WebSphere Application Server

Edge Components용 프로그래밍 안내서

버전 6.1

GA30-2915-00

WebSphere Application Server

Edge Components용 프로그래밍 안내서

버전 6.1

GA30-2915-00

주!

이 정보와 이 정보가 지원하는 제품을 사용하기 전에, 67 페이지의 『주의사항』의 일반 정보를 반드시 읽으십시오.

초판(2006년 5월)

이 책은 다음에 적용됩니다.
WebSphere Application Server, 버전 6.1

새 개정판에 달리 명시되지 않는 한 후속 릴리스 및 수정판에 적용됩니다.

IBM 담당자 또는 해당 지역의 IBM 지사를 통해 책을 주문할 수 있습니다.

목차

그림

이 책에 대한 정보

이 절은 *WebSphere® Application Server Edge Components용 프로그래밍 안내서*의 목적, 구성 및 규약에 대해 설명합니다.

이 책의 사용자

이 책에서는 WebSphere Application Server, 버전 6.1 Edge Components를 사용자 정의하는 데 사용 가능한 API(Application Programming Interface)에 대해 설명합니다. 이 정보는 플러그인 응용프로그램을 작성하고 다른 사용자 정의를 작성하는 프로그래머를 위한 것입니다. 네트워크 설계자 및 시스템 관리자에게도 가능한 사용자 정의 유형을 나타내는 데 이러한 정보가 도움이 될 수 있습니다.

사전 지식

이 책의 정보를 사용하려면, 사용할 API에 따라 Java™ 또는 C를 사용한 프로그래밍 프로시저를 이해하고 있어야 합니다. 인터페이스에서 사용 가능한 메소드 및 구조가 문서화되어 있지만, 사용자는 응용프로그램 구성 방법, 시스템에 맞는 컴파일 방법 및 테스트 방법을 알고 있어야 합니다. 일부 인터페이스에 대해 코드 샘플이 제공되지만, 샘플은 사용자의 응용프로그램을 구성하기 위한 예제로만 제공됩니다.

이 책에 사용된 규칙 및 용어

이 문서에서는 다음과 같은 서체와 키 규약을 사용합니다.

표 1. 이 책에 사용된 규약

규약	의미
굵은체	GUI와 관련될 때, 굵은체로 메뉴, 메뉴 항목, 레이블, 단추, 아이콘 및 폴더를 나타냅니다. 또한 주위의 텍스트와 혼동될 수 있는 명령 이름을 강조하는 데에도 사용될 수 있습니다.
모노스페이스	명령 프롬프트에 입력해야 할 텍스트를 모노스페이스로 표시합니다. 화면 텍스트, 코드 예제 및 파일 발췌 부분도 나타냅니다.
기울임꼴	제공해야 하는 변수값을 기울임꼴로 나타냅니다. (예를 들어, *fileName*에 파일 이름을 제공합니다.) 강조 및 책 제목을 표시합니다.
Ctrl-*x*	여기서, *x*는 키 이름으로 제어 문자 순서를 표시합니다. 예를 들어, Ctrl-c는 Ctrl 키를 누른 상태에서 c 키를 누르는 것을 의미합니다.
Return	Return, Enter 또는 왼쪽 화살표로 표시된 키를 나타냅니다.
%	루트 권한이 필요하지 않은 명령의 Linux 및 UNIX® 명령 쉘 프롬프트를 표시합니다.
#	루트 권한이 필요한 Linux 및 UNIX 명령 쉘 프롬프트를 표시합니다.
C:\	Windows 명령 프롬프트를 표시합니다.
명령 입력	명령을 "입력" 또는 "실행"할 때 명령을 입력하고 Return을 누릅니다. 예를 들어, "Enter the **ls** command"라는 명령은 명령 프롬프트에 **ls**를 입력하고 Return을 누르는 것을 의미합니다.

표 1. 이 책에 사용된 규약 (계속)

규약	의미
[ ]	구문 설명에 선택적 항목을 넣습니다.
{ }	선택할 항목이 있는 목록을 구문 설명에 넣습니다.
|	구문 설명에서 { }에 있는 선택사항 목록의 항목을 구분합니다.
...	구문 설명에서 줄임표는 앞의 항목을 한 번 이상 반복할 수 있다는 것을 의미합니다. 예제에서 줄임표는 간결하게 하기 위해 예제에서 정보를 생략했음을 의미합니다.

내게 필요한 옵션

내게 필요한 옵션 기능은 지체 부자유 또는 시각 장애와 같은 신체 장애를 가진 사용자가 소프트웨어 제품을 잘 사용할 수 있도록 도와줍니다. 다음은 WebSphere Application Server, 버전 6.1의 내게 필요한 옵션 기능입니다.

- 화면 판독기 소프트웨어와 디지털 음성 신디사이저를 사용하면 화면에 표시된 내용을 들을 수 있습니다. IBM® ViaVoice™와 같은 음성 인식 소프트웨어를 사용하여 데이터를 입력하고 사용자 인터페이스를 탐색할 수 있습니다.
- 모든 기능은 마우스 대신 키보드를 사용하여 작동할 수 있습니다.
- 제공된 그래픽 인터페이스 대신 표준 문서 편집기나 명령행 인터페이스를 사용하여 Application Server 기능을 구성하고 관리할 수 있습니다. 특정 기능의 액세스에 필요한 옵션에 대한 자세한 정보는 해당 기능 문서를 참조하십시오.

관련 문서 및 웹 사이트

- *Edge Components용 개념, 계획 및 설치*, GA30-2919-00
- *Caching Proxy 관리 안내서*, GA30-2916-00
- *Load Balancer 관리 안내서*, GA30-2917-00
- IBM 홈 웹 사이트, *www.ibm.com/*
- IBM WebSphere Application Server:
  www.ibm.com/software/webservers/appserv/
- IBM WebSphere Application Server 라이브러리 웹 사이트:
  www.ibm.com/software/webservers/appserv/library.html
- IBM WebSphere Application Server 지원 웹 사이트:
  www.ibm.com/software/webservers/appserv/support.html
- IBM WebSphere Application Server Information Center:
  www.ibm.com/software/webservers/appserv/infocenter.html
- IBM WebSphere Application Server Edge Components Information Center:
  www.ibm.com/software/webservers/appserv/ecinfocenter.html

고객 의견을 보내는 방법

고객의 피드백은 가장 정확하고 최상의 정보를 제공하는 데 매우 중요합니다. 이 책 또는 기타 WebSphere Application Server Edge Components 문서에 대한 의견이 있는 경우, 다음과 같은 방법으로 보내주십시오.

- 이 책의 뒤에 있는 IBM 한글 지원에 관한 설문 양식을 작성하여 보내주십시오. 책 이름, 부품 번호, WebSphere Application Server 버전 그리고 가능하면 의견 관련 특정 텍스트 위치(예: 페이지 번호 또는 테이블 번호)를 명시하여 주십시오.

제 1 장 Edge Components 사용자 정의 개요

제 1 장에서는 WebSphere Application Server Edge Components에 제공된 API(Application Programming Interface)에 대하여 설명합니다. (WebSphere Application Server Edge Components는 Caching Proxy 및 Load Balancer를 포함합니다.) 관리자는 제공되는 몇 가지 인터페이스를 사용하여 설치를 사용자 정의하고 Edge Components 사이의 상호작용 방법을 변경하거나 소프트웨어 시스템과의 상호작용을 가능하게 할 수 있습니다.

중요사항: Caching Proxy는 다음 경우를 제외하고 모든 Edge Components 설치에 사용할 수 있습니다.

- Caching Proxy는 Itanium 2 또는 AMD Opteron 64비트 프로세서에서 실행되는 Edge Components 설치에 사용할 수 없습니다.
- Caching Proxy는 IPv4용 및 IPv6용 Load Balancer의 Edge Components 설치에 사용할 수 없습니다.

이 문서의 API는 몇 가지 카테고리를 지정합니다.

Caching Proxy 사용자 정의

Caching Proxy는 표준 처리에 대해 사용자 정의 처리가 추가 또는 대체되며, 해당 처리 순서로 작성되는 몇 가지 인터페이스를 가지고 있습니다. 실행할 수 있는 사용자 정의에는 다음과 같은 변경 및 기능 보강 타스크가 포함됩니다.

- 클라이언트 인증
- 요청 권한 부여
- URL을 물리적 파일 경로로 변환
- 요청 서비스
- 로그
- 오류 조건에 응답

Caching Proxy 플러그인으로 알려진 사용자 정의 응용프로그램은 프록시 서버의 처리 순서의 사전 결정된 지점에서 호출됩니다.

Caching Proxy API는 특정 시스템 기능을 구현하는 데 사용됩니다. 예를 들어, 프록시 서버의 LDAP 지원은 플러그인으로 구현됩니다.

3 페이지의 제 2 장 『Caching Proxy API』에서는 인터페이스를 자세히 설명하며, 플러그인 프로그램을 사용하기 위해 프록시 서버를 구성하는 단계가 포함되어 있습니다.

Load Balancer 사용자 정의

고유 어드바이저를 작성하여 Load Balancer를 사용자 정의할 수 있습니다. 어드바이저는 서버에서의 실제 로드를 측정합니다. 사용자 정의 어드바이저를 사용하여 사용자가 제공하고 사용자 시스템의 로드를 측정하는 방법을 사용할 수 있습니다. 이는 사용자가 사용자 정의했거나 독점 웹 서버 시스템을 보유하는 경우에 특히 중요합니다.

45 페이지의 제 3 장 『사용자 정의 어드바이저』에서는 사용자 정의 어드바이저 작성 및 사용에 대한 자세한 정보를 제공합니다. 어드바이저 코드 샘플이 포함되어 있습니다.

코드 샘플 찾기

API의 샘플 코드는 Edge Components CD-ROM의 samples 디렉토리에 있습니다. 추가 코드 샘플은 WebSphere Application Server 웹 사이트인 www.ibm.com/software/webservers/appserv/에서 찾아볼 수 있습니다.

제 2 장 Caching Proxy API

제 2 장에서는 Caching Proxy API(Application Programming Interface)의 개념, 유용성 및 작동 방법에 대해 설명합니다.

중요사항: Caching Proxy는 다음 경우를 제외하고 모든 Edge Components 설치에 사용할 수 있습니다.

- Caching Proxy는 Itanium 2 또는 AMD Opteron 64비트 프로세서에서 실행되는 Edge Components 설치에 사용할 수 없습니다.
- Caching Proxy는 IPv4용 및 IPv6용 Load Balancer의 Edge Components 설치에 사용할 수 없습니다.

Caching Proxy API 개요

API는 Caching Proxy에 대한 인터페이스로 프록시 서버의 기본 기능을 확장합니다. 다음 예제와 같은 사용자 정의 처리를 수행하기 위해 확장 또는 플러그인을 작성할 수 있습니다.

- 기본 인증 루틴을 개선하거나 이를 사이트 특정 프로세스로 변경
- 문제점을 추적하거나 심각한 상황에 대한 경고를 위해 오류 처리 루틴 추가
- 요청 클라이언트에서 보내는 서버 조회 및 사용자 에이전트 코드 정보의 감지 및 추적

Caching Proxy API는 다음과 같은 이점을 제공합니다.

- **효율성**
  - API는 Caching Proxy를 통한 스레드 처리를 위하여 특별히 고안되었습니다.
- **융통성**
  - API에는 다양하고 많은 기능이 있습니다.
  - API는 플랫폼에 대해 독립적이며 언어에 중립적입니다. 모든 종류의 Caching Proxy 플랫폼에서 실행되고 이들 플랫폼에서 지원하는 프로그래밍 언어로 플러그인 응용프로그램을 작성할 수 있습니다.
- **용이성**
  - 값이 아니라 참조에 의해서 단순한 데이터 유형을 처리합니다. (예: long *, char *)
  - 각 기능에 대해 일정한 수의 매개변수가 있습니다.
  - C 언어 바인딩이 포함됩니다.

– 플러그인은 할당된 메모리에 영향을 주지 않습니다. 플러그인 응용프로그램은 다른 Caching Proxy 프로세스와 독립적으로 메모리를 할당하고 해제합니다.

API 프로그램 작성을 위한 일반적인 절차

Caching Proxy 플러그인 프로그램을 작성하기에 앞서 프록시 서버 작동 방법을 이해해야 합니다. 프록시 서버의 작동은 여러 개의 개별 처리 단계로 나눌 수 있습니다. 각각의 단계에서 API를 사용하여 사용자 정의 함수를 제공할 수 있습니다. 예를 들어, 클라이언트 요청이 읽혀진 후 그러나, 다른 처리를 수행하기 전에 어떤 일을 수행하기 원할 수도 있고, 또는 인증이 수행되는 동안 그리고, 요청된 파일이 전송된 후에 다시 특별한 루틴을 수행하기를 원할 수도 있습니다.

사전 정의 함수의 라이브러리는 API와 함께 제공됩니다. 사용자 플러그인 프로그램은 프록시 서버 프로세스(예: 요청을 조정하고, 요청 헤더를 읽거나 쓰고, 프록시 서버의 로그에 쓰는 프로세스)와 상호작용하기 위해 사전 정의된 API 함수를 호출할 수 있습니다. 이런 함수는 프록시 서버가 호출하는 사용자 작성 플러그인 함수와 혼동이 되어서는 안됩니다. 사전 정의된 함수는 18 페이지의 『사전 정의 함수 및 매크로』에서 설명합니다.

서버 구성 파일에서 해당 Caching Proxy API 지시문을 사용하여 적절한 단계에서 플러그인 기능을 호출하도록 프록시 서버에 지시할 수 있습니다. 이 지시문은 25 페이지의 『API 단계에 대한 Caching Proxy 구성 지시문』에서 설명합니다.

이 문서는 다음과 같은 내용을 포함합니다.

- 사용자 정의될 수 있는 Caching Proxy 단계에 대한 기본 설명(『서버 프로세스 단계』 참조)
- 플러그인 작성에 대한 지침(8 페이지의 『지침』 참조)
- 각 단계에서 사용자가 작성할 수 있는 사용자 정의 함수에 대한 프로토타입 및 해당 리턴 코드(10 페이지의 『플러그인 함수 프로토타입』 참조)
- 플러그인에서 호출할 수 있는 사전 정의 함수와 매크로의 정의 및 리턴 코드(18 페이지의 『사전 정의 함수 및 매크로』 참조)
- Caching Proxy API 구성 지시문(25 페이지의 『API 단계에 대한 Caching Proxy 구성 지시문』 참조)

이들 컴포넌트와 프로시저를 사용하여 고유 Caching Proxy 플러그인 프로그램을 작성할 수 있습니다.

서버 프로세스 단계

프록시 서버의 기본 운영은 단계가 진행되는 동안 서버를 수행하는 처리의 유형에 따라 단계를 구분할 수 있습니다. 각 단계는 사용자 프로그램의 지정된 파트가 실행할 수 있는 접속을 포함합니다. Caching Proxy 구성 파일(ibmproxy.conf)에 API 지시문을 추

가하여 특정 단계에서 호출하려는 플러그인 함수를 지정합니다. 해당 단계에 둘 이상의 지시문을 포함시켜 특정 프로세스 단계가 진행되는 동안 여러 개의 플러그인 함수를 호출할 수 있습니다.

몇몇 단계는 서버 요청 프로세스의 일부입니다. 즉, 프록시 서버는 요청을 처리할 때 각 단계를 실행합니다. 다른 단계는 요청 처리와 무관하게 수행됩니다. 즉, 서버는 요청이 처리되는지 여부와 관계없이 이들 단계를 실행합니다.

사용자의 컴파일된 프로그램은 운영 체제에 따라 공유 오브젝트(예: DLL 또는 .so 파일)에 상주합니다. 서버가 요청 프로세스 단계를 차례로 수행하면서, 함수 중 어느 하나가 요청을 처리하였음을 나타낼 때까지 각 단계와 관련된 플러그인 함수를 호출합니다. 특정 단계에 대해 둘 이상의 플러그인 함수가 지정된 경우, 지시문이 구성 파일에 나타나는 순서대로 함수가 호출됩니다.

플러그인 함수가 요청을 처리하지 않은 경우(해당 단계에 Caching Proxy API 지시문이 없거나 해당 단계에 플러그인 함수가 HTTP_NOACTION을 리턴한 경우), 서버는 해당 단계에 대해 기본 조치를 수행합니다.

주: 서비스 단계를 제외한 모든 단계에 적용됩니다. 서비스 단계에는 기본 조치가 없습니다.

6 페이지의 그림 1에서는 프록시 서버 프로세스의 단계를 설명하고 요청 처리와 관련된 단계에 대해 처리 순서를 정의합니다.

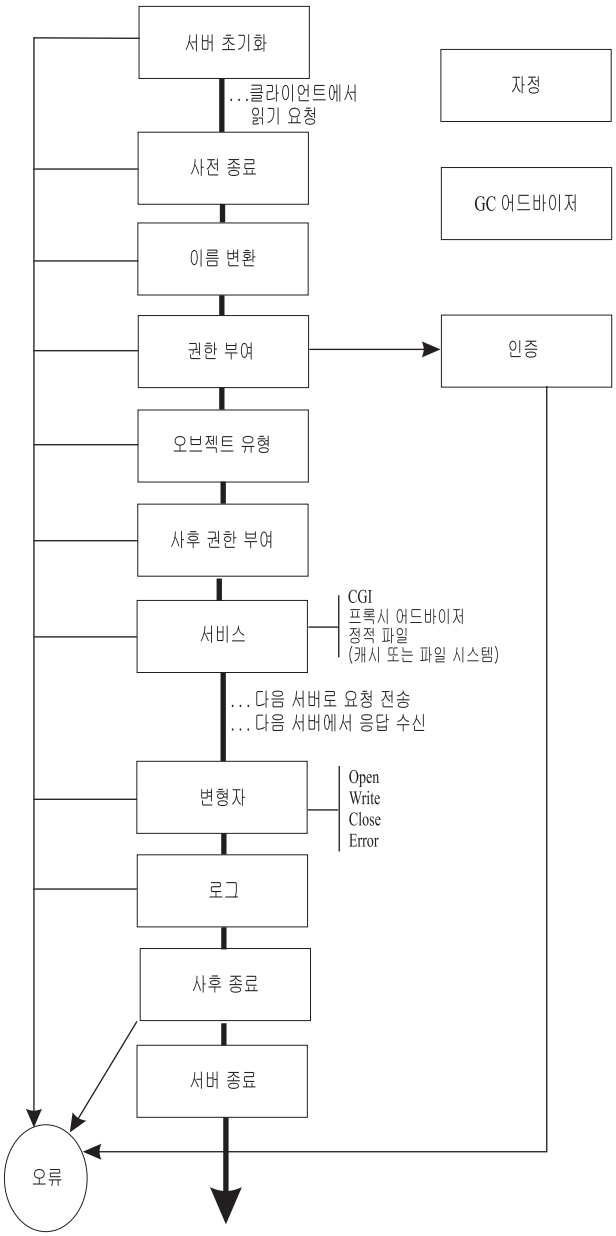

그림 1. 프록시 서버 프로세스 단계 플로우차트

다이어그램에서 네 단계는 클라이언트 요청의 처리와 무관하게 실행됩니다. 이들 단계는 프록시 서버의 실행 및 유지보수와 관련이 있습니다. 다음 사항이 포함됩니다.

- 서버 초기화
- 자정
- GC 어드바이저
- 서버 종료

다음 목록에서는 그림 1에 그려진 각 단계의 목적을 설명합니다. 모든 단계가 특정 요청에 대해 반드시 호출되지는 않습니다.

서버 초기화

프록시 서버가 시작되고 클라이언트 요청이 승인되기 전에 초기화가 수행됩니다.

자정 요청 컨텍스트 없이 자정에 플러그인을 실행합니다. 이 단계는 요청 프로세스의 일부가 아니므로 다이어그램에 개별적으로 표시됩니다. 즉, 단계의 실행은 요청과 무관합니다.

GC 어드바이저

캐시에서 파일에 대한 가비지 콜렉션 결정에 영향을 줍니다. 이 단계는 요청 프로세스의 일부가 아니므로 다이어그램에 개별적으로 표시됩니다. 즉, 단계의 실행은 요청과 무관합니다. 가비지 콜렉션은 캐시 *크기*가 최대 값에 도달할 때 수행됩니다. (캐시 가비지 콜렉션 구성에 대한 정보는 *WebSphere Application Server Caching Proxy 관리 안내서*에 들어 있습니다.)

사전종료

요청이 읽혀졌지만 다른 어떠한 작업도 수행되기 전의 처리를 수행합니다.

이 단계에서 요청 처리 완료 표시(HTTP_OK)를 리턴하는 경우, 요청 프로세스에서 다른 단계들은 생략하고 변형자, 로그 및 사후종료 단계만을 수행합니다.

이름 변환

(URL로부터의) 가상 경로를 물리적 경로로 변환합니다.

권한 부여

저장된 보안 토큰을 사용하여 보호사항, ACL 및 기타 액세스 제어에 대한 실제 경로를 확인하고 기본 인증에 필요한 WWW-Authenticate 헤더를 생성합니다. 사용자가 고유 플러그인 함수를 작성하여 이 단계를 바꾸려면 이 헤더를 사용자가 직접 생성해야 합니다.

자세한 정보는 40 페이지의 『인증 및 권한 부여』를 참조하십시오.

인증

보안 토큰을 해독하고, 확인한 후 저장합니다.

자세한 정보는 40 페이지의 『인증 및 권한 부여』를 참조하십시오.

오브젝트 유형

경로에 표시된 파일 시스템 오브젝트의 위치를 찾습니다.

사후 권한 부여

권한 부여 및 오브젝트의 위치 찾기 이후, 요청 완료 이전에 처리를 수행합니다.

이 단계에서 요청 처리 완료 표시(HTTP_OK)를 리턴하는 경우, 요청 프로세스에서 다른 단계들은 생략하고 변형자, 로그 및 사후종료 단계만을 수행합니다.

서비스 요청을 충족시킵니다(파일 전송, CGI 실행 등).

프록시 어드바이저
프록시와 캐시 결정에 영향을 줍니다.

변형자 클라이언트에 응답을 보내는 데이터 부분에 대한 쓰기 액세스 권한을 부여합니다.

로그 사용자 정의된 트랜잭션 로깅을 사용 가능하게 합니다.

오류 오류 상황에 대해 사용자 정의 응답을 가능하게 합니다.

사후종료
요청 처리에 할당된 자원을 정리합니다.

서버 종료
순서에 따라 시스템 종료 시 정리 처리합니다.

지침

- 서버 플러그인 함수의 구문과 지침에 따라 프로그램을 작성하십시오. 플러그인 함수 각각에 고유한 함수 이름을 부여하고, 필요에 따라 서버의 사전 정의된 함수를 호출하십시오.

  AIX® 시스템에서는 플러그인 함수를 나열하는 내보내기 파일(예: libmyapp.exp)이 필요하고 Caching Proxy API 가져오기 파일(예: libhttpdapi.exp)과 연결해야 합니다.

  Linux, HP-UX 및 Solaris 시스템에서는 libhttpdapi 및 libc 라이브러리와 연결해야 합니다.

  Windows® 시스템에서는 모듈 정의 파일(.def)이 필요하고 HTTPDAPI.LIB와 연결해야 합니다.

  반드시 함수 정의 HTAPI.h를 포함시키고 HTTPD_LINKAGE 매크로를 사용하십시오. 이 매크로는 모든 함수들이 같은 호출 규약을 사용하도록 합니다.
- 서버는 다중 스레드 환경에서 실행되므로 플러그인은 스레드에 안전합니다. 응용프로그램이 재진입 가능한 경우라도 성능이 저하되지 않습니다.
- 플러그인 내의 조치가 스레드 범위를 벗어나지 않도록 하십시오. 종료, 사용자 ID 변경 및 신호 처리기 등록과 같은 프로세스 범위에서 조치를 수행하지 마십시오.
- 글로벌 변수들을 사용하지 않거나, 상호 배타적인 세마포어를 사용하여 글로벌 변수를 보호하십시오.

- 데이터를 다시 클라이언트로 전송하기 위해서 HTTPD_write() 함수를 사용하는 경우 Contents-Type 헤더를 설정하십시오.
- 항상 리턴 코드를 확인하고, 필요한 경우 조건부 처리를 수행하십시오.
- 운영 체제에 맞게 사용자의 컴파일러 관련 책자를 참조하여 프로그램을 컴파일 및 연결하여 공유 오브젝트(예: DLL 또는 .so 파일)를 작성하십시오.

  다음의 컴파일 및 연결 명령을 참조하십시오.

  – IBM CSet++를 사용하는 **AIX**

    - **컴파일:**

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

    - **연결:**

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

      (이 명령은 읽기 쉽도록 두 행에 표시됩니다.)

  – HP C/ANSI C 개발자 번들 및 HP aC++ 컴파일러를 사용하는 **HP-UX**

    - **컴파일:**

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

    - **연결:**

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

  – Gnu Compiler C(GCC) 버전 3.2.X를 사용하는 **Linux**

    - **컴파일:**

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

    - **연결:**

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

  – Sun Workshop을 사용하는 **Solaris**

    - **컴파일:**

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

    - **연결:**

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

  – Microsoft® Visual C++를 사용하는 **Windows**

    - **컴파일:**

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

    - **연결:**

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

  내보내기를 지정하려면 다음 방법 중 하나를 사용하십시오.

  - 소스에 _declspec(dllexport) 정의를 추가하십시오.
  - LIB 명령행에 /EXPORT:entryname을 지정하십시오.
  - EXPORTS 명령문을 사용하여 모듈 정의 파일을 작성하십시오.
- Caching Proxy API 지시문을 구성 파일에 추가하여 사용자 프로그램 플러그인 함수를 해당되는 단계와 연관시키십시오. 각 서버 요청 처리 단계에는 별도의 지시문이 있습니다. 새로운 지시문이 효력을 발생하도록 하려면 서버를 정지했다가 다시 시작하십시오.

  주: Caching Proxy는 재시작하더라도 공유 오브젝트(DDL) 로드를 해제하지 않습니다. 공유 오브젝트를 해제하려면 서버를 정지한 후 시작하십시오.
- 현업 환경에서 프로그램을 사용하기 전에 엄격하게 검사하십시오. Caching Proxy가 스레드 서버이므로 분기 서버에 비해 더 정밀한 검사를 필요로 합니다. 프록시 서버가 사용자 프로그램을 직접 호출하고 그 프로그램들이 같은 프로세스 공간에서 실행되기 때문에 프로그램 내의 오류로 프록시 서버가 중단될 수도 있습니다.

플러그인 함수

『플러그인 함수 프로토타입』에 표시된 구문에 따라 정의된 요청 처리 단계에 적절한 사용자의 프로그램 함수를 작성할 수 있습니다.

각각의 사용자 함수들은 어떤 조치가 취해졌는지를 나타내는 값을 리턴 코드 매개변수에 지정해야 합니다.

- 코드 HTTP_NOACTION(값 0)은 아무런 조치도 수행되지 않았음을 의미합니다. 이 코드가 리턴되면 프록시 서버가 이 단계에 대해 기본 조치를 취합니다.
- 유효한 HTTP 리턴 코드 중 하나는 플러그인 함수가 단계를 처리함을 나타냅니다. (유효한 리턴 코드 목록에 대해서는 17 페이지의 『HTTP 리턴 코드 및 값』을 참조하십시오.) 유효한 HTTP 리턴 코드가 제공되면 이 요청의 해당 단계를 처리하기 위해 다른 어떤 플러그인 함수도 호출되지 않습니다.

플러그인 함수 프로토타입

각 Caching Proxy 단계에 대한 함수 프로토타입에서는 사용할 수 있는 형식을 보여주고 수행할 수 있는 처리 유형에 대하여 설명합니다. 함수 이름은 사전 정의되어 있지 않음에 주의하십시오. 사용자 함수의 고유 이름을 지정하고, 자신만의 이름 지정 규칙을 선택할 수 있습니다. 연상하기 쉽도록, 이 문서에서는 서버의 처리 단계와 관련된 이름을 사용합니다.

각각의 플러그인 함수에서 사전 정의된 특정 함수는 유효합니다. 일부 사전 정의된 함수는 모든 단계에 대해 유효하지 않습니다. 다음과 같은 사전 정의된 API 함수는 모든 플러그인 함수에서 호출될 때 유효합니다.

- HTTPD_set
- HTTPD_extract

- httpd_setvar
- httpd_getvar
- HTTPD_log* 함수

추가로 유효하거나 유효하지 않은 API 함수는 함수 프로토타입에 언급되어 있습니다.

함수로 전송되는 *handle* 매개변수의 값은 사전 정의된 함수에 대한 첫 번째 인수로 전달할 수 있습니다. 사전 정의된 API 함수는 18 페이지의 『사전 정의 함수 및 매크로』에서 설명합니다.

서버 초기화

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

서버 초기화 중에 모듈이 로드될 때 이 단계에서 정의된 함수가 호출됩니다. 이 때가 요청이 승인되기 전에 초기화를 수행할 수 있는 시기입니다.

모든 서버 초기화 함수가 호출되어도, 이 단계의 함수에서 오류 리턴 코드가 발생하면 서버는 오류 코드를 리턴한 함수와 동일한 모듈에서 구성된 다른 모든 함수를 무시합니다. (즉, 오류를 리턴한 함수와 동일한 공유 오브젝트에 포함된 다른 함수는 호출되지 않습니다.)

버전 매개변수에는 프록시 서버의 버전 번호가 들어 있습니다. Caching Proxy에 의해 제공됩니다.

사전종료

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

요청이 읽혀진 후 다른 처리가 수행되기 전에 이 단계에서 정의된 함수가 호출됩니다. 이 단계에서 플러그인은 Caching Proxy에 의해 처리되기 전에 클라이언트의 요청에 액세스하는 데 사용할 수 있습니다.

preExit 함수의 유효한 리턴 코드는 다음과 같습니다.

- 0(HTTP_NOACTION)
- 200(HTTP_OK)
- 4*xx* 또는 5*xx* 시리즈의 HTTP 오류(예: 404, HTTP_NOT_FOUND)

다른 리턴 코드는 사용할 수 없습니다.

이 함수가 HTTP_OK를 리턴하는 경우, 프록시 서버는 요청이 처리된 것으로 간주합니다. 모든 후속 요청 처리 단계가 생략되며 응답 단계(변형자, 로그 및 사후종료)만 수행됩니다.

이 단계에서는 사전 정의된 모든 API 함수가 유효합니다.

자정

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

이 단계에서 정의된 함수는 매일 자정에 실행되고 요청 컨텍스트를 포함하지 않습니다. 예를 들어, 로그를 분석하기 위해 하위 프로세스를 호출하는 데 사용할 수 있습니다. (이 단계의 확장 처리가 로깅을 방해할 수 있습니다.)

인증

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

이 단계에서 정의된 함수가 요청의 인증 설계에 따라 요청에 대해 호출됩니다. 이 함수를 사용하여 요청과 함께 전송된 보안 토큰의 검증을 사용자 정의할 수 있습니다.

이름 변환

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

이 단계에서 정의된 함수가 요청에 대해 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 이름 변환 단계는 요청이 처리되기 전에 발생하고 파일 이름처럼 오브젝트에 URL을 맵핑하는 메커니즘을 제공합니다.

권한 부여

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

이 단계에서 정의된 함수가 요청에 대해 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 권한 부여 단계는 요청이 처리되기 전에

발생하고 식별된 오브젝트가 클라이언트로 리턴될 수 있는지를 확인하는 데 사용할 수 있습니다. 기본 인증을 수행하는 경우라면, 필요한 WWW-Authenticate 헤더를 생성해야 합니다.

오브젝트 유형

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

이 단계에서 정의된 함수가 요청에 대해 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 오브젝트 유형 단계는 요청이 처리되기 전에 발생하고 오브젝트가 존재하는지 여부를 점검하는 데 사용할 수 있으며 오브젝트 입력을 수행합니다.

사후 권한 부여

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

요청이 허가된 후 다른 처리가 수행되기 전에 이 단계에서 정의된 함수가 호출됩니다. 이 함수가 HTTP_OK를 리턴하는 경우, 프록시 서버는 요청이 처리된 것으로 간주합니다. 모든 후속 요청 단계가 생략되며 응답 단계(변형자, 로그 및 사후종료)만 수행됩니다.

이 단계에서는 서버에서 사전 정의된 모든 함수가 유효합니다.

서비스

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

이 단계에서 정의된 함수가 요청에 대해 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 서비스 단계는 사전종료 또는 사후 권한 부여 단계에서 해결되지 못한 경우 요청을 해결합니다.

이 단계에서는 서버에서 사전 정의된 모든 함수가 유효합니다.

URL이 아닌 HTTP 메소드를 기반으로 실행되는 사용자 서비스 함수 구성 정보에 대해서는 *WebSphere Application Server Caching Proxy 관리 안내서*의 'Enable 지시문'을 참조하십시오.

변형자 이 프로세스 단계에서 호출된 함수는 스트림으로 필터 응답 데이터에 사용될 수 있습니다. 이 단계에 대한 네 개의 플러그인 함수가 순서대로 호출되며, 각 함

수는 데이터가 흐르는 파이프의 세그먼트로서 작동합니다. 즉, 사용자가 제공한 각 응답의 순서대로 *open*, *write*, *close* 및 *error* 함수가 호출됩니다. 각 함수는 차례대로 동일한 데이터 스트림을 처리합니다.

이 단계에서는 다음과 같은 네 가지 함수를 구현해야 합니다. (함수 이름이 이 이름과 일치할 필요는 없습니다.)

- **Open**

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

  open 함수는 이 스트림의 데이터를 처리하기 위해 필요한 초기화(예: 버퍼 할당)를 수행합니다. HTTP_OK 이외의 리턴 코드가 발생하면 이 필터는 중단됩니다. (write 및 close 함수는 호출되지 않습니다.) 사용자가 구조에 대한 공간을 할당할 수 있도록 함수가 void 포인터를 리턴하고, 이 포인터를 다시 다음 함수의 *correlator* 매개변수에서 사용자에게 전달합니다.

- **Write**

```
void  HTTPD_LINKAGE  writeFunction (
        unsigned char *handle,
        unsigned char *data,    /* response data sent by the origin server */
        unsigned long *length,  /* length of response data */
        void *correlator,       /* pointer returned by the 'open' function */
        long *return_code
        )
```

  write 함수는 데이터를 처리하고, 서버에서 사전 정의되고 새 데이터 또는 변경된 데이터가 있는 HTTPD_write() 함수를 호출할 수 있습니다. 플러그인으로 전달된 버퍼를 비우도록 하거나 받은 버퍼를 서버가 비우도록 해서는 안 됩니다.

  사용자가 write 함수의 범위에서 데이터를 변경하지 않으려면 해당 데이터를 클라이언트에 대한 응답으로 전달하기 위해 open, write 및 close 함수의 범위에서 HTTPD_write 함수를 계속 호출해야 합니다. *correlator* 인수는 *open* 루틴에서 리턴된 데이터 버퍼에 대한 포인터입니다.

- **Close**

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

  close 함수는 이 스트림의 데이터를 처리하는 데 필요한 제거 작업(예: correlator 버퍼의 플러쉬 및 비우기)을 수행합니다. *correlator* 인수는 *open* 루틴에서 리턴된 데이터 버퍼에 대한 포인터입니다.

- **Error**

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

error 함수는 오류 페이지가 전송되기 전에, 제거 작업(예: 버퍼 데이터의 플러쉬 및(또는) 비우기)을 수행할 수 있도록 합니다. 이때 오류 페이지를 처리하기 위해 open, write 및 close 함수가 호출됩니다. *correlator* 인수는 *open* 루틴에서 리턴된 데이터 버퍼에 대한 포인터입니다.

주:

- 변형자 단계에서 플러그인을 작성할 때 사용자는 open, write 및 close 함수 범위 내의 어느 시점에 HTTPD_open(), HTTPD_write() 및 HTTPD_close()를 호출해야 합니다. HTTPD_write()는 HTTPD_open() 함수가 호출된 후에만 호출될 수 있습니다. 이 사전 정의 함수의 목적은 서버에 제어를 넘겨 주어 다음 스트림 클래스가 호출될 수 있도록 하는 것입니다.
- 변형자 API 단계와 서버가 올바르게 실행할 수 있도록 HTTPD_* 함수에 대한 호출은 필요합니다. 예를 들면, HTTPD_open()과 HTTPD_close()가 호출되지 않으면, 헤더는 클라이언트에게 리턴되지 않습니다.
- 데이터 필터링 응용프로그램이 데이터 스트림 필터와 잘 맞지 않는 경우, 예기치 않은 결과가 발생할 수 있으므로 주의하십시오. 필터가 올바르게 수행되지 못한 경우 CGI가 제대로 작동하지 않고, GIF 파일들이 표시되지 않으며 다른 2진 스트림이 예상대로 작동하지 않을 수도 있습니다.
- 플러그인은 내용 본문을 버퍼할 필요는 없습니다. Caching Proxy는 내용의 길이를 자동으로 결정합니다.
- 사용자는 헤더에 대한 제어권을 서버에 넘겨줄 준비가 되어 있을 때 HTTPD_open()을 호출해야 합니다. 그러나 사용자가 API 프로그램에서 나중에 헤더를 설정해야 할 경우, write 또는 close 함수가 HTTPD_open() 함수를 호출할 때까지 기다릴 수 있습니다.

  주: HTTPD open() 함수를 호출하기 전에 HTTPD_set() 또는 httpd_setvar()를 사용하여 헤더를 설정해야 합니다.
- 데이터 스트림에는 헤더가 포함되지 않습니다. 헤더를 조작하려면 플러그인이 set 또는 extract 함수를 사용해야 합니다. 플러그인의 *open* 함수는 모든 헤더가 읽혀질 때까지 호출되지 않습니다.
- 구성 파일에 표시되는 순서로 호출될 여러 개의 변형자 플러그인을 사용할 수 있습니다.
- SSL 터널링은 변형자 플러그인을 통해 전달되지 않습니다.

GC 어드바이저

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

이 단계에서 정의된 함수가 가비지 콜렉션을 수행하는 동안 캐시에서 각 파일에 대해 호출됩니다. 이 함수를 사용하여 어떤 파일을 보관하고 어떤 파일을 버릴지 결정할 수 있습니다. 자세한 정보는 GC_* 변수를 참조하십시오.

프록시 어드바이저

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

이 단계에서 정의된 함수가 각 프록시 요청 서비스 중에 호출됩니다. 예를 들어, USE_PROXY 변수를 설정하기 위해 사용할 수 있습니다.

로그

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

각 요청이 처리되고 클라이언트에 대한 통신이 종료되면 이 단계에서 정의된 함수가 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 이 함수는 요청 처리의 성공 및 실패 여부와 관계없이 호출됩니다. 로그 플러그인으로 기본 로그 메커니즘을 덮어쓰지 않으려면 리턴 코드를 HTTP_OK 대신에 HTTP_NOACTION으로 설정하십시오.

오류

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

이 단계에서 정의된 함수가 실패한 요청에 대해 호출됩니다. 템플리트와 일치하는 요청에 대해서만 플러그인 함수가 호출되도록 하려는 경우, 구성 파일 지시문에 URL 템플리트를 지정할 수 있습니다. 오류 단계에서는 오류 응답을 사용자 정의할 수 있는 기회가 있습니다.

사후종료

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

이 단계에서 정의된 함수가 요청의 성공 및 실패 여부와 관계없이 요청에 대해 호출됩니다. 이 단계에서는 요청을 처리하는 플러그인이 할당한 모든 자원에 대한 타스크를 정리할 수 있습니다.

서버 종료

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

서버가 정상적으로 종료되면 이 단계에서 정의된 함수가 호출됩니다. 서버 초기화 단계에서 할당된 자원에 대한 제거를 사용 가능하게 합니다. 이 단계에서 HTTP_* 함수를 호출하지 마십시오. (예상치 못한 결과가 발생할 수 있습니다.) 사용자 구성 파일에 서버 종료를 위한 하나 이상의 Caching Proxy API 지시문이 있다면 이들 모두가 호출됩니다.

주: Solaris 코드의 현재 제한사항으로 인해 **ibmproxy -stop** 명령을 Solaris 플랫폼의 Caching Proxy 시스템을 종료하는 데 사용해도, 서버 종료 플러그인 단계가 실행되지 않습니다. Caching Proxy 시작 및 중지에 관한 정보는 *WebSphere Application Server Caching Proxy 관리 안내서*를 참조하십시오.

HTTP 리턴 코드 및 값

이들 리턴 코드는 World Wide Web Consortium에서 발표한 HTTP 1.1 스펙, RFC 2616을 따릅니다(www.w3.org/pub/WWW/Protocols/). 사용자 플러그인 함수는 다음 값 중 하나를 리턴합니다.

표 2. Caching Proxy API 함수에 대한 HTTP 리턴 코드

값	리턴 코드
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

표 2. Caching Proxy API 함수에 대한 HTTP 리턴 코드 (계속)

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

사전 정의 함수 및 매크로

서버의 사전 정의된 함수 및 매크로를 사용자 고유의 플러그인 함수로부터 호출할 수 있습니다. 반드시 사전 정의된 이름을 사용하고 아래에 나와 있는 형식을 따라야 합니다. 매개변수 설명에서 문자 `i`는 입력 매개변수를, 문자 `o`는 출력 매개변수를, 그리고 `i/o`는 입력 및 출력 매개변수를 모두 표시합니다.

요청의 성공 여부에 따라 HTTPD 리턴 코드 중 하나를 리턴합니다. 이 코드는 25 페이지의 『사전 정의 함수 및 매크로의 리턴 코드』에서 설명합니다.

이 함수를 호출할 때 첫 번째 매개변수로 사용자의 플러그인에 제공된 핸들을 사용하십시오. 그렇지 않으면 함수는 HTTPD_PARAMETER_ERROR 오류 코드를 리턴합니다. NULL은 유효한 핸들로 인정되지 않습니다.

HTTPD_authenticate()

사용자 ID나 암호 혹은 둘 모두를 인증합니다. 사전종료, 인증, 권한 부여 및 사후 권한 부여 단계에서만 유효합니다.

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

HTTPD_cacheable_url()

지정된 URL 컨텐츠를 Caching Proxy의 표준에 따라 캐시할 수 있습니다.

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

리턴값 HTTPD_SUCCESS는 URL 컨텐츠가 캐시 가능함을 표시하고, HTTPD_FAILURE는 컨텐츠가 캐시 불가능함을 표시합니다.
HTTPD_INTERNAL_ERROR도 이 함수에 대해 가능한 리턴값입니다.

HTTPD_close()

(변형자 단계에서만 유효합니다.) 스트림 스택의 다음 *close* 루틴으로 제어를 전송합니다. 원하는 프로세스가 완료되면 변형자 open, write 또는 close 함수에서 이 함수를 호출하십시오. 이 함수는 응답이 프로세스되고 Transmogrifier 단계가 완료되는 프록시 서버를 통지합니다.

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

HTTPD_exec()

이 요청을 충족시키기 위해 스크립트를 실행합니다. 사전종료, 서비스, 사후 권한 부여 및 오류 단계에서 유효합니다.

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

HTTPD_extract()

이 요청과 관련된 변수값을 추출합니다. *name* 매개변수에 대한 유효한 변수는 CGI에서 사용된 것과 동일합니다. 자세한 정보는 29 페이지의 『변수』의 내용을 참조하십시오. 이 함수는 모든 단계에서 유효하지만 모든 변수에 대해 유효하지 않음에 유의하십시오.

```
void  HTTPD_LINKAGE  HTTPD_extract (
         unsigned char *handle,        /* i; handle */
         unsigned char *name,          /* i; name of variable to extract */
```

```
        unsigned long *name_length,   /* i; length of the name */
        unsigned char *value,         /* o; buffer in which to put
                                         the value */
        unsigned long *value_length,  /* i/o; buffer size */
        long *return_code             /* o; return code */
    )
```

만일 이 함수가 코드 HTTPD_BUFFER_TOO_SMALL을 리턴하게 되면 사용자가 요청한 버퍼는 추출한 값에 비해 *크기*가 충분하지 않습니다. 이 경우, 함수가 버퍼를 사용하지 않지만, 이 값을 추출하기 위해 필요한 버퍼 *크기*로 value_length 매개변수를 갱신합니다. *크기*가 최소한 리턴된 value_length인 버퍼를 사용하여 다시 추출하십시오.

주: 추출 중인 변수가 HTTP 헤더용인 경우 요청이 동일한 이름의 여러 헤더를 포함하더라도 HTTPD_extract() 함수가 첫 번째 일치하는 어커런스만 추출합니다. httpd_getvar() 함수가 HTTPD_extract() 대신 사용될 수 있으며 다른 장점을 제공합니다. 자세한 정보는 20 페이지를 참조하십시오.

HTTPD_file()

이 요청을 충족시키기 위해 파일을 전송합니다. 사전종료, 서비스, 오류, 사후 권한 부여 및 변형자 단계에서만 유효합니다.

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

httpd_getvar()

사용자가 인수 길이를 지정할 필요가 없기 때문에 사용하기 쉽다는 것을 제외하고는 HTTPD_extract()와 같습니다.

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

헤더를 포함하는 배열의 색인 번호는 0부터 시작합니다. 배열의 첫 번째 항목을 가져 오려면 *n* 값으로 0을 사용하고, 다섯 번째 항목을 가져 오려면 *n* 값으로 4를 사용하십시오.

주: 리턴된 값의 내용을 버리거나 변경하지 말아야 합니다. 리턴된 문자열은 널로 종료됩니다.

HTTPD_log_access()

서버의 액세스 로그에 문자열을 기록합니다.

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

서버 액세스 로그에서 퍼센트 기호(%)를 기록할 경우 에스케이프 기호는 필요하지 *않습니다*.

HTTPD_log_error()

서버의 오류 로그에 문자열을 기록합니다.

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

서버 오류 로그에서 퍼센트 기호(%)를 기록할 경우 에스케이프 기호가 필요하지 *않습니다*.

HTTPD_log_event()

서버의 이벤트 로그에 문자열을 기록합니다.

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

서버 이벤트 로그에서 퍼센트 기호(%)를 기록할 경우 에스케이프 기호가 필요하지 *않습니다*.

HTTPD_log_trace()

이렇게 되면 서버의 추적 로그에 문자열을 기록합니다.

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

서버 추적 로그에서 퍼센트 기호(%)를 기록할 경우 에스케이프 기호가 필요하지 *않습니다*.

HTTPD_open()

(변형자 단계에서만 유효합니다.) 스트림 스택의 다음 루틴으로 제어를 전송합니다. 일단 원하는 헤더가 설정되고 작성 루틴의 시작이 준비되면 이를 변형자 open, write, close 함수에서 호출하십시오.

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

HTTPD_proxy()

프록시 요청을 합니다. 사전종료, 서비스 및 사후 권한 부여 단계에서 유효합니다.

주: 이것은 완료 함수로서 이 함수 다음에 요청이 완료됩니다.

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

HTTPD_read()

클라이언트 요청의 본문을 읽습니다. 헤더에 HTTPD_extract()를 사용합니다. 사전종료, 권한 부여, 사후 권한 부여 및 서비스 단계에서만 유효하며, PUT 또는 POST 요청이 수행된 경우에만 유용합니다. HTTPD_EOF가 리턴될 때까지 루프에서 함수를 호출하십시오. 이 요청에 대한 본문이 없는 경우 이 함수는 실패합니다.

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

HTTPD_restart()

모든 활성화 요청이 처리된 후 서버를 재시작합니다. 서버 초기화, 서버 종료 및 변형자를 제외한 모든 단계가 유효합니다.

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

HTTPD_set()

이 요청과 연관된 변수 값을 설정합니다. *name* 매개변수에 대한 유효한 변수는 CGI에서 사용된 것과 동일합니다. 자세한 정보는 29 페이지의 『변수』의 내용을 참조하십시오.

이 함수를 사용하면 변수도 작성할 수 있음에 유의하십시오. 사용자가 작성한 변수는 29 페이지의 『변수』에 설명된 HTTP_ 및 PROXY_ 접두부에 대한 협약에 의존합니다. HTTP_로 시작하는 변수를 작성하면 그 응답으로, HTTP_

접두부없이 클라이언트에 헤더로 전송됩니다. 예를 들어, 위치 헤더를 설정하려면 변수 이름 HTTP_LOCATION과 함께 HTTPD_set()을 사용하십시오. PROXY_ prefix로 작성된 변수는 컨텐츠 서버에 대해 요청한 헤더로 전송됩니다. CGI_ 접두부로 작성된 변수는 CGI 프로그램에 전달됩니다.

이 함수는 모든 단계에서 유효하지만 모든 변수가 모든 단계에서 유효하지는 않습니다.

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

주: httpd_setvar() function 함수를 사용하여 버퍼 및 길이를 지정하지 않고도 변수 값을 설정할 수 있습니다. 자세한 정보는 23 페이지를 참조하십시오.

httpd_setvar()

사용자가 인수 길이를 지정할 필요가 없어서 사용하기 쉽다는 것을 제외하고는 HTTPD_extract()와 같습니다.

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

addHdr 매개변수의 가능한 값에는 네 개가 있습니다.

- HTTPD_SETVAR_REPLACE — 헤더 변수의 모든 어커런스를 새 값으로 바꿉니다.
- HTTPD_SETVAR_REPLACE_ADD — 헤더 변수가 존재하면 첫 번째 어커런스를 새 값으로 바꾸고, 그렇지 않으면 새 값을 헤더에 추가합니다.
- HTTPD_SETVAR_ADD — 이 값을 헤더에 추가합니다.
- HTTPD_SETVAR_REMOVE_ALL — 이 헤더 변수의 모든 어커런스를 삭제합니다.

이 값은 HTAPI.h에서 정의됩니다.

httpd_variant_insert()

변형을 캐시에 삽입합니다.

```
void  HTTPD_LINKAGE  httpd_variant_insert (
         unsigned char *handle,    /* i; handle */
         unsigned char *URI,       /* i; URI of this object */
         unsigned char *dimension, /* i; dimension of variation */
```

```
unsigned char *variant,   /* i; value of the variant */
unsigned char *filename,  /* i; file containing the object */
long *return_code         /* o; return code */
)
```

주:

1. dimension 인수는 URI에서 헤더에 따라 변경되는 오브젝트의 헤더를 나타냅니다. 예를 들면, 위의 예제에서 dimension 값은 User-Agent가 될 수 있습니다.
2. variant 인수는 dimension 인수에서 주어진 헤더 값을 나타냅니다. 이것은 URI에서 변경됩니다. 예를 들면, 위의 예제에서 variant 인수의 값은 다음과 같습니다.

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

3. filename 인수는 변경된 내용을 저장하고 있는 파일 이름의 널 종료 사본을 지정하게 됩니다. 사용자에게는 이 조치가 이 함수로부터 리턴된 후 파일을 제거할 책임이 있습니다. 파일에는 헤더 없이 본문만 포함됩니다.
4. 변형을 캐시할 때 서버는 내용 길이 헤더를 갱신하고, ″Warning: 214″ 헤더를 추가합니다. Strong 엔티티 태그는 제거됩니다.

httpd_variant_lookup()

주어진 변형이 캐시에 있는지 판별합니다.

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

HTTPD_write()

응답의 본문을 기록합니다. 사전종료, 서비스, 오류 및 변형자 단계에서 유효합니다.

내용 유형을 설정하여 이 함수를 처음으로 호출할 경우, 서버는 사용자가 CGI 데이터 스트림을 전송하고 있는 것으로 받아들입니다.

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

주: 응답 헤더를 설정하려면 22 페이지를 참조하십시오.

주: HTTPD_* 함수가 리턴된 후에는 이와 함께 전달했던 메모리를 제거해도 안전합니다.

사전 정의 함수 및 매크로의 리턴 코드

서버는 요청의 성공 여부에 따라 리턴 코드 매개변수를 이들 값 중 하나로 설정합니다.

표 3. 리턴 코드

값	상태 코드	설명
-1	HTTPD_UNSUPPORTED	이 함수는 지원되지 않습니다.
0	HTTPD_SUCCESS	함수가 정상적으로 수행되었으며 출력 필드가 유효합니다.
1	HTTPD_FAILURE	함수가 실패했습니다.
2	HTTPD_INTERNAL_ERROR	내부 오류가 발생해서 이 요청을 계속 처리할 수 없습니다.
3	HTTPD_PARAMETER_ERROR	하나 이상의 올바르지 않은 매개변수가 전달되었습니다.
4	HTTPD_STATE_CHECK	이 함수는 이 프로세스 단계에 올바르지 않습니다.
5	HTTPD_READ_ONLY	(HTTPD_set 및 httpd_setvar에 의해서만 리턴됨.) 이 변수는 읽기 전용이며 플러그인으로 설정할 수 있습니다.
6	HTTPD_BUFFER_TOO_SMALL	(HTTPD_set, httpd_setvar 및 HTTPD_read에 의해 리턴됨.) 제공된 버퍼가 너무 작습니다.
7	HTTPD_AUTHENTICATE_FAILED	(HTTPD_authenticate에 의해서만 리턴됨.) 인증이 실패했습니다. 자세한 정보는 HTTP_RESPONSE 및 HTTP_REASON 변수를 점검하십시오.
8	HTTPD_EOF	(HTTPD_read에 의해서만 리턴됨.) 요청 본문의 끝을 표시합니다.
9	HTTPD_ABORT_REQUEST	클라이언트가 요청에 의해 지정된 조건에 일치하지 않는 엔티티 태그를 제공해서, 이 요청은 중단됩니다.
10	HTTPD_REQUEST_SERVICED	(HTTPD_proxy에 의해서만 리턴됨.) 호출된 이 함수가 이 요청에 대한 응답을 완료했습니다.
11	HTTPD_RESPONSE_ALREADY_COMPLETED	요청에 대한 응답이 이미 완료되어 이 함수가 실패했습니다.
12	HTTPD_WRITE_ONLY	이 변수는 쓰기 전용이며 플러그인으로 읽을 수 없습니다.

API 단계에 대한 Caching Proxy 구성 지시문

요청 프로세스의 각 단계에는 구성 지시문이 있고 이는 각 단계에서 호출하여 실행하고자 하는 플러그인 함수를 표시할 수 있게 합니다. 사용자 서버의 구성 파일(ibmproxy.conf)을 수동으로 편집 또는 갱신하거나, Caching Proxy 구성 및 관리 양식의 API 요청 처리 양식을 이용하여 지시문을 사용자 서버의 구성 파일에 추가할 수 있습니다.

API 사용시 주의사항

- Service 및 NameTrans 지시문을 제외하고 각 단계에 대한 API 지시문은 구성 파일에서 특정 순서로 나타날 필요가 없습니다. 이 목록의 뒤에서 설명되는 바와 같이 하나의 API 지시문의 여러 항목들 간의 순서는 매우 중요합니다.
- 모든 API 단계에 대한 항목을 포함할 필요는 없습니다. 특정 단계에 대한 플러그인이 없을 경우, 해당 지시문을 생략하면 해당 단계에 대해서는 표준 처리가 사용됩니다.
- Service 및 NameTrans 지시문은 다른 맵핑 지시문들(예: Pass 지시문)처럼 작용하며 구성 파일 내에 있는 다른 맵핑 지시문과 연관된 어커런스 및 위치는 구성 파일에 따라 다릅니다. 예를 들어, /cgi-bin/foo.so에 대한 규칙은 /cgi-bin/*에 대한 규칙 앞에 나와야 합니다.

  이는 서버가 구성 파일 내에서 Service, NameTrans, Exec, Fail, Map, Pass, Proxy, ProxyWAS 및 Redirect 지시문 순으로 처리함을 의미합니다. URL을 파일에 성공적으로 맵핑하면, 다른 지시문을 읽거나 처리하지 않습니다. (Map 지시문은 예외입니다. 프록시 서버 맵핑 규칙에 대한 자세한 정보는 *WebSphere Application Server Caching Proxy 관리 안내서*를 참조하십시오.)
- 한 단계에서 하나 이상의 구성 지시문을 지정할 수 있습니다. 예를 들어, 사용자는 서로 다른 플러그인 함수를 가리키는 두 개의 NameTrans 지시문을 가질 수 있습니다. 서버가 이름 변환 단계를 수행할 때는 구성 파일 안에 나타나는 순서대로 이름 변환 함수들을 처리합니다.

  주: Caching Proxy와 함께 제공된 플러그인 함수가 사용자가 작성한 플러그인과 동일한 API 지시문을 사용하는 경우, 사용자의 플러그인 지시문을 시스템의 플러그인 지시문 다음에 위치시키십시오.
- 특정 플러그인 함수가 모든 요청에 대해 실행될 필요는 없습니다.
  - 여러 지시문에는 URL 마스크가 포함됩니다. 이런 지시문으로 URL 마스크를 지정하면 URL이 패턴과 일치하는 요청에 대해서만 플러그인 응용프로그램이 호출됩니다. URL 마스크를 사용할 수 있는 단계에 대한 정보는 27 페이지의 『API 지시문 및 구문』을 참조하고 이 기능의 사용법에 대해서는 27 페이지의 『API 지시문 변수』를 참조하십시오.
  - Authentication 지시문에 인증 설계를 지정하여 특정 유형의 인증에 대해서만 플러그인을 호출할 것임을 나타낼 수도 있습니다. 현재 HTTP 프로토콜에서는 기본 인증만 지원합니다. 자세한 정보는 27 페이지의 『API 지시문 변수』를 참조하십시오.
- 서버가 특정 플러그인 함수의 로드에 실패하거나, 확인 리턴 코드를 리턴하지 않는 ServerInit 지시문이 있는 경우, 컴파일된 Caching Proxy 플러그인에 대한 다른 어

떤 플러그인 함수도 호출되지 않습니다. 현 시점까지 해당 플러그인에 대해 수행된 특정한 처리는 무시됩니다. 이런 지시문에 포함된 다른 Caching Proxy 플러그인과 그 함수는 영향을 받지 않습니다.

API 지시문 및 구문

이러한 구성 파일 지시문은 ibmproxy.conf 파일에서 한 행(여기서 명시적으로 지정한 공백 외에는 공백이 없음)으로 나타납니다. 구문 예제에서 읽기 쉽도록 행을 바꾸었더라도 실제 지시문의 이 지점에서는 공백이 없어야 합니다.

표 4. Caching Proxy 플러그인 API 지시문

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

API 지시문 변수

이 지시문들의 변수에는 다음과 같은 의미가 있습니다.

type 사용자 플러그인이 호출되는지 여부를 지정하는 인증 지시문에만 사용됩니다. 유효한 값은 다음과 같습니다.

- 기본 — 플러그인 함수는 기본 인증 요청에 대해서만 호출됩니다.
- * — 플러그인 함수는 모든 요청에 대해서 호출됩니다. HTTP 프로토콜은 현재 기본 인증만을 지원합니다. 기본 인증 요청이 아닌 것에 대해 사용자는 이 유형의 인증은 지원되지 않음을 나타내는 오류 코드를 리턴할 수 있습니다.

URL 사용자 플러그인 함수가 호출되는 요청을 지정합니다. 이 템플리트와 일치하는 URL의 요청으로 인해 플러그인 함수가 사용됩니다. 이 지시문의 URL 스펙은 가상이지만(프로토콜이 포함되어 있지 않음) 앞에 슬래시(/)가 옵니다. 예를 들

어, /www.ics.raleigh.ibm.com은 올바르지만 http://www.ics.raleigh.ibm.com은 올바르지 않습니다. 이 값을 특정 URL 또는 템플리트로 지정할 수 있습니다.

- 특정 URL — 플러그인 함수는 실행하는 URL에 대해서만 호출됩니다.
- URL 템플리트 — 플러그인 함수는 템플리트와 일치하는 모든 URL에 대해서 호출됩니다. 템플리트는 와일드 카드 문자를 포함할 수 있으며 */URL** 또는 /* 또는 * 양식으로 지정할 수 있습니다.

  주: 사용자가 경로 변환을 하고자 한다면 Service 지시문에 URL 템플리트를 *필수*로 갖추어야 합니다.

path/file

사용자의 컴파일된 프로그램에 대한 완전한 파일 이름

function_name

프로그램 내의 플러그인 함수에 부여한 이름

Service 지시문에서는 경로 정보에 액세스하고자 하는 경우 함수 이름 뒤에 별표(*)를 지정해야 합니다.

init_string

ServerInit 지시문의 이 선택적 파트는 사용자 플러그인 함수에 전달하려는 임의의 텍스트를 포함할 수 있습니다. httpd_getvar()을 사용하여 INIT_STRING 변수에서 텍스트를 추출하십시오.

구문을 포함하여 이 지시문에 대한 자세한 정보는 *WebSphere Application Server Caching Proxy 관리 안내서*를 참조하십시오.

다른 API와의 호환성

Caching Proxy API는 버전 4.6.1을 통해 ICAPI 및 GWAPI와의 역호환이 가능합니다.

CGI 프로그램 포팅

Caching Proxy API를 사용하려면 C로 작성된 CGI 응용프로그램을 포팅하는 지침을 사용하십시오.

- DLL을 작성할 수 있도록 main() 진입점을 제거하거나 이름을 변경하십시오.
- 글로벌 변수들을 없애거나 상호 배타적인 세마포어를 사용하여 변수를 보호하십시오.
- 사용자 프로그램에서 다음 호출들을 변경하십시오.
  - printf() 헤더 호출을 HTTPD_set() 또는 httpd_setvar()로 변경하십시오.
  - printf() 데이터 호출을 HTTPD_write()로 변경하십시오.

- getenv() 호출을 HTTPD_extract() 또는 httpd_getvar()로 변경하십시오. 이것은 비할당된 메모리를 리턴하므로 사용자는 반드시 결과를 제거해야 합니다.

- 서버는 다중 스레드 환경에서 실행되므로 플러그인 함수는 호출의 간섭없이 다중 동시 처리가 가능(thread safe)해야 합니다. 함수가 재진입 가능한 경우, 성능은 저하되지 않습니다.
- 데이터를 다시 클라이언트로 전송하기 위하여 HTTP_write()를 사용하는 경우, 컨텐츠 유형 헤더를 반드시 설정하십시오.
- 메모리 누출은 없는지 코드를 주의 깊게 확인하십시오.
- 사용자의 오류 경로에 대하여 생각해 보십시오. 사용자 스스로 오류 메시지를 만들어 HTML로 다시 보내려 한다면 반드시 서비스 함수 또는 함수들로부터 HTTPD_OK를 리턴해야 합니다.

Caching Proxy API 참조 정보

변수

API 프로그램을 작성할 때, 원격 클라이언트와 서버 시스템에 대한 정보를 제공하는 Caching Proxy 변수를 사용할 수 있습니다.

주:

- 사용자 정의 변수 이름에는 SERVER_ 접두부를 붙일 수 없습니다. Caching Proxy 서버 API 함수에서 SERVER_로 시작하는 변수는 모두 서버용으로 보유되어 있으므로 이들 변수는 읽기 전용입니다. 또한 접두부 HTTP_ 및 PROXY_도 HTTP 헤더용으로 보유되어 있습니다.
- 접두부 HTTP_가 있는 클라이언트(Set-Cookie 등)가 전송하는 모든 요청 헤더와 그 값은 추출할 수 있습니다. 헤더를 요청하는 변수에 액세스하려면 변수 이름에 접두부 HTTP_를 붙이십시오. HTTPD_set() 사전 정의 함수를 사용하여 새로운 변수를 작성할 수 있습니다. 이 헤더에 대한 자세한 내용은 25 페이지의 『사전 정의 함수 및 매크로의 리턴 코드』를 참조하십시오.
- 변수가 요청이나 응답할 헤더에 적용되는지를 나타내는 데 사용되는 두 가지의 변수 접두부(HTTP_ 및 PROXY_)가 있습니다. HTTP_ 접두부는 클라이언트와 Caching Proxy 간에 플로우하는 변수를 참조합니다. PROXY_ 접두부는 Caching Proxy와 기점 서버(또는 프록시 체인의 다음 서버) 간에 플로우하는 변수를 참조합니다. 변수는 요청 처리 단계 중에만 유효합니다.
  - HTTP_* 변수를 추출하면 프록시 서버에 대한 클라이언트의 요청에 사용된 헤더의 값이 사용자에게 제공됩니다.
  - HTTP_* 변수를 설정하면 프록시 서버에서 클라이언트로 전송된 응답 헤더를 설정하게 됩니다.

– PROXY_* 값을 추출하면 컨텐츠 서버에서 프록시 서버로 리턴된 헤더의 값이 사용자에게 제공됩니다.
– PROXY_* 변수를 설정하면 프록시 서버에서 컨텐츠 서버(또는 프록시 체인의 다음 서버)로 요청 헤더를 설정하게 됩니다.

그림 2에서는 Caching Proxy가 클라이언트 요청을 처리할 때 이런 접두부의 사용을 표시합니다.

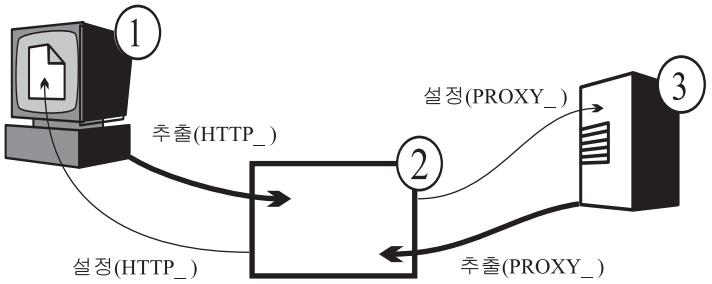

그림 2. HTTP_ 및 PROXY_ 변수 접두부. 범례: 1—클라이언트 시스템 2—Caching Proxy 3—기점 서버

- 일부 변수들은 읽기 전용입니다. 읽기 전용 변수 이름은 요청에서 추출할 수 있는 값을 나타내며, httpd_getvar() 사전 정의 함수에서 사용될 수 있습니다. 사용자가 httpd_setvar() 함수를 사용하여 읽기 전용 변수를 변경하려고 하는 경우 HTTPD_READ_ONLY 리턴 코드 결과를 가지게 됩니다.
- 읽기 전용이 아닌 변수는 읽을 수 있으며 httpd_getvar() 또는 httpd_setvar() 사전 정의 함수로 설정될 수 있습니다. 이들 변수는 사용자가 요청이나 응답에서 추출한 값 또는 요청이나 응답을 처리하는 동안 사용자가 설정하거나 작성할 수 있는 값을 나타냅니다.

변수 정의

주: HTTP_ 또는 PROXY_ 접두부로 시작하지 않는 헤더 변수가 모호합니다. 모호한 변수를 방지하려면 항상 헤더에 대한 변수 이름과 함께 HTTP_ 또는 PROXY_ 접두부를 사용하십시오.

ACCEPT_RANGES

컨텐츠 서버가 범위 요청에 응답할 수 있는지의 여부를 지정하는 Accept-Range 응답 헤더의 값을 포함합니다. PROXY_ACCEPT_RANGES를 사용하여 컨텐츠 서버가 프록시로 전송하는 헤더값을 추출하십시오.

HTTP_ACCEPT_RANGES를 사용하여 프록시에서 클라이언트로 전송된 헤더 값을 설정하십시오.

주: ACCEPT_RANGES는 모호합니다. 이러한 모호성을 없애려면 HTTP_ACCEPT_RANGES 및 PROXY_ACCEPT_RANGES를 대신 사용하십시오.

ALL_VARIABLES

읽기 전용. 모든 CGI 변수를 포함합니다. 예제:

```
ACCEPT_RANGES BYTES
CLIENT_ADDR 9.67.84.3
```

AUTH_STRING

읽기 전용. 서버가 클라이언트 인증을 지원할 경우, 이 문자열에는 클라이언트 인증을 위해 사용될 해독되지 않는 자격사항이 포함됩니다.

AUTH_TYPE

읽기 전용. 서버가 클라이언트 인증을 지원하고 스크립트가 보호되는 경우, 이 변수에는 클라이언트 인증에 사용되는 메소드가 포함됩니다. 예를 들면, Basic과 같습니다.

CACHE_HIT

읽기 전용. 프록시 요청이 캐시에서 발견되었는지 여부를 식별합니다. 리턴되는 값에는 다음이 포함됩니다.

- 0 - 캐시에서 요청을 찾지 못했습니다.
- 1 - 캐시에서 요청을 찾았습니다.

CACHE_MISS

쓰기 전용으로, 캐시 실패(miss)를 강요하는 데 사용됩니다. 유효한 값은 다음과 같습니다.

- 0 - 캐시 실패(miss)를 강요하지 않습니다.
- 1 - 캐시 실패(miss)를 강요합니다.

CACHE_TASK

읽기 전용. 캐시 사용 여부를 식별합니다. 리턴되는 값에는 다음이 포함됩니다.

- 0 - 요청이 캐시에 액세스하거나 캐시를 갱신하지 않았습니다.
- 1 - 요청이 캐시로부터 제공되었습니다.
- 2 - 요청한 오브젝트가 캐시 내에 있지만 다시 유효화되어야 합니다.
- 3 - 요청한 오브젝트가 캐시에 없으므로 추가됩니다.

이 변수는 PostAuthorization, PostExit, ProxyAdvisor 또는 Log steps에서 사용됩니다.

CACHE_UPDATE

읽기 전용. 프록시 요청이 캐시를 갱신했는지 여부를 식별합니다. 리턴되는 값에는 다음이 포함됩니다.

- 0 - 캐시를 갱신하지 않았습니다.

- 1 - 캐시를 갱신했습니다.

CLIENT_ADDR 또는 CLIENTADDR

REMOTE_ADDR과 같습니다.

CLIENTMETHOD

REQUEST_METHOD와 같습니다.

CLIENT_NAME 또는 CLIENTNAME

REMOTE_HOST와 같습니다.

CLIENT_PROTOCOL 또는 CLIENTPROTOCOL

요청을 하는데 사용되는 프로토콜의 이름 및 버전이 포함됩니다. 예를 들면, HTTP/1.1과 같습니다.

CLIENT_RESPONSE_HEADERS

읽기 전용. 서버가 클라이언트로 전송할 헤더를 포함하는 버퍼를 리턴합니다.

CONNECTIONS

읽기 전용. 수행되는 연결 수 또는 활성화된 요청 수를 포함합니다. 예를 들면, 15와 같습니다.

CONTENT_CHARSET

US ASCII와 같은 텍스트/*에 대한 응답의 문자 세트입니다. 이 변수를 추출하면 클라이언트로부터 오는 내용 문자 헤더에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대한 요청의 내용 문자 세트 헤더에 영향을 줍니다.

CONTENT_ENCODING

x-gzip과 같은 문서에서 사용된 인코딩을 지정합니다. 이 변수를 추출하면 클라이언트로부터 오는 내용 코드화 헤더에 영향을 줍니다. 이를 설정하면 컨텐츠 서버에 대한 요청의 내용 문자 세트 헤더에 영향을 줍니다.

CONTENT_LENGTH

이 변수를 추출하면 클라이언트 요청으로부터의 헤더에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더 값에 영향을 줍니다.

주: CONTENT_LENGTH가 모호합니다. 이러한 모호성을 없애려면 HTTP_CONTENT_LENGTH와 PROXY_CONTENT_LENGTH를 사용해야 합니다.

CONTENT_TYPE

이 변수를 추출하면 클라이언트 요청으로부터의 헤더에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더 값에 영향을 줍니다.

주: CONTENT_TYPE이 모호합니다. 이러한 모호성을 없애려면 HTTP_CONTENT_TYPE과 PROXY_CONTENT_TYPE을 사용해야 합니다.

CONTENT_TYPE_PARAMETERS

문자 세트가 아닌 다른 MIME 속성을 포함합니다. 이 변수를 추출하면 클라이언트로부터 요청된 헤더 값에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더 값에 영향을 줍니다.

DOCUMENT_URL

URL(Uniform Request Locator)을 포함합니다. 예제:

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

DOCUMENT_URI

DOCUMENT_URL과 같습니다.

DOCUMENT_ROOT

읽기 전용. 전달 규칙으로 정의된 문서 루트 경로를 포함합니다..

ERRORINFO

오류 페이지를 판별하기 위한 오류 코드를 지정합니다. 예를 들면, `blocked`와 같습니다.

EXPIRES

프록시 캐시에 저장된 문서들의 유효 기간을 나타냅니다. 이 변수를 추출하면 클라이언트 요청으로부터의 헤더에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더 값에 영향을 줍니다. 예제:

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

GATEWAY_INTERFACE

읽기 전용. 서버가 사용하고 있는 API 버전이 포함됩니다. 예를 들면, `ICSAPI/2.0`과 같습니다.

GC_BIAS

쓰기 전용으로, 이 부동 소수점 값은 가비지 콜렉션에 대해 고려되는 파일에 대한 가비지 콜렉션 결정에 영향을 줍니다. 입력된 값에 Caching Proxy 파일 형식의 품질 설정을 곱하여 순위를 결정합니다. 품질 설정은 범위가 0.0 - 0.1이고 프록시 구성 파일(ibmproxy.conf)의 AddType 지시문으로 정의됩니다.

GC_EVALUATION

쓰기 전용으로, 이 부동 소수점 값은 가비지 콜렉션에 대해 고려되는 파일에 대하여 제거(0.0)할 것인지 또는 보관(1.0)할 것인지를 판별합니다. 0.0 - 1.0 사이 값의 순서는 순위에 따라 지정됩니다. 즉, GC_EVALUATION 0.1 값의 파일은 GC_EVALUATION 0.9 값의 파일보다 더 제거되기 쉽습니다.

GC_EXPIRES

읽기 전용. 해당 파일이 캐시에서 만기될 때까지 몇 초가 남아있는지 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

GC_FILENAME

읽기 전용. 가비지 콜렉션에 대해 고려되는 파일을 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

GC_FILESIZE

읽기 전용. 가비지 콜렉션에 대해 고려되는 파일 크기를 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

GC_LAST_ACCESS

읽기 전용. 파일이 마지막으로 액세스된 시기를 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

GC_LAST_CHECKED

읽기 전용. 파일이 마지막으로 확인된 시기를 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

GC_LOAD_DELAY

읽기 전용. 파일을 검색하는 데 걸리는 기간을 식별합니다. 이 변수는 GC 어드바이저 플러그인만이 추출할 수 있습니다.

HTTP_COOKIE

읽기를 할 때 변수는 클라이언트에 의해서 설정된 Set-Cookie 헤더 값을 포함합니다. 또한 응답 스트림에서 새 쿠키를 설정하는 데 사용될 수도 있습니다(프록시와 클라이언트 사이). 이 변수를 설정하면 복제 헤더의 존재 여부를 막론하고 문서 요청 스트림 내의 새로운 Set-Cookie 헤더를 작성하게 됩니다.

HTTP_HEADERS

읽기 전용. 모든 클라이언트 요청 헤더를 추출하는 데 사용됩니다.

HTTP_REASON

이 변수를 설정하면 HTTP 응답의 이유 문자열에 영향을 줍니다. 또한 클라이언트에 대한 프록시의 응답의 이유 문자열에도 영향을 줍니다. 이 변수를 추출하면 컨텐츠 서버에서 프록시로 응답의 이유 문자열을 리턴합니다.

HTTP_RESPONSE

이 변수를 설정하면 HTTP 응답의 응답 코드에 영향을 줍니다. 또한 클라이언트에 대한 프록시의 응답의 상태 코드에도 영향을 줍니다. 이 변수를 추출하면 컨텐츠 서버에서 프록시로 응답의 상태 코드를 리턴합니다.

HTTP_STATUS

HTTP 응답 코드 및 이유 문자열이 들어 있습니다. 예를 들면, `200 OK`와 같습니다.

HTTP_USER_AGENT

클라이언트 웹 브라우저(예: `Netscape Navigator / V2.02`)의 이름인 사용

자 에이전트 요청 헤더를 포함합니다. 이 변수를 설정하면 클라이언트에 대한 프록시의 응답 헤더에 영향을 줍니다. 이를 추출하면 클라이언트로부터 요청된 헤더에 적용됩니다.

INIT_STRING

읽기 전용. ServerInit 지시문이 이 문자열을 정의합니다. 이 변수는 서버 초기화 단계에서만 읽을 수 있습니다.

LAST_MODIFIED

이 변수를 추출하면 클라이언트로부터 요청된 헤더 값에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더 값에 영향을 줍니다. 예제:

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

LOCAL_VARIABLES

읽기 전용. 모든 사용자 정의 변수입니다.

MAXACTIVETHREADS

읽기 전용. 최대 활성 스레드입니다.

NOTMODIFIED_TO_OK

클라이언트에 대한 완전 응답을 강요합니다. PreExit 및 ProxyAdvisor 단계에서 유효합니다.

ORIGINAL_HOST

읽기 전용. 요청의 호스트 이름 또는 대상 IP 주소를 리턴합니다.

ORIGINAL_URL

읽기 전용. 클라이언트 요청에서 전송되는 원래 URL을 리턴합니다.

OVERRIDE_HTTP_NOTRANSFORM

Cache-Control: no-transform 헤더가 있을 경우 데이터 수정을 허용합니다. 이 변수를 설정하면 클라이언트에 대한 응답 헤더에 영향을 줍니다.

OVERRIDE_PROXY_NOTRANSFORM

Cache-Control: no-transform 헤더가 있을 경우 데이터 수정을 허용합니다. 이 변수를 설정하면 컨텐츠 서버에 대한 요청에 영향을 줍니다.

PASSWORD

기본 인증을 위한 변수로, 해독된 암호를 포함합니다. 예를 들면, `password`와 같습니다.

PATH

완전히 변환된 경로를 포함합니다.

PATH_INFO

웹 브라우저에 의해 전송된 추가 경로 정보가 포함됩니다. 예를 들면, `/foo`와 같습니다.

PATH_TRANSLATED

PATH_INFO에 포함된 경로 정보의 해독되거나 변환된 버전이 포함됩니다. 예제:

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

```
/wwwhome/foo
```

PPATH

부분적으로 변환된 경로를 포함합니다. 이름 변환 단계에서 이를 사용하십시오.

PROXIED_CONTENT_LENGTH

읽기 전용. 프록시 서버를 통해 실제로 전송되는 응답 데이터의 길이를 리턴합니다.

PROXY_ACCESS

요청이 프록시 요청인지 여부를 정의합니다. 예를 들면, NO와 같습니다.

PROXY_CONTENT_TYPE

HTTPD_proxy()를 통해 이루어진 프록시 요청의 컨텐츠 유형 헤더를 포함합니다. POST 메소드를 사용하여 정보를 전송할 경우, 이 변수에는 포함된 데이터의 유형이 들어 있습니다. 프록시 서버 구성 파일에 사용자가 직접 컨텐츠 유형을 작성하고 이를 표시기에 맵핑할 수 있습니다. 이 변수를 추출하면 컨텐츠 서버 응답으로부터 받은 헤더값에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더에 영향을 줍니다. 예제:

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

PROXY_CONTENT_LENGTH

HTTPD_proxy()를 통해 이루어진 프록시 요청의 컨텐츠 길이 헤더를 포함합니다. 정보가 POST 메소드를 사용하여 전송될 때, 이 변수에는 데이터의 문자 수가 포함됩니다. 표준 입력을 사용하여 정보를 전달할 때 대개의 경우 서버는 파일 끝 플래그를 전송하지 않습니다. 필요한 경우, CONTENT_LENGTH 값을 사용하여 입력 문자열 끝을 판별할 수 있습니다. 이 변수를 추출하면 컨텐츠 서버 응답으로부터 받은 헤더값에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대해 요청한 헤더에 영향을 줍니다. 예제:

```
7034
```

PROXY_COOKIE

읽기를 할 때 변수는 기점 서버에 의해서 설정된 Set-Cookie 헤더 값을 포함합니다. 또한 요청 스트림에서 새 쿠키를 설정하는 데 사용될 수도 있습니다. 이 변수를 설정하면 복제 헤더의 존재 여부를 막론하고 문서 요청 스트림 내의 새로운 Set-Cookie 헤더를 작성하게 됩니다.

PROXY_HEADERS

읽기 전용. 프록시 헤더를 추출하는 데 사용됩니다.

PROXY_METHOD

HTTPD_proxy()를 통해 이루어진 요청에 대한 메소드. 이 변수를 추출하면 컨텐츠 서버 응답으로부터 받은 헤더값에 적용됩니다. 이를 설정하면 컨텐츠 서버에 대한 요청의 헤더에 영향을 줍니다.

QUERY_STRING

GET 메소드를 사용하여 정보를 전송할 경우, 이 변수는 물음표(?) 뒤에 나오는 정보를 포함합니다. 이 정보는 CGI 프로그램에 의해 해독되어야 합니다. 예제:

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

RCA_OWNER

읽기 전용. 요청된 오브젝트를 소유한 노드를 제공하여 수치를 리턴합니다. 이 변수는 사전 종료, 프록시 어드바이저 또는 로그 단계에서만 사용 가능하고 서버가 RCA(remote cache access)를 사용하는 캐시 배열의 일부일 때만 의미가 있습니다.

RCA_TIMEOUTS

읽기 전용. 모든 피어에 대해 RCA 요청에 시간 종료된 총계를 포함하는 수치를 리턴합니다. 모든 단계에서 이 변수를 사용할 수 있습니다.

REDIRECT_*

읽기 전용. 변수 이름에 해당하는 오류 코드의 경로 재지정 문자열을 포함합니다(예: `REDIRECT_URL`). 가능한 REDIRECT_ 변수 목록은 http://httpd.apache.org/docs-2.0/custom-error.html의 Apache 웹 서버에 대한 온라인 문서에 있습니다.

REFERRER_URL

읽기 전용. 브라우저의 마지막 URL 위치를 포함합니다. 이 변수는 서버를 위하여 Request-URL이 확보된 자원의 주소(URL)를 클라이언트가 지정할 수 있도록 합니다. 예제:

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

REMOTE_ADDR

웹 브라우저의 IP 주소가 포함됩니다. 예를 들면, `45.23.06.8`과 같습니다.

REMOTE_HOST

웹 브라우저의 호스트 이름이 포함됩니다. 예를 들면, `www.raleigh.ibm.com`과 같습니다.

REMOTE_USER

서버가 클라이언트 인증을 지원하고 스크립트가 보호되는 경우, 이 변수에는 인증을 위해 전달되는 사용자 이름이 포함됩니다. 예를 들면, `joeuser`와 같습니다.

REQHDR

읽기 전용. 클라이언트가 전송하는 헤더 목록을 포함합니다.

REQUEST_CONTENT_TYPE

읽기 전용. 요청 본문의 컨텐츠 유형을 리턴합니다. 예제:

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

REQUEST_CONTENT_LENGTH

읽기 전용. 정보가 POST 메소드를 사용하여 전송될 때, 이 변수에는 데이터의 문자 수가 포함됩니다. 표준 입력을 사용하여 정보를 전달할 때 대개의 경우 서버는 파일 끝 플래그를 전송하지 않습니다. 필요한 경우, CONTENT_LENGTH 값을 사용하여 입력 문자열 끝을 판별할 수 있습니다. 예를 들면, `7034`와 같습니다.

REQUEST_METHOD

읽기 전용. 요청을 전송하는데 사용되는 메소드(HTML 형식의 METHOD 속성을 사용하여 지정된)가 포함됩니다. 예를 들면, `GET` 또는 `POST`와 같습니다.

REQUEST_PORT

읽기 전용. URL에 지정된 포트 번호 또는 프로토콜에 따라 설정된 기본 포트를 리턴합니다.

RESPONSE_CONTENT_TYPE

읽기 전용. POST 메소드를 사용하여 정보를 전송할 경우, 이 변수에는 포함된 데이터의 유형이 들어 있습니다. 프록시 서버 구성 파일에 사용자가 직접 컨텐츠 유형을 작성하고 이를 표시기에 맵핑할 수 있습니다. 예를 들면, `text/html`과 같습니다.

RESPONSE_CONTENT_LENGTH

읽기 전용. 정보가 POST 메소드를 사용하여 전송될 때, 이 변수에는 데이터의 문자 수가 포함됩니다. 표준 입력을 사용하여 정보를 전달할 때 대개의 경우 서버는 파일 끝 플래그를 전송하지 않습니다. 필요한 경우, CONTENT_LENGTH 값을 사용하여 입력 문자열 끝을 판별할 수 있습니다. 예를 들면, `7034`와 같습니다.

RULE_FILE_PATH

읽기 전용. 구성 파일에 대한 완전한 파일 시스템 경로 및 파일 이름을 포함합니다.

SSL_SESSIONID

읽기 전용. 현재 요청이 SSL 접속 중일 때 SSL 세션 ID를 요청합니다. 현재 요청이 SSL 접속 중 수신되지 않았다면 NULL이 리턴됩니다.

SCRIPT_NAME

요청의 URL을 포함합니다.

SERVER_ADDR

읽기 전용. 프록시 서버의 로컬 IP 주소를 포함합니다.

SERVER_NAME

읽기 전용. 이 요청에 대한 컨텐츠 서버의 프록시 서버 호스트 이름 또는 IP 주소가 포함됩니다. 예를 들면, `www.ibm.com`과 같습니다.

SERVER_PORT

읽기 전용. 클라이언트 요청이 전송되는 프록시 서버의 포트 수가 포함됩니다. 예를 들면, `80`과 같습니다.

SERVER_PROTOCOL

읽기 전용. 요청을 하는데 사용되는 프로토콜의 이름 및 버전이 포함됩니다. 예를 들면, `HTTP/1.1`과 같습니다.

SERVER_ROOT

읽기 전용. 프록시 서버 프로그램이 설치된 디렉토리를 포함합니다.

SERVER_SOFTWARE

읽기 전용. 프록시 서버의 이름 및 버전이 들어있습니다.

STATUS

HTTP 응답 코드 및 이유 문자열이 들어 있습니다. 예를 들면, 200 OK와 같습니다.

TRACE

추적될 정보의 양을 결정합니다. 다음 리턴값이 사용됩니다.

- OFF - 추적 안함.
- V - 상세 모드.
- VV - 매우 상세한 모드.
- MTV - 최대 상세한 모드.

URI 읽기/쓰기. DOCUMENT_URL과 같습니다.

URI_PATH

읽기 전용. URL만을 위한 경로 부분을 리턴합니다.

URL 읽기/쓰기. DOCUMENT_URL과 같습니다.

URL_MD4

읽기 전용. 현재 요청에 대한 잠재적 캐시 파일의 파일 이름을 리턴합니다.

USE_PROXY

현재 요청에 대해 연결할 프록시를 식별합니다. URL을 지정하십시오. 예를 들면, `http://myproxy:8080`과 같습니다.

USERID

REMOTE_USER와 같습니다.

USERNAME

REMOTE_USER와 같습니다.

인증 및 권한 부여

먼저, 용어에 대해 간략히 살펴보기로 하겠습니다.

인증 요청자의 신원을 확인하기 위한 해당 요청과 연관되는 보안 토큰의 검증

권한 부여

요청자가 해당 자원에 액세스할 수 있는지 여부를 판별하기 위해 보안 토큰을 사용하는 프로세스

41 페이지의 그림 3에서는 프록시 서버의 인증 및 권한 부여 프로세스를 나타냅니다.

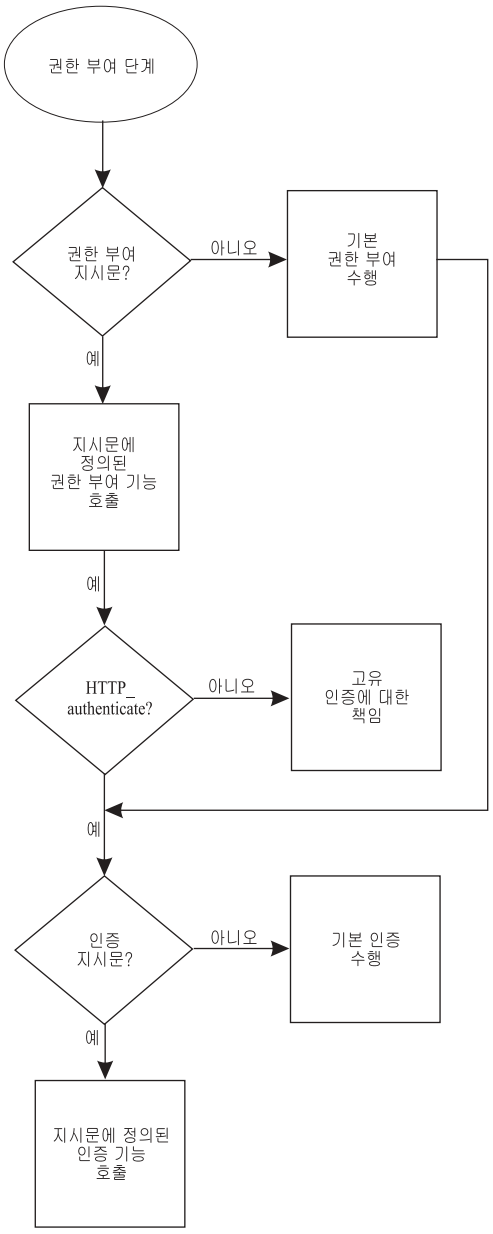

그림 3. 프록시 서버 인증 및 권한 부여 프로세스

그림 3에 표시된 것처럼 인증 프로세스 *초기화*는 서버의 인증 및 권한 부여 프로세스 중 첫 단계입니다.

Caching Proxy에서 인증은 권한 부여 프로세스의 일부입니다. 이는 인증이 필수일 경우에만 실행됩니다.

인증 및 권한 부여 프로세스

프록시 서버는 권한 부여가 필요한 요청을 처리할 때 이들 단계를 따릅니다.

1. 먼저, 프록시 서버는 구성 파일을 검사하여 인증 지시문이 포함되어 있는지 여부를 판별합니다.

- 인증 지시문이 구성 파일에 있으면 서버는 지시문에 정의된 인증 함수를 호출하고 2단계에서 인증을 시작합니다.
- 인증 지시문이 없으면 서버는 기본 인증을 수행한 후 바로 3단계에서 인증 과정에 들어갑니다.

2. 프록시 서버는 HTTP_authenticate 헤더가 클라이언트 요청에 들어 있는지 확인함으로써 인증 프로세스를 시작합니다.
   - 헤더가 있으면 서버는 인증 프로세스를 계속 진행합니다(3단계 참조).
   - 헤더가 없으면 인증은 다른 방법을 통해 수행되어야 합니다.
3. 프록시 서버는 프록시 구성 파일 안에 인증 지시문이 있는지 여부를 확인합니다.
   - 구성 파일 안에 인증 지시문이 있다면 서버는 지시문 안에 정의된 인증 함수를 호출합니다.
   - 지시문이 없다면 서버는 기본 인증을 수행합니다.

Caching Proxy 플러그인이 고유 권한 부여 프로세스를 제공할 경우, 이는 기본 서버 권한 부여 및 인증을 *덮어쓰기합니다*. 그러므로 사용자 구성 파일 안에 권한 부여 지시문이 있으면 이들과 연관되는 플러그인 함수 또한 필요한 모든 인증을 처리해야 합니다. 사전 정의된 HTTPD_authenticate() 함수가 사용하도록 제공됩니다.

사용자의 권한 부여 플러그인에 인증을 제공할 수 있는 방법에는 다음 세 가지가 있습니다.

- 사용자 자신의 별도 인증 및 권한 부여 플러그인 함수를 작성하십시오. 프록시 구성 파일에서 권한 부여 및 인증 지시문을 둘 다 사용하여 이 함수를 지정하십시오. 권한 부여 플러그인 함수에 HTTPD_authenticate() 함수 호출을 포함시켰는지 확인하십시오.

  권한 부여 단계가 실행되면 권한 부여 플러그인 함수가 수행되고 그에 따라 인증 플러그인 함수가 호출됩니다.
- 사용자 자신의 권한 부여 플러그인 함수를 작성하되, 이 함수가 기본 서버 인증을 호출하도록 하십시오. 사용자의 프록시 구성 파일에서 권한 부여 지시문을 사용하여 함수를 지정하십시오. 이 경우, 인증 지시문은 필요하지 않습니다. 권한 부여 플러그인 함수에서 HTTPD_authenticate() 함수를 호출했는지 확인하십시오.

  권한 부여 단계가 실행되면 권한 부여 플러그인 함수가 수행되고 그에 따라 기본 서버 인증이 호출됩니다.
- 사용자 자신의 권한 부여 플러그인 함수를 작성하고 필요한 모든 인증 처리를 포함시키십시오. 권한 부여 플러그인에 HTTPD_authenticate() 함수를 사용하지 마십시오. 사용자의 프록시 구성 파일에서 권한 부여 지시문을 사용하여 권한 부여 플러그인을 지정하십시오. 이 경우, 인증 지시문은 필요하지 않습니다.

권한 부여 단계가 실행되면 사용자 권한 부여 플러그인 함수가 수행되고 그에 따라 거기에 포함된 인증이 수행됩니다.

Caching Proxy 플러그인이 고유의 권한 부여 프로세스를 제공하지 않는 경우, 다음 메소드를 사용하여 사용자 정의 인증을 제공할 수 있습니다.

- 사용자 자신의 인증 플러그인 함수를 작성하십시오. 사용자의 프록시 구성 파일에서 권한 부여 지시문을 사용하여 함수를 지정하십시오. 이 경우, 권한 부여 지시문은 필요하지 않습니다.

권한 부여 단계가 실행되면 기본 서버 권한 부여가 수행되고 그에 따라 인증 플러그인 함수가 호출됩니다.

다음 사항을 기억하십시오.

- 사용자 구성 파일 안에 권한 부여 지시문이 없거나 지정한 플러그인 함수가 HTTP_NOACTION을 리턴하여 요청 처리를 거부할 경우, 서버의 기본 인증이 발생합니다.
- 사용자 구성 파일에 권한 부여 지시문이 있고 플러그인 함수에 HTTPD_authenticate()가 있는 경우, 인증 지시문에 지정된 대로 인증 함수를 호출합니다. 사용자에게 정의된 인증 정의문이 없거나 지정한 플러그인 함수가 HTTP_NOACTION을 리턴하여 요청 처리를 거부할 경우, 기본 인증이 발생합니다.
- 사용자의 구성 파일에 권한 부여 지시문이 있지만 해당 플러그인 함수에 HTTPD_authenticate()가 없을 경우, 인증 함수는 서버에 의해 호출되지 않습니다. 사용자는 권한 부여 플러그인 함수의 일부로서 인증 처리를 작성하거나 다른 인증 모듈을 직접 호출해야 합니다.
- 권한 부여 함수에서 사용자가 401 코드 또는 407 코드를 리턴하는 경우, Caching Proxy는 챌린지(사용자 ID 및 암호의 리턴을 묻는 프롬프트를 브라우저에 표시)를 자동으로 생성합니다. 그러나 이 조치가 올바르게 발생할 수 있도록 Caching Proxy에 보호 설정을 구성해야 합니다.

변형 캐싱

변형 캐싱을 사용하여 원래 문서(URI)의 수정된 형식으로 되어 있는 데이터를 캐시할 수 있습니다. Caching Proxy는 API를 통해 생성된 변형을 처리합니다. 변형은 기본 문서의 다른 버전을 말합니다.

일반적으로, 기점 서버가 변형을 보낼 경우, 그 서버는 그 변형을 식별할 수 없습니다. Caching Proxy는 플러그인에서 작성한 변형만을 지원합니다(예: 코드 페이지 변환).플러그인이 HTTP 헤더에 없는 기준에 따라 변형을 작성하는 경우, Caching Proxy가 기존 변형을 정확히 식별할 수 있도록 PreExit 또는 PostAuthorization을 포함시켜 의사 헤더(psuedoheader)를 작성해야 합니다.

예를 들어, 변형자 API 프로그램을 사용하여 브라우저가 전송한 User-Agent 헤더 값을 기준으로 사용자들이 요청한 데이터를 변경하십시오. *close* 함수에서 수정된 내용을 파일에 저장하거나 버퍼 길이를 지정한 다음 버퍼를 데이터 인수로 전달하십시오. 그런 다음 variant caching 함수 httpd_variant_insert() 및 httpd_variant_lookup()을 사용하여 내용을 캐시에 기록하십시오.

API 예제

Caching Proxy API 함수 시작하기에 대한 도움을 받으려면 Edge Components 설치 CD-ROM의 samples 디렉토리에 있는 샘플 프로그램을 보십시오. 추가 정보는 WebSphere Application Server 웹 사이트인 www.ibm.com/software/webservers/appserv/에서 찾아볼 수 있습니다.

제 3 장 사용자 정의 어드바이저

제 3 장에서는 Load Balancer에 대해 사용자 정의 어드바이저 작성을 설명합니다.

어드바이저 제공 로드 밸런스 정보

어드바이저는 Load Balancer 내에서 작업하면서 주어진 서버에 대한 로드 정보를 제공하는 소프트웨어 에이전트입니다. 표준 프로토콜(HTTP, SSL 및 기타)에 대하여 각각 다른 어드바이저가 존재합니다. Load Balancer 기본 코드는 주기적으로 어드바이저 주기를 수행하며, 이 주기를 통해 구성에 있는 모든 서버의 상태가 개별적으로 평가됩니다.

Load Balancer의 고유 어드바이저를 작성하여 서버 시스템의 로드를 판별하는 방법을 사용자 정의할 수 있습니다.

Windows 시스템의 경우: IPv4용 및 IPv6용 Load Balancer 설치를 사용하는 경우 시스템에서 IPv6 프로토콜을 사용하고 어드바이저를 사용하려고 하면 C:\windows\system32\drivers\etc\ 디렉토리에 있는 **protocol** 파일을 수정해야 합니다.

IPv6의 경우, protocol 파일에 다음 행을 삽입하십시오.

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

표준 어드바이저 기능

일반적으로 어드바이저는 다음과 같은 방법으로 로드 밸런스를 가능하게 합니다.

1. 어드바이저는 주기적으로 각 서버와의 연결을 열고 서버에 요청 메시지를 전송합니다. 메시지의 내용은 서버에서 실행 중인 프로토콜에 대해 고유합니다. 예를 들어, HTTP 어드바이저는 HEAD 요청을 서버로 전송합니다.
2. 어드바이저는 서버의 응답을 인식합니다. 응답을 받은 후, 서버에 대한 로드값을 계산하여 보고합니다. 어드바이저마다 다른 방법으로 로드값을 계산하지만, 대부분의 표준 어드바이저는 서버가 응답하는 데 걸린 시간을 측정하여 그 값을 로드값으로 보고합니다(밀리초 단위).
3. 어드바이저는 Load Balancer의 관리 프로그램 기능으로 로드를 보고합니다. 관리 프로그램 보고서의 포트 컬럼에 로드가 나타납니다. 관리 프로그램은 어드바이저가 보고한 로드값과 관리자가 설정한 가중치를 사용하여 서버에서 수신하는 요청의 로드 밸런스 방법을 판별합니다.
4. 서버가 응답하지 않는 경우, 어드바이저는 로드값으로 음수(-1)를 리턴합니다. 관리 프로그램은 이 정보를 사용하여 특정 서버에 대한 서비스의 일시중단 시기를 판별합니다.

Load Balancer와 함께 제공된 표준 어드바이저에는 다음과 같은 기능이 있는 어드바이저가 들어 있습니다. 어드바이저에 관한 자세한 정보는 *WebSphere Application Server Load Balancer 관리 안내서*를 참조하십시오.

- Connect
- DB2
- DNS
- FTP
- HTTP
- HTTPS
- IMAP
- LDAP
- NNTP
- Ping
- POP3
- Reach
- Self
- SIP
- SMTP
- SSL
- Telnet
- WebSphere Application Server
- WebSphere Application Server Caching Proxy
- Workload Manager

표준 어드바이저가 제공되지 않는 독점 프로토콜을 지원하려면 사용자 정의 어드바이저를 작성해야 합니다.

사용자 정의 어드바이저 작성

사용자 정의 어드바이저는 클래스 파일로 제공되는 짧은 Java 코드로, Load Balancer 기본 코드가 서버의 로드를 판별하기 위해 호출합니다. 기본 코드는 사용자 정의 어드바이저의 인스턴스 시작 및 정지, 상태 및 보고서 제공, 로그 파일에 히스토리 정보 기록, 관리 프로그램 컴포넌트에 어드바이저 결과 보고 등과 같은 필요한 모든 관리 서비스를 제공합니다.

Load Balancer 기본 코드에서 사용자 정의 어드바이저를 호출할 때, 다음 단계가 진행됩니다.

1. Load Balancer 기본 코드를 서버 시스템과 연결합니다.
2. 소켓이 열리면 기본 코드가 지정된 어드바이저의 GetLoad 기능을 호출합니다.
3. 어드바이저의 GetLoad 기능은 서버의 응답 대기를 비롯하여 사용자가 서버의 상태를 평가하기 위하여 정의한 단계들을 수행합니다. 응답을 받으면 기능은 실행을 종료합니다.
4. Load Balancer 기본 코드에서 서버가 있는 소켓을 닫고 관리 프로그램에 로드 정보를 보고합니다. 사용자 정의 어드바이저가 정상 모드 또는 수정 모드에서 작동하는지에 따라, GetLoad 기능이 종료된 후 기본 코드는 수시로 추가 계산을 수행합니다.

정상 모드와 수정 모드

사용자 정의 어드바이저를 *정상* 모드 또는 *수정* 모드에서 Load Balancer와 작동하도록 설계할 수 있습니다.

작동 모드의 선택은 사용자 정의 어드바이저 파일에서 constructor 메소드의 매개변수로서 지정됩니다. (각 어드바이저는 설계에 따라 이들 모드 중 하나만 운영합니다.)

정상 모드에서 사용자 정의 어드바이저는 서버와 데이터를 교환하고, 기본 어드바이저 코드는 교환 시간을 측정하여 로드값을 산출합니다. 기본 코드는 계산된 로드값을 관리 프로그램에 보고합니다. 사용자 정의 어드바이저는 성공을 표시하는 0값 또는 오류를 표시하는 1값을 리턴합니다.

정상 모드를 지정하려면, constructor의 replace 플래그를 *false*로 설정하십시오.

수정 모드에서 기본 코드는 시간 측정은 수행하지 않습니다. 사용자 정의 어드바이저 코드는 지정된 모든 조작을 해당되는 고유한 요구사항에 기초하여 수행한 다음, 실제 로드 번호를 리턴합니다. 기본 코드는 로드 번호를 승인하고 그대로 관리 프로그램에 보고합니다. 가장 좋은 결과를 위해서는 10부터 1000까지(빠른 서버는 10, 느린 서버는 1000) 로드 번호를 정규화하십시오.

수정 모드를 지정하려면, constructor의 replace 플래그를 *true*로 설정하십시오.

어드바이저 이름 지정 규칙

사용자 정의 어드바이저 파일 이름은 ADV_*name*.java 형태이며, 여기서 *name*은 사용자가 선택하는 어드바이저 이름입니다. 완료된 이름은 대문자 접두부 ADV_로 시작하고 나머지는 모두 소문자여야 합니다. 소문자로 제한하면 어드바이저를 실행하는 명령에서 대소문자를 구분하지 않아도 됩니다.

Java 규약에 따라 파일 내에 정의된 클래스 이름은 파일의 이름과 일치해야 합니다.

컴파일

사용자 정의 어드바이저는 Java 언어로 작성되어야 하며 개발 시스템에 설치된 Java 컴파일러로 컴파일되어야 합니다. 컴파일 시 다음의 파일을 참조합니다.

- 사용자 정의 어드바이저 파일
- *install_path*/servers/lib 디렉토리에 있는 기본 클래스 파일(ibmnd.jar)

컴파일 동안 classpath 환경 변수는 사용자 정의 어드바이저 파일과 기본 클래스 파일을 모두 가리켜야 합니다. 컴파일 명령은 다음 형식을 따릅니다.

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

이 예제에서는 기본 Linux 및 UNIX 설치 경로를 사용합니다. 어드바이저 파일 이름은 ADV_*name*.java이고 어드바이저 파일은 현재 디렉토리에 저장됩니다.

컴파일 출력은 클래스 파일입니다(예: ADV_*name*.class). 어드바이저를 시작하기 전에 클래스 파일을 *install_path*/servers/lib/CustomAdvisors/ 디렉토리로 복사하십시오.

주: 임의의 운영 체제에서 컴파일된 사용자 정의 어드바이저를 다른 운영 체제에서 실행할 수도 있습니다. 예를 들어, 어드바이저를 Windows 시스템에서 컴파일한 다음, 결과 클래스 파일(2진 형식)을 Linux 시스템으로 복사하여 사용자 정의 어드바이저를 실행할 수 있습니다.

사용자 정의 어드바이저 실행

사용자 정의 어드바이저를 실행하려면, 먼저 어드바이저의 클래스 파일을 Load Balancer 시스템의 lib/CustomAdvisors/ 하위 디렉토리로 복사해야 합니다. 예를 들어, myping이란 사용자 정의 어드바이저의 파일 경로는 *install_path*/servers/lib/CustomAdvisors/ADV_myping.class입니다.

Load Balancer를 구성하고, 해당되는 관리 프로그램 기능을 시작하고, 사용자 정의 어드바이저를 시작하는 명령을 발행하십시오. 사용자 정의 어드바이저는 ADV_ prefix 및 파일 확장자를 제외한 이름으로 지정됩니다.

```
dscontrol advisor start myping port_number
```

명령에서 지정한 포트 번호는 어드바이저가 대상 서버와 연결하는 포트입니다.

필수 루틴

다른 어드바이저들과 마찬가지로, 사용자 정의 어드바이저는 ADV_Base라는 어드바이저 기본 클래스 기능을 확장합니다. 어드바이저 기본은 대부분의 어드바이저 기능을 수행합니다. 예를 들어, 관리 프로그램의 가중치 알고리즘에 사용될 수 있도록 로드를 다시 관리 프로그램에 보고하는 기능을 수행합니다. 또한 소켓 연결 및 닫기 조작을 수행하며, 어드바이저에 사용될 send 및 receive 메소드도 제공합니다. 검토 중인 서버용으로 지정된 포트에서 데이터를 전송하거나 수신하는 데 어드바이저가 사용됩니다. 어드

바이저 기본 내의 TCP 메소드는 로드를 계산하기 위해 시간을 측정합니다. 어드바이저 기본 생성자 내의 플래그가 기존 로드 위에 어드바이저로부터 리턴되는 새로운 로드를 겹쳐쓰도록 할 수도 있습니다.

주: 생성자에 설정된 값에 따라서 어드바이저 기본은 가중치 알고리즘에 로드를 지정된 간격으로 제공합니다. 어드바이저가 처리를 완료하지 않아서 유효한 로드를 리턴할 수 없는 경우, 어드바이저 기본은 이전에 보고된 로드를 사용합니다.

어드바이저에는 다음의 기본 클래스 메소드가 있습니다.

- constructor 루틴. constructor는 기본 클래스 생성자를 호출합니다.
- ADV_AdvisorInitialize 메소드. 기본 클래스의 초기화 완료 후 추가 단계를 수행하는 방법을 제공합니다.
- getLoad 루틴. 기본 어드바이저 클래스가 소켓 열기를 수행합니다. getLoad 함수는 권고 주기를 완료하기 위해 적절한 송신 및 수신 요청을 발행해야 합니다.

필수 루틴에 대한 세부사항은 이 섹션의 뒷부분에서 설명합니다.

탐색 순서

사용자 정의 어드바이저는 원시 또는 표준 어드바이저를 탐색한 후에 호출됩니다. Load Balancer가 표준 어드바이저 목록에서 지정된 어드바이저를 찾지 못하면 사용자 정의 어드바이저 목록을 참조합니다. 어드바이저에 대한 자세한 정보는 *WebSphere Application Server Load Balancer 관리 안내서*를 참조하십시오.

이름 지정 및 파일 경로

사용자 정의 어드바이저의 이름과 경로에 다음의 요구사항이 적용됩니다.

- 사용자 정의 어드바이저의 이름은 소문자(영문자)로 지정되어, 작업자가 명령행에 명령을 입력할 때 대소문자를 구분하지 않아도 됩니다. 어드바이저 이름은 ADV_ 접두어로 시작합니다.
- 사용자 정의 어드바이저 클래스는 lib/CustomAdvisors 서브디렉토리 내에 위치해야 합니다. 이 디렉토리의 기본 위치는 Linux 및 UNIX 시스템의 경우 /opt/ibm/edge/lb/servers/lib/CustomAdvisors이고, Windows 시스템의 경우 C:\Program Files\IBM\edge\lb\
  servers\lib\CustomAdvisors\입니다.

사용자 정의 어드바이저 메소드 및 함수 호출

생성자(어드바이저 기본에서 제공)

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

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

sName

사용자 정의 어드바이저 이름.

sVersion

사용자 정의 어드바이저 버전.

iDefaultPort

호출에서 포트 번호를 지정하지 않은 경우 서버와 접속할 포트 번호.

iInterval

어드바이저가 서버를 조회하는 간격.

sDefaultLogFileName

이 매개변수는 필수지만 사용되지 않습니다. 유일하게 허용되는 값은 널(Null) 문자열, ""입니다.

replace

어드바이저 함수가 *수정* 모드인지 여부. 가능한 값은 다음과 같습니다.

- true – 어드바이저 기본 코드가 계산한 로드를 사용자 정의 어드바이저가 보고한 값으로 바꿉니다.
- false – 사용자 정의 어드바이저가 보고한 로드 값을 어드바이저 기본 코드가 계산한 로드 값에 더합니다.

ADV_AdvisorInitialize()

```
void  ADV_AdvisorInitialize()
```

이 메소드는 사용자 정의 어드바이저에 필요한 초기화를 수행하기 위해 제공됩니다. 이 메소드는 어드바이저 기본 모듈이 시작한 후에 호출됩니다.

표준 어드바이저를 포함한 대부분의 경우, 이 메소드는 사용되지 않으며 코드가 *return* 문으로만 구성됩니다. 이 메소드에서만 유효한 suppressBaseOpeningSocket 메소드를 호출하는 데 사용될 수 있습니다.

getLoad()

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

iConnectTime

접속이 완료되는 데 걸린 시간(밀리초). 이 로드 측정은 어드바이저 기본 코드에 의

해 수행되어 사용자 정의 어드바이저 코드로 전달되는데, 이는 로드 값을 리턴할 때 측정을 사용하거나 무시할 수 있습니다. 접속을 할 수 없을 경우 이 값을 -1로 지정하십시오.

caller

어드바이저 기본 메소드가 제공되는 어드바이저 기본 클래스의 인스턴스.

사용자 정의 어드바이저에 사용 가능한 함수 호출

다음 섹션에서 설명할 메소드 또는 함수는 사용자 정의 어드바이저에서 호출할 수 있습니다. 어드바이저 기본 코드는 이들 메소드를 지원합니다.

일부 함수 호출은 직접 호출될 수 있지만(예: *function_name*()), 나머지 호출은 caller 접두부가 필요합니다. *caller*는 실행 중인 사용자 정의 어드바이저를 지원하는 기본 어드바이저 인스턴스를 나타냅니다.

ADVLOG()

ADVLOG 함수를 사용하여 사용자 정의 어드바이저는 텍스트 메시지를 어드바이저 기본 로그 파일에 기록합니다. 형식은 다음과 같습니다.

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

logLevel

메시지가 로그 파일에 기록되는 상태 레벨. 어드바이저 로그 파일은 여러 단계로 구성됩니다. 가장 긴급한 메시지의 상태 레벨은 0이며 긴급하지 않은 메시지일수록 더 높은 번호의 레벨이 주어집니다. 가장 자세한 유형의 메시지의 상태 레벨은 5입니다. 레벨에 의해 사용자가 실시간으로 받는 메시지 유형을 제어합니다(**dscontrol** 명령으로 자세한 정도를 설정). 심각한 오류는 항상 레벨 0으로 기록되어야 합니다.

message

로그 파일에 기록될 메시지. 이 매개변수의 값은 표준 Java 문자열입니다.

getAdvisorName()

getAdvisorName 함수는 사용자 정의 어드바이저 이름의 접미부가 있는 Java 문자열을 리턴합니다. 예를 들어, ADV_cdload.java로 이름 지정된 어드바이저의 경우, 이 함수는 cdload 값을 리턴합니다.

이 함수에는 매개변수가 없습니다.

어드바이저의 한 인스턴스 내에서 이 값은 변경될 수 없습니다.

getAdviseOnPort()

getAdviseOnPort 함수는 호출 중인 사용자 정의 어드바이저가 실행되고 있는 포트 번호를 리턴합니다. 리턴값은 Java 정수(int)이며 함수에는 매개변수가 없습니다.

어드바이저의 한 인스턴스 내에서 이 값은 변경될 수 없습니다.

caller.getCurrentServer()

getCurrentServer 함수는 현재 서버의 IP 주소를 리턴합니다. 리턴값은 IP 주소 형식의 Java 문자열입니다(예: 128.0.72.139).

일반적으로, 어드바이저 기본 코드가 차례로 모든 서버 시스템을 조회하므로, 이 주소는 사용자가 사용자 정의 어드바이저를 호출할 때마다 변경됩니다.

이 함수에는 매개변수가 없습니다.

caller.getCurrentCluster()

getCurrentCluster 함수 호출은 현재 서버 클러스터의 IP 주소를 리턴합니다. 리턴값은 IP 주소 형식의 Java 문자열입니다(예: 128.0.72.139).

일반적으로, 어드바이저 기본 코드가 차례로 모든 서버 클러스터를 조회하므로, 이 주소는 사용자가 사용자 정의 어드바이저를 호출할 때마다 변경됩니다.

이 함수에는 매개변수가 없습니다.

getInterval()

getInterval 함수는 어드바이저 간격 즉, 어드바이저 주기 사이의 시간(초)을 리턴합니다. 이 값은 **dscontrol** 명령을 사용하여 런타임 시 값이 수정되지 않았으면 사용자 정의 어드바이저의 생성자에 설정된 기본값과 동일합니다.

리턴값은 Java 정수(int)입니다. 이 함수에는 매개변수가 없습니다.

caller.getLatestLoad()

getLatestLoad 함수는 사용자 정의 어드바이저가 주어진 서버 오브젝트의 최근 로드 값을 얻을 수 있습니다. 로드 값은 어드바이저 기본 코드 및 관리자 디먼에 의해 내부 테이블에 유지됩니다.

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

세 개의 인수가 함께 하나의 서버 오브젝트에 정의됩니다.

cluster_IP
현재 로드 값을 얻기 위한 서버 오브젝트의 클러스터 IP 주소. 인수는 IP 주소 형식의 Java 문자열이어야 합니다(예: 245.145.62.81).

port
현재 로드 값을 얻기 위한 서버 오브젝트 서버 오브젝트의 포트 번호.

server_IP
현재 로드 값을 얻기 위한 서버 오브젝트의 IP 주소. 인수는 IP 주소 형식의 Java 문자열이어야 합니다(예: 192.255.201.3).

리턴값은 정수입니다.

- 양의 리턴 값을 조회된 오브젝트에 할당된 실제 로드 값을 나타냅니다.
- -1은 요청한 서버가 중단되었음을 표시합니다.
- -2 값은 요청한 서버의 상태를 알 수 없음을 표시합니다.

이 함수 호출은 임의 프로토콜이나 포트가 다른 프로토콜이나 포트와 독립적으로 작동되도록 할 경우 유용합니다. 예를 들어, 동일한 시스템의 텔넷 서버가 사용 불가능한 경우, 특정 Application Server를 사용 불가능하게 한 사용자 정의 어드바이저에서 이 함수 호출을 사용할 수 있습니다.

caller.receive()

receive 함수는 소켓 접속에서 정보를 가져옵니다.

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

response 매개변수는 검색된 데이터가 위치할 문자열 버퍼입니다. 또한 함수는 다음과 같은 의미를 갖는 정수 값을 리턴합니다.

- 0은 데이터가 전송되었음을 나타냅니다.
- 음수는 오류를 표시합니다.

caller.send()

send 함수는 지정된 포트를 사용하여, 구축된 소켓 접속을 통해 데이터 패킷을 서버로 전송합니다.

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

command 매개변수는 서버에 전송할 데이터를 포함한 문자열입니다. 이 함수는 다음과 같은 의미를 갖는 정수 값을 리턴합니다.

- 0은 데이터가 전송되었음을 나타냅니다.
- 음수는 오류를 표시합니다.

suppressBaseOpeningSocket()

suppressBaseOpeningSocket 함수 호출을 사용하여 사용자 정의 어드바이저는 기본 어드바이저 코드가 사용자 정의 어드바이저를 대신해서 서버에 대해 TCP 소켓을 열 것인지 여부를 지정합니다. 어드바이저가 서버 상태를 판별하기 위해 서버와 직접 통신하지 않는 경우, 이 소켓을 열 필요는 없습니다.

이 함수 호출은 한 번만 발행될 수 있으며 ADV_AdvisorInitialize 루틴에서 발행되어야 합니다.

이 함수에는 매개변수가 없습니다.

예제

다음 예제에서는 사용자 정의 어드바이저가 구현되는 방법을 보여줍니다.

표준 어드바이저

이 소스 코드 샘플은 표준 Load Balancer HTTP 어드바이저와 유사합니다. 함수는 다음과 같습니다.

1. 전송 요청 "HEAD/HTTP" 명령이 발행됩니다.
2. 응답을 받았습니다. 정보는 구문 분석될 수 없으나 응답은 getLoad 메소드를 종료시켰습니다.
3. getLoad 메소드는 성공을 표시하기 위해서 0 또는 실패를 표시하기 위해서 -1을 리턴시켰습니다.

이 어드바이저는 정상 모드에서 작동하므로 로드 측정은 소켓의 열기, 전송, 수신 및 닫기 작업을 수행하는 데 경과한 시간(밀리초)에 기반합니다.

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

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

  public ADV_sample() {
    super(ADV_NAME, "3.0.0.0-03.31.00",
          ADV_DEF_ADV_ON_PORT, ADV_DEF_INTERVAL, "",
          false);
    super.setAdvisor( this );
  }

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

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

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

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

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

```
      iRc = caller.receive(sbReceiveData);   // receive the result

      // parse the result here if you need to

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

사이드 스트림 어드바이저

이 예제는 어드바이저 기본에 의해 열린 표준 소켓을 억제하는 경우를 설명합니다. 대신, 이 어드바이저는 사이드 스트림 Java 소켓을 열어 서버를 조회합니다. 이 프로시저는 서버가 어드바이저 조회를 인식하기 위해 정상 클라이언트 통신량에서 다른 포트를 사용하는 경우 유용합니다.

이 예제에서 서버는 포트 11999를 인식 중이며, 16진 정수 ″4″가 조회되면 로드 값을 리턴합니다. 이 예제는 바꾸기 모드에서 실행됩니다. 즉, 어드바이저 생성자의 마지막 매개변수가 true로 설정되고 어드바이저 기본 코드가 경과된 시간을 사용하지 않고 리턴된 로드 값을 사용합니다.

초기화 루틴에서 supressBaseOpeningSocket()에 대한 호출에 유의하십시오. 전송되는 데이터가 없는 경우 기본 소켓을 억제하지 않아도 됩니다. 예를 들어 어드바이저를 서버에 연결하기 위해 소켓을 열고자 할 경우가 있습니다. 이를 선택하기 전에 응용프로그램에서 이러한 선택이 필요한지 주의해서 검사하십시오.

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

public class ADV_sidea extends ADV_Base implements ADV_MethodInterface {
  static final String ADV_NAME = "sidea";
  static final int ADV_DEF_ADV_ON_PORT = 12345;
  static final int ADV_DEF_INTERVAL = 7;

  // create an array of bytes with the load request message
  static final byte[] abHealth = {(byte)0x00, (byte)0x00, (byte)0x00, (byte)0x04};
  public ADV_sidea() {
    super(ADV_NAME, "3.0.0.0-03.31.00", ADV_DEF_ADV_ON_PORT,
        ADV_DEF_INTERVAL, "",
        true);         // replace mode parameter is true
    super.setAdvisor( this );
  }

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

  public void ADV_AdvisorInitialize()
  {
```

```
    suppressBaseOpeningSocket();   // tell base code not to open the
                                   // standard socket
        return;
  }

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

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

    string sServer = caller.getCurrentServer();    // address of server to query
      try {
      socket soServer = new Socket(sServer, iControlPort);  // open socket to
                                                            // server
      DataInputStream disServer = new DataInputStream(soServer.getInputStream());
      DataOutputStream dosServer = new DataOutputStream(soServer.getOutputStream());
      int iRecvTimeout = 10000;  // set timeout (in milliseconds)
                                 // for receiving data
      soServer.setSoTimeout(iRecvTimeout);

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

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

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

포트가 두 개인 어드바이저

이 사용자 정의 어드바이저 예제에서는 서버의 한 포트 자체의 상태와, 동일한 서버 시스템의 다른 포트에서 실행 중인 다른 서버 디먼의 상태에 기반하여 서버의 한 포트에 대한 장애를 발견하는 기능을 보여줍니다. 예를 들어, 포트 80의 HTTP 디먼이 응답을 정지하는 경우, 포트 443의 SSL 디먼에 대한 통신량 경로 지정을 정지할 수도 있습니다.

이 어드바이저는 응답을 전송하지 않는 서버의 기능을 정지했다고 간주하여 종료로 표시한다는 점에서, 표준 어드바이저보다 더 적극적입니다. 표준 어드바이저는 반응이 없는 서버를 속도가 매우 느리다고 간주합니다. 이 어드바이저는 HTTP 및 SSL 포트 중 한 포트에서 응답이 없으면 모두에 대해 서버를 종료로 표시합니다.

이러한 사용자 정의 어드바이저를 사용하려면 관리자는 하나는 HTTP 포트에서, 하나는 SSL 포트에서 모두 두 개의 어드바이저 인스턴스를 시작합니다. 어드바이저는 하나는 HTTP용으로, 하나는 SSL용으로 모두 두 개의 정적 글로벌 해시 테이블을 인스턴스화합니다. 각 어드바이저는 해당 서버 디먼과의 통신을 시도하고, 이 이벤트의 결과를 해시 테이블에 저장합니다. 각 어드바이저가 기본 어드바이저 클래스에 리턴하는 값은 각자의 자체 서버 디먼과 통신할 수 있는 기능과 상대 어드바이저가 자체 디먼과 통신할 수 있는 기능에 따라 다릅니다.

다음 조정 메소드를 사용할 수 있습니다.

- ADV_nte()는 서버에 대한 정보가 들어 있는 간단한 컨테이너 오브젝트입니다. 이 오브젝트는 해시 테이블에 테이블 요소로 저장됩니다. 각 오브젝트에는 요소가 현재 사용 가능한지를 판별하는 데 사용되는 시간 소인이 있습니다.
- putNte() 및 getNte()는 두 개의 어드바이저 인스턴스가 제어된 방식으로 해시 테이블을 액세스하도록 하는 동기화 메소드입니다.
- getLoadHTTP는 HTTP 서버의 응답 상태를 조회하는 메소드입니다. 이 메소드는 하위 레벨 루틴으로서 SSL에 대한 정보를 집적하거나 사용하지 않습니다.
- getLoadSSL()은 SSL 서버의 응답을 조회하는 메소드입니다. 이 메소드는 하위 레벨 루틴으로서 HTTP에 대한 정보를 집적하거나 사용하지 않습니다.
- getLoad()는 이 사용자 정의 어드바이저에 대한 진입점 루틴입니다. 이 메소드는 두 가지 프로토콜을 모두 처리하고 해시 테이블에서 정보를 저장하고 페치할 수 있습니다. 이 메소드는 두 개의 포트를 연결하는 루틴입니다.

다음 오류 조건이 발견됩니다.

- 서버 시스템의 응답이 없습니다. — 기본 어드바이저 클래스가 서버 주소에 정기적으로 ping 신호를 전송합니다. 신호가 주소에 전송되지 않으면 기본 어드바이저 클래스는 서버를 종료로 표시합니다. 두 개의 사용자 정의 어드바이저 인스턴스 중 어느 것도 호출되지 않으며, 시스템에 있는 두 개의 서버가 모두 종료로 표시됩니다.
- 서버 시스템의 디먼 하나가 응답이 없으나, 다른 하나는 작동합니다. — 기본 코드가 서버에서 소켓을 열려고 시도하면 접속이 거부되며 이 프로토콜에 대한 기본 어드바이저가 서버를 종료로 표시합니다. 해당 프로토콜에 대한 사용자 정의 어드바이저 코드가 호출되지 않습니다. 다른 프로토콜에 대한 사용자 정의 어드바이저가 자신의 서버와 계속 통신을 하더라도, 다른 사용자 정의 어드바이저가 자신의 서버 디먼과 통신할 수 없다는 것을 해시 테이블을 통해 알게 됩니다. 따라서 두 번째 프로토콜의 어드바이저도 자신의 서버를 종료로 표시합니다.
- 하나의 디먼은 응답을 전송하지 못하며, 다른 하나의 디먼은 응답을 전송합니다. — 응답하지 못하는 프로토콜에 대해 사용자 정의 어드바이저는 통신 장애를 발견하고 서버를 종료로 표시하며 데이터를 해시 테이블에 저장합니다. 다른 포트의 사용자 정의 어드바이저는 해시 테이블에서 이러한 정보를 알게 되고 자신의 서버를 종료로 표시합니다.

이 예제는 HTTP에 대해 포트 80을 SSL에 대해 포트 443을 연결하도록 작성되었으나, 포트의 조합을 사용자에 맞게 조정할 수 있습니다.

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

```
import com.ibm.internet.lb.server.SRV_ConfigServer;

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

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

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

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

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

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

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

  public int getLoadValue() { return iLoad; }

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

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

}

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

public class ADV_twop extends ADV_Base
   implements ADV_MethodInterface, ADV_AdvisorVersionInterface {
```

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

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

  static HashTable htTwopHTTP = new Hashtable();
  static HashTable htTwopSSL = new Hashtable();

  static final String ADV_TWOP_NAME = "twop";
  static final int ADV_TWOP_DEF_ADV_ON_PORT = 80;
  static final int ADV_TWOP_DEF_INTERVAL = 7;
  static final string ADV_HTTP_REQUEST_STRING =
    "HEAD / HTTP/1.0\r\nAccept: */*\r\nUser-Agent: " +
    "IBM_LB_Custom_Advisor\r\n\r\n";

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

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

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

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

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

  public void ADV_AdvisorInitialize() {
        return;
  }

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

  synchronized ADV_nte getNte(Hashtable ht, String sName, String sHashKey) {
    ADV_nte nte = (ADV_nte)(ht.get(sHashKey));
    if (null != nte) {
      nte = (ADV_nte)nte.clone();
    }
    return nte;
  }
 synchronized void putNte(Hashtable ht, String sName, String sHashKey,
                          ADV_nte nte) {
   ht.put(sHashKey,nte);
```

```
      return;
}

 //--------
 // getLoadHTTP - determine HTTP load based on server response

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

WebSphere Application Server 어드바이저

WebSphere Application Server용 사용자 정의 어드바이저 샘플은 *install_path*/servers/samples/CustomAdvisors/ 디렉토리에 들어 있습니다. 이 문서에서는 전체 코드를 수록하지 않았습니다.

- ADV_was.java는 Load Balancer 시스템에서 컴파일하여 실행하는 어드바이저 소스 코드 파일입니다.
- LBAdvisor.java.servlet은 WebSphere Application Server 시스템에서 컴파일하여 실행하는 servlet 소스 코드입니다.

완전한 어드바이저는 이 예제보다 약간 복잡합니다. 즉, 위에서 본 StringTokenizer 예제보다 더 압축된 특수 구문 분석 루틴을 추가합니다.

코드 샘플의 더 복잡한 부분은 Java servlet에 있습니다. servlet에는 다른 메소드 중에서 servlet 스펙에 필요한 두 개의 메소드 init() 및 service()와 Java.lang.thread 클래스에 필요한 한 개의 메소드 run()이 들어 있습니다.

- init()는 초기화 시 servlet 엔진에 의해 한 번 호출됩니다. 이 메소드는 어드바이저의 호출과 독립적으로 실행되는 이름이 _checker인 스레드를 작성하며 이의 처리 루프를 재개하기 전에 일정 기간 동안 휴면합니다.
- service()는 servlet이 호출될 때마다 servlet 엔진에 의해 호출됩니다. 이 경우 메소드는 어드바이저에 의해 호출됩니다. service() 메소드는 ASCII 문자의 스트림을 출력 스트림에 전송합니다.
- run()에는 코드 실행의 코어가 들어 있습니다. 이 메소드는 init() 메소드에서 호출되는 start() 메소드에 의해 호출됩니다.

servlet 코드의 관련 단편들은 아래와 같습니다.

```
...

  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;
}

...
```

어드바이저에서 리턴된 데이터 사용

Application Server의 기존 부분에 대한 표준 호출을 사용하든, 사용자의 사용자 정의 어드바이저의 서버측 어드바이저가 될 새 코드를 추가하든 리턴되는 로드 값을 검사하고 서버 작동을 변경하려고 할 경우가 있습니다. Java StringTokenizer 클래스 및 이와 연관된 메소드를 사용하면 이러한 검사를 쉽게 할 수 있습니다.

일반적인 HTTP 명령은 GET /index.html HTTP/1.0입니다.

이 명령에 대한 일반적인 응답은 다음과 같습니다.

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

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

관심있는 항목, 특히 HTTP 리턴 코드가 첫 행에 포함됩니다.

HTTP 스펙이 다음과 같이 요약할 수 있는 리턴 코드를 분류합니다.

- 2xx 리턴 코드는 성공
- 3xx 리턴 코드는 경로 재지정
- 4xx 리턴 코드는 클라이언트 오류
- 5xx 리턴 코드는 서버 오류

서버가 리턴할 수 있는 코드를 정확히 알고 있는 경우, 코드를 이 예제와 같이 자세하게 작성할 필요가 없습니다. 그러나 발견하는 리턴 코드를 제한하면 나중에 프로그램의 융통성을 제한할 수 있다는 점에 유의해야 합니다.

다음 예제는 최소의 HTT 클라이언트를 포함하는 독립형 Java 프로그램입니다. 이 예제는 HTTP 응답을 조사하기 위한 간단한 범용 구문 분석기를 호출합니다.

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

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

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

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

    StringTokenizer st = new StringTokenizer(sResponse, " ");
    if (true == st.hasMoreTokens()) {
      sHTTPVersion = st.nextToken();
      if (sHTTPVersion.equals(sHTTP110) || sHTTPVersion.equals(sHTTP11)) {
        System.out.println("HTTP Version: " + sHTTPVersion);
      } else {
        System.out.println("Invalid HTTP Version: " + sHTTPVersion);
      }
      } else {
      System.out.println("Nothing was returned");
        return;
    }
```

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

      switch (iRc) {
      case(200):
        System.out.println("HTTP Response code: OK, " + iRc);
        break;
      case(400): case(401): case(402): case(403): case(404):
        System.out.println("HTTP Response code: Client Error, " + iRc);
        break;
      case(500): case(501): case(502): case(503):
        System.out.println("HTTP Response code: Server Error, " + iRc);
        break;
      default:
        System.out.println("HTTP Response code: Unknown, " + iRc);
        break;
      }
    }

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

주의사항

초판(2006년 5월)

이 정보는 미국에서 제공되는 제품 및 서비스용으로 작성된 것입니다.

IBM은 다른 국가에서는 이 자료에 기술된 제품, 서비스 또는 기능을 제공하지 않을 수도 있습니다. 현재 사용할 수 있는 제품 및 서비스에 대한 정보는 한국 IBM 담당자에게 문의하십시오. 이 책에서 IBM 제품, 프로그램 또는 서비스를 언급했다고 해서 해당 IBM 제품, 프로그램 또는 서비스만을 사용할 수 있다는 것을 의미하지는 않습니다. IBM의 지적 재산권을 침해하지 않는 한, 기능상으로 동등한 제품, 프로그램 또는 서비스를 대신 사용할 수도 있습니다. 그러나 비IBM 제품, 프로그램 또는 서비스의 운영에 대한 평가 및 검증은 사용자의 책임입니다.

IBM은 이 책에서 다루고 있는 특정 내용에 대해 특허를 보유하고 있거나 현재 특허 출원 중일 수 있습니다. 이 책을 제공한다고 해서 특허에 대한 라이센스까지 부여하는 것은 아닙니다. 라이센스에 대한 의문사항은 다음으로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터
전화번호: 080-023-8080

2바이트(DBCS) 정보에 관한 라이센스 문의는 한국 IBM 고객만족센터에 문의하거나 다음 주소로 서면 문의하시기 바랍니다.

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

다음 단락은 현지법과 상충하는 영국이나 기타 국가에서는 적용되지 않습니다.

IBM은 타인의 권리 비침해, 상품성 및 특정 목적에의 적합성에 대한 묵시적 보증을 포함하여(단, 이에 한하지 않음) 묵시적이든 명시적이든 어떠한 종류의 보증없이 이 책을 "현상태대로" 제공합니다. 일부 국가에서는 특정 거래에서 명시적 또는 묵시적 보증의 면책사항을 허용하지 않으므로, 이 사항이 적용되지 않을 수도 있습니다.

이 정보에는 기술적으로 부정확한 내용이나 인쇄상의 오류가 있을 수 있습니다. 이 정보는 주기적으로 변경되며, 이 변경사항은 최신판에 통합됩니다. IBM은 이 책에서 설명한 제품 및/또는 프로그램을 사전 통지 없이 언제든지 개선 및/또는 변경할 수 있습니다.

이 정보에서 언급되는 비IBM의 웹 사이트는 단지 편의상 제공된 것으로, 어떤 방식으로든 이들 웹 사이트를 옹호하고자 하는 것은 아닙니다. 해당 웹 사이트의 자료는 본 IBM 제품 자료의 일부가 아니므로 해당 웹 사이트 사용으로 인한 위험은 사용자 본인이 감수해야 합니다.

IBM은 귀하의 권리를 침해하지 않는 범위 내에서 적절하다고 생각하는 방식으로 귀하가 제공한 정보를 사용하거나 배포할 수 있습니다.

(i) 독립적으로 작성된 프로그램과 기타 프로그램(본 프로그램 포함) 간의 정보 교환 및 (ii) 교환된 정보의 상호 이용을 목적으로 본 프로그램에 관한 정보를 얻고자 하는 라이센스 사용자는 다음 주소로 문의하십시오.

135-270
서울특별시 강남구 도곡동 467-12, 군인공제회관빌딩
한국 아이.비.엠 주식회사
고객만족센터

이러한 정보는 해당 조건(예를 들어, 사용료 지불 등)하에서 사용될 수 있습니다.

이 정보에 기술된 라이센스가 있는 프로그램 및 이 프로그램에 대해 사용 가능한 모든 라이센스가 있는 자료는 IBM이 IBM 기본 계약, IBM 프로그램 라이센스 계약(IPLA) 또는 이와 동등한 계약에 따라 제공한 것입니다.

본 문서에 포함된 모든 성능 데이터는 제한된 환경에서 산출된 것입니다. 따라서 다른 운영 환경에서 얻어진 결과는 상당히 다를 수 있습니다. 일부 성능은 개발 단계의 시스템에서 측정되었을 수 있으므로 이러한 측정치가 일반적으로 사용되고 있는 시스템에서도 동일하게 나타날 것이라고는 보증할 수 없습니다. 또한 일부 성능은 추정을 통해 추측되었을 수도 있으므로 실제 결과는 다를 수 있습니다. 이 책의 사용자는 해당 데이터를 본인의 특정 환경에서 검증해야 합니다.

비IBM 제품에 관한 정보는 해당 제품의 공급업체, 공개 자료 또는 기타 범용 소스로부터 얻은 것입니다. IBM에서는 이러한 비IBM 제품을 반드시 테스트하지 않았으므로, 이들 제품과 관련된 성능의 정확성, 호환성 또는 기타 주장에 대해서는 확인할 수 없습니다. 비IBM 제품의 성능에 대한 의문사항은 해당 제품의 공급업체에 문의하십시오.

IBM이 제시하는 방향 또는 의도에 관한 모든 언급은 특별한 통지 없이 변경될 수 있습니다.

이 정보에는 일상의 비즈니스 운영에서 사용되는 자료 및 보고서에 대한 예제가 들어 있습니다. 이들 예제에는 개념을 가능한 완벽하게 설명하기 위하여 개인, 회사, 상표 및 제품의 이름이 사용될 수 있습니다. 이들 이름은 모두 가공의 것이며 실제 기업의 이름 및 주소와 유사하더라도 이는 전적으로 우연입니다.

이 정보를 소프트카피로 확인하는 경우, 사진과 컬러 삽화가 제대로 나타나지 않을 수도 있습니다.

상표

다음 용어는 미국 또는 기타 국가에서 사용되는 IBM Corporation의 상표입니다.

- AIX
- IBM
- ViaVoice
- WebSphere

Java 및 모든 Java 기반 상표는 미국 또는 기타 국가에서 사용되는 Sun Microsystems, Inc.의 상표입니다.

Microsoft, Windows, Windows NT 및 Windows 로고는 미국 또는 기타 국가에서 사용되는 Microsoft Corporation의 상표입니다.

Intel, Intel Inside(로고), MMX 및 Pentium은 미국 또는 기타 국가에서 사용되는 Intel Corporation의 상표입니다.

UNIX는 미국 또는 기타 국가에서 사용되는 The Open Group의 등록상표입니다.

Linux는 미국 또는 기타 국가에서 사용되는 Linus Torvalds의 상표입니다.

기타 회사, 제품 및 서비스 이름은 해당 회사의 상표 또는 서비스표입니다.

색인

[가]

[다]

[라]

[마]

[바]

[사]

[아]

[자]

[카]

[타]

[파]

A

C

G

H

I

L

P

R

S

W

IBM 한글 지원에 관한 설문

WebSphere Application Server
Edge Components용
프로그래밍 안내서
버전 6.1

GA30-2915-00

FAX : (02) 3787-0123

보내 주시는 의견은 더 나은 고객 지원 체제를 위한 귀중한 자료가 됩니다. 독자 여러분의 좋은 의견을 기다립니다.

성 명		직위/담당업무	
회 사 명		부 서 명	
주 소			
전화번호		팩스번호	
전자우편 주소			
사용중인 시스템	o 중대형 서버 o UNIX 서버 o PC 및 PC 서버		

1. IBM에서 제공하는 한글 책자와 영문 책자 중 어느 것을 더 좋아하십니까?
   그 이유는 무엇입니까?
   ☐ 한글 책자 ☐ 영문 책자
   (이유:)

2. 본 책자와 해당 소프트웨어에서 사용된 한글 용어에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

3. 본 책자와 해당 소프트웨어에서 번역 품질에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

4. 본 책자의 인쇄 상태에 대한 귀하의 평가 점수는?
   ☐ 수 ☐ 우 ☐ 미 ☐ 양 ☐ 가

5. 한글 소프트웨어 및 책자가 지원되는 분야에 대해 귀하는 어떻게 생각하십니까?
   ☐ 한글 책자를 늘려야 함 ☐ 현재 수준으로 만족
   ☐ 그다지 필요성을 느끼지 않음

6. IBM은 인쇄물 형식(hardcopy)과 화면 형식(softcopy)의 두 종류로 책자를 제공합니다.
   어느 형식을 더 좋아하십니까?
   ☐ 인쇄물 형식(hardcopy) ☐ 화면 형식(softcopy) ☐ 둘 다

IBM 한글 지원 서비스에 대해 기타 제안사항이 있으시면 적어주십시오.

☺ 설문에 답해 주셔서 감사합니다.
귀하의 의견은 저희에게 매우 소중한 것이며, 고객 여러분들께 보다 좋은 제품을 제공해 드리기 위해 최선을 다하겠습니다.

GA30-2915-00

Spine information:

WebSphere Application Server

Edge Components 프로그래밍 안내서

버전 6.1

GA30-2915-00