IPv4 and IPv6 telepítés használatakor, ha IPv6 protokollt alkalmaz a gépen, valamint tanácsadókat kíván futtatni, akkor módosítania kell a C:\windows\system32\drivers\etc\ könyvtárban található **protocol** fájlt.

IPv6 esetében a következő sort kell beillesztenie a protocol fájlba:

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

Normál tanácsadói szolgáltatások

A tanácsadók általában a következő módon segítik a terheléskiegyenlítést.

1. A tanácsadó rendszeres időközönként nyit egy kapcsolatot minden egyes szerverrel, majd elküld neki egy kérésüzenetet. Az üzenet tartalma a szerveren futó protokollra egyedileg jellemző; a HTTP tanácsadó például egy HEAD kérést küld a szervernek.
2. A tanácsadó fogadja a szerver válaszát. A válasz beérkezése után a tanácsadó kiszámítja és jelenti az adott szerver terhelési értékét. A különféle tanácsadók különböző módokon számítják ki a terhelési értékeket, ám a legtöbb tanácsadó a szerver válaszidejét méri, majd terhelésként ezt az ezredmásodpercben mért értéket jelenti.
3. A tanácsadó jelenti a terhelést a Load Balancer kezelő funkciójának. A terhelés a kezelő jelentésének Port oszlopában jelenik meg. A kezelő a tanácsadó által jelentett terhelést a rendszergazda által megadott súllyal együtt veszi figyelembe, amikor meghatározza a bejövő kérések a szerverek közötti terheléskiegyenlítésének módját.
4. Ha egy szerver nem válaszol, a tanácsadó negatív (-1) értéket ad vissza terhelési értékként. A kezelő a kapott információk alapján eldönti, hogy adott szerverhez mikor kell felfüggesztenie a szolgáltatást.

A Load Balancerhez mellékelt normál tanácsadók a következő funkciókhoz, szolgáltatásokhoz végeznek tanácsadást. A tanácsadókról részletes információkat a *WebSphere Application Server Load Balancer adminisztrációs útmutató* tartalmaz.

- Connect, csatlakozás
- DB2
- DNS
- FTP
- HTTP
- HTTPS
- IMAP

- LDAP
- NNTP
- Ping
- POP3
- Reach, elérés
- Self, önmaga
- SIP
- SMTP
- SSL
- Telnet
- WebSphere Application Server
- WebSphere Application Server Caching Proxy
- Workload Manager

Az olyan zárt protokollok támogatásához, amelyekhez nincs normál tanácsadó, saját tanácsadót kell készíteni.

Egyéni tanácsadó készítése

Az egyéni tanácsadók kisméretű Java programok, amelyek osztályfájlok formájában állnak rendelkezésre, a Load Balancer alapkódja hívja meg őket az egyes szerverek terhelésének meghatározására. Az alapkód minden szükséges adminisztrációs feladatot elvégez, ideértve az egyéni tanácsadók indítását és leállítását, az állapotadatok és jelentések átadását, az előzmények naplófájlba rögzítését, valamint a tanácsadók állapotának a kezelő összetevő felé végzett jelentését.

Amikor a Load Balancer alapkódja meghív egy egyedi tanácsadót, a következő lépések végrehajtására kerül sor.

1. A Load Balancer alapkódja kapcsolatot létesít a szervergéppel.
2. Ha a socket megnyílik, az alapkód meghívja a megadott tanácsadó GetLoad függvényét.
3. A tanácsadó GetLoad függvénye végrehajtja a felhasználó által a szerver állapotának meghatározásához előírt lépéseket, ideértve a szerver válaszára való várakozást is. A függvény futtatása a válasz beérkezésekor fejeződik be.
4. A Load Balancer alapkódja lezárja a szerver felé megnyitott socketet, majd jelenti a terhelési információkat a kezelőnek. Attól függően, hogy az egyéni tanácsadó normál vagy helyettesítő módban működik, az alapkód a GetLoad függvény futásának befejeződése után időnként további számításokat is elvégez.

Normál és helyettesítő mód

Az egyéni tanácsadók a Load Balancerrel *normál* vagy *helyettesítő* módú párbeszédet folytathatnak.

A működési mód kiválasztása az egyéni tanácsadó fájljában a konstruktor metódusban, egy paraméter segítségével történik. (Minden tanácsadó az említett módok egyikében működik, tervezésétől függően.)

Normál módban az egyéni tanácsadó adatcserét végez a szerverrel, és a tanácsadó alapkódja végzi az adatcsere időmérését és a terhelési érték kiszámítását. Az alapkód ezt követően jelenti a terhelési értéket a kezelőnek. Az egyéni tanácsadó nulla visszatérési értékkel jelzi a sikert, mínusz egyes értékkel pedig a hibát.

A normál mód kiválasztásához a konstruktor replace jelzőjének *false* értéket kell adni.

Helyettesítő módban az alapkód nem végez időmérést. Az egyéni tanácsadó kódja elvégzi a megadott műveleteket, az egyéni szükségletek szerint, majd átadja a tényleges terhelési értéket. Az alapkód fogadja a terhelési értéket, majd változatlan formában átadja a kezelőnek. A lehető legjobb eredmény elérése érdekében a terhelési értékeket 10 és 1000 közé kell normalizálni, ahol a 10-es érték gyors, az 1000-es érték pedig lassú szervert jelent.

A helyettesítő mód kiválasztásához a konstruktor replace jelzőjének *true* értéket kell adni.

Tanácsadók elnevezési megállapodásai

Az egyéni tanácsadók fájlneveinek esetében az ADV_*név*.java formátumot kell követni, ahol a *név* a tanácsadó számára kiválasztott név. A teljes névnek a nagybetűs ADV_ előtaggal kell kezdődnie, az ezt követő karaktereknek pedig kisbetűknek kell lenniük. A kisbetűk használatára vonatkozó követelmény betartásával biztosítható, hogy a tanácsadó futtatási parancsa érzéketlen legyen a kis- és nagybetűk használatára.

A Java megállapodások szerint a fájlon belül megadott osztály nevének egyeznie kell a fájl nevével.

Fordítás

Az egyéni tanácsadókat Java nyelven kell megírni, és a fejlesztői gépre telepített Java fordítóval kell lefordítani. A fordítás során a következő fájlokra történik hivatkozás:

- Az egyéni tanácsadó fájlja
- Az alaposztályok fájlja, ibmnd.jar, amely a *telepítési_útvonal*/servers/lib könyvtárban található

A fordítás ideje alatt a classpath környezeti változónak az egyéni tanácsadó fájljára és az alaposztályok fájljára egyaránt kell hivatkozást tartalmaznia. A fordítási parancs formátuma például a következő lehet:

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

A példában a Linux és UNIX alatti alapértelmezett telepítési útvonalat használtuk. A tanácsadó fájljának neve ADV_*név*.java, és a fájl az aktuális könyvtárban található.

A fordítás kimenete egy class fájl, például ADV_*név*.class. A tanácsadó elindítása előtt másolja át a class fájlt a *telepítési_útvonal*/servers/lib/CustomAdvisors/ könyvtárba.

Megjegyzés: Az egyéni tanácsadók a fordításkor használt operációs rendszertől eltérő rendszeren is futtathatók. Lehetőség van például arra, hogy a tanácsadó lefordítását Windows alapú rendszeren végezze, majd a kapott class fájlt (bináris formátumban) átmásolja egy linuxosra, majd azon futtassa az egyéni tanácsadót.

Egyéni tanácsadó futtatása

Az egyéni tanácsadó futtatásához először át kell másolni a tanácsadó class fájlját a Load Balancer gép lib/CustomAdvisors/ könyvtárába. Ha például az egyéni tanácsadó neve sajatping, akkor a fájl elérési útja *telepítési_útvonal*/servers/lib/CustomAdvisors/ADV_sajatping.class lesz.

A Load Balancert úgy kell konfigurálni, hogy indítsa el saját kezelő szolgáltatását, majd adja ki az egyéni tanácsadó elindításához szükséges parancsot. Az egyéni tanácsadó megadása a nevével történik, az ADV_ előtag és a fájlkiterjesztés elhagyásával:

```
dscontrol advisor start sajatping portszám
```

A parancsban megadott portszám azt a portot határozza meg, amelyen a tanácsadó kapcsolatot létesít a célszerverrel.

Kötelező rutinok

Ahogy minden tanácsadó, az egyéni tanácsadók is az ADV_Base nevű tanácsadói alaposztályt bővitik ki. A tanácsadói funkciók túlnyomó részét a tanácsadói alap végzi, ilyen például a kezelő súlyozó algoritmusa által használt terhelési értékek jelentése a kezelő felé. A tanácsadói alap feladata a socketek csatlakoztatása és lezárása, valamint a tanácsadó által használt küldési és fogadási metódusok biztosítása is. A tanácsadót a rendszer csak a vizsgált szerver felé, a megadott porton való adatküldésre és -fogadásra használja. A tanácsadói alap által biztosított TCP eljárások időméréssel vannak ellátva, ennek alapján történik a terhelés kiszámítása. A tanácsadó alapjának konstruktorában található egyik jelző szükség esetén alkalmas a meglévő terhelés a tanácsadóktól kapott új terheléssel való felülírására.

Megjegyzés: Egy a konstruktorban beállított érték alapján a tanácsadói alap meghatározott időközönként átadja a terhelést a súlyozó algoritmusnak. Ha a tanácsadó nem fejezte be a feldolgozást, és nem képes érvényes terhelési értéket átadni, akkor a tanácsadói alap az előzőleg jelentett terhelést alkalmazza.

A tanácsadók a következő alaposztálybeli metódusokkal rendelkeznek:

- Konstruktor rutin. A constructor az alaposztály konstruktorát hívja meg.
- ADV_AdvisorInitialize metódus. Az alaposztály inicializálásának elvégzése után további lépések végrehajtását teszi lehetővé.
- getLoad rutin. A tanácsadói alaposztály elvégzi a socket megnyitását; a tanácsadási ciklus befejezéséhez a getLoad függvénynek csak a megfelelő küldési és fogadási kéréseket kell kiadnia.

A kötelező rutinokkal kapcsolatos részleteket a jelen szakasz egy későbbi része tartalmazza.

Keresési sorrend

Az egyéni tanácsadók meghívására a natív, más szóval szabvány tanácsadók keresése után kerül sor. Ha a Load Balancer nem találja valamelyik megadott tanácsadót a szabványos tanácsadók listáján, akkor az egyéni tanácsadók listájához fordul. A tanácsadók használatáról további információkat a *WebSphere Application Server Load Balancer adminisztrációs útmutató* tartalmaz.

Névadás és a fájlok elérési útjai

Az egyéni tanácsadók névadása és elérési útjának megválasztása során vegye figyelembe az alábbi követelményeket.

- Az egyéni tanácsadót csak kisbetűket tartalmazó névvel kell ellátni, ezzel elkerülhetők a kis- és nagybetűk használatából fakadó problémák, amikor az operátorok begépelik a parancsokat a parancssorba. A tanácsadók nevét az ADV_ előtaggal kell ellátni.
- Az egyéni tanácsadó osztályát a lib/CustomAdvisors alkönyvtárban kell elhelyezni. A könyvtár alapértelmezett helye /opt/ibm/edge/lb/servers/lib/CustomAdvisors a Linux és a UNIX rendszereken, illetve C:\Program Files\IBM\edge\lb\servers\lib\CustomAdvisors\ a Windows rendszereken.

Egyéni tanácsadómetódusok és függvényhívások

Konstruktor (a tanácsadó alap által biztosított)

```
void ADV_Base Constructor (
        string sName;
        string sVersion;
        int iDefaultPort;
```

```
        int iInterval;
        string sDefaultLogFileName;
        boolean replace
)
```

sName
Az egyéni tanácsadó neve.

sVersion
Az egyéni tanácsadó változata.

iDefaultPort
A szerverhez való csatlakozás során használt port száma, amennyiben a hívásban nincs ilyen megadva.

iInterval
A tanácsadó ennyi időnként kérdezi le a szervert.

sDefaultLogFileName
Kötelező paraméter, de a rendszer nem használja. Az egyetlen elfogadható értéke az üres karaktersorozat ("").

replace
Megadja, hogy a tanácsadó *helyettesítő* módban működjön vagy sem. Lehetséges értékei a következők:

- true – Alkalmazásakor a tanácsadói alap által számított terhelést felülírja az egyéni tanácsadó által jelentett érték.
- false – Alkalmazásakor az egyéni tanácsadó által jelentett terhelési érték hozzáadódik a tanácsadói alapkód által számítotthoz.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

A metódus segítségével az egyéni tanácsadó esetében szükséges inicializálási műveleteket lehet elvégezni. A metódus meghívására a tanácsadó alapmoduljának elindulása után kerül sor.

Sok esetben, ideértve a szabványos tanácsadókat is, ezt a metódust nem használják semmire, és kódja csupán egy *return* utasítást tartalmaz. A metódus alkalmas a suppressBaseOpeningSocket metódus meghívására, amelynek használata csak ebből a metódusból érvényes.

getLoad()

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

iConnectTime
A csatlakozás befejezése ennyi ideig tartott, ezredmásodpercben mérve. A terhelésmérést a tanácsadói alapkód végzi el, majd átadja az értéket az egyéni tanácsadónak, amely a terhelési érték megadásakor figyelembe veheti vagy figyelmen kívül hagyhatja azt. Ha a csatlakozás sikertelen, az érték -1.

caller
A tanácsadói alaposztálynak az alap tanácsadói metódusokat biztosító példánya.

Az egyéni tanácsadók számára elérhető függvényhívások

Az alábbi részekben ismertetett metódusokat és függvényeket az egyéni tanácsadókból lehet meghívni. A metódusokat a tanácsadói alapkód biztosítja.

A függvényhívások egy része közvetlenül is elindítható, például *függvénynév*() formában, egy részüknél azonban alkalmazni kell a hívó előtagot. A *hívó* az éppen futtatott egyéni tanácsadót támogató alap tanácsadópéldányt jelenti.

ADVLOG()

Az ADVLOG függvénnyel az egyéni tanácsadó szöveges üzenetet tud írni a tanácsadói alap naplófájljába. A formátum a következő:

```
void  ADVLOG  (int naplószint, string üzenet)
```

logLevel
Az az állapotszint, amelyen az üzenet naplófájlba való írása történik. A tanácsadói naplófájl több szintre szerveződik, a legfontosabb üzenetek nullás állapotszintet kapnak, míg a kevésbé fontosak magasabb számokat. A legrészletesebb típusú üzenetek az 5-ös állapotszintet kapják. A szintek segítségével szabályozni lehet a felhasználók számára valós időben átadott üzenetek típusát. (A részletességet a **dscontrol** paranccsal lehet beállítani.) A katasztrofális hibákat mindig nullás szinten kell naplózni.

üzenet
A naplófájlba kiírni kívánt üzenet. A paraméter értéke szabványos Java karaktersorozat.

getAdvisorName()

A getAdvisorName függvény egy Java karaktersorozatot ad vissza, amely az egyéni tanácsadó nevének utótag részét tartalmazza. Az ADV_cdload.java nevű tanácsadó esetében például a függvény a cdload értéket adja vissza.

A függvénynek nincs paramétere.

Megjegyezzük, hogy az érték a tanácsadó példányosítása során való megváltoztatására nincs lehetőség.

getAdviseOnPort()

A getAdviseOnPort függvény annak a portnak a számát adja vissza, amelyen a hívó egyéni tanácsadó fut. A visszatérési érték egy Java integer (int), és a függvénynek nincs paramétere.

Megjegyezzük, hogy az érték a tanácsadó példányosítása során való megváltoztatására nincs lehetőség.

caller.getCurrentServer()

A getCurrentServer függvény a jelenlegi szerver IP címét adja vissza. A visszatérési érték egy Java karaktersorozat, IP cím formátummal, például: 128.0.72.139.

A cím jellemzően az egyéni tanácsadó minden meghívásánál más, ugyanis a tanácsadói alapkód sorra lekérdezi az összes szervergépet.

A függvénynek nincs paramétere.

caller.getCurrentCluster()

A getCurrentCluster függvényhívás a jelenlegi szerverfürt IP címét adja vissza. A visszatérési érték egy Java karaktersorozat, IP cím formátummal, például: 128.0.72.139.

A cím jellemzően az egyéni tanácsadó minden meghívásánál más, ugyanis a tanácsadói alapkód sorra lekérdezi az összes szerverfürtöt.

A függvénynek nincs paramétere.

getInterval()

A getInterval függvény a tanácsadási időközt adja vissza, ami a tanácsadói ciklusok közötti másodpercek száma. Az érték egyenlő az egyéni tanácsadó konstruktorában megadott alapértelmezett értékkel, hacsak az értéket a futás során nem módosították a **dscontrol** paranccsal.

A visszatérési érték egy Java integer (int). A függvénynek nincs paramétere.

caller.getLatestLoad()

A getLatestLoad függvény segítségével az egyéni tanácsadó lekérdezheti egy adott szerverobjektum legutolsó terhelési értékét. A terhelési értékek nyilvántartása a tanácsadói alapkód és a kezelődémon belső táblázataiban történik.

```
int caller.getLatestLoad (string fürt_IP, int port, string szerver_IP)
```

A három argumentum együttesen egy kiszolgáló objektumot határoz meg.

fürt_IP
Annak a szerver objektumnak a fürt IP címe, amelynek az aktuális terhelési értékét le kell kérdezni. Az argumentumnak Java karaktersorozatnak kell lennie, IP cím formátumban, például: 245.145.62.81.

port
Annak a szerver objektumnak a portszáma, amelynek az aktuális terhelési értékét le kell kérdezni.

szerver_IP
Annak a szerver objektumnak az IP címe, amelynek az aktuális terhelési értékét le kell kérdezni. Az argumentumnak Java karaktersorozatnak kell lennie, IP cím formátumban, például: 192.255.201.3.

A visszatérési érték integer típusú.

- Ha a visszatérési érték pozitív, akkor a lekérdezett objektumhoz tartozó jelenlegi terhelési értéket adja meg.
- A -1-es érték azt jelzi, hogy az adott szerver leállt.
- A -2-es érték azt jelzi, hogy az adott szerver állapota ismeretlen.

A függvényhívás használatának akkor van értelme, ha valamelyik protokoll vagy port viselkedését egy másik viselkedésétől kell függővé tenni. A függvényhívás alkalmas például arra, hogy egy egyéni tanácsadó letiltson egy meghatározott alkalmazáskiszolgálót, ha az azonos gépen futó Telnet szervert letiltották.

caller.receive()

A receive függvény információkat fogad a socketkapcsolatról.

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

A *válasz* paraméter egy karaktersorozat-puffer, ebbe kerülnek a kapott adatok. A függvény emellett egy integer értéket is átad, amelynek jelentése a következő:

- A 0 érték az adatok sikeres elküldését jelzi.
- A negatív értékek hibát jeleznek.

caller.send()

A send függvény a már létrejött socketkapcsolatot veszi igénybe egy csomagnyi adat a megadott porton keresztül a szerverhez való küldésére.

```
caller.send(string parancs)
```

A *parancs* paraméter egy karaktersorozat, amely a szervernek elküldeni kívánt adatokat tartalmazza. A függvény egy integer értéket ad vissza, amelynek jelentése a következő:

- A 0 érték az adatok sikeres elküldését jelzi.
- A negatív értékek hibát jeleznek.

suppressBaseOpeningSocket()

A suppressBaseOpeningSocket függvényhívással az egyéni tanácsadó megadhatja, hogy az alap tanácsadó kód nyisson-e TCP socketet a szerver felé az egyéni tanácsadó nevében. Ha a tanácsadó a szerver állapotának meghatározásához nem végez közvetlen kommunikációt a szerverrel, akkor a socket megnyitása valószínűleg szükségtelen.

A függvényhívást csak egyszer, és csak az ADV_AdvisorInitialize rutinból lehet elindítani.

A függvénynek nincs paramétere.

Példák

Az alábbi példák az egyéni tanácsadók megvalósítását szemléltetik.

Szabványos tanácsadó

A minta forráskód hasonló a Load Balancer szabványos HTTP-tanácsadójához. Működése a következő:

1. Kiad egy küldési kérést, egy ″HEAD/HTTP″ parancsot.
2. Fogadja a választ. Az információkat nem értelmezi, de a válasz hatására a getLoad metódus futása befejeződik.
3. A getLoad metódus 0 visszatérési értékkel jelzi a sikert, -1 értékkel pedig a hibát.

A tanácsadó normál módban működik, vagyis a terhelésmérés azon alapul, hogy ezredmásodpercben mérve mennyi ideig tart a socket megnyitása, a küldés, a fogadás és a lezárás végrehajtása.

```
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;                                  // általában üres rutin
  }

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

```
  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;       // inicializálás elérhetetlenre

    iRc = caller.send(ADV_SEND_REQUEST);     // a HTTP-kérés küldése
                                             // a szervernek
    if (0 <= iRc) {                          // ha a küldés sikeres
      StringBuffer sbReceiveData = new StringBuffer("");   // puffer foglalása
                                                           // a válasz számára
      iRc = caller.receive(sbReceiveData);   // az eredmény fogadása

      // szükség esetén az eredmény értelmezése

      if (0 <= iRc) {          // ha a fogadás sikeres
        iLoad = 0;             // 0 visszatérési érték jelzi a sikert
      }                        // (normál módban az alap figyelmen kívül
    }                          //  hagyja a tanácsadó terhelési értékét)
    return iLoad;
  }
}
```

Oldaladatfolyam tanácsadó

A minta a tanácsadói alap által megnyitott szabványos socket felülbírálását szemlélteti. A szerver lekérdezéséhez a tanácsadó megnyit helyette egy oldaladatfolyam Java socketet. Az eljárás főként olyan szervereknél alkalmazható, amelyek a normál ügyfélforgalomtól eltérő portot használnak a tanácsadói lekérdezések fogadására.

A példában a szerver a 11999-es porton fogadja a kéréseket, és terhelési értékként a hexadecimális int ″4″ értéket adja vissza. A mintakód helyettesítő módban fut, vagyis a tanácsadó konstruktorának utolsó paramétere true értékre van állítva, és a tanácsadói alapkód az eltelt idő helyett a kapott terhelési értéket használja.

Megjegyezzük, hogy a supressBaseOpeningSocket() meghívása az inicializálási rutinból történik. Ha nincs szükség adatok küldésére, akkor az alapsocket felülbírálása is szükségtelen. A socket megnyitásával például ellenőrizni lehet, hogy a tanácsadó tud-e csatlakozni a szerverhez. A döntés meghozatala előtt gondosan vizsgálja meg saját alkalmazásának igényeit.

```
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;

  // bájttömb létrehozása a terheléskérési üzenettel
  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);         // a helyettesítő mód paramétere true
    super.setAdvisor( this );
  }
```

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

  public void ADV_AdvisorInitialize()
  {
    suppressBaseOpeningSocket();   // az alapkód felszólítása arra, hogy
                                   // ne nyissa meg a szabványos socketet
    return;
  }

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

  public int getLoad(int iConnectTime, ADV_Thread caller) {
    int iRc;
    int iLoad = ADV_HOST_INACCESSIBLE;    // -1
    int iControlPort = 11999;   // ezen a porton folyik a kommunikáció a szerverrel

    string sServer = caller.getCurrentServer();   // a lekérdezni kívánt szerver címe
    try {
      socket soServer = new Socket(sServer, iControlPort);  // socket megnyitása
                                                            // a szerver felé
      DataInputStream disServer = new DataInputStream(
                                     soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(
                                      soServer.getOutputStream());

      int iRecvTimeout = 10000;  // az adatok fogadására vonatkozó időkorlát
                                 // beállítása (mértékegysége: ezredmásodperc)
      soServer.setSoTimeout(iRecvTimeout);

      dosServer.writeInt(4);     // üzenet küldése a szervernek
      dosServer.flush();

      iLoad = disServer.readByte();   // a szerver válaszának fogadása

    } catch (exception e) {
      system.out.println("Kivétel történt: " + e);
    }
    return iLoad;    // a szerver által jelentett terhelés átadása
  }
}
```

Kétportos tanácsadó

Ez az egyéni tanácsadói minta azt a lehetőséget szemlélteti, hogy adott szerver valamely portjának hibáját fel lehet ismerni egyrészt a saját állapota, másrészt egy ugyanazon a szervergépen, másik porton futó másik szerverdémon állapota alapján. Ha például a 80-as porton futó HTTP-démon elnémul, akkor érdemes lehet a 443-as porton futó SSL-démon felé irányuló forgalom útválasztását is leállítani.

A tanácsadó erőszakosabb a szabványos tanácsadóknál, ugyanis minden olyan szervert, amely nem küld választ, meghibásodottnak vesz, illetve *leálltként* jelöl meg. A szabványos tanácsadók a nem válaszoló szervereket rendkívül lelassultakként kezelik. Ez a tanácsadó a szervert a HTTP és az SSL porthoz egyaránt leálltként jelöli meg, ha a szerver a kettő közül akár csak az egyik porton nem válaszol.

Az egyéni tanácsadó használatához a rendszergazdának két példányt kell elindítania belőle: egyet a HTTP, és egyet az SSL porthoz. A tanácsadó két globális kivonattáblát példányosít, egyet a HTTP, egyet pedig az SSL számára. Mindegyik tanácsadó megpróbál kommunikálni a saját szerverdémonjával, majd a művelet eredményeit eltárolja a kivonattáblában. Az egyes tanácsadók által az alap tanácsadó osztálynak átadott értékek egyrészt a saját szerverdémonnal

folytatott kommunikáció sikerességétől, másrészt a partnertanácsadó a saját démonjával folytatott kommunikációjának sikerességétől függ.

Az alkalmazott egyéni metódusok a következők.

- Az ADV_nte() egy egyszerű tárolóobjektum adott szerver adatainak tárolására. Ezek az objektumok a kivonattáblában tárolódnak mint táblaelemek. Minden objektumnak van egy időpecsétje, amely alapján eldönthető, hogy az elem naprakész információkat hordoz-e.
- A putNte() és a getNte() szinkronizált metódusok, amelyek biztosítják a két tanácsadópéldány a kivonattáblához való hozzáférésének ellenőrzöttségét.
- A getLoadHTTP a HTTP-szerver válaszkészségét lekérdező eljárás. Alacsony szintű rutin, az SSL protokollról nem gyűjt vagy használ információkat.
- A getLoadSSL() az SSL-szerver válaszkészségét ellenőrző metódus. Alacsony szintű rutin, a HTTP protokollról nem gyűjt vagy használ információkat.
- A getLoad() az egyéni tanácsadó belépési pont rutinja. Mindkét protokoll kezelésére képes, valamint képes adatokat tárolni és lekérdezni a kivonattáblába, illetve kivonattáblából. Ez az a rutin, amely összeköti a két portot.

A következő hibaállapotok felismerésére van lehetőség.

- Válaszképtelen szervergép — Az alap tanácsadóosztály rendszeres időközönként egy ping jelet küld a szerver címére. Ha a cím nem érhető el, az alap tanácsadó osztály leálltként jelöli meg a szervert. Az egyéni tanácsadó két példánya közül egyik sem hívódik meg, és a rendszer a gépen futó mindkét szervert leálltként jelöli meg.
- A szervergép egyik démonja válaszképtelenné válik, a másik azonban működik — Amikor az alapkód megpróbál megnyitni egy socketet a szerverrel, a csatlakozási kísérlet elutasításra kerül, és az adott protokoll alap tanácsadója leálltként jelöli meg a szervert. Az adott protokoll egyéni tanácsadójának kódja nem hívódik meg. Bár a másik protokoll egyéni tanácsadója folytatja a kommunikációt a saját szerverével, a kivonattáblából értesül arról, hogy a másik egyéni tanácsadó nem tud kommunikálni a hozzá tartozó szerverdémonnal. Ekkor a második protokoll tanácsadója is leálltként jelöli meg a hozzá tartozó szervert.
- Az egyik démon nem válaszol, de a másik igen — A válaszképtelen protokoll egyéni tanácsadója felismeri a kommunikációs hibát, leálltként jelöli meg a szervert, majd beírja az adatokat a kivonattáblába. A másik port egyéni tanácsadója a kivonattábla alapján értesül erről, és a saját szerverét is leálltként jelöli meg.

A példa a HTTP 80-as és az SSL 443-as portját köti össze, de bármely portkombinációra átírható.

```
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;

//--------
// Az egyéni tanácsadóban használt kivonattáblák táblázatelemének meghatározása

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();
  }

//--------
// az elem érvényességének/lejártának ellenőrzése
  public boolean isCurrent(ADV_twop oThis) {
    boolean bCurrent;
    int iLifetimeMs = 3 * 1000 * oThis.getInterval();   // az élettartam
                                                           beállítása
                                                        // 3 tanácsadói
                                                           ciklusra
    Date dNow = new Date();
    Date dExpires = new Date(dTimestamp.getTime() + iLifetimeMs);

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

//--------
// értékhozzáférő(k)

  public int getLoadValue() { return iLoad; }

//--------
// klónozás (a szálak közötti sérülések elkerülésére)

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

}

//--------
// az egyéni tanácsadó megadása

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;

  //--------
  // a portokhoz tartozó előzményinformációkat tároló táblázatok megadása

  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";

 //--------
 // bájttömb létrehozása az SSL-ügyfelek hello üzenetével

 public static final byte[] abClientHello = {
   (byte)0x80, (byte)0x1c,
   (byte)0x01,                // ügyfél hello
   (byte)0x03, (byte)0x00,    // SSL-változat
   (byte)0x00, (byte)0x03,    // rejtjel megadás hossza (byte)
   (byte)0x00, (byte)0x00,    // munkamenet-azonosító hossza (byte)
   (byte)0x00, (byte)0x10,    // kihívási adatok hossza (byte)
   (byte)0x00, (byte)0x00, (byte)0x03,   // rejtjel kiválasztása
   (byte)0x1A, (byte)0xFC, (byte)0xE5, (byte)Ox20,  // kihívási adatok
   (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 = a load balancer méri a válaszidőt
   setAdvisor ( this );
 }

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

 public void ADV_AdvisorInitialize() {
   return;
 }

 //--------
 // szinkronizált PUT és GET hozzáférési rutinok a kivonattáblákhoz

 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 - a HTTP-terhelés meghatározása a szerver válasza alapján

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

   int iRc = caller.send(ADV_HTTP_REQUEST_STRING);  // kérésüzenet küldése
                                                    // a szervernek
   if (0 <= iRc) {           // a kérés sikertelenséggel zárult?
     StringBuffer sbReceiveData = new StringBuffer("")    // puffer foglalása
                                                          // a válasz számára
     iRc = caller.receive(sbReceiveData);    // a szerver válaszának fogadása

     if (0 <= iRc) {             // a fogadás sikertelenséggel zárult?
```

```
      if (0 < sbReceiveData.length()) {      // kapunk adatokat?
        iLoad = SUCCESS;        // a kapott adatok figyelmen kívül hagyása
                                // és sikert jelző kód visszaadása
      }
    }
  }
  return iLoad;
}

//--------
// getLoadSSL() - az SSL alapú terhelés meghatározása a
szerver válasza alapján

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

  int iSocket = caller.getAdvisorSocket();   // hexadecimális kérés
  küldése a szervernek
  CMNByteArrayWrapper cbawClientHello = new CMNByteArrayWrapper(
                                              abClientHello);
  int iRc = SRV_ConfigServer.socketapi.sendBytes(iSocket, cbawClientHello);

  if (0 <= iRc) {           // a kérés sikertelenséggel zárult?
    StringBuffer sbReceiveData = new StringBuffer("");  // puffer foglalása
                                                        // a válasz számára
    iRc = caller.receive(sbReceiveData);     // a szerver válaszának
                                             // fogadása
    if (0 <= iRc) {             // a fogadás sikertelenséggel zárult?
      if (0 < sbReceiveData.length()) {      // kapunk adatokat?
        iLoad = SUCCESS;   // a kapott adatok figyelmen kívül hagyása és a
        sikert jelző kód visszaadása
      }
    }
  }
  return iLoad;
}

//--------
// getLoad - a HTTP és az SSL metódus eredményeinek egyesítése

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

  String sCluster = caller.getCurrentCluster();   // a jelenlegi fürtcím
  int iPort = getAdviseOnPort();
  String sServer = caller.getCurrentServer();
  String sHashKey = sCluster = ":" + sServer;     // a kivonattábla kulcsa

  if (ADV_TWOP_PORT_HTTP == iPort) {              // HTTP-szerver kezelése
    iLoadHTTP = getLoadHTTP(iConnectTime, caller);  // a HTTP alapú terhelés
                                                       lekérdezése

    ADV_nte nteHTTP = newADV_nte(sCluster, iPort, sServer, iLoadHTTP);
    putNte(htTwopHTTP, "HTTP", sHashKey, nteHTTP);  // a HTTP alapú terheléssel
                                                    // kapcsolatos
                                                    információk elmentése
    ADV_nte nteSSL = getNte(htTwopSSL, "SSL", sHashKey);  // az SSL
    protokollal kapcsolatos
                                                          // információk lekérdezése
    if (null != nteSSL) {
      if (true == nteSSL.isCurrent(this)) {         // az időpecsét ellenőrzése
        if (ADV_HOST_INACCESSIBLE != nteSSL.getLoadValue()) {    // működik
                                                                 // az SSL?
          iLoad = iLoadHTTP;
```

```
          } else {    // az SSL nem működik, ezért a HTTP-szervert leálltként
          kell megjelölni
            iLoad= ADV_HOST_INACCESSIBLE;
          }
        } else {      // az SSL protokollal kapcsolatos információk elévültek,
                      // ezért a HTTP-szervert leálltként jelöljük meg
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {        // nincs terhelési információ az SSL protokollról,
                      // a getLoadHTTP() eredményeit kell jelenteni
        iLoad = iLoadHTTP;
      }
    }
    else if (ADV_TWOP_PORT_SSL == iPort) {           // SSL-szerver kezelése
      iLoadSSL = getLoadSSL(iConnectTime, caller);   // az SSL alapú
      terhelés lekérdezése

      ADV_nte nteSSL = new ADV_nte(sCluster, iPort, sServer, iLoadSSL);
      putNte(htTwopSSL, "SSL", sHashKey, nteSSL);   // az SSL alapú
      terheléssel kapcsolatos információk mentése

      ADV_nte nteHTTP = getNte(htTwopHTTP, "SSL", sHashKey);   // a HTTP protokollal
                                                               // kapcsolatos
                                                               információk lekérdezése
      if (null != nteHTTP) {
        if (true == nteHTTP.isCurrent(this)) {       // az időpecsét ellenőrzése
          if (ADV_HOST_INACCESSIBLE != nteHTTP.getLoadValue()) {  // a HTTP
                                                                  // működik?
            iLoad = iLoadSSL;
          } else {   // a HTTP-szerver nem működik, az SSL megjelölése leálltként
            iLoad = ADV_HOST_INACCESSIBLE;
          }
        } else {     // a HTTP protokollal kapcsolatos információk elévültek, ezért az
        SSL protokollt leálltként kell megjelölni
          iLoad = ADV_HOST_INACCESSIBLE;
        }
      } else {       // nincs terhelési információ a HTTP protokollról,
                     // a getLoadSSL() eredményeit kell jelenteni
        iLoad = iLoadSSL;
      }
    }

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

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

WebSphere Application Server tanácsadó

A WebSphere Application Serverhez a *telepítési_útvonal*/servers/samples/CustomAdvisors/ könyvtárban található egy egyéni tanácsadó példa. A jelen dokumentum nem tartalmazza a teljes kódot.

- Az ADV_was.java a tanácsadó forráskódját tartalmazó fájl, melyet a Load Balancer gépen kell lefordítani és futtatni.
- Az LBAdvisor.java.servlet a szerver kisalkalmazás forráskódja, ezt LBAdvisor.java névre kell átnevezni, majd a WebSphere Application Server gépen kell lefordítani és futtatni.

A teljes tanácsadó csak kismértékben bonyolultabb a példánál. Bővítésként rendelkezik egy különleges értelmező rutinnal, amely tömörebb a fenti példában szereplő StringTokenizer eljárásnál.

A példakód összetettebb része a Java szerver kisalkalmazásban található. Az egyéb metódusok mellett a szerver kisalkalmazás két a szerver kisalkalmazások specifikációja által megkövetelt metódust tartalmaz: az init() és a service() metódust; továbbá a run() metódust, amelynek meglétét a Java.lang.thread class igényli.

- Az init() eljárást a szerver kisalkalmazás motor egyszer, az inicializáláskor hívja meg. A metódus létrehoz egy _checker nevű szálat, amely a tanácsadótól érkező hívásoktól függetlenül fut, és adott ideig alvó állapotban marad, mielőtt folytatná a feldolgozási ciklusát.
- A service() eljárást a szerver kisalkalmazás motor a szerver kisalkalmazás minden meghívásakor lefuttatja. Ebben az esetben a metódust a tanácsadó hívja meg. A service() metódus ASCII karakterek folyamát küldi el a kimeneti adatfolyamra.
- A run() a kód végrehajtásának magját tartalmazza. A start() metódus hívja meg, amely meghívására viszont az init() metódusból kerül sor.

A szerver kisalkalmazás kódjának vonatkozó részei az alábbiakban szerepelnek.

```
...

  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;
  }

  ...
```

A tanácsadók által szolgáltatott adatok használata

Amikor szabványos hívást intéz az alkalmazáskiszolgáló valamely meglévő részéhez, illetve új kódrészletet ad hozzá, mint saját egyéni tanácsadójának szerveroldali megfelelőjét, akkor szüksége lehet a kapott terhelési értékek vizsgálatára és a szerver viselkedésének módosítására. A Java StringTokenizer osztállyal és a hozzá tartozó metódusokkal könnyen elvégezheti a vizsgálatot.

Egy jellegzetes HTTP-parancs tartalma a következő: GET /index.html HTTP/1.0

A parancsra adott jellemző válasz például a következő lehet.

```
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: hu

<!DOCTYPE HTML PUBLIC "-//w3c//DTD HTML 3.2 Final//HU">
<HTML><HEAD><TITLE>Tesztoldal</TITLE></HEAD>
<BODY><H1>Apache szerver</H1>
<HR>
<P><P>Ez a webszerver az Apache 1.3.12-es változatát futtatja.
<P><HR>
<P><IMG SRC="apache_pb.gif" ALT="">
</BODY></HTML>
```

A jelen esetben érdekes elemek az első sorban találhatók; ezek közül is a legfontosabb a HTTP visszatérési kód.

A HTTP specifikációja osztályokba sorolja a visszatérési kódokat, ezeket a következőképpen foglalhatjuk össze:

- 2xx - sikert jelző visszatérési kódok
- 3xx - átirányítást jelző visszatérési kódok
- 4xx - ügyfélhibát jelző visszatérési kódok
- 5xx - szerverhibát jelző visszatérési kódok

Ha pontosan tudja, hogy a szerver várhatóan milyen kódokat ad vissza, akkor nincs szüksége olyan részletes kódra, mint amilyen a példában szerepel. Ne feledje azonban, hogy a felismert visszatérési kódok körének korlátozásával csökkentheti a program későbbi rugalmasságát.

Az alábbi példában egy önálló, Java alapú program szerepel, amely magában foglal egy alapszintű HTTP-ügyfelet. A példában egy egyszerű, általános célú értelmezőt hívunk meg a HTTP-válaszok vizsgálatára.

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

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

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

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

  StringTokenizer st = new StringTokenizer(sResponse, " ");
  if (true == st.hasMoreTokens()) {
    sHTTPVersion = st.nextToken();
    if (sHTTPVersion.equals(sHTTP110) || sHTTPVersion.equals(sHTTP11)) {
      System.out.println("HTTP-változat: " + sHTTPVersion);
    } else {
      System.out.println("Érvénytelen HTTP-változat: " + sHTTPVersion);
    }
  } else {
    System.out.println("Nincs válasz");
    return;
  }

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

    switch (iRc) {
    case(200):
      System.out.println("HTTP válaszkód: OK, " + iRc);
      break;
    case(400): case(401): case(402): case(403): case(404):
      System.out.println("HTTP válaszkód: ügyfélhiba, " + iRc);
      break;
    case(500): case(501): case(502): case(503):
      System.out.println("HTTP válaszkód: szerverhiba, " + iRc);
      break;
    default:
      System.out.println("HTTP válaszkód: ismeretlen, " + iRc);
      break;
    }
  }

  if (true == st.hasMoreTokens()) {
    while (true == st.hasMoreTokens()) {
      sbText.append(st.nextToken());
      sbText.append("  ");
      }
```

```
      System.out.println("HTTP válaszmondat: " + sbText.toString());
    }
  }
}
```

Megjegyzések

Első kiadás (2006. Május)

A jelen információkat az Amerikai Egyesült Államokban elérhető termékekhez és szolgáltatásokhoz dolgoztuk ki.

A jelen dokumentumban tárgyalt termékek, szolgáltatások vagy összetevők elérhetőségét az IBM más országokban nem feltétlenül biztosítja. A tartózkodási helyén jelenleg elérhető termékekkel és szolgáltatásokkal kapcsolatban az IBM helyi képviselőjétől kérhet tájékoztatást. Bármely IBM termékre, programra vagy szolgáltatásra tett hivatkozás semmilyen formában nem jelenti azt, hogy kizárólag az adott IBM termék, program vagy szolgáltatás vehető igénybe. Bármely funkcionalitását tekintve egyenértékű termék, program vagy szolgáltatás, amely nem sérti az IBM szellemi tulajdonát, alkalmas a helyettesítésre. A felhasználó felelőssége ugyanakkor, hogy értékelje és ellenőrizze a nem az IBM által szállított termékek, programok és szolgáltatások működését.

Az IBM szabadalmakkal vagy bejegyzés alatt lévő szabadalmakkal rendelkezhet a jelen dokumentum témáját illetően. A jelen dokumentum kiadása semmilyen használati engedélyt nem jelent az Ön számára ezekhez a szabadalmakhoz. Ha használati engedélyt kíván kérni, írjon a következő címre:

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

A duplabájtos (DBCS) információkkal kapcsolatos használati engedélyekkel kapcsolatban forduljon a saját országában található IBM iroda szellemi tulajdonokkal foglalkozó részlegéhez, illetve a következő címre küldje el kérését:

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

Az alábbi bekezdés nem vonatkozik az Egyesült Királyságra és azokra a további országokra, amelyekben a hasonló óvintézkedések összeférhetetlenek a helyi jogszabályokkal:

AZ INTERNATIONAL BUSINESS MACHINES CORPORATION A JELEN DOKUMENTUMOT “A JELENLEGI FORMÁJÁBAN” BOCSÁTJA RENDELKEZÉSRE, BÁRMILYEN HALLGATÓLAGOS VAGY KIFEJEZETT GARANCIA NÉLKÜL, IDEÉRTVE TÖBBEK KÖZÖTT, DE NEM KIZÁRÓLAGOSAN A SZERZŐI JOGOK MEG NEM SÉRTÉSÉRE, AZ ELADHATÓSÁGRA VAGY AZ ADOTT CÉLRA VALÓ ALKALMASSÁGRA VONATKOZÓ BURKOLT GARANCIAVÁLLALÁST VAGY KIKÖTÉSEKET. Bizonyos államok bizonyos tranzakciók esetében nem engedélyezik a kifejezett vagy burkolt garanciára vonatkozó jogkizárási nyilatkozatokat, ezért lehetséges, hogy ez a nyilatkozat az Ön esetére nem vonatkozik.

A jelen információk technikai pontatlanságokat vagy szedési hibákat tartalmazhatnak. Az itt szereplő információk rendszeres időközönként végrehajtott módosítások tárgyát képezik; ezek a módosítások a dokumentum újabb kiadásaiba kerülnek be. Az IBM a jelen kiadványban

ismertetett termékben/termékekben és/vagy programban/programokban bármikor, bármilyen értesítés nélkül jogosult fejlesztéseket és/vagy módosításokat végrehajtani.

A jelen információkban a nem az IBM által üzemeltetett webhelyekre utaló hivatkozások kizárólag kényelmi szempontok miatt szerepelnek, és semmilyen beleegyezést nem jelentenek a webhelyekre vonatkozóan. Az ezeken a webhelyeken szereplő anyagok nem képezik az IBM termékéhez tartozó anyagok részét; a webhelyek szolgáltatásait kizárólag a saját felelősségére vegye igénybe.

Az IBM az Ön által megadott információkat jogosult tetszőleges általa megfelelőnek tartott módon felhasználni vagy terjeszteni, anélkül, hogy Ön felé bármilyen felelősségvállalást tenne.

A program azon licencelői, akik további információkat szeretnének kapni ahhoz, hogy alkalmassá tegyék a következő célokra: (i) információcsere független készítésű programok és egyéb programok között (ideértve ezt a programot is) és (ii) a kicserélt információk kölcsönös felhasználása; a következő címmel lépjenek kapcsolatba:

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

Az ilyen jellegű információk a megfelelő feltételek és kikötések mellett érhetők el, ami bizonyos esetekben díjfizetést is magába foglalhat.

A jelen dokumentumban ismertetett, licencelt programot és a hozzá tartozóan elérhető, licencelt anyagokat az IBM az IBM nemzetközi program-licencszerződésének hatálya vagy bármely közöttünk létrejött egyéni megállapodás hatálya alatt teszi elérhetővé.

Az itt szereplő bármely teljesítményadatokat ellenőrzött környezetben állapították meg. Az egyéb működési környezetekben kapott eredmények emiatt jelentősen eltérhetnek. A mérések egy részét fejlesztői szintű rendszereken végezték, ezért semmilyen garancia nincs arra nézve, hogy ezek a mérések az általánosan elérhető rendszereken is azonosak lesznek. Emellett bizonyos mérési eredményeket extrapolálással nyertek. A tényleges eredmények eltérők lehetnek. A jelen dokumentum használóinak ellenőrizniük kell a saját környezetükre vonatkozó adatokat.

A nem IBM termékekre vonatkozó információkat az egyes termékek szállítói adták, nyilvános bejelentéseikből, illetve egyéb nyilvános forrásokból származnak. Az IBM nem tesztelte ezeket a termékeket, ezért nem tudja megerősíteni a nem IBM termékek teljesítményére, kompatibilitására és egyéb jellemzőire vonatkozó állításokat. A nem IBM termékek képességeire vonatkozó kérdéseit a termékek szállítóinak tegye fel.

Az IBM a jövőbeli irányvonalaira vagy szándékaira vonatkozó bármely kijelentését értesítés nélkül megváltoztathatja vagy visszavonhatja, és ezek csak célokat és célkitűzéseket jelentenek.

A jelen információk a mindennapi üzleti üzemeltetésnél alkalmazott adatokra és jelentésekre vonatkozóan tartalmaznak példákat. A lehető legteljesebb szemléltetés érdekében a példák egyének, vállalatok, márkák és termékek nevét tartalmazhatják. Mindezek a nevek kitaláltak, bármely hasonlóságuk a tényleges vállalatok által használt nevekkel és címekkel kizárólag a véletlen műve.

Ha az információkat elektronikus formában tekinti meg, akkor lehetséges, hogy a fényképek és a színes illusztrációk nem jelennek meg.

Védjegyek

A következő kifejezések az IBM Corporation védjegyei az Egyesült Államokban és/vagy más országokban:

- AIX
- IBM
- ViaVoice
- WebSphere

A Java és az összes Java alapú védjegy a Sun Microsystems, Inc. védjegye az Egyesült Államokban és/vagy más országokban.

A Microsoft, a Windows, a Windows NT és a Windows logó a Microsoft Corporation védjegye az Egyesült Államokban és/vagy más országokban.

Az Intel, az Intel Inside (logók), az MMX és a Pentium az Intel Corporation védjegye az Egyesült Államokban és/vagy más országokban.

A UNIX a The Open Group bejegyzett védjegye az Egyesült Államokban és más országokban.

A Linux Linus Torvalds védjegye az Egyesült Államokban és/vagy más országokban.

Az egyéb vállalat-, termék- és szolgáltatásnevek egyéb vállalatok védjegyei vagy szolgáltatás védjegyei lehetnek.

Tárgymutató

N

O, Ó

P

R

S

SZ

T

U, Ú

V

W

Nyomtatva Dániában

GC22-0411-00

Spine information:

WebSphere Application Server Programozási útmutató az Edge összetevőkhöz *6.1 Változat* GC22-0411-00